TUHH

Hamburg University of Technology

Comparing Naïve Bayesian Networks to Support Vector Machines to predict Stock Prices Based on Press Release Sentiment

– Master Thesis –

Author:
Max Lübbering, 21599173

Supervisors:
Dr. Julian Kunkel
Dr. Patricio Farrell

Examiner:
Dr. Patricio Farrell

December 4, 2018

Contents

List of Figures

List of Tables

Listings

In this master thesis, we trained and evaluated two models, namely a naïve Bayesian network and a support vector machine, to predict stock price trends based on count vectorized press releases published in the after hours. Additionally, we introduced a trivial classifier to put the results of the other two into perspective.

The stock price trend prediction was solved in two steps: In the first step, we built for each of the three algorithms a classifier in order to predict the impact of a press release. For training and evaluation, every press release was assigned the label *impact*, if the stock price had changed at least 8% from the exchange closing time to exchange opening time plus an offset of 5 hours and *no impact*, otherwise. In the second step, all press releases with no impact were discarded. The remaining ones were reassigned to the classes *sentiment* and *no sentiment* based on the direction of their impact. Afterwards, we built a model for each of the three algorithms to predict the sentiment of press releases.

After applying grid search on an extensive grid, the impact models of the naïve Bayesian network, the support vector machine and trivial classifier had an accuracy of 76%, 78% and 77%, respectively. The balanced dataset contained 919 training samples and 231 samples in the holdout set. These high accuracies show that impact prediction is per se possible, even though count vectorization destroys all the semantics.

The sentiment models of the naïve Bayesian network, the support vector machine and trivial classifier had an accuracy of 47%, 53% and 53%, respectively. The balanced training set contained 426 samples and the holdout set 108. The weak results reveal that sentiment prediction is far more complex than impact prediction and cannot be captured by the word frequency in a document.

As part of our press release exploration, we demonstrated that over night press releases cause inefficiencies in the market for the entire next trading day. As a consequence, we provided an example that clearly contradicts with the theoretical efficient market hypothesis. If our models become more reliable, these inefficiencies can be exploited.

1 Introduction

This section sheds light on the different aspects of algorithmic trading in general and the benefits of leveraging press releases for stock market prediction. After having motivated the topic in Section 1.1 which partly already appeared in [18], we state the goals and requirements for a press release trading system in Section 1.2 and introduce the three concrete approaches for such a system in Section 1.3. Finally, an outline on the entire thesis is provided.

1.1 Motivation

Nowadays, the Internet offers a vast amount of business news that a single individual is unable to process. There are thousands of news articles about exchange traded companies being published every single day.

Additionally, publicly traded companies are legally required to publish their quarter results in form of press releases and important insider information by ad hoc announcements in order to prevent insider trading.

What if we were able to process this information reliably within seconds and were able to react to market change before any other party? This is essentially what stock traders do, with the manual manner of their work taking them a couple of minutes to adapt to market change [17]. They usually install a notification alarm of some news provider, giving them access to the latest news regarding the companies in their portfolio. Broadly speaking, they continuously read these news articles and choose one of the three trading options: *hold*, *buy* or *sell*.

We have shown in previous work, that the task of labeling news articles by the companies they mention is a rather complex task, however not impossible in practice [18]. As part of our research, we pursued two approaches. In the first one, we trained a k-nearest neighbor classifier based on either count vectorized or tf-idf vectorized company descriptions, which were labeled by the company name. For prediction of news articles we then picked the k company descriptions, that were most similar to the news article and performed majority vote (ordinary k-nearest neighbor on word distributions). For k-nearest neighbor with count vectorization we reached an accuracy of 47% and for k-nearest neighbor with tf-idf vectorization an accuracy of 51%, where the task was to find the correct company mentioned in a news article out of 480 companies.

In the second approach we used either the count vectorized or the tf-idf vectorized company descriptions to build a topic model using Latent Dirichlet Allocation (LDA). We had at least seven company descriptions for each of the 480 companies and parameterized the algorithm to cluster 480 topics. Our intention was thereby to give the algorithm a bias towards representing each company by one topic. Using the topic distributions of each company description, we trained a k-nearest neighbor classifier and predicted companies in the same fashion as in the first approach. The difference here is, that LDA reduces the feature space of the k-nearest neighbor algorithm to 480, which is less than a tenth of feature space in the first approach, making prediction less computationally expensive. However, this approach performed worse, having an accuracy of 37% for count vectorized company descriptions and 13% for tf-idf vectorized company descriptions.

Even though the results are not applicable in the field, yet, we think that our findings are promising given the fact that we have chosen the parameters only by our best intuition and we have not performed a proper hyperparameter optimization, yet. Also, we should keep in mind, that this is not a binary classification problem but in fact a classification problem with

480 possible classes, which means that our performance is at least 60 times better than random guessing.

Having demonstrated that news article to company matching is possible in theory and practice after further optimization, our objective in this master thesis is to further automate a day traders job. Specifically, we want to investigate, whether it is possible to utilize press releases to derive trading strategies based on their sentiment. For instance, when a company's quarter results are being released and they are better than the investors had predicted them to be, then we would expect the company's stock price to spike, because at this very moment the shares are undervalued and many traders will want to buy shares to make profits. Our model similar to the the day trader is supposed to recognize these kind of positive or negative sentiments and predict the trend of future stock prices. Utilizing press releases labeled with the corresponding company, we can combine them with the stock price development of each company. This dataset is passed to the supervised machine learning algorithm to train models, that predict the trend of a stock based on a press release.

It would have been a logical conclusion to not only use press releases but also to use arbitrary business news articles. However, press releases have a huge advantage over usual news articles e.g., published by the New York Times, as press releases contain most of the time the company's ticker symbol. Searching for ticker symbols using a regular expression is an easy way to label the press releases. We will stick to press releases until we have found a good refinement of the parameterization of the news article to company matching model of our previous work and will then expand the trading system to all kind of news articles.

1.2 Goals

In particular, we want to solve the following tasks:

- Data Acquisition:
  We need to develop a crawler that retrieves the press releases and another one that retrieves the corresponding stock prices. Both press releases and stock prices have to be tagged with the company symbol and a timestamp, such that we can set both datasets into relation.

- Preprocessing:
  The press releases have to be represented in a way, such that they can be forwarded to the respective machine learning algorithms.

  In order to predict stock price trends, we have to find a scale-free measure, that allows to compare stocks of different price categories. For instance, a stock that is traded at 1,000 $ per share is probably fluctuating more than a stock that is worth 1 $ per share. Also, we have to find an appropriate time frame, when comparing price changes. For example, if a press release was published at 1 AM, do we take last stock price before the exchange closed as reference and compare it to the exchange open price or do we compare it to the stock price e.g., 5 h after the exchange opened? It is not clear what time frames work best and it is part of our goals to determine those.

  Another important factor is the market itself: When the market is bullish, all stock prices tend to increase. If the market is bearish, the stock prices tend to move in the opposite direction. In a bullish market our model might not learn the true impact of press releases' sentiments as they are shifted by a rising trend. This is why, part of our analysis is to investigate if these market effects are also relevant for shorter terms (i.e., five hours). One possibility to compensate for that, is to subtract the relative price development of indices like the Dow Jones which reflects the US market, from the stock prices.

- Modeling:
  The supervised machine learning algorithms, that we choose for stock price trend prediction based on press releases, are supposed to provide a model that meets the following criteria:
  - For a given press release we want the prediction to be either one of *rising*, *falling* or *steady*.
  - We want the model to be applicable to all stocks and not limited to a subset of stocks (e.g., only the banking industry)
  - The model should work with a variety of press releases, while providing stable predictions.
- Evaluation:
  For each trained model, we want to apply the same evaluation methods such that we can compare each model's performance. We also want to pinpoint the weaknesses of each algorithm qualitatively. For instance, there might be a company, which always tries to massage their bad quarter results and there could be a model that tends to get fooled in this situation. We want to locate these kind of weaknesses for each model.

1.3 Approaches

The task of predicting the trend of a stock price i.e., *rising*, *falling* or *steady* is highly complex as the model has to learn weather a press release has an impact and if so whether it is positive or negative. This is why, this task is being solved by two separate models, namely an impact model, which predicts whether the stock price will significantly diverge due to the press release and a sentiment model which predicts the direction of the stock price development. If the impact model predicted a press release to have an impact, then this press release is being forwarded to the sentiment model, which predicts the direction of the divergence from the stock price before the press release was published.

The model training and prediction of both tasks is represented by the schematic sketch, shown in Figure 1. For both tasks we have a stock dataset, that contains the stock prices for each company at one minute frequency. These stock prices are transformed during the preprocessing into a scale-free quantity by calculating for each timestamp the relative change of each stock after a fixed offset. This quantity is later introduced as the simple net opportunity (SNO).

Each raw press release contains plain HTML. Obviously, this representation is not applicable to be passed to machine learning algorithms. Therefore, we perform several preprocessing steps including text extraction, tokenization, stop word and punctuation removal, part-of-speech tagging (POS tagging) and lemmatization, which are explained in Section 4.2.2. After having cleaned the texts, we need a representation of the texts that the machine learning algorithms can deal with. We will use count vectorization which is introduced in Section 4.2.3.

Then, we assign each preprocessed press release the corresponding relative stock price change (SNO) by carrying out the following steps:

```
1 For each company c of all companies:
2   For each press release p of c's press releases:
3     Assign p relative stock price change (SNO) at time p.retrieval_timestamp
```

This essentially boils down to an inner join of the press releases and relative stock price changes on the companies and timestamps.

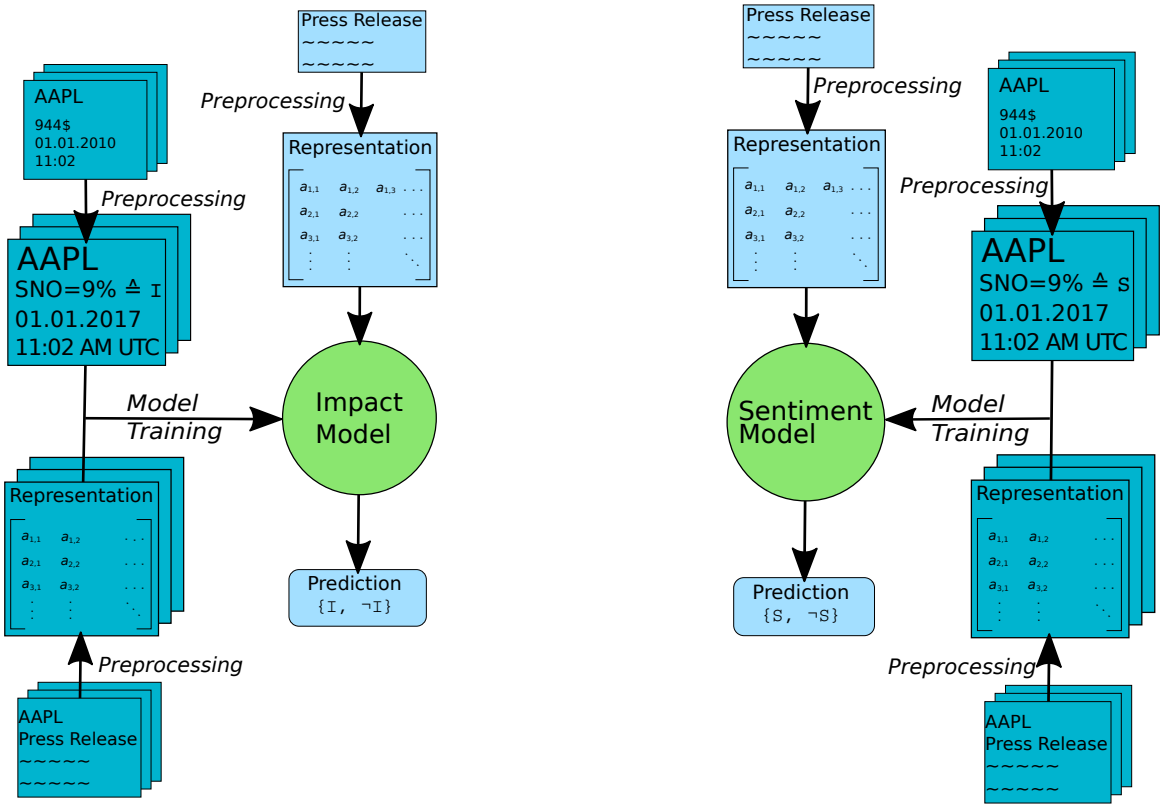
The result is a dataset which contains the preprocessed texts and the stocks, as shown in Table 1.

Prep. Texts	SNO
Text 1	0.05
Text 2	-0.01
...	...
Text n	0.11

Table 1: Exemplary samples of press releases with SNO as target variable

Depending on which task we want to solve i.e., impact or sentiment, we have to discretize the continuous SNO appropriately. For the impact model we want to distinguish between the two classes *impact* i.e., falling or rising trends and *no impact* i.e., steady trends. The two classes will be denoted by $\{I, \neg I\}$, respectively. For the sentiment model we differentiate between press releases having positive or negative sentiment, thus leading to a falling or rising trend. The two classes will be denoted by $\{S, \neg S\}$, respectively.

After having discretized the relative stock price changes using pre-defined thresholds, we train the model which is represented by the green circles in Figure 1a and Figure 1b. For prediction, we preprocess the press releases the same way as in the training set and pass them to the model. Then, the impact model predicts one of the classes $\{I, \neg I\}$ and the sentiment model one of the classes $\{S, \neg S\}$.



(a) Impact model

(b) Sentiment model

Figure 1: Schematic sketches of training and prediction with the impact model and the sentiment model

From a technical point of view the impact model and the sentiment model differentiate only by the discretization of the target variable, which is part of the preprocessing. This is why, we can apply our concrete approaches to both tasks without having to adapt anything. Next, we will briefly introduce each approach applied to the task of impact modeling.

Approach 1: Trivial Approach

This approach's objective is to put the performance of the following algorithms into perspective by providing comparative values. The model is not sophisticated at all and is not necessarily intended to be competitive.

During model training we split the training set into the two classes $\{\neg I, I\}$ and keep the k most frequent words of each class. For prediction of a press release's impact, the class is being selected whose k most frequent words appear most often in the press release.

Undoubtedly, this approach has various shortcomings e.g., if a most frequent word appears in both classes this word does not contribute to classification. However, we will see that albeit its simplicity and methodological errors, this approach yields even competitive results.

Approach 2: Naïve Bayesian Networks

This special case of Bayesian networks has two layers, where the features are the leaves and the class that is to be predicted is the root. Naïve Bayesian networks can be used to classify texts by treating the words in the documents as features, as shown in Figure 2.

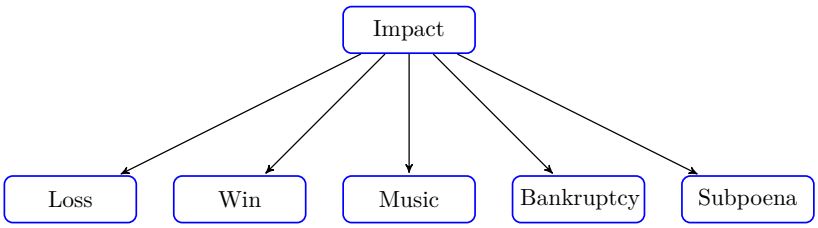


Figure 2: Example of naïve Bayesian network for text classification

A typical query against our model would be the probability of impact given that the press release contains the words *loss*, *bankruptcy* and does not contain the words *win*,*subpoena* and *music*, which would be expressed in mathematical terms by $p(Impact = True \,|\, Loss = True, Bankruptcy = True, Win = False, Subpoena = False, Music = False)$.

Approach 3: Support Vector Machines

In contrast to the naïve Bayesian network approach, in this approach we do not build a probability model but try to separate the press releases in a high dimensional feature space. After having count vectorized the press releases, we can represent each press release as a vector and project them into the feature space. The objective of the support vector machine (SVM) is to find a hyperplane that separates the press releases of both classes best. We will see that this optimization problem boils down to solving a convex optimization problem.

For prediction of a sample, we only check on which side of the hyperplane the sample is located.

1.4 Outline

This master thesis is structured in eight sections. After this introductory chapter, we will provide the reader with comprehensive background information, including an introduction to stock trading as well as the mathematical foundations of the two machine learning algorithms and feature selection strategies in Section 2, to enable him to understand the subsequent sections of this thesis.

Section 3 deals with the related work and research others have conducted on text classification. This section also gives an overview of the analysis that other researchers have performed on the influence of the media on stock prices.

In Section 4, we explain how the datasets have been acquired and preprocessed such that the machine learning algorithms can learn based on them. We also explain how the machine learning algorithms, whose theoretical aspects have been only covered in Section 2, can be applied to text classification and how these models are evaluated.

The implementation of the methods stated in Section 4 is covered in Section 5. In particular, we explain how the two datasets (press release set and stock price set) are mined, how we preprocess them and finally how the models are implemented.

In Section 6, we explore the two datasets for possible holes, redundancies and other criteria and evaluate each impact and sentiment model.

In Section 7 all the work is being summed up and reflected in terms of our initial goals stated in Section 1. Based on this, we give a prospect on what is left for future work in Section 8.

2 Background

We familiarize the reader in Section 2.1 with important terms and concepts from the trading domain. Afterwards we explain the mathematics and algorithms used in our research. First, we give a short introduction to Bayesian statistics in Section 2.2. This provides the basis to understand the Bayesian networks and naïve Bayesian networks which are explained in Section 2.3 and Section 2.4, respectively. The theory behind support vector machines is explained in Section 2.5. In order to limit the number of features passed to the machine learning algorithms, feature selection strategies are introduced in Section 2.6.

2.1 Introduction to Stock Trading

This section explains financial terms and concepts imperative to understand the subsequent chapters.

2.1.1 Efficient Market Hypothesis

The efficient market hypothesis initially created by Fama [7], states that it is impossible to beat the market, because the price of an asset reflects all information that is available at any time. That is why according to this hypothesis, it is impossible to predict price movements using fundamental or technical analysis.

As with any hypothesis, there are also critics. In [19] Malkiel collected and reviewed patterns that are considered to enable traders to beat the market. On the one hand, he concluded that markets are generally efficient apart from rare bubbles in the past. On the other hand looking at smaller time periods, in his opinion there might be some inefficiencies in the stock markets.

One of our main goals within the scope of this thesis is to analyze, whether there are any inefficiencies due to press releases. If this was the case, we could build models that take advantage of these inefficiencies and later on we could even trade stocks on the basis of press releases. As another consequence, we could confidently reject the efficient market hypothesis, which would lead to new research possibilities.

2.1.2 Entities

In this section, we briefly explain the entities involved in trading, namely exchange, broker, trader and stock.

Stock

Stocks, also known as shares, represent portions in the ownership of a company. This entitles the owner to have a vote in company decisions and to have a share of profits in form of dividends. From a legal perspective the owner does only own a portion of the shares and not a portion of the company's assets (e.g., you cannot leave the company's office with a conference table even if it matches the value of your shares). However, you can cash on the shares by selling them to a buyer.

Exchange

An exchange is the marketplace, where stocks and other assets are being traded publicly. The exchange's job is to provide a transparent environment, such that there is a fair order execution. In our work we focus on companies whose stocks are traded on NASDAQ and NYSE, which are the two biggest exchanges in the USA.

It is not possible to trade at the exchanges 24/7. Every exchange has its own trading hours, which are for NASDAQ and NYSE from 1:30 PM UTC to 8:00 PM UTC, as shown by Figure 3.

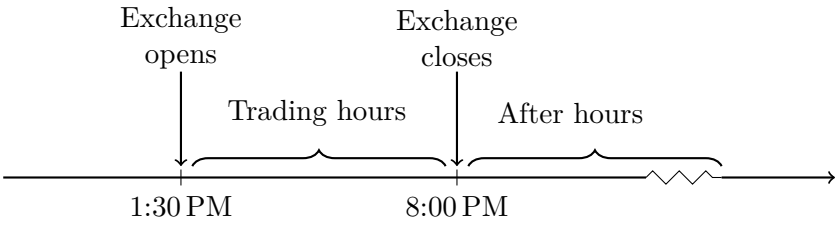


Figure 3: Exchanges' trading hours

Electronic Communication Networks (ECNs) break up this limitation, allowing traders to engage in the market even when the exchanges are closed. ECNs directly match buyers and sellers anonymously using their own order books. This made them especially popular among institutions as they can hide their trading strategies. Today, also private traders use these networks to benefit from the possibilities of extended trading hours. In [12], Hendershott states that already in 2002 the volume traded on ECNs made up approx. 40% of the total volume in NASDAQ securities[1].

Trader

A trader is a person, who buys and sells stocks in order to earn money from short term stock price movements. This is the main distinction from investors, who hold stocks for a long period of time.

Broker

A broker acts as proxy between a trader and an exchange. A trader orders a broker to sell or buy a stock, which then executes the order at the exchange. In return the broker charges the trader a commission fee.

2.1.3 Reading Stock Charts

When we want to analyze stock prices of a certain company, we are interested in their development over time, as exemplary showed in Table 2 for Apple Inc.

Every minute we are provided with the latest stock prices. Since trading happens continuously, stock values not only change with every minute tick. In order to also track trends within these periods, we store the stock price at the beginning (open) and at the end of each period (close), as well as, the maximum (high) and minimum (low) stock prices within each period. That is why for every 1 min period we receive four values.

[1] `https://www.investopedia.com/terms/s/security.asp`

	datetime	exchange	symbol	open	close	high	low
0	2018-04-11 16:00:00	nasdaq	aapl	173.20	173.23	173.26	173.13
1	2018-04-11 16:01:00	nasdaq	aapl	173.22	173.30	173.30	173.22
2	2018-04-11 16:02:00	nasdaq	aapl	173.30	173.42	173.43	173.27
3	2018-04-11 16:03:00	nasdaq	aapl	173.42	173.47	173.52	173.42
4	2018-04-11 16:04:00	nasdaq	aapl	173.46	173.74	173.83	173.44

Table 2: Apple stock prices (open, close, high, low) in Dollars from April 11, 2018 4:00 PM to April 11, 2018 4:04 PM

As manually analyzing these stock price tables is too difficult, we need a better representation, such that we can identify trends at first glance. One approach is to plot the open, close, low and high stock price for each minute as a Japanese candle stick, as shown in Figure 4.

The thicker vertical lines represent the open and closing stock values within the respective minute. Depending on the trend within each period, the sticks are colored red (falling) or green (rising). So the candle stick is colored green when the open value is lower than the close price.

The upper thin end of the stick shows the highest stock value within the period. Likewise, the lower thin end of the stick shows the lowest stock value within the period.

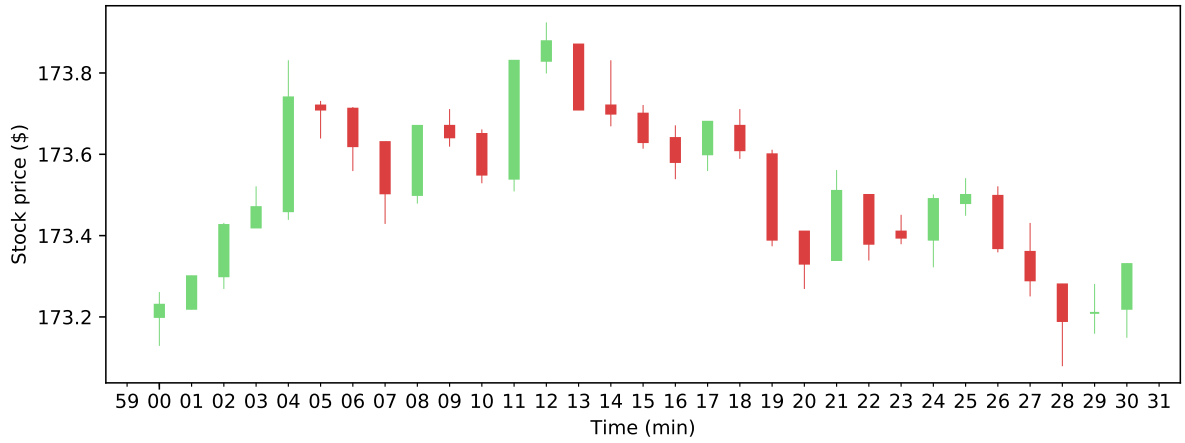


Figure 4: Example of Japanese candle stick chart: Apple stock from April 11, 2018 4:00 PM to April 11, 2018 4:30 PM

One way to interpret the Japanese candle stick chart in Figure 4 would be to conclude, that from 4:00 PM to 4:12 PM there was and upwards trend, which was followed by a downward trend from 4:13 PM to 4:30 PM. Also notable, after each strong upward stick (see 4:04 PM or 4:11 PM) there is a slight decrease. This process is called correction. After a steep upwards trend the traders come to the conclusion, that the market price of a stock is overvalued, so they either sell or stop buying. The lack of interest in this stock automatically leads to a correction to the true market value (see efficient market hypothesis in Section 2.1.1). Of course, the same holds for strong trends in the opposite direction.

The correction behavior can be seen in short-term, as well as, for long-term periods. Every time the market overreacts, the traders will spot their opportunity.

2.2 Introduction to Bayesian Statistics

As Bayesian networks build upon Bayesian statistics, we will give a short introduction on the basics of Bayesian statistics and how to use it. This section was partly adopted from [18] and is based on the great book on Bayesian statistics called "Bayesian Data Analysis" [9], whose approach and notation we will follow.

Other than the text book, we will specifically mark a distribution's arguments when it is unclear whether they are constants or variables. For example, given the distribution $p(y|\Theta)$ it is unclear to the reader, whether this is a function of y or of Θ. To prevent any confusion, we will mark the distribution's variables by underlines e.g., for $p(y|\underline{\Theta})$, y would be constant and Θ would be the variable.

When performing Bayesian inference, we always have an unknown population. For instance this population could be the true number of males and females in a country. Next, we select a probability distribution that we assume fits the population and that is parameterized by Θ. The parameter Θ defines the shape of the distribution and for simplicity of our example, we say that Θ also represents the sex ratio (females divided by population size) in the population.

One objective of Bayesian inference is to estimate Θ for given samples y (observed data). Let $H = \{h_1, h_2, \ldots, h_n\}$ be the hypothesis space i.e., the set of values Θ can attain, then for each h_i, Θ has a certain posterior probability $p(\Theta = h_i|y)$. Assuming we have 10 males and 12 females as evidence, we want to determine the probability of $p(\Theta = 0.1 \,|\, males = 12, females = 10)$, which is low because the observed data contradicts with such Θ, but for $\Theta = 0.55$ the posterior probability distribution would peak.

The other objective assumes, that we do know the distribution of $p(\underline{\Theta}|y)$ (i.e., we know the probability of all sex ratios in a country given 10 males and 12 females) and we want to predict the probability of unseen data given some observations $p(\tilde{y}|y)$. In the example, we would want to know how likely it is, that a person picked at random is male:
$p(y = \text{male} \,|\, \text{males} = 10, \text{females} = 12)$.

In summary, Bayesian inference means to draw conclusions on the parameter(s) Θ of the population given some observed data y, or to predict unknown data $\tilde{y}$ given some observed data y. We define $p(\underline{\Theta}|y)$ as the *posterior density* and $p(\underline{\tilde{y}}|y)$ as the *posterior predictive distribution*.

Bayes' theorem

In order to make probability predictions of parameter Θ for given data y, we calculate the joint probability $p(\Theta, y)$ first.

$$p(\Theta, y) = p(\Theta)p(\underline{y}|\Theta)$$

The joint probability is the product of the *prior probability* $p(\Theta)$ and the *sampling distribution* $p(\underline{y}|\Theta)$.

Now, we condition on y, which gives us the *posterior probability* $p(\underline{\Theta}|y)$. This step is known as Bayes' Theorem:

$$\boxed{p(\underline{\Theta}|y) = \frac{p(\Theta, y)}{p(y)} = \frac{p(\Theta)p(y|\underline{\Theta})}{p(y)}}. \tag{1}$$

It is important to note that the *posterior density* is a function of Θ and that therefore also $p(y|\underline{\Theta})$ is a function of Θ. The function $\Theta \mapsto p(y|\underline{\Theta})$ is called *likelihood*.

Since $p(y)$ is by definition constant (y is given), we can leave it out of Equation (1) and define

the resulting equation as the *unnormalized posterior density* [2] by

$$p(\underline{\Theta}|y) \propto p(\Theta)p(y|\underline{\Theta}). \tag{2}$$

Prediction

When there is no observed data y available so far, all we can do is calculate the *prior predictive distribution* $p(y)$. This marginal distribution[3] is being calculated from the *joint probability* by summing out / integrating over the hypothesis space H of Θ:

$$p(y) = \int_H p(y, \Theta = h)dh = \int_H p(\Theta = h)p(y|\Theta = h)dh. \tag{3}$$

From the point on when data is available, we can calculate the probability of some unknown data $\tilde{y}$ given observed data y.

We will now derive the formula of **Full Bayesian learning (FBL)** for calculating $p(\tilde{y}|y)$:

$$\begin{aligned} p(\tilde{y}|y) &= \int_H p(\tilde{y}, \Theta = h|y)dh \\ &= \int_H p(\tilde{y}|\Theta = h, y)p(\Theta = h|y)dh \\ &= \int_H p(\tilde{y}|\Theta = h)p(\Theta = h|y)dh. \end{aligned} \tag{4}$$

The last step follows from the conditional independence of y and $\tilde{y}$ given Θ. By integrating over the hypothesis space H, we get a weighted average over the predictions of each hypothesis, where the weights are denoted by the posterior density $p(\Theta = h|y)$ and the prediction $p(y|\Theta = h)$.

As integrating over the entire hypothesis space H of Θ is in most cases intractable, a common approximation is to use **Maximum a Posteriori (MAP) learning**. Instead of integrating over the hypothesis space H, we pick the one that maximizes the posterior density $p(\Theta = h|y)$ i.e., the hypothesis that explains the observations y best.

$$h_{MAP} = \arg\max_{h \in H} p(\Theta = h|y) \tag{5}$$

$$= \arg\max_{h \in H} \frac{p(\Theta = h, y)}{p(y)} \tag{6}$$

$$= \arg\max_{h \in H} \frac{p(y|\Theta = h)p(\Theta = h)}{p(y)} \tag{7}$$

$$= \arg\max_{h \in H} p(y|\Theta = h)p(\Theta = h) \tag{8}$$

Using hypothesis h_{MAP} as the only hypothesis for Θ our prediction simplifies to

$$p(\tilde{y}|y) \approx p(\tilde{y}|\Theta = h_{MAP}). \tag{9}$$

With more observations y the MAP learning approaches FBL, because the other hypotheses in H become less likely. It is worth noting that the prior probability $p(\Theta)$ plays an important role in MAP and FBL as it gives a bias towards which hypotheses are being favored.

[2] Here, $\propto$ indicates proportionality.

[3] A probability distribution that only depends on a subset of the random variables of the original distribution. Marginal distributions are retrieved by summing out the unwanted random variables.

If each hypothesis is equally likely or we do not have any knowledge about the prior probability $p(\Theta)$, one way is to assume it to be uniformly distributed. This further simplification is called **Maximum Likelihood (ML) learning**. We pick the hypothesis that maximizes the likelihood of the data y

$$h_{ML} = \arg\max_{h \in H} p(y|\Theta = h). \tag{10}$$

The prediction

$$p(\tilde{y}|y) \approx p(\tilde{y}|\Theta = h_{ML}) \tag{11}$$

works the same as in MAP learning. Note that to overcome the influence of the prior probability $p(\Theta)$, we need a large dataset.

Candy example

To show how FBL works in practice, we give a simple example, which we have taken from [25]. Let us suppose there is a candy manufacturer who provides lime and cherry candy which are wrapped identically. Then the consumer is only able to tell the flavor after tasting the candy. The manufacturer likes surprises and decides to send the purchaser one of the bag types $H = \{h_1, h_2, \ldots, h_5\}$:

- Type h1: 100% cherry
- Type h2: 75% cherry, 25% lime
- Type h3: 50% cherry, 50% lime
- Type h4: 25% cherry, 75% lime
- Type h5: 100% lime

The manufacturer picks the bags according to the distribution

$$p(H) =< 0.1, 0.2, 0.4, 0.2, 0.1 > . \tag{12}$$

Now, we buy a bag, whose lime cherry ratio we do not know. We unwrap one candy after another and determine its flavor. We denote all candies classified so far by $D = \{d_1, d_2, \ldots, d_n\}$, where d_n is the last candy that we picked. After each pick, we calculate the posterior probability

$$p(h_i|d_1, d_2, \ldots d_n) = \alpha\, p(d_1, d_2, \ldots d_n|h_i)p(h_i) \tag{13}$$

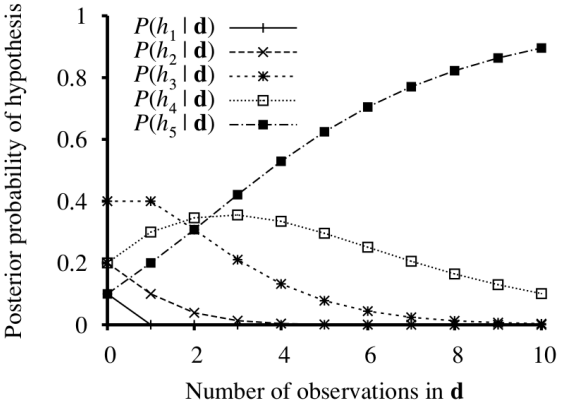
of each bag type h_i given the n unwrapped candies seen so far and calculate the posterior predictive probability of the next picked candy being of flavor lime

$$p(x = lime|d_1, d_2, \ldots d_n). \tag{14}$$

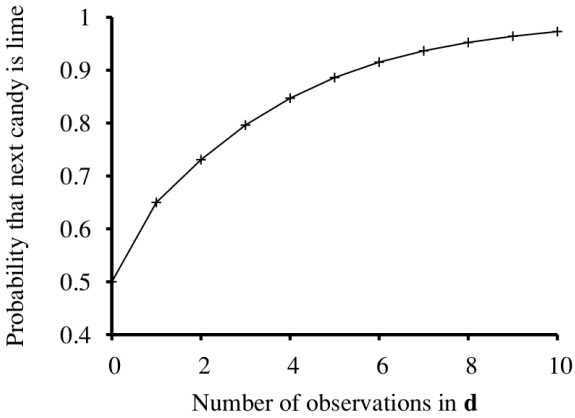
Now, we suppose that the manufacturer has sent us a bag of candies of type h_5 and we try the first ten, which are obviously all lime. After each trial, we calculate the posterior of each bag type according to Equation (13), as shown in Figure 5a and the predictive posterior probability of next candy being lime according to Equation (14), as shown in Figure 5b.

After n observations, we would pick a bag as a best guess according to:

$$h_{FBL} = \arg\max_{h} p(h|d_1, d_2, \ldots d_n) = \alpha\, p(d_1, d_2, \ldots d_n|h)p(h). \tag{15}$$



(a) Posterior probability after each observation for each



(b) Posterior predictive probability of next candy being lime

Figure 5: Posterior probability of each bag given candies **d** and posterior predictive probability of next candy given candies **d** [25]

2.3 Bayesian Networks

In this section, we first motivate Bayesian Networks by working out the disadvantages of the full joint distribution (FJD) and then explain how Bayesian networks improve upon them. Afterwards we derive the parameter learning and inference of Bayesian networks.

2.3.1 Motivation by the Full Joint Distribution

Let $Y = \{Y_1, Y_2, \ldots, Y_n\}$ and $Z = \{Z_1, Z_2, \ldots, Z_m\}$ be two distinct sets of random variables that fully describe a probabilistic model. The joint distribution $p(Y, Z)$ of all random variables is called the full joint distribution. Using this distribution, we can answer any query to the probabilistic model (inference) by performing **marginalization**

$$p(Y) = \sum_{z \in Z} p(Y, z) \tag{16}$$

or by **conditioning**

$$p(Y) = \sum_{z \in Z} p(Y|z)p(z), \tag{17}$$

where $\sum_{z \in Z}$ denotes in both cases all possible combinations of the values of the random variables in Z.

One of the shortcomings of the FJD is its disability to scale. Given n random variables, which can take k values each, the full joint distribution is described by k^n probabilities. A probabilistic model with 20 binary random variables would already be described by 1,048,576 probabilities.

In order to overcome this shortcoming, Bayesian networks were introduced. They make use of conditional independence between random variables to build an acyclic, directed graph, whose parameters only grow linearly in the number of random variables.

2.3.2 Introduction to Bayesian Networks

Bayesian networks make use of conditional independence of random variables, in order to reduce the number of parameters to be trained. Before going too much into detail, we briefly define independence and conditional independence. Two random variables A and B are **independent** if the following equation holds:

$$p(A, B) = p(A)p(B), \tag{18}$$

where $p(A, B)$ is the joint probability of A and B. A typical example for independence is rolling a dice twice. The first outcome is independent of the second and vice versa.

Conditional independence between two random variables A and B given random variable C is present, if one of the two equations holds:

$$p(A, B|C) = p(A|C)p(B|C) \tag{19}$$

$$p(A|B, C) = p(A|C). \tag{20}$$

Let us visualize conditional independence via an example: Imagine a patient has an appointment with the dentist and gets his tooth checked for cavity. The dentist determines the tooth's condition on whether the probe catches in or not. The probability that the probe catches in given the fact that the patient has a cavity is not influenced whether he has toothache or not

$$p(catch|cavity, toothache) = p(catch|cavity). \tag{21}$$

We now pick up the dentist example, again and extend it by the random variable *weather* at that day, which is obviously independent of all the other random variables (*catch*, *toothache* and *cavity*). A Bayesian network can be represented as an acyclic, directed graph, as we did in Figure 6 for the dentist example. The link between two nodes (these are the random variables) means that the parent directly influences its child random variable.

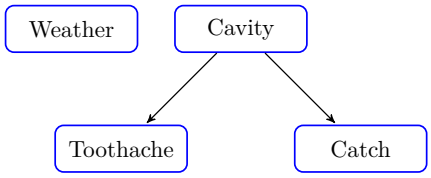


Figure 6: Bayesian network of dentist example

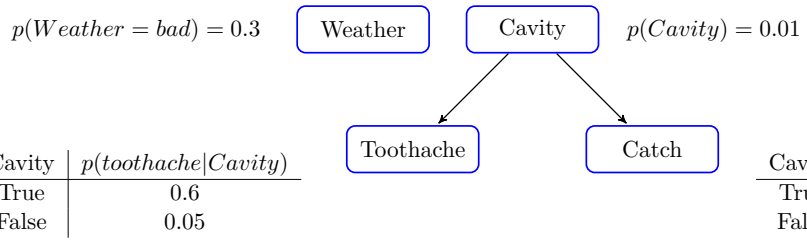
The random variable weather is independent of all the other random variables and therefore there are no links to any of the other random variables. Cavity on the other handy directly influences / causes toothache and the probe to catch in, leading to the corresponding links.

The network not only encodes independence but also conditional independence:
A node is independent of its non-descendants given its parents.

In the example *toothache* is independent of *catch* given *cavity*, which also feels intuitive. When we already know that there is a cavity, the probability of the probe catching in is not influenced by whether the patient has toothache or not (see Equation (21)).

The independence and conditional independence assumptions greatly reduce the number of parameters. When we assert nothing, the FJD has $2^4 = 32$ parameters, of which we had to determine only 31, because the sum over all probabilities in the FJD is equal to one. The parameters in a Bayesian network are represented in conditional probabilities tables (CPTs),

as shown in Figure 7. The Bayesian Network has only six free parameters (probabilities in the CPTs) that have to be determined.



Cavity	$p(toothache\|Cavity)$
True	0.6
False	0.05

Cavity	$p(catch\|Cavity)$
True	0.9
False	0.01

Figure 7: Bayesian network of dentist example with invented probabilities in the CPTs (network parameters)

This locally structured system i.e., each subcomponent (here: random variable) interacts with a bounded number of other subcomponents, leads to a complexity that is rather linear than exponential in the number of subcomponents [25].

Supposing we have n Boolean variables and each variable is influenced by at most k parents, then each CPT has at most 2^k parameters and the entire system at most $n2^k$ parameters. In contrast, the FJD has 2^n parameters.

Finally, we have to show that a Bayesian network (topology and CPTs) in fact represents the FJD. We define the FJD to be

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Theta(x_i|parents(x_i)), \tag{22}$$

where $\Theta(x_i|parents(x_i))$ denotes the parameters of the Bayesian network in the CPTs. We have to show that $\Theta(x_i|parents(x_i))$ corresponds to the conditional probability $p(x_i|parents(x_i))$.

Since the general proof is more sophisticated, we refer the reader to [25]. The simpler proof for the exemplary Bayesian network, which is shown in Figure 8, is discussed instead.

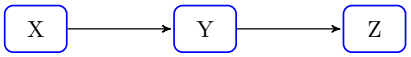


Figure 8: Simple Bayesian Network with random variables X, Y and Z

The proof begins with representing the conditional probability $p(z|y)$ in terms of the FJD by applying Bayes theorem and marginalization:

$$p(z|y) = \frac{p(z,y)}{p(y)} \tag{23}$$

$$= \frac{\sum_{x \in X} p(z,y,x)}{\sum_{z' \in Z} \sum_{x \in X} p(y,z',x)}. \tag{24}$$

Next, we rewrite the expression using Equation (17) in terms of the network parameters

$$p(z|y) = \frac{\sum_{x \in X} \Theta(x)\Theta(y|x)\Theta(z|y)}{\sum_{x \in X} \sum_{z' in Z} \Theta(x)\Theta(y|x)\Theta(z'|y)}, \tag{25}$$

which can be rewritten as

$$p(z|y) = \frac{\Theta(z|y) \sum_{x \in X} \Theta(x)\Theta(y|x)}{\sum_{x \in X} \Theta(x)\Theta(y|x) \sum_{z' in Z} \Theta(z'|y)} \tag{26}$$

by moving the sum over z' inwards. This expression can be further simplified by taking into account that $\sum_{z' in Z} \Theta(z'|y) = 1$ and we can cancel $\sum_{x \in X} \Theta(x)\Theta(y|x)$ which proves that

$$p(z|y) = \Theta(z|y). \tag{27}$$

Now, we have shown for this Bayesian network that its topology and CPTs in fact represent the FJD distribution by

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Theta(x_i | parents(x_i)). \tag{28}$$

2.3.3 Exact Inference by Enumeration

To explain exact inference in Bayesian networks we introduce another example (taken from [25]). Suppose we are out of town and our house has a burglar alarm installed. This alarm can not only be set off by a burglar but also erroneously by an earthquake. There are two neighbors John and Mary, who call when the alarm was set off, but who might also call for other reasons. This situation can be modeled using a Bayesian network, whose topology is shown in Figure 9.

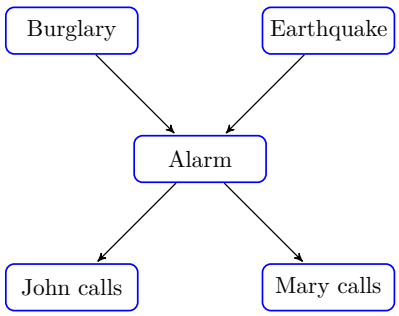


Figure 9: Bayesian network of burglary example

This Bayesian network encodes independence of *Burglary* and *Earthquake*

$$p(\text{Burglary, Earthquake}) = p(\text{Burglary})\, p(\text{Earthquake})$$

and conditional independence of *John calls* and *Mary calls* given *Alarm*

$$p(\text{John calls, Mary calls} \,|\, \text{Alarm}) = p(\text{John calls} \,|\, \text{Alarm})\, p(\text{Mary calls} \,|\, \text{Alarm}),$$

John calls and *Burglary* given *Alarm*

$$p(\text{John calls, Burglary} \,|\, \text{Alarm}) = p(\text{John calls} \,|\, \text{Alarm})\, p(\text{Burglary} \,|\, \text{Alarm}),$$

John calls and *Earthquake* given *Alarm*

$$p(\text{John calls, Earthquake} \,|\, \text{Alarm}) = p(\text{John calls} \,|\, \text{Alarm})\, p(\text{Earthquake} \,|\, \text{Alarm}),$$

Mary calls and *Burglary* given *Alarm*

$$p(\text{Mary calls, Burglary} \,|\, \text{Alarm}) = p(\text{Mary calls} \,|\, \text{Alarm})\, p(\text{Burglary} \,|\, \text{Alarm}),$$

as well as, *Mary calls* and *Earthquake* given *Alarm*

$$p(\text{Mary calls, Earthquake} \,|\, \text{Alarm}) = p(\text{Mary calls} \,|\, \text{Alarm})\, p(\text{Earthquake} \,|\, \text{Alarm}).$$

In the following, we will denote every of the five random variables by its first letter.

As in any probabilistic inference system, we are interested in the posterior probability $p(X|E)$, where X is the query variable and E is the evidence. The variable Y denotes the hidden variables we have to sum over, as we will see in Equation (31). With regard to the previous burglary example, a typical query could be the probability whether there is a burglary given that John and Mary call: $p(B|j, m)$.

The idea behind the inference algorithm is to rewrite the posterior probability as the joint probability of the query variable and the evidence variable and divide the expression by the prior of the evidence:

$$p(x|e) = \frac{p(x, e)}{p(e)}. \tag{29}$$

As the posterior probability is a function of x, we can replace the denominator by a constant α and describe the posterior by marginalizing y out the full joint distribution:

$$p(x|e) = \alpha p(x, e) \tag{30}$$

$$= \alpha \sum_{y \in Y} p(x, e, y). \tag{31}$$

The FJD can then be expressed in terms of the the probabilities in the CPTs. Applying these steps to the example query $p(B|j, m)$ yields

$$p(B|j, m) = \alpha\, p(B, j, m) \tag{32}$$

$$= \alpha \sum_{e \in E} \sum_{a \in A} p(B, j, m, e, a) \qquad \text{(marginalization)} \tag{33}$$

$$= \alpha \sum_{e \in E} \sum_{a \in A} p(B)\, p(e)\, p(a|B, e)\, p(j, a)\, p(m|a). \qquad \text{(topology)} \tag{34}$$

In the worst case scenario, we would have to sum out all of the n variables but one, thus leading to a complexity of of $O((n-1)2^{n-1})$, where $n-1$ are the multiplications and 2^{n-1} are the number of combinations of the variable values.

One improvement is to move as many variables as possible out of the sums. This simplifies the calculation for the example to:

$$p(B|j, m) = \alpha\, p(B) \sum_{e \in E} p(e) \sum_{a \in A} p(a|B, e)\, p(j, a)\, p(m|a). \qquad \text{(reordering)} \tag{35}$$

The variable elimination algorithm is a further improvement over the the inference by enumeration algorithm. Looking at Figure 10, which visualizes each calculation step of Equation (35), we see that there are calculations which are performed multiple times. The variable elimination algorithm prevents repeated calculations by calculating them once and storing the results. A full description of the variable elimination algorithm can be found in [25].

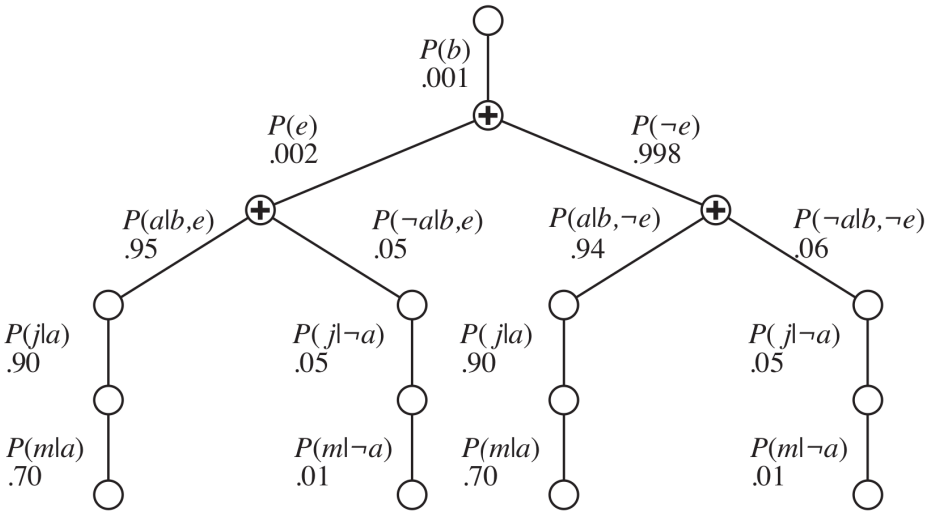


Figure 10: Visualized evaluation of the query $p(B|j,m)$, where $B = True$ [25]

2.3.4 Parameter learning

We will now explain how the parameters, which are the probabilities in the CPTs, can be learned in a Bayesian network that meets the following criteria:

- Known structure (topology)
- All random variables (nodes) are observable
- No missing data

Think of another manufacturer that provides candy bags, whose lime-cherry ratio $\Theta \in [0, 1]$ is unknown for each new bag. For each new bag the ratio is sampled from a uniform prior distribution $p(\Theta)$. We can represent this as a Bayesian network with one node (flavor), as shown in Figure 11.

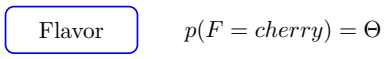


Figure 11: Minimal Bayesian network with one random variable flavor

As we unwrap N candies, the likelihood of the data can be calculated for each possible lime-cherry ratio Θ by

$$p(\mathbf{d}|\Theta) = \prod_{j=1}^{N} p(d_j|\Theta) = \Theta^c(1-\Theta)^l, \tag{36}$$

where the vector $\mathbf{d}$ contains the flavor of each of the N candies, c denotes the number of cherry candies in $\mathbf{d}$ and l the number of lime candies in $\mathbf{d}$.

Our objective is to determine the Θ, that maximizes the likelihood of the data. As the logarithm of a function keeps the function's monotonicity in place, the extrema will be the same. The log-likelihood $L(|\Theta) = \log(p(\mathbf{d}|\Theta))$ transforms the products into summations, which are easier to calculate for a computer and are also simpler to differentiate. The log-likelihood L can be

expressed in terms of Θ, c and l by

$$L(\mathbf{d}|\Theta) = \log p(\mathbf{d}|\Theta) \tag{37}$$

$$= \sum_{j=1}^{N} \log p(d_j|\Theta) \tag{38}$$

$$= c \log \Theta + l \log(1-\Theta). \tag{39}$$

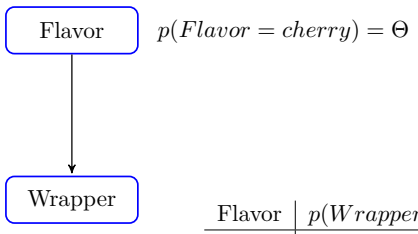
Next, the log-likelihood is differentiated to determine its maximum:

$$\frac{dL(\mathbf{d}|\Theta)}{d\Theta} = \frac{c}{\Theta} - \frac{l}{1-\Theta} = 0. \tag{40}$$

Solving for Θ, it follows that Θ, that maximizes the likelihood, is exactly the ratio of cherries in the bag:

$$\Rightarrow \Theta = \frac{c}{c+l} = \frac{c}{N}. \tag{41}$$

We now extend the example and suppose that the wrapper of the candies are selected according to an unknown conditional distribution $p(wrapper \in \{red, green\}|flavor \in \{cherry, lime\})$.



Flavor	$p(Wrapper = red\|Flavor)$	$p(Wrapper = green\|Flavor)$
cherry	Θ_1	$1-\Theta_1$
lime	Θ_2	$1-\Theta_2$

Figure 12: Bayesian network of extended candy example

Our objective again is to learn the parameters Θ, Θ_1 and Θ_2 after having unwrapped N candies of which c are cherries and l are limes. We denote the number of cherry candies having red wrapper as r_c, the number of cherry candies having green wrappers as g_c, the number of lime candies having red wrappers as r_l and the number of lime candies having green wrappers as g_l.

The likelihood of the unwrapped candies $\mathbf{d}$ given the parameters is given by

$$\begin{aligned} p(\mathbf{d}|\Theta, \Theta_1, \Theta_2) = & \; p(Flavor = cherry)^c \, p(Flavor = lime)^l \\ & p(Wrapper = red|Flavor = cherry)^{r_c} \\ & p(Wrapper = green|Flavor = cherry)^{g_c} \\ & p(Wrapper = red|Flavor = lime)^{r_l} \\ & p(Wrapper = green|Flavor = lime)^{g_l} \end{aligned} \tag{42}$$

$$= \Theta^c (1-\Theta)^l \Theta_1^{r_c} (1-\Theta_1)^{g_c} \Theta_2^{r_l} (1-\Theta_2)^{g_l} \tag{43}$$

Again we calculate the log-likelihood and determine the parameters that maximize it

$$L = \log(p(\mathbf{d}|\Theta, \Theta_1, \Theta_2)) \tag{44}$$

$$= [c \log \Theta + l \log(1-\Theta)] + [r_c \log \Theta_1 + g_c \log(1-\Theta_1)] + [r_l \log \Theta_2 + g_l \log(1-\Theta_2)], \tag{45}$$

which once again are the respective frequencies.

A final note on missing data: If our dataset did not provide any samples e.g., having flavor cherry then according to Equation (43) we would learn that the probability of getting a cherry flavored candy is 0. This means that any query involving a cherry flavored candy, would have 0 probability (see Equation (45)).

In text classification we often deal with missing data, which is why we need to compensate for this. One approach which is called Lidstone smoothing adds exactly one sample to each feature

$$\Theta = \frac{c + \alpha}{N + d\alpha}, \tag{46}$$

where the smoothing parameter $\alpha = 1$ and d is the number of features [3]. For $\alpha \in [0, 1)$ this is called Laplace smoothing, where a value of 0 means no smoothing at all. Note, that the smoothing parameter α and the number of features d are unrelated to the preceding nomenclature regarding Bayesian networks and have been chosen to be consistent with the nomenclature used in the literature regarding smoothing in natural language processing.

2.4 Naïve Bayesian Networks

Naïve Bayesian networks are a special case of Bayesian networks. At the root of the network, there is the class variable, that is to be predicted. The features follow directly as the leaves of the network. Therefore, naïve Bayesian networks always have exactly two layers. The term naïve stems from the fact, that the network assumes conditional independence between its features given the class.

Figure 13 shows such a naïve Bayesian network. Its objective is to classify emails whether they are spam or not based on the words present in the email. At the root we have the class variable *Spam*. The features *Lottery*, *Nudes* and *Viagra* provide strong evidence for spam if any of those is present in an email. In contrast, if any of the words *Science*, *Computer* or *Journal* is present, we can be quite sure that this is a regular email. Using this network, we can give an example why this network is rather naïve: Its conditional independence property states, that $p(science \,|\, journal, \neg spam) = p(science \,|\, \neg spam)$, which means that in a regular mail the probability of the word science is not influenced whether the mail also contains the term journal or not. This does not really reflect reality. However, the successful application of naïve Bayesian networks for spam detection in the field has been shown by Sahami et al. in [26] and others.

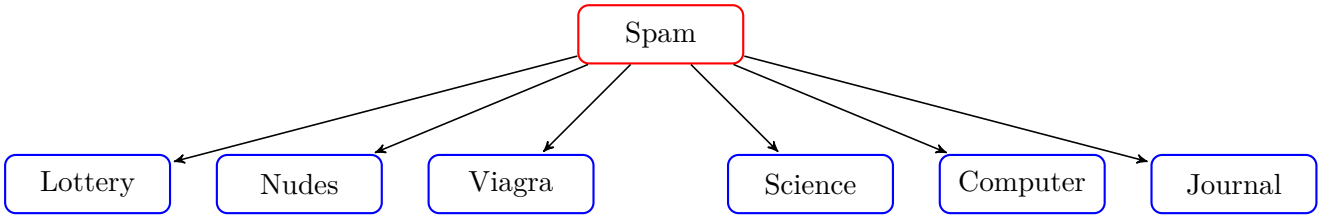


Figure 13: Exemplary Naïve Bayesian Network

A naïve Bayesian network is trained in the same way as explained in Section 2.3.4 for Bayesian networks. Likewise, inference is performed as derived in Section 2.3.3 for Bayesian networks.

2.5 Support Vector Machines

In this section, we explain how support vector machines (SVMs) work. We will follow the derivation approach provided by [1], but extend it by additional steps in between and visualizations.

Before introducing the foundations of SVMs, the reader needs to get familiar with linear discriminant functions, which we from now on call decision boundaries. Our objective is to classify a sample by its position relative to the decision boundary. All samples above the decision boundary are supposed to have the same label (e.g., 1), whereas all samples below the decision boundary are assigned another label (e.g., -1).

In Figure 14, we plotted an exemplary decision boundary (red line), which is defined in terms of the weight vector $\mathbf{w}$ and the bias b by

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b. \tag{47}$$

For an arbitrary point $\mathbf{p}$ that lies on the decision boundary $y(\mathbf{p}) = 0$. Any sample $\mathbf{x}$ that does not lie on the decision boundary, is classified by the following decision rule:

$$1 \quad \text{if} \quad y(\mathbf{x}) > 0 \tag{48}$$
$$-1 \quad \text{if} \quad y(\mathbf{x}) < 0 \tag{49}$$

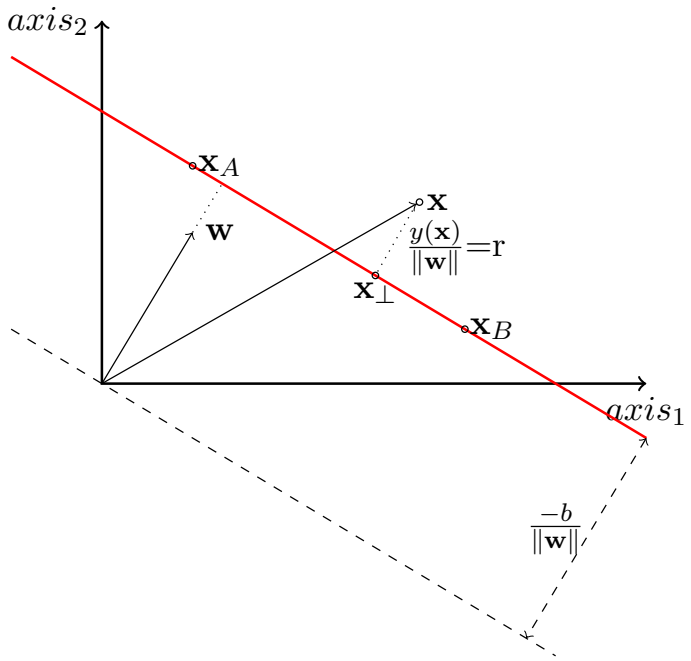


Figure 14: Classification by decision boundary (red line)

Let the points $\mathbf{x}_A$ and $\mathbf{x}_B$ lie on the decision boundary, then $y(\mathbf{x}_A) = 0$ and $y(\mathbf{x}_B) = 0$. Subtracting both equations yields $\mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) = 0$. From this we can conclude that $\mathbf{w}$ is perpendicular to any vector collinear to the decision boundary and therefore determines the orientation of the decision boundary.

The displacement is defined by b. Given a point $\mathbf{x}_A$ that lies on the decision boundary the equation

$$y(\mathbf{x}_A) = \mathbf{w}^T\mathbf{x}_A + b = 0 \tag{50}$$

holds. By applying equivalent transformations, we get $\mathbf{w}^T\mathbf{x}_A = -b$ and normalizing by the magnitude of $\mathbf{w}$ we receive $\frac{\mathbf{w}^T}{\|\mathbf{w}\|}\mathbf{x}_A = -\frac{b}{\|\mathbf{w}\|}$. The left part of the equation is the inner product of the point $\mathbf{x}_A$ and the unit vector $\frac{\mathbf{w}}{\|\mathbf{w}\|}$. Therefore, $\frac{b}{\|\mathbf{w}\|}$ is the shift of the decision boundary from the origin in direction of $\mathbf{w}$.

Next, we show that $y(\mathbf{x})$ returns the actual distance r from the decision boundary. Let $\mathbf{x}$ be a point in the feature space not lying on the decision boundary. Orthogonally projecting $\mathbf{x}$ onto the decision boundary leads to point $\mathbf{x}_\perp$. Therefore, the point $\mathbf{x}$ can be represented by $\mathbf{x}_\perp$ and a shift by a multiple of $\mathbf{w}$. Note, that the vector $\mathbf{w}$ is perpendicular to the decision boundary and therefore the distance can be represented by a multiple of the unit vector $\frac{\mathbf{w}}{\|\mathbf{w}\|}$.

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{\|\mathbf{w}\|}. \tag{51}$$

Multiplying both sides with $\mathbf{w}$ from the left and adding b, we get

$$\mathbf{w}^T\mathbf{x} + b = \mathbf{w}^T(\mathbf{x}_\perp + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) + b \tag{52}$$

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x}_\perp + b + \frac{\mathbf{w}^T r\mathbf{w}}{\|\mathbf{w}\|} \tag{53}$$

$$y(\mathbf{x}) = \|\mathbf{w}\|r \tag{54}$$

$$r = \frac{y(\mathbf{x})}{\|\mathbf{w}\|}. \tag{55}$$

From Equation (53) to Equation (54), we made use of Equation (50). The distance r of a point $\mathbf{x}$ to the decision boundary is given by $r = \frac{y(\mathbf{x})}{\|\mathbf{w}\|}$, as determined in Equation (54). The sign of r can be used to classify any instance in the feature space.

During model training of a SVM, the objective is to determine the model parameters $\mathbf{w}$ and b of the decision boundary

$$y(\mathbf{x}) = \mathbf{w}^T\Phi(\mathbf{x}) + b, \tag{56}$$

where the function $\Phi(\mathbf{x})$ is a feature space transformation, which we explain later.

We presume the set of samples $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, having labels $T = \{t_1, t_2, \ldots, t_N\}$, where $t_i \in \{-1, +1\}$, to be linearly separable in the feature space such that

$$y(\mathbf{x}_i) > 0 \quad \forall \quad i \in \{i \,|\, t_i = 1\} \tag{57}$$

$$y(\mathbf{x}_i) < 0 \quad \forall \quad i \in \{i \,|\, t_i = -1\}, \tag{58}$$

which can be rewritten as

$$t_i y(\mathbf{x}_i) > 0 \quad i \in \{1, 2, \ldots, N\}. \tag{59}$$

In practice, there are usually many possible decision boundaries, that linearly separate the samples based on their class. This is why, we need an objective function that we can optimize to find the best decision boundary. In the SVM approach, we pick the decision boundary, which maximizes the margin, as shown Figure 15. Choosing the decision boundary maximizing the margin, yields the best robustness for outliers and thus a strong generalization performance. The samples closest to the decision boundary are called support vectors.

We will now demonstrate how to frame the optimization problem as a convex optimization problem that can be solved with Lagrange multipliers. The absolute distance of a point $\mathbf{x}_i$ to

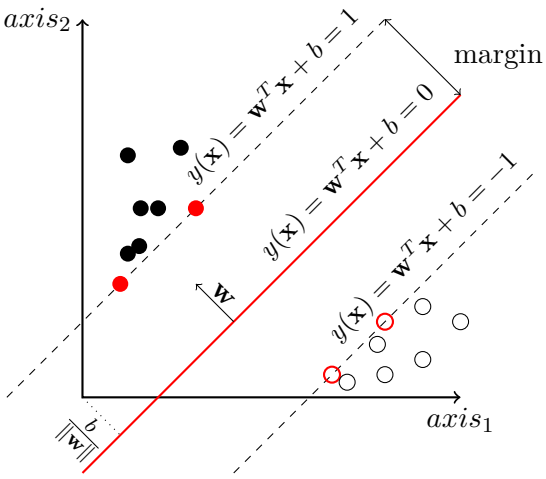


Figure 15: Visualization of the best decision boundary and its margin

the decision boundary is given by

$$|r_i| = \frac{t_i y(\mathbf{x}_i)}{\|\mathbf{w}\|} = \frac{t_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b)}{\|\mathbf{w}\|}. \tag{60}$$

The distance of the closest point is supposed to be maximized, thus the objective function $o(\mathbf{w}, b)$ becomes

$$o(\mathbf{w}, b) = \min_{i \in \{1,2,\dots N\}} \frac{t_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b)}{\|\mathbf{w}\|} \tag{61}$$

$$o(\mathbf{w}, b) = \frac{1}{\|\mathbf{w}\|} \min_{i \in \{1,2,\dots N\}} t_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b). \tag{62}$$

We want to maximize $o(\mathbf{w}, b)$ with respect to $\mathbf{w}$ and b.

Since this optimization function is rather complex, we apply a trick to simplify the function. If we rescale $\mathbf{w} \to k\mathbf{w}$ and $b \to kb$, then the distance of any point $\mathbf{x}_i$ to the decision boundary does not change:

$$r_i = \frac{t_i y(\mathbf{x}_i)}{\|\mathbf{w}\|} \tag{63}$$

$$= \frac{t_i(k\mathbf{w}^T\Phi(\mathbf{x}_i) + kb)}{k\|\mathbf{w}\|} \tag{64}$$

$$= \frac{kt_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b)}{k\|\mathbf{w}\|} \tag{65}$$

Making use of this scaling invariance, we can set the samples closest to the decision boundary to fulfill the constraint $t_i y(\mathbf{x}_i) = t_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) = 1$ and for all the other points $t_i y(\mathbf{x}_i) = t_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1$.

Now, the objective function simplifies to $o(\mathbf{w}, b) = \frac{1}{\|\mathbf{w}\|}$, which we want to maximize. Due to mathematical convenience, we choose to minimize $o^*(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2$ instead, which is mathematically equivalent to the maximization of the original objective function $o(\mathbf{w}, b)$.

The full optimization problem now corresponds to

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{66}$$

subject to

$$t_i(\mathbf{w}^T\Phi(\mathbf{x}_i)+b) \geq 1 \qquad i=1,\ldots,N. \tag{67}$$

We can solve this with the method of Lagrange multipliers. Minimizing the Lagrange function $L(\mathbf{w},b,\mathbf{a})$

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} a_i\{t_i(\mathbf{w}^T\Phi(\mathbf{x}_i)+b)-1\}, \tag{68}$$

where the Lagrange multipliers are denoted by $\mathbf{a} = (a_1, a_2, \ldots, a_N)^T$ and each $a_i \geq 0$. Note, that to minimize L, we have to maximize for $\mathbf{a}$ and minimize for $\mathbf{w}, b$.

Calculating the derivatives of L for $\mathbf{w}, b$ and setting them to zero we get the two equations:

$$\mathbf{w} = \sum_{i=1}^{N} a_i t_i \Phi(\mathbf{x}_i) \tag{69}$$

$$0 = \sum_{i=1}^{N} a_i t_i \tag{70}$$

Plugging $\mathbf{w}$ into L and eliminating b, we get a dual problem $\tilde{L}$, in which we have to maximize over $\mathbf{a}$:

$$\begin{aligned}
L(\mathbf{w},b,\mathbf{a}) &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} a_i[t_i(\mathbf{w}^T\Phi(\mathbf{x}_i)+b)-1] && (71)\\
&= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} a_i t_i \mathbf{w}^T\Phi(\mathbf{x}_i) - \sum_{i=1}^{N} a_i t_i b + \sum_{i=1}^{N} a_i && (72)\\
&= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\sum_{i=1}^{N} a_i t_i \Phi(\mathbf{x}_i) - b\sum_{i=1}^{N} a_i t_i + \sum_{i=1}^{N} a_i && (73)\\
&= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\mathbf{w} + \sum_{i=1}^{N} a_i && (74)\\
&= -\frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{N} a_i && (75)\\
&= -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \Phi(\mathbf{x}_n)^T\Phi(\mathbf{x}_m) + \sum_{i=1}^{N} a_i && (76)\\
&= \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n,\mathbf{x}_m) && (77)\\
&=: \tilde{L}(\mathbf{a}), && (78)
\end{aligned}$$

where $k(\mathbf{x},\mathbf{x}') \equiv \Phi(\mathbf{x})^T\Phi(\mathbf{x}')$ is kernel function, that corresponds to the scalar product in the feature space $\Phi(\mathbf{x})$ is projecting $\mathbf{x}$ to.

With $\tilde{L}(\mathbf{a})$, we can define a dual representation of the prior problem in which we have to

maximize the objective function over $\mathbf{a}$:

$$\arg\max_{\mathbf{a}} \tilde{L}(\mathbf{a}) = \arg\max_{\mathbf{a}} \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{79}$$

subject to

$$0 = \sum_{i=1}^{N} a_i t_i \tag{80}$$

$$a_i \geq 0 \qquad i = 1, \ldots, N. \tag{81}$$

The big advantage of the dual problem representation is the fact that it allows the use of kernel functions. This way, we do not have to project the points to a higher feature space explicitly and also spare the subsequent calculation of the dot product.

The decision boundary is defined in terms of $\mathbf{a}$ by

$$y(\mathbf{x}) = \sum_{i=1}^{N} a_i t_i k(\mathbf{x}, \mathbf{x}_i) + b \tag{82}$$

and can be derived as follows using Equation (69):

$$y(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b \tag{83}$$

$$= \Big(\sum_{i=1}^{N} a_i t_i \Phi(\mathbf{x}_i) \Big) \Phi(\mathbf{x}) + b \tag{84}$$

$$= \sum_{i=1}^{N} a_i t_i \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}) + b. \tag{85}$$

Note, that $\mathbf{x}$ is a point that is passed to the decision boundary function y and that each sample point $\mathbf{x}_i$ is element of the sample set earlier denoted by X.

The value of b can be calculated by

$$b = \frac{1}{N_S} \sum_{n \in S} \Big(t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \Big), \tag{86}$$

as proved by [1], where N_S is the number of support vectors and S is the indices set of the support vectors.

Often, even in a high dimensional space the training data is not linearly separable, which is why we would not find a solution using the approach presented so far. To solve this problem, one approach is to add slack variables to the constraints, which allow each training sample to meet the criteria. However, for each slack variable we add a cost proportional to its value to the objective function. This approach, which is called soft margin classification, allows the decision boundary to correctly divide the majority of the data, while not taking few noisy samples into account. A full derivation of this approach can be found in [20].

2.6 Feature Selection Strategies

In this section, we introduce two different methods to intelligently select a subset of features (here: words) for training and evaluation.

2.6.1 Mutual Information

Mutual Information is a measure to determine the mutual dependence between two random variables. Given the two random variables X and Y, we calculate the mutual information I by

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}. \tag{87}$$

which boils down to how much information the random variable X contains about random variable Y.

The concept of mutual information is closely related to entropy and conditional entropy, which we define now, in order to highlight certain properties of mutual information.

The amount of information a random variable contains is usually being measured in terms of the entropy[4] of a random variable:

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{88}$$

Conditional entropy is defined by

$$H(X|Y) = -\sum_{y \in Y} \sum_{x \in X} p(x,y) \log p(x|y), \tag{89}$$

which is the amount of information that Y does not reveal about X.

The relationship between mutual information, entropy and conditional entropy can be represented in a Venn diagram, as shown in Figure 16.

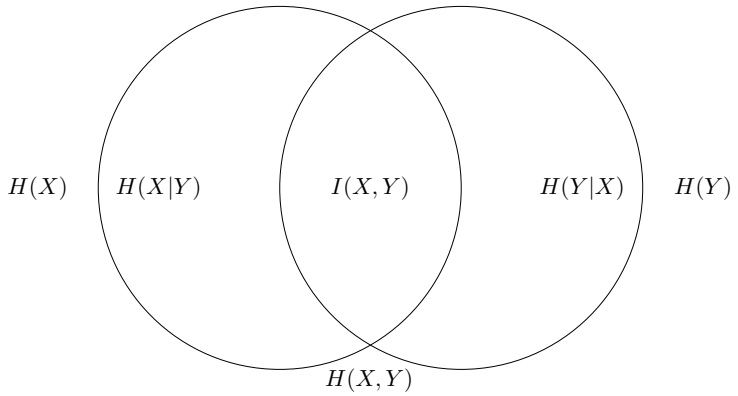


Figure 16: Mutual information and entropy relationship

We want mutual information to be zero if two random variables are independent of each other and to be maximum if the two variables are deterministic functions of each other (i.e., maximum dependency possible). We now show that these two properties are fulfilled by the mutual information concept.

If random variable X is independent from random variable Y, we expect the mutual information

[4] "It is said that von Neumann recommended to Shannon that he use the term entropy, not only because of its similarity to the quantity used in physics, but also because 'nobody knows what entropy really is, so in any discussion you will always have an advantage'." [1]

to be zero. Hence, let X be $X \perp Y$, then the mutual information amounts to

$$I(X,Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \tag{90}$$
$$= \sum_{y\in Y}\sum_{x\in X} p(x,y)\log 1 \tag{91}$$
$$= 0. \tag{92}$$

We now derive the second property, which states that mutual information is maximum, if the variable Y is a deterministic function of X and vice versa:

$$I(X,Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \tag{93}$$
$$= \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x|y)p(y)}{p(x)p(y)} \tag{94}$$
$$= \sum_{y\in Y}\sum_{x\in X} p(x,y)(\log p(x|y) - \log p(x)) \tag{95}$$
$$= \sum_{y\in Y}\sum_{x\in X} p(x,y)\log p(x|y) - \sum_{y\in Y}\sum_{x\in X} p(x,y)\log p(x) \tag{96}$$
$$= \sum_{y\in Y}\sum_{x\in X} p(x,y)\log p(x|y) - \sum_{x\in X}\log p(x)\sum_{y\in Y} p(x,y) \tag{97}$$
$$= \sum_{y\in Y}\sum_{x\in X} p(x,y)\log p(x|y) - \sum_{x\in X} p(x)\log p(x) \tag{98}$$
$$= H(X) - H(X|Y). \tag{99}$$

This property can be directly deduced from the Venn diagram in Fig. 16. If our objective is to maximize $I(X,Y)$, we have to minimize $H(X|Y)$ such that it approaches zero. If the two circles of the random variables completely overlap, then the conditional entropy becomes zero, meaning that if we know the value of X, then Y can be directly deduced from X and vice versa. Therefore, if X and Y are deterministic functions of each other, the mutual information approaches its maximum.

When performing feature selection based on mutual information, we determine the mutual information between every feature candidate and the classes. We pick the ones having highest mutual information i.e., the features that contain most information about the classes.

2.6.2 χ^2 Test Statistic

The χ^2 test for independence is used to determine if two random variables are independent by calculating the χ^2 test statistic between two sample sets of the random variables. For instance, consider the random variables *Gender*, whose value is one of $\{male, female\}$ and *Empathy*, whose value is one of $\{low, high\}$. We want to determine, if there is no relationship between *Gender* and *Empathy* (i.e., independence between *Gender* and *Empathy*) by evaluating the gender sample set and the empathy sample set.

The null hypothesis h_0 in this case states that there is no relationship between *Gender* and *Empathy*. The alternative hypothesis h_1 states, that there is a relationship between *Gender* and *Empathy*. Our objective is to test whether we can reject h_0. To do so, we calculate the χ^2

test statistic, which is given by

$$\chi^2 = \sum_{e \in \{0,1\}} \sum_{g \in \{0,1\}} \frac{(O_{e,g} - E_{e,g})^2}{E_{e,g}}, \tag{100}$$

where $E_{1,1}$ is the expected frequency of samples having features $empathy = high$ and $gender = female$ under the assumption that random variables *empathy* and *empathy* are independent. The variable $O_{1,1}$ is the number of samples in the sample set that have features $empathy = high$ and $gender = female$ (observations). Both expected frequency and observed frequency can be visualized in a contingency table, as shown in Table 3.

	Gender = male	Gender = female
Empathy = low	$O_{0,0}$, $E_{0,0}$	$O_{0,1}$, $E_{0,1}$
Empathy = high	$O_{1,0}$, $E_{1,0}$	$O_{1,1}$, $E_{1,1}$

Table 3: Contingency table of gender and empathy example

The expected frequency is calculated assuming independence between the two variables:

$$E_{1,1} = O\, p(empathy = high)\, p(gender = female), \tag{101}$$

where O is the sample set size. The probabilities are calculated by the frequencies in the datasets, such that Equation (101) can be expressed by

$$E_{1,1} = O \frac{O_{1,1} + O_{1,0}}{O} \frac{O_{1,1} + O_{0,1}}{O}. \tag{102}$$

The χ^2 test statistic defined in Equation (100) can be interpreted as the summation of all squared differences between the expected frequencies (under the assumption of independence between the two random variables) and the observed frequencies in the sample set normalized by the expected frequencies. If the test statistic has a low value close to zero, then the two random variables are independent (expected frequencies equal observed frequencies). If they exceed a certain threshold, then we can say that they are dependent up to a certain confidence level and can reject h_0.

When using the χ^2 test for feature selection we are not interested whether or not we can reject the null hypothesis. We solely calculate the χ^2 test statistic for each feature and the classes and pick the features that have the highest χ^2 test statistic. For instance, let us say we wanted to distinguish seabass from salmon (both fish species) based on the features $color \in \{bright, dark\}$ and $length \in \{\geq 15cm, < 15cm\}$. Assuming our classifier only allows only one feature, we need to decide for one of the two features. Using the χ^2 test statistic, we would calculate the statistic between *length* and the two classes as well as between *color* and the two classes. We would pick the feature having the highest χ^2 test statistic.

For text classification we use the terms as features and select the ones having the highest χ^2 test statistic.

3 State of the Art and Related Work

This chapter is divided in two parts. In Section 3.1, we show how other researchers have solved the issue of classifying news articles in a variety of domains. We then present the current state of research regarding financial trading systems based on news articles in Section 3.2

3.1 Text Classification

The task of text classification has been well researched over the past years. There are different strategies to vectorize texts e.g., bag-of-words model [20], tf-idf vectorization [20] and n-gram models [2] that can be understood by classification algorithms. Among these classification algorithms there are support vector machines, naïve Bayesian networks, neural networks and others, which have been evaluated and compared to one another regarding text classification, as done by McCallum et al. in [21] for instance. They compared the two types of naïve Bayesian networks, namely the Bernoulli model and the multinomial model and found out, that both models perform similarly well on shorter texts, whereas the multinomial model clearly outperforms the the Bernoulli model on longer texts.

In [27], Schumaker compared different vectorization strategies of news articles (bag of words, noun phrases, named entities) as training input for SVM regression.

Another well explored algorithm for text classification are support vector machines. In [13]. Joachims empirically determined, that support vector machines outperform decision trees, k-nearest neighbor classifiers, as well as, naïve Bayesian networks. While showing a great performance, support vector machines do not need a sophisticated parameter tuning and never showed total failures in his experiments.

As many machine learning algorithms are sensitive to noise and vectorized texts usually span a huge feature space, it is important to limit the number of features. In [8], Forman compared different feature selection strategies like information gain, χ^2 test statistic or bi-normal separation, which outperformed previous feature selection strategies.

Due to he huge feature space, it is critical to have machine learning algorithms, that are optimized for fast training and prediction. Facebook's researchers Joulin et al. [14] therefore developed the algorithm FastText based on neural networks, that is able to classify 500,000 sentences each of them belonging to one of 312,000 classes, in less than one minute on an average multicore CPU.

3.2 Influence of the Media on Stock Prices

Financial news articles deal with a variety of aspects, that describe a company's situation e.g., they comment on a company's prospects based on the company's quarter results or they may even be investigative by uncovering a major scandal. That is why news articles and press releases have been a trader's concern since the beginning of stock trading.

The influence of the media on stock prices has been covered in many financial papers. Tetlock found out in [28], that pessimistic news articles are predictive for falling stock prices. Additionally, in [29] Tetlock et al. found out that negative words in news articles are good predictors for earnings and returns of a company.

There have been many approaches analyzed in recent years to utilize machine learning algorithms for stock market prediction using news articles and press releases. It has been shown by [11], that a naïve Bayesian network is able to weakly predict stock markets. They used a

corpus of 5,000 news articles dealing with a total of 12 stocks to train the model for the three movement classes *up*, *down* and *unchanged* relative to a relevant index.

Mittermayer [23] introduced a news categorization and trading system based on support vector machines. He used press releases and categorized them by the impact they allegedly had on the stock by looking at the respective stock prices. The SVM model outperformed a random trader.

Koppel et al. [15] mined a set of news articles covering 12,000 stories on companies listed on the S&P 500 index. They built an SVM based on the top hundred word features according to information gain. In 70% of the cases their model was able to distinguish good from bad news. However, the effects of the news articles were immediately reflected in the stock prices, which supports the efficient market theory explained in Section 2.1.1. However, Dellavigna et al. [5] analyzed the duration it takes an earnings report to be represented in a stock price. They differentiated between every business day and came to the conclusion that traders are especially inattentive on Fridays. On average the effects had a 70% higher delayed response, thus leading to greater trading opportunities.

In [10], the authors Généreux et al. utilized the dataset by Koppel and Shtrimberg [15] and automatically labeled the news articles based on the stock price change from the time before and after the release. Having trained a linear support vector machine, their results show a pessimistic view on forward looking investment opportunities.

In [6], Das et al. utilized messages about stocks posted on Yahoo!'s message board, to build four classifiers each being based on a different machine learning algorithm. The classifiers were stacked in a boosting fashion. The resulting model, worked better on indexes than individual stocks.

4 Methodology

In this section, we describe the methodology that we applied to build and evaluate the three models introduced in Section 1.3.

The research project is split up into the following parts:

- *Data Acquisition (Section 4.1)*
- *Preprocessing (Section 4.2)*
- *Modeling (Section 4.3)*
- *Evaluation (Section 4.4)*

In each part we dealt with a variety of subproblems and solved these with different approaches, which are explained in this section.

4.1 Data Acquisition

In the data acquisition part we used the following two crawlers to populate our data base:

- Press Release Crawler:
  Uses Rich Site Summy (RSS) feeds to collect news articles
- Google Stock Crawler:
  The latest stocks are crawled from the Google stock API whose endpoint is available at `https://finance.google.com/finance/getprices`[5]. The response format is in CSV, which contains the same columns as in Table 2 in Section 2.1.3.

The press releases need to be put in relation with the corresponding stock price changes. To do that, we first have to determine the company that is mentioned in the press release. Fortunately, almost all press releases name the company symbol in the format (exchange:symbol). Using a regular expression, we are able to filter the company symbol and tag the press releases with the corresponding company.

4.2 Preprocessing

The preprocessing chapter has been partly adapted from [18], due to the large methodological overlap. The press release crawler provides HTML which needs preprocessing. First, the text has to be extracted from the HTML code (text extraction). Then, we apply multiple preprocessing steps (text cleaning) to the raw text string such that we can pass the vectorized representation to the machine learning algorithms.

4.2.1 Text Extraction

HTML code represents a tree structure, where each branch is enclosed by HTML tags. Using BeautifulSoup[6] we traverse that tree until we reach the node that encapsulates the text we want to obtain.

[5] Note, that as of July 31, 2018 this API has been closed.
[6] `https://www.crummy.com/software/BeautifulSoup/`

4.2.2 Text Cleaning

After having extracted the text from HTML code, we apply the following preprocessing steps to the corpus, such that the vectorized result can be passed to other machine learning algorithms.

Tokenization

Tokenization is the process of splitting a document by whitespace and punctuation. The result is a list of tokens.

Removal of stop words and punctuation

Documents usually contain words that are too common in a language such that they do not convey any information. In the English language these are words like *a*, *the*, *is*, *are*, which are called stop words in natural language processing. Fortunately, there are multiple pre-defined stop words lists available. We use of NLTK's stopword list[7] and remove these words from the press releases.

Part-of-speech tagging

Part-of-speech tagging (POS tagging) marks up each token in a document with its part of speech. There are several POS taggers available that build up on a variety of machine learning algorithms including hidden Markov models, neural network and support vector machine approaches. For an extensive list of state of the art POS tagging refer to [4]. Python's NLTK library sticks to the part-of-speech tags defined by the Linguistic Data Consortium of the University of Pennsylvania[8].

Lemmatization for certain POS

The lemmatizer uses the part of speech tags and a dictionary of known word forms to reduce each word to its normalized form. In contrast to stemming, it takes the role of part of speech into account. The token *computing* would be stemmed to *comput*, dropping the *ing* suffix. If *computing* was a verb it would be lemmatized to *compute*, if it was a noun it would keep *computing* as it is already in the normalized noun form.

4.2.3 Count Vectorization

Since we cannot pass press releases directly to the machine learning algorithms, we need to transform them into a vectorized representation. One way to do this is to apply count vectorization, which is sometimes also referred to as the bag-of-words model. Count vectorization builds a vocabulary of all words in a corpus (set of all documents) and counts the number of occurrences of each word for each document. This can be represented as a matrix

$$B_{M,|\mathbf{v}|} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,|\mathbf{v}|} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,|\mathbf{v}|} \\ \vdots & \vdots & \ddots & \vdots \\ b_{M,1} & b_{M,2} & \cdots & b_{M,|\mathbf{v}|} \end{pmatrix},$$

[7] `http://www.nltk.org/`
[8] `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

where M is the number of documents in the corpus, $|\mathbf{v}|$ is the length of the vocabulary $\mathbf{v}$ (set of all distinct words in a set of documents) and an element $b_{i,j}$ represents how often word j appears in document i.

Since this approach takes all the features (distinct words) into account, we have to deal with the two problems:

- Having a huge feature space, this automatically leads to a lot of features which do not contribute to the classification (noise).
- Complex models with many degrees of freedom tend to fit these noisy features, leading to poor generalization performance.

Therefore, to prevent overfitting, we have to carefully select the features with high discriminability.

4.2.4 Feature Selection

For feature selection we treat the class and each term as a random variable. Based on the feature selection scoring functions χ^2 *test statistic* and *mutual information*, which we explained in Section 2.6, we select those features whose random variable provides most information about the class random variable.

4.2.5 Stock Prices Preprocessing

All the stock prices, that we receive from our API, are absolute values in Dollars. Naturally, a stock traded at 1000 \$ per share has a higher volatility compared to a stock, that is being traded at 1 \$ per share. Predicting absolute stock prices would mean, that the classifier has to learn, that higher stock prices have higher price jumps. This can be tricky, especially for stock price values, that the classifier has not been trained on.

Therefore, we need a scale-free measure to be able to compare stock price changes. Tsay introduced the k-periods simple net return in his book [30]. This scale-free measure is defined as

$$R_k(t) = \frac{p_t - p_{t-k}}{p_{t-k}}, \tag{103}$$

where p_t is the stock price at time t and k is the offset that we look into the past for the reference price p_{t-k}. In a nutshell, the k-periods simple net return calculates the return at time t, that we would have received, if we had invested into the company at time $t-k$. The return is normalized by p_{t-k} and thereby scale-free.

When we receive a press release at time t, it does not make sense to only look back in time, because the effect of the press release will happen in the future. Therefore, we introduce a new measure, which we will from now on call *simple net opportunity (SNO)* and define it by

$$O_{k_1,k_2}(t) = \frac{p_{t+k_1} - p_{t-k_2}}{p_{t-k_2}}, \quad \text{with } k_1, k_2 \geq 0. \tag{104}$$

Part of our analysis in Section 6.1.3 will be to determine the parameters k_1 and k_2, such that on average they maximize the absolute value of the SNO for all press releases.

4.3 Modeling

In this section, we first formally define the machine learning problem. Since the sentiment model is practically the same as the impact model but with different labels for each sample, we only define the impact model in Section 4.3.1.

Having generally explained the trivial classifier in Section 1.3, the naïve Bayesian network in Section 2.4 and the support vector machine in Section 2.5, we show how the three algorithms can be applied to the text classification problem in Section 4.3.2, Section 4.3.3 and Section 4.3.4, respectively.

4.3.1 Formal Description of the Machine Learning Problem

Our objective is to learn whether a press release has an impact I or no impact $\neg I$ on the stock price.

Formally, this classification problem can be expressed as follows: Every press release is represented by a vector $\mathbf{d} \in \mathbb{X}$, where $\mathbb{X}$ is the document space. There are many different ways to specify the document space. A simple way is to represent each press release in a vector space that is spanned by the vocabulary $\mathbf{v}$ of the corpus. The vector representation has dimensions $|\mathbf{v}|$ and the j^{th} component counts how often the press release contains the j^{th} word in the vocabulary. In this case the document space can be denoted by $\mathbb{N}^{|\mathbf{v}|}$. This kind of vectorization is called count vectorization, which we already explained in Section 4.2.3

For classification, we need to specify the set of classes, that can be assigned to a press release. Since we want to predict the impact of a press release on the stock prices, we decided to label each press release by one of the two classes $\mathbb{C} = \{I, \neg I\}$, where $\neg I$ corresponds to no impact and I means either a positive or negative impact.

The naïve Bayes network and support vector machine learn the classification function γ, that maps each press release representation $\mathbf{d}$ to one of the classes of $\mathbb{C}$:

$$\gamma : \mathbb{X} \to \mathbb{C} \tag{105}$$

For sentiment prediction, we only consider the press releases that had an impact on the stock prices and reassign the labels *sentiment* (S) or *no sentiment* ($\neg S$), depending on whether a press release had a positive or negative impact, respectively.

4.3.2 Trivial Classifier

This trivial classifier groups the training samples of each class and then ranks for each group the words by their frequency. For each class we only consider the most frequent terms, providing us with two lists as exemplary shown in Table 4.

I	$\neg I$
great	constant
profit	steady
loss	dog
decrease	kitchen
...	...

Table 4: Exemplary word list for each class

For prediction, we count how often each term in the class lists appeared in the press release. The class that has the most correspondences is picked for classification. Note, that the terms in each class are considered of equal importance to prediction. As an improvement one could weight the terms of a list by the frequency they occur in the documents of this class.

To put the trivial classifier into mathematical terms, we first have to count vectorize the press release corpus $\mathcal{D}$ consisting of M documents, which yields the document matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \dots \mathbf{b}_M]^T \in \mathbb{N}^{M \times |\mathbf{v}|}$, where the vector $\mathbf{v}$ denotes the vocabulary of the corpus.

Each document has a class label $c \in \mathbb{C} = \{I, \neg I\}$. We store all the class labels of the press releases in the vector y, where the i^{th} element of y corresponds to the class of the i^{th} press release in the corpus.

Model building

For model building we group the training sample indices by their class label $c \in \mathbb{C}$:

$$G_c = \{i \in \mathbb{N} \,|\, y_i = c\} \qquad \forall \quad c \in \mathbb{C}. \tag{106}$$

So G_I would filter all the document indices in $\mathcal{D}$ belonging to class I.

We can then calculate the term frequencies in documents having class label c by summing up all the $\mathbf{b}_i$, whose index is in G_c:

$$\mathbf{f}_c = \sum_{i \in G_c} \mathbf{b}_i \tag{107}$$

For each class label c we calculate $\mathbf{f}_c$ and store the indices of the k terms having highest frequency in the set T_c.

Prediction

The document we want to predict is given by $\mathbf{s} = [s_1, s_2, \dots, s_{|\mathbf{v}|}]$, where s_i is the count of how often the i^{th} term of the vocabulary $\mathbf{v}$ occurred in the document (count vectorized representation). For each class label c, we calculate how strongly $\mathbf{s}$ belongs to this label and finally pick the one with the highest affiliation by:

$$\text{prediction}(\mathbf{s}) = \arg\max_{c \in \mathbb{C}} \sum_{i \in T_c} s_i. \tag{108}$$

4.3.3 Naïve Bayesian Networks for Text Classification

There are two different types of naïve Bayesian networks for text classification, namely the multinomial model and the Bernoulli model. A comprehensive description of both models can be found in [20]. This section gives a brief overview on both models based on the previously cited textbook's explanatory approach and its notation.

Both models pick the class $c_{MAP} \in \mathbb{C}$, that maximizes the a posteriori probability, which was introduced by Equation (5) in Section 2.2:

$$c_{MAP} = \arg\max_{c \in \mathbb{C}} p(c|\mathbf{d}) \tag{109}$$

$$= \arg\max_{c \in \mathbb{C}} \frac{p(\mathbf{d}|c)p(c)}{p(\mathbf{d})} \tag{110}$$

$$= \arg\max_{c \in \mathbb{C}} p(\mathbf{d}|c)p(c), \tag{111}$$

where c is the class over which we maximize and $\mathbf{d}$ is the given document.

The two models differentiate in how they represent the document $\mathbf{d}$ in the document space $\mathbb{X}$. In the multinomial model, $\mathbf{d}$ is represented by the sequence of terms $< t_1, \ldots, t_{n_d} >$, where n_d is it the length of the document. The Bernoulli model represents $\mathbf{d}$ as a vector having the size of the vocabulary $\mathbf{v}$. A vector element has the value either 1 or 0, depending on whether the token in $\mathbf{v}$ at the same index is present in $\mathbf{d}$, formally $< e_1, \ldots, e_{|\mathbf{v}|} >$, where $e_i \in \mathbb{X} = \{0, 1\}^{|\mathbf{v}|}$.

The multinomial model assumes conditional independence of each term in the text given the class and positional independence of terms, which can be expressed by

$$p(\mathbf{d}|c) = p(< t_1, \ldots, t_{n_d} > |c) = \prod_{1 \leq k \leq n_d} p(X_k = t_k|c) \tag{112}$$

and

$$p(X_i = t|c) = p(X_j = t|c) \qquad \forall \quad i, j \in \{1, \ldots, n_d\}, \tag{113}$$

where $\mathbf{d}$ is the document of length n_d represented by the vector consisting of tokens t_1 to t_{n_d}. So as given by Equation (112), the probability of $\mathbf{d}$ given class c is the product of the probability of each term given c, thus making use of the conditional independence assumption for any token given the class. Note, that X_k is the random variable at position k in the document over the the vocabulary $\mathbf{v}$. Positional independence is defined by Equation (113), which means that a token given c is equally likely at every position.

The Bernoulli model assumes conditional independence between the presence of vocabulary's tokens given the class, which is expressed by

$$p(\mathbf{d}|c) = p(< e_1, \ldots, e_{|\mathbf{v}|} > |c) = \prod_{1 \leq i \leq |\mathbf{v}|} p(U_i = e_i|c), \tag{114}$$

where U_i is the random variable for the i^{th} token in the vocabulary and e_i value depending whether the i^{th} term is present in $\mathbf{d}$.

A key property of multinomial models is their ability to take multiple token occurrences into account, which Bernoulli models are unable to do. Assuming we have a Bernoulli model that assigns documents to countries: A document dealing generally with Elephants living in Africa but also mentions Asian Elephants once, might be misclassified. That is why generally speaking, the Bernoulli model should be only used for the prediction of shorter texts, whereas the multinomial can deal with longer documents as well.

As important words like *profit* or *loss* are present in almost every press release, it is crucial for impact prediction how often these words are used for each press release. This is why, we chose the multinomial model.

4.3.4 Support Vector Machines for Text Classification

Each count vectorized press release is projected into the corpus' feature space, to perform model training and prediction as defined in Section 2.5

There are multiple kernels available. We will evaluate the methods on polynomial kernels with different degrees. String kernels, which have been introduced by the authors of [16], have shown promising results in recent research according to [20]. As they are not implemented so far in the scikit-library, we will consider an implementation and evaluation on press release impact prediction for future work.

4.4 Performance Measures

In binary classification, when predicting an instance we get either one of the four cases, shown in Table 5. This is a representation of the confusion matrix, where TP is the number of positive instances that have been correctly predicted, TN is the number of negative instances that have been correctly identified as negative, FP is the number of instances that were predicted as positive instances, but in fact are negative and finally, FN, which is the number of instances that have been predicted as negative but are actually positive. This matrix representation gives more insights on how the model performs than a single scalar value.

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	True Positive (TP)	False Negative (FN)	P′
	n′	False Positive (FP)	True Negative (TN)	N′
	total	P	N	

Table 5: Confusion matrix for binary classification

We also evaluate the model performance by its *accuracy*, which builds upon the confusion matrix.

The accuracy of a model is defined by

$$\text{acc}_{\text{model}} = \frac{TP + TN}{TP + FP + FN + TN}. \tag{115}$$

It takes all correctly classified instances of a dataset and divides them by the number of all items in the dataset.

The models will be trained and evaluated on stratified sampling sets. Therefore, the accuracy values in the confusion matrix do not directly represent the accuracies expected in the field. As previously shown in Figure 29, only a small fraction of press releases do have a major influence on the stock price. Due to the fact that $p(\neg I) \gg p(I)$, where $p(I)$ is the prior probability of a press release having an impact and $p(\neg I)$ is the prior probability of a press release having no impact, it could be more likely that a sample predicted as having an impact truly has no impact and therefore is just a false positive.

Making use of the prior probabilities, we need to determine $p(I|\hat{y})$ and $p(\neg I|\hat{y})$, where $\hat{y}$, denotes the prediction of a press release to have an impact and $\neg\hat{y}$ denotes the prediction of a press release to have no impact:

$$
\begin{aligned}
p(I|\hat{y}) &= \frac{p(\hat{y}|I)p(I)}{p(\hat{y})} \\
&= \frac{p(\hat{y}|I)p(I)}{\sum_{i\in\{I,\neg I\}} p(\hat{y}|i)p(i)} \\
&= \frac{p(\hat{y}|I)p(I)}{p(\hat{y}|I)p(I) + p(\hat{y}|\neg I)p(\neg I)} \\
&= \frac{\frac{TP}{TP+FN}p(I)}{\frac{TP}{TP+FN}p(I) + \frac{FP}{FP+TN}p(\neg I)}
\end{aligned} \tag{116}
$$

$$
\begin{aligned}
p(\neg I|\hat{y}) &= \frac{p(\hat{y}|\neg I)p(\neg I)}{p(\hat{y})} \\
&= \frac{p(\hat{y}|\neg I)p(\neg I)}{\sum_{i\in\{I,\neg I\}} p(\hat{y}|i)p(i)} \\
&= \frac{p(\hat{y}|\neg I)p(\neg I)}{p(\hat{y}|I)p(I) + p(\hat{y}|\neg I)p(\neg I)} \\
&= \frac{\frac{FP}{FP+TN}p(\neg I)}{\frac{TP}{TP+FN}p(I) + \frac{FP}{FP+TN}p(\neg I)}
\end{aligned} \tag{117}
$$

The two probabilities can be directly computed from the confusion matrix and the prior probabilities. If $p(\neg I|\hat{y}) > p(I|\hat{y})$, it is more likely that a press release, that was predicted to have influence, truly has no influence.

For the sentiment model this comparison is not necessary, since the negative and positive press releases are almost balanced, which can also be seen in Figure 29.

5 Implementation

This section deals with the implementation of the approaches and methods for impact and sentiment prediction as suggested in Section 4. We will describe in detail how the press release mining has been worked out in Section 5.1 and show in Section 5.2 how we implemented the stock price crawler. The preprocessing of the press releases using natural language processing tools is covered in Section 5.3. The implementation of each of the three machine learning approaches is described in Section 5.4.

5.1 Press Release Mining

This crawler has been implemented as part of a prior research project and has been documented in [18]. We adapted those texts with minor changes.

As introduced in Section 4.1, we implemented a crawler that uses RSS feeds provided by news outlets to mine press releases. When the crawler retrieved a press release, it will be added to a CSV file by dumping the request's raw HTML response (including the press release's URL, timestamp etc.) into the file. Later on, the preprocessing pipeline reads in the CSV file, extracts the press release texts out of the HTML codes and then uses these texts to build a representation that can be passed to machine learning algorithms.

5.1.1 Crawling

The crawler is implemented in Python and has the following key features:

- **Request limits:** The crawler provides a throttle on a domain basis. This means that we can limit the requests per second for each domain.
- **Scheduling:** The crawler schedules press release downloads based on the press release's feed origin. For each feed we can individually define a download pattern e.g., every 10 minutes 7 times in total. This has been developed for news articles as they might get modified / updated over time.
- **Highly parallel:** To make sure no bottlenecks occur in daily use, every time-consuming task is parallelized. An example for this property is the retrieval of RSS feeds. The predefined feeds list is split up into sublists of equal lengths. For each sublist, we start a job that downloads the feeds. Another example is data storage. There exists one thread whose only purpose is to acquire write jobs from a queue and to write the data to disk. Since queues are thread-safe, we can create new write jobs from any part of the code.
- **No dependencies:** The crawler does not rely on any database management system (DBMS), nor does it rely on any framework. However, it makes extensive use of open-source libraries like requests[9].

As shown in Figure 17, the crawler is object-oriented. In the following paragraph, we will describe the classes and their relationships to each other by starting with the classes that have few dependencies.

Throttle is used to limit the number of requests per second made to a each domain. Every time a *get_pass* of the *Throttle* object is called, a timer is started for the requested domain. This timer blocks another *get_pass* function call for that domain for a pre-defined duration.

[9] `https://github.com/requests/requests`

Requester is a wrapper class for the Requester class defined in the requests library and is extended by a throttling behavior. When calling the wrapper's *request* function, the function calls *get_pass* of the *Throttle*.

The ***Writer*** class waits until a write job is added to a queue and writes the data coming with the write job to the hard drive. Even though the *Writer* class performs all kinds of writes to disk (e.g., writing logs to disk), we only focus on writing downloaded press releases to disk in this description.

Scheduler schedules press release downloads using the *APScheduler*[10] library. It also keeps track of the press releases, that are already known to the scheduler, such that only unknown press releases are scheduled for downloads.

The central unit in the crawler is ***Crawler***. When *Crawler*'s thread is being started, it performs *warmup_iterations* many warmup runs. All press releases that are being collected during these runs, are flagged as known (*Scheduler:known_urls*). RSS feeds do always show the last n press releases and not only the new ones. Since we are solely interested in press releases whose publish date is known, we perform these warmup runs. Afterwards *Crawler* initializes the crawling runs in a fixed interval (*rss_feed_crawl_period*).

In each warmup or crawling run, *Crawler* instantiates ***OutletCrawlers***[11] and passes a *Scheduler* object, a *Requester* object and one of the subsets of the *feeds* list. Then the *Crawler* executes the *run* method of each *OutletCrawler* to initiate the threads. Inside the threads the *OutletCrawlers* retrieve all feeds that are defined in their *feeds* sublist and pass the article URLs to the *Scheduler*.

The *Scheduler* moves all press release download jobs that are due to a queue. The ***DownloadWorker*** reads from that queue and downloads the respective press releases. All the downloaded press releases are then passed to the *Writer*'s queue in order to write them to disk.

Figure 18 shows how the feeds are being retrieved and how the press release download jobs are being scheduled. The *Crawler* initiates a crawling run by calling the *OutletCrawler*s' *run* method.

Each *OutletCrawler* then requests the feeds and passes the press release URLs to the *Scheduler*. The *Scheduler* then schedules the press release download jobs for press releases that have not been known so far.

When a press release download job is due, the *APScheduler* calls a callback function that was set when the press release download job was scheduled. In our case, we defined a callback method, that automatically adds the download job to a *Due Job Queue* (see Figure 19).

On the other end of the queue, there are the *DownloadWorker*s waiting for due press release download jobs. If there is such a job, one of the DownloadWorkers will acquire it (note that queues are generally thread-safe in Python) and download the press release using the *Requester*. After the press release download is finished, the *DownloadWorker* puts the press release into the *Writer Queue*.

The *Writer* instance writes all the press release downloads to disk.

[10] `https://apscheduler.readthedocs.io/en/latest/`

[11] Admittedly, the class' name might be confusing and should be rather named FeedCrawler

[12] The class diagram does only show the attributes and methods that are necessary to understand the structural design of the crawler. For extensive insights please refer to `https://github.com/le1nux/crawly`.

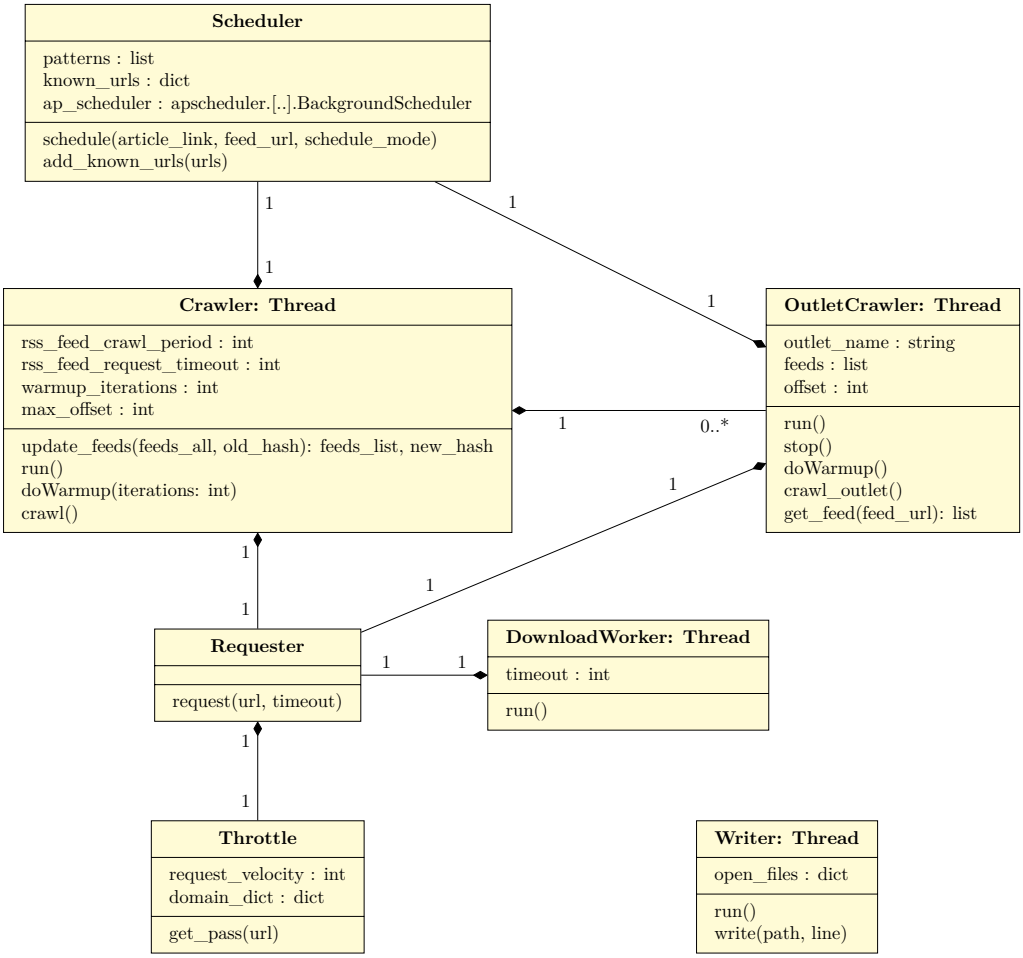


Figure 17: Class diagram[12]of the crawler

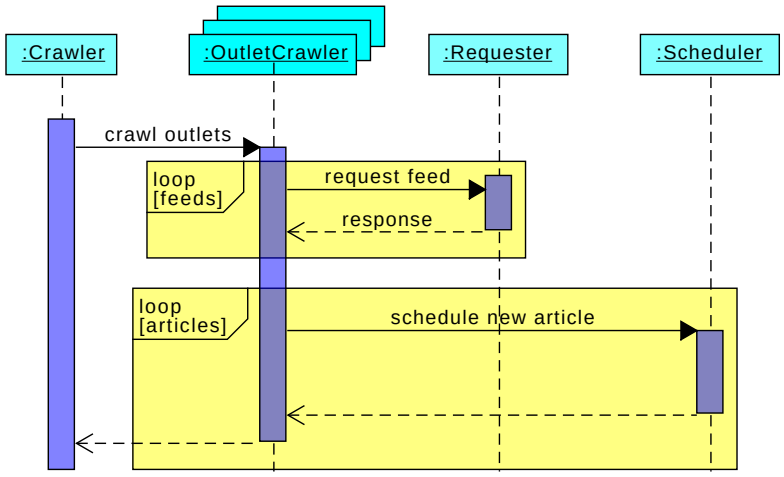


Figure 18: Sequence diagram of feed crawling run

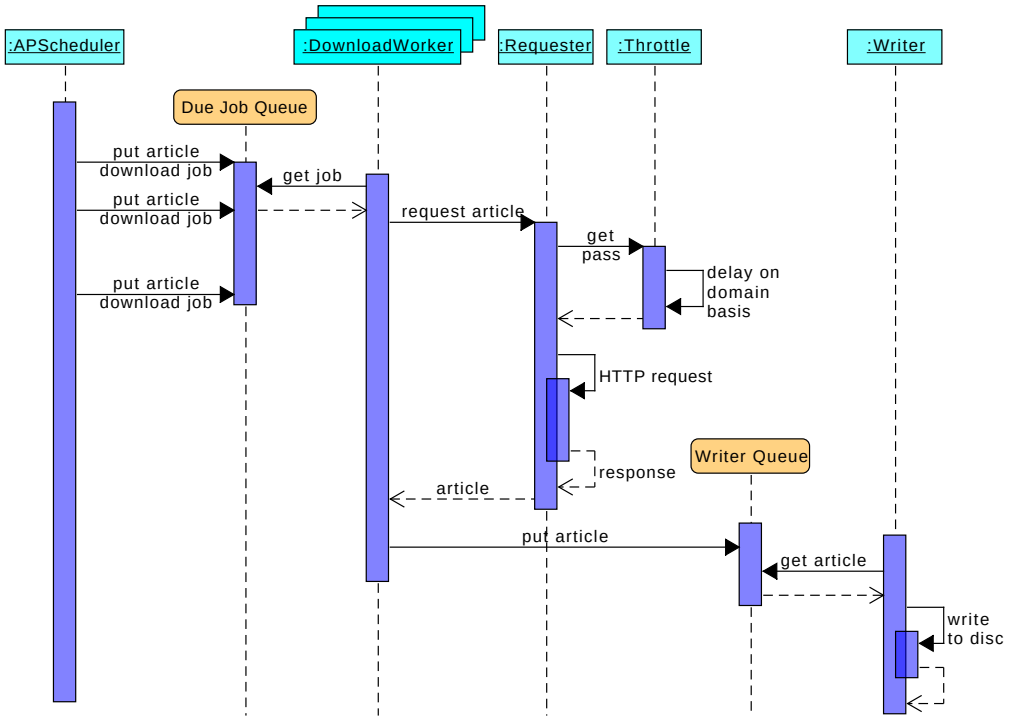


Figure 19: Sequence diagram of article retrieval

5.1.2 Storage

The *Writer* instance writes the downloaded press releases into a CSV file, that has the following column structure:

```
1 timestamp_retrieved, feed_url, link_to_article, html_download
```

The unix time when the press release was downloaded is being stored in the first column. The column *feed_url* stores the URL of the feed where the press release was found in and *link_to_article* stores the URL where the press release was published on the outlet's website. The raw HTML content of the press release download is being dumped into the *html_download* column.

5.2 Stock Price Mining

The stock price crawler is being passed a CSV file containing all companies traded at NASDAQ and NYSE. The file has the format:

```
1 symbol,company,LastSale,MarketCap,IPOyear,Sector,industry,Summary
2 AAPL,Apple Inc.,163.65,$830.36B,1980,Technology,Computer Manufacturing,https
      ://www.nasdaq.com/symbol/aapl
3 ...
```

Every week the crawler is being initiated by a cron job[13]. The crawler then iterates over all stock symbols and for each iteration downloads last week's stock prices from the Google stock API endpoint `https://finance.google.com/finance/getprices`. The result is then dumped into a CSV file.

[13] `http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html`

5.3 Preprocessing

Since we already covered the preprocessing of news article texts in [18], we adopted the text with minor changes regarding press release preprocessing.

The press release crawler dumps plain HTML strings into the *html_download* column of the CSV files enriched with further attributes (columns). Each row is being preprocessed, such that it is in an appropriate shape to be passed to subsequent machine learning algorithms.

There are two stages in the preprocessing pipeline. In the first one, we extract the part that is relevant for us i.e., the press release text and discard the rest (e.g., JavaScript, images or comments). The second stage is about cleaning the texts that we previously extracted, by applying the methods defined in Section 4.2.2.

5.3.1 Extraction Stage

For text extraction BeautifulSoup is being used to locate and extract the information we are interested in.

Since every press release provider has a similar website structure, we developed an algorithm that is parameterized by a JSON dictionary instead of implementing one algorithm per outlet and company description provider. The algorithm works as follows:

```
Given:
  entry points list
  unwanted html tags list ([] allowed)
  tags of interest list ([] allowed)
1)
  a) find path in HTML tree that contains all entry points
  b) keep HTML code at the node of lowest entry point
2) delete all HTML tags that are element of the unwanted html tags list
3) only keep HTML tags that are element of the tags of interest list
4) filter and return remaining text (BeautifulSoup deletes tags
    automatically)
```

Listing 1: Extraction algorithm

A representative parameterization of the algorithm is given below. The filter is defined in JSON and does not only allow to define arbitrary HTML tags but also to specify their attributes and attribute values as shown for the div tag in the entry point section.

```
abcd_outlet_filter =
{
  "entrypoints": [{"tag": "div",
            "attr": "class",
            "value": "article-body"}],
  "delete": [
    {"tag": "script"},
    {"tag": "div", "attrs":
      [{"attr": "class",
        "values": ["ad-container"]}]}
  ]
...
}
```

Listing 2: Exemplary parameterization of extraction algorithm

5.3.2 Cleaning Stage

After the preceding stage has extracted the texts, we now perform the cleaning steps defined in Section 4.2.2, as shown in Listing 3.

The method expects a DataFrame[14] as input arguments, as well as, the name of the DataFrame's column, that contains the text. The first step in the cleaning process is to tokenize the text. Next, all stop words are removed and all remaining tokens are POS tagged. The POS tags and tokens are then passed to the lemmatizer. The lemmatized tokens, that are punctuation symbols or contain digits are removed in the last step.

```
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize, pos_tag
import string

def pre_process_documents(df, column="text"):
    # Tokenization
    df[column] = df[column].astype(str)
    df.loc[:,"tokens"] = df[column].transform(lambda txt: nltk.word_tokenize
    (txt.lower()))

    # stop words
    stopwords_set = set(stopwords.words('english'))
    df.loc[:,"tokens"] = df["tokens"].apply(lambda words: [i for i in words
    if i not in stopwords_set])

    # tagging and lemmatization
    wnl = nltk.stem.WordNetLemmatizer()
    # if wnl.lemmatize(.) is only provided a token without POS it ALWAYS
    lemmatizes it as a noun!
    df.loc[:,"tokens"] = df["tokens"].apply(lambda tokens:[wnl.lemmatize(
    token,pos[0].lower()) if pos[0].lower() in ['a','n','v'] else wnl.
    lemmatize(token) for token,pos in nltk.pos_tag(tokens)])

    # remove punctuation
    df.loc[:,"tokens"] = df["tokens"].apply(lambda tokens: [token for token
    in tokens if token not in string.punctuation])
    # remove numbers
    df.loc[:,"tokens"] = df["tokens"].apply(lambda tokens: [token for token
    in tokens if not any(c.isdigit() for c in token)])

    return df
```

Listing 3: Text cleaning algorithm

In practice, we often deal with huge datasets, that cannot be preprocessed in a single thread anymore from a temporal point of view. This is also the case for our datasets, which is why we split up our original dataset into multiple data frames and create a preprocessing job for each data frame. These preprocessing jobs are then passed to an ipyparallel computation cluster[15]. The preprocessing results are aggregated after all preprocessing jobs are done.

The press releases usually contain the company's ticker symbol and exchange in the format *(exchange:symbol)*. The code snippet in Listing 4 uses a regular expression to extract the ticker symbol from the text, which is then used as the label of the press release.

```
def get_stock_symbol(text):
    nasdaq_regex = re.compile("\((?:Nasdaq|NASDAQ|nasdaq):\s*([a-zA-Z]+)\)")
     #TODO: bad performance
```

[14] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
[15] https://ipyparallel.readthedocs.io/en/latest/

```
    nyse_regex = re.compile("\((?:Nyse|NYSE|nyse):\s*([a-zA-Z]+)\)") #TODO:
    bad performance
    nasdaq_result = nasdaq_regex.search(text)
    nyse_result = nyse_regex.search(text)
    if nasdaq_result:
        return "nasdaq", nasdaq_result.group(1).lower()
    elif nyse_result:
        return "nyse", nyse_result.group(1).lower()
    else:
        return None, None
```

Listing 4: Labeling process of press releases

5.4 Modeling

In this section, the concrete implementation of every approach is being shown. The naïve Bayesian network and SVM model are part of two bigger pipelines which include count vectorization, feature scoring and selection and the respective machine learning model. We explain the implementation of the two models in terms of their complete pipelines. For the trivial classifier we only need to pass the count vectorized samples, which is why there was no need for implementing a pipeline structure.

5.4.1 Trivial Classifier

The trivial classifier's implementation according to the specification in Section 4.3.2 is shown in Listing 5. The algorithm expects the number of features per class (*top_feature_count*) as parameterization. For training, the method *fit* is called, thereby passing the count vectorized press releases X and the corresponding labels as y to the algorithm.

For prediction, we call the method *predict* with the count vectorized press releases X as parameters. For every press release in X the method returns a prediction.

```
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np

class TrivialTextClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, top_feature_count=2):
        self.top_feature_count = top_feature_count

    def fit(self, X, y=None):
        if(X.shape[1] < self.top_feature_count):
            raise Exception('X.shape[1] < top_feature_count')
        y_series = pd.Series(y).reset_index(drop=True)
        self.X = X
        self.top_indices = y_series.groupby(by=y_series).apply(lambda label:
     self.get_top_feature_indices(X[label.index, :]))
        return self

    def get_top_feature_indices(self, X):
        feature_counts = X.sum(axis=0)
        top_features_indices = feature_counts.argsort()[::-1][0 : self.
    top_feature_count]
        return top_features_indices

    def predict(self, X, y=None):
        try:
            getattr(self, "top_indices")
        except AttributeError:
```

```
25              raise RuntimeError("You must train classifer before predicting
    data!")
26          labels = [self.determine_label(x, self.top_indices) for x in X]
27          return labels
28
29      def determine_label(self, sample, indices):
30          top_index = pd.Series(indices.index)
31          indice = np.argmax(np.take(sample, indices.tolist()).sum(axis=1))
32          return top_index[indice]
```

Listing 5: Implementation of trivial classifier

The instantiation of the trivial classifier, its training and prediction is shown in Listing 6.

```
1 classifier = TrivialTextClassifier(top_feature_count=feature_count)
2 classifier.fit(X_train_cv, y_train)
3
4 y_hat = classifier.predict(X_test)
```

Listing 6: Training and prediction of trivial classifier

5.4.2 Naïve Bayesian Network Pipeline

The instantiation of the naïve Bayesian network pipeline is shown in Listing 7. The pipeline contains a count vectorizer, which transforms the raw text into its count vectorized representation. Then the *SelectKBest* model filters the best features according to a feature scoring function. Finally, the count vectorized press releases with reduced features are passed to the multinomial naïve Bayesian network.

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 from sklearn.pipeline import Pipeline
3 from sklearn.feature_selection import SelectKBest
4 from sklearn.naive_bayes import MultinomialNB
5
6 pipe = Pipeline([("count_vectorizer", CountVectorizer()), ("
    feature_selection", SelectKBest()), ("naive_bayes", MultinomialNB())])
```

Listing 7: Implementation of naïve Bayesian network pipeline

After the instantiation, the pipeline is completely unparameterized. During grid search we iteratively set new parameters to find the best parameter settings.

5.4.3 Support Vector Machine Pipeline

The support vector machine pipeline has the same structure as naïve Bayesian network pipeline apart from the naïve Bayesian network which we replaced by the support vector machine, as shown in Listing 8.

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 from sklearn.pipeline import Pipeline
3 from sklearn.feature_selection import SelectKBest
4 from sklearn import svm
5
6 pipe = Pipeline([("count_vectorizer", CountVectorizer()), ("
    feature_selection", SelectKBest()), ("svm", svm.SVC())])
```

Listing 8: Implementation of SVM pipeline

5.4.4 Grid Search

This grid search implementation is inspired by GridSearchCV provided by scikit-learn. The idea behind our implementation is to search for the best parameters for different datasets. The press release set is labeled with continuous simple net opportunities as defined by Equation (104), which need to be discretized for the classification problem we defined in Section 4.3.1. From a scientific point of view, we have neither any prior knowledge nor reliable intuition of discretization choices regarding the simple net opportunity. This is why we test for multiple discretization thresholds to see how the model performances change.

Since finding the best model parameters on multiple datasets is computationally expensive, we made our grid search algorithm embarrassingly parallel such that it can be easily run by Slurm[16]. For every parameter combination Slurm starts a job on the high performance computing cluster (HPC cluster) Bombus[17] at the Hamburg University of Technology. Each job then runs the algorithm specified in Listing 9.

At first, we define a list of discretization thresholds in which each tuple defines the discretization settings for one dataset. The tuples in the class label list define the corresponding class labels. For instance, the first dataset is generated by discretizing the simple net opportunity at the thresholds -0.01 and 0.01. According to the first tuple in the class label list every sample having simple net opportunity of less than -0.01 or greater than 0.01 is assigned 1 and 0 otherwise, meaning impact and no impact, respectively. If we want to perform discretization for the sentiment dataset, the class would be (-1, 0, 1), which corresponds to (negative, neutral, positive) and we would discard all samples having label 0.

Since we are dealing with small datasets, we want to maximize the utility of every sample. This is why, we perform cross validation with 20 folds (*cv_folds*) for every parameter combination on every dataset.

The pipe is parameterized by the dictionary *params*.

The model is then cross validated on every dataset using the pre-defined model parameters. The cross validation results on the training and test set are each averaged, providing us with a sound estimate on how good the model with the specified parameters would have performed on every dataset if we deployed it in the field.

```
1  import numpy as np
2  from sklearn.model_selection import StratifiedKFold
3  from sklearn.pipeline import Pipeline
4  from sklearn.feature_selection import chi2, mutual_info_classif, SelectKBest
5  from sklearn.model_selection import train_test_split
6  from sklearn.model_selection import cross_validate
7
8  discretizer_thresholds_list = [(-0.01, 0.01), (-0.02, 0.02), (-0.03, 0.03),
       (-0.04, 0.04), (-0.05, 0.05), (-0.08, 0.08), (-0.1, 0.1), (-0.15, 0.15)]
9  class_labels_list = [(1, 0, 1), (1, 0, 1), (1, 0, 1), (1, 0, 1), (1, 0, 1),
       (1, 0, 1), (1, 0, 1), (1, 0, 1)]
10 cv_folds = 20
11
12 pipe.set_params(**params) # set hyperparameters passed by slurm
13
14 for i in range(len(discretizer_thresholds_list)):
15     thresholds = discretizer_thresholds_list[i]
16     class_labels = class_labels_list[i]
17
```

[15] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[16] https://slurm.schedmd.com/
[17] https://www.tuhh.de/rzt/services/hpc/hardware.html

```
18      target = press_releases["go"].apply(lambda value: value_discretizer(
        value, thresholds, class_labels))
19      indices_cv, indices_test  = balanced_split(target, test_size=0.2)
20
21      X_cv = press_releases.loc[indices_cv]["tokens"].apply(lambda l: ' '.join
        (l))
22      y_cv = target.loc[indices_cv]
23
24      # perform cross validation
25      cv_results = cross_validate(pipe, X_cv, y_cv, cv=cv_folds,
        return_train_score=True, n_jobs=-1)
26
27      mean_train_score = np.mean(cv_results['train_score'])
28      mean_test_score = np.mean(cv_results['test_score'])
```

Listing 9: Evaluating the model performance for a given parameter on different datasets

6 Results and Evaluation

This section deals with the evaluation of the models' performances. We shortly introduce the datasets and explore their properties in Section 6.1. The impact models' performances and sentiment models' performances are discussed in Section 6.2 and Section 6.3, respectively.

6.1 Data Sets Exploration

We first explore the stock prices dataset and the press release dataset individually and then explore both datasets in an interrelative manner.

6.1.1 Stock Prices Data Set Exploration

From April 9, 2018 to July 27, 2018 the stock price crawler retrieved stock prices of NASDAQ and NYSE listed companies.

As of Oct 16, 2018, the total number of companies listed at NASDAQ and NYSE are 3,307 and 3,144, respectively[18]. The total number of companies indexed by the S&P 500 index amounts to 505. As shown in the first row of Table 6, we could only retrieve all companies for the S&P 500 index and missed approx. 600 companies of NASDAQ and NYSE each. Since the S&P 500 index represents the 505 biggest companies in the US, this indicates that the crawler primarily missed smaller companies.

In order to check data coverage per symbol, we summed up the number of stock prices for each stock symbol. Then, we calculated the average, median, maximum and minimum number of stock prices per symbol. From this small evaluation displayed in Table 6, we can already see that the S&P 500 has by far the best data coverage. The average number of stock prices per symbol is more than twice as high as the average number stock prices of the NASDAQ companies. For the companies traded on NYSE this only looks slightly better.

Additionally, if we compare the minimum and maximum number of stock prices per symbol, it is noticeable, that the spread is especially high for NASDAQ and NYSE. Taking the maximum number of stock prices per symbol as ground truth,which is 30,318 stock prices, we have a data coverage of 3% for the symbol with the least number of stock prices. For this symbol, we could expect two stock prices per hour, assuming that Google provides stock prices for this symbol in equidistant intervals.

With regard to the S&P 500 index the data coverage looks far better. Here, the data coverage is at least 60%, which means in general, that we can expect a stock price update every two minutes in the worst case scenario, given again that Google provides stock prices in equidistant intervals.

Next, we investigate how many stock prices we received for each company[19]. Our approach is to count the number of stock prices for each company and then display these counts by histograms in order to identify important properties in the stock prices datasets. Within the stock crawling time frame there are in total 79 business days, where trading has taken place for 6.5 hours. As we have a stock price resolution of one stock price per minute, we can expect up to 30,810 stock prices per company.

[18] `https://www.nasdaq.com/screening/company-list.aspx`

[19] In the following exploration, we only consider companies for which we received at least one stock price and discard those that we did not receive any stock prices for.

	NASDAQ	NYSE	S&P 500
Retrieved companies	2,742	2,492	505
Total companies at exchange	3,307	3,144	505
No. stock prices	34,175,756	45,028,401	14,410,141
Avg. no. stock prices per symbol	12,463	18,069	28,534
Median. stock prices per symbol	10,438	19,795	29,594
Max. stock prices per symbol	30,318	30,307	30,318
Min. stock prices per symbol	501	514	11,222
Start	2018-04-09	2018-04-09	2018-04-09
End	2018-07-27	2018-07-27	2018-07-27

Table 6: Key figures of each dataset

For each of the NASDAQ, NYSE and S&P 500 companies sets, we have plotted one histogram, as shown in Figure 20. All histograms have the same axes scaling and bins.

In Figure 20a, we plotted the histogram of stock price counts of the companies traded at NASDAQ. From 8,000 stock prices to 28,000 stock prices, company counts are almost constant. At the limits, the company counts increase compared to the company counts in between, whereas the companies having the least stock prices greatly exceed those having most stock prices.

The NYSE histogram, which is shown in Figure 20b, looks rather different from the previous one. From approx. 800 stock prices to 26,000 stock prices the company counts are almost constant. At the right boundary the company counts surge, having a maximum of 145 companies at approx. 30,000 stock prices.

As shown in Figure 20c, the S&P 500 index companies have by far the best data coverage. There are almost no companies having less than 25,000 stock prices. The majority of companies is located around 30,000 stock prices, which reflects a data coverage of almost 100%.

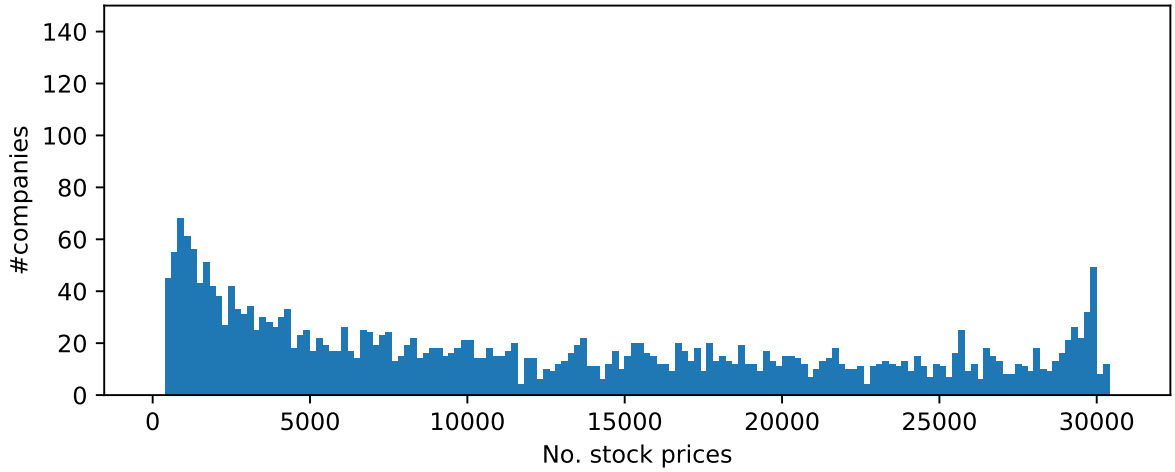
Again, we see, that the stock price crawler primarily retrieved larger companies in terms of market capitalization[20]. The smaller a company, the less stock prices we can expect for this company. The question remains for these companies, whether we retrieve stock prices only for a few days and for the other days no stock prices at all or whether stock price data comes in continuously but at a lower rate.

To check our concern on the data quality (i.e., holes of multiple days) of the smaller companies, we count for each company the number of days, where we received at least one stock price. With regard to Table 7, Table 8 and Table 9, we can conclude, that there are relatively few companies for which we did not receive stock data every day.

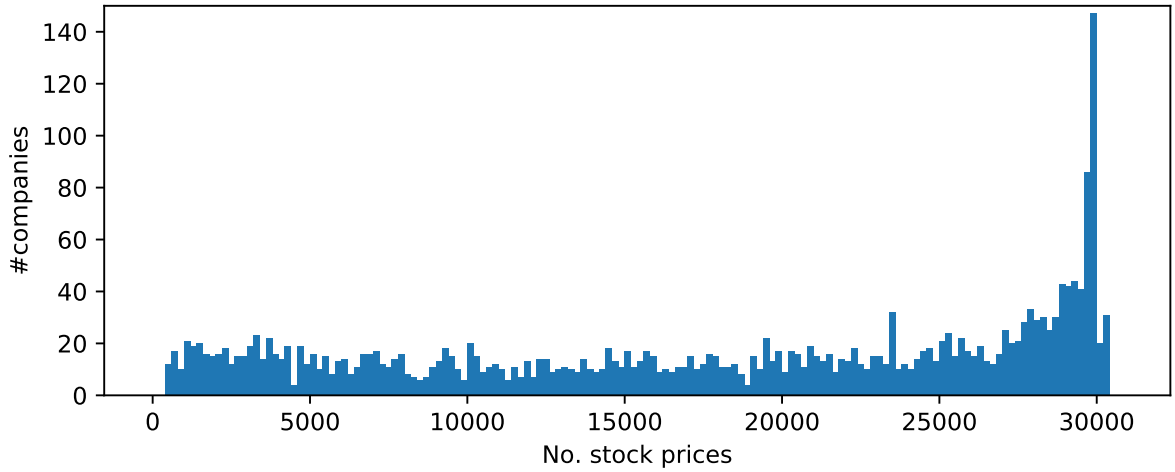
Furthermore, proportionally there are less outliers for the S&P 500 indexed companies compared to NASDAQ and NYSE traded companies. This can be due to multiple reasons. There are companies, which might have changed their ticker symbol, might have stopped or started being traded at one of the exchanges or simply, Google just did not provide sufficient data for these stocks. Due to the better data situation for the S&P 500 indexed companies compared to the sets of NASDAQ and NYSE companies, we can conclude that the issues mentioned above, do not apply to the biggest American companies. Once a company is listed on the S&P 500 index, it is highly unlikely to be unlisted and obviously even unlikelier that it is taken down form the exchange completely.

It is also notable, that the stock price crawler runs weekly for more than 24 hours straight.

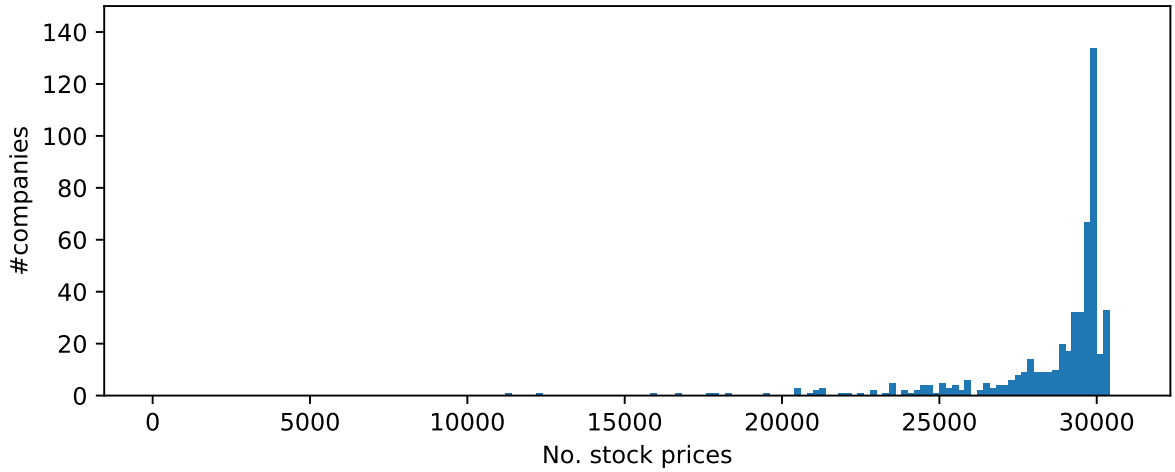
[20]The S&P 500 index lists the 505 biggest companies in terms of market capitalization in the USA.



(a) NASDAQ



(b) NYSE



(c) S&P 500

Figure 20: Histograms of stock prices count per company for NASDAQ, NYSE and S&P 500 companies

This is why, there are some stock symbols that have an additional day of stock prices and the companies in the second last row of Table 7 to Table 9 are not necessarily lacking one day of stock prices.

business days	companies
13	4
14	2
18	1
19	1
22	1
23	4
24	1
25	1
28	3
29	1
32	1
33	2
34	1
38	3
39	2
42	1
46	1
47	1
48	2
49	1
50	1
51	1
53	2
54	2
58	3
59	1
63	4
64	1
65	1
69	3
70	4
71	3
72	3
73	2
74	2
75	3
76	3
77	13
78	2093
79	563

Table 7: Business day counts for NASDAQ companies

business days	companies
14	4
15	2
17	2
18	1
19	1
21	1
23	3
28	2
33	3
36	1
37	1
38	3
39	2
40	1
41	2
42	1
44	2
47	2
48	1
50	1
53	4
57	1
58	5
62	1
63	2
68	4
70	2
71	3
73	2
76	1
77	5
78	1892
79	534

Table 8: Business day counts for NYSE companies

business days	companies
28	1
33	1
41	1
47	1
63	1
78	381
79	119

Table 9: Business day counts for S&P 500 indexed companies

Having shown that we are provided with stock data for almost every company on a daily basis, we now calculate how many stock prices per minute on average we received for each company. The Google API provides stock prices in minute intervals, which is why in the optimal case a company would have exactly one stock price per minute on average. For every company we determine the number of stock prices we received in total within the time frame given in Table 6 and divide them by the number of seconds of such time frame.

The result, which we from now on call stock price retrieval rate r, is the number of stock prices, that we would expect per minute for a company. For clarity, we define r to be

$$r(\text{prices}_c, \text{start}, \text{end}) = \frac{|\text{prices}_c|}{\text{end} - \text{start}}, \tag{118}$$

where prices_c are all stock prices of company c, that have been crawled within the time frame $\text{end} - \text{start}$ given in seconds.

Next, we consider NASDAQ, NYSE and S&P 500 companies separately and calculate the general

descriptive statistics measures on the stock price retrieval rates in each group, as shown in Table 10. Looking at the mean and standard deviation of the three groups, we see that the stock price retrieval rates of NASDAQ and NYSE not only have a higher variance in terms of the stock price retrieval rate, but also are generally way smaller. This observation is supported by the interquartile range (IQR). NASDAQ and NYSE have a bigger spread compared to S&P 500, while the quartiles are also closer to 0.

	NASDAQ	NYSE	S&P 500
mean	40.5%	58.6%	92.6%
std	30.7%	31.6%	8.2%
min	1.6%	1.7%	36.4%
25%	11.8%	30.1%	91.6%
50%	33.9%	64.2%	96.1%
75%	65.9%	89.6%	97.0%
max	98.4%	98.4%	98.4%

Table 10: Descriptive statistics on the stock price retrieval rates of NASDAQ, NYSE and S&P 500 companies

To visualize the numbers, we plotted one boxplot for each of the three company sets in Figure 21. At first glance, one can see that the median of the S&P 500 stock price retrieval rate is close to 1.0. Nevertheless, not all companies provide stock price retrieval rates this high, which is why there are some outliers to the left. These outliers, however, never reach retrieval rates smaller than 0.3. Looking at the boxplots of NASDAQ and NYSE, we see that even the 75% quartiles of both groups never surpass the 25% quartile of the S&P 500 boxplot.

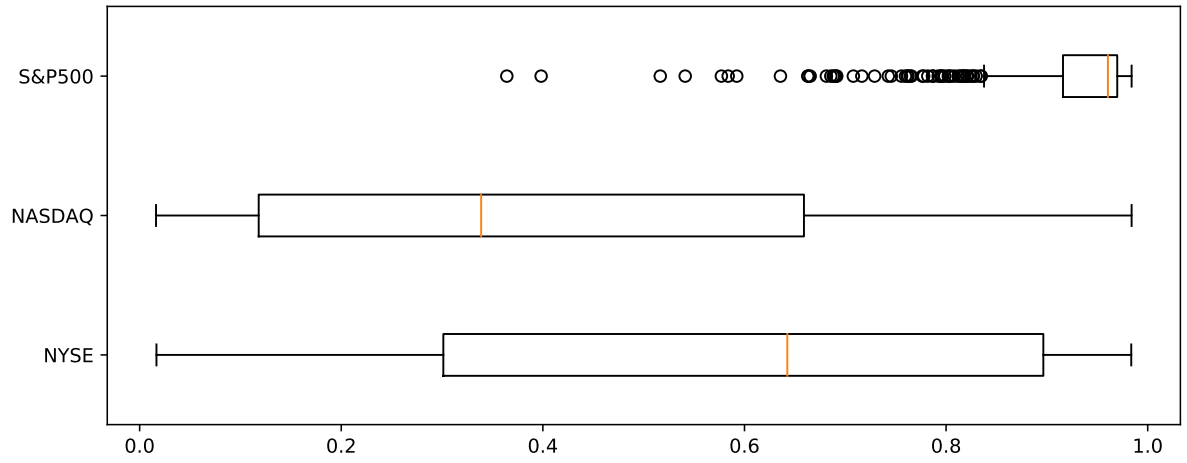


Figure 21: NASDAQ, NYSE, S&P 500 stock price retrieval rate boxplots

In summary, we have shown that many NASDAQ, NYSE and S&P 500 companies are being crawled every day with minor exceptions. Since the number of stock prices per company for NASDAQ and NYSE were very much less compared to S&P 500 indexed companies, our concern was that we might be dealing with huge holes in the NASDAQ and NYSE datasets. By calculating the number of business days with stock prices for each company separately, we had proven that major holes were only the case for vanishingly few companies.

Then, we calculated the stock price retrieval rate for each company over the complete time frame the crawler was running. Evaluating the stock price retrieval rates of NASDAQ, NYSE and S&P 500 separately revealed that the stock price retrieval rate of NASDAQ and NYSE

companies was on average much less than for companies indexed in the S&P 500 index. Also, the stock price retrieval rate of NASDAQ and NYSE companies highly varies in contrast to the stock prices of S&P 500 companies.

How do these insights bring us any further? Due to the occasionally sparsity of NASDAQ's and NYSE's stock price data, labeling of press releases becomes more complicated. Usually, we would calculate the simple net opportunity (SNO) at each point during trading hours by cross correlating the stock prices with a $[\,-\mathbf{1}\,|\,0\,|\ldots|\,0\,|\,1\,]$ kernel and then divide the new time series by the original one. Since we have a lot of missing data and only stock data during trading hours, we would have to find these holes in the stock price series and fill them up with e.g., -infinity to be able to later identify these false values, that were calculated from the holes. For labeling, we would have to merge the stock price dataset and the press release dataset on the company name and the crawl timestamp and then check if we hit a hole. If we hit a hole, we would have to look at the boundaries and decide if both are close enough such that interpolation of the hole makes sense.

Since this approach is complex and error prone, we will follow another one: We will pick the stock price, that is closest to the publish date of the press release and pick the stock price that is closest to the publish date with the additional time offset of e.g., 30 min to calculate the SNO of the press release. However, since there are companies which lack stock prices of multiple hours, we need to implement checks, to keep the integrity of our labeling process. If there is no suitable stock price data available for a press release, we need to discard it. In a nutshell, we do not preprocess all stock prices but only those that are relevant to the press releases.

6.1.2 Press Release Data Set Exploration

The press release crawler retrieves the press releases of all the 6,451 companies traded on NASDAQ and NYSE. From April 17, 2018 to October 8, 2018 the press release crawler retrieved a total of 26,379 press releases covering 3,567 companies traded at NASDAQ or NYSE. A total of 6,637 press releases were published by 393 companies listed on the the S&P 500 index.

All press releases were retrieved from three press release providers, which provided 19749, 3024, and 3606 press releases, respectively.

In our first exploration of the press release dataset, we look at the number of press releases published each day, as shown in Figure 22. The blue and red bars represent the number of press releases published during business days and weekends, respectively. During the weekend by far less press releases are being published compared to business days, thus leading to the 5-2 pattern.

Every three months a company that is traded on NASDAQ or NYSE is supposed to release their quarter results. With an offset of about four to eight weeks to the quarters of a year, a company usually provides their quarter results. During these times the number of press releases spike, which can be also seen in Figure 22. The highest number of press releases per day is reached between the end of April and the beginning of May. Then it continuously decreases to a minimum in the middle of June. From that time on it increases until it spikes again in the beginning of August.

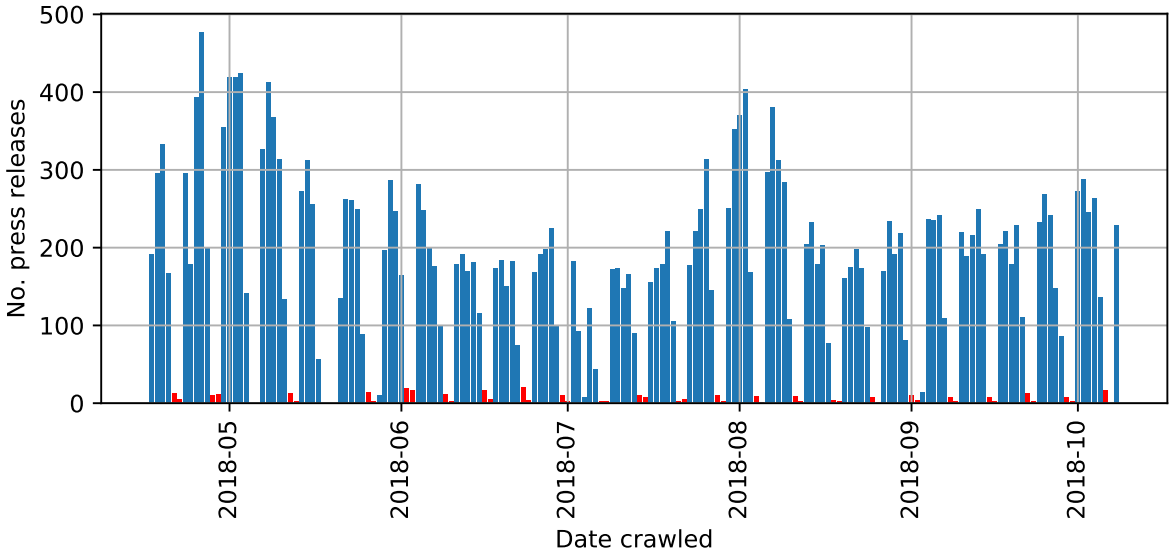


Figure 22: Number of press releases published each day

Next, we look at each day of the week separately and count the number of press releases for each day of week. As shown in Figure 23, most press releases are published during the middle of the week. Mondays and Fridays are the business days with the least press releases. On weekends, there are almost no press releases being published at all.

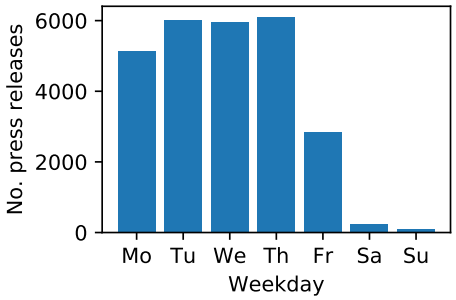


Figure 23: Number of press releases per weekday

Now, let us analyze the number of press releases published at specific hours of the day, as shown in Figure 24. The orange bars represent the after hours and the blue bars show the trading hours. Interestingly, during trading hours only few press releases are published. One reason for this might be that the companies do not want their stock prices to overreact,due to a buy order and sell order imbalance. In this scenario exchanges might even enforce a non-regulatory halt[21]. Shortly, before and after the trading hours, there is a spike in press releases. Especially at 8 PM UTC a maximum of press releases is being published. From 12 PM UTC to 9 AM UTC we receive almost no press releases.

[21] https://www.investopedia.com/terms/t/tradinghalt.asp

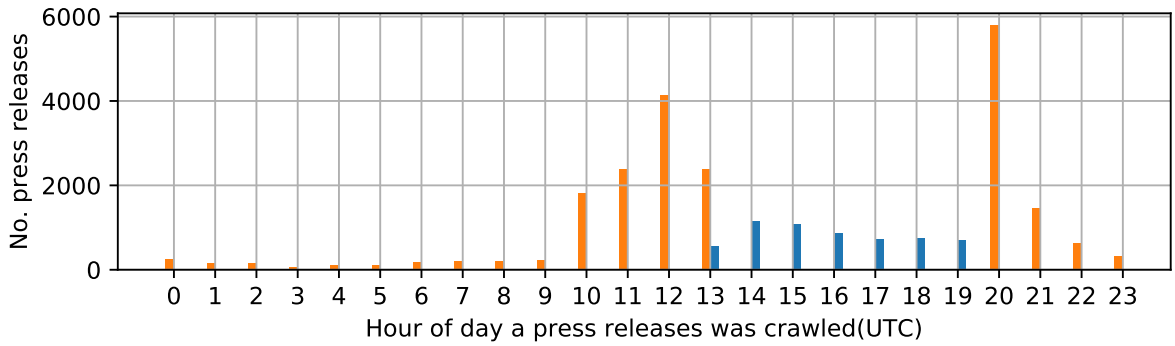


Figure 24: Number of press releases by hour of day

We now continue our exploration by looking at the minute of an hour a press releases is published. Figure 25 shows that most press releases have been crawled within the first ten minutes. From there it continuously decreases to a minimum at minute 60. Besides the downward trend, every ten minutes there is a minimum. This repeating pattern stems from the fact that our crawler checks for new press releases every 10 minutes plus a randomized offset.

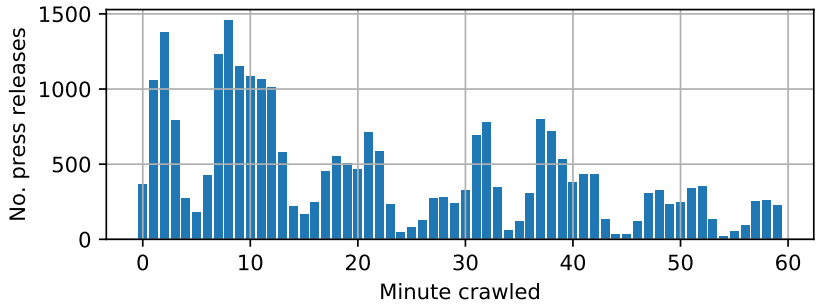


Figure 25: Number of press releases per minute

The histogram in Figure 26 shows the distribution of the number of press releases for each company. Most companies have one to four press releases. Interestingly, in this range only 11% of the companies belong to the S&P 500 index, whereas in the range of above ten press releases, 38% of the companies are indexed by the S&P 500 index. In conclusion, this indicates a positive correlation between the importance of a company (in terms of market capitalization) and the number of press releases it publishes.

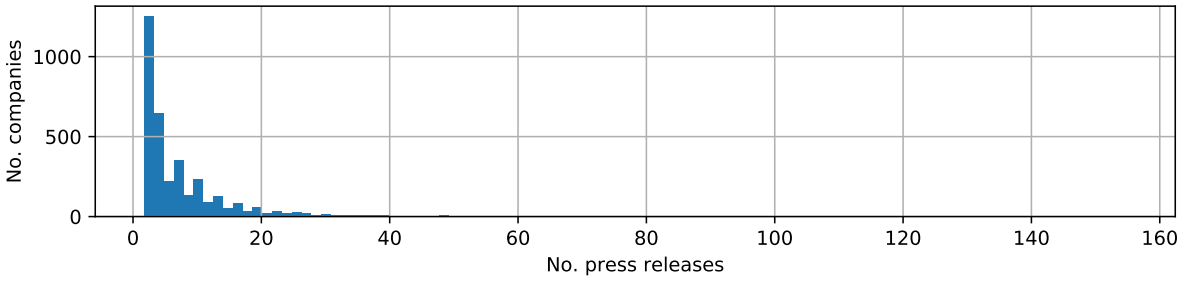


Figure 26: Histogram of number of press releases per company

Furthermore, six companies out of the top ten publishing companies are listed on the S&P 500

index, as shown in Table 12. If all companies traded on NASDAQ or NYSE published the same amount of press releases, we would have only expected 7% to be published by companies listed on the S&P 500 index.

Even when we look at the ten companies that have published the least, there are still two of them in the S&P 500 index. However it is noteworthy, that there are many companies having only one press release. So in Table 11, we might have picked too many S&P 500 companies by chance. In the given dataset, the proportion of S&P 500 companies having one press release over all companies having one press release is 10%.

exchange	symbol	count	in S&P 500
nasdaq	aaon	1	False
nyse	cms	1	True
nyse	cnp	1	True
nyse	cnx	1	False
nyse	cpe	1	False
nyse	cuz	1	False
nyse	cvi	1	False
nyse	cvrr	1	False
nyse	dac	1	False
nyse	dcf	1	False

Table 11: Companies with least press releases

exchange	symbol	count	in S&P 500
nasdaq	amzn	154	True
nyse	acn	149	True
nasdaq	rmni	132	False
nyse	lrn	124	False
nyse	mco	107	True
nyse	ppg	105	True
nyse	awk	103	True
nasdaq	masi	101	False
nasdaq	egov	101	False
nyse	wfc	86	True

Table 12: Companies with most press releases

6.1.3 Joint Exploration

We now set the stock prices dataset and the press releases dataset in interrelation. Our objective is to find out whether press releases influence the stock prices significantly and if so, how long it takes a stock to fully map the information contained in a press release onto the stock price. Answering both questions is critical for labeling press releases with the most accurate simple net opportunity, thereby also directly affecting the model's performance.

As previously shown by Figure 24 and also shown by Figure 27, the majority of press releases are published during the after hours. The blue line in Figure 27 shows the number of press releases being released by either a NASDAQ or NYSE company after the exchange had closed for each day. The orange line shows the same but during trading hours. The green and red line display the press releases published by companies listed on the S&P 500 index and that were released in the after hours and during trading hours, respectively.

Each line expresses a repeating pattern. During the weekend, we receive almost no press releases resulting in a 5-2 pattern. Another pattern repeating on a larger scale, can be seen, when looking at three months periods. Every three months, companies traded on exchanges are obliged to present their quarterly report, which results to the quarterly peak of press releases counts.

Additionally, the majority of press releases are published by NASDAQ and NYSE companies after the exchanges have been closed. On top of that, the number of S&P 500 press releases published during trading hours is similar to the NYSE and NASDAQ press releases published during trading hours. Compared to the previous three lines, the S&P 500 companies during trading hours have the lowest daily press release frequency.

Not only are the press releases published during after hours being overrepresented, they also might have a different effect on the stock market. Traders have plenty of time to react to these

stocks, since the time until the market opens again is usually far away. When the exchange opens, the market might instantly overreact, since there are many traders who already have seen the news and possibly came to the same conclusion. In contrast to this, we expect the stock market to react to press releases published during trading hours not in such a drastic way, because people need time to gather the news and read them.

Only 8% of the press releases are published during trading hours. Labeling of these press releases is more difficult than the SNO, as defined in Equation (104), because it depends on the release time. For after hours press releases we only have to take exchange open time and exchange close time into account, which are constant. This is why, we limit the dataset to press release published during after hours. This leaves us with the majority of the press releases and a dataset that is in itself more consistent.

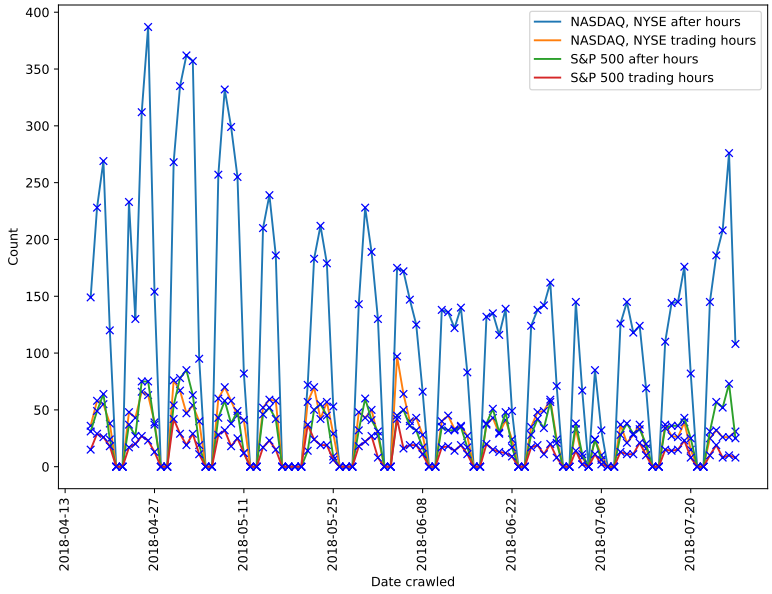


Figure 27: Number of press releases of NASDAQ,NYSE and S&P 500 companies during trading and after hours over time

In order to evaluate the influence of press releases, we have to label each press release with its simple net opportunity. Since we only consider after hours press releases, the calculation depends only on exchange closing and opening time. Our approach, which is shown in Figure 28, takes the last stock prices before the press release has been published i.e., those that are part of window w_1 and calculates the mean stock price. Under the assumption that a press release had magnificent influence, we expect the stock price to diverge significantly from the original stock price when the exchange closed. At this point we suppose for now that the influence will have been reflected in the stock price at least after 5 hours after the exchange had opened. This is why, we set a second window w_2 from 6:30 PM to 7:30 PM and calculate its average stock price.

Subtracting the average stock price of w_2 from the average stock price of w_1 and then dividing by the average stock price of w_1 provides an estimate for the true simple net opportunity of a given press release. If we bought a stock before the press release had been published and sold it five hours after the exchange had opened again, then we could expect this relative change in the stock price.

Note, that the position of w_2 was set arbitrarily by the offset and will be further optimized as part of the joint exploration.

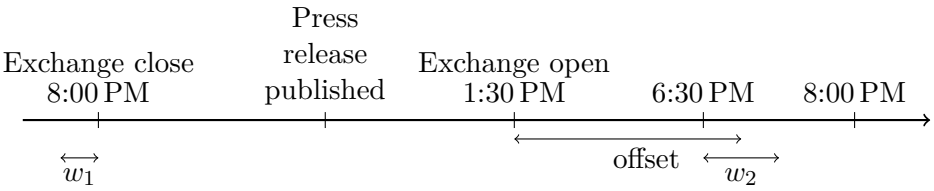


Figure 28: Simple net opportunity calculation of press releases (time in UTC)

Having labeled all press releases accordingly, we plot a histogram of the press releases' simple net opportunity, which is shown by the blue bars in Figure 29. This Gaussian shaped histogram around the origin reveals a general balance between positive and negative press releases.

To put the press releases' SNOs into perspective, we calculate the SNO for every company for each day, when there was no press release published. We call this dataset general SNO set. Finally, we sample for every press release of a company 30 SNOs of this company from the general SNO set and plot this sample set as another histogram. The result is displayed by the orange histogram in Figure 29.

Similar to the previous SNO histogram, this one is also Gaussian shaped with its mean close to the diagram origin. However, its variance is a little lower, thus leading to a narrower shape. This is an important finding, as it clearly shows that for this specific window setting (w_1, w_2) the press releases do, in fact, have an influence on the stock prices.

The SNO labels of the press releases are continuous and need to be discretized for classification. This diagram reveals that the thresholds for such a discretization need to be chosen carefully. For the impact model we have two thresholds (t_1, t_2), where $t_1 < t_2$. Every SNO smaller than t_1 or greater than t_2 is discretized to *impact* (I) and every other SNO is discretized to *no impact* ($\neg I$). Setting the thresholds to $\pm 0.5\%$ would lead to lots of noise among the samples labeled as i. A reasonable choice would be $\pm 8\%$, as the classes *negative* and *positive* would most likely not contain any press releases that had a respective SNO due to usual stock price changes. On the other hand, setting the thresholds far from 0 comes at the expense that samples that truly have an impact end up being labeled $\neg I$ and that the number of samples having class I, are greatly reduced. Analogously, the dataset size problem also arises for the sentiment model.

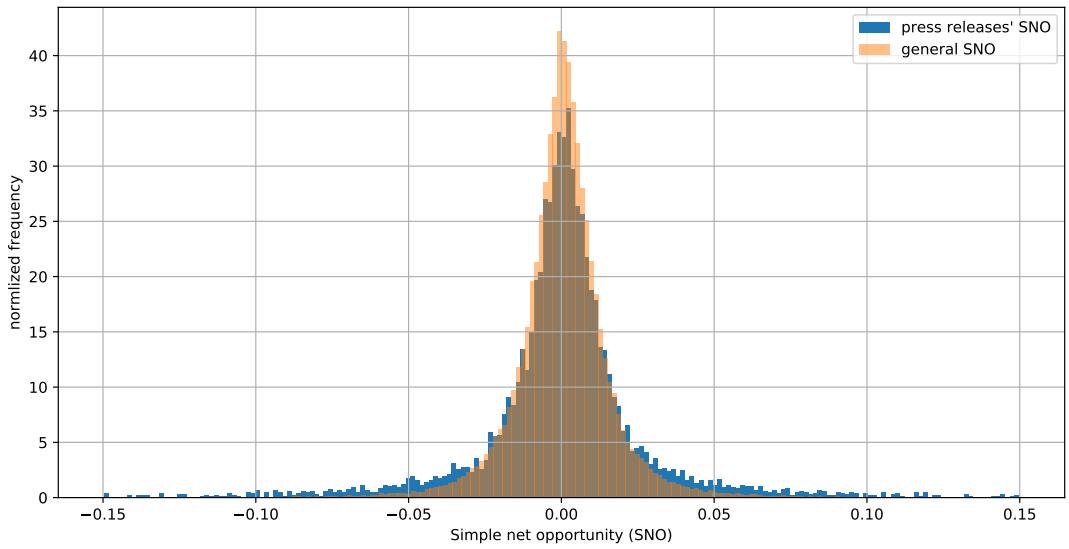


Figure 29: Press releases' simple net opportunity vs. general simple net opportunity with an offset of 5 h

Next, we analyze the impact of the press releases on a per hour basis, as shown in Figure 30. We compare the stock price before and half an hour after the press release was published. There is a special case for press releases published during the after hours, as there are usually no stock prices half an hour after the publishing. This is why we take the close price and the price half an hour after the exchange opened as references. A simple net opportunity less than -3% is considered negative, a simple net opportunity greater than 3% *positive* and all values in between are considered *neutral*. Note, that the discretization thresholds have been chosen by the authors best intuition and we will show later there are better discretization thresholds than $(-3\%, 3\%)$, as well as, a better offset than 0.5 h.

The diagram reveals that during trading hours almost all of the press releases have been labeled *neutral*. At 8 PM UTC press release numbers spike and also the total number of press releases having an impact are highest for this hour. From 12 PM UTC to 10 AM UTC almost no press releases are being published, while they spike again from 10 AM UTC to 1 PM UTC.

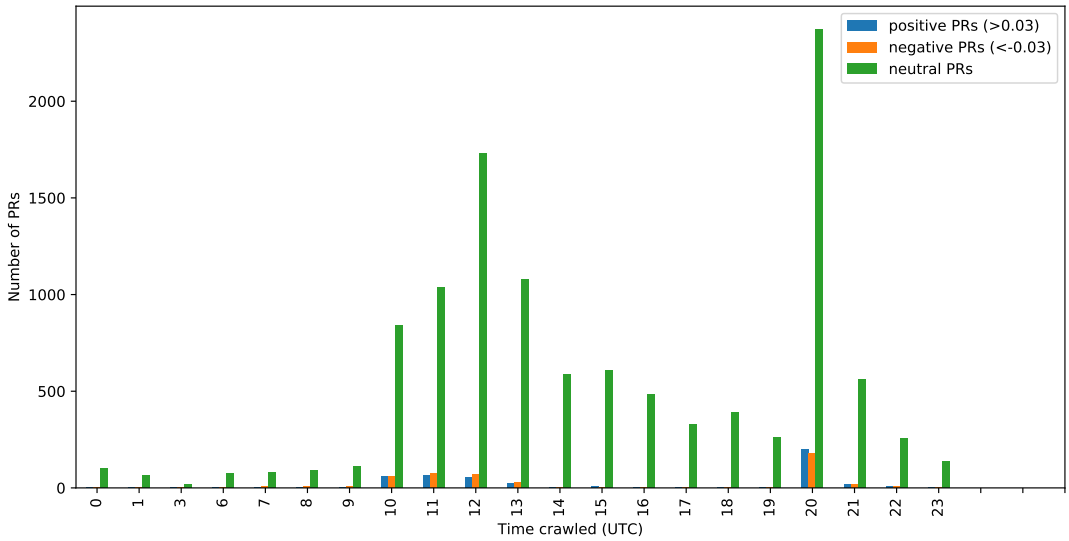


Figure 30: Number of NASDAQ and NYSE press releases published each hour and labeled as one of {pos., neutral, neg.} after discretizing the 30 min SNO on thresholds $[-0.03, 0.03]$

So far, only absolute numbers of press releases have been analyzed. Now, we put the focus on the relative frequency of labels for each hour i.e., all bars of a specific hour sum up to 100%, which is shown in Figure 31. This representation allows to compare the label proportions of every hour. We see that during the after hours the proportions of the labels *positive* and *negative* vary a lot. For instance at 7 AM UTC the negative press releases outnumber the positive ones, whereas at 8 PM UTC the relation is vice versa. This prior knowledge could be exploited as a feature for our sentiment models. Also, if we compare the bars at 8 PM UTC to the bars at 11 PM UTC, we see that it is by far likelier that a press release published at 8 PM UTC has an impact than one published at 11 PM UTC. This is useful prior knowledge for impact prediction.

Unfortunately, these effects might just have arisen due to chance as there are only few press releases with a positive or negative sentiment in each hour bin. Nevertheless, this will be investigated as part of our future work, once we gathered a bigger labeled press releases set.

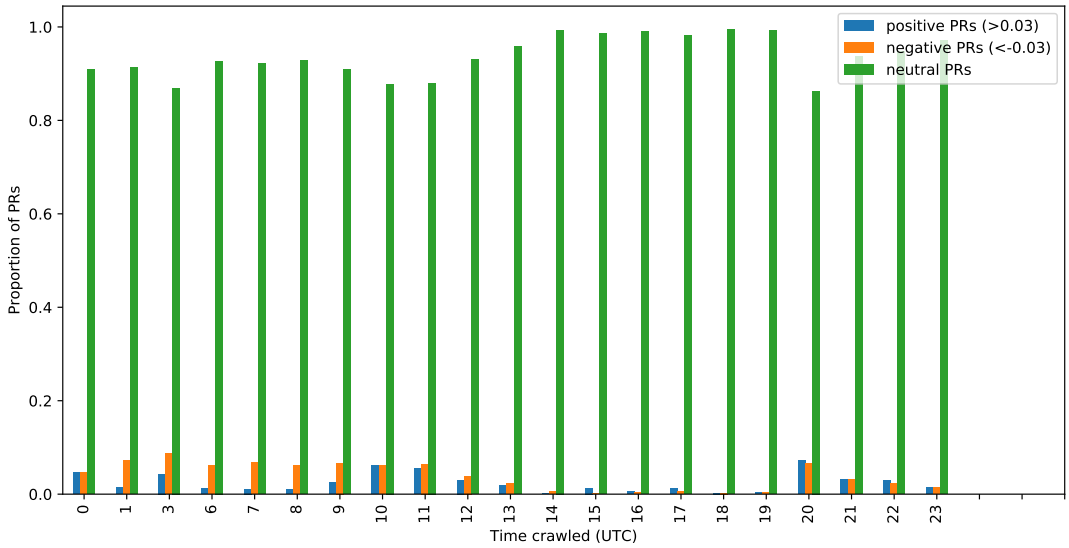


Figure 31: Class proportions of labeled NASDAQ and NYSE press releases published each hour after discretizing the 30 min SNO on thresholds $[-0.03, 0.03]$

At this point, we recapitulate what we have found out so far and what we can infer:

- Almost all press releases are published during the after hours. This is why we decided to limit the model building to this subset of press releases.
- Comparing the press releases' simple net opportunity with a five hours offset to the general simple net opportunity, revealed that press releases indeed show a significant influence beyond normal stock price changes.

Moreover, in the second bullet point we assumed that five hours is a reasonable offset for simple net opportunity calculation. To suffice academic requirements, we now perform an extensive analysis of press release influence by varying the offset, as shown in Figure 32.

The left window w_1 is fixed at 8 PM UTC, while the other window w_2 is being slided over the interval 1:30 PM UTC to 6:00 PM UTC according to different offsets o_i. For each position of w_2, we calculate the mean of each window. The values are then used to calculate the influence the same way as done for the SNO in Figure 28.

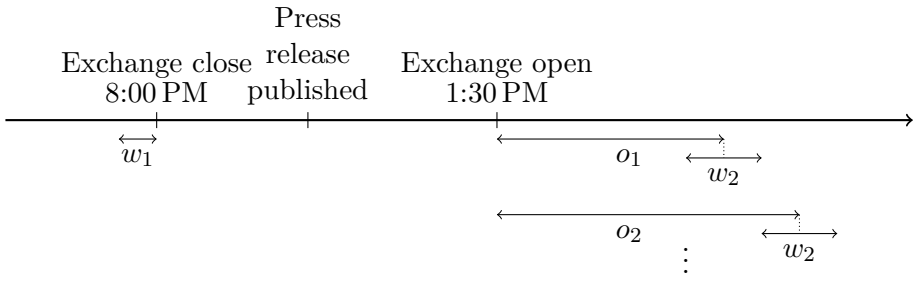


Figure 32: Press release SNO calculation for different windows w

Since we want to the offset after which the press releases' influence is fully mapped onto the stock price and are not interested in the direction of the influence, yet, we only consider the absolute value fo the SNOs (short: ASNOs).

We calculated the mean and median ASNOs on two different press release sets for multiple

offsets. The first dataset (D_1) contains all press releases that have been published during the after hours. The second dataset (D_2) contains only press releases that have an ASNO of more than 3% five hours after the exchange opened. The lengths of window w_1 and w_2 was set to 1 hour and 10 minutes, respectively. The window w_1 has been fixed to the exchange closing time, while the position of w_2 changes with different offsets o_i.

The result is shown in Figure 33, in which the curves show highest slope in the beginning, which then constantly reduces until it reaches almost 0 at 6 PM, UTC thus leading to the logarithmic shaped curves.

The green line shows the mean ASNO calculated on D_2. It exceeds the median ASNO (red line) of D_2 for all offsets o_i. This property suggests that there are few press releases which have an unusually high impact. This insight also holds for the other dataset D_1.

From the median ASNO of D_1, which is at 1%, we can conclude that the majority of the press releases have almost no impact whatsoever. Taking the daily variability of the stock prices into account which has been addressed in Figure 29, assigning a positive or negative sentiment to these press releases would lead to lots of noise. Note, that this finding is independent of the position of window w_2, which we had only shown for the specific offset of 5 h in Figure 29 so far .

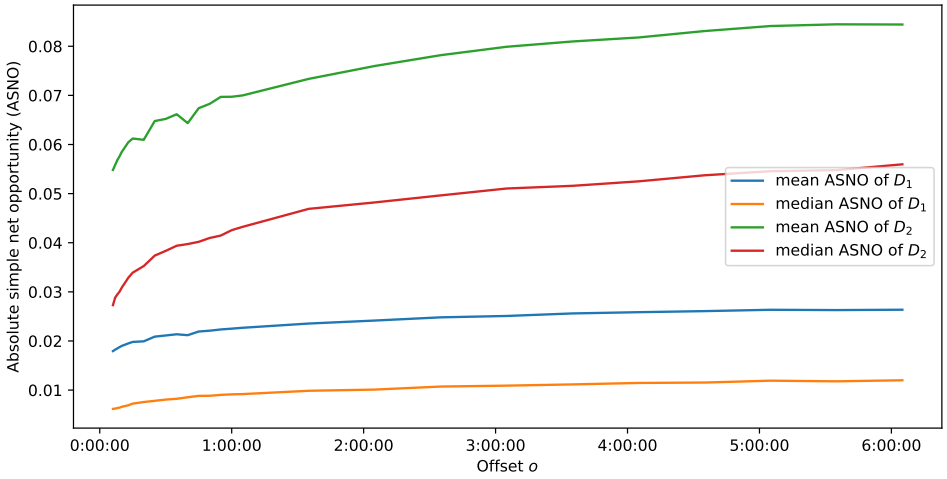


Figure 33: Median and mean absolute simple net opportunity (ASNO) on the datasets D_1 (all press releases) and D_2 (only press releases having ASNO of at least 3% after 5 h) for different offsets o

Next, we put the ASNO graphs of the press releases datasets into perspective by calculating the general ASNO that we would normally expect, when there was no press release published. We apply exactly the same procedure from which we calculated the general SNO histogram in Figure 29.

By plotting the general ASNO of companies in D_1 and D_2, we can visually compare them to the mean ASNO of D_1 and D_2. Both general ASNO graphs increase minimally in the beginning for one hour and then keep at a constant level. The general ASNO of companies that are listed in D_2 do not only vary more compared to the ones listed in D_1 but are also twice as high, such that they even exceed the mean ASNO of D_1 for certain offsets. This again clearly shows that if we discretized the ASNO with thresholds close to zero, then we would have to deal with lots of noise in our dataset.

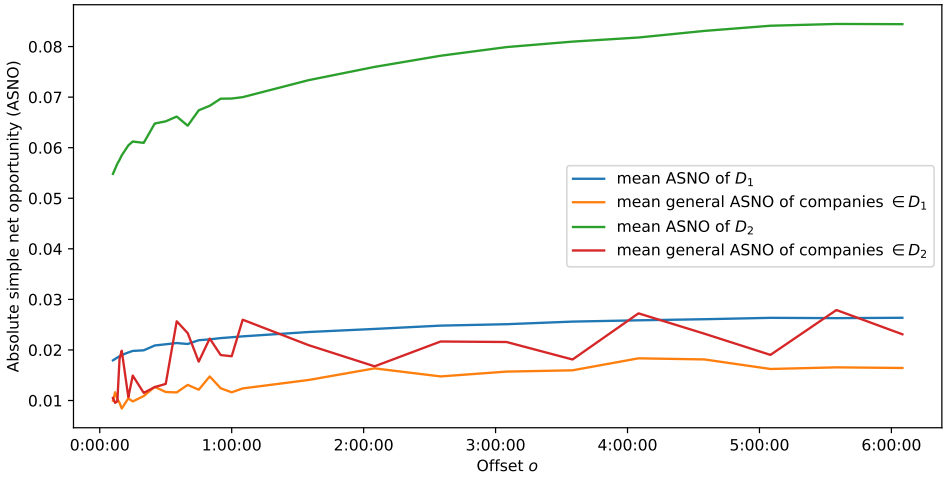


Figure 34: Comparison of mean absolute simple net opportunity (ASNO) on the datasets D_1 (all press releases) and D_2 (only press releases having ASNO of at least 3% after 5 h) for different offsets o to the general mean ASNO of companies in D_1 and D_2

The mean and median curves of Figure 33 and the general ASNO curves of Figure 34, show that offsetting window w_2 at five hours after exchange open is a valid choice as it maximizes the ASNO. Additionally, general ASNO keeps at the same level over time, as shown by Figure 34, which is why also the difference of the mean ASNO and mean general ASNO is maximized.

In Figure 29, we compared the SNO on D_1 to the general SNO on D_1, while w_2 was positioned at five hours after exchange opening in both cases. Our conclusion was that $\pm 8\%$ are reasonable choices as discretization thresholds for the labels *negative*, *neutral* and *positive*.

Having labeled all the press releases respectively, we now analyze the expected stock price curve for each class as follows:

1. Split the press release dataset by its labels
2. For each label:
    a) For each press release filter the respective stock price series in the interval six hours before exchange closed and six hours after exchange opened
    b) Normalize each stock price series by dividing it by its maximum stock price
    c) As the stock price dataset contains holes, as shown in Section 6.1.1, we fix these holes by linear interpolation.
    d) Average all stock price series to get one averaged curve.
    e) Calculate the median stock price curve.

The average stock prices curves for classes *positive* and *negative* have been plotted in Figure 35 and Figure 36, respectively. In all three cases the normalized stock prices stay almost at a constant level before the exchange closes. When the exchange opens again, the stock price already jumped, leaving us with roughly half of the theoretical ASNO we would have had if we had bought the stocks the day before. After the jump the information continues to be reflected onto the stock price for about five hours.

In Section 2.1.2, we introduced Electronic Communication Networks (ECNs), which enable traders to also trade during the after hours. As 50% of a press releases impact is already

represented in the stock price directly after the exchange opens, ECNs are worth considering to increase profits.

Moreover, the diagrams reveal that being fast pays off. If we miss the first hour after the exchange opened, we are only able to cash on a small portion of the profits that would have been possible.

So far our data resolution was one minute. Having discovered that press releases' impact is mapped onto the stock price until the exchanges closes, allows us to reduce the data resolution to the daily open, close, high and low stock price. These datasets come at a fraction of the price of the intra-day stock prices with a minute resolution.

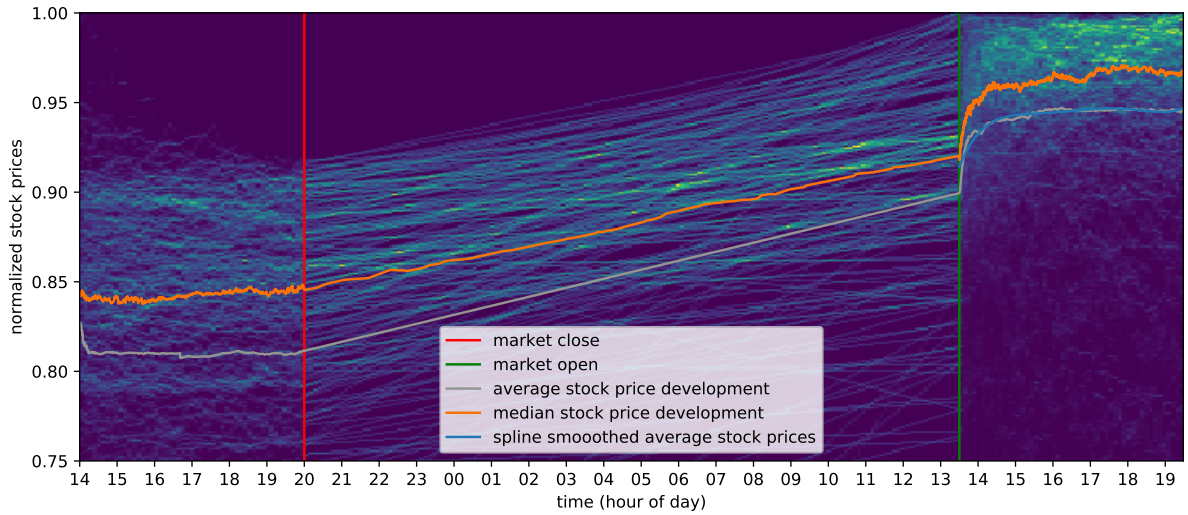


Figure 35: Stock price development heat map of press releases with SNO of at least 8% after an offset of 5 h after exchange opens

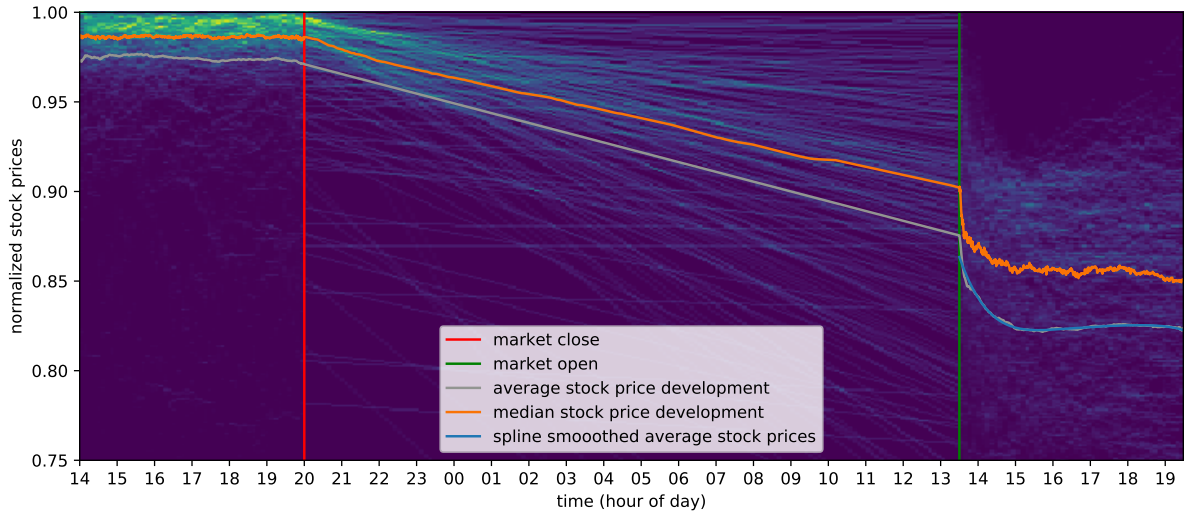


Figure 36: Stock price development heat map of press releases with SNO of less than -8% after an offset of 5 h after exchange opens

The stock prices curves for the neutral class vary within the interval $[-0.08, 0.08]$. For a final exploration, we plotted the average and median curves for the intervals $[0, 0.08]$ and $[-0.08, 0]$,

as shown in Figure 37 and Figure 38. Due to many press releases having almost no influence at all, these averaged curves show an SNO of ±1%.

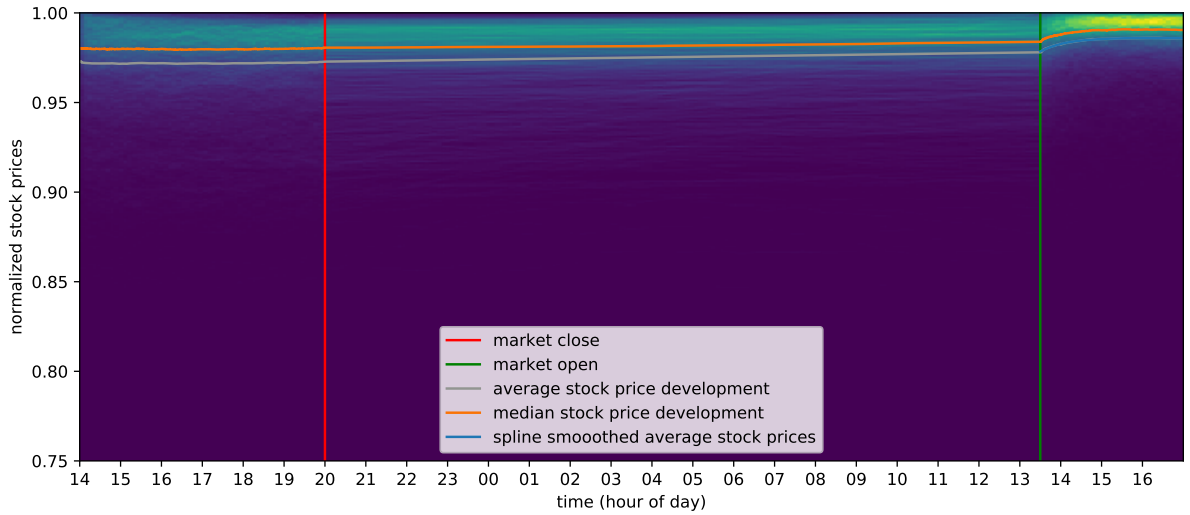


Figure 37: Stock price development heat map of press releases with SNO in $[0, 8\%]$ after an offset of 5 h after exchange opens

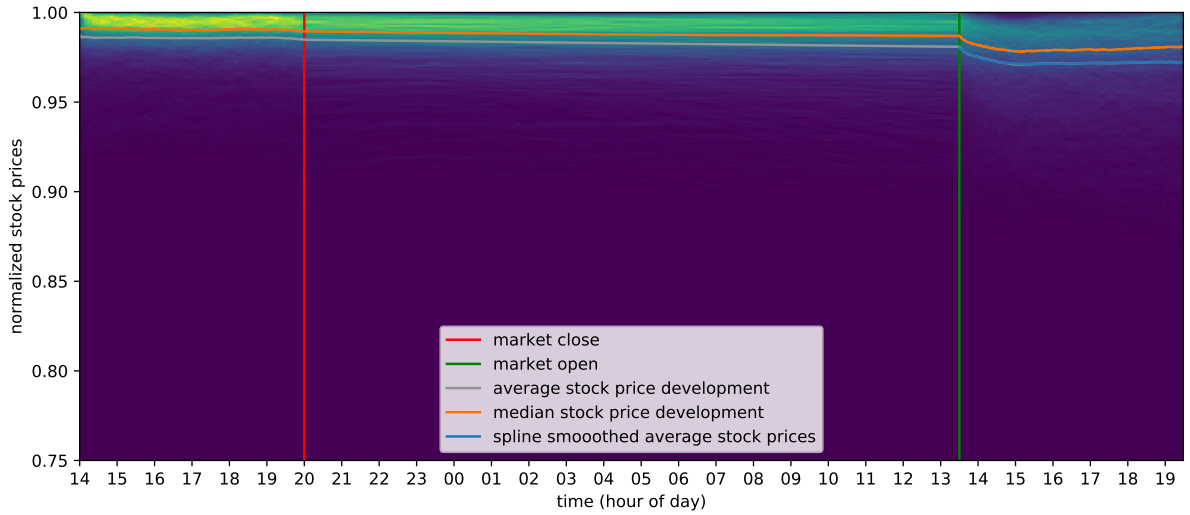


Figure 38: Stock price development heat map of press releases with SNO in $[-8\%, 0]$ after an offset of 5 h after exchange opens

After the press releases' SNO has been discretized by the thresholds ±0.08, the three resulting class (*positive*, *neutral*, *negative*) have the following distribution:

- positive: 267
- neutral: 8178
- negative: 298

Therefore, press releases with no impact are 14 times more likely than press releases with an impact. After having trained models on a balanced press releases set, the estimated accuracies will not represent those once deployed. This is why, we have to compensate for the overrepresentation of neutral press releases later in the evaluation as already stated in Section 4.4.

The low numbers of press releases with label positive and negative might also make the models learn noise and therefore not generalize well. For future work we will definitely consider increasing the number of labeled press releases. As of this thesis submission date, the press release crawlers are still mining press releases. However, since Google discontinued their stock price API, as stated in Section 4.1, we will have to buy daily stock prices from a financial data provider.

6.2 Performance Comparison of Impact Models

Previously, we have shown in Figure 34 that the offset placed at 5 hours after exchange open is a reasonable choice for calculating the SNO of the press releases to which we will stick to.

We also showed in Figure 29, that the discretization threshold $(-0.08, 0.08)$ is a well-chosen trade-off between the unwanted downsizing of the press release dataset and avoiding noise due to general stock price volatility in the impact class. However, this discretization threshold does not necessarily maximize the models performance, since a discretization threshold that has stronger constrains towards impact e.g., $(-0.15, 0.15)$ contains most likely no noise among all press releases labeled as having an impact. Since in general most press releases are distributed closely around 0 SNO, this means that the two sample sets *impact* and *no impact* are more different to each other than when discretized on e.g., $(-0.01, 0.01)$.

These hidden effect are difficult to consider, which is why we decided to train the three classifiers on press release sets with different discretization thresholds, as shown in Table 13. In total there are 8,743 press releases labeled by the respective SNO. Based on the discretization thresholds the press releases are assigned to one of the classes *impact* (I) or *no impact* ($\neg I$). A press release is labeled as I if its SNO is less than the left threshold or greater than the right threshold and is labeled as $\neg I$ otherwise.

Obviously, the wider the bin for neutral press releases gets, the smaller the number of samples having assigned class *impact* become. Notably, the data situation becomes very poor for the threshold $(-0.15, 0.15)$. When we increase the threshold width, then for both classes the quarter result rate rises. This means that quarter results generally have a higher impact than usual press releases.

Threshold	No impact ($\neg I$); QR rate	Impact (I); QR rate
(-0.01, 0.01)	3920; 0.14	4823; 0.29
(-0.02, 0.02)	5973; 0.15	2770; 0.38
(-0.03, 0.03)	6853; 0.16	1890; 0.45
(-0.04, 0.04)	7358; 0.18	1385; 0.49
(-0.05, 0.05)	7694; 0.18	1049; 0.52
(-0.08, 0.08)	8178; 0.20	565; 0.59
(-0.10, 0.10)	8369; 0.21	374; 0.63
(-0.15, 0.15)	8562; 0.22	181; 0.64

Table 13: Number of press releases and quarter result rates for each class (*impact*, *no impact*) depending on the discretization threshold

Each dataset is split, such that 80% of the press releases are used for grid search and the remaining 20% make up the holdout set. During grid search we perform a stratified 10 fold cross-validation for each parameter combination.[24] This means that we split the grid search dataset into 10 equally sized folds, each fold having the same number of samples per class

(stratification). In total we train and evaluate ten models for each parameter combination, while each time using a different bin as the validation set and the other bins as training. The models are trained on the training sets and evaluated on the validation sets. Thus, we get 10 accuracy measures per parameter combination, which we average. The averaged accuracy, which is also called validation score, gives an estimate on how well a model would perform if trained on the entire dataset.

For each dataset in Table 13 the hyperparameter combination yielding highest averaged accuracy on the validation sets is being determined by grid search. In order to compare the approaches to each other, we will finally train and evaluate each model with its best hyperparameters on the dataset discretized on thresholds $(-0.08, 0.08)$. In our previous analysis, we have shown that this dataset provides enough samples with class *impact* and at the same time is affected by too much noise.

6.2.1 Trivial Classifier

The trivial classifier's performance is evaluated separately on the press release sets introduced in Table 13. On each press release set we utilize grid search to find the best *feature_count* among the ones defined in Listing 10. This parameter determines how many words we consider per class as previously described in Section 4.3.2.

```
{
"feature_counts": [1, 5, 10, 50, 100, 150, 200, 250, 300, 350, 400, 450,
    500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000, 1500, 2000, 2500,
     3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000]
}
```

Listing 10: Best trivial classifier hyperparameters to test for during grid search

After having trained all the models, we determine for each press release set the *feature_count* which provided the highest average validation score, as shown in Table 14. We see that the optimal *feature_count* is at about 500 words per class and that the accuracy on the training set as well as on the validation set grows with the distance of the two discretization thresholds.

thresholds	feature_count	train score	val score
(-0.01, 0.01)	550	0.61	0.61
(-0.02, 0.02)	600	0.67	0.66
(-0.03, 0.03)	200	0.69	0.69
(-0.04, 0.04)	650	0.73	0.72
(-0.05, 0.05)	550	0.75	0.73
(-0.08, 0.08)	450	0.78	0.75
(-0.10, 0.10)	550	0.80	0.74
(-0.15, 0.15)	450	0.86	0.78

Table 14: Best trivial classifier for each dataset found by grid search

The averaged accuracies on the training set and the validation set for each dataset have been plotted as lines in Figure 39. The graphs show that for closer discretization thresholds, the accuracies are almost equal, while for wider discretization thresholds the accuracy on the training set exceeds the validation set accuracy.

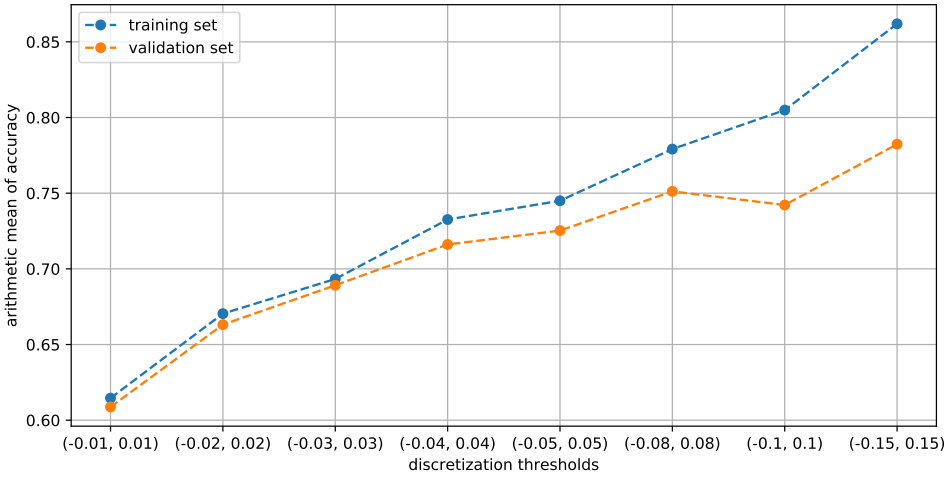


Figure 39: Best train and validation score of trivial classifier for different discretization thresholds

Our final model was trained on the dataset discretized by $(-0.08, 0.08)$ and parameterized by a feature count of 450.

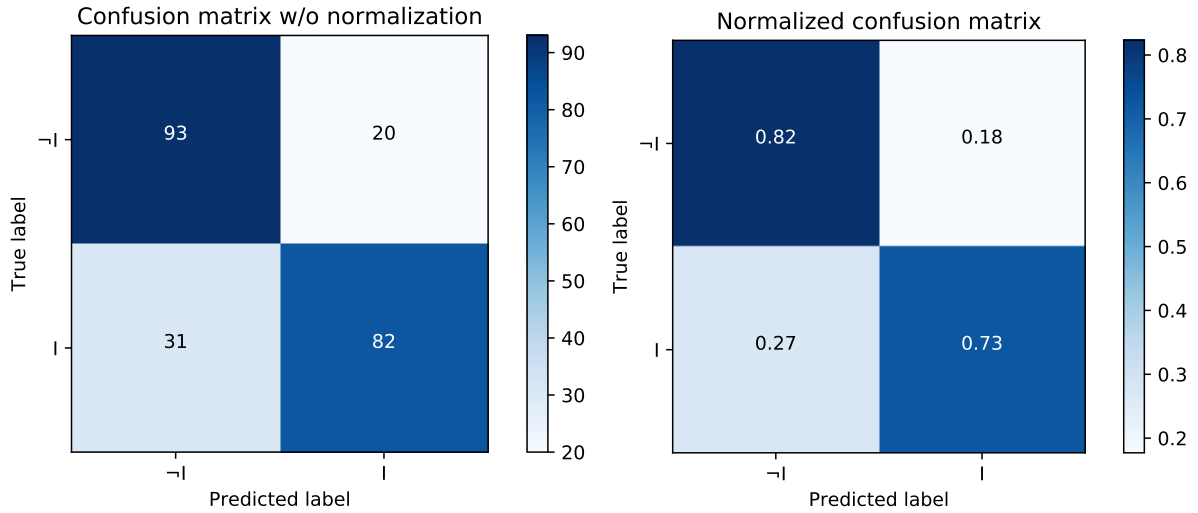
After having trained the model on the union of the training and the validation set, we evaluate the model on the test set which has never been used during grid search, making thereby sure that there are no information leaks. The resulting model has an accuracy of 77% on the test set.

Figure 40 shows the evaluations results in form of two confusion matrices. The first one contains the absolute counts in which bin the press releases ended up and the second confusion matrix is the row-wise normalized version of the prior one. The result is a conditional probability table, where the true label is given as evidence.

The model reaches for both classes very good results given the simplicity of the model. However, when estimating the performance in the field with a major class imbalance as shown in Table 13, then these good results are put into perspective.

We calculate the performance that we can expect in the field using Equation (116) and Equation (117). The probability of a press release having an impact given that it was predicted to have an impact $p(I|\hat{y})$ amounts to only 22%, while the probability of a press release having no impact given that it was predicted to have one $p(\neg I|\hat{y})$ amounts to 78%. So in practice among all press releases that have been predicted to have an impact approx. only one fifth do really have one.

We also manually checked the press releases for reoccurring patterns in each bin of the confusion matrix. Interestingly, we found out that most press releases in the TP bin are about quarter results, which rises the question how the quarter results are generally distributed over the four bins. We determined the rate of press releases having the key words *quarter* and *result* in their title for each bin in the confusion matrix. The results of our investigation are plotted in Figure 41. The quarter result rate for the TPs is 74%, while the quarter result rate for the TNs is only 11%. Among all press releases having impact as the true label, 29% are quarter results, compared to only 11% quarter results among all press releases having no impact. This means that even for a simple model like this one, this edge can be learned.



(a) Confusion matrix with absolute counts

(b) Confusion matrix normalized by true label counts

Figure 40: Confusion matrix of best trivial classifier trained and evaluated on press release dataset discretized by $[-0.08, 0.08]$

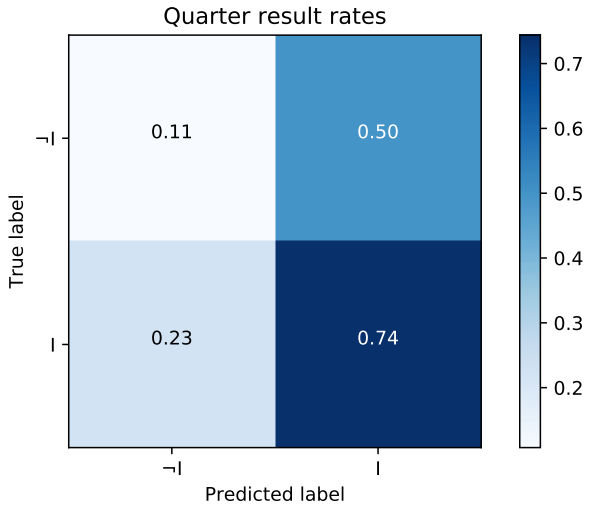


Figure 41: Quarter result rates of each bin of the trivial classifier's confusion matrix

Looking at the 50 most used terms in each class (*impact*, *no impact*), we see that both basically deal with quarter results. This means that even though press releases make up only 29% and 11% of the press releases with impact and without impact, that the quarter results use a similar vocabulary, while the other press releases are not that alike.

The intersection of the two lists contains 39 words, which as a consequence means that the most used words are not suitable for classification. In fact, this is also the reason why all models picked on average 500 words per class for model training and not much less, as previously shown in Table 14.

```
million | company | quarter | result | financial | net | statement | first | include |
income | increase | revenue | measure | year | sale | share | adjust | business |
compare | may | expense | per | fiscal | operate | cash | information | product | cost |
 loss | performance | period | tax | new | operation | report | end | risk | call | use
| expect | investor | release | provide | market | growth | future | management | second
```

```
| impact | continue
```

Listing 11: Top 50 words of class *impact* according to trivial classifier's vocabulary ranking

```
company | million | quarter | statement | include | result | financial | business | may
| first | information | share | increase | new | net | service | income | security |
market | product | year | investor | risk | provide | report | measure | management |
adjust | release | time | per | sale | investment | cash | call | contact | use |
customer | solution | performance | end | operate | expense | available | technology |
revenue | compare | total | future | march
```

Listing 12: Top 50 words of class *no impact* according to trivial classifier's vocabulary ranking

6.2.2 Naïve Bayesian Network

Grid search is performed on the naïve Bayesian network pipeline using the parameter combinations defined in Listing 13. There are two different scoring functions for feature selection, namely the χ^2 test statistic and mutual information, which were introduced in Section 2.6. The parameter k determines how many features are used for the model and the parameter α is the Laplace smoothing factor introduced Equation (46) of Section 2.3.4.

```
{
"feature_selection__score_func": [chi2, mutual_info],
"feature_selection__k": [10, 50, 100, 200, 500, 1000, 2000, 3000, 4000,
    5000, 6000, 7000, 8000, 9000, 10000, "all"],
"naive_bayes__alpha": [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
}
```

Listing 13: Naïve Bayesian network pipeline hyperparameters to test for during grid search

After having performed the grid search for each dataset, we determined for each dataset the parameter combination that yielded the highest validation score. We see that for all datasets the best model favored rather many features. For the other parameters (naive_bayes_alpha, feature_selection_score), we cannot see any preferences.

thresholds	val score	train score	nb_α	fs_k	fs_score_func
(-0.01, 0.01)	0.62	0.64	0.8	10000	mutual
(-0.02, 0.02)	0.66	0.69	0.4	10000	mutual
(-0.03, 0.03)	0.70	0.75	0.1	all	chi2
(-0.04, 0.04)	0.72	0.75	0.3	7000	mutual
(-0.05, 0.05)	0.73	0.82	0.1	6000	chi2
(-0.08, 0.08)	0.75	0.86	0.3	7000	chi2
(-0.10, 0.10)	0.76	0.88	0.4	all	chi2
(-0.15, 0.15)	0.79	0.95	0.1	6000	chi2

Table 15: Best hyperparameters of naïve Bayesian network pipeline found by grid search for each dataset

The validation score and the train score, which as a reminder are the average accuracies for the validation set and the training set, show a strong correlation with the distance of the two discretization thresholds, as shown in Figure 42. While the accuracies on both sets monotonically increase with the width of the threshold pairs, the accuracy of the training set always exceeds the one of the validation set. Even though the training set almost reaches 95% accuracy the validation sets accuracy still increases, which means that we are not overfitting, yet.

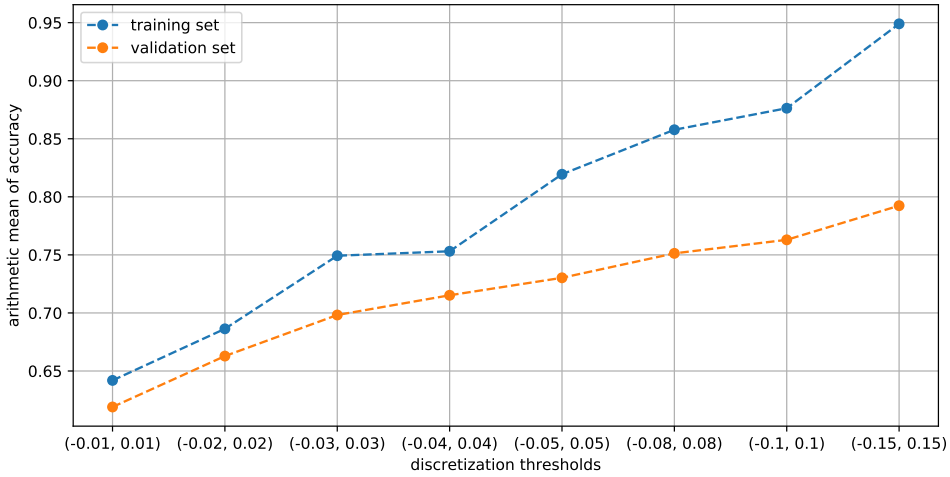
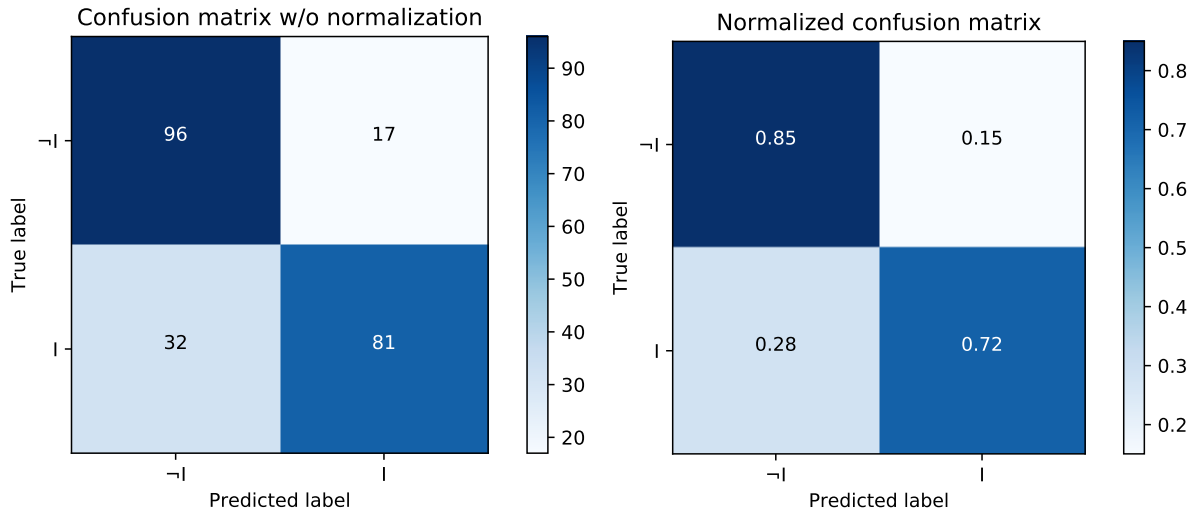


Figure 42: Best train and validation score of naïve Bayesian network pipeline on each dataset

As for the trivial classifier, we trained the algorithm on the press release set discretized by $[-0.08, 0.08]$, while setting the hyperparameters to *naive_bayes_alpha* $= 0.3$, *feature_selection_k* $= 7{,}000$ and *feature_selection_score_function* $=$ chi2. The accuracy of this model on the hold-out set is 78% according to Equation (115). Plotting the confusion matrix of the predictions shows that 85% of the no impact press releases were correctly predicted, while press releases with impact were predicted only in 72% of the cases correctly.



(a) Confusion matrix with absolute counts

(b) Confusion matrix normalized by true label counts

Figure 43: Confusion matrix of best naïve Bayesian network pipeline trained and evaluated on press release dataset discretized by $[-0.08, 0.08]$

Next, we calculate the performance that we can expect in the field using Equation (116) and Equation (117). The probability of a press release having an impact given that it was predicted to have an impact $p(I|\hat{y})$ amounts to only 25%, while the probability of a press release having no impact given that it was predicted to have one $p(\neg I|\hat{y})$ amounts to 65%. So in practice, among all press releases that have been predicted to have an impact, we expect only one quarter to really have an impact.

We also manually checked the press releases for reoccurring patterns in each bin of the confusion matrix. Interestingly, we found out that most press releases in the TP bin, were about quarter results, which rises the question if we learned a dummy model. The model might be only looking for the two key words *quarter* and *result* and then classify each press release based on the presence of the two words. This is why we determined the rate of press releases having the key words *quarter* and *result* in their title for each bin of the confusion matrix separately. The results of our investigation are plotted in the Figure 44. The quarter result rate for the TPs is 78%, while the one for the TNs is only 9%. Our first conclusion is that the model does not apply such a dummy model, due to the 9% of quarter results among the TNs. However, the 65% quarter result rate in FP shows that the model indeed learns that quarter results usually have a great impact, which is not per se false but also not a good rule on its own.

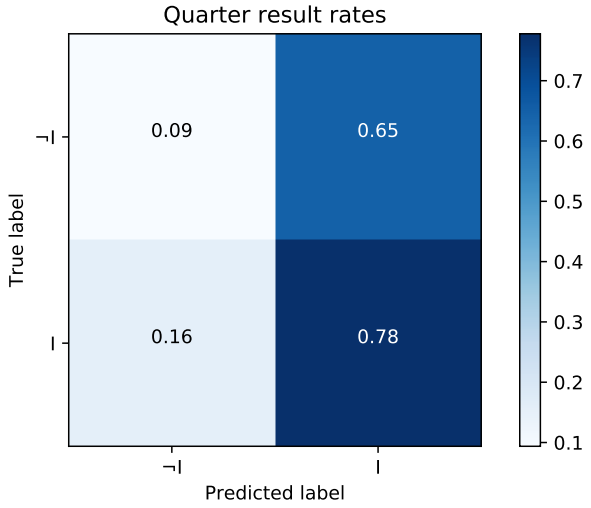


Figure 44: Quarter result rates in each bin of the naïve Bayesian network pipeline's confusion matrix

We now look at the top 200 features according to the χ^2 test statistic, as shown in Listing 14. The selected features clearly deal with quarter results. The overrepresentation of quarter results having class *impact*, as already shown in Table 13, therefore also reflects in the top features for discrimination.

```
million:3594 | quarter:2643 | net:1928 | revenue:1800 | fiscal:1607 | result:1564 |
measure:1489 | income:1416 | company:1385 | sale:1366 | financial:1321 | compare:1315 |
first:1282 | year:1227 | adjust:1192 | increase:1122 | expense:1038 | statement:952 |
per:824 | operate:812 | tax:792 | cost:768 | loss:767 | margin:767 | share:761 | include
:758 | period:748 | cash:706 | dilute:705 | gross:596 | operation:596 | prior:577 |
performance:576 | expect:575 | impact:526 | operating:523 | exclude:515 | may:493 | end
:484 | comparable:446 | earnings:439 | due:436 | growth:415 | product:414 | use:411 |
second:408 | continue:394 | call:391 | rate:380 | business:379 | profit:377 | fourth:364
 | future:364 | believe:362 | relate:354 | segment:350 | compensation:350 | report:345 |
 de:343 | basis:337 | change:335 | ability:335 | item:327 | guidance:322 | approximately
:309 | flow:307 | factor:301 | adjusted:298 | reflect:293 | release:286 | reconciliation
:285 | primarily:284 | risk:281 | expectation:277 | decrease:276 | amortization:273 |
charge:272 | outlook:254 | percent:251 | customer:241 | certain:241 | conference:240 |
plan:227 | month:219 | management:217 | investor:213 | march:213 | information:211 |
benefit:208 | new:206 | provide:204 | press:202 | evaluate:196 | estimate:195 | related
:192 | currency:189 | range:185 | accordance:182 | consider:180 | intangible:179 |
exchange:178 | total:177 | lower:176 | last:175 | adjustment:173 | uncertainty:172 |
base:172 | market:169 | effect:169 | activity:169 | present:166 | condition:162 | three
:161 | foreign:160 | associate:159 | strong:157 | higher:156 | reduction:154 | price:154
 | accounting:153 | attributable:152 | third:151 | drive:150 | could:149 | table:148 |
acquisition:148 | offset:148 | useful:148 | anticipate:145 | mix:144 | point:143 |
differ:142 | regard:141 | act:140 | depreciation:139 | ongoing:139 | actual:137 |
comparison:137 | account:136 | define:136 | initiative:135 | restructuring:135 | brand
:134 | make:134 | also:132 | contain:131 | decline:131 | potential:128 | calculate:128 |
```

```
 limitation:127 | store:127 | diluted:127 | amount:125 | update:125 | transaction:124 |
cause:124 | form:123 | additional:122 | trend:122 | well:120 | expenditure:119 | current
:115 | impairment:115 | inventory:114 | addition:113 | consolidated:113 | improvement
:113 | time:112 | security:111 | december:111 | rite:111 | standard:110 | event:109 |
favorable:109 | capital:108 | versus:108 | materially:108 | international:107 | en:106 |
 interest:104 | debt:103 | commission:102 | equivalent:102 | order:101 | subscription
:101 | section:101 | number:100 | obligation:100 | la:100 | percentage:99 | achieve:99 |
 aid:98 | historical:97 | filing:97 | sec:96 | effective:96 | represent:95 | operational
:94 | available:93 | identify:90
```

Listing 14: Top 200 words for the impact models according to χ^2 test statistic score

The top 200 words according to the mutual information score function are shown in Listing 15. Again, the top words deal with quarter results. The intersection of both lists contains 165 words, which shows that both methods perform similarly.

```
result:0.16 | quarter:0.16 | statement:0.15 | million:0.15 | net:0.14 | compare:0.13 |
risk:0.13 | uncertainty:0.13 | company:0.13 | include:0.12 | factor:0.12 | measure:0.12
| expect:0.12 | increase:0.12 | expense:0.12 | end:0.11 | sale:0.11 | believe:0.11 |
release:0.11 | cash:0.11 | share:0.11 | differ:0.11 | financial:0.11 | actual:0.11 |
materially:0.11 | revenue:0.11 | margin:0.11 | period:0.11 | income:0.11 | may:0.11 |
year:0.1 | future:0.1 | per:0.1 | continue:0.1 | report:0.1 | cost:0.1 | expectation:0.1
 | could:0.1 | update:0.1 | operating:0.1 | tax:0.1 | dilute:0.1 | performance:0.1 |
relate:0.1 | first:0.1 | adjust:0.1 | gross:0.09 | impact:0.09 | operate:0.09 | growth
:0.09 | exchange:0.09 | comparable:0.09 | security:0.09 | due:0.09 | certain:0.09 |
reconciliation:0.09 | reflect:0.09 | contain:0.09 | cause:0.09 | obligation:0.09 |
operation:0.08 | exclude:0.08 | loss:0.08 | change:0.08 | call:0.08 | form:0.08 | use
:0.08 | item:0.08 | prior:0.08 | outlook:0.07 | press:0.07 | commission:0.07 | plan:0.07
 | table:0.07 | ability:0.07 | current:0.07 | rate:0.07 | strong:0.07 | business:0.07 |
accordance:0.07 | consider:0.07 | estimate:0.07 | evaluate:0.07 | act:0.07 | fiscal:0.07
 | total:0.07 | compensation:0.07 | amortization:0.07 | conference:0.07 | base:0.07 |
earnings:0.06 | basis:0.06 | related:0.06 | march:0.06 | condition:0.06 | anticipate
:0.06 | capital:0.06 | highlight:0.06 | december:0.06 | primarily:0.06 | litigation:0.06
 | investor:0.06 | adjusted:0.06 | filing:0.06 | flow:0.06 | date:0.06 | information
:0.06 | historical:0.06 | reform:0.06 | effect:0.06 | second:0.06 | well:0.06 | currency
:0.06 | amount:0.06 | profit:0.06 | approximately:0.06 | product:0.05 | guidance:0.05 |
event:0.05 | today:0.05 | trend:0.05 | also:0.05 | decrease:0.05 | benefit:0.05 |
additional:0.05 | useful:0.05 | limitation:0.05 | interest:0.05 | law:0.05 | substitute
:0.05 | drive:0.05 | intend:0.05 | market:0.05 | new:0.05 | calculate:0.05 | accounting
:0.05 | make:0.05 | involve:0.05 | depreciation:0.05 | regard:0.05 | present:0.05 |
percentage:0.05 | mean:0.05 | project:0.05 | management:0.05 | equivalent:0.05 |
potential:0.05 | annual:0.05 | acquisition:0.05 | comparison:0.05 | associate:0.05 |
activity:0.05 | charge:0.05 | segment:0.05 | development:0.05 | time:0.05 | file:0.04 |
addition:0.04 | price:0.04 | supplemental:0.04 | available:0.04 | reduction:0.04 |
subject:0.04 | adjustment:0.04 | month:0.04 | section:0.04 | offset:0.04 | range:0.04 |
identify:0.04 | similar:0.04 | sec:0.04 | successfully:0.04 | within:0.04 | provide:0.04
 | quarterly:0.04 | affect:0.04 | require:0.04 | provision:0.04 | expand:0.04 | ongoing
:0.04 | initiative:0.04 | improvement:0.04 | mix:0.04 | revise:0.04 | debt:0.04 |
fluctuation:0.04 | average:0.04 | significant:0.04 | assumption:0.04 | exist:0.04 |
percent:0.04 | balance:0.04 | attributable:0.04 | private:0.04 | intangible:0.04 | live
:0.04 | limited:0.04 | higher:0.04 | inventory:0.04 | economic:0.04
```

Listing 15: Top words 200 for the impact models according to mutual information score

From the naïve Bayesian network it is possible to generate two word lists, which contain the words whose presence is most important for each of the classes impact I and no impact $\neg I$. Since naïve Bayesian networks treat all terms as random variables, we can formally define the i^{th} term in the vocabulary as T_i, where the value t_i denotes presence and $\neg t_i$ denotes absence of term i. The CPT of term i contains $p(t_i|I)$ and $p(t_i|\neg I)$, which express the probability of term i appearing in one of the respective classes. The two probabilities cannot be used directly to produce the words list, as they do not add up to one. Thus, a term having $p(t_1|I) = p(t_1|\neg I) = 1$ would be picked for both lists over $p(t_2|I) = 0.9$ and $p(t_1|\neg I) = 0.01$, even though t_2 would definitely be a better fit for the word list of class I.

This is why, we need another approach. We calculate the probability of each class given term t_i instead. These probabilities $p(I|t_i)$ and $p(\neg I|t_i)$ sum up to one, which means that we can pick for each class the words that provide highest conditional probability. The probabilities are derived from the probabilities in the CPTs as follows:

$$p(I|t_i) = \frac{p(I,t_i)}{p(t_i)} \tag{119}$$

$$= \frac{p(t_i|I)p(I)}{p(t_i)} \tag{120}$$

$$= \alpha p(t_i|I), \tag{121}$$

under the constraint that $\alpha(p(t_i|I) + p(t_i|\neg I))) = 1$. Since the dataset is balanced $p(I) = p(\neg I) = 0.5$ and since $p(t_i)$ is in the evidence part of $p(I|t_i)$, this means that both terms can be replaced by the constant α. This constant has to be determined such that $p(I|t_i)$ and $p(\neg I|t_i)$ sum up to one, which is enforced by the constraint. In conclusion, we can generate our word lists, by scaling the probabilities in the CPTs such that their sum adds up to one and pick for each class the terms whose presence have the highest probabilities for this class.

The resulting word lists for the the classes *impact* and *no impact* class are shown in Listing 16 and Listing 17, respectively. For the top words of class I, we see that these words deal mostly with words mentioned in quarter results: *fiscal*, *gross*, *profit*, *revenue*, *margin*. Therefore, words that deal with real numbers i.e., measurable facts, have the most influence on stock prices. The top word *rite* in word list of class I and the top words *en, de, la* of the other top words list show signs of overfitting. These words have been only present in one of the classes by chance and the model learned them falsely, due to the frequency approach. On a lager dataset their appearance probability for each class should be equal.

It is also noteworthy, that the word order found by the χ^2 test statistic and mutual information for feature reduction is completely different from the word order of the top 50 words for each class found by the naïve Bayesian network. On the one hand, this shows that the feature selection scoring functions do not work as good as expected. This is also why the best models on each dataset picked almost all features. On the other hand, we can already see, that the top words stem from overfitting, which makes feature selection strategies an important mean.

```
rite:0.99 | fiscal:0.79 | subscription:0.77 | aid:0.76 | mix:0.73 | gross:0.72 | profit
:0.72 | dilute:0.7 | margin:0.68 | comparable:0.67 | compensation:0.67 | fourth:0.67 |
revenue:0.66 | favorable:0.66 | intangible:0.66 | restructuring:0.64 | comparison:0.64 |
 adjusted:0.63 | expenditure:0.63 | compare:0.62 | versus:0.62 | diluted:0.61 |
reduction:0.61 | sale:0.61 | prior:0.6 | measure:0.6 | outlook:0.6 | inventory:0.6 | net
:0.6 | amortization:0.6 | limitation:0.59 | tax:0.59 | depreciation:0.59 | operating
:0.59 | reconciliation:0.59 | equivalent:0.59 | exclude:0.58 | expense:0.58 | initiative
:0.58 | useful:0.58 | impairment:0.58 | adjust:0.58 | percentage:0.58 | loss:0.58 |
period:0.57 | due:0.57 | evaluate:0.57 | accounting:0.57 | guidance:0.57 | income:0.57 |
 ongoing:0.57 | million:0.56 | charge:0.56 | year:0.56 | cost:0.56 | adjustment:0.56 |
foreign:0.56 | segment:0.56 | quarter:0.56 | improvement:0.55 | define:0.55 |
attributable:0.55 | item:0.55 | operate:0.55 | impact:0.55 | consolidated:0.55 | last
:0.55 | table:0.55 | point:0.54 | lower:0.54 | consider:0.54 | account:0.54 | decline
:0.54 | per:0.54 | trend:0.54 | expect:0.54 | primarily:0.54 | order:0.54 | calculate
:0.54 | accordance:0.53 | reflect:0.53 | relate:0.53 | first:0.53 | standard:0.53 |
expectation:0.53 | operation:0.53 | flow:0.53 | cash:0.53 | offset:0.53 | increase:0.53
| result:0.52 | range:0.52 | basis:0.52 | decrease:0.52 | earnings:0.51 | performance
:0.51 | strong:0.51 | third:0.51 | believe:0.51 | effect:0.51
```

Listing 16: The top 50 words for class I ranked by probability $p(I|t_i)$ out of 200 features

```
en:0.98 | de:0.96 | la:0.95 | security:0.64 | available:0.63 | time:0.63 | information
:0.62 | market:0.62 | new:0.62 | capital:0.61 | interest:0.6 | provide:0.6 | also:0.6 |
investor:0.6 | business:0.6 | make:0.6 | management:0.59 | event:0.59 | total:0.59 |
well:0.59 | risk:0.58 | current:0.58 | additional:0.58 | march:0.57 | acquisition:0.57 |
 customer:0.57 | store:0.57 | form:0.57 | release:0.57 | may:0.57 | materially:0.57 |
base:0.57 | debt:0.57 | number:0.57 | sec:0.56 | international:0.56 | company:0.56 |
report:0.56 | represent:0.56 | could:0.56 | contain:0.56 | exchange:0.56 | conference
:0.56 | update:0.56 | include:0.56 | uncertainty:0.56 | product:0.55 | addition:0.55 |
actual:0.55 | obligation:0.55 | act:0.55 | section:0.55 | plan:0.55 | commission:0.54 |
statement:0.54 | press:0.54 | identify:0.54 | condition:0.54 | brand:0.54 | cause:0.54 |
 percent:0.54 | transaction:0.54 | call:0.54 | potential:0.54 | regard:0.53 | differ
```

```
:0.53 | future:0.53 | effective:0.53 | use:0.53 | month:0.53 | amount:0.52 | factor:0.52
 | price:0.52 | drive:0.52 | share:0.52 | three:0.52 | higher:0.52 | approximately:0.52
| historical:0.52 | present:0.51 | activity:0.51 | filing:0.51 | change:0.51 | estimate
:0.51 | related:0.51 | growth:0.51 | associate:0.51 | continue:0.51 | second:0.51 |
december:0.51 | end:0.51 | certain:0.5 | currency:0.5 | benefit:0.5 | anticipate:0.5 |
operational:0.5 | achieve:0.5 | financial:0.5 | rate:0.49 | ability:0.49
```

Listing 17: The top 50 words for class $\neg I$ ranked by $p(\neg I|t_i)$ out of 200 features

6.2.3 Support Vector Machine

As for the naïve Bayesian network pipeline and the trivial classifier pipeline, we perform grid search on the support vector machine pipeline for each press release set defined in Table 13. The dictionary in Listing 18 defines all the hyperparameter combinations that we test for. The features selection parameters are the same as the ones used for the naïve Bayes classifier pipeline. This support vector machine always uses a polynomial kernel having degrees between 1 and 5. The hyperparameter C is a penalty factor multiplied by the slack for all constraints not being fulfilled, as explained at the end of Section 2.5. The higher the penalty factor the more we try to fit all data points including noise.

```
1 {
2 "feature_selection__score_func": [chi2, mutual_info],
3 "feature_selection__k": [10, 50, 100, 200, 500, 1000, 2000, 3000, 5000,
      8000, 10000, "all"],
4 "svm__kernel" : ["poly"],
5 "svm__degree": [1, 2, 3, 4, 5],
6 "svm__C": [0.1, 0.3, 0.5, 0.7, 0.9, 1]
7 }
```

Listing 18: Support vector machine pipeline hyperparameters to test for during grid search

Table 16 shows the best hyperparameterization for each press release set and the resulting averaged validation accuracy and averaged train accuracy. Interestingly, except for the last press release set, the best SVM had a linear kernel. The last press release set contains only few press releases but the model has the highest complexity with its quadratic kernel, which could be a hint for overfitting. And in fact, if we compare the accuracy of this model with the best model trained on the preceding press release sets, we see that the accuracy on the validation set decreased while the accuracy on the training set spiked. This is a strong indicator that our model was overfitting and should rather be trained on a bigger training set.

The number of features for this model leads to another interesting observation. It is at least half the size of the other ones. Being able to greatly reduce the feature space, while having a big slack penalty C in place, means that the data is rather easy to split. This effect is amplified the quadratic kernel and the small dataset size.

thresholds	val score	train score	fs_k	fs_score_func	svm_C	svm_degree	svm_kernel
(-0.01, 0.01)	0.62	0.66	5000	mutual	0.9	1	poly
(-0.02, 0.02)	0.67	0.72	2000	chi2	0.9	1	poly
(-0.03, 0.03)	0.71	0.75	1000	chi2	0.5	1	poly
(-0.04, 0.04)	0.71	0.76	1000	chi2	0.5	1	poly
(-0.05, 0.05)	0.74	0.80	1000	mutual	0.5	1	poly
(-0.08, 0.08)	0.77	0.82	1000	chi2	0.3	1	poly
(-0.10, 0.10)	0.77	0.82	5000	chi2	0.7	1	poly
(-0.15, 0.15)	0.76	0.93	500	mutual	0.9	2	poly

Table 16: Best hyperparameters of support vector machine pipeline for each dataset found by grid search

The effect of overfitting can also be recognized in Figure 45, which shows the averaged accuracy on the training set as well as on the validation set for the different datasets (denoted by the discretization thresholds). For the thresholds with largest width, the train accuracy escalates while the validation score drops, a strong indicator for overfitting.

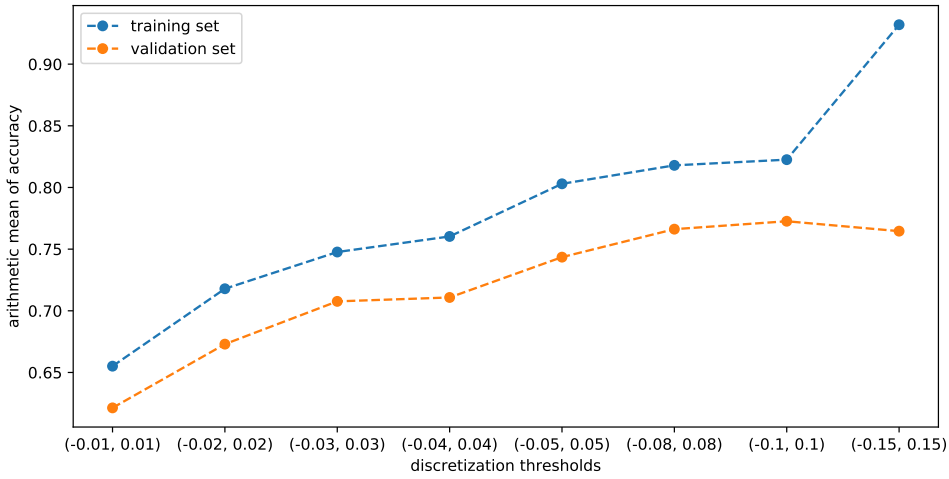


Figure 45: Best train and validation score of support vector machine pipeline on each dataset

As for the previous algorithms, we train the support vector machine pipeline on the dataset discretized by $(-0.08, 0.08)$ and parameterize it with the best hyperparameters we found for this dataset, as listed in Table 16. The accuracy of this model on the holdout set is 76%. The confusion matrices in Figure 46 show that the model performs slightly better on classifying press releases without influence. In general, the model works well when provided with a balanced dataset.

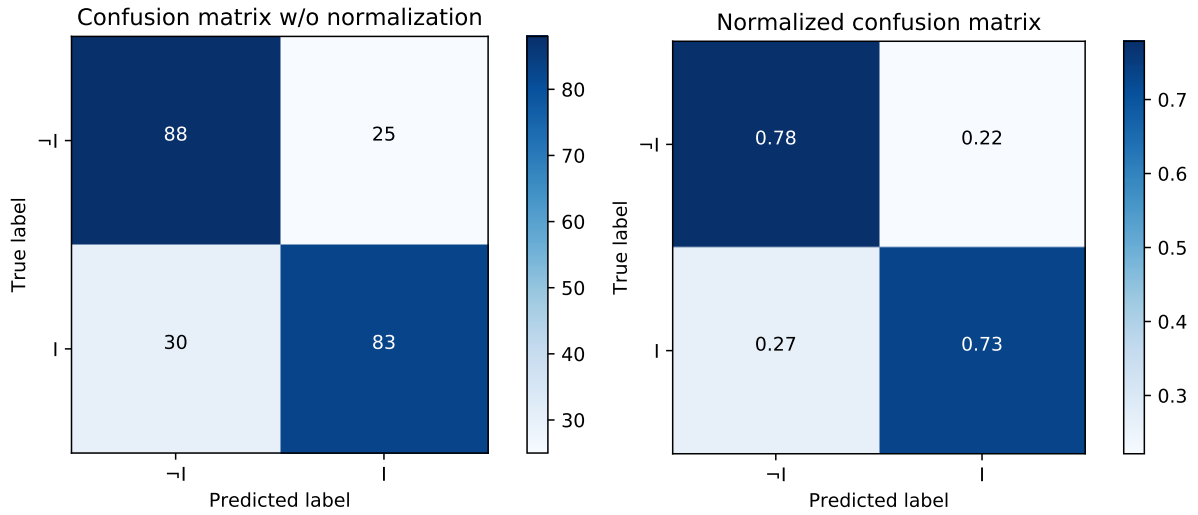
As for the other models, this quickly changes when deployed to the field, where press releases with an impact greater than 8% SNO make up only 6%. Then, the probability of a press release to actually have an impact given the prediction to have an impact $p(I|\hat{y})$ amounts to 0.19 as per Equation (116) and the probability $p(\neg I|\hat{y})$ amounts to 0.81. Thus, a press release predicted as having an impact is still $\frac{4}{5}$ times more likely to have no impact.

As for the previous algorithms, we also take a look at the quarter results rates for each bin in the confusion matrix. In Figure 47, we see that for the TPs the quarter result ratio is highest among all bins. At the same time many quarter results which truly have no impact have been falsely classified as having an impact, which can be deduced from the 40% quarter result rate in the FP bin. This means that similar to the other algorithms the SVM algorithm learned that quarter results usually have a high impact.

6.2.4 Discussion

In this section, we compare the three algorithms with regard to the feature selection process (number of features, scoring function) and to their performances in terms of accuracy. Throughout this discussion, we draw conclusions on the models, as well as, on the dataset and derive possible improvements.

For each of the three algorithms, grid search determined the best hyperparameterization on each of the eight stratified data sets. The trivial classifier used between 200 and 650 features per class on each dataset, while the naïve Bayesian network pipeline used at least 6000 features. The total



(a) Confusion matrix with absolute counts

(b) Confusion matrix normalized by true label counts

Figure 46: Confusion matrix of best support vector machine pipeline trained and evaluated on press release dataset discretized by $[-0.08, 0.08]$

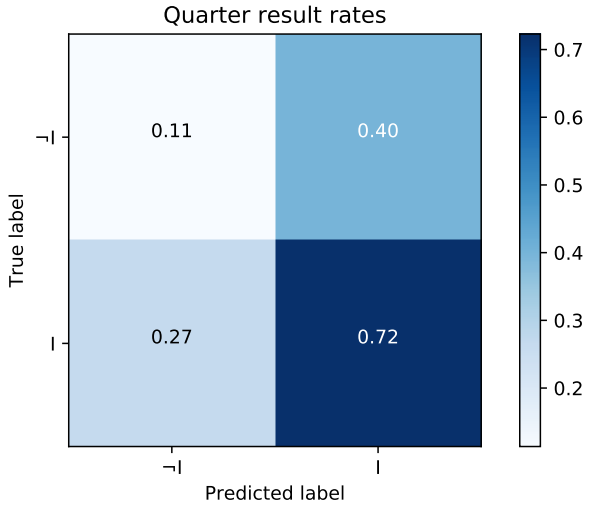


Figure 47: Quarter result rates in each bin of the support vector machine pipeline's confusion matrix

number of features working best for the SVM pipeline ranged between 500 and 5000 features. This makes the trivial classifier superior in this regard, even though the scoring function for the trivial classifier is rather simple. It just takes the k most used words for every class into account, thus not favoring words for discrimination. If we applied proper features selection strategies to the trivial classifier, we expect its accuracy to improve further while reducing its feature space.

The χ^2 test statistic and the mutual information function try to retrieve only words occurring either in press releases with impact or in press releases without impact. We have shown that the top 200 words found by these two scoring functions intersected 165 times, which means that in the end the choice between those does not have a major influence on the model performance. This is why, the best models for each of the two algorithms picked one of the scoring methods without any reoccurring pattern. The top 200 words found by the two features selection strategies dealt with quarter results, which is expected as for class *impact* 59% of the press releases were quarter results and for class *no impact* only 20% were quarter results.

After we intersected the top 50 words found by χ^2 test statistic, the top 50 words found by mutual information and the two top 50 words lists determined by the trivial scoring function, the intersection still had a size of 32. In conclusion, the most used words in each class had discrimination abilities, even though they appeared as most used words in both classes.

For a final evaluation, the three algorithms were trained on the stratified press release set discretized by thresholds $(-0.08, 0.08)$. The accuracies for the trivial model, naïve Bayes model and the SVM model were 77%, 78% and 76%, respectively. Unexpectedly, the trivial model was as good as the more sophisticated ones. Even though we disproved that the two sophisticated models learned plain word counts like the trivial model, it is clear that the two models have a need for improvement. All three models failed many times on the prediction of quarter results which had truly no impact. The count vectorization in each pipeline might not have been the best way of representing an article, as the structure of the text gets lost. One simple improvement would be to use n-gram models at the expense that the feature space might become intractable for larger n.

In the field, all models showed performances, that were a lot worse than on the balanced dataset. Press releases with impact only appear in about 6% of all press releases. This prior probability had a huge influence for classification. For $p(I) \ll p(\neg I)$, the probability $p(I|\hat{y})$ might become less than $p(\neg I|\hat{y})$, such that even though our model predicted a press release to have an impact, it is still likelier that it did not have one. Hence, for our models we estimated that it is four to five times likelier to have a false positive (press release truly having no impact but predicted to have an impact) than having a true positive (press release truly having an impact and predicted to have an impact).

As a final conclusion, it is generally possible for all models to predict the impact of press releases. Based on its low feature space, competitive accuracies and good interpretability, the trivial classifier outperformed the more sophisticated approaches and most likely can even be improved further. Nevertheless, in the field, where press releases make up only 6%, impact prediction becomes unreliable and at the very most might be used as another indicator.

6.3 Performance Comparison of Sentiment Models

After having evaluated the impact models, we now do the same for the sentiment models. The models are based on the same algorithms as the impact models and are also evaluated on the same hyperparameter grid. However, the models are evaluated on a different press release set. We took the previous press release set, discarded the press releases with no impact and assigned

the remaining press releases a negative or positive sentiment. The assignment is based on two discretization thresholds. The lower one determines the maximal boundary up to which press releases are considered as negative in terms of SNO and the upper one takes all press releases with a SNO greater than this threshold as positives.

As for the impact model, the models are evaluated on multiple press release sets each time discretized by a different threshold, as shown in Table 17. We see that for thresholds with a wider spread the number of press releases in each class greatly reduce making it difficult to build reliable models.

Threshold	Neg. sent. ($\neg S$); QR rate	Pos. sent. (S); QR rate
(-0.01, 0.01)	2189; 0.32	2634; 0.28
(-0.02, 0.02)	1259; 0.40	1511; 0.35
(-0.03, 0.03)	881; 0.47	1009; 0.43
(-0.04, 0.04)	650; 0.52	735; 0.47
(-0.05, 0.05)	495; 0.55	554; 0.50
(-0.08, 0.08)	267; 0.60	298; 0.57
(-0.10, 0.10)	171; 0.65	203; 0.61
(-0.15, 0.15)	81; 0.64	100; 0.63

Table 17: Number of press releases and quarter result rates for each class (negative sentiment $\neg S$, positive sentiment S) depending on the discretization threshold

For comparison we also train and evaluate every model on the dataset discretized by $(-0.08, 0.08)$ using its best hyperparameters, as done before for the impact models.

6.3.1 Trivial Classifier

For each press release set, the trivial classifier performed grid search with the same hyperparameters as defined in Listing 10. The hyperparameters of the best performing models for each press release set are shown in Table 18. For datasets with a small threshold width the accuracy on the training set as well as on the validation set is even worse than random guessing. With bigger threshold spreads the accuracies rise to up to 97% on the training set and 58% on the validation set. The feature counts vary between 150 and 2000 while the median is at around 400.

thresholds	feature_count	train score	val score
(-0.01, 0.01)	550	0.44	0.41
(-0.02, 0.02)	700	0.50	0.45
(-0.03, 0.03)	200	0.51	0.48
(-0.04, 0.04)	700	0.56	0.48
(-0.05, 0.05)	350	0.54	0.51
(-0.08, 0.08)	150	0.60	0.55
(-0.10, 0.10)	250	0.68	0.57
(-0.15, 0.15)	2000	0.97	0.58

Table 18: Best trivial classifier for each dataset after applying grid search

Figure 48 shows the score on the training set and validation set for the different thresholds. At a threshold of $(-0.05, 0.05)$ the accuracy on the validation set surpasses random guessing

and peaks at $(-0.15, 0.15)$ reaching a maximum of 58%. In contrast to the accuracy of the validation, which increases rather linearly, the accuracy on the training set spikes for thresholds (-0.1, 0.1) and (-0.15, 0.15) overproportionally.

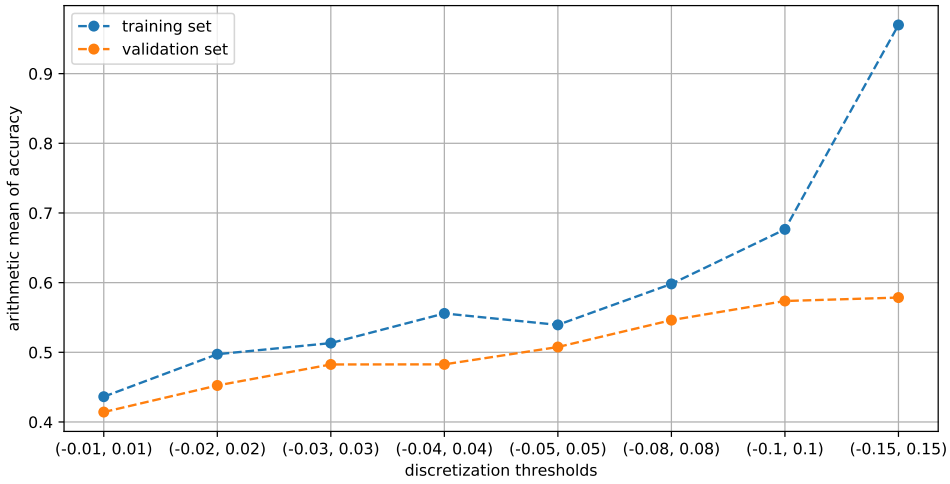
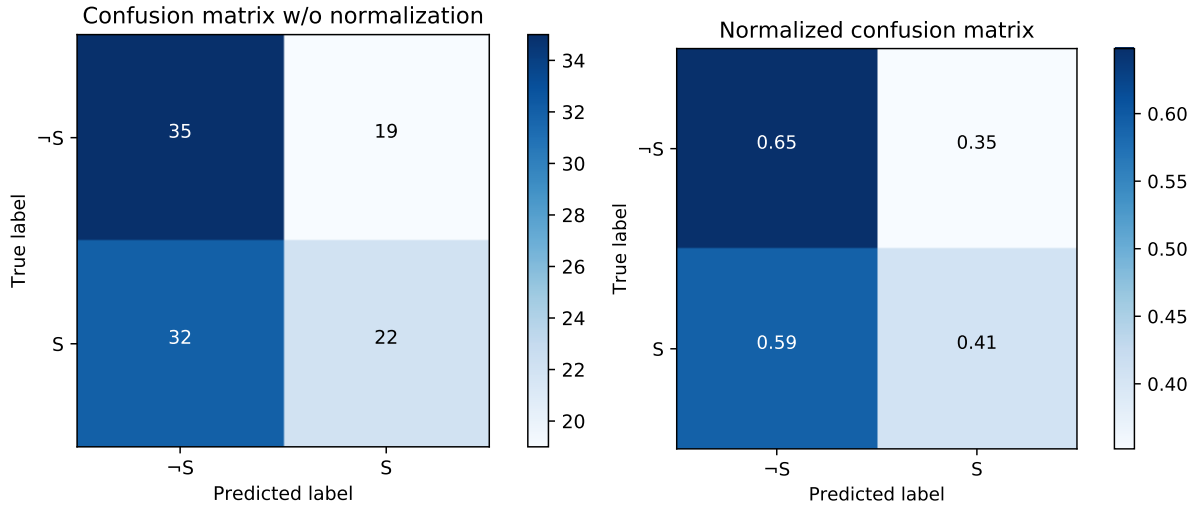


Figure 48: Best train and test score of trivial classifier for different datasets

We trained and evaluated the final model on the press release set discretized by (-0.08, 0.08) using the best hyperparameters for this dataset according to Table 18. The accuracy of this model is 53%, while the predictions are distributed as shown in Figure 49. The prediction of press releases with negative sentiment yields high accuracies, whereas for press releases with positive sentiment we see rather poor results.



(a) Confusion matrix with absolute counts

(b) Confusion matrix normalized by true label counts

Figure 49: Confusion matrix of best trivial classifier trained and evaluated on press release dataset discretized by $[-0.08, 0.08]$

Taking into account that the quarter result rates are almost balanced at approx. 60% for both classes, as shown in Table 17, we see in Figure 50 that the quarter result in the FP bin to be peculiar high. This means that the model learned that if there is a quarter result it is likely to be of positive sentiment.

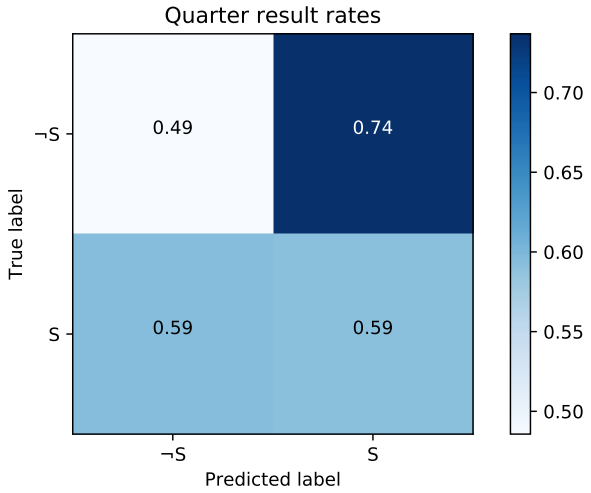


Figure 50: Quarter result rates in each confusion matrix bin of best trivial classifier

As the quarter result rates are at around 60% in both classes, it is not surprising, that the top words in both classes all stem from the quarter result domain, as shown by Listing 19 and Listing 20.

```
million | company | quarter | result | statement | financial | net | first | include |
income | increase | measure | revenue | adjust | year | sale | business | may | share |
expense | compare | operate | cash | per | product | information | end | period | cost |
 performance | fiscal | new | expect | report | call | loss | operation | use | risk |
investor | provide | tax | growth | release | future | market | management | earnings |
security | continue
```

Listing 19: Top 50 words of class sentiment (S) according to trivial classifier's ranking

```
million | company | quarter | result | net | statement | financial | include | first |
income | year | increase | revenue | measure | share | sale | adjust | compare |
business | may | expense | fiscal | per | operate | cash | cost | product | information
| tax | report | period | loss | risk | performance | operation | end | new | expect |
call | market | release | impact | use | investor | security | future | margin | growth
| management | provide
```

Listing 20: Top 50 words of class no sentiment ($\neg S$) according to trivial classifier's ranking

6.3.2 Naïve Bayesian Network

The naïve Bayesian network pipeline, which we performed grid search on using the same hyper-parameters as for the impact pipeline, yielded the results shown in Table 19. Interestingly, this time the algorithm always favored the χ^2 test statistic as the better scoring function and also the number of features have been halved compared to the corresponding impact model.

As shown in Figure 51, the accuracies on the different training and test sets again rise with the width of the two thresholds. An accuracy on the training set of close to 100% means that it is very well possible to learn the training set, however the applicability of this knowledge to unseen data is limited as shown by the accuracies on the test sets (low generalization performance).

Having again trained the model on the press release set discretized on the thresholds (-0.08, 0.08) and plotted its performance on the test set in form the confusion matrix as shown in Figure 52, we see that the distributions are exactly the same as for the trivial model. However, this is only up to chance as there is no complete overlap but many correspondences which we checked manually. As the trivial model, this model has an overall accuracy of 53% on the test set.

val score	train score	nb_α	fs_k	fs_score_func	thresholds
0.55	0.72	1.0	2000	chi2	(-0.01, 0.01)
0.56	0.84	0.2	10000	chi2	(-0.02, 0.02)
0.58	0.82	0.1	2000	chi2	(-0.03, 0.03)
0.59	0.79	0.7	1000	chi2	(-0.04, 0.04)
0.58	0.86	1.0	3000	chi2	(-0.05, 0.05)
0.63	0.96	0.1	6000	chi2	(-0.08, 0.08)
0.59	0.94	0.8	5000	chi2	(-0.10, 0.10)
0.63	0.98	0.2	3000	chi2	(-0.15, 0.15)

Table 19: Best naïve Bayesian network pipeline for each threshold after applying grid search

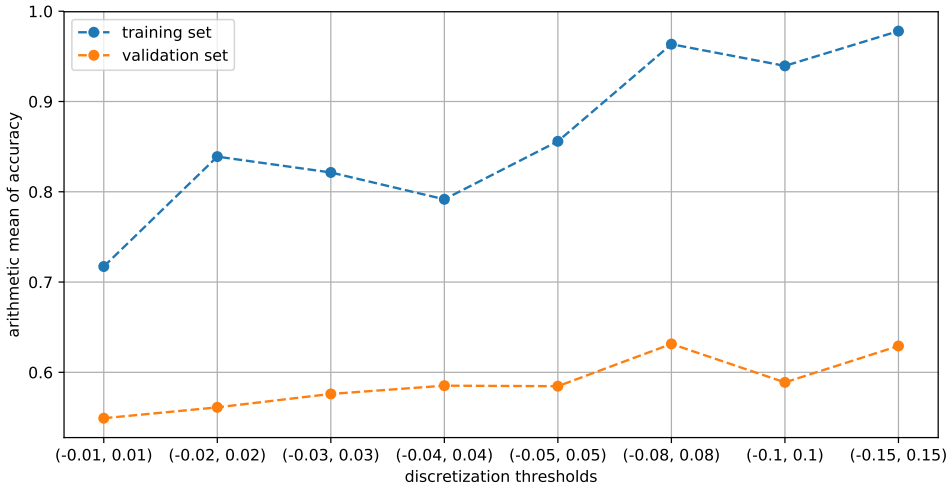
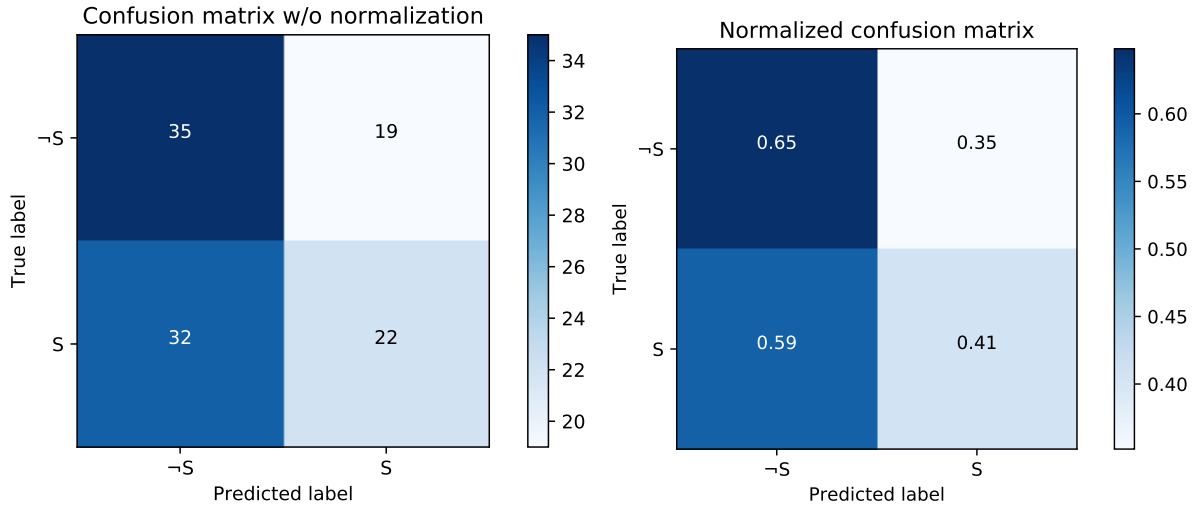


Figure 51: Best train and validation score of naïve Bayesian network pipeline on each dataset



(a) Confusion matrix with absolute counts

(b) Confusion matrix normalized by true label counts

Figure 52: Confusion matrix of best naïve Bayesian network pipeline trained and evaluated on the press release dataset discretized by $[-0.08, 0.08]$

In the quarter results plots of Figure 53 we see that the quarter results appear more frequently in the misclassification bins. Since the true quarter result rates are at approx. 60% we would expect them to be equally represented in each bin, which however is not the case. Thus, this model has difficulties interpreting quarter results.

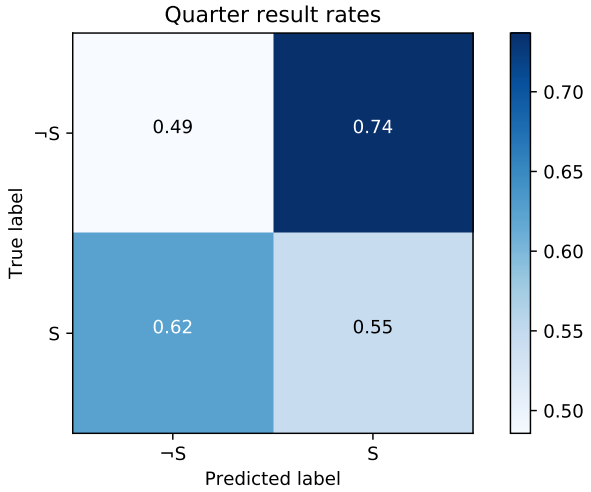


Figure 53: Quarter result rates in each confusion matrix bin of best naïve Bayesian network pipeline

Earlier we stated that the naïve Bayes classifier always picked the χ^2 test statistic as the favored feature selection scoring function. If we look at the top 200 words according to each scoring function as shown in Listing 21 and Listing 22, we see that both are completely different and intersect only in 45 elements. The χ^2 test statistic list does not deal with any specific topic, whereas in comparison especially the top words in the mutual information list deal with quarter results.

Even though the literature states that both scoring functions should provide the same results in practice, here we clearly have an example where this is not the case. In fact, the χ^2 test statistic is superior, as it was used as the scoring function for every best model.

If we compare the scores themselves to the scores in the impact words lists, we see that here the scores are much lower. This indicates that it is more difficult to assign words to the classes S ,$\neg S$ than to the classes I, $\neg I$.

```
rite:111 | der:92 | store:91 | offer:79 | relic:77 | aid:76 | shack:75 | offering:69 |
die:66 | retail:61 | switch:60 | western:60 | frontier:57 | cadence:57 | company:56 |
beauty:56 | decrease:55 | rh:55 | aircraft:55 | prospectus:54 | system:52 | tenet:50 |
pivotal:49 | window:48 | coal:45 | signet:45 | stock:43 | axon:43 | limelight:42 | sec
:41 | fourth:41 | fiscal:41 | de:40 | deposit:40 | mainly:39 | gallery:37 | due:37 |
share:36 | color:35 | partly:35 | corn:34 | impact:34 | registration:34 | pump:33 |
compare:33 | polaris:33 | bank:33 | solicitation:32 | lower:32 | receivables:32 |
elizabeth:32 | merger:32 | total:32 | trading:31 | griffon:31 | airline:31 | diabetes:30
 | partnership:30 | video:29 | horizon:29 | constant:29 | division:28 | pharmacy:28 |
luna:28 | automation:28 | global:27 | module:27 | agency:27 | insulin:27 | distributor
:27 | net:27 | fabric:26 | ore:26 | intercept:26 | shipment:26 | billing:26 | currency
:26 | fragrance:25 | propose:25 | flex:25 | slim:25 | glucose:25 | year:25 | billion:25
| carrier:25 | three:25 | third:24 | import:24 | quarter:24 | motorcycle:24 | von:24 |
cigarette:24 | rpm:24 | stellar:24 | print:24 | mail:24 | leaf:23 | coronary:23 | shake
:23 | architectural:23 | medical:23 | federate:23 | crc:23 | connecticut:22 | drop:22 |
subscription:22 | sleep:22 | water:22 | nutrition:21 | psychosis:21 | cosmetic:21 | skin
:21 | sensor:21 | ovarian:21 | proxy:20 | basin:20 | month:20 | tandem:20 | morgan:20 |
borrower:20 | group:20 | michael:20 | placebo:20 | finance:20 | steel:20 | exceed:19 |
dividend:19 | parkinson:19 | dental:19 | hormone:19 | disability:19 | cliff:19 |
gibraltar:19 | mattress:19 | artery:19 | favorable:19 | strong:19 | vessel:19 | ca:19 |
technology:18 | marketing:18 | client:18 | hair:18 | tender:18 | charter:18 | hearing:18
 | nielsen:18 | renal:18 | wheat:18 | nut:18 | adjusted:18 | camera:18 | buy:18 | gogo
```

```
:18 | taco:18 | kid:18 | ratio:18 | brazil:18 | jurisdiction:18 | charge:18 | tax:18 |
challenge:17 | drilling:17 | merchandise:17 | entertainment:17 | mass:17 | tropical:17 |
 drill:17 | rig:17 | auf:17 | paramount:17 | kitchen:17 | hear:17 | shall:17 | case:17 |
 strategic:17 | modern:16 | central:16 | auto:16 | ralph:16 | swap:16 | cantaloupe:16 |
tomato:16 | zebra:16 | blink:16 | reviewer:16 | den:16 | hedge:16 | center:16 | split:16
 | unfavorable:16 | diluted:16 | joint:15 | satellite:15 | organic:15 | margin:15 |
sixteen:15 | variance:15 | immunotherapy:15 | notably:15
```

Listing 21: Top 200 words according to χ^2 test statistic score

```
quarter:0.1 | million:0.09 | company:0.07 | revenue:0.07 | result:0.06 | income:0.06 |
increase:0.06 | fiscal:0.06 | adjust:0.06 | financial:0.06 | loss:0.06 | measure:0.06 |
include:0.06 | store:0.05 | year:0.05 | cash:0.05 | prior:0.05 | first:0.05 | net:0.05 |
 business:0.05 | march:0.05 | end:0.05 | share:0.04 | period:0.04 | total:0.04 | new
:0.04 | sale:0.04 | guidance:0.04 | compare:0.04 | percent:0.04 | security:0.04 |
product:0.04 | operate:0.04 | sec:0.04 | billion:0.04 | relate:0.04 | second:0.04 |
expense:0.04 | service:0.04 | decrease:0.04 | currency:0.04 | stock:0.04 | cost:0.04 |
tax:0.04 | june:0.04 | patient:0.04 | study:0.03 | customer:0.03 | network:0.03 | report
:0.03 | growth:0.03 | offering:0.03 | credit:0.03 | fourth:0.03 | file:0.03 | special
:0.03 | common:0.03 | jurisdiction:0.03 | offer:0.03 | impact:0.03 | trial:0.03 | use
:0.03 | know:0.03 | approximately:0.03 | record:0.03 | capital:0.03 | performance:0.03 |
 may:0.03 | per:0.03 | addition:0.03 | lower:0.03 | flow:0.03 | volume:0.03 | accordance
:0.03 | operation:0.03 | buy:0.03 | segment:0.03 | rate:0.03 | plan:0.03 | target:0.03 |
 interest:0.03 | development:0.03 | approval:0.03 | treatment:0.03 | point:0.03 |
acquisition:0.03 | item:0.03 | statement:0.03 | available:0.03 | gross:0.03 | expect
:0.03 | phase:0.03 | data:0.03 | earnings:0.02 | america:0.02 | exchange:0.02 | number
:0.02 | group:0.02 | change:0.02 | asset:0.02 | constant:0.02 | primarily:0.02 | market
:0.02 | risk:0.02 | margin:0.02 | solicitation:0.02 | free:0.02 | software:0.02 | ratio
:0.02 | potential:0.02 | comparable:0.02 | last:0.02 | prospectus:0.02 | technology:0.02
 | equipment:0.02 | video:0.02 | month:0.02 | basis:0.02 | third:0.02 | price:0.02 |
exclude:0.02 | continue:0.02 | respectively:0.02 | associate:0.02 | operating:0.02 |
system:0.02 | gain:0.02 | global:0.02 | document:0.02 | mobile:0.02 | retail:0.02 |
organic:0.02 | six:0.02 | lease:0.02 | order:0.02 | purchase:0.02 | information:0.02 |
center:0.02 | platform:0.02 | offset:0.02 | close:0.02 | higher:0.02 | combination:0.02
| research:0.02 | international:0.02 | additional:0.02 | cancer:0.02 | ability:0.02 |
program:0.02 | general:0.02 | incur:0.02 | level:0.02 | care:0.02 | contract:0.02 |
foreign:0.02 | three:0.02 | value:0.02 | unlawful:0.02 | brand:0.02 | registration:0.02
| health:0.02 | make:0.02 | expand:0.02 | due:0.02 | generally:0.02 | transaction:0.02 |
 production:0.02 | disease:0.02 | standard:0.02 | therapy:0.02 | pension:0.02 |
automotive:0.02 | ago:0.02 | condition:0.02 | reconciliation:0.02 | benefit:0.02 |
represent:0.02 | qualification:0.02 | executive:0.02 | subsidiary:0.02 | portfolio:0.02
| estimate:0.02 | useful:0.02 | life:0.02 | distribution:0.02 | activity:0.02 | exceed
:0.02 | strategic:0.02 | strong:0.02 | news:0.02 | april:0.02 | reflect:0.02 | set:0.02
| average:0.02 | recent:0.02 | marketing:0.02 | describe:0.02 | medical:0.02 | rise:0.02
 | repurchase:0.02
```

Listing 22: Top 200 words according to mutual information score

In Section 6.2.2, we synthesized two word lists from the naïve Bayesian network that contained the words whose presence was the best indicator for one of the classes I or $\neg I$. We have seen that especially terms out of quarter result domain are good predictors for impact.

Analogously, we created a word lists for each of the classes S and $\neg S$ as shown in Listing 23 and Listing 24, respectively. The two lists do not contain any words that a human being would characterize as especially positive or negative. In the two lists there are two signs for overfitting. First, there are many words in each list having probability close or equal to 1. Since Bayesian networks only learn the frequencies of the samples, this shows that some words just did not appear in the other class, thus leading to these high probabilities. Secondly, in Listing 23 there are terms like *parkinson*, *hormone*, *placebo*, *diabetes*, *insulin* and *immunotherapy*, which possibly indicate successful clinical studies causing some pharmaceutical stocks to rise. Unfortunately, these words are not applicable to the car manufacturing industry nor do they express any strong sentiment. In conclusion, these word lists do not provide any insights, as could be already seen when evaluating the naïve Bayesian model on the holdout set.

```
der:1.0 | relic:1.0 | shack:1.0 | die:1.0 | western:1.0 | frontier:1.0 | cadence:1.0 |
rh:1.0 | tenet:1.0 | signet:1.0 | axon:1.0 | limelight:1.0 | gallery:1.0 | insulin:1.0 |
 ore:1.0 | intercept:1.0 | glucose:1.0 | slim:1.0 | von:1.0 | rpm:1.0 | architectural
:1.0 | shake:1.0 | coronary:1.0 | psychosis:1.0 | tandem:1.0 | borrower:1.0 | cliff:1.0
| parkinson:1.0 | artery:1.0 | gibraltar:1.0 | hormone:1.0 | camera:1.0 | nut:1.0 | taco
```

```
:1.0 | rig:1.0 | auf:1.0 | tropical:1.0 | drill:1.0 | den:0.99 | ralph:0.99 | zebra:0.99
 | blink:0.99 | cantaloupe:0.99 | tomato:0.99 | leaf:0.97 | drop:0.97 | connecticut:0.97
 | de:0.96 | pump:0.96 | drilling:0.96 | crc:0.94 | sensor:0.94 | ovarian:0.94 |
diabetes:0.94 | automation:0.93 | pivotal:0.92 | basin:0.92 | receivables:0.91 | window
:0.91 | hearing:0.91 | modern:0.9 | hear:0.87 | placebo:0.87 | ca:0.86 | swap:0.86 |
immunotherapy:0.85 | steel:0.85 | agency:0.82 | stellar:0.81 | video:0.8 | entertainment
:0.8 | subscription:0.8 | exceed:0.79 | water:0.79 | central:0.78 | hedge:0.77 | finance
:0.75 | case:0.72 | medical:0.7 | system:0.7 | organic:0.69 | third:0.67 | group:0.66 |
three:0.65 | strong:0.65 | charge:0.63 | month:0.63 | technology:0.63 | quarter:0.52 |
net:0.51 | year:0.5 | company:0.5 | tax:0.5 | margin:0.49 | compare:0.49 | share:0.49 |
fiscal:0.47 | impact:0.47 | total:0.46 | adjusted:0.45 | due:0.45 | currency:0.44 |
billion:0.44 | global:0.44 | strategic:0.43 | diluted:0.42 | lower:0.41 | decrease:0.41
| center:0.41 | stock:0.4 | sec:0.4 | dividend:0.4 | fourth:0.4 | marketing:0.4 |
constant:0.39 | favorable:0.39 | joint:0.38 | ratio:0.37 | challenge:0.35 | client:0.34
```

Listing 23: The top 50 words for class S ranked by probability $p(S|t_i)$ out of 200 features

```
rite:1.0 | polaris:1.0 | elizabeth:1.0 | airline:1.0 | griffon:1.0 | luna:1.0 | pharmacy
:1.0 | flex:1.0 | motorcycle:0.99 | cigarette:0.99 | sleep:0.99 | cosmetic:0.99 | morgan
:0.99 | dental:0.99 | disability:0.99 | mattress:0.99 | gogo:0.99 | kid:0.99 | paramount
:0.99 | kitchen:0.99 | reviewer:0.99 | color:0.97 | corn:0.97 | beauty:0.96 | horizon
:0.96 | module:0.96 | switch:0.95 | deposit:0.95 | nutrition:0.95 | aircraft:0.94 |
wheat:0.94 | renal:0.94 | nielsen:0.94 | sixteen:0.93 | variance:0.93 | fabric:0.93 |
fragrance:0.92 | mail:0.92 | import:0.9 | split:0.89 | coal:0.89 | partly:0.89 | hair
:0.87 | tender:0.87 | charter:0.87 | print:0.87 | federate:0.87 | satellite:0.86 | aid
:0.85 | mass:0.84 | skin:0.84 | auto:0.84 | billing:0.82 | division:0.82 | carrier:0.82
| brazil:0.81 | vessel:0.8 | notably:0.8 | mainly:0.8 | trading:0.78 | prospectus:0.78 |
 bank:0.76 | merchandise:0.76 | solicitation:0.74 | distributor:0.74 | michael:0.74 |
registration:0.74 | shall:0.73 | buy:0.71 | offering:0.7 | shipment:0.7 | unfavorable
:0.69 | partnership:0.68 | offer:0.68 | propose:0.68 | jurisdiction:0.67 | proxy:0.67 |
retail:0.67 | store:0.67 | merger:0.66 | client:0.66 | challenge:0.65 | ratio:0.63 |
joint:0.62 | favorable:0.61 | constant:0.61 | marketing:0.6 | fourth:0.6 | dividend:0.6
| sec:0.6 | stock:0.6 | center:0.59 | decrease:0.59 | lower:0.59 | diluted:0.58 |
strategic:0.57 | global:0.56 | billion:0.56 | currency:0.56 | due:0.55 | adjusted:0.55 |
 total:0.54 | impact:0.53 | fiscal:0.53 | share:0.51 | compare:0.51 | margin:0.51 | tax
:0.5 | company:0.5 | year:0.5 | net:0.49 | quarter:0.48 | technology:0.37 | month:0.37 |
 charge:0.37 | strong:0.35 | three:0.35 | group:0.34 | third:0.33 | organic:0.31
```

Listing 24: The top 50 words for class $\neg S$ ranked by probability $p(\neg S|t_i)$ out of 200 features

6.3.3 Support Vector Machine

The support vector machine pipeline for sentiment prediction has been evaluated on the same hyperparameters as the support vector machine pipeline for impact prediction using grid search and provided the results shown in Table 20. In contrast to the naïve Bayes pipeline, the SVM model has primarily chosen the mutual information as the scoring function. If the model picked the χ^2 test statistic, then it was always combined with a high penalty C and a low polynomial degree. If the model picked mutual information as the scoring function then there is no such pattern present. However, we see that a higher polynomial degree is often connected to lower penalty C. Which means, that we allow the model to be complex due to the higher degree of the kernel. At the same time we, allow the model to be more inaccurate using a lower penalty. This adversary parameter combination enables the model to be complex without fitting to much noise.

If we plot the best performance for each dataset, we see that the maximum accuracy of 62% is being achieved for the (-0.08, 0.08) validation press release set, as shown in Figure 54. For bigger discretization spreads the validation accuracy decreases while the training accuracy approaches close to 100%. As training and validation set become smaller with higher discretization spreads, we deal with overfitting here.

We trained and evaluated the SVM pipeline on the press release set discretized by (-0.08, 0.08) and achieved an accuracy of only 47% making this classifier worse than random guessing. On the validation set we achieved earlier an accuracy of 62%, which shows that the prediction results are not stable.

val score	train score	fs_k	fs_score_func	svm_C	svm_degree	svm_kernel	thresholds
0.56	0.67	1000	chi2	1.0	1	poly	(-0.01, 0.01)
0.56	0.71	1000	mutual	0.7	1	poly	(-0.02, 0.02)
0.56	0.88	100	mutual	0.1	4	poly	(-0.03, 0.03)
0.60	0.78	1000	chi2	1.0	1	poly	(-0.04, 0.04)
0.58	0.78	2000	mutual	0.9	1	poly	(-0.05, 0.05)
0.62	0.87	200	mutual	0.1	2	poly	(-0.08, 0.08)
0.60	0.88	1000	mutual	1.0	1	poly	(-0.10, 0.10)
0.58	0.93	50	mutual	0.1	4	poly	(-0.15, 0.15)

Table 20: Best support vector machine pipeline for each threshold after applying grid search

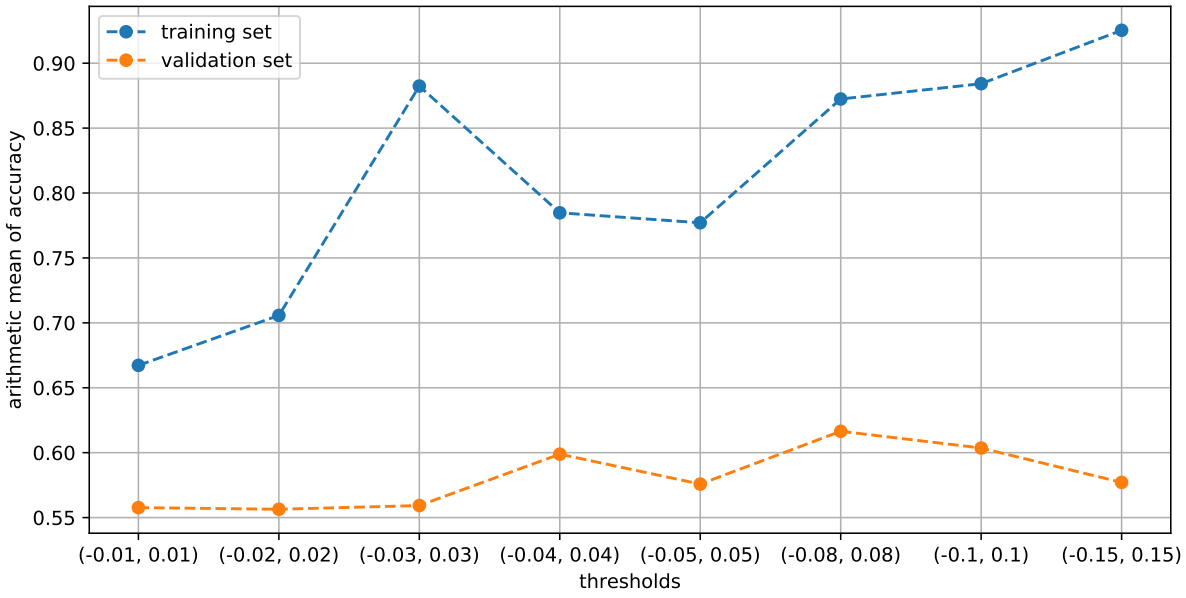
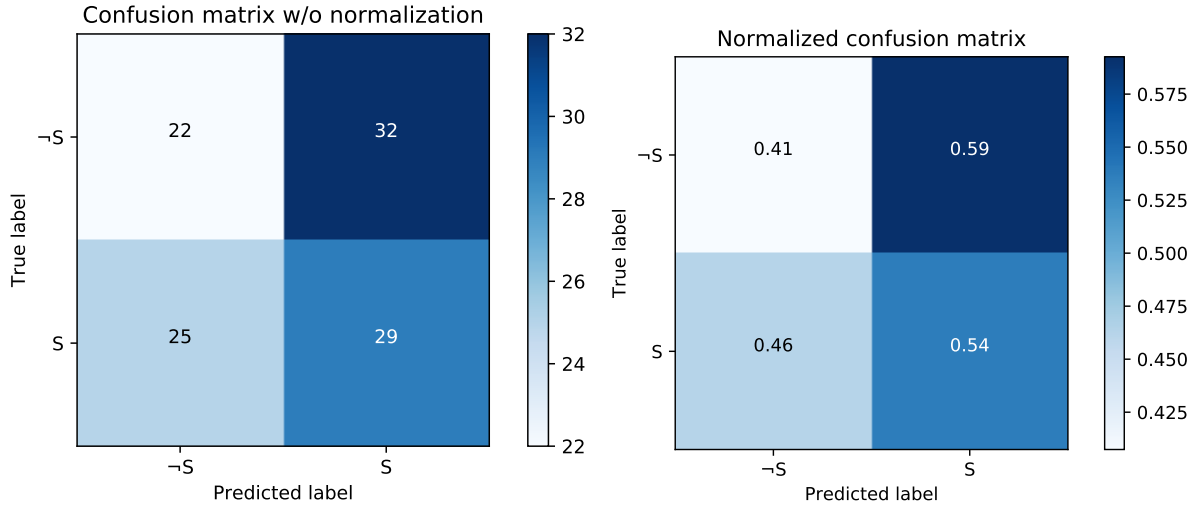


Figure 54: Best train and validation score of support vector machine pipeline on each dataset

As shown in Figure 55, the SVM classifier's low performance stems mostly from the negative sentiment samples. A percentage of 59% of truly negative samples are classified as positive.



(a) Confusion matrix with absolute counts

(b) Confusion matrix normalized by true label counts

Figure 55: Confusion matrix of best support vector machine classifier trained and evaluated on press release dataset discretized by $[-0.08, 0.08]$

The performance issues also cannot be explained, taking the quarter result rates for each confusion matrix bin into account. In fact, they would suggest that the performance especially on truly positive press releases is weak as the quarter result rate on the misclassifications is highest with 76%. Finally, one would have to investigate whether the size of the dataset could be the origin of the poor performance.

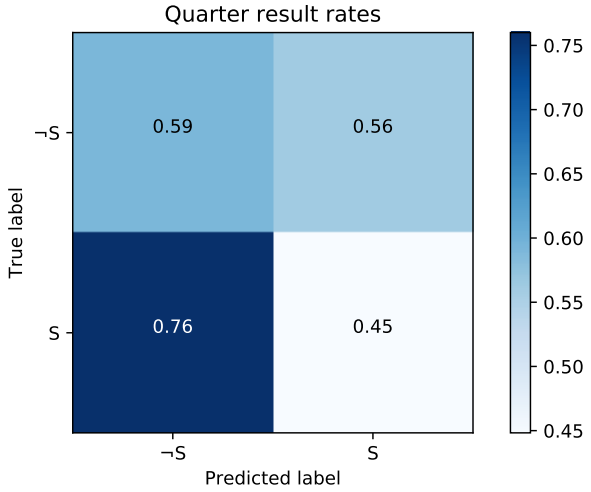


Figure 56: Quarter result rates for each true label and predicted label combination

6.3.4 Discussion

In this section, we compare the three algorithms on their sentiment prediction performance. Our main finding is that sentiment prediction is impossible due to the count vectorization of

the press releases. The sentiment in a text relies on the semantics, which gets lost completely when applying count vectorization.

For each of the three algorithms we performed grid search on each of the eight stratified press release sets. The accuracy of the best models on each data set increased with the width of the discretization thresholds. This result is expected, as the samples in each class become more and more dissimilar as we increase the width of the discretization threshold.

The feature count of the trivial classifier amounted to about 500, while the naïve Bayes classifier used about 10 times as many. The SVM classifier lay in between having about 1000 features, however varying between 50 and 2000 features. This relation was also seen for the impact models. Apparently, the SVM models generally needed less features than naïve Bayesian networks. The naïve Bayesian network chose the χ^2 test statistic exclusively, while the SVM classifier picked the χ^2 test statistic only in combination with a high penalty and mutual information otherwise.

We have shown, that the features themselves provided by χ^2 test statistic and mutual information contained almost no common terms. Surprisingly, none of the top 200 terms in both lists contained words with real sentiments even though we had two classes which are supposed to contain strong sentiments.

On the other hand, this shows that press release sentiment is more complicated and cannot be captured by counting frequencies (count vectorization). One major reason is that texts lose their entire structure, thus missing important aspects like negations, irony or the conjunction of words, which as a whole provide sentiment.

For a final evaluation each algorithm was trained on the stratified press release set discretized by $(-0.08, 0.08)$. Following the GIGO axiom "garbage in garbage out", the three algorithms provided poor results. The trivial classifier and the naïve Bayesian network both had an accuracy of 53%. The SVM classifier was even worse having an accuracy of 47%.

At first glance our weak results contradict with Koppel's in [15]. Koppel demonstrated that sentiment prediction based on news articles with an average length of 100 words is possible. He labeled the news articles by discretizing the SNO on (-0.078, 0.1). The binary vectorized news articles were used to train a linear SVM, naïve Bayesian network and a decision tree. These models provided all an accuracy of approximately 70%.

This rises the question, why do his models yield such high accuracies, while ours are a complete failure. First of all, the data sets are not the same. In contrast to the companies publishing a negative press release, journalists do not tend to beat around the bush, especially in shorter news. This is why, we can expect stronger words in the news articles than in press releases.

However, switching to sentiment prediction based on news articles is not an option. Koppel showed that the approach was not practical in the field, as the stock price had already adapted, when an article was published. This means that the information must have been available to the traders beforehand. Since the SNO of the news articles had been discretized on thresholds with a large width , we are confident that these news articles were referring to corporate news which had been already published in form of press releases. In our experience news with such a strong sentiment almost always stem from quarter results, earnings warnings or company acquisitions. This renders news article analysis useless and makes press release analysis mandatory.

To sum up, we have shown, that sentiment prediction is a difficult task – more difficult than impact prediction. None of our models showed a performance that was significantly better than random guessing. As it has been pointed out, news articles are not a solution switch to and improving the models on press releases is the way to go. The crucial part here is to

find a representation that keeps the sentiment and can also be understood by machine learning algorithms.

7 Summary

In this research project, we developed and evaluated two models, namely a naïve Bayesian network and a support vector machine, to predict stock price trends based on press releases. Additionally, we introduced the trivial classifier in order to put the results of the two more sophisticated models into perspective. Furthermore, we investigated if despite the efficient market hypothesis, the market expresses any inefficiencies due to press releases that we could leverage.

As part of the data mining process, we deployed the infrastructure to retrieve after hours press releases and stock prices with one minute resolution covering all companies traded on the NASDAQ and NYSE. In the preprocessing, the press releases were extracted from the HTML and cleaned (tokenization, stop word removal, POS tagging, lemmatization and count vectorization). Using a regular expression, every cleaned press release was assigned to the company, whose stock symbol appeared in the text. This assignment made it possible to label the press releases with the relative stock prices changes from exchange closing to exchange opening plus an offset. We called the relative stock price change *simple net opportunity (SNO)*.

In order to capture the influence of press releases, it was important to determine the correct offset. In our analysis, we compared the general stock price volatility over night to the stock price changes over night, when a press release was published. According to our analysis the press releases' influence was fully reflected on the stock price 5 h after exchange opening. Using this window, we plotted a histogram of the general SNO when there was no press release and another histogram of the SNO when there was a press release published. The former one was narrower than the latter one, thereby showing that press releases have an influence which is not immediately reflected on the stock price. As a consequence, the market is de facto not always efficient and thereby contradicting with the efficient market hypothesis.

The Gaussian shaped histogram of the press releases also revealed that most of the press releases had no influence. This is why, we proposed a two step solution: In the first step, we trained a model to predict, whether a press has an impact on the stock prices or not, regardless of the direction. For the second step, we only considered the press releases with an impact and built another model to predict the direction i.e., the sentiment, of the press releases.

Since the models in each step solve classification tasks, the continuous SNO values assigned to the press releases needed to be discretized. The two histograms revealed that the discretization threshold $(-0.08, 0.08)$, was a well-chosen trade-off between shrinking the stratified and balanced dataset and allowing too much noise due to regular stock price volatility. In the context of datasets balanced means that every class is represented equally. The balanced impact data set contained 565 press releases with impact and the sentiment data set contained 267 press releases with positive sentiment.

Having trained and evaluated the three impact models of the first step on the discretized data set, the trivial model had an accuracy of 77% which is competitive to the SVM classifier and naïve Bayesian network which had an accuracy of 76% and 78%, respectively. Even though the high accuracies seem to be promising, a press release predicted to have an impact is still five times more likely to be of class *no impact*. This stems from the distribution of press releases, where high impact press releases make up only 6% of the press releases.

From the CPTs in the naïve Bayesian network, we derived the words whose presence are the best predictors for each class. Here, we saw that for class *impact* the model learned that terms related to quarter results are the best predictor. For the *no impact* class we could not detect any topic.

The three sentiment models, trained by trivial classifier, naïve Bayesian network and the support vector machine, reached accuracies of 53%, 53% and 47% on the holdout set, respectively. In conclusion these three models did not learn anything that generalizes well. Again, we generated sentiment word lists from the CPTs in the naïve Bayesian network. These words also did not provide any useful information and no sentiment whatsoever.

The reasons why the sentiment models have such a bad performance are diverse. On the one hand, a company always wants to avoid bad news and keep its reputation. This is why, they do their best to hide bad news behind positive words. On the other hand, our methodological approach has its limitations. The sentiment cannot be expressed by the frequency of words in a document (count vectorization). All the context goes missing along with the semantics. A negation, for instance, would never be recognized by a model that was trained on count vectorized press releases. This insight is really important, since this gives us a starting point to begin with. In Section 8, we will present a novel idea how to enhance impact and sentiment prediction.

In summary, it is generally possible to reliably predict press release impact if the data sets are balanced. Also, for a trader it is a useful information if among five press releases there is one having an impact of at least 8%. This saves lots of manual filtering time. At the same time, we are confident that we can further push the accuracy, if we have more data available and rethink the representation of press releases.

8 Appendix

References

[1] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN: 9780387310732.

[2] William B Cavnar, John M Trenkle, et al. "N-gram-based text categorization." In: *Ann arbor mi* 48113.2 (1994), pp. 161–175.

[3] Stanley F Chen and Joshua Goodman. "An empirical study of smoothing techniques for language modeling." In: *Computer Speech & Language* 13.4 (1999), pp. 359–394.

[4] The Association for Computational Linguistics. *POS Tagging (State of the art)*. https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art)/. Accessed: 2018-02-23. July 2016.

[5] STEFANO DELLAVIGNA and JOSHUA M. POLLET. "Investor Inattention and Friday Earnings Announcements." In: *The Journal of Finance* 64.2 (), pp. 709–749. DOI: 10.1111/j.1540-6261.2009.01447.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.2009.01447.x.

[6] Sanjiv R Das and Mike Y Chen. "Yahoo! for Amazon: Sentiment extraction from small talk on the web." In: *Management science* 53.9 (2007), pp. 1375–1388.

[7] Eugene F Fama. "The behavior of stock-market prices." In: *The journal of Business* 38.1 (1965), pp. 34–105.

[8] George Forman. "An extensive empirical study of feature selection metrics for text classification." In: *Journal of machine learning research* 3.Mar (2003), pp. 1289–1305.

[9] A. Gelman et al. *Bayesian Data Analysis, Third Edition*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2013.

[10] Michel Généreux, Thierry Poibeau, and Moshe Koppel. "Sentiment analysis using automatically labelled financial news." In: *LREC 2008 Workshop on Sentiment Analysis: Emotion, Metaphor, Ontology and Terminology*. 2008.

[11] Gyozo Gidofalvi and Charles Elkan. "Using news articles to predict stock price movements." In: *Department of Computer Science and Engineering, University of California, San Diego* (2001).

[12] Terrence Hendershott. "Electronic trading in financial markets." In: *IT professional* 4 (2003), pp. 10–14.

[13] Thorsten Joachims. "Text categorization with Support Vector Machines: Learning with many relevant features." In: *Machine Learning: ECML-98*. Ed. by Claire Nédellec and Céline Rouveirol. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142.

[14] Armand Joulin et al. "Bag of tricks for efficient text classification." In: *arXiv preprint arXiv:1607.01759* (2016).

[15] Moshe Koppel and Itai Shtrimberg. "Good news or bad news? let the market decide." In: *Computing attitude and affect in text: Theory and applications*. Springer, 2006, pp. 297–301.

[16] Huma Lodhi et al. "Text classification using string kernels." In: *Journal of Machine Learning Research* 2.Feb (2002), pp. 419–444.

[17] Ann C Logue. *Day trading for dummies*. John Wiley & Sons, 2014, p. 158.

[18] Max Lübbering. "Predicting Companies Mentioned in News Articles, a Comparison of Two Approaches: Latent Dirichlet Allocation with k-Nearest Neighbor versus Bag of Words with k-Nearest Neighbor." In: (May 2018). URL: https://hps.vi4io.org/_media/research/labs/2018/2018-04-max_l_bbering-predicting_companies_mentioned_in_news_articles_a_comparison_of_two_approaches_latent_dirichlet_allocation_

with_k_nearest_neighbor_versus_bag_of_words_with_k_nearest_neighbor-report.pdf.

[19] Burton G Malkiel. "The efficient market hypothesis and its critics." In: *Journal of economic perspectives* 17.1 (2003), pp. 59–82.

[20] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[21] Andrew McCallum, Kamal Nigam, et al. "A comparison of event models for naive bayes text classification." In: *AAAI-98 workshop on learning for text categorization*. Vol. 752. 1. Citeseer. 1998, pp. 41–48.

[22] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[23] M. A. Mittermayer. "Forecasting Intraday stock price trends with text mining techniques." In: *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*. 2004. DOI: 10.1109/HICSS.2004.1265201.

[24] A.C. Müller, M.A. C, and S. Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016.

[25] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[26] Mehran Sahami et al. "A Bayesian approach to filtering junk e-mail." In: *Learning for Text Categorization: Papers from the 1998 workshop*. Vol. 62. Madison, Wisconsin. 1998, pp. 98–105.

[27] Robert P. Schumaker and Hsinchun Chen. "Textual Analysis of Stock Market Prediction Using Breaking Financial News: The AZFin Text System." In: *ACM Trans. Inf. Syst.* 27.2 (Mar. 2009), 12:1–12:19. ISSN: 1046-8188. DOI: 10.1145/1462198.1462204. URL: http://doi.acm.org/10.1145/1462198.1462204.

[28] Paul C Tetlock. "Giving content to investor sentiment: The role of media in the stock market." In: *The Journal of finance* 62.3 (2007), pp. 1139–1168.

[29] Paul C Tetlock, Maytal Saar-Tsechansky, and Sofus Macskassy. "More than words: Quantifying language to measure firms' fundamentals." In: *The Journal of Finance* 63.3 (2008), pp. 1437–1467.

[30] Ruey S Tsay. *Analysis of financial time series*. Vol. 543. John Wiley & Sons, 2005.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig sowie ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Masterarbeit stehen. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Ich versichere an Eides Statt, dass ich nach bestem Wissen die reine Wahrheit gesagt und nichts verschwiegen habe.

Vor Aufnahme der obigen Versicherung an Eides Statt wurde ich über die Bedeutung der eidesstattlichen Versicherung und die strafrechtlichen Folgen einer unrichtigen oder unvollständigen eidesstattlichen Versicherung belehrt.

Hamburg, 3. Dezember 2018

_____

Max Lübbering