UPPSALA
UNIVERSITET

Examensarbete 30 hp
Januari 2016

Detecting Twitter topics using Latent Dirichlet Allocation

Johan Risch

Institutionen för informationsteknologi
Department of Information Technology

Abstract

Detecting Twitter topics using Latent Dirichlet Allocation

Johan Risch

Latent Dirichlet Allocations is evaluated for its suitability when detecting topics in a stream of short messages limited to 140 characters. This is done by assessing its ability to model the incoming messages and its ability to classify previously unseen messages with known topics. The evaluation shows that the model can be suitable for certain applications in topic detection when the stream size is small enough. Furthermoresuggestions on how to handle larger streams are outlined.

Handledare: Erik Thorselius
Ämnesgranskare: Matteo Magnani
Examinator: Lars-Åke Nordén
UPTEC IT 16001
Tryckt av: Reprocentralen ITC

Contents

1 Introduction

Twitter is a popular micro-blogging service used for social interaction online that allows anyone to freely publish micro-blog posts. The published posts have a limit of 140 characters and are called tweets. The medium is largely used to share any kind of news and the editorial quality of tweets varies a lot. Since the quality varies, algorithms that can sort out relevant topics in this medium could be of great use.

The goal of this thesis is to automatically classify tweets in a stream with respect to an underlying topic using nothing but the actual text content of the tweet. This kind of classification has been done successfully on news articles published by news agencies[2, 6], Wikipedia articles[5] and other kinds of textual bodies[13]. One very common method for identifying topics in these kinds of feeds is the Latent Dirichlet Allocation. This thesis will examine whether Latent Dirichlet Allocation is suitable to identify topics in a medium with very short messages such as Twitter.

In 2013 there was on average 500 million[1] tweets posted per day. An unsupervised algorithm that can identify topics is necessary if these volumes of data are to be processed. Therefore if Latent Dirichlet Allocation can yield a good enough result it could be applied to the corpus of all tweets ever posted, classify them, and generate a huge data set of classified tweets.

A good topic detection algorithm can be used by entities ranging from intelligence agencies to private companies. An intelligence agency can use topic detection to automatically detect terrorist chatter by analysing streams of messages. The detected messages can then be flagged as interesting and sent for a manual review. This would decrease the stream of potential interesting messages allowing for a more thorough manual analysis per message. On the other side of the spectrum a company can use topic detection to find out what people say about them and their products. This might be extra useful when releasing new products and features since it can be used as an indicator of how the customers react to the changes by detecting negative and positive reactions.

In order to evaluate Latent Dirichlet Allocation a system written in Java was created. Latent Dirichlet Allocation was evaluated both quantitatively with Perplexity, a measurement on how well a model represents reality; and qualitatively by subjectively analysing output from the system.

There are a lot of differences between tweets and news articles. One important difference is that when identifying topics in news articles the documents are considered filtered and trustworthy. As Gayo Avello suggests, this is not the case for tweets[4]. The issue is that Latent Dirichlet Allocation does not consider whether the documents are noise, spam, or relevant which suggests that many topics may be caused by spam. Latent Dirichlet Allocation uses a collection of words, vocabulary, which the system accepts when identifying topics. An issue with Twitter is that the language constantly evolve rendering the vocabulary

1. http://www.internetlivestats.com/twitter-statistics/ accessed September 2015

non-static. Since Latent Dirichlet Allocation requires a static vocabulary[2] this could be an issue. There has been work done to avoid using a static vocabulary. Ke Zhai and Jordan Boyd-Graber suggests a method that replaces the vocabulary with a distribution of all possible combinations of characters[13]. This distribution is then weighted based on what words are observed. However, due to its late discovery and since using an infinite vocabulary is out of the scope of this thesis, see **Section 1.2**, it was omitted.

1.1 Research question

When applying Latent Dirichlet Allocation on a stream of tweets there are certain requirements set on the algorithm; the algorithm has to qualify as a *streaming algorithm*. An algorithms which can be called a steaming algorithm must fulfill the requirements at least three requirements: *memory-bound*, *single glance*, and *real-time*, described in **Section 3.2**. Whether Latent Dirichlet Allocation fulfills these requirements in a realistic manner is the first research question of the thesis.

As mentioned above there are no editors filtering out irrelevant tweets for Twitter. Since [2, 6], [5] and [13] all evaluate Latent Dirichlet Allocation on a medium that could be considered relatively spam free it would be interesting to investigate what effect spam has on the outcome and performance of Latent Dirichlet Allocation.

Blei et. al claims that the static vocabulary in a medium with rapidly changing vocabulary will cause Latent Dirichlet Allocation to perform poorly; and together with the two paragraphs above yields the four research questions of the thesis.

1. Is it feasible to apply Latent Dirichlet Allocation on a stream in terms of it being able to handle a stream as outlined in **Section 3.2**?
2. How does spam affect the performance of Latent Dirichlet Allocation?
3. Will Latent Dirichlet Allocation converge in terms of perplexity when a static vocabulary is used on a stream of tweets?
4. Does a static vocabulary mean that Latent Dirichlet Allocation will perform worse with an old outdated vocabulary compared to a vocabulary created for the data set?

1.2 Scope

The scope of thesis is analysing tweets based on their textual body only. While an obvious extension would be to use some of the meta-data included in tweets the scope was still set to exclude this meta-data since other media with short text messages might not have the same set, or any at all, of meta-data. If Latent Dirichlet Allocation could identify topics in a stream of tweets without using any meta-data from the tweets it might mean that it could be applied to other media of short texts as well.

1.3 Contributions

An implementation of Online Learning for Latent Dirichlet Allocation created by Michael Berkovsky [1] was modified and integrated into a system that was created in order to evaluate Latent Dirichlet Allocation. In **Section 7** it is shown that Online learning for Latent Dirichlet Allocation converges in terms of perplexity when using a static vocabulary. It is also shown that an outdated vocabulary does not necessarily mean that Online learning for Latent Dirichlet Allocation will perform worse. In the tests set up in **Section 7.3**

Online learning for Latent Dirichlet Allocation is tested on how well it can classify a held out test set once it has been trained. The performance was measured with precision, recall and F-measure and Online learning for Latent Dirichlet Allocation performed better or equally in all those terms when using an old out-dated vocabulary. Furthermore it is shown that spam indeed does affect the outcome of Latent Dirichlet Allocation and that while Online Learning for Latent Dirichlet Allocation theoretically is applicable on a stream it might not be fast enough as is and a proposal on how to set up a system that might be able to handle the throughput is outlined in the discussion.

2 Preliminaries

In order to avoid confusion this section will describe the terms used in the report. There is no need to read this section straight away, it can be taken aside while reading to quickly find the definitions.

P:1 word - is an item from a vocabulary $\{1, ..., V\}$. Words are represented as unit basis vectors with one component equal to one and the rest equal to zero[6].

$$w^u = \begin{cases} 1 & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}$$

P:2 document - a collection of words in a data set, denoted $D = \{w_1, ..., w_n\}$ where w_n is the nth word and N is the total number of words in the collection [6].

P:3 corpus - a collection of documents in a data set. It is denoted $C = \{D_1, ..., D_M\}$ where the D_mth document in the corpus and M is the total number of documents [6]

P:4 Precision, recall and F-measure - are measures used in information retrieval. Precision is the fraction of returned documents compared to the expected outcome. Recall is the fraction of relevant documents returned. F-measure is often called the precision of a test. It uses both precision and recall together to form a more descriptive score.

P:5 Prior - The prior of a model is the belief the model has before any observation has been done.

P:6 Posterior - The posterior of a model is the modified belief of the model after observations are made.

P:7 Twitter Firehose - is a payed service by Twitter which allows you to access an unfiltered and unsampled feed which is much larger that the stream obtained from the Twitter Streaming API.

P:8 Multinomial distributions - Are a generalisation of a binomial distribution. A binomial distribution takes two parameters, n and p. The distribution is obtained by doing n yes or no trials with a success rate of p. Multinomial distributions generalises this by allowing trials with k number of outputs from a trial rather than only yes or no. In other words a binomial distribution is a 2-Multinomial distribution.

P:9 Latent variables - variables that cannot be directly observed. Latent variables can be inferred from other observable variables [6]. Given a weighted die one can view the weight as a latent variable. If we never observe a toss there is no way to identify which number the die favors. However, if we toss the die several times we could deduce the latent weight by analysing the observed die tosses. In other words, latent variables are unobservable variables that affect an outcome. These variables can then be deduced by observing and analysing the observed variables.

P:10 Posterior inference In Bayesian inference the posterior inference is updated when an observation is made. For instance, if we have a bag of black and white marbles with unknown ratio between them and our initial assumption is that there are 50/50 of each color. When a marble is taken from the bag this ratio is updated based on the color of the marble. Let us assume that after 20 marbles we have gotten 15 black and 5 white; our posterior inference of the ratio would then be 75/25. This is a very simple method of posterior inference and there exist a lot of different methods for it. In this thesis an online version of the Variational Bayes inference is used. Figure 1 shows how Bayes theorem can be used to get the prior from the posterior.

$$p(\alpha|\beta) = \frac{p(\beta|\alpha) * p(\alpha)}{p(\beta)} \tag{1}$$

$$p(\beta) = \int p(\beta|\alpha) d\alpha \tag{2}$$

Figure 1: Bayes theorem. This theorem is the basis on which Variational Bayesian inference is built upon. $p(\alpha|\beta)$ is the posterior, $p(\beta|\alpha)$ is the likelihood, $p(\alpha)$ prior and $p(\beta) = \int p(\beta|\alpha)d\alpha$ is the marginal likelihood of the model, or in other words: the model evidence

P:11 Hyper parameters - Are parameters controlling the prior of the model. The hyper parameters are used to fit the model on the problem and or data at hand. In this report α is a hyper parameter for the topic distribution. If α is well fitted to the problem our model will work better than if it is not.

P:12 Simplex - A simplex is used to describe distributions of parameters in an object. In the scope of this thesis, a simplex is used with the variational inference. While Latent Dirichlet Allocation uses a *k-simplex* where *k+1* is the number of topics. This section will describe the principles of a *2-simplex* since they are easier to understand. In Figure 2 the objects are documents and the parameters are the topics of which a document consists of. The total sum of e_0, e_1 and e_2 for any document sums to 1 or 100%. If a document is on any of the edges between any two topics it means that the third topic is absent in the distribution of the document. When expanding the simplex to a 3-simplex we get a tetrahedron instead of a triangle. The same principles apply on how a document consists in different parts of all topics. As mentioned above; when modelling over k topics a *(k-1)-simplex* is used.

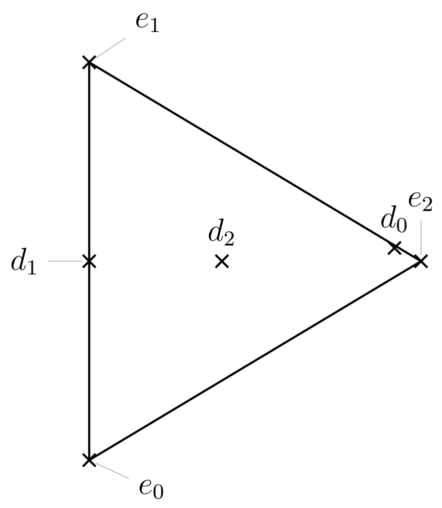


Figure 2: Show a 2-Simplex over three topics e_0,e_1 and e_2. The three points d_0, d_1 and d_2 are documents and their position in the simplex shows how much of each topic they contain. d_0 consists of approximately 90% topic e_2 and 10% topic e_1 and almost none of topic e_0. d_1 contains 50% of topics e_0 and e_1 and none of e_2 while d_2 has an equal part of all three topics.

3 Background

Consider an application where we want to serve an unbiased stream of tweets for different topics. Not only is it affected by hash tags but also dependent on the actual content in the tweets. This can be done with a *Topic model* that is trained and run on a *stream* of tweets. *Latent Dirichlet Allocation* is evaluated on how well it is suited to solve this problem. Before Latent Dirichlet Allocation can be evaluated on a stream the algorithms needs to be extended, allowing it to be run on a stream. An implementation of Hoffman et al's *Online learning for Latent Dirichlet Allocation* was used when evaluating the suitability on tweets.

3.1 Topic modelling

Identifying topics in texts is called *Topic Modelling.* This is usually done by detecting patterns in a collection of documents called a *corpus*[P:3], and grouping the words used into topics. One key part of the topic model is how the model assumes the documents are being generated. There are several different approaches to this. The topics are usually defined as a density function $p_i(w)$ where w is a word and p_i is the density function for topic i. Thus $p_i(w)$ is the probability for the word w given topic i. A generative topic model is a topic model that allows for classification of unseen documents once the model has been trained without going through the whole corpus. Which is an important feature for a model when run on a stream of documents, which is the same as a possibly infinite corpus.

3.2 Requirements for streaming algorithms

Since Online learning for Latent Dirichlet Allocation will be evaluated on a stream of tweets it is important to understand what criteria are set on the algorithm in order to be suitable for streams. There are at least three main criteria that must be met by an algorithm in order for them to work on streams.

1. **Memory-bound** - When working on a stream of data the size of the data set should be assumed to be infinite, in other words, the stream should be assumed to have no end. This means that the algorithm cannot save the data for later use, it must discard old data in the same rate as new data comes in. By doing so the algorithm can become memory-bound. The output of a streaming algorithm is usually a stream as well. However, if the output is dumped to disk this does not affect the memory-bound status of the algorithm.

2. **Single glance** - The data received from a stream is usually grouped into so called windows. The window can be all the data points received the last ten seconds or it might be the last 100 data points received. The important part is that the window is somehow moving forward discarding old data points. These data points cannot be processed once they have been discarded. This means that an algorithm is allowed to view the data points inside a window several times together; but once the window has moved forward the data points cannot be viewed again.

3. **Real-time** - Since the data stream continues to arrive whether the algorithm has processed the previous data or not it is important that the

algorithm has a strategy when the stream is overflowing. This can either be done by optimisation or by throwing away incoming data when the processor is busy.

3.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation is a generative topic model. Latent Dirichlet Allocation assumes that each word in a document is generated from a topic that in turn is picked from the topic distribution for each document. The topic distribution for each document is generated from a Dirichlet distribution [2] which means that Latent Dirichlet Allocation allows a document to consist of several topics in different magnitude.

3.3.1 Online learning for Latent Dirichlet Allocation

Hoffman et al. provides an online version of Latent Dirichlet Allocation. It modifies the original Latent Dirichlet Allocation to work on a stream. The documents are collected into mini-batches and the algorithm is run on each mini-batch. Each mini-batch updates a matrix that represents the topics and their word frequency spectrum. This algorithm is suitable for streaming since it satisfies the criteria for a streaming algorithm:

1. **Memory-bound** - Each mini-batch updates a matrix, which is constant in size.
2. **Single glance** - Each mini-batch is only processed once.
3. **Real-time** - While the algorithm has a linear complexity in terms on the number of topics modelled it still gets quite expensive when using a large set of topics. A possible solution to this issue is outlined in **Section 8.2**

4 Methodology

The thesis was divided into three major phases. The first phase was a literature study with the main goal of understanding Latent Dirichlet Allocation. During this phase the theory behind Latent Dirichlet Allocation together with Bayesian statistics were studied. The second phase was an experimental phase where Latent Dirichlet Allocation was applied to various media, including Tweets. Phase two allowed for a better understanding on how Latent Dirichlet Allocation behaved when used on different data sets. There were no quantitative measurements done during this phase since its main purpose was to deepen the understanding of the algorithm. The knowledge acquired from the second phase was taken into consideration when setting up the quantitative experiments in phase three.

4.1 Phase one: Literature study

As mentioned above the purpose of this phase was to get acquainted with Bayesian statistics and more specifically Latent Dirichlet Allocation. The literature study consisted of papers from Blei et al. [2], Hoffman et al. [5] and Zhai et al. [13]. [2] goes through the original theory behind Latent Dirichlet Allocation and was studied a lot in order to acquire proper knowledge and understanding of Latent Dirichlet Alloctaion. The second paper [5] suggested modifications to Latent Dirichlet Allocation which allows it to be applied to a stream of documents. The paper by Zhai et al. suggests an algorithm that has an infinite vocabulary and thus would render the static vocabulary issue irrelevant. While this method is out of the scope of this thesis it is an interesting method to further evaluate in future work.

4.2 Phase two: Experimenting

During the literature study a Java implementation of Hoffman et als. Online learning for Latent Dirichlet Allocation created by Michael Berkovsky [1] was discovered. Since this implementation only needed a few modifications to work on a stream of Tweets, see **Section 6.5.1**, it was used during the rest of this thesis.

In phase two the implementation was run on news articles, tweets, and Wikipedia articles. The reason for testing the different media was to see the difference in how Latent Dirichlet Allocation performed on the different data sets in order to better understand it. Furthermore phase two was used to find out a good way to actually evaluate Latent Dirichlet Allocation. The original research question was just *Evaluate Latent Dirichlet Allocation* on a stream of tweets, the last part of phase two was dedicated to how this evaluation can be done.

4.2.1 Unexpected issues

During this phase there were two major issues that were not expected.

- Tweets are very unstructured and a lot of them only have a short comment followed by a link to an article. These types of tweets are hard to classify without the content of the link and these links polluted the vocabularies in the data set which meant that the vocabularies would have to be groomed manually in order to perform well.

- It is hard to objectively measure topic modeling on a quantitative way if there is no test data present [10]. Furthermore it was very hard to find any large data set of manually labeled tweets, the biggest found contained only 5 000 tweets which is not enough to measure Latent Dirichlet Allocation since it needed far more tweets to be trained in the first place. This resulted in an improved solution where tweets were queried in a way that would yield tweets with roughly the same topics, see **Section 7.3.1** for more details.

4.3 Phase three: Evaluating Latent Dirichlet Allocation

In phase three the evaluation procedure from phase two was implemented and executed. The evaluation was divided into two parts. The first part was done by measuring perplexity. Perplexity is a measurement on how well the results represents reality; it is not an absolute metric but it can be used to compare several algorithms, or the same algorithm with different parameters, on the same data set. The second part measured how well Latent Dirichlet Allocation could identify topics in tweets from an automatically generated test set with known topics. This test was run in two flavors; one on a data set which could be considered relatively spam free and the other with the same data set with injected spam. By controlling the amount of spam injected into the data set it should yield an overview on the effects of spam on Latent Dirichlet Allocation. The measurements were done using *precision*, *recall* and *F-measure*. These metrics are often used in data mining, *F-measure* is a weighted combination of *precision* and *recall* that can be a good indicator on how well an algorithm performs.

5 Related work

This report uses Online learning for Latent Dirichlet Allocation. Other methods of topic modelling that might be of interest are Hierarchical Dirichlet Processes [11], A correlated topic model of Science [3], and Automatic Topic Detection with an Incremental Clustering Algorithm [14]. In order to achieve a realistic scope only one method had to be evaluated. Due to the many recent publications on Latent Dirichlet Allocation and derivations thereof it was considered a good candidate to focus on in the thesis. This assumption was reinforced when discussing with the supervisor and reviewer.

Latent Dirichlet Allocation was proposed by Blei et al. in their paper from 2003 [2]. This model was then improved to allow for online learning seven years later in 2010 by Hoffman et al. [5]. When doing topic modelling there are certain details that are needed to be considered. These details are how the data being modeled is structured, removing stop words and finally modelling how the bodies of texts are generated.

5.1 Stop words

Blei et al. suggests that one should remove common stop words in a document before running Latent Dirichlet Allocation [2]. Stop words are words that are very common and do not alter the meaning of a sentence [6], typical stop words include "the", "in" and "so" etc. Kim et al. automatically filtered out words from the vocabulary that were present in more than 50% or less that 5% of the documents [6]. They state that it was an effective way of removing both stop words, misspelled words and non-words. In this thesis a static list of common stop-words[2] was used as suggested by Blei et al. [2]

5.2 Topic modeling

A topic model is a statistical method for detecting abstract topics in documents. Let us start by defining what an abstract topic is. In this thesis it is some underlying variable that causes documents with the same value of the variable to produce bodies of text with similar word frequency spectrum.

The general idea of a topic model is that a generative process is defined. For instance, the words used in an e-mail to a family member are probably less formal than those sent to a superior at work. Let's assume that it is true that the relationship between the sender and the receiver of a message affects the type of words used; and that each type of relationship has its own bag of words where the probability of each word is dependent on how often it is used in the bags relationship. If the sender is writing a message to a very *close friend* most words would be picked from the *close friends* relationship bag of word. However, if the message is sent to a co-worker with an equal amount of professional and personal relationship half of the words would be picked from each relationship bag of words. This is a description of a generative process. If the process is consistent with reality one can use reverse engineering to infer what relationship

2. Found on the github page https://github.com/johanrisch/TwitterTopicMining

the sender had to the receiver by putting the words in the message into a bag of words. This bag of words could then be matched to all relationships bag of words to find how similar the message bag is to different relationships.

The process described in the previous paragraph is an example of a topic model that has some resemblance to Latent Dirichlet Allocation. The generative process is never used to produce any documents, rather, it is a hypothesis on how the documents observed are being generated.

There are several different models that can be used. Blei et al. goes through the following models and explains why their Latent Dirichlet Allocation is superior. Section 5.2.1 through 5.2.4 describes how one of the most simple models, the *Unigram model* can be extended step by step to become the Latent Dirichlet Allocation model. As mentioned above, these generative processes are not used to actually generate documents, they are the basis for which the posterior inferences are made. All the mathematical details are not provided in this report. To get these details the paper written by Blei et al. [2] in 2003 should be read.

5.2.1 Unigram model

The unigram model does not take any topics into consideration. However, it is a good starting point when explaining the different models. As shown with the plate notation in figure 3 the unigram model generates a document by selecting words from a completely independent multinomial distribution[P:8].

$$p(w) = \prod_{n=1}^{N} p(w_n)$$

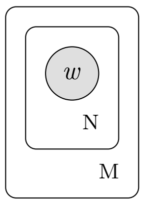


Figure 3: The plate notation for the Unigram model. A corpus consists of M documents which in turn consists of N words w that are independently drawn from a multinomial distribution

In the context of the example given at the beginning of Section 5.2 this would imply that each document if completely independent of the relationship between the sender and receiver of the message. Each message is written without any bias towards the relationship.

5.2.2 Mixture of unigrams

The mixture of unigrams model is created by adding a latent[P:9] discrete random topic variable z to each document. Each document has a multinomial distribution[P:8].

When generating a document with the mixture of unigrams model a topic is selected for each document. Once the topic is selected the words are drawn from the multinomial distribution[P:8], for the selected topic. [2]

$$p(w) = \sum_{z} p(z) \prod_{n=1}^{N} p(w_n|z) \tag{3}$$

As the plate notation in figure 4 and equation 3 suggests there is one disadvantageous limitation to this model. Each document consists of one, and only one topic. In our example from Section 5.2 this would mean that a sender could not be friends with his co-workers or boss. This might be to limiting when modelling the real world. Furthermore Blei et al. made an empirical study that proved this assumption is to limiting when modelling a large collection of documents [2].

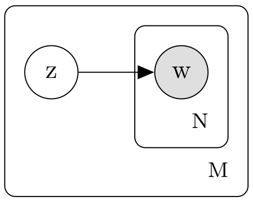


Figure 4: The plate notation for the Mixture of Unigrams model. In this model each document is assigned a topic. The words in each document is then drawn from the multinomial distribution for the assigned topic.

5.2.3 Probabilistic latent semantic indexing

Probabilistic latent semantic indexing *pLSI* expands the model further by adding an observed document[P:2] d that is conditionally independent to the word w_n given an unobserved topic z [2]. pLSI allows for documents to consist of several topics since $p(z|d)$ serves as a mixture weight of topics for the document d [2]. However, there is an issue with pLSI. The document is an observed variable and thus we have no way of dealing with new documents that are not a part of our training data which means that the pLSI model is not a well defined generative model. In other words, there is no natural way of classifying a previously unseen document [2]. From the example above this would mean that we could classify all of the previously sent messages by a person; but not classify a new message that is written without going through all of the e-mails on the senders account.

It is important to note that the parameters needed to calculate scale linearly with the number of documents in the training set [2]. If we have a k-topic pLSI model the parameters are k multinomial distributions[P:8] of size V and M mixtures over k hidden topics which gives us $kV + kM$ parameters with a linear growth in M [2]. The consequences of this is that the pLSI is not well suited for streaming applications since it's run-time increases with the number of documents and that all older documents needs to be processed for each new document violating the single-glance criteria from **Section 3.2**.

$$p(d, w_n) = p(d) \sum_{z} p(w_n|z)p(z|d)$$

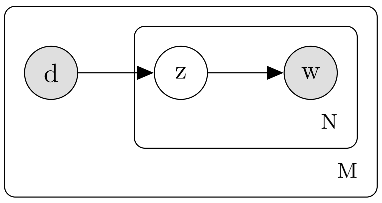


Figure 5: The plate notation for the probabilistic latent semantic indexing. This model relaxes the assumption from *Mixture of unigrams* that each document can only have one topic. Instead we have a latent variable that is the topic allowing documents to consist of several topics with different magnitude.

5.2.4 Latent Dirichlet Allocation

The Latent Dirichlet Allocation can intuitively be described as a model that identifies topics of documents based on the word frequency spectrum in each document. In other words: the words in a document are dependent on the *latent* topic distribution. The Latent Dirichlet Allocation works with a Dirichlet prior that the words in a document are generated based on the topic distribution of the document [2]. If we assume that this is the case we can use any method of posterior inference[P:10] to infer the latent[P:9] variables in the Latent Dirichlet Allocation model. Both Blei et al. and Kim et al. used an online version of the variational Bayes inference m.

As the plate notation in figure 6 suggests the document d was changed into a latent[P:9] multinomial distribution[P:8] parameter θ and added two more parameters α and β [2]. α is the hyper parameter[P:11] for the Dirichlet distribution θ; α controls the generation of θ. β is a $k \times V$ matrix where each cell represents the distribution of words for each topic. β_{in} = the probability for word w_n in topic i. If we put all of this together we get that for each document we do the following:

- Generate a multinomial topic distribution θ from α.
- Generate N words by picking one topic from θ_n for each word and then generate a word w_n from the column representing θ in β.
- Repeat for each document.

In Latent Dirichlet Allocation the words in both seen and unseen documents are generated from topics that are randomly generated given α. Thus an unseen document may be classified in the same way as a training document and we have a well defined generative model for topic detection.

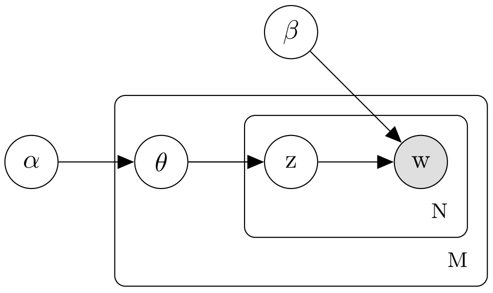


Figure 6: The Latent Dirichlet Allocation expands the model by adding a matrix β that represents the words distribution for each topic, a Dirichlet hyper parameter[P:11] α that is used to generate the multinomial distribution[P:8] parameter θ that represents the topic distribution for each document. A document with N words is then generated by choosing N topics from θ and then for each topic pick one word from the column in β that represents the topic.

5.3 Latent Dirichlet Allocation Inference and Parameter Estimation

In order to make use of our Latent Dirichlet Allocation model we need to have some method of inference and parameter estimation [2]. This section will go through the method used in the paper from Blei et al. [2]. In order to make use of the Latent Dirichlet Allocation model we need to infer the posterior distribution of the latent[P:9] variables [2]. Blei et al. formulates this problem mathematically as shown in equation 4

$$p(\theta, z|w, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta))}{p(w|\alpha, \beta)} \tag{4}$$

Equation 4 is a hard equation to solve and Blei et al. suggests that we marginalise over the latent variables. For an in depth run-down on how to get to Equation 5 please read the paper by Blei et al. [2]. Equation 5 has a high coupling between θ and β that makes it intractable [2]. However it is still possible to use some kind of variational approximation, including but not limited to: variational approximation, Markov chain Monte Carlo and Laplace approximation [2]. Blei et al. describes how to use a convexity-based variational inference to solve equation 5.

$$p(w|\alpha, \beta) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \int \left(\prod_{i=1}^{k} \theta_i^{\alpha - i} \right) \left(\prod_{n=1}^{N} \prod_{i=1}^{k} \prod_{j=1}^{V} (\theta_i \beta_i)^{w_n^j} \right) d\theta \tag{5}$$

5.3.1 Variational inference

Since equation 5 is intractable Blei et al. modifies the model by removing the troubling edges between θ, z and w resulting in figure 7. This variational distribution can also be written as equation 6 [2]

The parameters $(\gamma^*(w), \phi^*(w))$ are document specific and the Dirichlet parameters $\gamma^*(w)$ are viewed as providing a representation of a document in the topic simplex[P:12] [2]

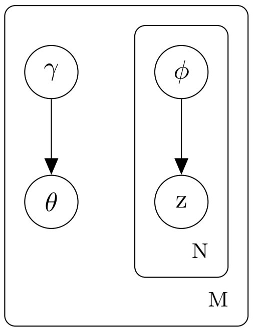


Figure 7: This is the model used to approximate the posterior in Latent Dirichlet Allocation. Kullback–Leibler divergence) is used to optimise the parameters γ and ϕ in order to avoid loss of information when using this posterior distribution.

.

$$p(\theta, z|\gamma, \phi) = q(\theta|\gamma) \prod_{n=1}^{N} q(z_n|\phi_n) \tag{6}$$

$$(\gamma^*, \phi^*) = argmin_{(\gamma,\phi)} D(q(\theta, z|\gamma, \phi)||p(\theta, z|w, \alpha, \beta)). \tag{7}$$

Blei et al. shows that this new simplified model leads us to the optimisation problem shown in equation 7. Which is basically minimising the divergence between the variational distribution and the true posterior $p(\theta, z|w, \alpha, \beta)$ [2]. An iterative fixed-point, **Section 5.3.2** method can be used to achieve this minimisation [2] and Blei et al. shows how this leads to the following two update equations

$$\phi_{ni} \ni \beta_{iw_n} expE_q[log(\theta_i|\gamma)] \tag{8}$$

$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni} \tag{9}$$

The expectation multinomial update can be computed with equation 10 [2]

$$expE_q[log(\theta_i|\gamma] = \Psi(\gamma_i) - \Psi(\sum_{j=1}^{k} \gamma_j) \tag{10}$$

Where Ψ is the first derivative of the $\log \Gamma$ function witch is computable via Taylor approximations [2]. Γ is defined as $\Gamma(x) = (x-1)!$ One important detail that has been omitted until now is that the new variational model is actually a conditional distribution varying as a function of w [2]. If we take a closer look on the optimisation problem in equation 7 we can see that we obtain (γ^*, ϕ^*) when optimising $q(\theta, z|\gamma, \phi)$ against a fixed w thus we can state this dependency explicitly by writing the variational distribution as $q(\theta, z|\gamma^*(w), \phi^*(w))$ [2]. It should now be clearer how $q(\theta, z|\gamma^*(w), \phi^*(w))$ can be used as an approximate posterior to the original distribution $p(\theta, z|w, \alpha, \beta)$ [2]. In their paper Blei et al. clearly explains the variational inference algorithm with the pseudo code in algorithm 1

```
initialise $\phi_{ni}^0 := 1/k$ for all $i, n$
initialise $\gamma_i := \alpha_i + N/k$ for all $i$
repeat
    for $n$ = 1 to $N$ do
        for $i$ = 1 to $k$ do
            $\phi_{ni}^{t+1} := \beta_{iw_n} exp(\Psi(\gamma_i^t))$;
        end
        normalize $\phi_{ni}^{t+1}$ to sum to 1
    end
    $\gamma^{t+1} := \alpha + \sum_{n=1}^N \phi_n^{t+1}$
until Convergence;
```
Algorithm 1: Variational inference for Latent Dirichlet Allocation

When inspecting this algorithm one can see that each iteration inside the repeat block takes $O((N+1)k)$ operations [2]. Empirical studies done by Blei et al. showed that the number of iterations needed was dependent on the number of words in a document which gave roughly $O(N^2k)$ iterations [2]

5.3.2 Fixed-point iteration

Fixed-point iteration is a method that can be used to approximate roots of a function. Given a function $f(x)$ re-write $f(x) = 0$ to $x = g(x)$. Then, set that x_{i+1}, that is x for iteration $i + 1$ to $g(x_i)$. This leads to the following formula: $x_{i+1} = g(x_i)$. As an example, lets find the roots to $x^2 - x - 1$.

$$\begin{aligned} f(x) &= x^2 - x - 1 \\ x^2 - x - 1 &= 0 \end{aligned} \tag{11}$$

$$\begin{aligned} x^2 - x - 1 &= 0 \\ x^2 &= x + 1 \\ x &= 1 + \frac{1}{x} \\ x_{n+1} &= 1 + \frac{1}{x_n} \quad (F1) \end{aligned} \qquad \begin{aligned} x^2 - x - 1 &= 0 \\ x^2 - x &= 1 \\ x(x-1) &= 1 \\ x &= \frac{1}{x-1} \\ x_{n+1} &= \frac{1}{x_n - 1} \quad (F2) \end{aligned}$$

As shows in Equation 11 we can get two different equations on the form $x = g(x)$. There is one more version but it is omitted in this example. Before we start our iteration we need to guess an x_0. Lets say $x_0 = 0$ for $F1$ and $x_0 = 1.6$ for $F2$ we then get the following iterations with $F1$ and $F2$

$$(F1) \quad x_{n+1} = 1 + \frac{1}{x_n}$$

$$\begin{aligned} x_0 &= 2 \\ x_1 &= g(x_0) = 1.5 \\ x_2 &= g(x_1) = 1.667 \\ x_3 &= g(x_2) = 1.6 \\ x_4 &= g(x_3) = 1.625 \\ x_5 &= g(x_4) = 1.6153 \\ x_6 &= g(x_5) = 1.619 \end{aligned}$$

$$(F2) \quad x_{n+1} = \frac{1}{x_n - 1}$$

$$\begin{aligned} x_0 &= 1.6 \\ x_1 &= g(x_0) = 1.667 \\ x_2 &= g(x_1) = 1.5 \\ x_3 &= g(x_2) = 2 \end{aligned}$$

This proves that $F1$ converges quite quickly while $F2$ diverges. It can be proved that if x_r is the root to $f(x)$ and $|g'(x_r)| < 1$ then fixed point iteration converges. Otherwise it will diverge.

5.3.3 Parameter estimation

The last piece of the puzzle is the parameter estimation. There are at least two different ways of estimating these parameters, the empirical Bayes method and the fuller Bayesian method [2]. We will go through the empirical Bayes method here, for the full Bayesian method take a look at the paper from Blei et al.

Given a corpus of documents $C = \{D_1, D_2, ..., D_m\}$ we want to find α and β such that the marginal log likelihood is maximised [2], this is shown in equation 12

$$\ell(\alpha, \beta) = \sum_{d=1}^{M} \log p(w_d|\alpha, \beta) \tag{12}$$

As mentioned previously computing $p(w|\alpha, \beta)$ is not tractable but with variational inference we get a tractable lower bound on the log likelihood [2]. Blei et al. shows us that we can approximate the estimates for Latent Dirichlet Allocation with an *alternating variational Expectation Maximisation (EM) procedure* [2]. For a detailed derivation see the report on Latent Dirichlet Allocation by Blei et al. They derive the following EM-algorithm described in Algorithm 2 which is repeated until the lower bound on the log likelihood converges [2]. Furthermore Blei et al. shows us that the M step in Algorithm 2 can be written as equation 13. Lastly they also show that α can be implemented using an efficient Newton-Raphson method in which the Hessian optimising is inverted in linear time [2].

$$\ell(\alpha, \beta) = \sum_{d=1}^{M} \log p(w_d|\alpha, \beta) \tag{13}$$

5.4 Latent Dirichlet Allocation performance

Blei et al. used two sets of training corpora to test the generalisation of the Latent Dirichlet Allocation compared to other models such as *Mixture of unigrams* and *probabilistic latent semantic indexing.* The documents in the corpora were unlabelled and they measured the density estimation. Their result was that

1. **(E-step)** For each document, find the optimising values of the variational parameters $\{\gamma_d^*, \phi_d^* : d \in D\}$. This is done as described in the previous section.

2. **(M-step)** Maximise the resulting lower bound on the log likelihood with respect to the model parameters α and β. This corresponds to finding maximum likelihood estimates with expected sufficient statistics for each document under the approximate posterior which is computed in the E-step.

Algorithm 2: Parameter estimation algorithm as described by Blei et al. in their paper on Latent Dirichlet Allocation [2]

the Latent Dirichlet Allocation had a lower perplexity than the other methods described when the number of topics increased. A lower perplexity means that the model is better at generalisation, or in other words, better at modelling documents that is previously unseen [2].

6 System overview

The system was created in Java consisting of five major components as seen in Figure 8. Each component is loosely coupled with an interface allowing each part to be exchanged. The first component is the *Stream provider* and this component is responsible for fetching the tweets (documents). Two Stream providers were created, one that read tweets from a file and one that read live from Twitter. The second component is the filter component. For every tweet the filter may choose to pass through the tweet or remove it. In order to remove unwanted topics a spam filter could be applied here. The third component, the tokeniser, splits the tweet into an array of strings where each string is one word, or token. In the fourth component the tokens are stemmed using a stemmer which outputs an array of stemmed tokens. The stemmed tokens are passed to the fifth component which is a modified version of Berkovsky's Online learning for Latent Dirichlet Allocation [1]. The original implementation took a folder of files and did not allow for a continuous stream of tweets; thus the program was modified by adding the Stream provider, Filter, Tokeniser and Stemmer.

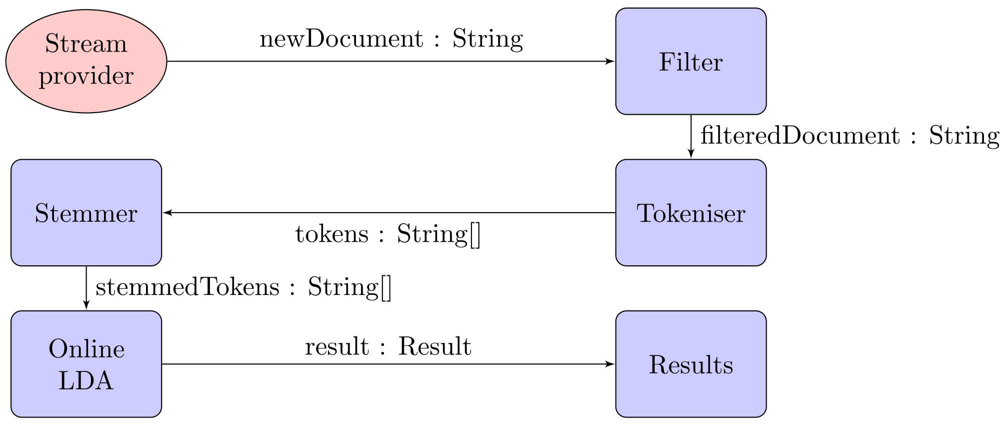


Figure 8: Overview of the data flow in the system. The stemmer used is the Snowball stemmer (Section 6.4). The Stream provider notifies the filter whenever a new document is read. The filter may just pass through each document or it could be a fully fledged spam filter. The tokeniser splits the documents into an array of words that is later stemmed by the stemmer. The Online learning for Latent Dirichlet Allocation waits until it has collected a mini-batch of documents and then runs the mini-batch through the Latent Dirichlet Allocation and yields and intermediate result with topics and perplexity.

This implementation was modified to be able to get documents from an input stream allowing the stream of tweets to be piped into the Latent Dirichlet Allocation program. Evaluation of Latent Dirichlet Allocation was made in two phases. First in an experimental manner by subjectively viewing the results while developing the system and tuning the parameters. Once the first phase was completed the observations were taken into consideration when deciding what ranges of parameters to use when running a script to find combination of parameters that yielded lower perplexity score. Furthermore the system was tested on *precision*, *recall*, and *F-measure* using a dataset containing tweets with known topics.

6.1 Stream provider

In order to get access to the Twitter streaming API [12] a Twitter application had to be created in order get an access token. Once the token was obtained a Java module was created. When connecting to Twitter streaming API there are two types of streams: either a sampled stream of all tweets or a filtered stream based on query parameters. When using the filtered stream there are two major query types;:either supplying geographic areas in the form or rectangles or supplying a number or terms contained in the tweet.

Geographical queries have one issue: the streaming API will only supply geographically tagged tweets. A test was set up to count the number of geographically tagged tweets compared to those who are not. The test ran through 50,000 tweets containing any of the words *Twitter*, *Facebook*, *Instagram*, *Google*, *Apple* or *life*. Of these 50,000 tweets 279, or 0.0056%, where geographically tagged. This is an issue since we get a lot less data when using geographical queries. *Content queries* on the other hand filter on the terms and this gives us biased content in streams. Ideally the Twitter Firehose[P:7] would be used. However, the Firehose API hard to get access to.

6.2 Filter

The filter step is used to remove tweets that are not of any interest. Since what tweets are relevant is dependent on the application of the tweets the filter module was made exchangeable. One common application of topic modelling is categorising news articles. There are several differences between news articles and tweets. One of these differences is that news articles could be viewed as spam free, since news agencies have legally responsible publishers. These publishers make sure that the content published is both legal, but also something interesting and relevant for readers. Tweets are posted by anyone and thus there is no one stopping spam from being published.

There are several ways this filter could be implemented. One way is any version of a naive Bayes spam filter [8]. A naive Bayes spam filter works similar to Latent Dirichlet Allocation with only two topics, spam and not spam. In the conducted tests which contained spam it was known whether or not the tweet was injected as spam. Thus the filter used in the experiments was created to be able to easily change the amount of spam let through the filter.

6.2.1 Spam

The definition of spam in this thesis is bodies of texts that we are not interested in. This means that spam is very context specific. If the application wants to identify tweets with high news value then most of the tweets about a users private life would be considered spam. However, if the application was to identify topics that people talk about that are relevant to their private life then tweets from news agencies would be considered spam.

If the data set used to train the model on is polluted with spam the model might actually consider the spam as a topic itself. This might not be an issue if the persons analysing are aware that some of the topics are spam. This means that Latent Dirichlet Allocation might actually be able to sort out the spam it self. As long as those analysing the model are aware of it.

6.3 Tokeniser

When processing texts of natural language a tokeniser is often used. The tokeniser splits the text into tokens and can also annotate the texts. For instance there is a part of speech tokeniser called TweetNLP[3]. The advantage of a good tokeniser is that it might help the model better understand texts. For instance TweetNLP tries to identify proper nouns. Instead of saving each Proper noun such as: *Facebook, Twitter, Instagram, Ikea, Uppsala* as separate word all of these could be grouped into one single token (^) in TweetNLPs case. This way

3. Found at http://www.ark.cs.cmu.edu/TweetNLP/ accessed July 28, 2015

other type of topics could emerge. Instead of the topics: *"Positive reactions to Facebook"*, *"Positive reactions to Twitter"*, *"Positive reactions to Instagram"*, *"... to Ikea"* and *"... to Uppsala"* a single topic *"Positive reactions to ˆ"* could emerge.

During evaluation two different tokenisers were used. The first one simply split a string into tokens based on all non alpha-numeric characters. The second tokeniser was was based on O'connor et al. tokeniser used in TweetMotif [9]; which is a specialised tokeniser for tweets.

6.4 Stemming

Stemming was defined by Lovins [7] in 1968 as:

> "... computational procedure which reduces all words with the same root (or, if prefixes are left untouched, the same stem) to a common form, usually by stripping each word of its derivational and inflectional suffixes."

In this project the Java version of the Snowball Stemmer[4], which is based on the stemmer proposed by Lovins, was used. Stemmers do not take context into consideration. For instance the Snowball stemmer stems both *informational* and *informality* to the word *inform.* Since Informational and informality do not mean the same thing there is a risk of loosing information when using stemming. However, it seems to be common practice to use a stemmer when doing natural language processing. All tests in this project where executed with the snowball stemmer.

6.5 Online learning for Latent Dirichlet Allocation

Hoffman et al. provides an online version of Latent Dirichlet Allocation, see Section 5.2.4. The difference from the original Latent Dirichlet Allocation is that it requires only one pass over the data set. They define Online learning for Latent Dirichlet Allocation as shown in Algorithm 3; definitions of all symbols can be found in Table 1. While working in a similar manner to the regular Latent Dirichlet Allocation there are a few differences. ρ_t is a parameter that controls how fast old iterations are forgotten. τ_0 controls the weight of the early iterations while κ defines the decay rate of old λ. λ is the result of the previous mini-batches. More precisely it is the variational parameter on the matrix with word frequency spectrum of all the topics.

While Algorithm 3 shows that it can process each document t separately it is a common practice to collect a mini-batch of documents and then run the algorithm on all documents in each mini-batch [2, 5, 13]. Furthermore Hoffman et al. show that if and only if $\kappa \in (0.5, 1]$ convergence is guaranteed [5].

6.5.1 Berkovsky's Java Online learning for Latent Dirichlet Allocation

Michael Berkovsky has created and shared a Java implementation of Online learning for Latent Dirichlet Allocation (JOLDA). Berkovsky's program was

4. Found at http://snowball.tartarus.org/texts/introduction.html as of July 28, 2015

```
Define $\rho_t \equiv (\tau_0 + t)^{-\kappa}$
initialise $\lambda$ randomly
for $t = 0$ to $\infty$ do
    E step:
    Initialise $\gamma_{tk} = 1$. (The constant 1 is arbitrary.)
    repeat
        Set $\phi_{twk} \propto \exp E_q[\log(\theta_{tk})] + E_q[\log(\beta_{kw})]$
        Set $\gamma_{tk} = \alpha + \sum_w \phi_{twk} n_{tw}$
    until $\frac{1}{K} \sum_k |$ change in $\gamma_{tk}| < 0.00001$;
    M step:
    Compute $\tilde{\lambda}_{kw} = \eta + D n_{tw} \phi_{twk}$
    Set $\lambda = (1 - \rho_t)\lambda + \rho_t \tilde{\lambda}$
end
```

Algorithm 3: Online learning for Latent Dirichlet Allocation

Table 1: Shows what each symbol in Algorithm 3 represents.

symbol	Definition
α	The hyper-parameter for the Dirichlet distribution θ
θ	The k-dimensional Multinomial parameter that specifies the topic distribution for a document
β	a $k \times V$-matrix where each row describes the probability of words for a specific topic
γ	The variational Dirichlet parameter
ϕ	The variational Multinomial parameter
λ	The variational parameter on β
η	The hyper-parameter for the Dirichlet distribution on β
κ	Controls the rate at which old values of λ are forgotten
τ_0	Slows down early iterations of the Online variational Bayes algorithm
k	Number of topics
D	Size of corpus ($D \to \infty$) in a true online setting [5]

modified to receive texts from a blocking queue instead of reading the documents at initialisation from a folder. This allows two programs to be coupled with any type of stream provider. Support for a stemmer was added.

When attempting to categorise new tweets with a profiler it showed that almost 80% of the run time was spent looking up words in the vocabulary. The reason for this is that the supplied Vocabulary in Java Online learning for Latent Dirichlet Allocation was a plain vocabulary that used a list as the underlying data structure. Using a list for looking up words yields a time complexity of O(n) and an average of (n/2); since the vocabularies contain upwards 50 000 - 200 000 terms this becomes quite expensive. To avoid this unnecessary computation a vocabulary using a trie data structure was used. With a trie the look up of a word is only dependent on the length of the word and not on the number of words in the data structure.

6.5.2 Solving Latent Dirichlet Allocation

The classification of a document could be described as reverse engineering the process of generating documents, formally called inferring the topic. Inferring the topic is done by solving the intractable equation [2] found in **Section 5.3**. Blei et al. use a convexity based variational inference. In short the Latent Dirichlet Allocation model is modified making it tractable to solve. The simplified model is optimised using Kullback-lieber divergence for which Jensen's inequality, also know as the secant rule, is used to calculate.

Jensen's inequality - generalises the fact that the secant of a convex function $f(x)$ is always above the graph and its integral is always larger than the integral of $f(x)$. In our case it means that if X is a random variable and ρ is a convex function then $\rho(E(X)) \leq E[\rho(X)]$. This is useful when trying to solve hard or intractable integrals. Given the two graphs in figure 9 it can be observed that the area under the secant lines, $secant_1$ and $secant_2$, is always larger than the area under the convex graph, $f(x)$. If the function is concave the same principle applies, only changing that the area beneath the secant is always smaller than the area under $f(x)$.

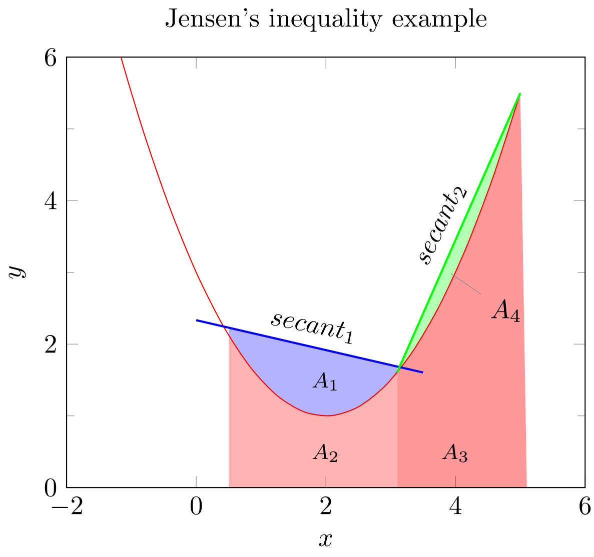


Figure 9: Jensen's inequality exemplified. As seen in the figure the area under $secant_1$ is $A_1 + A_2$ which is larger than the area under $f(x)$ which is A_2. The same goes for the area under $secant_2$ compared to the area under $f(x)$. This fact can be used to approximate the integral of $f(x)$ by dividing the graph into several secants and then calculating the integral for each secant and then summing them together.

This method can be explained using the equations in Eq 14. If $f(x)$ is the function that we want to approximate the integral for and $g(x_0, x_1, x)$ is the function defining the secant that intersect $f(x)$ at x_0 and x_1. Then we can divide $f(x)$ into smaller ranges and calculate the integral of the secant for each range.

$$\begin{aligned} f(x) &= x^2/2 - 2x + 3 \\ g(x_0, x_1, x) &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} x + f(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0} x_0 \\ \forall x_i, x_{i+1}; \int_{x_i}^{x_{i+1}} f(x)dx &\leq \int_{x_i}^{x_{i+1}} g(x_i, x_{i+1}, x)dx \end{aligned} \tag{14}$$

Kullback–Leibler divergence - is a way of measuring the data loss when using a simplified model Q to approximate another model P. It is denoted $D_{KL}(P||Q)$ however the KL can be omitted when the context is clear. The KL-divergence is used to measure the data loss when simplifying a model in order to allow for calculations on it. KL-divergence can be described mathematically for both discrete and continuous functions. Eq. 15 defines the discrete version while Eq. 16 defines the continuous one. Suppose that $Q(i) = 0$ for some i; this will render a division by zero which is invalid. The KL-divergence handles this problem by being defined only if $Q(i) = 0$ then $p(i) = 0$. This is valid because of the fact that $\lim_{x \to 0} x \ln(x) = 0$

$$D_{KL}(P||Q) = \sum_i P(i) * \ln \frac{P(i)}{Q(i)} \tag{15}$$

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} P(x) * \ln \frac{P(x)}{Q(x)} dx \tag{16}$$

Getting the integral for the KL-divergence will often be intractable due to the complex distributions of both P and Q. This is where the *Jensen's inequality* comes in handy. Even if the integral is intractable we can always input values for x to create secants that can very easily be integrated and summed together.

6.5.3 Creating the vocabulary

A Javascript application was written to generate a vocabulary given a file with a document on each line. Early experiments suggested that vocabularies older than a few days would make the algorithm perform poorly. However, in **Section 7.3** it is shown that this might not be the case. Comparing two vocabularies, the first one that was created 2015-06-09 and the second one created 2015-06-10 yielded the following results:

- The first and second vocabulary had **48 204** and **46 605** terms respectively
- The first vocabulary contained **37 402** terms not present in the second one.
- The second vocabulary contained **35 803** terms not present in the first one.

Additionally Table 10 shows the ten most common words in both vocabularies. As seen in the figure most words are the same or similar.

2015-06-09	2015-06-10
love	love
people	people
today	good
follow	follow
good	zaynmalik
time	today
happy	time
life	days
summer	life

Figure 10: The top ten words of two vocabularies generated one day from each other. One of the interesting words that came up in the later vocabulary is *zaynmalik*, which is a Twitter user. This word was not present in the first vocabulary.

7 Evaluation of Latent Dirichlet Allocation on Tweets

A test was set up to find the parameters that yielded the best measurements on a run with 200 000 tweets. The measurements were made with perplexity which Blei et al. and Hoffman et al. also used [2, 5].

The parameters used in the end were found by first empirically testing different values for each parameter. This led to the set of parameter ranges found in Table 2. A script was created that ran the system with all permutations of these parameters on the same 200 000 tweets. The program measured perplexity after each mini-batch together with the average perplexity and the standard deviation of the perplexity with a decaying window of 50 mini-batches. The best parameters were found in the run that yielded the lowest sum of average perplexity and standard deviation times three. All of these files can be found on the repository[5]

Parameters	min	max	step
Mini-batch size	20	100	20
tau	0.5	1	0.1
kappa	0.5	1	0.1
alpha	0.05	0.35	0.05
eta	0.05	0.35	0.05

Table 2: The ranges of the parameters used in script to find the optimal parameters.

$$OptArg = Min(AVG_{perplexity} + 3 * StDev_{perplexity}) \tag{17}$$

Equation 17 was used to pick the best combination of parameters. By weighting the standard deviation higher than the average yielded series with less variance and more uniform values for perplexity. The ratio between average and standard deviation was discovered empirically.

7.1 Perplexity

Perplexity is used to measure how well our model fits the data. It measures how well our model estimates the density of word occurrences in topics. The lower the perplexity score the better [2]. Blei et al. defines perplexity as

> "Algebraically equivalent to the inverse of the geometric mean per-word likelihood."

which can be written for a test set of M documents [2]:

$$perplexity(D_{test}) = \exp\{-\frac{\sum_{d=1}^{M} \log p(w_d)}{\sum_{d=1}^{M} N_d}\} \tag{18}$$

5. https://github.com/johanrisch/TwitterTopicMining

It is important to remember that perplexity is not an absolute metric. It can only be used to compare different models on the same data set. The reason for this is because the per-word likelihood of the model is compared to the per-word likelihood of the data used.

7.2 Perplexity from runs with optimal parameters found

Figure 11 to 15 show the perplexity, average perplexity and the standard deviation of perplexity output from the test runs. The parameters used to produce these values can be found in **Table 6**. Note that **Figure 11 to 14** have a maximum of 10 000 on the Y-axis while **Figure 15** has a maximum of 20 000.

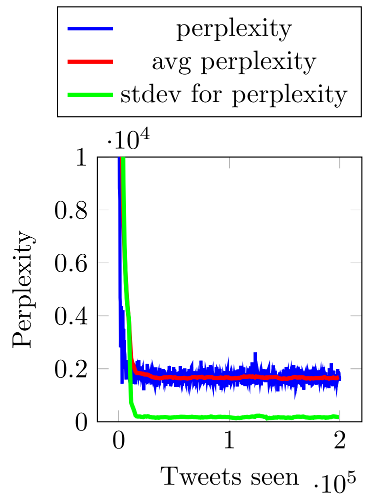


Figure 11: Test run data for 20 Topics

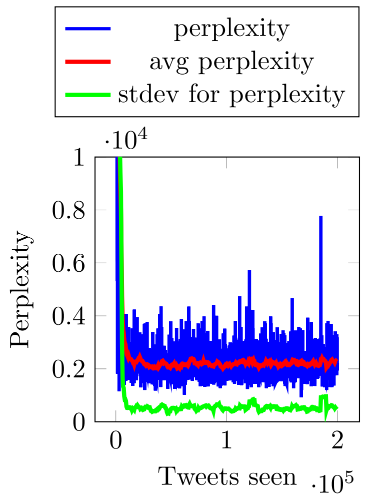


Figure 12: Test run data for 40 Topics

As **Figure 11 and 12** show both of these runs perplexity drops very fast after the first few mini-batches. Both series are close to a perplexity of 2 000 but the series with 20 topics has a more stable and lower standard deviation. The same trend can be observed when looking at the runs with higher topic counts in **Figure 13 to 15**.

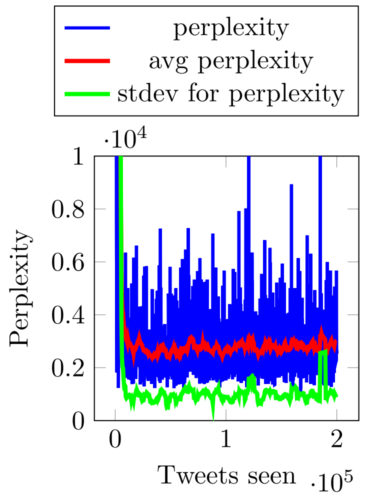


Figure 13: Test run data for 60 Topics

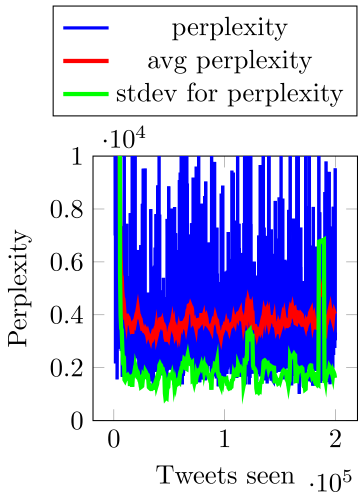


Figure 14: Test run data for 80 Topics

"

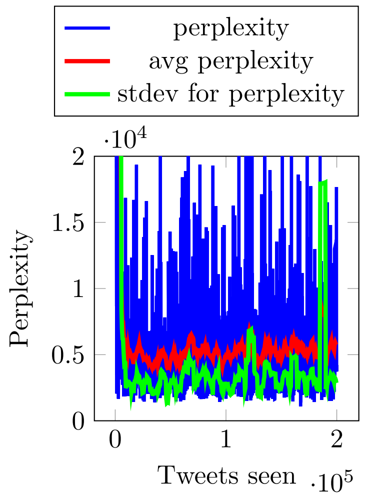


Figure 15: Test run data for 100 Topics. Note that this figure has a higher maximum on the Y-axis compared to the other figures.

The parameters used in **Figure 11 to 15** can be found in **Appendix A**.

7.3 Testing the ability of Latent Dirichlet Allocation to classify known topics

A second test was set up to test how well Latent Dirichlet Allocation could predict topics in tweets once trained. This was done by training the system on 170 000 tweets containing four major topics. Once the system was trained the *precision*, *recall* and *F-measure* were measured on a held-out test set of the data containing 28 394 tweets.

7.3.1 Creating a dataset

In order to create a test set containing four topics Twitter handles *(@-handle)* where used. An *@-handle* is a phrase that starts with an at (@) sign which any user can use to make their tweet visible when searching for that handle. These handles usually have a specific topic which they address. Four topics were collected by fetching approximately 50 000 tweets from four different *@-handles*:

1. **@espn** - as a topic about sports.
2. **@charitywater** - as a charity topic about water.
3. **@president** - as a topic about American politics concerning the president.
4. **@TechCrunch** - as a general technology topic.

When fetching tweets with an *@-handle* all the tweets received will contain the search term somewhere in the tweet. Therefore each *@-handle* used was added to the list of stop words. Once tweets from all *@-handles* were collected these files where combined[6] and each tweet was marked with the *@-handle* used.

Are these topics worth evaluating on? Since the tweets are not manually classified there might be some unrelated tweets inside each *@-handle*, therefore separate vocabularies were created for each *@-handle*. These vocabularies were compared to see how different each *@-handles* word frequency spectrum was. **Table 3** shows the percentage of unique words when comparing two vocabularies. Furthermore all words starting in *http* or digits where removed together with words that occurred less than 6 times.

	espn	charitywater	president	TechCrunch	Total no. of terms
espn	0	97%	96%	80%	3091
charitywater	59%	0	93%	57%	247
president	64%	95%	0	68%	328
TechCrunch	76%	96%	96%	0	2570

Table 3: Showing the percentage of unique terms when comparing the generated dictionaries. Row one **(espn)** and column 2 **(charitywater)** shows that 97% of the espn dictionary is unique when comparing it to the words in charitywater's dictionary. Note the difference in number of terms for charitywater and president compared to TechCrunch and espn.

As shown in **Table 3** the word frequency spectras are quite different from each other. An interesting note is that both *espn* and *TechCrunch* almost have ten times the number of terms compared to *charitywater* and *president*. Since

6. Combined file can be found at the Github repository https://github.com/johanrisch/TwitterTopicMining

the vocabularies were generated from approximately the same number of tweets it suggests that *president* and *charitywater* are more narrow topics compared to *espn* and *TechCrunch*.

This makes sense when looking at what topics these *@-handles* discuss. *President* and *charitywater* are both *@-handles* that discuss very specific topics, political news regarding the president of the United states of America and news about the charity foundation *Charity: water*.

Is this dataset adequate for evaluating the performance of Latent Dirichlet Allocation on a stream of tweets? One could argue that it is advantageous that the topics are of different width allowing the dataset to reflect some of the variance in the Twitter medium. However, the dataset only contains four major topics which is not representative of all the tweets posted. On the other hand, if this dataset is used with a vocabulary created specifically for the tweets in the dataset is compared to a vocabulary created for the perplexity measures in **Section 7.2** it will give us an indication on how important the vocabulary is for Latent Dirichlet Allocation to work properly on a stream of tweets.

7.3.2 Test design

Firstly the system was run with 4, 8 and 16 topics on the dataset separately. Once the model was trained on the dataset the held out data was classified. This was done in two iterations. In the first iteration each tweet was classified to a topic and the label of the tweet was added to the topic. After the first iteration the purity for each topic can be calculated. This is done by normalising the count of each label type in each of the 4, 8 and 16 topics; the purity is then the value of the label with the highest value. For each topic count there were test runs with different tokenisers, and level of spam injected into the stream.

Before iteration two each of the 4, 8 and 16 topics is labeled according to the dominant classification found when calculating the purity. However, when running the test on 4 topics charitywater never became the dominant one. **Table 4** shows the purity table together with the top five words for each topic.

Topic	Purity	Main label	Top 5 words
0	0.388	president	water, clean, hous, white, peopl
1	0.395	TechCrunch	report, vote, annual, tech, pretti
2	0.308	espn	app, googl, launch, livelokai, play
3	0.310	president	watch, meet, stewart, secret, amaz

Table 4: Purity table with the top five words for each topic for the test run with four topics. As can be observed president occurs two times and charitywater is missing. Since each label must have a topic one of the president topics will have to set to charitywater.

While *topic 0* has a higher purity for the label *president* as can be observed in **Table 4** the top two words for *topic 0* are *water* and *clean*. Therefore the label of *topic 0* was manually set to *charitywater*. On the test runs with 8 and 16 topics each label was represented and the purity label was used to label each topic.

7.3.3 Spam injection

Several executions with different amount of spam were run. In the first half of the runs both the training data and held out test set were injected with spam, and in the second half only the held out test set was injected with spam. By only injecting the held out test set with spam we effectively emulate previously unknown types of spam coming into the model. If the spam was present during the training the mode would recognise the spam; and thus a good metric on how Latent Dirichlet Allocation handles known spam versus unknown spam might be achieved.

The spam injected was taken from the data set used when measuring perplexity. The tweets contained in that data is a good mix of all types of tweets since there were fetched using the sampled Twitter streaming endpoint[7].

One important aspect with spam in this test is that we can not guarantee that the training set is completely spam free since they are automatically fetched and each tweet is not reviewed individually to assert that it is in fact a correctly labeled tweet. Therefore the percentage of spam in **Figure 17** and **Figure 18** should be seen as a minimum amount of spam in the data set. In other words; if the percentage of spam is set to 50% at least 50%, maybe more, of the tweets processed were spam.

7.3.4 Precision, recall and F-measure

Precision, recall and F-measure will be calculated in the same way that Rajagopal et al. do in their paper [10], the formulas used can be found in Eq 19. Before precision, recall and F-measure is calculated true positive, T_p, true negative, T_n, false positive, F_p, and false negative,F_n, must be defined.

1. T_p - When a tweet with label A is assigned to a topic with label A
2. T_n - When a tweet that is not labeled A is not assigned to a topic with label A
3. F_p - When a tweet with label A is assigned to a topic that is not labeled A
4. F_n - When a tweet with label A is not assigned to a topic with label A

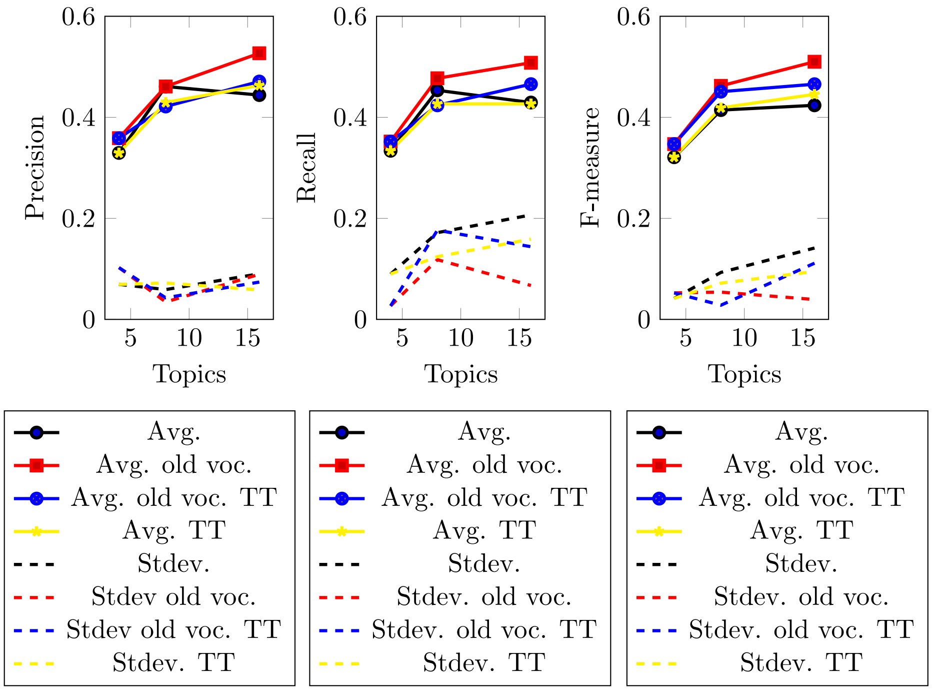
$$\begin{gathered}\text{precision} = \frac{T_p}{T_p + F_p} \qquad \text{recall} = \frac{T_p}{T_p + F_n} \\ \text{F-measure} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}\end{gathered} \tag{19}$$

7. https://dev.Twitter.com/streaming/reference/get/statuses/sample accessed September 2015

7.3.5 Test result

The precision, recall and F-measure were recorded and put into **Appendix B**. The exact data used in these figures can also be found in **Appendix B**. The labels could be grouped into two groups, the first one could be TechCrunch as the only label to get better performance with 16 or 4 topics; and the rest in the second group. The second group had a peak performance with 8 topics.

In order to test how dependent Latent Dirichlet Allocation is on an up-to-date vocabulary the test was run with the dictionary used when measuring perplexity from **Section 7.2**. The results from this run can be seen in **Figure 16**. The raw data used in these figures can be found in **Appendix B**.



(a) Precision (b) Recall (c) F-Measure

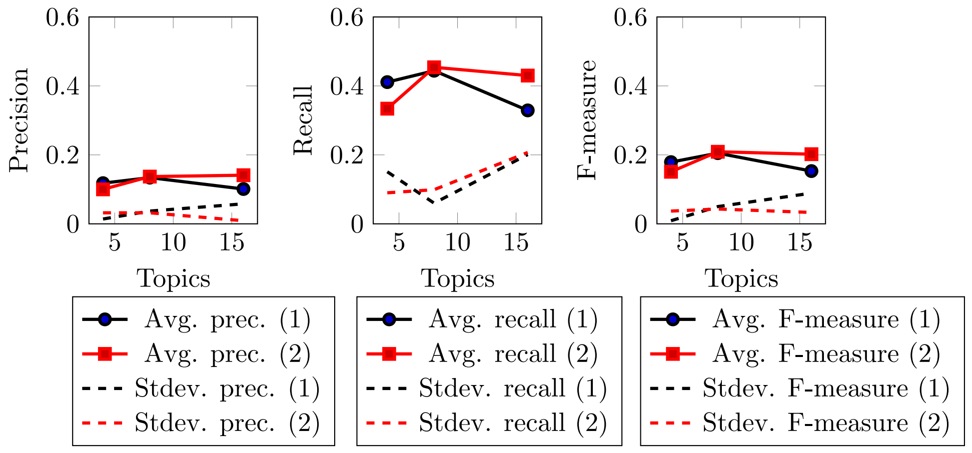
Figure 16: **Figure 16a to 16b** compares the average precision, recall and F-measure together with the standard deviation when using the vocabulary generated from the data set compared to when using the old vocabulary generated from the perplexity test data. Furthermore the test was ran using a basic tokenizer splitting the tweets into tokens on all non-alpha-numeric characters and also using a tokenizer specialized on tweets *TT*

Comparing vocabularies When comparing the results from the vocabulary generated specifically for the test set with the old vocabulary there are some interesting observations to be made. The average precision, recall and F-measure over all the labels for each topic count is consistently higher or equal to when using the old vocabulary compared to the one created for the data set. Standard deviation is lower for precision, recall and F-measure in all topic counts except for precision and F-measure with 4 topics.

One could argue that the old vocabulary generated from the perplexity test data is not an issue from the results in **Figure 16**. The vocabulary generated over the actual data-set, which contains all words present in the data set, performed poorer than the old vocabulary. Of course, there is one issue that remain with an old vocabulary; a model using it will not be able to pick up new phrases and words that might become one of the defining words for a topic.

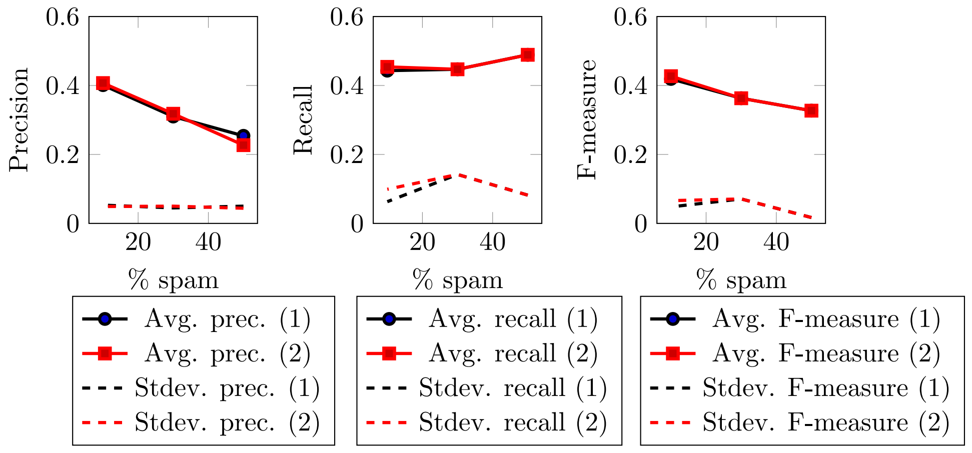
Measuring the effect of spam on Latent Dirichlet Allocation. The data in **Figure 17** and **Figure 18** is quite expected. The F-measure and precision is directly affected by the amount of spam in the stream. However, the recall is not as affected as F-measure and precision. Recall that precision was defined as the number of correct classifications, *true positives*, divided by the number of *true positives* with the number of *false positives.* Each spam message added will be classified as one of the four topics which leads to a *false positive.* This means that the precision should be directly correlated to the amount of spam injected in the stream; which can be seen in **Figure 18a**. Since F-measure is a weighted measurement between recall and precision it makes sense that spam decreased the F-measure value as well.

What at first could be called unexpected is that the recall did not get affected much by the percentage of spam injected. **Figure 17b** shows the that the recall is much more affected by the number of topics compared to the amount of spam injected. **Figure 18b** shows us that recall roughly stayed the same, the standard deviation of the recall on the three data points is 0.22, no matter what amount of spam was injected. Recalling the definition of recall which was the number of *true positives* divided by the sum of number of *true positives* and *false negatives* explains this phenomena; since spam will only affect the test outcome by adding *false positives* which is not included in the calculations of recall.



(a) Precision (b) Recall (c) F-Measure

Figure 17: **Figure 17a to 17c** compares the impact of spam on precision, recall, and F-measure. (1) means that the model was trained on a data set containing spam while (2) means that it was trained on a spam free data set. Both (1) and (2) used a held out test set where 70% of all tweets were injected spam



(a) Precision (b) Recall (c) F-Measure

Figure 18: **Figure 18a to 18c** compares the impact of spam on precision, recall, and F-measure. (1) means that the model was trained on a data set containing spam while (2) means that it was trained on a spam free data set. The model used eight topics in each test run and the spam was decreased by 20% between each test run.

7.4 Throughput of the system

While the theory of how well Online learning for Latent Dirichlet Allocation is covered in **Section 3.2** the number of documents processed per second was recorded for 2, 20, 50, 100, and 200 topics. The measurements was made on a 2014 Macbook pro retina 13" with a 3 GHz Inte Core i7 and 16GB 1600 MHz DDR3 ram. **Table 5** lists to number of tweets processed per second for each topic count and as it shows it does really scale linearly with the number of topics. However, when looking at how many tweets are posted per second, roughly 10 000 tweets per second[8], it becomes clear that it might still be a problem. Read more on this in **Section 8.2**.

Table 5: Table showing number of tweets processed per second for 2, 20, 50, 100 and 200 topics.

Number of topics	Tweets processed per second
2	5556
20	1176
50	520
100	243
200	132

8. http://www.internetlivestats.com/one-second/ can be used to check, september 2015 there usually was around 10 000 tweets per second

8 Discussion and Future work

While both the perplexity test and the precision, recall and F-measure tests yielded promising results. There are issues remaining when using Latent Dirichlet Allocation on a stream of tweets.

8.1 Removing spam

Since spam affected the performance of Latent Dirichlet Allocation it is important to be aware of this when using the model in an application. If the application is tolerant to *false positives* less effort could be made when trying to remove spam from the stream compared to an application that is very sensitive to *false positives*. One could consider leaving room for spam in the model by expanding the number of topics in the model to allow for some of the topics to detect spam messages.

8.2 Acquiring high enough throughput

The number of topics in all the tweets posted is probably a lot higher than 100 topics. When the system ran the perplexity tests on 100 topics it managed to process around 330 tweets per second. While the algorithm used fits all the criteria needed to be called a streaming algorithm it is still quite slow and the stream of tweets would have to be reduced into a size that the model can handle.

One approach would be to query the Twitter streaming api with phrases that would limit the number of topics returned in the stream. This could be done similar to the test in Section 7.3. However, the work needed to get the model to work on a data set with four topics was cumbersome and it would be hard to get the model working on a new set of topics without a lot of manual work.

If a hierarchy of computers were used to successively split up the stream in smaller and smaller chunks, the system might be able to handle the throughput of Twitter. Imagine that the first level in the hierarchy simply filtered away spam and sent the tweets labeled as not spam further down the hierarchy. The next level could split the stream into two different topics, *Science* and *Not science*, which in turn would be sent down one more level to two separate systems that in turn would split up the incoming topics. The results from Section 7.2 shows that Latent Dirichlet Allocation converges faster with lower topic count which further supports this approach. As long as each level can handle the incoming stream this set-up should be able to model a large stream with a lot of topics. The drawback would be that the latency to get the identified topics would increase since there will be a transmission time between each level in the hierarchy.

8.3 Tweets that link to other websites

One other problem with this approach is how many tweets are structured. Many tweets are just a short comment followed by a link. These links is what would define the topic in the tweet but the cost of fetching these links while mining the stream would probably take too much time. A possible solution to this could be to attempt to identify as many main topics from top domains as possible and simply skip those which there is not enough time to process. This approach could probably work quite well when the links are to specialised web pages that focus on one or a few topics.

8.4 Looking at the sender of a tweet

The goal was to evaluate Latent Dirichlet Allocation when modelling tweets using only the textual body. However, tweets contain a lot of meta data such as the user who wrote it. For instance, the probability of *@espn* to post something about the house prices in Sweden would probably be very low. So a user who follows *@espn* and other users that are classified as sports in the US posts a tweet chances are higher that the topic is related to sports rather than house pricing in Sweden.

This can be done by analysing a Twitter user's follower and who they follow. Suppose a follower is treated as an incoming link and a follow as an outgoing link; and we apply a page index algorithm such as the HITS algorithm the users who were classified as the biggest authorities could be manually classified if they have a very specific topic they address. Once these users have been classified users could be given a topic distribution based to the users they follow by measuring how far away in the graph they are from the manually classified users. This could in turn be used to weight the distribution of topics on a tweet from a specific user.

In the beginning of this thesis this was attempted. However, due to the limitation in requests of user profiles in the Twitter API it was not feasible to create the graph of users. The Twitter API has a rate limit on how many requests of user profiles are allowed within 15 minutes which made it infeasible to create a large enough graph to use it.

8.5 Conclusion

In **Section 7** it is shown that an outdated vocabulary does not necessarily mean that Latent Dirichlet Allocation will perform poorly. However, there are a few issues that remain when mining a stream of tweets using Latent Dirichlet Allocation. Since Twitter contains numerous topics the throughput of an online version of Latent Dirichlet Allocation that tries to model every topic might be too low. In **Section 8.2** an outline of a hierarchal system structure could allow for a larger throughput at the expense of higher latency.

Future work could be done to improve the use of Twitter meta data, such as extending the model to use the sender of the tweet. As discussed in **Section 8.4** one could use page-indexing on the user graph to find authorities on different topics. These authorities can be used to modify the topic distribution of a tweet based on how far its sender is from these authorities.

References

[1] Michael Berkovsky. https://github.com/miberk/jolda.

[2] David Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 2003(3):994–1022, January 2003.

[3] David M. Blei and John D. Lafferty. A correlated topic model of science. *Ann. Appl. Stat.*, 1(1):17–35, 06 2007.

[4] Daniel Gayo-Avello. "I Wanted to Predict Elections with Twitter and all I got was this Lousy Paper" – A Balanced Survey on Election Prediction using Twitter Data. *arXiv:1204.6441 [physics]*, April 2012. arXiv: 1204.6441.

[5] Matthew Hoffman, Francis R. Bach, and David M. Blei. Online Learning for Latent Dirichlet Allocation. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 856–864. Curran Associates, Inc., 2010.

[6] Alexander Sjöberg Kim Wedenberg. Online inference of topics. Technical Report UPTEC F14010, IT, Uppsala, January 2014.

[7] Julie Lovins. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics,*, 11:22–31, June 1968.

[8] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam Filtering with Naive Bayes - Which Naive Bayes? Mountain View, California USA, July 2006.

[9] Brendan O'Connor, Michel Krieger, and David Ahn. TweetMotif: Exploratory Search and Topic Summarization for Twitter. May 2010.

[10] Dheeraj Rajagopal, Daniel Olsher, Erik Cambria, and Kenneth Kwok. Commonsense-Based Topic Modeling. *ACM 978-1-4503-2332-1/13/08*, August 2013.

[11] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

[12] Twitter. https://dev.twitter.com/streaming/overview, May 2015.

[13] Ke Zhai, Jordan Boyd-Graber, Jordan Boyd-Graber, and Jordan Boyd-Graber. Online Latent Dirichlet Allocation with Infinite Vocabulary, 2013.

[14] Xiaoming Zhang and Zhoujun Li. Automatic topic detection with an incremental clustering algorithm. In FuLee Wang, Zhiguo Gong, Xiangfeng Luo, and Jingsheng Lei, editors, *Web Information Systems and Mining*, volume 6318 of *Lecture Notes in Computer Science*, pages 344–351. Springer Berlin Heidelberg, 2010.

A Optimal parameters found from script

Parameters	20-topics	40-topics	60-topics	80-topics	100-topics
Mini batch-size	200	100	100	100	100
tau	0.9	0.9	0.9	0.9	0.9
kappa	1.0	1.0	1.0	1.0	1.0
alpha	0.15	0.15	0.15	0.15	0.15
eta	0.15	0.15	0.15	0.15	0.15
Avg. Perplexity	1768	2340	2951	3992	5723
St. Dev	937	1636	2791	6857	9471

Table 6: The optimal parameters found in the test runs described in Seciton 7. The best parameters where found by running the program over the same set of 200 000 tweets and varying each parameter between specified ranges. The best parameters where selected as the combination that yielded the lowest average perplexity and Standard deviation combined after 200 000 tweets.

B Tables for precision recall and F-measure test

Topic	Label	Precision	Recall	F-measure
0	charitywater	0.241	0.451	0.314
1	TechCrunch	0.395	0.339	0.365
2	espn	0.309	0.235	0.267
3	president	0.373	0.310	0.338

Table 7: Result for the test when run with **4 topics**. Note that the manually set topic was the one with the highest recall.

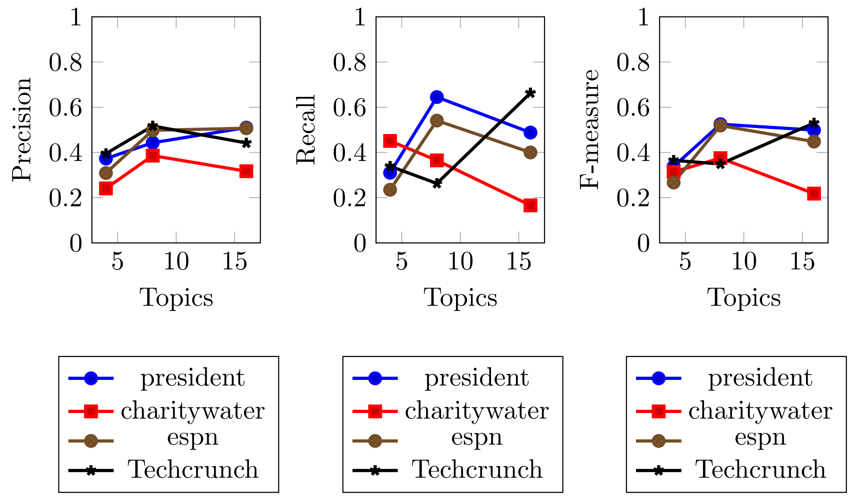
Topic	Label	Precision	Recall	F-measure
0	charitywater	0.386	0.365	0.375
1,6	espn	0.498	0.541	0.519
2,3,5,7	president	0.443	0.645	0.525
4	techcrunch	0.517	0.263	0.349

Table 8: Result for the test when run with **8 topics**. The recall is overall higher.

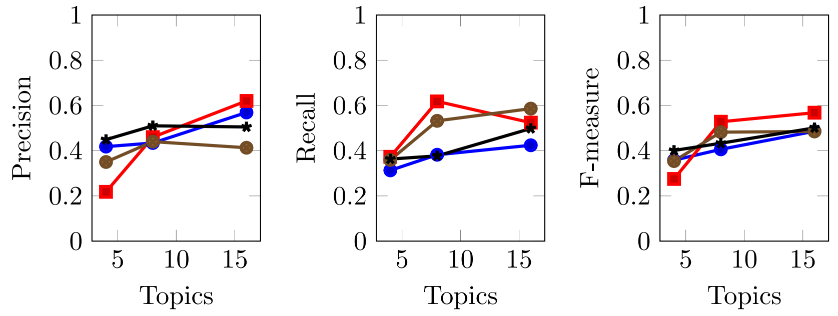
Topic	Label	Precision	Recall	F-measure
6	charitywater	0.317	0.166	0.218
1,2,3,9	espn	0.507	0.400	0.448
0,10,11,13	president	0.510	0.488	0.499
4,5,7,8, 12,14,15	techcrunch	0.442	0.663	0.530

Table 9: Result for the test when run with **16 topics**. The recall is overall higher.

B.1 Graphical representation of the test result



(a) Precision (b) Recall (c) F-Measure



(d) Precision old vocabulary (e) Recall old vocabulary (f) F-Measure old vocabulary

Figure 19: 19a show the Precision for each of the known topics, 19b show the recall and 19c shows the F-measure.