TUHH

Hamburg University of Technology

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

– Research Project –

Author:
Max Lübbering, 21599173

Supervisors:
Dr. Julian Kunkel
Dr. Patricio Farrell

Examiner:
Dr. Patricio Farrell

May 23, 2018

Contents

List of Figures

List of Tables

Listings

This report proposes a pipeline consisting of Latent Dirichlet Allocation (LDA) and k-nearest neighbor to predict the company a given news article is about. This approach builds the ground work for a fully autonomous trading system based on news article sentiment.

We show that a LDA topic model trained with company descriptions, can classify news articles when combined with k-nearest neighbor using Kullback-Leibler divergence as similarity measure.

The results of this approach are competitive with the results of a simple bag-of-words k-nearest neighbor pipeline, while reducing the feature space to a fraction.

Nomenclature

Bayesian Statistics

Θ	Model parameter(s)
$\tilde{y}$	Unobserved data
$p(\cdot\|\cdot)$	Conditional probability
$p(\cdot)$	Marginal distribution / marginal density
$p(a, b)$	Joint probability of random variables a and b
y	Observed data

Latent Dirichlet Allocation (LDA)

β_k	Word distribution of the k^{th} topic
$\boldsymbol{w_d}$	All words in document d
$\mathbf{v}$	The set of all words in a corpus (vocabulary)
$\mathcal{D}$	Corpus consisting of D documents.
Θ_d	Distribution of the topics in the d^{th} document
$\Theta_{d,k}$	Proportion of topic k in document d
d	Document index
K	No. of topics in the model
k	Topic index
M	No. of documents in the corpus $\mathcal{D}$
N	No. of words in a specific document
n	Word index
$w_{d,n}$	n^{th} word in document d
z_d	Topic assignments for all words in document d
$z_{d,n}$	Topic assigned to n^{th} word in document d

1 Introduction

In this section, we motivate why news article analysis is a big deal for financial decision making and analyze the requirements for an autonomous trading system. We introduce an approach that builds the ground work for financial decision making and introduce a more simple approach, we want to compete with.

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.

Publicly traded companies are also 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 [10]. They usually have installed a notification alarm by some news provider, providing them with the latest news regarding the companies in their portfolio. They continuously read these news articles and choose one of the three options: *hold*, *buy* or *sell*.

The objective of this paper is to utilize machine learning and data mining algorithms to automatically match news articles to the corresponding companies. Based on this work, we can perform further analysis (e.g. sentiment analysis), which brings us a couple of steps closer to autonomous financial decision making.

1.2 Goals

In particular, we want solve the following tasks:

- News article acquisition:
  For the subsequent tasks we have to collect a huge amount of news articles and store them into a database.
- News article to company matching:
  We want to train a machine learning model that can predict the company mentioned in a given news article.

- Sentiment prediction:
  For a given news article, we want to know how positive or negative it is.
  (As the preceding task turned out to be rather complex and already covers the entire scope of the research project, we decided to move this goal to the author's master thesis.)

In this paper, we utilize company descriptions, that are labeled by the corresponding company name / stock symbol, to build a model that can predict the companies mentioned in a news article.

Our goal is to build a model that performs more reliably than a simple bag-of-words approach. The bag-of-words approach 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, which can be represented as a matrix

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

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.

The simple idea is to represent the company descriptions as well as the news articles in a bag-of-words model and for each news article use its k most similar company descriptions for prediction (e.g. k-nearest neighbor algorithm with majority vote).

This approach compares token counts directly, which often leads to the two problems:

- Bag-of-words models often have a huge feature space, which makes similarity calculations between documents expensive and time consuming.
- Unimportant features add a lot of noise to the similarity measurement, thus leading to models that do not generalize will in practice.

To circumvent these shortcomings, we propose another approach based on Latent Dirichlet Allocation (LDA) and k-nearest neighbor. This approach takes the bag-of-words result and trains a topic model using LDA. A topic model contains a set of topics and predicts the topic distribution for a given news article. For instance, if we had a news article about Tesla Inc., then this news article would probably have high amplitudes for the topics *Green energy*, *Mobility*, *Car manufacturing* and *Electricity*. The other unrelated topics (e.g. *Soccer*, *Food* or *Clothing*) would have amplitudes close to zero.

We then compare the news article's topic distribution with the topic distribution of each company description and pick the k closest ones. In fact, this is the k-nearest neighbor algorithm. We assume that Tesla's company descriptions are most likely more similar to the news article than the other company descriptions and therefore generally leads to the correct results.

1.3 Outline

This research report is structured in eight sections. After this introductory chapter, we will explain the statistics and intuition behind Latent Dirichlet Allocation and k-nearest neighbor. We then give an overview of the current algorithms used for solving problems in this area and the research others have been doing on this topic. In Section 4, we explain the machine learning pipeline for each approach, including data acquisition, preprocessing, model training and company prediction. Section 5 is structured identically to the preceding section and shows how each part of the pipeline is implemented. In the following section, we evaluate both approaches and discuss each approach's advantages and disadvantages. We conclude with a summary of our findings and an outlook for future work.

2 Background

In this section, we explain the mathematics and algorithms used in our research. First, we give a short introduction to Bayesian statistics. Then we show, how Latent Dirichlet Allocation uses the Bayesian statistics framework to calculate a topic model from a set of documents. Finally, we show how the k-nearest neighbor algorithm can be used for categorical distributions.

2.1 Introduction to Bayesian Statistics

As LDA builds upon Bayesian inference, we will give a short introduction on the basics of Bayesian statistics and how to use it. There exists a great book on Bayesian statistics called "Bayesian Data Analysis"[8], whose approach and notation we will follow.

Other than the text book, we will specifically mark distribution's arguments when it is unclear whether they are parameters / 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 (i.e. for $p(y|\underline{\Theta})$, y would be the constant and Θ would be the variable).

When performing Bayesian inference, we always have an unknown population. 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). Assuming we have 10 males and 12 females as evidence, we want to determine the likelihood of $p(\Theta = 0.1 \,|\, males = 12, females = 10)$, which is low because the observed data contradicts with such Θ.

The other objective assumes that we do know the distribution of $p(\Theta|y)$ (i.e. we know the likelihood of all sex ratios in a country given 10 males and 12 females) and we want to predict the probability of unseen data $p(\tilde{y}|y)$. 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 Θ for a given 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* [1] 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[2] is being calculated from the *joint probability* by summing out / integrating over Θ:

$$p(y) = \int p(y, \Theta)d\Theta = \int p(\Theta)p(y|\Theta)d\Theta. \tag{3}$$

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

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

[2] 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.

We will now derive the formula for calculating $p(\tilde{y}|y)$.

$$\begin{aligned} p(\tilde{y}|y) &= \int p(\tilde{y}, \Theta|y) d\Theta \\ &= \int p(\tilde{y}|\Theta, y) p(\Theta|y) d\Theta \\ &= \int p(\tilde{y}|\Theta) p(\Theta|y) d\Theta \end{aligned} \tag{4}$$

The last step follows from the conditional independence of y and $\tilde{y}$ given Θ.

Example (adopted from [8])

We now illustrate Bayesian inference by estimating the sex ratio in a population using a single parameter model. The sex ratio is denoted as Θ and y is the number of girls out of n births. The sex of two subsequent births can be considered independent of each other and both have the same sex probabilities. That is why in this example the sampling distribution can be modeled using the binomial distribution

$$p(\underline{y}|\Theta) = Bin(y|n, \Theta) = \binom{n}{y} \Theta^y (1-\Theta)^{n-y}.$$

Note that n is a hyperparameter that is set once for the entire experiment and is therefore not part of the model.

Making use of Bayes theorem (see Equation (1)) and assuming the prior probability $p(\Theta)$ to be uniform on $[0, 1]$, we get the unnormalized posterior density:

$$p(\underline{\Theta}|y) = \frac{p(y|\underline{\Theta}) p(\Theta)}{p(y)} \tag{5}$$

$$\propto p(y|\Theta) p(\Theta) \tag{6}$$

$$\propto p(y|\Theta) \tag{7}$$

$$\propto \binom{n}{y} \Theta^y (1-\Theta)^{n-y}. \tag{8}$$

Since y is constant, $\binom{n}{y}$ and $p(\Theta)$ can be ignored up to a scaling factor. The unnormalized posterior density is a function of Θ and belongs to the family of Beta distributions

$$\Theta|y \sim Beta(y+1, n-y+1).$$

The $\sim$ notation states that the random variable Θ given y is beta distributed.

Now, let us assume that we have evidence that suggests that Θ is not uniformly distributed but in fact a beta distribution parameterized by the hyperparameters μ and ν. In that

case we have to modify the posterior density as shown in Equation (9)

$$p(\underline{\Theta}|y) \propto \overbrace{\Theta^y(1-\Theta)^{n-y}}^{likelihood}\overbrace{\Theta^{\alpha-1}(1-\Theta)^{\beta-1}}^{prior} \tag{9}$$

$$= \Theta^{y+\mu-1}(1-\Theta)^{n-y+\nu-1}$$

$$= Beta(\Theta|\mu+y, \nu+n-y) \tag{10}$$

It is important to note here, that the posterior density is also a beta distribution just like the prior probability $p(\Theta)$. This property is known as the *conjugacy property*, which is formally defined in the next paragraph. In other words, the beta distribution is a conjugate family for the binomial likelihood.

De facto it is very convenient to make use of conjugate priors, because not only do they simplify the calculation of the posterior density, they also make the posterior density more interpretable. On the other hand, if evidence contradicts with the conjugate prior distribution, we have to come up with a more sophisticated prior distribution at the expense of the two advantages above.

Conjugate prior distributions

The conjugate prior distribution is defined by

$$p(\Theta|y) \in \mathcal{P} \quad \forall \quad p(\underline{\cdot}|\Theta) \in \mathcal{F} \quad \wedge \quad p(\cdot) \in \mathcal{P}, \tag{11}$$

where $\mathcal{F}$ is a class of sampling distributions $p(y|\Theta)$ and $\mathcal{P}$ is a class of prior distributions for Θ.

Class $\mathcal{P}$ is conjugate for class $\mathcal{F}$ if Definition (11) holds.

To make this more illustrative, we annotated Equation (2) with their corresponding classes in Equation (12). No matter what distribution out of $\mathcal{P}$ we pick as the prior distribution and what distribution we pick out of $\mathcal{F}$ for the likelihood , the posterior density has to be in $\mathcal{P}$. If this condition holds for all variations, the prior distributions family is a conjugate family for the likelihood's family.

$$\overbrace{p(\Theta|y)}^{\in\mathcal{P}} \propto \overbrace{p(\Theta)}^{\in\mathcal{P}}\overbrace{p(y|\underline{\Theta})}^{\in\mathcal{F}}) \tag{12}$$

Definition (11) is rather vague. For instance if we set $\mathcal{P}$ to be the class of all distributions. Then in this trivial example every prior probability is conjugate independently of the class of the likelihood.

In practice we often stick to natural conjugate prior families, which enforces $\mathcal{P}$ to be of

the same function form as the likelihood. The sex ratio of births example from above was modeled in the second step using natural conjugate priors.

2.2 Topic Modeling: Foundations of Latent Dirichlet Allocation

A topic model represents a corpus by the the topics it contains. If we had a corpus (set of documents) that dealt with universities, there could be the following topics: *student life*, *lectures*, *research areas* and so on.

The unsupervised learning task of learning the topics of a corpus is done by Latent Dirichlet Allocation (LDA) and explained in detail in the following section. We strictly follow the nomenclature introduced by Blei in his original paper on LDA [4]. For future reference the nomenclature is also listed in the beginning of the report.

Intuition of the generative process

LDA makes the assumption that documents are being generated from a fixed number of topics. Each topic has the full vocabulary (set of all distinct words in a corpus) and assigns a probability to each word. This probability represents how connected a word is to the respective topic, as shown in Figure 1. The first topic deals with genetics and therefore assigns a high probability to words like *gene*, *DNA*, or *genetic*. The last topic deals with computer science and therefore assigns high probabilities to words like *data*, *number* and *computer*.

When starting the document generation, we first choose the number of words N of the document. We then sample a categorical distribution over the topics, which is displayed as a bar chart in Figure 1. The topic distribution is sampled from a Dirichlet distribution, which we will discuss later in this chapter.

For each of the N undetermined words, we sample a topic according to the topic distribution. These assignments are displayed as the colored circles left to the bar chart.

Each of the undetermined words in the document then samples a word from the vocabulary distribution according to its topic assignment.

To make this more illustrative: The first topic assignment is the upper pink circle. The word is therefore being sampled from the second topic. In this specific case the word life was picked from the vocabulary distribution of the second topic.

Listing 1 shows the basic steps of LDA that we described above:

```
1.) Choose number of words N of the document
2.) Sample distribution of topics
3.) Assign topic to each word by sampling from the topic distribution
```

```
4.) Choose concrete word by sampling from the assigned topic
```

Listing 1: Basic steps of Latent Dirichlet Allocation

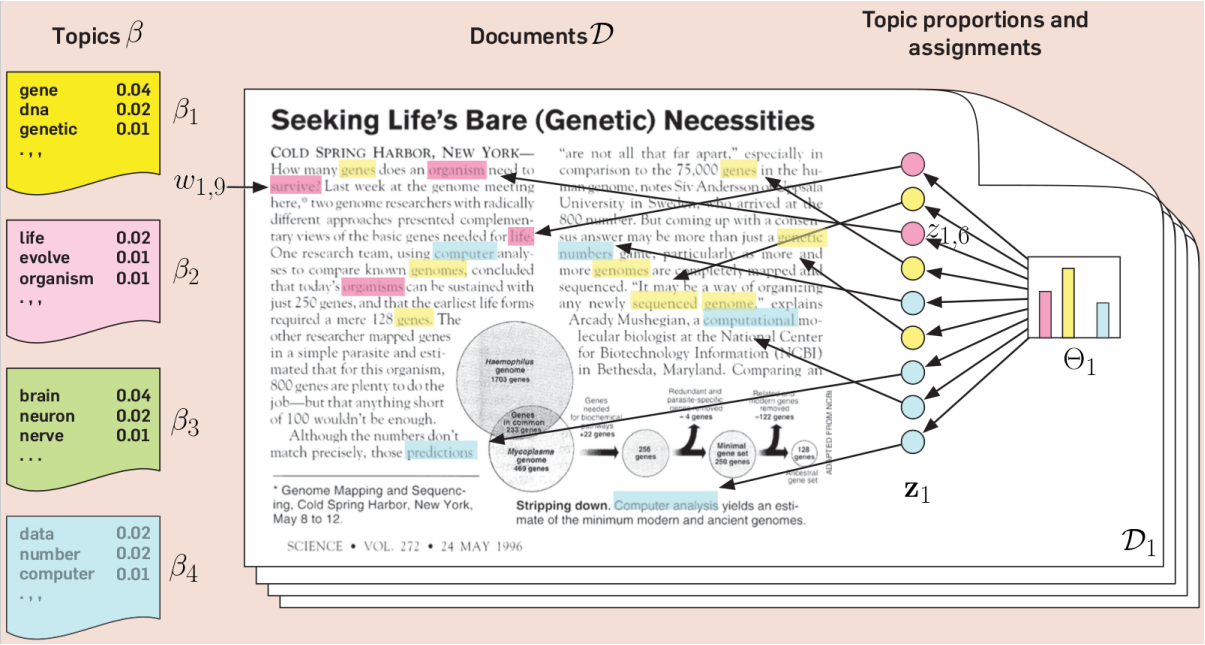


Figure 1: Generation of documents[3][3]

LDA depends on a lot of random variable, which are briefly described in Table 1.

[3]We took the liberty of adding the random variables to the diagram.

Symbol	Dim.	Description
$\mathcal{D}$	D	Corpus consisting of M documents.
N	1	No. of words in a specific document (for simplicity we assume that each document has the same length)
K	1	No. of topics in the model
M	1	No. of documents in the corpus $\mathcal{D}$
n		Word index
k		Topic index
d		Document index
$\mathbf{v}$	$\lvert\mathbf{v}\rvert$	The set of all words in a corpus (vocabulary)
$w_{d,n}$	1	n^{th} word in document d
$\boldsymbol{w_d}$	N	Row vector containing the words of document d
w	$M \times N$	Matrix containing the words of each document
β_k	$\lvert\mathbf{v}\rvert$	Word distribution of the k^{th} topic
β	$K \times \lvert\mathbf{v}\rvert$	Word distributions of all topics
$\Theta_{d,k}$	1	Proportion of topic k in document d
Θ_d	K	Distribution of the topics in the d^{th} document
Θ	$M \times K$	Distribution of the topics of each document
$z_{d,n}$		Topic assigned to n^{th} word in document d
$\mathbf{z}_d$	N	Topic assignments for all words in document d
z	$M \times N$	Matrix of topic assignments for all words in each document

Table 1: Symbols used in Latent Dirichlet Allocation

Once we understand the entire generative process, we can calculate the joint probability of all hidden (latent) variables and the observed variables (words of the documents). We then condition the joint probability on the words to get the posterior density. As we will see, the marginal probability $p(words)$ is part of the equation and is unfeasible to calculate. The true posterior density therefore has to be approximated (e.g. using sampling-based algorithms or variational algorithms)[3].

Generative process in-depth

The generative process of a complete corpus $\mathcal{D}$ can be broken down into the steps of iteratively creating single documents. The creation of a single document is shown in Listing 2. According to the nomenclature, index d refers to the d^{th} document in the corpus. Note that we omitted this index, since the generation of a specific document is being discussed here, which renders the use of index d redundant.

At first, we sample the number of words N our document shall consist of, from a Poisson distribution[4], that was parameterized by ξ. Blei notes that N is independent of all the

[4]The $\sim$ notation is used a little different, here. $N \sim Poisson(\xi)$ usually means that the random variable N has the probability distribution of $Poisson(\xi)$. However, in our case we can think of it as sampling from the Poisson distribution directly.

other data generating variables (i.e. the document's topic distribution Θ and the words' topic assignments $\boldsymbol{z}$) and can therefore be considered as an ancillary variable[5][4]. This is why the randomness of the number of words N can be ignored, when deriving the formulae of the generative process.

After having sampled / set the document's number of words, we determine the documents' topic distribution Θ, by sampling from a Dirichlet distribution $Dir(\boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ is a vector that parameterizes the distribution.

The following paragraph describes how the sampling process works. The document's topic distribution Θ is defined as a vector

$$\Theta = \begin{pmatrix} \Theta_1 \\ \Theta_2 \\ \vdots \\ \Theta_K \end{pmatrix}, \text{where} \quad \Theta_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{K} \Theta_i = 1. \tag{13}$$

Each Θ_i represents the proportion of the i^{th} topic in the document. Figure 2 shows an exemplary topic distribution which states that the document is primarily generated by the second and third topic. The literature often says that Θ is a multinomial distribution. However, as this is the special case of a multinomial distribution "with one trial", we can also refer to it as the categorical distribution.

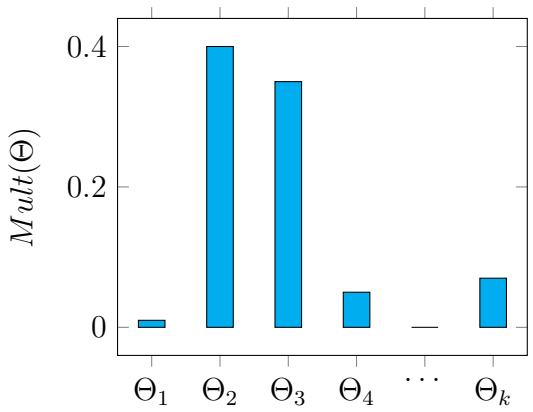


Figure 2: Multinomial distribution of Θ

After having sampled a topic distribution, we are now able to sample each word individually (see for-loop in listing 2). For the n^{th} word we pick a topic assignment z_n by sampling a topic from the distribution $Mult(\Theta)$. To determine the word, we sample from $p(w_n|z_n, \beta)$, which is the probability of a each word in the vocabulary, conditioned on the topic assignment z_n and the word distributions for each topic β.

[5]Variable that changes due to external characteristics.

```
Choose N ~ Poisson(ξ)
Choose Θ ~ Dir(α)
For each of the N words w_n:
  Choose a topic z_n ~ Multionomial(Θ)
  Choose a word w_n from p(w_n|z_n, β)
```

Listing 2: Algorithmic representation of LDA's generative process [4]

Dirichlet distribution

As we sample Θ from a Dirichlet distribution, we now formally define the Dirichlet distribution by

$$p(\Theta|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} \Theta_i^{\alpha_i - 1} \quad \text{s.t.} \quad \Theta_i \geq 0, \tag{14}$$

$$\sum_{i=1}^{K} \Theta_i = 1 \quad \forall \quad i \in \{1, 2, \ldots, K\} \quad \text{and}$$

$$\alpha_i > 0 \quad \forall \quad i \in \{1, 2, \ldots, K\}$$

The function $B(\alpha)$ is the multivariate beta function and serves as a normalizing constant. We define the beta function in terms of the gamma function $\Gamma(x)$:

$$B(\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)}. \tag{15}$$

With regard to the constraints of the Dirichlet distribution, its restricted domain is limited to a $K-1$ simplex, as shown in figure 3. The green area represents all valid topic distributions of a document.

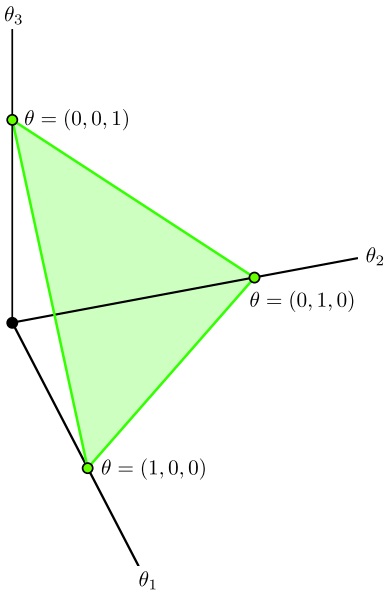
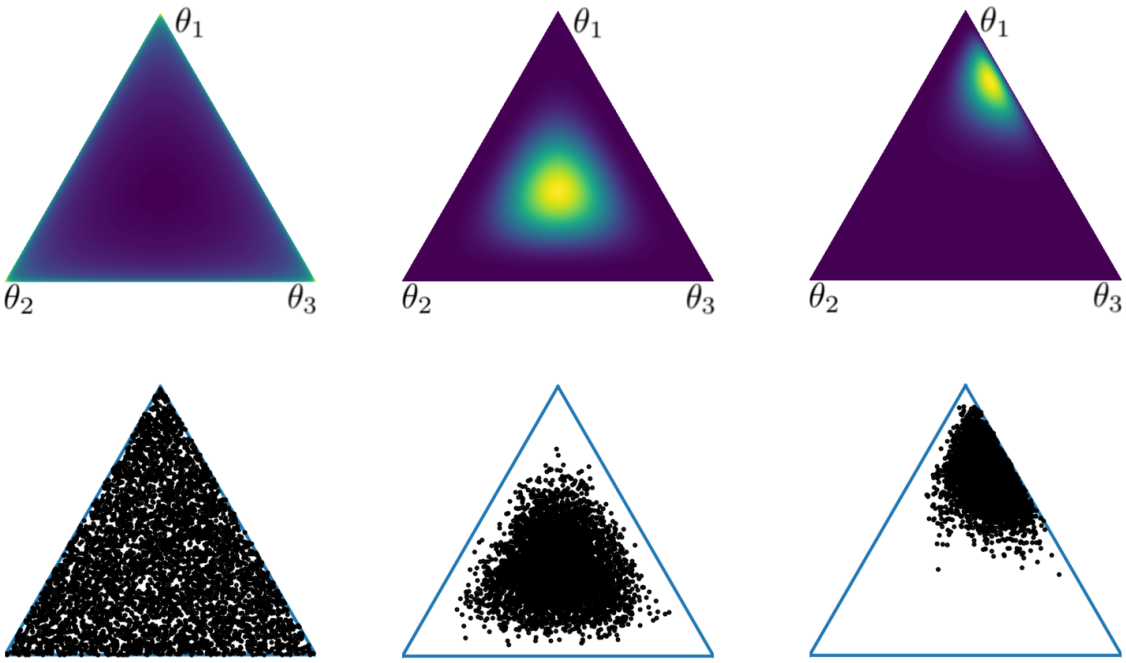


Figure 3: The K-dimensional Dirichlet distribution's domain is defined as a $K-1$ simplex due to the constraint $\sum_{i=1}^{K} \Theta_i = 1$. Thus, it is a triangle in this case of a 3-dimensional Dirichlet distribution.

Figure 4 shows examples of 3-dimensional Dirichlet distributions with different parameterizations. In the first image all topic distributions are equally likely which can be also seen from the sampling below. In the middle subfigure, topic distributions having topics of equal likelihood are primarily sampled. In the last image almost only topics distributions having high probability for topic 1 are being sampled.



(a) $\boldsymbol{\alpha} = (0.999, 0.999, 0.999)$ (b) $\boldsymbol{\alpha} = (5, 5, 5)$ (c) $\boldsymbol{\alpha} = (2, 5, 15)$

Figure 4: Plots of Dirichlet distributions with different parameterization $\boldsymbol{\alpha}$ (upper row). Lower row represents points sampled from the corresponding distribution.[6]

An important feature of Dirichlet distributions is that they are conjugate priors to multinomial distributions. This property is very convenient later on, when we will be estimating LDA's corpus level parameters.

Putting the generative process into mathematics

For the sake of simplicity, we make the following assumptions:

1. The number of topics K in the corpus is fixed. It follows that the dimensions of the topic-word assignments $\boldsymbol{z}$ is also fixed.
2. β is also regarded fixed and has to be estimated (see Paragraph *Estimation of latent variables* in this chapter). β's dimensionality is $K \times |V|$ and $\beta_{i,j}$ the probability of word j in topic i.
3. In Listing 2, we chose the length of a document N from a Poisson distribution. As N is independent of Θ and $\mathbf{z}$, we treat N as an ancillary variable and assume it to be constant.

The Dirichlet parameterization α and topic distributions β are corpus-level parameters that are sampled once in the process of generating the corpus. The topic distribution Θ_d is a document-level parameters and therefore sampled once for each document. Finally, the

[6] We used and modified the script provided by GitHub user tboggs. [1]

words' topic assignments $z_{d,n}$ and the words $w_{d,n}$ themselves are word-level parameters. They are sampled for each word in each document.

Given α and β, we can the describe the generative process of one document by the joint distribution of Θ, set of N topics assignments $\mathbf{z}$ and the set of N words in $\mathbf{w}$:

$$p(\Theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\Theta|\alpha) \prod_{n=1}^{N} p(z_n|\Theta)p(w_n|z_n, \beta). \tag{16}$$

Now that we have determined the joint probability, we are able run any query against the generative process. This means that we can calculate the probability of a single document given α and β by summing out z and integrating over Θ. The result is the marginal distribution of a single document

$$p(\mathbf{w}|\alpha, \beta) = \int p(\Theta|\alpha) \left(\prod_{n=1}^{N} \sum_{z_n} p(z_n|\Theta)p(w_n|z_n, \beta) \right) d\Theta, \tag{17}$$

from which we can sample to generate new documents. LDA makes the assumption that each document is generated independently from the other documents, which also reflects the process of how news articles are being published. Admittedly, outlets tend to pick up on each other's stories, but taking the amount of news being published into account, we can assert their independence with confidence. This means that we can easily extend the generation of a single document to an entire corpus:

$$p(\mathcal{D}|\alpha, \beta) = \prod_{d=1}^{M} \int p(\Theta_d|\alpha) \left(\prod_{n=1}^{N_d} \sum_{z_{d,n}} p(z_{d,n}|\Theta_d)p(w_{d,n}|z_{d,n}, \beta) \right) d\Theta_d. \tag{18}$$

Equation (18) can be visualized using plates notation, as shown in Figure 5. The random variables are represented by circles. The latent variables (β, α, Θ, z) are unshaded. Only $\mathbf{w}$ is observable and therefore shaded in gray. A random variable depends on another random variable if both are connected by an arrow. The arrow's tip points at the dependent variable.

The rectangles are the so called plates, which symbolize repetition of the random variables within the plate. The inner plate repeats z and w N times. w represents the random variable of one word in the document and z its topic assignment. As we have N words in the document both variables have to be repeated N times. Each document (in total M) has its own topic distribution according to Θ. The inner plate as well as Θ are therefore repeated M times by the outer plate.

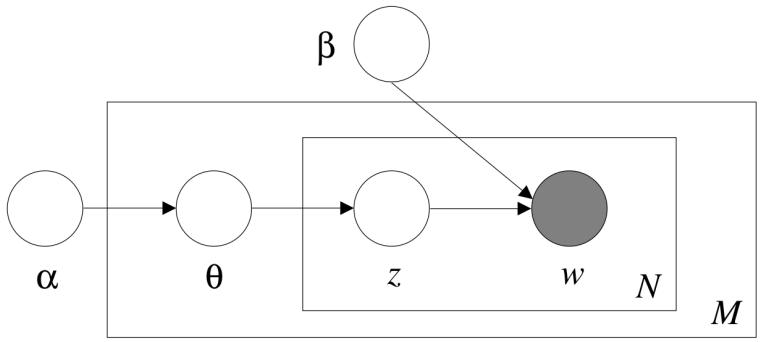


Figure 5: Graphical model representation of LDA in plates notation [4]

Inference

Inference is the prediction of a document's topic distribution Θ and its word-topic-assignments $\mathbf{z}$, given the document $\mathbf{w}$, $\boldsymbol{\alpha}$ and β. We can calculate the posterior distribution as follows:

$$p(\Theta, \mathbf{z}|\mathbf{w}, \boldsymbol{\alpha}, \beta) = \frac{p(\Theta, \mathbf{z}, \mathbf{w}|\boldsymbol{\alpha}, \beta)}{p(\mathbf{w}|\boldsymbol{\alpha}, \beta)} \tag{19}$$

The equation's numerator is the joint probability we defined in equation 16. The denominator is the marginal distribution of a document as defined in equation 17, which can be shown to be intractable to calculate [7].

Instead of calculating the posterior distribution exactly, we can apply variational inference as Blei shows in his paper [4] to get an approximate solution.

Estimation of latent variables

Given a corpus $\mathcal{D} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M\}$ (each $\mathbf{w}_i$ is a document), we want to find $\boldsymbol{\alpha}$ and β that maximize the log likelihood:

$$l(\boldsymbol{\alpha}, \beta) = \sum_{d=1}^{M} \log(p(\mathbf{w}_d|\boldsymbol{\alpha}, \beta)). \tag{20}$$

As already said for the inference, $p(\mathbf{w}_d|\boldsymbol{\alpha}, \beta)$ cannot be computed tractably. Approximation is performed using variational Expectation Maximization algorithm (see [4]).

Perplexity

Perplexity is a measure to evaluate topic models using a holdout set[7]. It monotonically decreases in the likelihood of the test data and is algebraically equivalent to the inverse of the geometric mean per-word likelihood [4]. Therefore, the lower the perplexity the higher the generalization performance.

[7] A subset of the data set that was not used for model training and can therefore be used to evaluate the model's generalization performance.

Perplexity is defined as follows:

$$perplexity(\mathcal{D}_{test}) = \exp\left(- \frac{\sum_{d=1}^{M} \log\, p(\mathbf{w}_d)}{\sum_{d=1}^{M} N_d} \right), \tag{21}$$

where $\mathcal{D}_{test}$ is the hold-out set, M is the number of documents in $\mathcal{D}_{test}$, N_d is the number of words in the d^{th} document and $\mathbf{w}_d$ is the set of words in the d^{th} document.

Next, we transform the definition of perplexity, such that it is easier to interpret. Looking at the denominator of the exponent in Equation (21), we define $T = \sum_{d=1}^{M} N_d$, where T is the total number of words in the corpus.

$$perplexity(\mathcal{D}_{test}) = \frac{1}{\exp\left(\frac{\sum_{d=1}^{M} \log\, p(\mathbf{w}_d)}{T} \right)} \tag{22}$$

$$= \frac{1}{\sqrt[T]{\exp(\sum_{d=1}^{M} \log\, p(\mathbf{w}_d))}} \tag{23}$$

$$= \frac{1}{\sqrt[T]{p(\mathbf{w}_1)p(\mathbf{w}_2)\dots p(\mathbf{w}_M)}} \tag{24}$$

By $p(\mathbf{w}_i)$ we actually mean $p(w_i|\boldsymbol{\alpha}, \beta)$, which is the likelihood of generating the document $\mathbf{w}_i$ given $\boldsymbol{\alpha}$ and β. We defined $p(\mathbf{w}_i|\boldsymbol{\alpha}, \beta)$ was defined as the marginal distribution in Equation (17). As mentioned earlier, this distribution cannot be computed tractably, which is why perplexity also needs to make use of approximation algorithms.

Looking at Equation (24), we now see that perplexity actually is the inverse of the geometric mean per-word likelihood.

2.3 k-nearest Neighbor for Categorical Distributions

k-nearest neighbor is an instance-based learner that memorizes all the training examples instead of deriving rules from the training set [16]. During prediction the algorithm calculates the distance between the instance that is to be predicted and all training instances. The k closest ones are used for prediction by applying majority or majority vote weighted by distance.

When comparing categorical distributions euclidean distance is not applicable, as it is only defined for the euclidean space and not for categorical distributions on a simplex.

Instead, we will use Kullback-Leibler divergence also known as relative entropy, which provides a dissimilarity measure between two distributions. Note that this is not a metric,

since it is not symmetric of its arguments.

Kullback-Leibler divergence

The Kullback-Leibler divergence is defined by

$$KL(p||q) = -\int p(x) \ln q(x) dx - \left(-\int p(x) \ln p(x) dx \right) \tag{25}$$

$$= -\int p(x) \ln \left(\frac{q(x)}{p(x)} \right) dx, \tag{26}$$

where $p(x)$ and $q(x)$ are the two distributions, whose dissimilarity to each other, we want to calculate.

Kullback-Leibler divergence can be easily interpreted by looking at Equation (25). The subtrahend is the entropy of the distribution p and therefore reflects the average amount of information of a sample x from that distribution. The minuend is the average amount of information needed if we used $q(x)$ to describe x. Thus, the subtraction describes the additional amount of information needed if x (coming from p) was actually described by distribution q.

One can show that the Kullback-Leibler divergence satisfies $KL(p||q) \geq 0$, while being only equal to zero iff $p(x) = q(x)$, making Kullback-Leibler divergence a valid measure of dissimilarity of distributions [2].

Jeffreys entropy

As Kullback-Leibler divergence is not symmetric with respect to its input variables, Jeffreys entropy was introduced, which is defined as follows:

$$JD(p||q) = \frac{1}{2} KL(p||q) + \frac{1}{2} KL(q||p). \tag{27}$$

All it does is, it calculates it calculates the mean of Kullback-Leibler divergence and Kullback-Leibler divergence with swapped arguments for two distributions p and q. Note that this still is not a metric, since the triangle inequality is not fulfilled. [6].

3 State of the Art and Related Work

In this section, we show the current state of research regarding financial trading systems based on news articles and how other researchers have solved the issue of classifying news articles in a variety of domains.

News article trading

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 [9], that a naïve Bayesian net 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 [12] 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.

Similarly, [15] compares different vectorization strategies of news articles (bag of words, noun phrases, named entities) as training input for SVM regression.

News article classification approaches

There has been quite a bit of research performed on news article classification (or in a broader context text classification) using topic models, as for instance done by [13]. They chose a "universal dataset" and used it to train a topic model. The topic model then predicted the topics of a labeled data set, which were then used to train multiple classifiers (e.g. Naïve Bayes, SVM). The model they trained was able to classify short and sparse texts.

Classification has also been applied to domain specific corpora. Sarioglu et al. [14] built a topic model from clinical reports in order to represent them in a more compact feature space than one built from a bag-of-words model. The representations were then used for classification of CT imaging reports using SVM. Their results showed that the topic model approach was competitive with a bag-of-words approach, while having the number of features greatly reduced.

4 Methodology

In this section, we describe the methodology applied to build a model that can predict the companies mentioned in news articles. In this set up, we only consider companies that are indexed by the S&P 500 index. Furthermore, each news article deals with exactly one of these companies. The reason for this limitation is that LDA training would become intractable, especially in combination with grid search.

The research project is split up into the phases:

- *Data Acquisition*
- *Preprocessing*
- *Modeling*
- *Evaluation*

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 use the following crawlers to populate our data base:

- News Article Crawler:
  Uses Rich Site Summy (RSS) Feeds to collect news articles
- Google News Crawler:
  Searches for the latest news articles by the company name and downloads these articles.
- Google Finance Crawler:
  Searches for the latest news articles by the companie's ticker symbols and downloads these articles.
- Company Description Crawler:
  This crawler downloads the company descriptions from several providers.

The news articles collected by the news article crawler are unlabeled, which leaves us with unsupervised learning procedures.

As a workaround, we make use of the URLs collected by the Google Finance Crawler and Google News Crawler. For each URL we know the corresponding symbol. If the URL is also present in the news article data, we are able to assign the correct stock symbol.

4.2 Preprocessing

All the crawlers provide 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 it 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[8] we traverse that tree until we reach the node that encapsulates the text that we want to obtain.

4.2.2 Text Cleaning

After having extracted the documents' text from HTML code, we apply multiple preprocessing steps to the corpus, such that it 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 list available. We use of NLTK's stopword list[9] and remove these words besides any punctuations from the tokens.

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 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 [5]. Python's

[8] `https://www.crummy.com/software/BeautifulSoup/`
[9] `http://www.nltk.org/`

NLTK library sticks to the part-of-speech tags defined by the Linguistic Data Consortium of the University of Pennsylvania[10].

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. Given 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.

4.3 News Article Classification

For the task of news article classification we propose a classification pipeline based on LDA and k-nearest neighbor and compare it to a more simple bag of words and k-nearest neighbor approach. Both approaches expect the text data to be in a vectorized format. We give a short introduction on how the two vectorizers work that we used.

Count vectorization

When applying count vectorization, we build a vocabulary from all tokens in the corpus and count the number of occurrences of every token for each document. The result can be represented as a matrix

$$B_{M,|V|} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,|V|} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,|V|} \\ \vdots & \vdots & \ddots & \vdots \\ b_{M,1} & b_{M,2} & \cdots & b_{M,|V|} \end{pmatrix}, \tag{28}$$

where M is the number of documents in the corpus, $|V|$ is the length of the vocabulary V and an element $b_{i,j}$ represents how often word j appears in document i. In general, count vectorization is equivalent to bag of words.

Term frequency-inverse document frequency vectorization

Term frequency-inverse document frequency (short: tf-idf) vectorization builds up on count vectorization by rescaling the frequencies of each token in a document:

[10] `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

$$tfidf(t,d) = tf(t,d) \cdot idf(t) \tag{29}$$

$$= tf(t,d) \cdot \log \frac{N}{df(t)}, \tag{30}$$

where tf is the frequency how often a given token t appeared in document d. tf is equivalent to the result of the count vectorizer. $idf(t)$ takes the logarithm of the total number of documents N in a corpus, over the document frequency df of token t. The document frequency is the number of documents that contain token t.

tf-idf is highest when t appears frequently in a small number of documents and lowest when t appears in all documents [11], which is exactly what we need for classifying business documents. Let's say we have a company description of *Apple Inc.* Tokens like *Mac*, *iPhone* or *Apple* will appear frequently in this document. The other company descriptions that are not about Apple will most likely not mention any of the tokens above. That is why above tokens will have a high weight when vectorizing apple documents.

The LDA and bag of words approach are described in the following two subsections. We will use count vectorizer and tf-idf vectorizer in each approach.

4.3.1 Latent Dirichlet Allocation approach

The idea is to train a LDA model based on the company descriptions, we retrieved for the S&P 500 indexed companies. As shown in Figure 6, all preprocessed company descriptions are vectorized first. The vector $\mathbf{d}$ contains the preprocessed tokens of the corpus (i.e. $\mathbf{d}_i$ is a row vector containing the preprocessed tokens of the i^{th} company description). An exemplary vector could be [*tesla*, *inc*, *car*, *manufacturer*, *electric*, *vehicle*, ...]. The company descriptions vector $\mathbf{d}$ is passed to the vectorizer which either returns the bag of words matrix (see Equation 28) or the tf-idf representation (see Equation 29).

LDA model training

The LDA algorithm takes the vectorized representation of the company descriptions as input to calculate the topic model. Besides the vectorized representation of the company descriptions, the algorithm also needs the number of topics to be specified as a hyper parameter. We set this hyper parameter to be the number of distinct companies in our company description data set. The intention is that we achieve a bias towards each topic exactly representing one company.

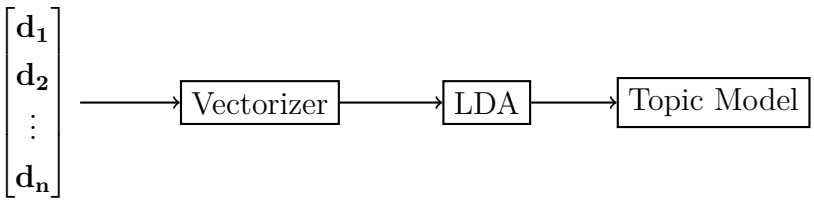


Figure 6: LDA model training

Having calculated the LDA topic model, we are now able to predict the topic distributions of the company descriptions and the news articles (see Figure 7). To do so, we first vectorize the company description corpus and the news article analogously to the vectorization step of the model training.

The vectorized documents are then passed to the topic model which estimates the topic distributions of each document.

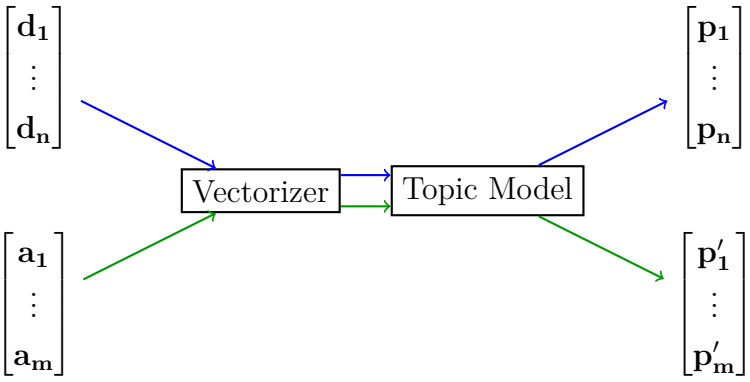


Figure 7: LDA company description and news article topic prediction

Each LDA prediction $\mathbf{p_i}$ is a categorical distribution over the topics. Therefore, the vector $\mathbf{p}$ has to fulfill the following constraints:

$$\mathbf{p_i}^T = \begin{pmatrix} p_{i1} \\ \vdots \\ p_{iK} \end{pmatrix}, \text{ where } \quad p_{ij} \geq 0 \quad \text{and} \quad \sum_{j=1}^{K} p_{ij} = 1 \text{ and } K \text{ no. of topics.}$$

k-nearest neighbor prediction

For every company we get multiple topic distributions (one for each company description). To predict a company mentioned in a news article, we take its topic distribution and calculate the distribution's similarity to the topic distribution of every company descriptions. Next, the company descriptions that are most similar to the news article are being picked for classification by performing majority vote or weighted majority

vote. The attentive reader might already have a clue, that this is k-nearest neighbor in a nutshell.

For measuring similarity Kullback-Leibler divergence can be applied as similarity measurement $d(p'_i, p_j)$ of categorical distributions p'_i and p_j.

The similarity of news articles and company descriptions is displayed in Table 2. For example, given news article 1 the LDA algorithm returned the topic distribution p'_1. We calculate the similarity of p'_1 to p_i for i going from 1 to n. Then we pick the k closest distributions and perform voting on their corresponding symbols for classification.

		company descriptions			
		p_1	p_2	$\dots$	p_n
news articles	p'_1	$d(p'_1, p_1)$	$d(p'_1, p_2)$		$d(p'_1, p_n)$
	p'_2	$d(p'_2, p_1)$	$d(p'_2, p_2)$		$d(p'_2, p_n)$
	$\vdots$				
	p'_m	$d(p'_m, p_1)$	$d(p'_m, p_2)$		$d(p'_m, p_n)$

Table 2: Distance measurements between news articles and company descriptions

Note that Kullback-Leibler divergence is not symmetric with respect to its input parameters and we might want to use Jeffreys entropy instead.

4.3.2 Bag of Words Approach

In this approach we skip the LDA part and try to classify news articles directly by finding the closest company descriptions (i.e. closest bag of words or tf-idf representation).

As described in Figure 8, the vectorizer vectorizes the company descriptions $\mathbf{d}$ and news articles $\mathbf{a}$ and returns the corresponding vectorized representation.

Since the bag of words model represents absolute counts of word occurrences and not all documents have the same length, they have to be normalized. The same holds for the tf-idf representation, which is, as mentioned before, an extension to the bag of words model. Normalization is performed by dividing each row element of the matrix shown in Equation (28) by the sum of all elements of its row. This makes sure that every document's vectorized representation is a categorical distribution over the vocabulary of the corpus.

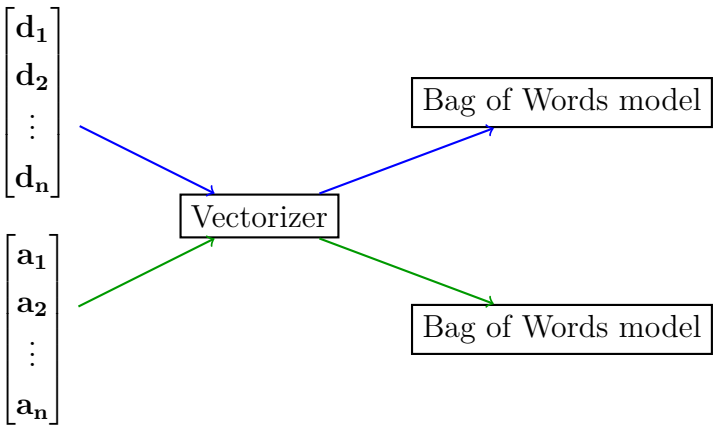


Figure 8: Bag of words model training

Now that every document has its distribution over the vocabulary, we calculate the distance of one news article to all company descriptions. Again, we apply one of the similarity measures that were introduced in section 2.3. The k closest company descriptions are used for classification (e.g. majority vote). Like in the preceding approach, this step reduces to k-nearest neighbor classification.

5 Implementation

This section deals with the realization of the approaches and methods suggested in Section 4. We will describe in detail how the news article mining has been worked out and how the articles are being preprocessed using natural language processing tools.

In Section 5.3.1, we use the preprocessed articles to build a topic model using LDA. Later on we will compare this model to the trivial bag of words model that is being implemented in Section 5.3.2.

5.1 News Article Mining

As introduced in section 4.1, we implemented a crawler that uses RSS feeds provided by news outlets to mine news articles. When the crawler retrieved a new article, it will be added to a Comma Separated Values (CSV) file by dumping the request's raw HTML response (including the article's URL, timestamp etc.) into the file. Later on, the preprocessing pipeline reads in the CSV file, extracts the articles' 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 article downloads based on the articles' feed origin. For each feed we can individually define a download pattern, e.g. every 10 minutes 7 times in total.
- **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 pre-defined 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[11].

As shown in Figure 9, 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 predefined duration.

Requester is a wrapper class for the Requester 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 jobs 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 articles to disk in this description.

Scheduler schedules article downloads using the *APScheduler*[12] library. It also keeps track of the articles, that are already known to the scheduler, such that only unknown articles 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 articles that are being collected during these runs, are flagged as known (*Scheduler:known_urls*). RSS feeds do always show the last n articles and not only the new ones. Since we are solely interested in articles 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***[13] 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 article download jobs that are due to a queue. The ***Download-Worker*** reads from that queue and downloads the respective articles. All the downloaded articles are then passed to the *Writer*'s queue in order to write them to disk.

[11] `https://github.com/requests/requests`
[12] `https://apscheduler.readthedocs.io/en/latest/`
[13] Admittedly, the class' name might be confusing and should be rather named FeedCrawler

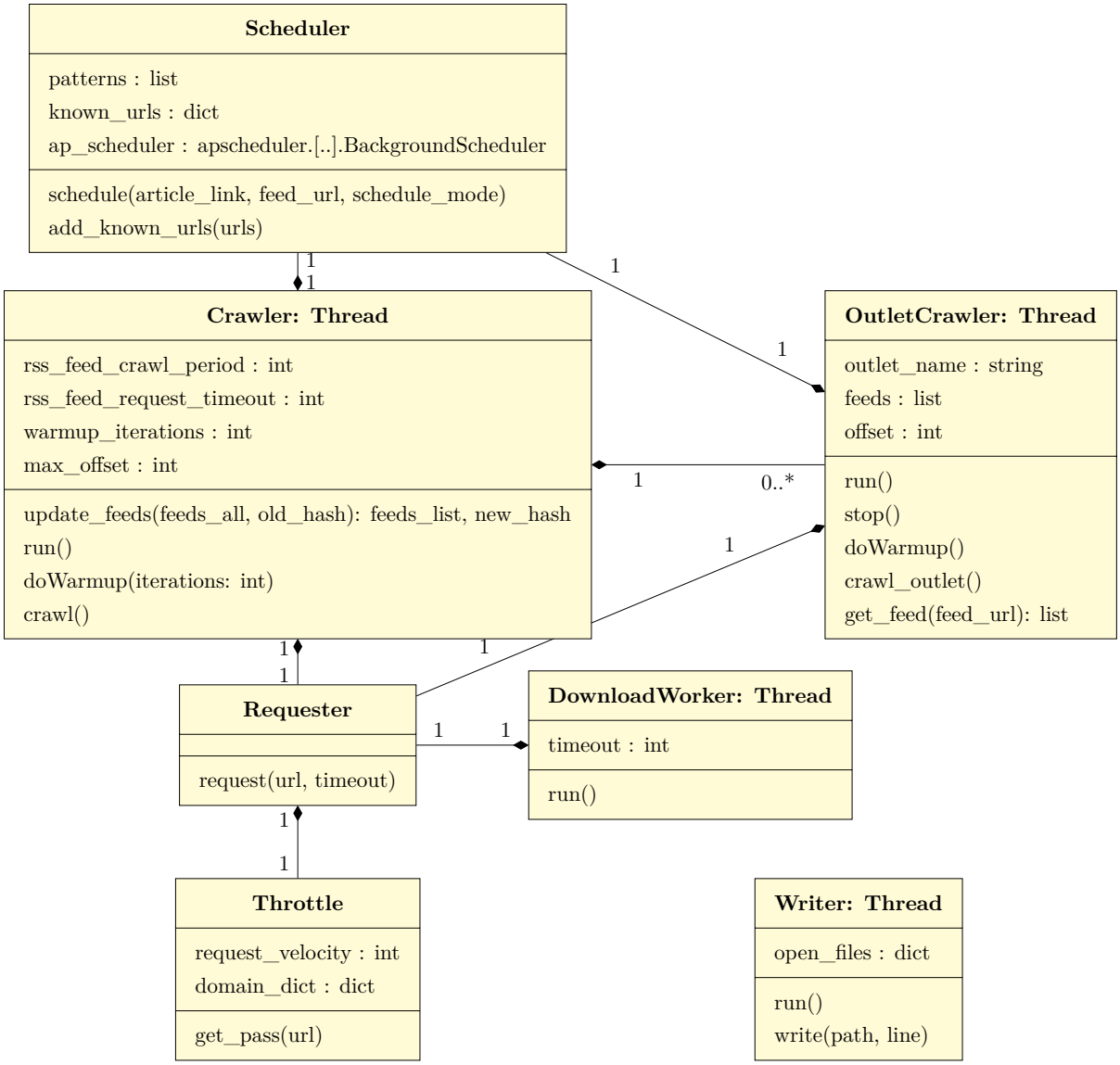


Figure 9: Class diagram[14]of the crawler

Figure 10 shows how the feeds are being retrieved and how the news article 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 article URLs to the *Scheduler*. The *Scheduler* then schedules the article download jobs for articles that have not been known so far.

[14]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`.

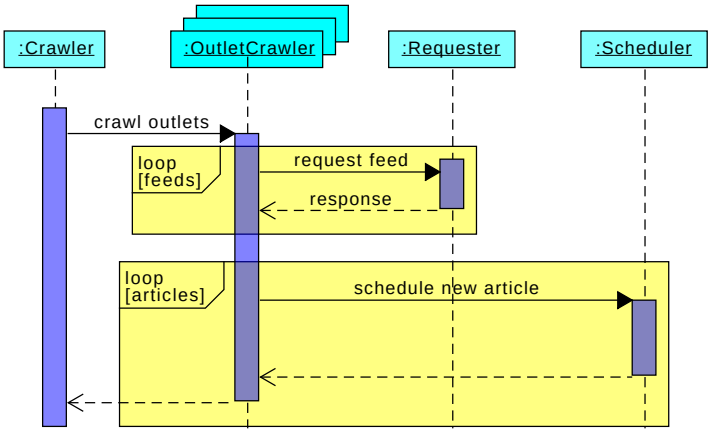


Figure 10: Sequence diagram of feed crawling run

When a news article download job is due, the *APScheduler* calls a callback function that was set when the article 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 11).

On the other end of the queue, there are the *DownloadWorker*s waiting for due article 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 article using the *Requester*. After the article download is finished, the *DownloadWorker* puts the article into the *Writer Queue*.

The *Writer* instance writes all the article downloads to disk.

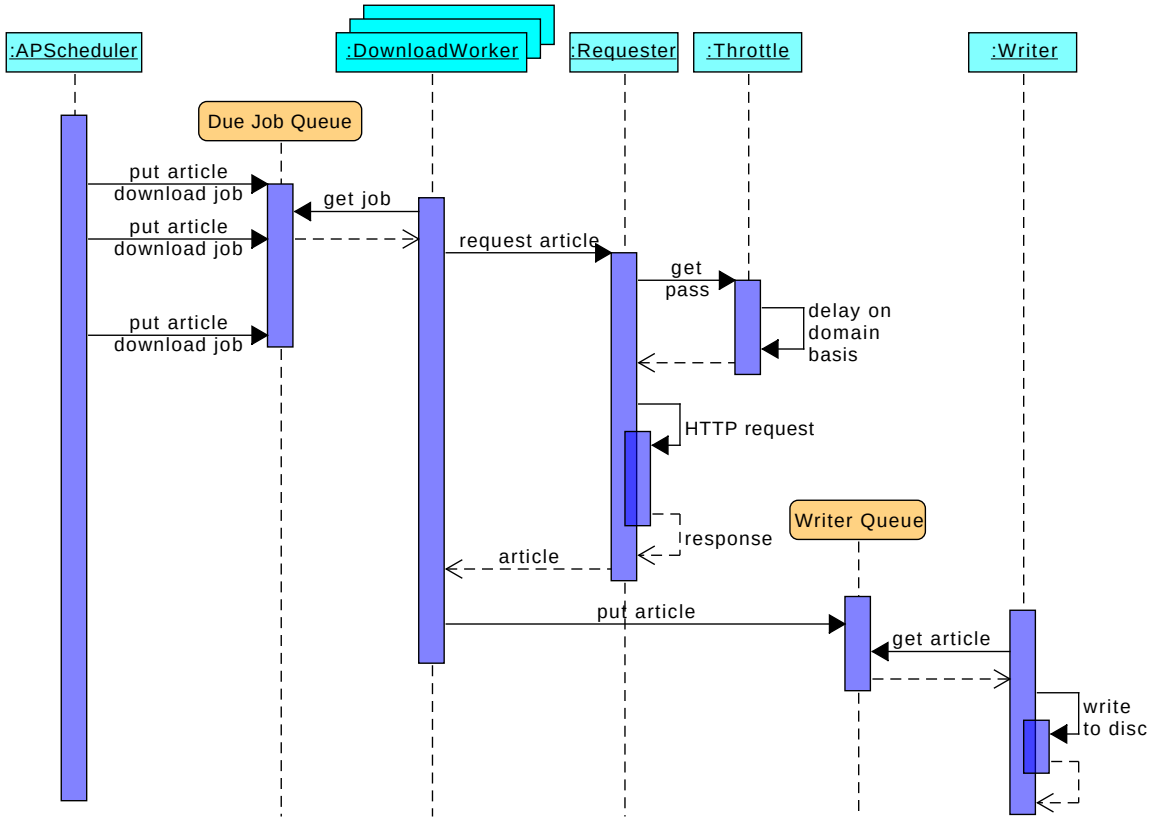


Figure 11: Sequence diagram of article retrieval

5.1.2 Storage

The *Writer* instance writes the downloaded articles 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 article was downloaded is being stored in the first column. *Feed_url* stores the URL of the feed where the article was found in and *link_to_article* stores the URL where the article was published on the outlet's website. The raw HTML content of the article download is being dumped into the *html_download* column.

5.2 Preprocessing

Each crawler dumps plain HTML strings into CSV files. Each row of the files contains a single download enriched with further attributes (columns). These files are being preprocessed, such that they are 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 (e.g. news article texts) 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.

As shown in Table 3, all crawlers need the extraction stage. The cleaning stage is only needed for the outputs of the News Article Crawler and the Company Description Crawlers. Google News Crawler and Google Finance Crawler only provide URLs to label news articles by the company that was mentioned in the respective news article. That's why we do not need any further preprocessing after having located the URLs in the extraction stage.

Crawler	**Extraction**	**Cleaning**
News Article Crawler	✓	✓
Company Description Crawler	✓	✓
Google News Crawler	✓	✗
Google Finance Crawler	✓	✗

Table 3: Preprocessing steps for each crawler

5.2.1 Extraction Stage

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

Since every outlet or company description 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:

```
1 Given:
2   entry points list
3   unwanted html tags list ([] allowed)
4   tags of interest list ([] allowed)
5 1)
6   a) find path in HTML tree that contains all entry points
7   b) keep HTML code at the node of lowest entry point
8 2) delete all HTML tags that are element of the unwanted html tags
      list
9 3) only keep HTML tags that are element of the tags of interest list
10 4) filter and return remaining text (BeautifulSoup deletes tags
      automatically)
```

Listing 3: 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 4: Representative parameterization of extraction algorithm

5.2.2 Cleaning Stage

After the preceding stage has extracted the texts, we perform now the cleaning steps defined in Section 4.2.2, as shown in Listing 5.

The method expects a DataFrame[15] 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. All remaining tokens are then 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'))
```

[15] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

```
13      df.loc[:,"tokens"] = df["tokens"].apply(lambda words: [i for i in
        words if i not in stopwords_set])
14
15      # tagging and lemmatization
16      wnl = nltk.stem.WordNetLemmatizer()
17      # if wnl.lemmatize(.) is only provided a token without POS it
       ALWAYS lemmatizes it as a noun!
18      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)])
19
20      # remove punctuation
21      df.loc[:,"tokens"] = df["tokens"].apply(lambda tokens: [token for
        token in tokens if token not in string.punctuation])
22      # remove numbers
23      df.loc[:,"tokens"] = df["tokens"].apply(lambda tokens: [token for
        token in tokens if not any(c.isdigit() for c in token)])
24
25      return df
```

Listing 5: Text cleaning algorithm

In practice, we often deal with huge data sets, 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 one. These preprocessing jobs are then passed to an ipyparallel computation cluster[16]. The preprocessing results are aggregated after all preprocessing jobs are done.

5.3 Company Classification of News Articles

In this section, we describe the implementation of the methodology explained in preceding Section 4.3.

5.3.1 Latent Dirichlet Allocation Approach

Using the preprocessed tokens, we first vectorize the documents using sklearn's vectorizers (CountVectorizer, TfidfVectorizer), as shown in Listing 6. The vectorizers are paramterized with `max_df`, which sets the upper boundary for document frequency of tokens and `min_df`, which sets the lower boundary for document frequency of tokens. If `min_df` is

[16] `https://ipyparallel.readthedocs.io/en/latest/`

set to an integer value greater than one, then we discard all tokens that appear in an absolute number of documents less than the given value.

```
text_train = preprocessed_company_descriptions["tokens"].apply(lambda
    l: ' '.join(l))
vectorizer =  CountVectorizer(max_df=max_df, min_df=min_df )
# Instead of a count vectorizer we also use
# tf-idf vectorizer by instantiating one as follows:
# vectorizer = TfidfVectorizer(max_df=max_df, min_df=min_df)
company_descriptions_vectorized = vectorizer.fit_transform(text_train
    )
```

Listing 6: Vectorization of company descriptions

Next, the vectorized company descriptions are used to train the LDA model, which is shown in Listing 7. We define the number of topics (`n_components`) of the model to match the number of companies in our corpus. `lda_n_jobs` is the number of CPUs that we would like to use. Setting this to -1 enbables the algorithm to use all CPUs.

LDA's `fit_transform` method fits the model (member variable of LDA) and also returns the topic prediction of each document.

```
number_topics = int(len(preprocessed_company_descriptions.groupby("
    symbol")))
lda_n_jobs = -1 # use all cores
lda = LatentDirichletAllocation(n_components=number_topics,
    learning_method="batch", max_iter=15, random_state=0)
company_description_topics = lda.fit_transform(
    company_descriptions_vectorized)
```

Listing 7: LDA modeling

For each company description, we know its company and therefore can use the LDA topic prediction and the company labels to build a K-nearest Neighbor classifier, as shown in Listing 8.

To get a better prediction performance, we normalize the topic prediction of each company description such that we get a topic distribution (`company_descriptions_topics`). The topic distributions are passed to the `KNeighborsClassifier`'s `fit` method together with the respective company symbols.

For similarity measurement we use Kullback-Leibler divergence (see Section 2.3), which is already implemented in Python's scipy library[17].

The number of neighbors that are considered for classification is set by `n_neighbors`. There exist two different voting strategies. Firstly, there is majority vote, which takes

[17] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html`

the neighbors and picks the class that is present the most in the neighbors set. Secondly, there is weighted majority vote in which each neighbor is weighted by the distance it has to the sample we want to predict. On these weighted measurements we perform majority vote for classification. The voting scheme can be set by the variable `weights`.

Finding the k-nearest neighbors is a computationally complex task and there exist different data structures like K-D trees and ball trees that simplify this task. However since the brute force method (comparing the sample to all labeled data points) is easier to interpret, we will stick to that method throughout this paper. The algorithm is to be specified in the `algorithm` parameter.

```
company_descriptions_topics = (company_descriptions_topics.T/
    company_descriptions_topics.sum(axis=1)).T

knn_clf = neighbors.KNeighborsClassifier(n_neighbors=20, weights="
    distance", algorithm="brute", metric=scipy.stats.entropy, n_jobs
    =-1)
knn_clf.fit(company_descriptions_topics, company_descritions_symbols)
```

Listing 8: K-nearest neighbor modeling

Having built the k-nearest neighbor model, we can now predict the company a news article is dealing with. We pick a news article, vectorize it and calculate its topic distribution. We then let the k-nearest neighbor model calculate the k closest topic distributions and predict the company.

```
knn_prediction = knn_clf.predict(labeled_articles_topics)
```

Listing 9: Prediction

5.3.2 Bag of Words Approach

Like in the LDA approach, we first vectorize the company descriptions either using `CountVectorizer` or `TfidfVectorizer`.

After that we normalize the bag of words representation such that each company descriptions' bag of words representation is a distribution of the vocabulary of the corpus. Since Kullback-Leibler divergence cannot handle zero values, we replace them by a value close to zero but not equal to zero.

```
# count vectorization
tokens = preprocessed_company_descriptions["tokens"].apply(lambda l:
    ' '.join(l))
vectorizer = CountVectorizer(max_df=max_df, min_df=min_df )
# Instead of a count vectorizer we also use
# tf-idf vectorizer by instantiating one as follows:
```

```
# vectorizer = TfidfVectorizer(max_df=max_df, min_df=min_df )
company_descriptions_vectorized = vectorizer.fit_transform(tokens)
company_descriptions_vectorized = company_descriptions_vectorized.
    toarray()

#replace 0 elements
company_descriptions_vectorized = company_descriptions_vectorized.
    astype(float)
company_descriptions_vectorized[company_descriptions_vectorized == 0]
     = 10**-10
# normalization
company_descriptions_vectorized = (company_descriptions_vectorized.T/
  company_descriptions_vectorized.sum(axis=1)).T
```

Listing 10: Vectorization and normalization of company descriptions

Next, we build the k-nearest neighbor model from the company description vocabulary distributions and the labels. Again, since we are dealing with categorical distributions (distributions over vocabulary of the corpus) we use Kullback-Leibler divergence for similarity measurement.

```
knn_clf = neighbors.KNeighborsClassifier(n_neighbors=20, weights="
    distance", algorithm="brute", metric=scipy.stats.entropy, n_jobs
    =-1)
knn_clf.fit(X=company_descriptions_vectorized, y=
    preprocessed_company_descriptions["symbol"])
```

Listing 11: K-nearest neighbor modeling

After having built the model, we can predict the company a news article deals with. We vectorize the article and pass it to the `fit` method.

```
prediction = knn_clf.predict(labeled_articles_vectorized)
```

Listing 12: Prediction of news articles vectorized by LDA

6 Results and Evaluation

This section deals with the evaluation of the model performance. We shortly introduce the data sets and then show the performance measurements (accuracy, recall, precision) for each model.

6.1 Data Sets Exploration

Before going into the evaluation of the trained model, we introduce each data set's properties such that the reader gets used to them and has a better understanding on how to interpret the results.

News article data set

The news articles have been crawled for about half a year and their corpus has the following properties:

Started:	2017-07-10 17:57:21
Stopped:	2018-03-12 00:59:49
Data set size:	11,002,838
Unique articles:	2,168,404
No. of outlets:	94

The number of news articles published each day varies depending on whether it is a weekday or weekend. As expected, news article counts drop during weekends, leading to the periodicity shown in Figure 12.

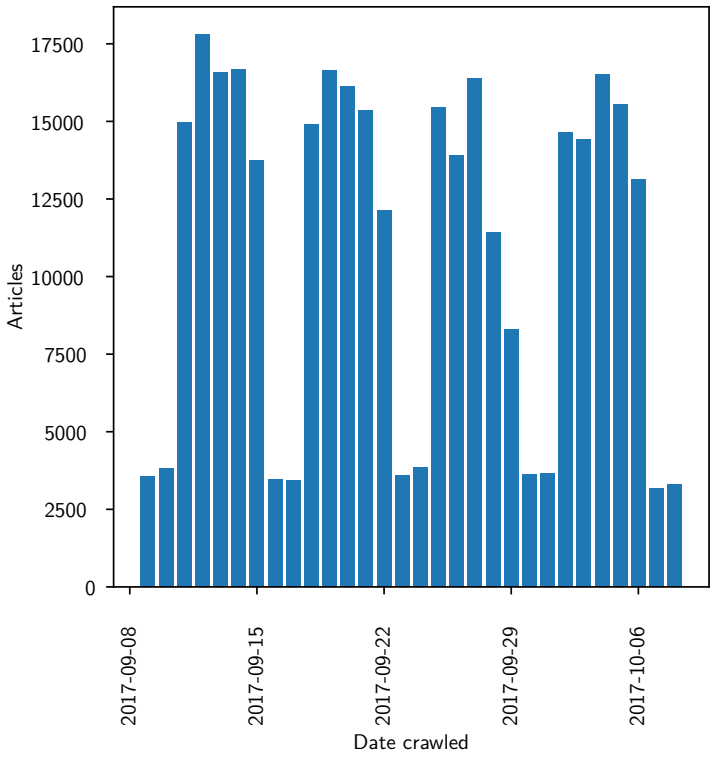
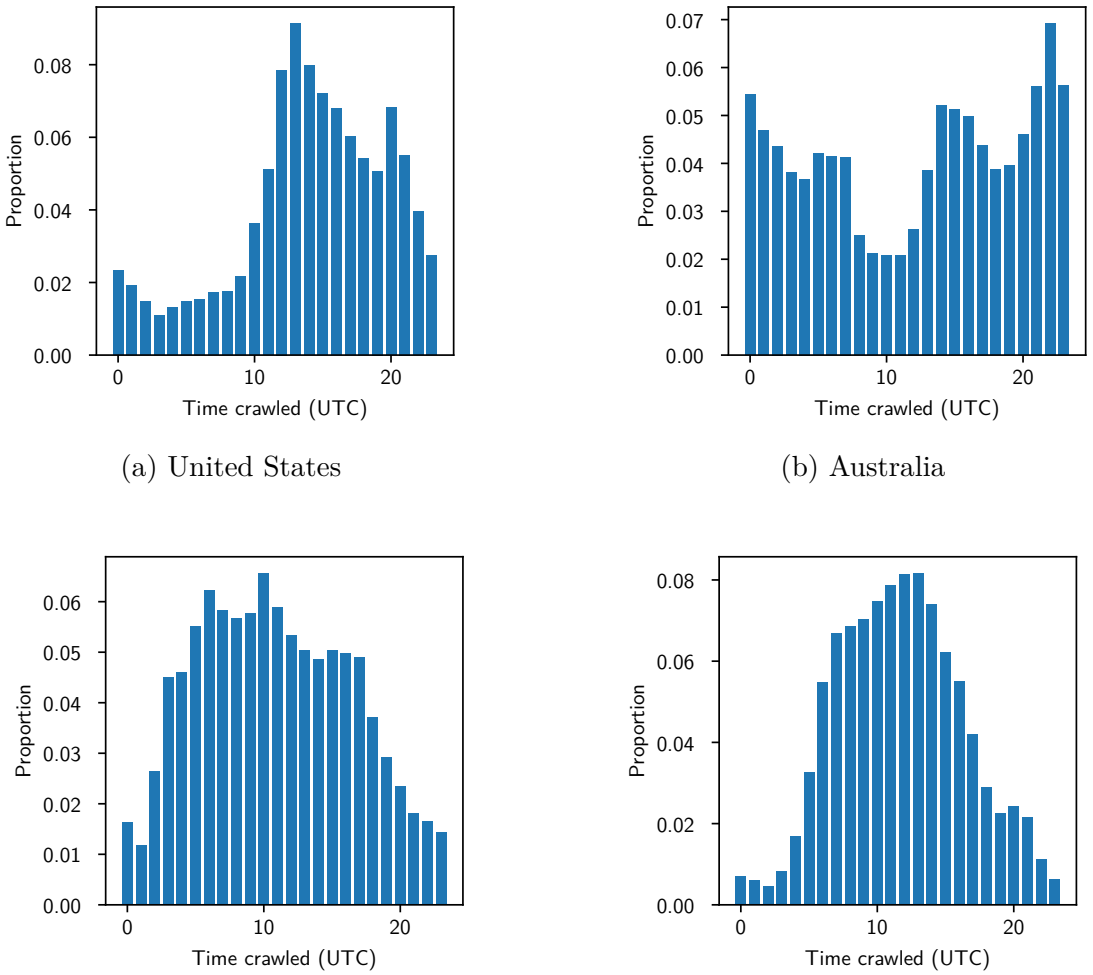


Figure 12: Periodicity of per day news article counts

Next, we look at when articles are being published in different countries. During night time in a country, there is a big drop in article publications at that time in that country, as shown in Figure 13. We can therefore draw the conclusion, that if we want to trade on the basis of news article sentiments, we will have to cover outlets all around the world. When Volkswagen has a new scandal it is most likely that a German outlet will pick up on that story. Australia, which is inactive in terms of publishing news articles at German day time, is probably contributing to the story in the morning (Australian time).

Also notable, Germans like to stick to their 9-to-5 jobs. The articles that are published early in the morning (5AM to 7AM) have been often written the day before and their publication has been scheduled for the morning. If we reduced the numbers of articles published in the early morning (5AM - 7 AM), we would exactly end up with the usual 9-to-5 job. In contrast to this, the shapes of the other countries are a little broader.



(a) United States

(b) Australia

(c) India

(d) Germany

Figure 13: Histograms of times of day articles are published (by country)

The news articles distribution over the outlets is shown in Figure 14. The two press release outlets, which spread companies' press releases, are the most publishing outlets. Nevertheless, usual news articles make up the majority of publications compared to press releases. The majority of news articles and press releases are published by a quarter of all outlets.

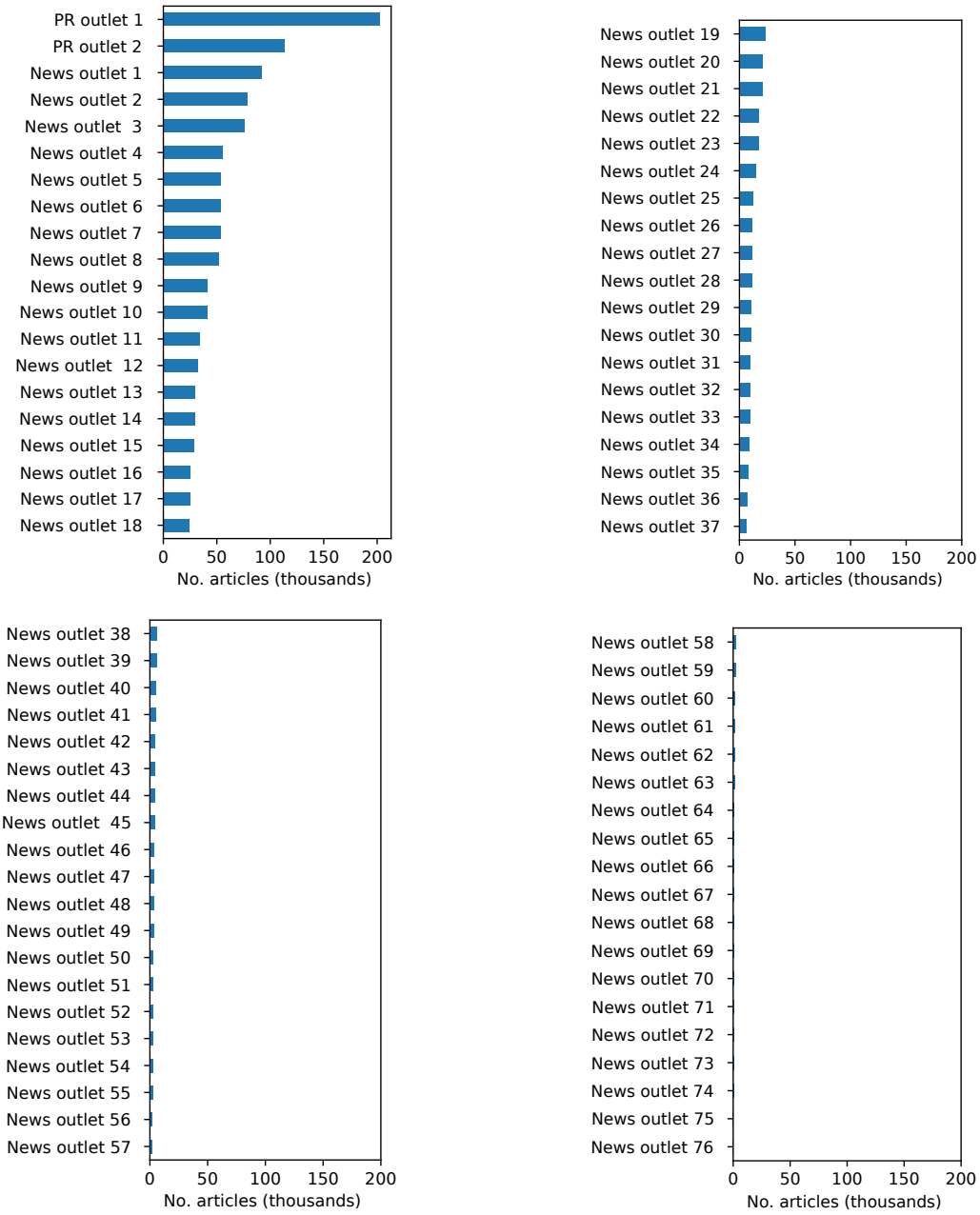


Figure 14: Total number of English news articles published by each outlet(in thousands)

Google Finance and Google News data sets

As shown in Table 4, we crawled 3.3 M URLs of news articles via the Google News crawler and 2 M via the Google Finance crawler from December, 2017 to February, 2018. The number of URLs of articles that have been crawled by both crawlers amounts to a total of 500 k.

For each of the URLs that are present in both corpora, we check whether their articles have been also crawled by the news article crawler. From these news articles we delete those that do not belong to a company of the S&P 500 index reducing our labeled news

article corpus to a size of 572 articles. Admittedly, this is a really sparse data set. However, we only need it to test our model as training is performed by the company description corpus.

Since we cannot be sure, that Google classifies all news articles correctly, we manually checked the majority of them and did not find any wrong labeling. That is why, we regard the Google labeling as correct for the subsequent model performance evaluation.

No. Google News URLs:	3,310,706
No. Google Finance URLs:	2,038,685
No. URLs present in both sets:	515,001
No. labeled news articles:	2,125
No. of labeled news articles of S&P 500 companies:	572

Table 4: Key figures about URL set crawled by Google Finance and Google News crawler

Next, we group the labeled news articles by outlets and determine the number of articles of each outlet. Looking at Table 5, it turns out that 315 of 572 news articles are press releases.

Outlet	**No. labeled articles**
PR outlet 2	315
News outlet 10	71
News outlet 32	63
News outlet 7	30
News outlet 14	26
News outlet 11	16
News outlet 18	12
News outlet 3	8
News outlet 9	5
News outlet 35	5
News outlet 70	4
News outlet 53	4
News outlet 44	3
News outlet 51	2
News outlet 46	2
News outlet 40	2
News outlet 41	1
News outlet 38	1
News outlet 24	1
News outlet 17	1

Table 5: Labeled article counts by outlet

Company description data set

We crawled the company descriptions of 5,092 companies traded on NYSE and NASDAQ from the 8 providers listed in Table 6. For each provider we received between 3700 and 5000 descriptions. The median description length varies greatly. Reuters provides by far the longest company descriptions (median character length 3145), whereas Zacks provides the shortest ones with a median length of 466 characters.

As we only want to predict S&P 500 indexed companies, we discard all company descriptions, that do not belong this index.

Provider	**No. descr.**	**Med. length**	**S&P**
Provider 1	4960	3145	488
Provider 2	4536	466	488
Provider 3	5015	2015	488
Provider 4	4298	641	485
Provider 5	4951	654	488
Provider 6	3714	976	482
Provider 7	4982	535	488
Provider 8	4779	1006	488

Table 6: Key figures regarding company descriptions downloaded by each company description provider

Figure 15 shows for each company description provider a histogram of their company descriptions' character length. The x-axis of each histogram is scaled by the maximum character length of the corresponding outlet. Therefore the histograms axes are not equally scaled which enables us to better recognize the structure of each histogram.

Each histogram has exactly 100 bins. The bin width is the maximum character length of a company description divided by the number of bins, leading to different bin widths for each histogram.

The shortest company descriptions often do not contain any information about the company and instead contain error messages like "Company XYZ not found". If these short company descriptions actually contain information about the company, it is often only one general sentence like "Company XYZ is a automobile manufacturer". These descriptions greatly reduce the models performance, which is why we decided to set a minimum threshold for company description character length at 300 characters. The company descriptions that are affected by this threshold are marked red in the histograms.

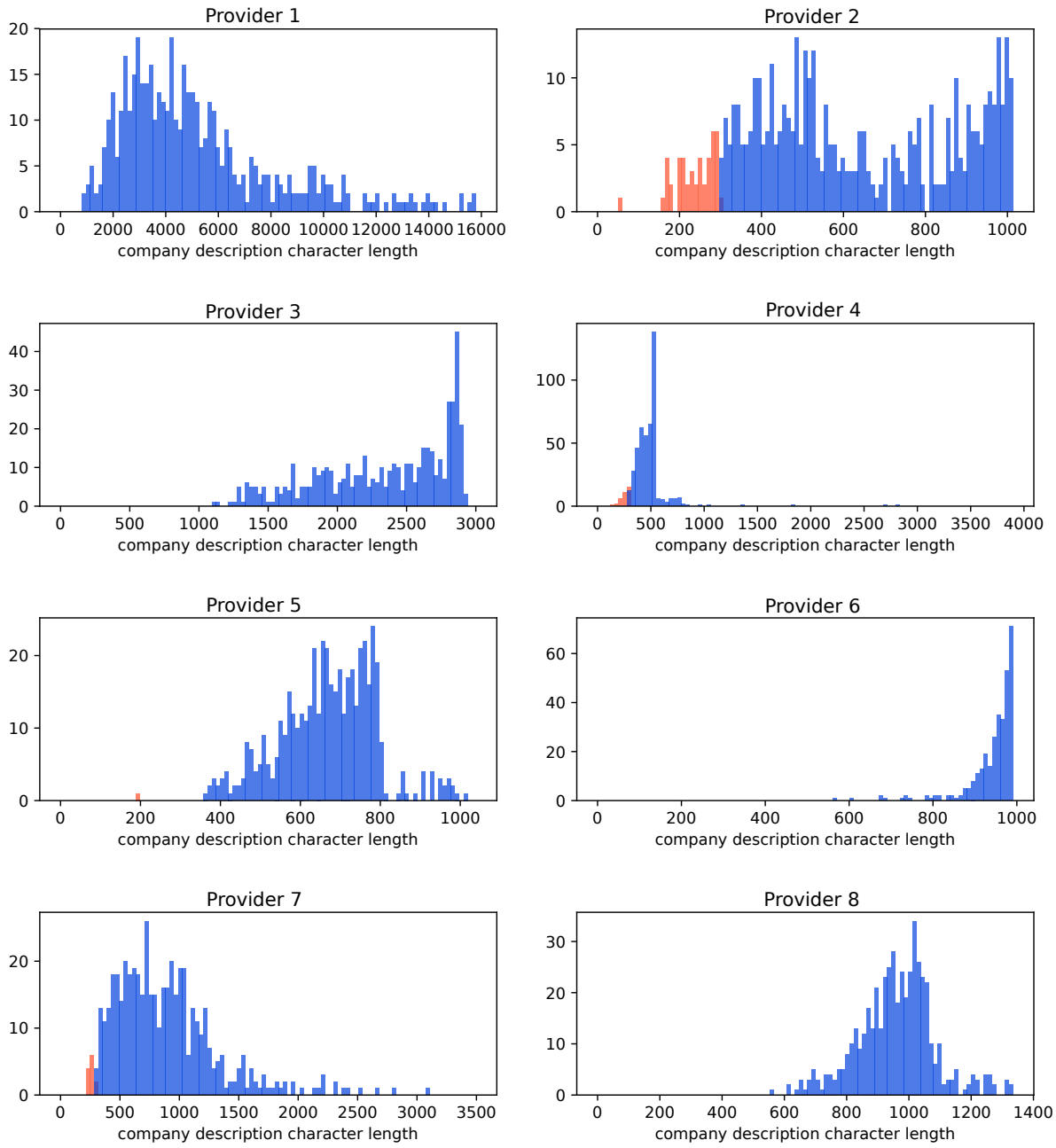


Figure 15: Histograms of company descriptions' character length (by provider). Red bars show company descriptions with a character length less than 300 characters. We remove these company descriptions from the corpus, as they caused drops in model performance.

Having deleted all articles with character length less than 300, we group the company descriptions corpus by company and count how many company description we get for each symbol (i.e. for each company). Then we plot the counts as a histogram, as shown in Figure 16.

Since we want every company to be similarly represented by the model, we discard all company descriptions that belong to companies that have less than seven company de-

scriptions. With regard to Figure 16, we therefore only keep the most right two bars for model training.

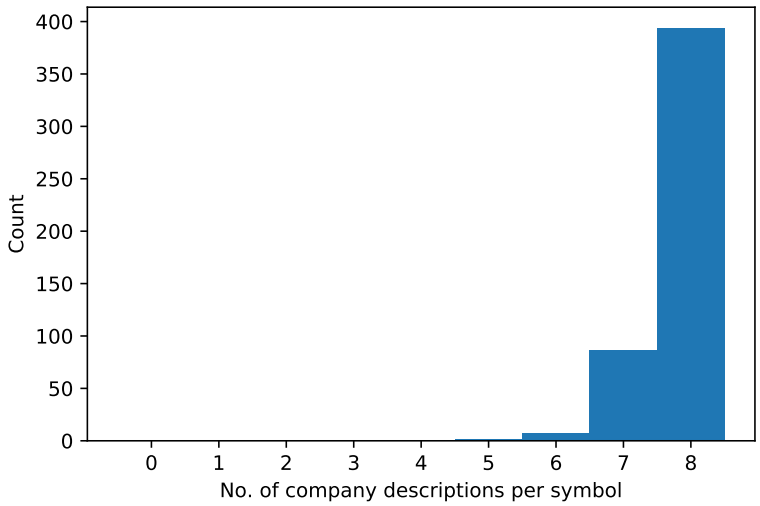


Figure 16: Histogram of number of company descriptions per company

Finally, we end up with a corpus, that has at least seven company descriptions for each company while covering 480 out of 505 S&P 500 index companies. This corpus was used to train the subsequent LDA models and bag-of-words models.

6.2 Performance Measures

We perform an extrinsic evaluation by considering the LDA and bag-of-words approach as black boxes and check the overall performance using the labeled news article set consisting of 572 labeled news articles.

Note that, neither the outlets nor the symbols are balanced in the labeled news article set and that the data set is really sparse (some symbols are not even present). That is why, we cannot have too much confidence in our measures. After all, it reflects only the performance of the model as if it was deployed, while assuming that future data is similarly distributed in terms of symbols and outlets as our labeled news articles corpus.

On top of that, questions regarding the likelihood that a news article about *Apple Inc.* published by *The New York Times* will be correctly matched to *Apple Inc.*, cannot be answered by the performance measures directly.

Nevertheless, we think that the performance measures will give an initial impression, whether it is reasonable to continue improving the approaches and assessing their performance in further research.

In binary classification, when predicting an instance we get either one of the four cases, shown in Table 7. This is a representation of the confusion matrix, where TP are all positive instances that have been correctly predicted, TN are all 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 ones and finally FN, which are the the instances that have been predicted as negative but are actually positive.

		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 7: Confusion matrix for binary classification

We evaluate the model's performance by *accuracy*, *recall* and *precision*, which build upon the confusion matrix.

The accuracy of a model is defined by

$$\text{acc}_{\text{bin}}(\text{model}) = \frac{TP + TN}{TP + FP + FN + TN}. \tag{31}$$

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

We use the recall of a model

$$\text{rec}_{\text{bin}}(\text{model}) = \frac{TP}{TP + FN} \tag{32}$$

to determine the proportion of true positives over all instances that actually positive.

Finally, we use the precision

$$\text{prec}_{\text{bin}}(\text{model}) = \frac{TP}{TP + FP} \tag{33}$$

to determine the proportion of true positives over all instances that have been predicted

to be positive.

The measures above are defined for binary classification problems. As the news article classification problem at hand is a multi class classification problem, we need to adapt these measures.

A solution is to extend the table by all possible classes and thereby getting a confusion matrix having dimensions |S&P500 companies| × |S&P500 companies|. Now we calculate the accuracy, recall and precision of each class separately, making it |S&P500 companies| many binary classification problems. We then average the calculations of each measure to get an overall performance.

Let the data set instances be denoted by

$$X = \{x_1, x_2, \ldots, x_N\} \tag{34}$$

and the set of possible class labels be

$$L = \{l_1, l_2, \ldots, l_M\}. \tag{35}$$

The true class of $x \in X$ is given by

$$y : X \mapsto L \tag{36}$$

and its prediction by

$$h : X \mapsto L. \tag{37}$$

The accuracy measure then can be express by

$$\text{acc}_{\text{mult}}(\text{model}) = \frac{|\{x \in X \,|\, y(x) = h(x)\}|}{N}. \tag{38}$$

All it does is, it takes the number of instances that have been correctly classified and divides them by the sum of all elements in the confusion matrix / the number of all instances in the data set.

We define the set of instances having predicted class l_j by

$$X^j = \{x \in X \,|\, h(x) = l_j\}, \tag{39}$$

and the set of instances having actual class l_j by

$$X'^j = \{x \in X \,|\, y(x) = l_j\}, \tag{40}$$

The precision measure then can be calculated by

$$\text{prec}_{\text{mult}}(\text{model}) = \frac{1}{M}\sum_{j=1}^{M}\frac{|\{x \in X^j \,|\, y(x) = h(x)\}|}{|X^j|}. \tag{41}$$

The outer summation sums over all class labels in L. For each class label we calculate its precision by dividing the number of instances that are true positives and have predicted class l_j by the total number of instances having predicted class l_j. The overall result is then divided by the number of classes to get the average precision.

The recall measure is given by

$$\text{rec}_{\text{mult}}(\text{model}) = \frac{1}{M}\sum_{j=1}^{M}\frac{|\{x \in X'^j \,|\, y(x) = h(x)\}|}{|X'^j|}. \tag{42}$$

The outer summation sums over all class labels in L. For each class label we calculate its recall by dividing the number of instances that are true positives and have actual class l_j by the total number of instances having actual class l_j. The overall result is then divided by the number of classes to get the average precision.

6.3 Latent Dirichlet Allocation Approach

In this section we evaluate extrinsically the performance of the LDA approach with count vectorization and with tf-idf vectorization. Both cases are identically parameterized, as shown Table 8. The hyperparameters have been chosen by the author's intuition and do not reflect a well-configured set up. We will see that even with little domain knowledge, we can already achieve good results. Full parameter tuning using e.g. grid search is left for future work.

LDA with count vectorization

- Count vectorization
  - `max_df`: 0.4
  - `min_df`: 5
- LDA modeling
  - `n_components`: 480
  - `max_iter`: 15
  - `learning_method`: batch
- KNN modeling
  - `n_neighbors`: 20
  - `weights`: distance
  - `metric`: Kullback-Leibler div.

LDA with tf-idf vectorization

- Tf-idf vectorization
  - `max_df`: 0.4
  - `min_df`: 5
- LDA modeling
  - `n_components`: 480
  - `max_iter`: 15
  - `learning_method`: batch
- KNN modeling
  - `n_neighbors`: 20
  - `weights`: distance
  - `metric`: Kullback-Leibler div.

Table 8: Hyperparameterizations for LDA approach with count vectorization and with tf-idf vectorization

After having trained the two models with the hyperparameterization given in Table 8, we achieved the results shown in Table 9. The accuracy of the LDA with count vectorization (37%) is almost three times as high as the LDA with tf-idf vectorization approach. The accuracy of 37% might still seem low. However, let us put this into perspective. Picking randomly a news article has an accuracy of $\frac{1}{480} = 0.0021$, as there are 480 different companies. Therefore, LDA with count vectorization has an accuracy that is 176 times higher than random guessing. Comparing the recall and precision of both approaches, we see that again LDA with count vectorization exceeds LDA with tf-idf vectorization.

Measure	**CV**	**Tf-idf**
accuracy	0.37	0.13
recall	0.44	0.19
precision	0.36	0.12

Table 9: LDA prediction performance measures

In Table 10, we show the performance of each approach for every outlet. It turns out that we can predict press releases with an accuracy above the average. As the data is really sparse for news articles and especially for news articles per outlet, the accuracy fluctuates a lot. More data would be necessary to check, whether all outlets' accuracy move closer to a common average accuracy as more data comes in.

outlet URL	total	CV correct	CV acc (%)	Tf-idf correct	Tf-idf acc (%)
News outlet 17	1	0	0.00	0	0.00
News outlet 7	30	6	0.20	3	0.10
News outlet 3	8	4	0.50	2	0.25
News outlet 53	4	2	0.50	1	0.25
News outlet 9	5	3	0.60	1	0.20
News outlet 24	26	12	0.46	4	0.15
News outlet 70	4	0	0.00	1	0.25
News outlet 32	63	21	0.33	5	0.08
News outlet 41	1	0	0.00	0	0.00
News outlet 44	3	2	0.67	0	0.00
News outlet 10	71	22	0.31	13	0.18
News outlet 38	1	0	0.00	0	0.00
News outlet 51	2	1	0.50	0	0.00
News outlet 35	5	0	0.00	0	0.00
News outlet 11	16	3	0.19	2	0.12
News outlet 24	1	0	0.00	1	1.00
PR outlet 2	315	133	0.42	41	0.13
News outlet 46	2	0	0.00	1	0.50
News outlet 40	2	1	0.50	0	0.00
News outlet 47	12	4	0.33	0	0.00

Table 10: LDA prediction performance for each outlet

6.4 Bag-of-Words Approach

Equally to Section 6.3, we perform an extrinsic evaluation of the bag-of-words approach. The bag-of-words approach only differs from the LDA approach in that it leaves out the LDA component. Therefore we kept the other components' parameterization exactly as in the LDA approach, as shown in Table 11.

BOW with count vectorization

- Count vectorization
  - `max_df`: 0.4
  - `min_df`: 5
- KNN modeling
  - `n_neighbors`: 20
  - `weights`: distance
  - `metric`: Kullback-Leibler div.

BOW with tf-idf vectorization

- Tf-idf vectorization
  - `max_df`: 0.4
  - `min_df`: 5
- KNN modeling
  - `n_neighbors`: 20
  - `weights`: distance
  - `metric`: Kullback-Leibler div.

Table 11: Hyperparameterizations for bag-of-words approach with count vectorization and tf-idf vectorization

Having built the two models with the hyperparameterization given in Table 11 and having measured the models' performance, we received the results, shown in Table 12. In contrast to the preceding measurement, this time the tf-idf vectorization exceeds the count vectorization approach, having an accuracy of 51%. Either of the bag-of-words approaches is better than two LDA approaches for all three measures.

Measure	CV	Tf-idf
accuracy	0.47	0.51
recall	0.56	0.63
precision	0.44	0.46

Table 12: Bag-of-words prediction performance measures

In Table 13, we show the performance of each approach for every outlet. Again, the news articles prediction accuracy fluctuates a lot and the press releases have a promising performance.

outlet URL	total	CV correct	CV acc	Tf-idf correct	Tf-idf acc
News outlet 17	1	0	0.00	0	0.00
News outlet 7	30	12	0.40	12	0.40
News outlet 3	8	5	0.62	5	0.62
News outlet 53	4	4	1.00	4	1.00
News outlet 9	5	3	0.60	4	0.80
News outlet 24	26	7	0.27	13	0.50
News outlet 70	4	3	0.75	3	0.75
News outlet 32	63	25	0.40	25	0.40
News outlet 41	1	0	0.00	0	0.00
News outlet 44	3	0	0.00	2	0.67
News outlet 10	71	35	0.49	35	0.49
News outlet 38	1	1	1.00	1	1.00
News outlet 51	2	2	1.00	2	1.00
News outlet 35	5	0	0.00	0	0.00
News outlet 11	16	8	0.50	8	0.50
News outlet 24	1	1	1.00	1	1.00
PR outlet 2	315	157	0.50	169	0.54
News outlet 46	2	1	0.50	1	0.50
News outlet 40	2	1	0.50	1	0.50
News outlet 47	12	5	0.42	6	0.50

Table 13: Bag-of-words prediction performance for each outlet

6.5 Discussion

Both LDA approaches and both bag-of-words approaches are promising and we expect either of them to perform significantly better, if their hyperparameters were properly tuned using grid search. With regard to the LDA with tf-idf vectorization approach's performance, we might just have used a poor hyperparameter setting, whose generalization performance was close to a minimum. Nevertheless, our models are at least 60 times better than random guessing.

As already mentioned, the sparsity of the labeled news articles might also be an issue, since not all symbols and outlets are represented by the data set. On top of that it is completely unbalanced, looking at *PR outlet 2* which contains the majority of news articles. For a better assessment of the models' performance we need more data.

Another important aspect to consider is that both LDA approaches reduce the feature space tremendously compared to the bag-of-words models. The vectorizers have a vocabulary of size 6300, leading to feature space of this size for the k-nearest neighbor algorithm in the bag-of-words approach.

In contrast to this, the LDA approach has 480 topics (one for each company) and since the topic distributions are forwarded to the k-nearest neighbor algorithm, the feature space is only $\frac{8}{100}$ the size of the bag-of-words feature space.

However, both LDA approaches also have a disadvantage compared to the bag-of-words approaches, that one has to consider. Assuming the LDA model does provide exactly one topic per company, we would have to recalculate the LDA model if we did not want to lose accuracy over the long run, when adding new companies. In the bag-of-of words approach we would just vectorize the company descriptions (note that it might be necessary to also recalculate the vectorizer model) and would add the vectorized company descriptions to the k-nearest neighbor model.

7 Summary

In this research project we developed and deployed the infrastructure to retrieve news articles that cover world-wide economic developments. Our objective was to build a model, that was able to predict the companies that the news articles were about.

We introduced two approaches:

1. **LDA k-nearest neighbor approach**
   We retrieved company descriptions about companies of the S&P 500 index. These descriptions together with the news articles were vectorized (count vectorization, tf-idf vectorization). The vectorized descriptions were used to train a LDA topic model. The LDA topic model was used to predict the topic distributions of the vectorized company descriptions and vectorized news articles. As the company descriptions were labeled, we used the k-nearest neighbor algorithm to find the k closest company descriptions to a given news article.

2. **Bag-of-words k-nearest neighbor approach**
   Similarly to the LDA k-nearest neighbor approach, we retrieved news articles and company descriptions and vectorized them. However, we did not train a LDA model, but forwarded the vectorized news articles and company descriptions directly to the k-nearest neighbor algorithm.

With regard to Section 1.2, we now recap the goals we defined in this section and assess whether we fulfilled those within this research project.

Our main objective was to build a model that performs more reliable than a simple Bag-of-words k-nearest neighbor approach. At this point we can say that the LDA k-nearest neighbor approach's results are at least competitive. If trained in a grid-search fashion, it is possible that it exceeds the simple approach.

One drawback of the bag-of-words approach was its high feature space that makes calculations expensive and time consuming. We have shown that the LDA approach reduces the feature space to $\frac{7}{100}$ of the feature space size of the bag-of-words approach.

We trained the LDA topic model such that each company is represented by one topic and thereby automatically reduced the number of noisy features.

8 Future Work

There is quite a bit of research left for the LDA k-nearest neighbor approach to compare it with the bag-of-words k nearest neighbor approach.

First, we need a more resilient evaluation of the model's performance. Therefore, we need more labeled news articles and have to determine good model parameters using for instance grid search. Then, we are finally able to say, if we can cope with the simple bag-of-words approach.

Secondly, it could be worth investigating, if we can find patterns in which the models perform well or poorly. For instance, we could assign each company to its branch and check whether the branches are predicted better than we would expect them to be. If this was the case, we could train a model that predicts a branch of a news article. For each branch, we would train another classifier that predicted the company for a given news article.

Other researchers had promising results for text classification using Latent Dirichlet Allocation and support vector machines and thereby replacing the k-nearest neighbor algorithm. Applying this approach to match news articles to company descriptions could be an attempt worth examining.

Looking far ahead with regard to autonomous trading systems, we need a sentiment model for the news articles, such that we can finally predict stock market changes. One idea is to learn the sentiment model directly from news articles by comparing them with stock market changes.

9 Appendix

References

[1] *A script to generate contour plots of Dirichlet distributions*. https://gist.github.com/tboggs/8778945. Accessed: 2017-12-01, Note: Used script to generate Dirichlet plots.

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

[3] David M. Blei. "Probabilistic Topic Models." In: *Commun. ACM* 55.4 (Apr. 2012), pp. 77–84. ISSN: 0001-0782. DOI: 10.1145/2133806.2133826. URL: http://doi.acm.org/10.1145/2133806.2133826.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation." In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[5] 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.

[6] Gavin E Crooks. "On measures of entropy and information." In: *Tech. Note* 9 (2017), p. v4.

[7] James M Dickey. "Multiple hypergeometric functions: Probabilistic interpretations and statistical uses." In: *Journal of the American Statistical Association* 78.383 (1983), pp. 628–637.

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

[9] 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).

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

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. "Introduction to Information Retrieval." In: New York, NY, USA: Cambridge University Press, 2008, p. 119. ISBN: 0521865719, 9780521865715.

[12] 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.

[13] Xuan-Hieu Phan, Le-Minh Nguyen, and Susumu Horiguchi. "Learning to Classify Short and Sparse Text & Web with Hidden Topics from Large-scale Data Collections." In: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. Beijing, China: ACM, 2008, pp. 91–100. ISBN: 978-1-60558-085-2. DOI:

10.1145/1367497.1367510. URL: http://doi.acm.org/10.1145/1367497.1367510.

[14] Efsun Sarioglu, Kabir Yadav, and Hyeong-Ah Choi. "Topic modeling based classification of clinical reports." In: *51st Annual Meeting of the Association for Computational Linguistics Proceedings of the Student Research Workshop*. 2013, pp. 67–73.

[15] 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.

[16] I.H. Witten, E. Frank, and M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011, pp. 78–81. ISBN: 9780080890364.

Eidesstattliche Erklärung

Ich versichere an Eides, dass die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, May 23, 2018

Max Lübbering