Latent Dirichlet Allocation: Providing recommendations in a world of spam

Haris Kodžaga

Vrije Universiteit, Amsterdam, The Netherlands
h.kodzaga@student.vu.nl

Abstract. Recommendation systems have become an increasingly popular solution for online services to get users what they want. This paper will look into the feasibility of Latent Dirichlet Allocation as a recommendation system that is determines what is relevant based on content only. This means that it will return recommendations relative to the document it is processing at a particular moment. The use case of this implementation is SlideWiki where a significant proportion of spam has been detected. In order to combat this abundance of an ensemble consisting of Naïve Bayes classification and LDA has been built for spam filtering. The efficacy of the ensemble as a spam filter is compared to the individual models that made up the ensemble. Furthermore, the paper will look into the effect spam filtering has on the effectiveness of the recommendation system to return relevant recommendations. The dataset used for experimentation is an English-only subset of roughly 17K presentations that is derived from SlideWiki's scraped database of presentations. **Keywords:** Latent Dirichlet Allocation, recommendation system, spam classification

1 Introduction

The internet is a seemingly endless source of information, however most of the information on the internet could be best described as a huge mountain of books regarding structure. The wealth of information might be plentiful, but how can one find something that is relevant to what he/she is currently interested in? Among other solutions, recommendation systems have been developed to help users to filter through the clutter and provide content that is likely relevant to a user's information need. Real-world applications of such recommendation systems have become common and critical [1] for online services such as YouTube,[1] Netflix[2] and Facebook[3] in fact.

This paper will look into Latent Dirichlet Allocation as a possible method for content recommendation. With LDA an X amount of topics can be specified to create a model of topics which each contain keywords that are typical to a particular topic. A document can then be processed through the model and its matches with the topics in the model will be returned [2]. When processing a corpus through a LDA model the output could

[1] https://www.youtube.com/
[2] https://www.netflix.com/
[3] https://www.facebook.com/

be best compared to weighted soft-clustering [3]. A document can belong to more than one cluster (topic) and its associations with the clusters is expressed with a decimal number rather than a Boolean value. Compared to weighted soft-clustering, perhaps the biggest advantage with LDA is that the topics generated are more informative than mere clusters. This is especially useful when there is no prior knowledge about the corpus. The LDA model allows for a more precise evaluation loop where one can evaluate the output and decide whether the topics and their respective keywords make sense and tweak the model where necessary.[4] Furthermore, LDA has been noted to work particularly well with processing text [4]. Given that only the text of presentations is being processed in this project, LDA seems to be a good fit for this use case as well.

The use case of the proposed recommendation method will be SlideWiki.[5] The aforementioned mountain of books is replaced by decks of slides (presentations) and the goal is to categorize these decks in their own figurative library where related presentations (decks of slides) can be easily found given a presentation (deck of slides). SlideWiki is a slide-sharing platform which could best be compared to GitHub,[6] but for presentations instead of code. On SlideWiki users can create slides, share them with other users, build upon existing slides on the platform (equivalent to forking on GitHub) comment on slides, make annotations et cetera.[7] SlideWiki as it currently stands has around 27K decks, however a large proportion of these decks do not hold much relevant content with most of this proportion being spam.

The aim of this paper is to look into the feasibility of creating a recommendation system with the help of Latent Dirichlet Allocation. Three spam filters will also be compared in terms of their efficacy as a spam filter for SlideWiki and their impact on the recommendation system's efficacy will be evaluated as well. The following three implementations will be compared: Naïve-Bayes spam classification, spam classification using LDA and an ensemble which combines the two models. The implementation is completely done in Python. This is due to the familiarity with the programming language, as well as the availability of packages, such as Numpy[8] (contains useful mathematical computations) and scikit-learn[9] (machine learning package containing various models to work with, e.g. Naïve Bayes and LDA). The dataset for this project contains around 17K decks, after filtering small decks and non-English decks from the original 27K deck dataset.

This paper will try to answer the following two research questions:

- How can a recommendation system be built that provides relevant suggestions relative to the content consumed at a particular moment in the domain of SlideWiki?
- What model will be most effective in dealing with spam from SlideWiki's collection of decks?
- Will spam filtering aid the efficacy of the recommendation system?

---

4 https://canvas.vu.nl/files/356218/download?download_frd=1
5 https://slidewiki.org/
6 https://github.com/
7 https://slidewiki.org/presentation/84941-3/slidewiki-guide-introduction/84941-3/
8 http://www.numpy.org/
9 http://scikit-learn.org/stable/

The assumption is that the Naïve Bayes with LDA ensemble will prove to be the best answer to both research questions. First of all, the ensemble will likely prove to be the most effective spam filter and secondly, the belief is also that a corpus with less noise (spam in this case) will lead to more relevant topics to be generated by the LDA model and thus result in more relevant recommendations.

Regarding the usage of Latent Dirichlet Allocation as a way to build a recommendation system, it should likely prove to be effective as well. First of all, there is no prior knowledge about all the subjects covered on SlideWiki. This makes LDA comparatively easier to implement to other models such as clustering, due to the topics with keywords that are generated. As a result, evaluating and tweaking the model is easier to do compared to more traditional clustering models for example.[10]

Furthermore, since the total span of subjects of SlideWiki's presentations is unknown, there is neither any prior quantitative information on SlideWiki's presentations that can describe the presentations. That is, there is no quantitative description on a presentation that is for example about clustering for an algorithm to actually see the presentation as clustering. LDA is able to provide documents such descriptive values such that these documents can be linked together based on ranked retrieval through content [5]. LDA will likely thus also be a good basis for a recommendation system that also allows for further augmentations to the recommendation system to come. All in all, a ranked retrieval recommendations just seem as a logical basis for the project and also seems more feasible within the scope of this project.

2 Background information

This section will briefly explain the theory behind the models implemented in this project. Related work contains some papers that go into more detail about the aforementioned subjects, in the case if the following explanations about any of these subjects not be sufficient and/or satisfactory.

2.1 Latent Dirichlet Allocation

Without going in too much detail, Latent Dirichlet Allocation can be best compared to weight soft-clustering in terms of what it does. Documents are clustered based on their content, however these clusters are not exclusive. This means that a document can belong to more than one cluster. Furthermore, since the clusters are not exclusive, a document's association with a cluster is expressed with a decimal between 0 and 1, rather than a Boolean value. In the end, these associations are supposed to sum to 1 however. Thus if a document for example has an equal association with two clusters, the score for these clusters given this document will be 0.5 for both clusters (which sums to 1).

Where LDA differs from just clustering first of all is that instead of clusters, there are topics and every topic has keywords called labels that are typical to the topic. One

[10] https://canvas.vu.nl/files/356218/download?download_frd=1

such example of a topic is the following: *topic 0: clustering cluster data method algorithm object feature set point similarity measure distance graph using cont subspace number matrix based model*

This topic has clear associations with clustering, since it contains labels that are typical to clustering. Should a document that is largely about clustering be classified be classified by a LDA model which contains this topic 0 above, then it will likely have a high association with this topic (score of 0.7 or higher for example). Given that that LDA is able to create such topics with labels that describe the topic, it makes optimization rather easy and intuitive. One can evaluate the topics on coherence. That is, simply evaluate whether it makes sense for the labels of a topic i to be in a set together. The challenge in this form of evaluation however, is that the one who is evaluating a topic actually has to be familiar with the labels in a topic. A computer scientist will likely have difficulty with determining whether a topic which is about e.g. philosophy actually has coherent labels or not.

Now, the largest distinction between Latent Dirichlet Allocation and clustering however, is that LDA is not clustering at all actually. Technically speaking Latent Dirichlet Allocation is a multi-class classifier, since before any documents are assigned to topics, a model is fitted with the help of a training set. As apposed to usual procedure, a test and validation set is not required. In fact, the target set itself can be the training set that fits the LDA model. So is also the case in this project. The entire dataset of SlideWiki is used to train the model, just to classify the same documents in their respective topics that were used to train the model.

The end result is thus a model that contains k-number of topics that contain labels that are typical to their respective topics. Processing a document through the LDA classifier will then distribute scores between 0 and 1 for all topics where the sum of these scores is equal to 1. These scores express the document's associations with the topics and a higher score on a topic means a higher association with a topic. The following is an example of how this LDA classifier is applied to determine associations over topics given the document 131: *'131':{'topic_16': 0.38630525854011871, 'topic_21': 0.58348096819024797}*. It should be noted that this distribution over topics does not sum to 1, since the LDA model used in this project always assigns some score to a topic, but this score is not always significant. For example, a score of 0.01 on a topic means an association of 1% which is not particularly relevant, thus those scores have been filtered out.

2.2 Naïve Bayes classification

Naïve Bayes classification is probabilistic text classification based on conditional probability. In other words, it estimates a class for e.g. a document based on examples it had seen before and it will pick the class that returns the highest probability. The fact that it is a supervised model is also obvious, since the "examples it had seen before" are labelled documents from a training set. The labels on the documents from the training set correspond to a class one wishes the classifier would classify a similar target document in.

A typical example where Naïve Bayes is being applied, is the task of spam classification (which happens to be a task in this project as well). Before classification or even training occurs, a dataset has to be prepared first. This is done by taking some samples from the target dataset (decks from SlideWiki in this case), manually evaluate the contents of the document and provide the documents with the appropriate labels. In the case of spam classification, the classes are "ham" and "spam" and these will thus be the labels as well. After a set of labelled example documents has been created, training can be performed. Naïve Bayes then trains its classifier by looking at relative occurrence of words that are present in the respective labelled datasets. This relative occurrence is expressed as a probability, i.e. the likelihood of a word to be "ham" or "spam."

In the use case of SlideWiki, there was a large prevalence of presentations that were about Facebook customer support. Decks such as these were considered as spam and in the training set they were labelled as such. Training thus resulted in words such as "facebook", "customer" and "support" to have a relatively higher probability to be spam than to be ham. The classification of a deck of slides is in the end performed by calculating the probability that a deck of slides is ham given the words it contains and also calculating the probability that a deck of slides is spam given the words it contains. The classifier will then pick the class which returns the highest probability. In the case of a presentation which is about Facebook customer support, the deck of slides will be classified as spam, since the probability given the words contained in the deck of slides is higher for spam compared to ham.

2.3 Related work

While Latent Dirichlet Allocation was briefly explained above, those wanting a more technical description of the model can read the paper on Latent Dirichlet Allocation by David M. Blei et all [2].

More on Naïve Bayes classification can be found in Peter Flach's book on Machine Learning in chapter 9 [6]. It provides a more in-depth explanation on namely the mathematics that go begin Naïve Bayes classification.

3 Recommendation system using LDA

The decision behind choosing Latent Dirichlet Allocation started with an interview with assistant professor Tobias Kuhn. He explained that LDA might be a nice start as a sort-of clustering method. Furthermore, given the fact that there was no prior knowledge on all subjects present on SlideWiki, he added that LDA may be useful for this particular reason as well, since LDA's model space contains k-number of topics with their respective labels that are typical to them. This not only makes evaluating and tuning the model more intuitive by simply judging whether a topic's set of labels is coherent (i.e. it makes sense for the labels to be grouped together), but it first and foremost gives sort-of overview of all the subjects that occupy the model.

Other reasons for why Latent Dirichlet Allocation can be feasible as a recommendation system have already been briefly explained in the introduction of this paper. It is a

classification method that works particularly well with text content [4]. Furthermore, it performs classification by assigning scores to documents for each topic. These scores express a document's associations with the individual k-number of topics in the LDA model. In other words, the LDA model provides descriptive, quantitative values to documents [5] which can in turn be used to link documents together based on their associations with the topics in the model. It gives some quantifiable descriptions on documents that previously were not present in the domain of SlideWiki. Finally, implementing Latent Dirichlet Allocation for a ranked retrieval recommendation system seems as logical basis. A ranked retrieval recommendation system will always be able to provide recommendations, even during complications such as cold start problems [5]. Cold start problems describe the event when a new user registers on e.g. a website and providing personal recommendations is difficult, since there is little known about this new user [7]. The LDA recommendation system negates these issues, since it does not look at user preference at all, it merely links documents together in terms of similarity in content [5].

When considering a recommendation system for a website one requirement it definitely has to meet is an ability to provide recommendations quickly. This means in SlideWiki's case when a user opens a deck, the recommendations have to be loaded along with the rest of the webpage. In order to provide such recommendations quickly, two things can be optimized: computational power and ease of computation. Given that improving computational power is not always the most straightforward solution, improving the ease of computations seems more feasible. In this project it is done by dividing the task of providing recommendations, such that the heavy lifting is done beforehand in order to make fast recommendations achievable, since the final computations for the recommendation system are rather easy.

Taking the resulting database of the ensemble spam filter as an example, 3781 decks of slides are processed in this project. Assuming that every deck is unique (not a duplicate of another), 3781 different sets of recommendations can be returned by the recommendation system. While every recommendation is potentially different, all the recommendations from the set of 3781 decks have one major thing in common. They are namely all based on the same LDA model. This means for any given deck, the recommendation system has to look up with which topics a deck is associated with and which other decks are related with the target deck in terms of similar associations over these same topics. However, in order to find out which decks are related to the target deck, for every single deck of the entire collection of decks it has to be determined with which topics a deck is associated as well.

Performing these heavy calculations of finding out to what topics all the decks of the collection are associated with (classification) beforehand reduces the final search of related decks to be recommended to simple lookups that are thus considerably faster and more feasible to run real-time on demand. As a result, the task of providing recommendations is split in two parts. Namely, the score builder and the recommendation system itself. The score builder performs classification on the set of decks, creates sets of topics which are associated with a deck for every n-number of decks and it creates sets of decks that are associated with a topic for every k-number of topics. The

recommendation system just searches through these sets to find possible decks to recommend for a given target deck based on overall similarity to the target deck. The implementation of the recommendation system along with the spam filter models can be found on the GitHub page.[11]

3.1 Recommendation system: Score builder

As mentioned in the previous section, the recommendation system is split in two parts. This part will discuss the preparations which are performed by the score builder. The LDA model is from the scikit-learn[12] package for Python. The Medium article from Aneesha Bakharia on Topic Modeling with Scikit Learn has been used as a starting point of which the training implementation is based on.[13] The article shows how a LDA model can be trained when providing it a dataset and how the LDA model returns topics with labels for each topic after it has been trained. Everything in the implementation beyond training the LDA model has been built from scratch.

The first step is thus to train the LDA classifier by providing a training set. In the case of this project, the training set is the target as well. In other words, the target dataset trains the classifier so that the classifier can classify the documents in the target set. Before training running the training there are some parameters that have to be considered relative to the target set. In the case of this project the following parameters have been set:

- Number of topics has been set to 50
- The model will run 2500 iterations
- Learning offset has been set to 10
- Batch size has been set to 1024

These parameters were optimized for the 17K decks dataset where no spam filtering has been applied. This means that the LDA spam filter and the baseline recommendation system (no spam filtering applied) run optimally. The LDA spam filter in the ensemble and the recommendation systems where spam filtering have been applied do not run optimally under these parameters, since they have smaller target sets that differ from the 17K dataset (due to spam filtering). Parameters have been kept constant across all models for the sake of enforcing consistency during experimentation. After training, the LDA model is able to classify documents in the target set by assigning it scores for each topic. An example of how a topic can look like is the following: *Topic 3: data clustering cluster object set method point feature attribute analysis value distance similarity using algorithm measure number matrix hierarchical graph.*

[11] *https://github.com/bosnyan/Bachelor-Project-SlideWiki*
[12] http://scikit-learn.org/stable/
[13] https://medium.com/mlreview/topic-modeling-with-scikit-learn-e80d33668730

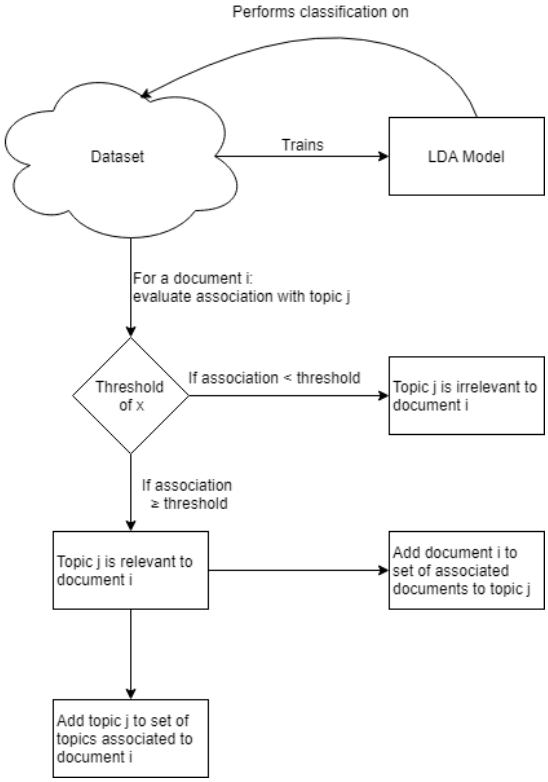

Fig. 1. Visualization of the process behind how the score builder works

It is rather clear that topic 3 is trained to have high associations with documents that are about clustering, since it contains labels that are typical to clustering. It would make sense for a documents that has absolutely nothing to do with clustering to have an association of 0 with topic 3. However, the LDA classifier from scikit-learn will always return a score higher than 0 for every single topic given a document. Some of these scores can get as low as 8.43881857e-04, which translates to a match of roughly 0.09%. Such matches are clearly irrelevant and thus a threshold has been set that a topic has to meet in order for the topic to be considered to have a meaningful association with the target document. The threshold has to be set relative to the number of topics, since a higher number of topics allows for the likelihood of a document's association to be split over a high number of topics. In this case where the number of topics has been set to 50, the threshold has been set to 0.05.

Should for a given document the association with a topic be higher than the threshold then the topic is saved in a set of topics that are associated with the document and the document is saved in a set of documents that are associated with the topic. After classification has been finished (all the documents in the set are processed), there are two collections of sets that will be used for cross searching related documents in the recommendation model. There are n-number of sets of topics that have associations with a document i in n and k-number of sets of documents that have associations with a topic

j in k. When classifying a set of SlideWiki's decks the output for a deck ID 101888 and topic 6 look as follows. *'101888':{'topic_6': 0.40411581834506022, 'topic_10': 0.10530399922914256, 'topic_11': 0.42686432288928561}.* Here can be seen that the set of deck ID 101888 contains three topics. In other words, the deck is associated with three different topics. *'topic_6': {'101661': 0.10342629284133516, '101685': 0.32628935596278791, '101864': 0.39136141755081905, '101888': 0.40411581834506022, ...}*. The set of topic 6 contains among other deck 101888, but also several other decks that have an association with topic 6. Aside from the deck ID, every deck also contains the score that expresses its association with topic 6.

Thus in the case of SlideWiki's decks, there are n-number of sets of topics that have associations with a deck ID i in n and k-number of sets of deck IDs that have associations with a topic j in k.

3.2 Recommendation system: Providing recommendations

After the score builder has finished, the recommendation system can provide recommendations based on similarities in topic associations between documents. Due to the fact that the heavy lifting has been done beforehand with the score builder, the relatively easy computations performed by the recommendation system allow it to return recommendations in an instant.

Given a document ID i, the recommendation system looks at which topics are associated with the document and what the respective scores are for each topic. The system will then loop through each topic associated with the document. For every associated topic it will do the following:

- Take the association score of the topic with document ID i and use it as a target value
- In the set of decks associated with the current topic:
    - Perform a confidence interval
    - Sort to lowest scores first
    - Discard first 10 results (to avoid duplicates)
- Save this order of documents as potential candidates to be recommended to the target document ID i
    - Check whether a document has been detected before in the loop (thus association with more than one topic)
    - If yes, sum the old total score of the document with current topic score of the document and divide it with the current count (e.g. 2)

After looping through every topic associated with document i, there is a list of potential candidates that could be recommended to document i as being related. These candidates have an intermediate confidence interval score of which the final score still has to be computed. The calculation for the final score heavily favors matching over the most number topics that are associated with the target document i. For example, imagine if a document i has associations with 5 different topics and a potentially recommended document has associations with 2 just of the same topics, while another document has associations with all 5 of the same topics. The latter will much more likely be selected as a recommendation for document i due to its match over all 5 topics. This approach has

been chosen, since selecting on the lowest confidence interval alone does not yield the best recommendation. Assuming matching the number of topics to be of equal importance as the confidence interval can lead to situations where an irrelevant document is recommended over other relevant documents due to it having an extremely low confidence interval on a single topic.

In order to negate these irrelevant recommendations, the emphasis has thus been set on heavily favoring documents that match over the most number of topics to the target document. The final score is computed as follows:

$$A = \#\ of\ associated\ topics\ of\ a\ target\ deck$$
$$B = \#\ of\ associated\ topics\ shared$$

$$\text{final score} = \frac{\text{intermediate score}}{\mathrm{B}^{\mathrm{B}}}(\mathrm{A} - \mathrm{B} + 1)$$

The intermediate confidence interval is thus divided by the number of topics that a potentially recommendation j shares with document i to the power of this same number. Should this number be equal to 1, then the intermediate score will be divided by 1. The final score is then finally calculated by multiplying the aforementioned calculation by the number of topics document i is associated with, minus number of topics that a potentially recommendation j shares with document i, plus 1. The nice thing about this calculation is that it does not harm target documents that are associated with a low number of topics (such as 1). The final calculation will then simply be the intermediate score divided by 1, times 1. This is equal to the intermediate score itself.

After the final confidence intervals have been determined, the list is once again sorted from lowest to highest. Due to a large abundance of duplicates in SlideWiki, the first 5 results are discarded as an attempt to avoid them. Forking is among other a key feature in SlideWiki that allows users to create duplicates for themselves. Thus there is a large presence of duplicates that are either (almost) identical to the target document or (almost) identical to one of the recommendations that were already determined. Simply deleting duplicates from the dataset or omitting them from the recommendation system is not a suitable solution in the use case of SlideWiki where forking is a key feature. Given that document retrieval using LDA is not Boolean retrieval (due to determining matches through scores between 0 and 1, instead of a Boolean value), it means that a target document can always have recommendations. The extent of how related a recommended document then is to a target document i is dependent on the score/confidence interval. It is thus assumed that discarding the first 5 lowest results does not hurt relevance feedback, since there should be plenty of documents that can be related to the target document i anyway.

After discarding the 5 lowest results, the system loops through the final list with the goal to find 10 document recommendations for document i. If the confidence interval of a document j is not equal to the confidence interval of the previously recommended document, save document j as a recommendation for document i. In other words, save document j as a recommended document k, only if document j is not a duplicate of recommended document k-1 (confidence intervals are different/not equal). The

recommendation system will keep searching for documents to recommend until it has found 10 recommendations for document i or until it has exhausted its options for potential recommendations for document i.

Going back to deck 101888, these are the recommendations returned for this particular deck: *{'106477': 0.0013723926992843102, '99262': 0.0019579153229656266, '108790': 0.0019626365690208654, '97809': 0.0020098828243271488, '358': 0.0038274020059378393, '3201': 0.0048214531486790845, '2754': 0.0048214531489461981, '99587': 0.0049867929403360524}*. According to the recommendation system; these should all be decks that are similar in content to 101888. These are all deck IDs that correspond to actual decks on SlideWiki. The link to for example deck 3201 can be found here: *https://slidewiki.org/deck/3201*.

4 Spam filtering

There were some naïve assumptions going into this project. One of them being that there was no consideration of the possible presence of spam on SlideWiki. However, after scraping the website of its decks it quickly became clear that there was a large proportion of spam which had to be addressed in order to get the most out of the recommendation system. Simply put, spam can be seen as noise which has to be filtered out of the dataset during preprocessing. Not only will spam filtering likely affect the efficacy of the recommendation system, but the score builder will likely perform faster as well. Latent Dirichlet Allocation is a rather slow process. It took the score builder well beyond 15 hours to complete its process with the 17K unfiltered set of decks. Cutting down on the size of the dataset by filtering out spam will likely improve run time as the it seems to be heavily influenced by the size of the dataset.

The final approach to spam classification is an ensemble that consists of Naïve Bayes classification and Latent Dirichlet Allocation. The individual models of which the ensemble is made up will be explained in the following sections, as well as the ensemble itself. The implementations of the models along with the recommendation system can be found on the GitHub page.[14]

4.1 Naïve Bayes spam filter

The initial idea was to just tackle the spam head-on and build a Naïve Bayes spam filter. Naïve Bayes classification was the first solution that came to mind when faced with the large presence of spam in the SlideWiki dataset, since the textbook example of Naïve Bayes being applied is it being trained and implemented as spam filter [6]. The implementation of the Naïve Bayes spam filter is largely based on an example that can be

[14] *https://github.com/bosnyan/Bachelor-Project-SlideWiki*

found on Cambridge Spark.[15] The article by Ekaterina Kochmar shows how a spam filter based on Naïve Bayes classification can be built using Python's NLTK package.[16]

Aside from its associations as being a spam filter, Naïve Bayes has some other characteristics that make it interesting in this particular problem. A Naïve Bayes spam classifier is relatively easy to train. Give it a set of labelled examples of both classes "ham" and "spam." Train the model on this labelled training set such that it learns what documents should be considered spam and which documents should not be considered as spam based on the examples provided in the training set. In this case, the resulting labelled set consisted of 1151 "ham" labelled examples and 3808 "spam" labelled examples. 85% of this set is used for training, while the remaining 15% was set aside as a test set to evaluate the Naïve Bayes spam filter. This resulted in the following sets:

- 978 ham training
- 173 ham test
- 3236 spam training
- 571 spam test

Another upside to using Naïve Bayes classification is the relatively high speed when performing classification. The spam filter is able to process the 17K set of English decks in 11 minutes and 5 seconds.

A downside with Naïve Bayes however is the performance of the spam filter itself. The classifier has an accuracy of 85% on the test set. This is by no means bad, but it is far from perfect. Furthermore, the evaluation on the test set only returned the accuracy. Its precision and recall on the test set is unknown. A high recall is of course desirable, since it would mean that the classifier is sensitive enough to filter out a large proportion of spam, however precision is arguably much more important. When evaluation a spam classifier's efficacy in terms of costs of the resulting errors, it becomes clear why precision is much more important than recall. In the case of a false negative, the worst what can happen is that a spam document is given as a recommendation by the recommendation system. This would be mildly infuriating at most, while a false positive results in losing a non-spam document, as it was incorrectly classified as spam. With a non-spam SlideWiki deck it would mean that for example someone's lecture on Artificial Intelligence got filtered away. That is valuable information that is lost due to an incorrect classification.

Furthermore, Naïve-Bayes uses supervised learning. The spam filter is constructed by providing labelled examples that are spam and not spam. Unless the sample sets are constantly updated as new decks are uploaded, the spam filter risks to become worse as the website grows in more and more distinct content. Especially when considering that SlideWiki aspires to be a platform to create and share decks about any subject, worse performance of this spam filter over time seems inevitable.

---

[15] https://cambridgespark.com/content/tutorials/implementing-your-own-spam-filter/index.html

[16] https://www.nltk.org/_modules/nltk/classify/naivebayes.html

4.2 Latent Dirichlet Allocation spam filter

An alternative to Naïve-Bayes spam filtering could be spam filtering using the same LDA classifier that was used for the score builder. However, instead of identifying with which topics a document i is associated with, the classifier will determine whether a document i is spam by evaluating its associations with a set of preidentified "spam topics." These spam topics are topics that contain labels that are typical vocabulary present in spam documents. An example of a spam topic is the following: *Topic 2: gmail password recovery forgot warehouse passwordcall way help instant reliable passworddial effective entire operational helpusa wayusa helpuse helpget solutionusa analytical.* Evaluating the labels present in topic 2, it becomes rather clear that this topic is likely associated with spam documents.

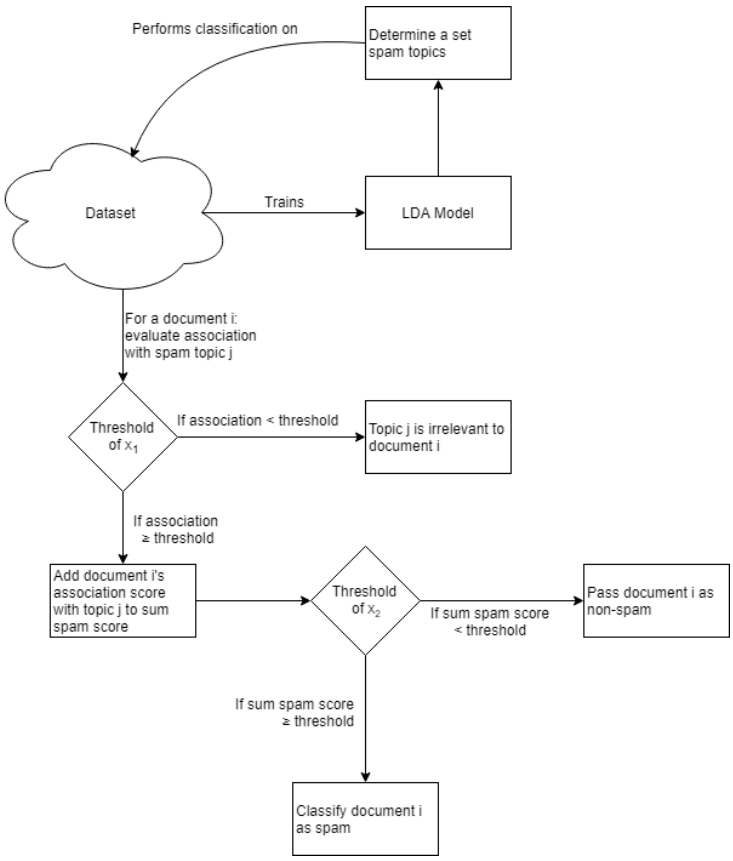

Fig. 2. Visualization of the process behind how the LDA spam filter works

There are several reasons to use LDA spam classification, the first one being the ease of implementation in the case of this project. The recommendation system which is based on LDA was already developed for this project. Adapting this implementation to work as a spam filter requires minor tweaks. As seen in section 3.1, the LDA classifier

returns a document's associations with topics expressed as a decimal value. Given a set of spam topics, the spam filter will look into what a document i's associations are with the set of spam topics and take the sum of these values. If this sum association exceeds a certain threshold (in this case it has been set to 0.7), the document will be classified as spam. If this sum value is below the threshold, the document will not be classified as spam and it will remain in the final dataset.

The set of spam topics are determined beforehand and their indices are contained in a list (e.g. topic 0 sits at index 0 etc.) through which the spam filter will loop to evaluate a document's association with a spam topic i in the set of spam topics. The final implementation of LDA spam classification contained 19 spam topics after training on the 17K dataset.

Manually determining which topics should be seen as spam topics has its merits, but there are also some issues. One of the issues being that the LDA spam filter will require constant human intervention to again determine what the spam topics are every time the LDA model has updated. The upside to this approach and LDA in general is that there is much more possibility for finetuning the model to either precision or recall. If recall needs to be maximized, just set the threshold lower and be rather liberate with the selection of spam topics. Precision will of course suffer under this, but that can again be prioritized by raising the threshold and being more conservative with the selection of spam topics. Further optimization can be done by finding the optimal number of topics, iterations, learning offset, batch size etc. relative to the target dataset.

The biggest downside with using LDA for spam classification however is its relatively slow performance. It took the LDA classifier roughly more than 16 hours to process the 17K unfiltered set of English decks. This is considerably slower to Naïve Bayes' performance, which was able to process the same dataset in just beyond 11 minutes.

4.3 Ensemble spam filter

When considering the strengths and weaknesses of both models, an ensemble consisting of LDA on top of Naïve Bayes classification seems to be a solution where both models will play to each other's strengths. LDA is a slow classification technique, especially when compared to Naïve Bayes classification which is considerably faster. Naïve Bayes does not have many options regarding tuning, while LDA is much more flexible and it can be prioritized towards either precision or recall. Furthermore, a Naïve Bayes model runs the risk of becoming outdated in an ever-updating and growing database. A LDA model can just train on the new database again, such that it is updated on the newest contents of the database.

Regarding the use cases of the respective spam classifiers, Naïve Bayes classification can be best seen as a classifier that is ran once to perform a rough, but quick cleanup of a large dataset. LDA on the other hand is best suitable for handling smaller datasets due to its slower performance, but high flexibility regarding tuning. Such small datasets are for example intermediate datasets that are returned by the Naïve Bayes classifier where some spam still may be present, or even batches of new uploads that have to be evaluated on being spam or not. In the long run, the LDA model seems as a more

sustainable solution however, since it has the possibility to be constantly updated and tweaked such that it remains able to effectively classify spam. Furthermore, the LDA classifier is perhaps rather slow, but its implementation will likely just handle small batches of newly uploaded decks at a time. This makes the slow performance rather negligible.

All in all, the Naïve Bayes classifier does the rough filtering and returns an intermediate dataset where most of the spam has been filtered out. The LDA classifier trains on the intermediate dataset and tries to pick out the remaining spam.[17]

5 Dataset and Methodology

In this project two experiments were ran where the efficacy of the spam filter was evaluated, as well as the efficacy of the recommendation system. The same datasets and the same parameters for the models were used over the two experiments. The datasets and parameters used will thus be discussed separately in order to prevent repetition of the same information when discussing the individual experiments.

5.1 Dataset description

The recommendation system or any classifier in fact needs a dataset to train on. In this case the dataset consists of SlideWiki's decks in plain text. The entire collection of SlideWiki's decks was scraped on the 28th of May 2018 and 27012 decks were retrieved. The getDeckId function on SlideWiki's own deckservice[18] was repurposed in Python to scrape all the decks that were available on SlideWiki at that time. Since SlideWiki uses simple integer value for their deck IDs, a for loop of 0 to 999999 was used to check for every value whether it was an existing deck ID. A response code of 200 meant that the deck with the given deck ID i exists and it could thus be scraped. As the deck is formatted in HTML, the first step was to only get the text from the response. LXML[19] proved to be very helpful with that. The last step was retrieve the plain text content of the deck by searching for text that fell within a defined regex pattern.

While plain text is a good start regarding data to process, the dataset could still use some pre-processing. The initial dataset consists of decks written in various languages (i.e. Hindi, Russian, Greek et cetera) and there is also a rather large proportion of very short decks. These shorter decks were first of all mostly decks that were made for testing purposes and furthermore, other non-test decks that are of such small size will not be informative anyway. The first course of action on the initial dataset was

[17] It should be noted however that the LDA model of the ensemble in this project is just a duplicate to the lone LDA classifier. This means that the LDA classifier of the ensemble is trained on the initial 17K unfiltered dataset, instead of on an intermediate dataset that was returned by the Naïve Bayes classifier.

[18] https://deckservice.slidewiki.org/documentation#!/deck/getDeckId

[19] https://lxml.de/index.html#documentation

filtering out any decks where its plain text content was smaller than 1kB. This resulted in a dataset of 20460 decks

After removing the small decks, the contents still require further processing. Non-Latin symbols, punctuation marks and stopwords were removed. Furthermore, all content was turned into lowercase text and lemmatization from NLTK's lemmatization library was applied to the decks as the final step. Lemmatization is particularly useful for this use case, as it tries to unify all inflicted forms of a term into a single word. This prevents information loss and it also prevents the likelihood of the inflicted forms of a term (car, cars, …) to be split over multiple classes in the model space of our classifiers.

Trying to keep the scope of this project somewhat narrow, non-English decks were removed as well. Non-English decks were filtered out using a Python library, called langdetect.[20] This library is a port of a Google library and it has the function of estimating the possible language(s) of a given input string. Filtering non-English decks with the help of langdetect resulted in a total of 16723 likely English slides. One should however keep in mind of that of course that the filtering process was automated process where the program merely estimates the likelihood that a deck i is either English or not. Given that this is not a perfect world and despite all optimizations made, the "English-only" classifier is not perfect either. This does mean that errors did occur where non-English decks were overlooked by the classifier and actual English decks were unjustly filtered out as being considered non-English by the classifier. Regarding the classifier's efficacy, precision and recall was measured. 200 random samples were retrieved, 100 from the resulting English set of decks and 100 from the resulting non-English set of decks. The classifier has a recall of 0.81 and precision of 0.92. The fact that the dataset is imperfect (i.e. not completely "English-only") should be kept in mind when commenting on the efficacy of both the spam filters and the recommendation systems.

After pre-processing the data is in principle ready to be processed through the LDA topic modelling classifier. However from the set of 16723 decks, there is still a large prevalence of spam present in the dataset. Three different models have been considered for this project and they will later on be compared in terms of performance in one of the experiments. Regarding the final sets, there are four of them.

- Naïve Bayes spam filtering resulted in 4476 decks,
- LDA spam filtering resulted in 3948 decks
- Spam filtering by the ensemble resulted in 3781 decks.

5.2 Parameter description

As mentioned before, the same parameters were used on the models across all two experiments. Any setting that has not been mentioned in the paper, but actually is an adjustable setting for any of the models was kept at its default values.

[20] https://pypi.org/project/langdetect/

Furthermore, while the ensemble could possibly be more effective when training its LDA model on the non-spam dataset returned by Naïve Bayes classifier and optimizing the parameters of its LDA model on that dataset, for the sake of further enforcing consistency in experimentation the LDA model of the ensemble spam filter is basically a copy of the lone LDA spam filter model. This means that both LDA spam filters are trained on the initial 17K dataset and have the same parameters set that make the classifiers work optimally on the initial 17K dataset regarding spam classification. These are the following settings of the spam filter models:

- **LDA spam filter**
  - NO_TOPICS = 50
  - REQ = 0.05
  - SPAM_REQ = 0.7
  - SPAM_TOPICS = [0, 2, 4, 5, 8, 9, 14, 18, 23, 29, 31, 33, 34, 39, 41, 43, 46, 47, 48]
- **Naïve Bayes spam filter**
- *Labelled set:*
  - 1151 ham
  - 3807 spam
- *train_set, test_set, classifier = train(all_features, 0.85)*
  - 978 ham training
  - 173 ham test
  - 3236 spam training
  - 571 spam test

The LDA recommendation system has the same parameters across all four different spam filter approaches (baseline, Naïve Bayes, LDA and ensemble). These are the parameters of the recommendation system:

- **LDA model**
  - NO_TOPICS = 50
  - MAX_ITER = 2500
  - learning_offset = 10
  - batch_size = 1024
- **Score retrieval**
  - NO_TOPICS = 50
  - REQ = 0.05

5.3 Experiment 1: Spam filter evaluation

The first experiment will try to answer the second research question on what model will be most effective in dealing with spam from SlideWiki's collection of decks. The efficacy of the three spam filters will be compared in terms of precision and recall when filtering spam decks from non-spam decks. Those three spam filters are: Naïve Bayes spam classification, Latent Dirichlet Allocation spam filtering and the ensemble spam filter of LDA on top of Naïve Bayes classification.

For all three spam filters, random samples of 1500 decks will be taken from both the resulting spam and non-spam sets. These samples will be checked manually on being spam or not. This ultimately means that there will be a total random sample set of 9000 decks.

- 1500 random samples from Naïve-Bayes' spam set,
- 1500 random samples from the Naïve-Bayes' non-spam set,
- 1500 random samples from LDA's spam set,
- 1500 random samples from LDA's non-spam set,
- 1500 random samples from the ensemble's spam set
- 1500 random samples from the ensemble's non-spam set

When measuring precision and recall the question is often on which of the two is more important. When looking at precision and recall in terms of cost, it could be argued that precision is more important when performing spam filtering. It is tolerable if there is still some spam present in the final set after spam filtering, since the negative consequences will relatively minimal. Especially when comparing the cost of incorrectly classifying a non-spam deck as spam. It would be considerably more egregious if a non-spam document were actually to be classified as spam. In the domain of SlideWiki, incorrectly classifying a deck of slides as spam could for example mean incorrectly deleting someone's lecture on Artificial Intelligence, since it was classified as spam.

5.4 Experiment 2: Recommendation evaluation

The second experiment will mainly answer the research question on whether spam filtering will aid the efficacy of the recommendation system while also give some insight on the first research question; "How can a recommendation system be built that provides relevant suggestions relative to the content consumed at a particular moment in the domain of SlideWiki?"

Four models were compared to each other.

- Recommendation system based on Latent Dirichlet Allocation with no spam filtering performed on the set of decks.
- Recommendation system based on Latent Dirichlet Allocation with spam filtering performed on the set of decks by Naïve Bayes classification.
- Recommendation system based on Latent Dirichlet Allocation with spam filtering performed on the set of decks by Latent Dirichlet Allocation.
- Recommendation system based on Latent Dirichlet Allocation with spam filtering performed on the set of decks by the ensemble model (Latent Dirichlet Allocation on top of Naïve Bayes classification).

A sample of 100 decks will selected to measure efficacy. The same sample set will be used across all three models to enforce consistency in the measurements.

Efficacy will be measured by the number of relevant recommendations a model will provide over all sample decks. The recommendation system always returns 10 recommendations for a given deck, thus in order for a model to have a perfect score of

100%, it should provide 1000 good recommendations over the 100 sample decks. Determining what is "relevant" remains a subjective matter to some extent of course, but some guidelines have been set to determine whether a recommendation is relevant or not. Information need is assumed based on the content of the sample deck. Based on this statement, the following rules are be set:

- Duplicate is considered to be redundant ⟶ irrelevant recommendation
- Spam is considered to be noise ⟶ irrelevant recommendation
- Recommendation about the same topic as the sample, but different content is considered to be relevant (e.g. sample is about RDF and recommendation as well) ⟶ relevant recommendation
- Recommendation about a similar topic as the sample is considered to be relevant (e.g. sample is about clustering and recommendation is about LDA topic modelling) ⟶ relevant recommendation
- Recommendation with an unrelated topic to the sample is considered to be irrelevant (e.g. sample is about marketing channels and recommendation is about SQL databases) ⟶ irrelevant recommendation

Recommendations will individually be given a binary score of 1 if a recommendation is relevant or 0 if a recommendation is irrelevant. For every deck, the recommendation system can thus get a maximum of 10 points, provided that all recommendations returned are good. Every recommendation that is considered not to be good, the model will not receive any points. For example, in the case where the control model returns recommendations for deck 33 and 6 of those recommendations are considered to be good, the control model will have 6 points for deck 33. These scores will in the end then be summed and divided by the maximum number of points (1000) to determine the model's final score.

Regarding spam filtering, the sample decks are ran through the spam filters. This is to sketch a more realistic picture of the real-life situation where a genuinely informative (or at least non-spam) deck can potentially not be included in the recommendation system's database if it were to be unjustly classified as spam by the spam filter. If a sample is classified as spam, it will not be included into the recommendation system and recommendations to that given deck can thus not be provided. In that case it will not get any points.

6 Results and findings

6.1 Results and findings: experiment 1

After evaluating 1500 random ham samples and 1500 random spam samples for each of the three spam classification approaches, the results show that the ensemble of Latent Dirichlet Allocation on top of Naïve Bayes classification is most effective in filtering spam. The ensemble has overall the highest f-score of 0,957. It should be noted that the Naïve Bayes classifier has a precision of 1. This means that no false positives were observed in the samples of the Naïve Bayes classifier. In other words, there is likely no

loss of valuable non-spam decks when performing Naïve Bayes classification on the 17K dataset of SlideWiki's decks.

Even when assuming a confidence interval of 99%, the precision of the Naïve Bayes classifier lies between 0,99 and 1. Thus in the worst case scenario, the Naïve Bayes classifier may have falsely classified 123 non-spam decks at most. These results bolster the claim of Naïve Bayes classification to be useful as an initial filter to quickly clean up SlideWiki's database of most its spam without causing much damage in terms of falsely classifying non-spam decks as spam.

Another interesting finding is the ensemble's higher recall to the other two models. Keep in mind that this particular implementation of the ensemble is basically the LDA model which is a duplicate of the lone LDA classifier that runs right after the Naïve Bayes classifier. Both models are trained on the same 17K unfiltered dataset and thus the LDA classifier of the ensemble was not even optimized to run classification on the subset of filtered decks that was returned by the Naïve Bayes classifier. This first of all means that better efficacy regarding spam filtering would be possible if the LDA classifier were to be optimized on the intermediate subset, but it also means that the ensemble benefits of a sort-of unified recall between the LDA and Naïve Bayes classifier. Both classifier seem to be looking at different spam-related features and combining the two classifiers leads to a higher recall than if either models were to run separately.

Table 1. Efficacy of the three respective spam filters

	NB classifier	**LDA classifier**	**Ensemble**
Precision	1	0,998	0,998
Recall	0,812	0,881	0,92
F-score	0,896	0,936	0,957

6.2 Results and findings: experiment 2

Evaluating the relevance of the 100 preselected samples over the four different recommendation system has shown that the recommendation system that had spam classification performed by the ensemble on its dataset is the most effective in providing relevant recommendations. Especially when comparing the ensemble's efficacy to the baseline, it shows the impact that spam classification can have on the efficacy of a recommendation system.

It should be noted however that when comparing the ensemble's efficacy to the other two recommendation systems that utilized Naïve Bayes classification and LDA respectively that the differences between the three recommendation systems become less significant. Then again, this is rather understandable given the smaller differences of spam proportion between the subsets of the classifiers compared to the dataset of the baseline and the subset of the ensemble for example. The subset of the ensemble contains almost 14K fewer spam decks compared to the dataset of baseline. Compared to the subset of the Naïve Bayes classifier, the subset of the ensemble merely contains around 690 fewer spam decks.

Despite the ensemble's ability to filter out most of the spam in the 17K dataset, it still had an accuracy of 60,2%. While definitely higher than the baseline, it does show that there is more to improvements in relevancy than merely spam filtering. Given how approach to document recommendation is based on content, the extent of relevant the recommendations will be is also dependent of how well represented the subjects of the document are in the dataset. Subjects such as Information Retrieval are very well-represented on SlideWiki and sample decks that were about Information Retrieval usually received 8 or more relevant recommendations. Sample decks with subjects that were less well-represented on SlideWiki of course suffered from worse recommendations.

Most irrelevant recommendations in the ensemble were either duplicate recommendations within a set of recommendations for a sample or recommendations that were unrelated in subject to the sample. Significant improvements in relevancy of recommendations will thus not be found through further improvements in spam classification.

Table 2. Accuracy of the four different approaches to the recommendation system

	Baseline	**NB**	**LDA**	**Ensemble**
N	16723	4476	3948	3781
Accuracy	0,363	0,508	0,527	0,602
%	36,3%	50,8%	52,7%	60,2%

7 Conclusion

To first give an answer to the second research question, the ensemble of Latent Dirichlet Allocation on top of Naïve Bayes classification has shown to be the most effective in spam classification of this project's dataset. Despite the fact that the LDA classifier of the ensemble was not optimized to the intermediate subset returned by the Naïve Bayes classifier, its precision based on the samples was near-perfect with a score of 0,998 alongside with a recall of 0,92. Actually tuning the LDA classifier to this filtered subset will likely lead to even better efficacy regarding spam classification.

Moving on to the third research question, it can also be seen that spam filtering definitely does help with improving the effectiveness a LDA recommendation system in terms of providing relevant recommendations. Between the ensemble and the baseline, there is an improvement of 23,9% in terms of returning relevant recommendations. There is still plenty room of improvement however and merely focusing on further optimizing spam classification will likely not result in considerably better results. As discussed in 6.2, the errors in the ensemble's recommendations were usually either due to a deck's irrelevance in topic to the sample or due to the fact that the recommended deck is a duplicate of either the sample deck or an earlier recommendation. Addressing these issues will among other things likely come down to optimizing the LDA classifier of the recommendation system.

All in all however, a recommendation system using Latent Dirichlet Allocation appears at very least to be good proof-of-concept in use cases such as SlideWiki. First of all, Latent Dirichlet Allocation provides quantitative descriptions to SlideWiki's decks with its topics where there previously was none. In other words, LDA creates values for SlideWiki's decks such that they can be put in relation with each other. This can be a good basis for further implementations, such as collaborative filtering or context-aware recommendation systems.

Furthermore, a sampled accuracy of 60,2% after noise reduction through spam classifications shows some promising signs in terms of effectiveness. Addressing the issue of duplicate recommendations and further optimizing the LDA classifier of the recommendation system will likely lead to better recommendations.

8 Discussion

There were several things that could have been done better when reflecting on how this project went. First of all, the evaluation of the recommendation system was rather flawed. Evaluating a total of 4000 recommendations on relevancy on your own is not only an enormous evaluation to conduct, but inconsistencies in the evaluations are bound to happen and most likely have happened. Dividing these recommendations over several evaluators and determining a Cohen's Kappa versus my own evaluations would likely result in a more reliable result. Cohen's Kappa namely shows the extent of consensus on results between evaluators.

Furthermore, the scoring on the second experiment could perhaps use more granularity as well. While the limited timeframe and the scale of the research project limited the evaluation method of a recommendation to either a 0 or 1 score to keep things rather simple, evaluating a recommendation on relevancy with a score between 0 and 9 would perhaps be a more complete evaluation on relevancy.

The paper already touched a few times on the fact how further optimization of the LDA model and the classifier's parameters would likely lead to better results in terms of returning relevant recommendations. The LDA model and the classifier were optimized for the baseline recommendation system and the lone LDA spam filter, since the optimal parameters were determined based on the 17K dataset. While good for experimentation purposes, the parameters set for the recommendations systems that were based of the non-spam subsets returned by the different spam classifiers were not ideal. Meaning, to get the most out of the subset that was returned by for example the ensemble, the LDA model and the parameters of the classifier would have to be tuned to the subset returned by the ensemble.

8.1 Future work

Even after taking some measures to minimize the duplicates in the recommendation system by preventing equal confidence intervals between recommendations, the prevalence of duplicate errors in the recommendations still persisted largely and it

affected the relevancy of the recommendations provided by the recommendation system. Despite the fact that duplicate decks were syntactically similar or even equal to each other, they still had different confidence intervals (i.e. different topic associations) that made them slip through the measures put in place in the recommendation system. As it has been earlier discussed in the paper, simply deleting the duplicate decks or excluding them from the recommendation system is not an option. This would go against one of the key features of SlideWiki, namely forking. Forking is essentially creating a duplicate for personal use and thus deleting these forked duplicates or omitting them from the recommendation system would be counterintuitive to the feature.

One way the problem could be addressed is by evaluating whether a potential recommendation is a duplicate based on the syntactical match it has against the target deck and the recommendations already provided. This could be performed beforehand similar to how the score builder utilizes Latent Dirichlet Allocation. There could be a master document that is the oldest/most informative and a set of documents that are likely derived from this master document. If a deck is selected as a potential recommendation and it appears to be derived from a different master document, then the master document is provided as the recommendation instead. Furthermore, if a master document is already given as a recommendation, then all the derived documents will be prevented from being suggested as a recommendation to the target deck.

Given how this recommendation system using Latent Dirichlet Allocation can be seen as a nice basis, further improvements on the recommendation system outside of LDA is interesting for future work. For example, a content-based system could take user feedback into account to create an idea of a user's interests/information needs outside of the deck that is consumed at that moment [8].

9 Acknowledgements

I would like to thank Tobias Kuhn for having this interview with me where he suggested Latent Dirichlet Allocation as a potential way to provide recommendations from the content. It was this interview that got me interested in LDA and which eventually got the ball rolling to making this recommendation system and this thesis around it.

References

[1] G. Shani and A. Gunawardana, "Evaluating Recommendation Systems," *Recommender Systems Handbook,* pp. 257-297, 2011.

[2] D. M. Blei, A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research 3,* vol. 3, pp. 993-1022, 2003.

[3] A. B. Raut and G. R. Bamnote, "Soft clustering: An overview," *IJCCT*, vol. 1, no. 4, pp. 370-372, 2010.

[4] D. Inkpen and A. H. Razavi, "Topic Classification using Latent Dirichlet Allocation at Multiple Levels," *IJCLA*, vol. 5, no. 1, pp. 43-55, 2014.

[5] T. Luostarinen and O. Kohonen, "Using Topic Models in Content-Based News Recommender Systems," in *Linköping Electronic Conference Proceedings #85*, Helsinki, 2013.

[6] P. Flach, "CHAPTER 9 Probabilistic models," in *MACHINE LEARNING The Art and Science of Algorithms that Make Sense of Data*, Cambridge, Cambrdige University Press, 2012, pp. 274-282.

[7] J. Višnovský, O. Kaššák, M. Kopman and M. Bieliková, "The Cold-start Problem: Minimal Users' Activity Estimation," *RecSys*, vol. 14, pp. 6-10, 2014.

[8] M. J. Pazzani and D. Billsus, "Content-Based Recommendation Systems," *The Adaptive Web*, pp. 325-341, 2007.