Technische Hochschule Ingolstadt

Technische Hochschule Ingolstadt

Faculty of Electrical Engineering and Computer Science

Computer Science

Bachelor's thesis

Subject: Extracting Reliable Topics using Ensemble Latent Dirichlet Allocation

Name and Surname: Tobias Brigl

Issued on: 07/03/2018

Submitted on: 03/08/2018

First examiner: Hahndel, Prof. Dr. rer. nat. Stefan

Second examiner: Regensburger, Prof. Dr. rer. nat. Franz

Abstract

LDA is a widely known model that finds topics in text. But, similar to k-means, you have to decide how many topics to look for. The stochastic nature of machine learning algorithms also causes the model to have instabilities, which results in topics being a mixture of multiple topics. This is the issue that led to the development of the “Ensemble LDA” model, which detects the stable topics in the text by using an ensemble of LDA topic models. Ensembles have already been successfully used in the past for decision trees as random forest. This thesis will answer the question to whether or not the proposed way of making an ensemble of LDA models works robustly. Furthermore, various validation methods were explored and improvements on the existing code were discussed.

Table of Contents

1. Introduction to LDA

This chapter is mainly based on a video by Andrius Knispelis about the idea and use case behind LDA Topic Models [7].

1.1 Purpose

The LDA topic model is a model[i] that finds topics in text. By training a LDA model over newspaper articles, it will tell you when the article is from the “Sports” topic by looking at the frequency of certain characteristic words in that article. Just like a probability distribution, the frequency refers to which words are more often in the text than others. That frequency is what is being observed by the model during the trianing.

The LDA model learns whether or not a word (for example “running”) is characteristic for a certain Topic (for example “Sports”). When that word occurs very often in a text, it is likely to be of the “Sports” topic. There are various other words that are also characteristic for the “Sports” topic, which also helps to identify the correct topic.

There is a probability of the article belonging to a different topic though. The “Sports” topic would be the one that is the most likely, but the article might contain the word “soup” once, which means that the article could actually belong to the “Food” topic, but only with a small probability. Articles that are about the right food when doing sports are also possible, in that case the text has multiple topics, which would affect the probability distribution to show two topics with a high probability. All probabilities sum up to 1.

“A document is a probability distribution over topics” and “A topic is a probability distribution over words” [7]

However, it doesn’t know that the “Sports” topic is actually about Sport. It will just return the topic number and the user has to decide how to call that topic. In that way it similar to k-means, where the classification is the centroid index and the meaning of that centroid has to be interpreted by a human. Furthermore, similar to k-means, it has to be decided how many topics to look for. This number of topics is often called k.

1.2 Dirichlet Distribution

The Dirichlet distribution is a “distribution over distributions” [3]. To be more specific, it is a distribution over multinomial distributions. A more intuitive explanation of this would be the probability of a dice having a specific shape, which decides about the outcome probability of dice rolls. Sampling from a Dirichlet distribution yields a probability mass function (PMF).

---

i A mathematical way to simulate or describe something from the real world, often based on statistics.

Consider a dice that is not fair but rather has a lot of imperfections that are randomized for each dice. Various dices will have various probabilities for each side to be on the top. One of the dices might have a relatively large probability to have the number “6” on the top after rolling it, whereas another one is very likely to never show “3” (also called “Samples”). They all have various probabilities for each side (which always sum up to 1). This randomness of selecting differently weighted dies is represented by drawing multinomial distributions from a Dirichlet. [1]

In the case of LDA, each text is represented by the frequency of individual words, which is also known as bag of words. Furthermore, the text is represented by a PMF of topic memberships. This PMF is represented by a Dirichlet Distribution. The randomness of the probability of words in the bag of words is represented by as many additional Dirichelts as the number of topics of the model.

1.2.1 Simplex

A Simplex is the simplest geometric shape that a k-dimensional space can contain. This is a triangle (k = 3, which means 3 vertices/corners, also known as topics) for a 2D *(k - 1)* space.

$$\Delta_k = \{ q \in \mathbb{R}^k | \sum_{i=1}^{k} q_i = 1, q_i \geq 0 \, for \, i = 1,2,..,k \} \quad [1]$$

Δ_k is the set of all possible points on that simplex. q is a point on that simplex, which means it is a vector of k elements: $(q_i, q_{i+1}, \ldots, q_k)$. Consider $k = 2$, now each point will consist of two components. One of them could be *(1, 0)* and another one could be *(0, 1)*, all other points lie in between those two points because the components are always between 0 and 1, which means the simplex for k = 2 is a line. *(0,2), (1,2)* and such are invalid, because sum is larger than 1. Negative numbers are also invalid.

For example if each vertex was multiplied (which is a vector in the $\mathbb{R}^n$ space) by $^1/_n$ and they were added together, the coordinates of the center of the simplex would be yielded, because the simplex center is the arithmetic mean of all the vertex.

In the case of LDAs, each vertex/corner of the simplex corresponds to a single topic. [7]

The coordinates for a Simplex in $\mathbb{R}^3$ are: (0, 0, 1), (0, 1, 0) and (1, 0, 0), as seen in Figure 1. Those coordinates are affine independent, because they cannot be multiplied with a factor != 0 and summed to a vector (0, 0, 0).

A fair dice has an uniform probability for each side to be on the top, which means that the probability for each side to be on the top after rolling it is $^1/_6$. When visualizing this in a simplex, it would be therefore at the center of the probability simplex. This simplex has one vertex for each possible outcome of the dice roll, which is also the length of the PMF vector. It is 6 in the case of the dice. When the probability of one side to show up would be higher, the point in the probability simplex would move closer to one of the vertex.

1.2.2 Parameters alpha and eta

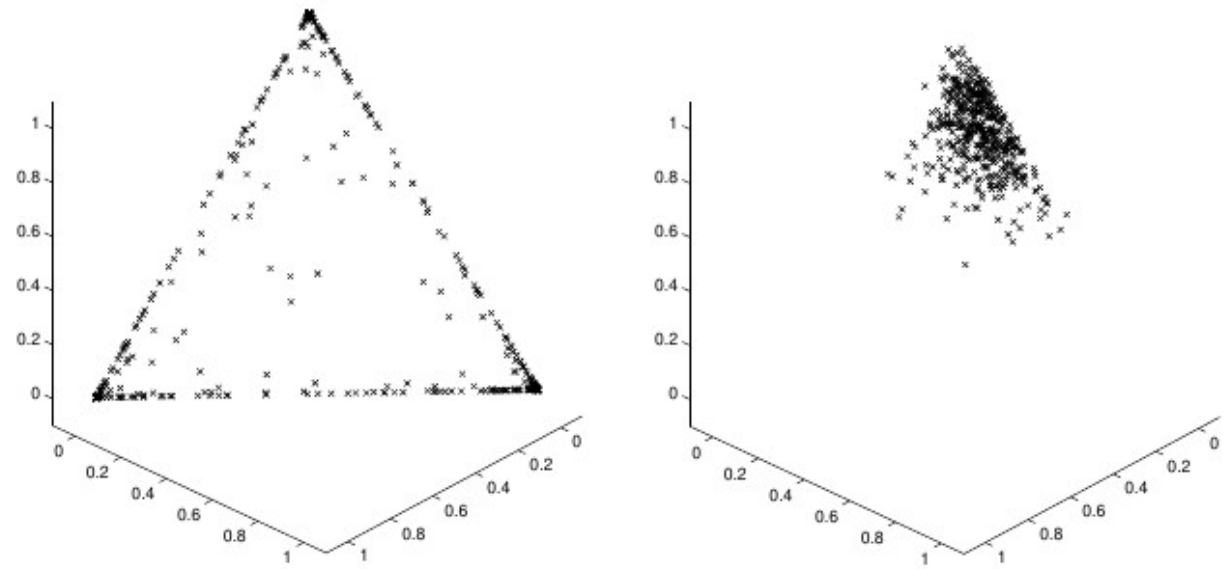

Figure 1: The distribution of probabilities depends on its parameter α. On the picture are two simplices with 3 vertex, that means k = 3. Random samples drawn from the Dirichlet distribution contained in the simplex are the black points. [1]

The probability of a point q given parameters α in the Dirichlet distribution is defined as [1]:

$$\alpha_0 = \sum_{i=1}^{k} \alpha_i$$

$$f(q;\alpha) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} q_i^{\alpha_i - 1},$$

With α and q being vectors of size k. The parameters alpha and eta of the model are both used for α in the above Formula.

There is one Dirichlet in the LDA model from which the probability of topics for a randomly generated document can be sampled. There are also k Dirichlet distributions, one for each topic, that give insight about the probability of words for each topic.

The parameter alpha describes the prior belief about which topic distribution is likely for a document. Eta is the prior belief about the topic word distributions of each topic, i.e. how each topic is believed to look like. Both are hyperparameters of the model, but they can be left blank in which case they will be initialized to default values.

This is done to change the way the model is initialized. If there exists prior knowledge of the nature of the documents, proper alpha and eta parameters can be used to randomly initialize the topics in a way that helps to make the model converge more reliable and faster.

High alpha values mean that the text samples are strongly influenced by multiple, various topics, whereas low alpha values mean that there are only few topics that are represented in each document, as can be seen in Figure 1.

When the Dirichlet parameter values sum lower than 1, the probabilities are higher at the corners and lower in the middle. When they sum to 1, the samples are evenly distributed on the simplex surface. Not only does the alpha value specify the variance of the distribution, it also specifies the center of it.

The parameter **eta** describes how individual the word distribution is for each topic. A low beta value means that each topic contains very individual words, a high value means that the various topics contain words that are similar [7]. But eta can also be a matrix with k rows, so that each topic gets a different Dirichlet parameter for the priors.

As the parameters of the model can be used to generate new bag of word documents, "High alpha will make **documents** appear more similar to each other and high beta will make **topics** appear more similar to each other." [7]

1.2.3 Multinomial Distribution

The multinomial distribution is the probability for a given combination of outcomes after taking a few samples. The order of outcomes is not important, it only counts how often e.g. outcome A and B occurred.

Each document can be generated by a multinomial distribution over topics and another multinomial distribution over the words in each topic. So one topic that gets a lot of weight will influence the generated document more than others. But except the observed word frequency in each document, no variable can be observed. So the generative process is reversed and all the hidden latent variables like theta and beta that are going to be explained in more detail in the following chapter are inferred.

1.3 How is an LDA Topic Model being Trained?

References to functions and variables in this chapter point to the LdaModel class of Gensim. The model is trained using estimation maximization (EM). This is a statistical method of fitting distributions to existing data points in such a way, that they have a high likelihood given the current model. The likelihood is the product of all the data points probabilities and is high, when there are many data points close to the mean of the distributions. So this refers to the probability given the fitted distributions, not the probability of how often that document is observed or similar measurements.

It works similar to k-means, but only with probabilities and not binary classifications. That means, a data point has a probability that it belongs to a given cluster and not a single label.

The number of clusters is set beforehand, which is the number of distributions that are modified to fit to the data. This parameter is known as *num_topics* and k. Then there are also D data points x_d, which is the corpus.

The corpus is divided into chunks, that means a subset of the complete pool of documents, and the following steps are performed on that chunk. The parameters of the Dirichlet distributions, that are being fitted to the documents, are initialized randomly using gamma distributions. This approach is described in [1]. Note, that variables in Gensims code often have the logarithm applied to them, which means that the values are subtracted instead of divided.

There are two sets of Dirichlet parameters initialized from gamma. One for θ (theta), and k for the topic-word distributions called beta. Theta is a k-vector that describes from which topics a given document is influenced. For example by a factor of 0.2 from the sports topic and by 0.8 from the food topic. And each topic is defined by a multinomial probability of individual words.

1.3.1 Iterations, Inference and Expectation

The first step, also known as the expectation step in EM, fits θ onto the documents. Beta remains constant for now. For each document, the following iteration is done as often as the *iterations* hyperparameter or until it converged.

Theta is initialized randomly. The frequency of each word (token) in the document is compared with the probability of each word in beta, given theta. That means, when there are 4 topics, there are 4 probabilities for a given word depending on the topic. The probabilities add up given the weights in theta. This can be written as matrix multiplication. Example:

t: token, T: Topic

φ phinorm$_d$ =

Beta β

	t_1	t_2	...	t_N
T_1	.01	.09	...	.01
T_2	.02	.05	...	.08
T_3	.03	.01	...	.09
T_k	.01	.02	...	.07

0.4	0.2	0.3	0.1	.02	.05	...	.09
T_1	T_2	T_3	T_k				

Theta θ_d

The resulting φ is the expected probability of each word in the document.

Now the parameter for the Dirichlet distribution that provided theta as its mean value is changed in such a way, that the result of the previous matrix multiplication fits more close to the observed (true) word frequency. When a probability in φ is larger than the observed frequency of the n-th word, that means that the Dirichlet parameter that produced θ needs to be changed such that the topic that would produce more of the n-th word gets less weight and vice versa.

This is done by dividing the vector of the observed frequency (called *cts* in the code) with the vector φ element wise. A high frequency divided by a small probability yields a high value as a result, which means that more weight should be added to that word. But it is not providing any information about which topic needs to be changed yet.

Then another matrix multiplication with the transposed beta matrix is done. The shape of the divisions result is (1, N). The shape of the transposed beta is (N, k). This step weights it by the expected probability of the words in each topic from beta. The result is a k-vector that has a high value for those topics that should be closer to the document. Finally, the expected theta is multiplied onto it element wise, which means that those Dirichlet parameters (of the document-topic Dirichlet from which theta is drawn) will receive a stronger change that are assumed to be important for the doc than those, that are considered unimportant. Alpha is then added to it, which is the prior parameter that can be defined by the user. The result is a new gamma, which is used as the parameter for Dirichlet, which samples a new theta. When one of the gamma values increases, that means that the Dirichlet moves closer to one edge of the topic simplex. Which means the next sampled theta will be closer to a theta that describes that document.

The parameters that generated theta at the beginning are changed as well, though no new beta will be sampled during the iterations. The variable for this is *sstats*. It will be changed in such a way, that the likelihood of the document would increase given the same theta. *Sstats* is a matrix of the shape (k, V), similar to beta. It contains the parameters for beta, which is the expected value (or mean) from the Dirichlet distribution produced by each row of *sstats*. So there are k possible different Dirichlets that can be produced from *sstats*. The row that corresponds to a high value in the inferred theta for a document will receive the most change.

So there is quite some similarity with k-means. k-Means is competitive though and moves the nearest centroid closer to the data point. LDA iterates over data points as well, but moves all the distributions closer to it in a weighted way. But only after theta is estimated, which is another process of fiting a distribution to the data point. This is the difference to the classical EM algorithm. For the (k, V) topic-word Dirichlet parameters to be fitted, another (1, k) Dirichlet parameter for theta has to be fittet first. EM first calculates the probability of the data point given the clusters, which can be as simple as calculating the value of the data point in the probability density function, and infers the probability of the cluster given the data point using bayes rule for example. The result is a classification and based on that, the important distributions can be moved closer to the data point.

$$p(a \mid x) = p(x \mid a) \cdot p(a) / p(x)$$

1.3.2 Maximization

After that, the resulting *sstats* matrix is added to the *sstats* matrix of the model. One *sstats* matrix is approximated for every chunk of documents. Merging the models *sstats* and the new *sstats* is simply done by putting a weight w onto the newly inferred *sstats*, and putting a weight of (1-w) onto the old *sstats*. Then they are added together. As w and (1-w) adds up to 1, the magnitude of it does

not change. A property that was very handy in “4.1.1 Changes on the API“. The same way that alpha was added as user defined prior for calculating the mean of theta in the Dirichlet distribution, eta is added to *sstats* when calculating the mean of beta.

Then the algorithm continues to the next chunk of data. After the complete corpus was processed once, one pass has been done. When the number of passes is higher than 1, Gensim will iterate over all the chunks again. In the next pass at the beginning, a better beta will be used according to the previously gained knowledge.

1.4 How does LDA classify new documents?

New documents are classified using the same technique described in “1.3.1 Iterations, Inference and Expectation“. But this time, the internal state *sstats* will not be changed. The parameters for the Dirichlet that generates theta will be normalized and returned in that case.

2. Limitations of LDA

One important limitation of LDA is, just like many other machine learning algorithms, that it sometimes settles down on local minima depending on the random initialization. This is not a reliable behavior.

The second limitation is, that the number of topics is a hyperparameter that might influence the quality of the results.

3. Proposal of Ensemble LDA

3.1 Background, purpose, how does it solve the problems

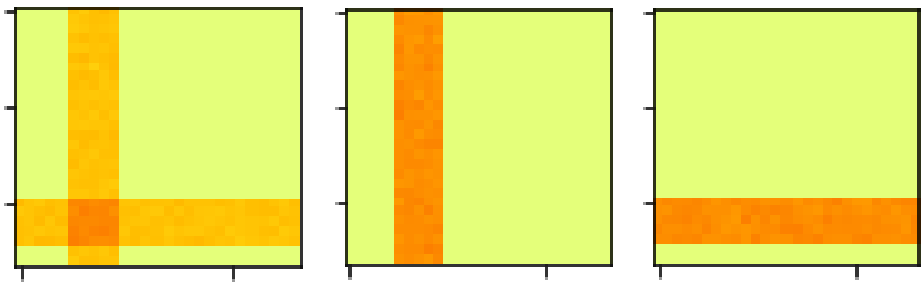

Figure 2: One learned topic next to two synthetic true topics. Each cell in this matrix represents a word, the darker the color the more likely a word is in that topic-word distribution. There are some words that belong to all topics. The learned topic to the left is a mixture of the two others, which is a local minima.

Each cell in this 24x24 matrix represents a word. During experiments a few of those synthetic, that means generated by a computer in a very defined way, topics were generated like this. Those are not topics learned by a model, those are “true topics” that can be used to generate training and test data. Each of those two dark bars marks the words that are characteristic for those.

The LDA is a generative model. One can generate new bag of words by randomly selecting words from the word distribution per topic. The generated text has no correct grammar.

A is the horizontal topic, B the vertical topic and AB is the mixture from Figure 2.

A and B are the true topics and AB is a restored topic that is heavily influenced by both topics. This would be an undesired case. And this is how the classical LDA model, as used in the Gensim package, can be unreliable. How to detect those topics that are invalid (without having prior knowledge of the synthetic/true topics), will be explained in “3.2 How the Algorithm Works“.

A third possibility besides restoring the true topic, and having a mixture of topics would be an empty topic. The probability distribution of words in a topic is normalized to sum to 1, so it is not actually empty, but it can be characterized by a lot of noise or uniform distributions.

In the ensemble, the number of topics is not a number that defines the size of the model output anymore, but rather works roughly as an upper limit of topics. But it is also possible to detect more topics than this hyper parameter in the ensemble, which depends on how other hyper parameters are set. 5 Models with 5 topics each produce 25 topics. There could be more than 5 stable topics in that list depending on the thresholds of the cluster sizes.

3.2 How the Algorithm Works

The Algorithm was first proposed in [8].

Assume topic A and B to be true topics that are generated by an algorithm. Topic AB is the mixture of both of those topics, to which the classic model converges sometimes.

$$AB = (A + B) / 2$$

To properly detect topic mixtures, topic A looks at topic AB and sees that **there is** a large portion of itself inside that mixture topic. Vice versa, when topic AB looks at topic A, it will see that there is a large chunk of itself **missing** inside topic A. By doing so, topic A can be detected as a **core** by the DBSCAN algorithm, whereas this is not the case for AB. And for this to work, one has to use an asymmetric distance matrix for the DBSCAN algorithm, which causes the distance from A to AB to be lower than the reverse distance. This is called ADBSCAN then.

The process of looking at how much of topic A is represented in topic B – also known as the distance from A to B – , can be done by:

1. Keep as many from the largest[ii] words from topic A, so that the probability of them sums up to 95%. This means the algorithm selects the largest 95% of the words by mass. Remove the other words, which will throw away the noise. From that, create a mask[iii] that contains a 1 for each word that is kept and a 0 for each word that is thrown away.
2. Remove the words from topic B that correspond to a zero in the mask of step 1 to create the filtered topic B. Both topic A and topic B are filtered. Note, that this might temporarily throw away words in topic B, that have a high probability. The goal is to see how good topic A is represented in topic B, so the algorithm doesn't care about the other words.
3. Calculate the cosine similarity between the filtered topic B and the filtered topic A. The **cosine similarity** is about the angle between two vectors using the dot product. The angle represents the "distance". The length of the vector is not important, only the direction is. Also note, that there is a maximum cosine distance, which is represented by 2, that can be achieved by topic B being topic -A. Since negative probabilities don't show up, the maximum in this case is 1. If the sum of probabilities in the filtered topic B is smaller than 5%, it is either mostly orthogonal to A or empty and the distance is set to the maximum of 1.

$$\delta(A,B) = 1 - \frac{A \cdot B}{|A| \cdot |B|}$$

4. Write that distance down into the pairwise asymmetric distance matrix. In Graph Theory, a **distance matrix** is a matrix that shows the distance from node A to node B, C, and so on. To look for the distance from A to C, the number in column 1 (A) and row 3 (C) is queried. This is called pairwise, because this matrix shows the distance between every possible pair of nodes. An asymmetric distance matrix means, that the distance in one direction (A → C) is different than the distance in the other (C → A). If one used a metric measurement between two Points in real world Space, it would be symmetric of course, but because of the masking in the ensemble LDA, the distance from the filtered topics A to B is different than the return distance. This happens because, because when topic A is represented in a topic mixture, that topic mixture is not well represented in topic A.
5. Repeat this process for every possible pair of topics of all the trained models from the ensemble. So there are pairs of topics from different models as well.
6. Now apply ADBSCAN to cluster the topics based on the distances from the pairwise distance matrix. When the distance from node A to B is **small**, node **A** is a valid topic. When the return distance, that means from node B to node A, is **large** at the same time, **B** is not valid. In that case B still supports A in becoming a core, but won't become a core of that

ii The most likely words.

iii A vector that contains binary numbers and that is used to select elements from another vector.

cluster itself. When the distance in both directions is small, both topics are similar and valid. Furthermore, for a topic to be identified as a core, multiple topics (“neighbors”) need to be close by. In the python package, this threshold is set to half of the number of models.

As the clustering takes a negligible amount of time to finish, the results can easily be reclustered with different hyper parameters to get the best results possible.

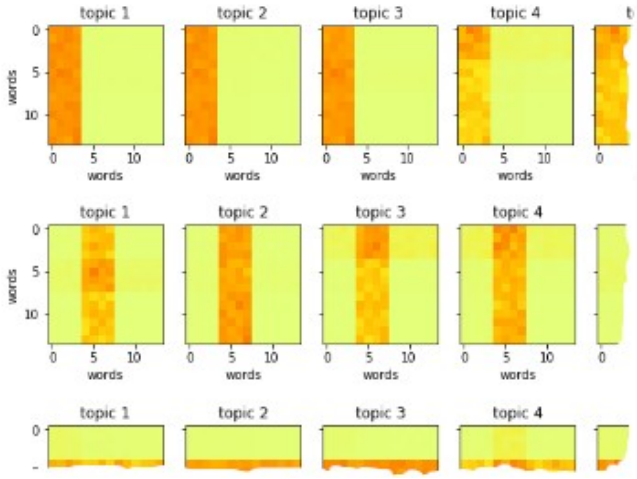

Figure 3: This is an example of how similar topics from various models will be clustered together. The mean of the topics in each cluster will represent a stable topic.

7. When a cluster is large enough, that means when it contains enough cores, it will be identified as topic. In the python package this is currently 1 plus the number of models divided by 4. Also, it is not higher than 3 cores and always 1 core or more.

8. The last step is to take the mean of the topic word distributions of each core and use this as the topic that represents the cluster.

For a more visual example, a corpus of 3 different words and documents that are a mixture of those words can be considered. The documents are mostly influenced by only one word, but sometimes the other two words are also apparent, which makes the documents appear noisy in Figure 4. The following visualization also gives insight about how the simplex plays a role in LDA.

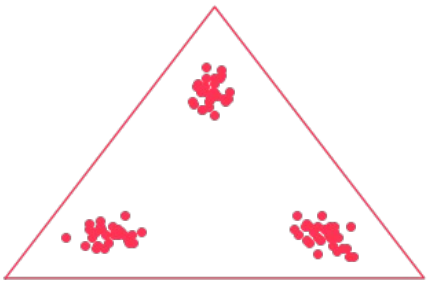

Figure 4: Documents placed in a word simplex. The probability of words in each document sums to 1 and is larger than or equal to 0, which forms the simplex shape. Each document is made out of a certain mixture of words, based on how close the document is to one of the vertex. The vertex of the simplex resemble 100% probability of one word, and 0% probability for the other words.

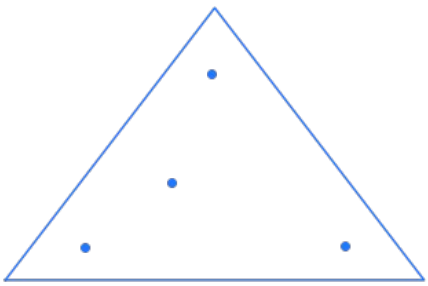

Figure 5: Now an LdaModel is being trained on that corpus with num_topics set to 4. Originally, 3 clusters were visible so obviously this parameter is wrong. However, the number of underlying topics is not always known. The fourth topic appears as topic mixture, which is the point that is close to the middle. Those 4 points are actually the vertex of the 3-Dimensional topic simplex.

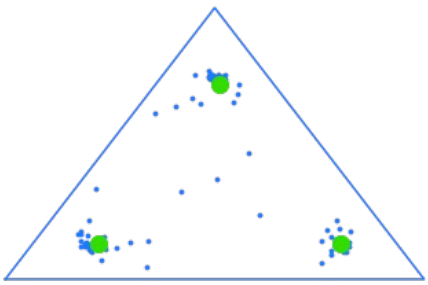

Figure 6: After training many models, the results can be clustered to remove topic mixtures. The green points are detected cluster means from the ADBSCAN algorithm.

3.3 Example

1. Assume the following example topics. T_k is the distribution of words in a topic.

T_1:

House 0.6	Garden 0.3	Roof 0.06	Floor 0.04

T_2:

House 0.6	Garden 0.2	Roof 0.19	Floor 0.01

2. Sort and set **mask** indices on T_1 based on the most characteristic 95% of the words in T_1 by mass, which will get rid of the noise, to calculate mask(T_1, T_2). So the probabilities are summed up from left to right as long as the sum is lower than 0.95.

T_1:

House 0.6	Garden 0.3	Roof 0.06	Floor 0.04

T_1:

1	1	0	0

3. **Mask** T_2. Masking means to select elements from the distribution T_2 based on the binary numbers from the list above. The total probability does not need to sum up to 1 after removing an element, because the cosine distance does not care about the magnitude.

T_2:

House 0.6	Garden 0.3

4. Now do the same for mask(T_1, T_1), which just throws away the lower 5% of the distribution by Mass. As can be seen here, mask(T_1,T_1) is very similar to mask(T_1,T_2):

T_1:

House 0.6	Garden 0.2

5. now the **cosine distance** has to be calculated. This is done using the following formula, which uses the dot product between vectors:

$$M_{12} = \delta(T_{1,}T_2) = 1 - \frac{mask(T_{1,}T_1) \cdot mask(T_{1,}T_2)}{|mask(T_{1,}T_1)| \cdot |mask(T_{1,}T_2)|}$$

$$M_{12} = 0.01$$

All those M_{ij} distances for each pair form the distance matrix M. The distance is very small in this example, which means the two topics T_1 and T_2 are very close together.

6. After doing this for a few topics, the result might be like this. 0.01 was rounded to 0 here:

M =

	1	*2*	*3*	…
1		**0**	0.3	…
2	0		0.2	…
3	0.9	0.2		…
…	…	…	…	…

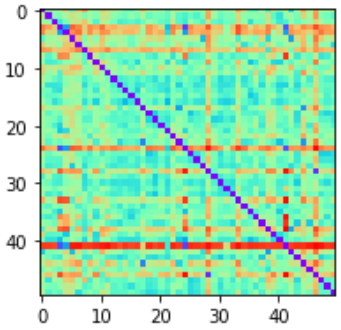

Figure 7: Example of a real world distance matrix. Cold is low, warm is high distance.

It is asymmetric because the pair ij does not result into the same as ji, it might be very similar or very dissimilar. The return distance to a distance is the number in this matrix on the opposite side of the diagonal.

7. It will **cluster** based on the distance between those topics using **CBDBSCAN**. Now assume Epsilon is 0.5. So Topic 1 (T_1) is close enough to T_2 and T_3. Assuming the required number of neighbors for a core is 2, it will successfully become a core of the first cluster. This is the first core of that cluster, so the used “Check Back” step is not required.

8. The algorithm will continue to search for members of that cluster recursively and proceed to T_2. The distance **from** T_2 **to** T_1 and T_3 is small as well, so T_3 will be the second core for the cluster, but only if the distance to 25% of the existing members of the cluster is also small. In this example it is obviously true, as it is close to the single existing member of the cluster.

9. The next element to check for is T_3, which is only close enough to T_2. Note, that the distance to T_1 is large, even though the distance from T_1 to T_3 is small. This might be the case because T_3 is a mixture of multiple topics. Since it does not exceed the number of required neighbors, it will not be identified as a core.

10. If the cluster has more elements than a predefined threshold, the mean of the cores is used in order to generate a new stable topic

3.4 Pseudocode

Step 1: Training the Models

```
initialize ttdas list empty (topic term distribution array)
for i in num_models do
   model = LdaModel(corpus, id2word, …)
   append ttda of model to ttdas
```

This trains all the models in the ensemble. The ttdas array will now contain all the topics from every model in the ensemble. It has as many rows as the number of models multiplied with the number of topics, and as many columns as the unique number of terms in the corpus. This step will take the most computing power. The following steps to do the clustering are comparatively easy to do. The corpus parameter is a mapping of token id to the number of occurrences in the training documents. The id2word mapping is needed to convert words in new documents to the right identifier. By doing so, the corpus can be converted to a list of numbers, which reduces the memory usage. From this ttda, the topic pairs for the distance matrix will be selected.

Step 2: Asymmetric Distance Matrix

```
initialize distance matrix M: (n_topics · n_models) x (n_topics · n_models)
for each topic Tₙ in ttdas do
   for each topic Tₖ in ttdas do

       Create mask from Tₙ: True for the largest 95% of terms by mass
       Remove terms from Tₙ and Tₖ that correspond to False in mask

       if Σ of tokens in Tₖ < 0.05 then
           Set δ = 1
       else
           Set δ = 1 – dot(Tₙ, Tₖ) / (|Tₙ| · |Tₖ|)

       Set Mₙₖ = δ
```

The important step that makes this approach of ensembling special is the creation of the asymmetric distance matrix, as it uses certain characteristics found in LDA topic models. All other steps can be altered more freely. Since the distance matrix M is asymmetric, classic DBSCAN will not be sufficient. For a neighbor to be a potential core of a cluster, the return distance needs to be small as well, which indicates that two topics are very similar. This is called Asymmetric Density Based Spatial Clustering. The following pseudocode describes a simplified version of it, a more complicated and better performing version is included in the current EnsembleLda package.

Step 3: ADBSCAN

```
Set eps = 0.1
Set label = 0

for each topic Tₙ in ttdas that is not visited do
   search(Tₙ, label)
   Increment label

def search(Tₙ, label)
   mark Tₙ as visited
   count topics Tₖ in ttdas for which Mₙₖ < ε
   if count > 3 then
       mark topic Tₙ as core of cluster label
       for each topic Tₖ in ttdas for which Mₖₙ < ε and Mₙₖ < ε do
           search(Tₖ, label)
```

The difference to classic DBSCAN is, when the return distance needs to be smaller than epsilon to expand the cluster in that direction. Also, neighboring points could be topic mixtures that are able to support multiple other topics in becoming cores, not only one.

Finally, the stable topics are created from the clusters. The threshold for "*#Topics*" below can be manually selected or dynamically based on the number of models for example.

Step 4: Generating Stable Topics from Clusters

```
for each cluster C do
   if #Topics in C > 3
       Set Tc = mean of all cores in cluster C
       Append Tc to list of stable topics
```

3.5 NMF Ensemble

Other approaches to making ensembles of topic models have been tried before. LDA is not the only algorithm to detect topics, there is also "Non-Negative Matrix Factorization" (NMF), which divides the document-term matrix (corpus), that contains probabilities of a term given a document, into two matrices H and W. H contains a row for each topic and is the probability of a term given a topic. W contains the classification of the documents, which is a weight vector that shows how much a document is influenced by each topic. For k topics, this weight vector is k elements large. It is then tried to find H and W in such a way, that performing a matrix multiplication of H x W will result in the document-term probabilities of the corpus. This cannot be algebraically solved so it has to be approximated. It is affected by instabilities and influenced by random initialization as well. To ensemble multiple of such NMF topic models, the topics from each model were concatenated, similar to the ensemble LDA as proposed in this paper. Then this concatenated matrix was used as the corpus for another NMF step. [2] [11]

4. The creation of a Programming Package for it

Basic python classes to handle the ensemble of classic LDA models have already been implemented, but were not ready to be used as a package. EnsembleLda is based on the topic modeling library called Gensim. Gensim offers the advantage of being independent of the corpus size, because a corpus that is larger than the available RAM can be loaded into the model by using the matrix market format [10]. The functionalities described in the following chapter were added during the work on this thesis paper.

4.1 Improvements of the Existing Code

4.1.1 Changes on the API

Most notably in the implementation was a missing clean API. The code was improved in a way that the ensemble could be trained by using a single constructor and as similar to classic Gensim models as possible. Also, the persisting[iv] of the model and the corpus was reimplemented in a more clean way using the “Persistable” python package. Based on Persistable, an object that could preprocess and persist text in a standardized way was also created to assist further development.

To make all of Gensims functions available for EnsembleLda, the stable topics from the ensemble were used to create a new Gensim model. A new prior eta was created from the learned topic-word distribution, from which a new LDA model was trained afterwards. The results – that means the learned topics from the model – were similar, but inaccurate. The new model itself had to be trained, even though the stable topics from the ensemble were already very precise, to make the new model converge based on this prior. Furthermore, the perplexity calculation was depending on the prior eta, so by modifying eta in that way, perplexity was rendered useless. However, there is a variable in the Gensim LDA model available that represents the training state that could be directly overwritten. This state variable is called “sstats” and for the best result, this sstats variable had to be created in such a way, that the classic model is able to return the exact same topics as the stable topics from the ensemble. The ensemble itself does not feature this sstats variable.

Sstats, topics as well as the stable topics are all matrices consisting of a row for each topic and a column for each individual word of the whole dictionary. Each cell contains the probability for a word to be of a certain topic.

The sum of the whole Sstats matrix will be normalized to the total number of words in the documents. Classic models calculate the final topic word distribution by adding the “eta” prior to sstats and then normalizing it in such a way, that the sum of each probability over words for a single topic equals 1. Since the prior eta does not change during the training and therefore has a fixed magnitude, the stable topics could easily be normalized to match the magnitude of the number of words plus the ensembles prior eta. In the following calculation, eta is the prior that was set as hyper parameter in the ensemble and sstats is unknown.

iv Persisting means to store the model in a file, to be able to analyze or reuse the results later

$$|eta| + |sstats| = m$$

$$stableTopics \cdot m = eta + sstats$$

Now the eta matrix has to be subtracted from it to get the sstats state variable.

$$stableTopics \cdot m - eta = \mathbf{sstats}$$

After creating a new classic model, its sync_state() function is used to finish the creation process. The constructor for this new classic model will receive 0 as passes and iterations. By doing so, no additional unnecessary training of the model will take place. Now that a proper Gensim model can be created, that contains the same state as the ensemble, each of the functions of the Gensim API can be used. For example the perplexity functions, with which the quality of the detected topics from the ensemble can be verified.

This Gensim model, that is used as classic representation of the ensemble, is created automatically after the training of the ensemble and stored inside of it. It does not take much computing power to do so, as the state is quickly synthesized from the stable topics. The goal was to make the EnsembleLda module fit as good into the existing ecosystem of Gensim models as possible, by making the API similar, so aliases for a few important functions were added to the ensemble, that forward calls to it to the internal classic model representation and return the results. This goes hand in hand with one of the goals of Gensim: “We wish to minimise the number of method names and interfaces that need to be memorised in order to use the package.” [10]

The calculation of the asymmetric similarity matrix M was rewritten to be more clean. Also, a few unneeded or obsolete functions and files were removed from the package to clean the code up.

4.1.2 Rating of Topic Stabilities

In the sourcecode of the classic LdaModel module, it says: “Unlike LSA, there is no natural ordering between the topics in LDA.” [10]. However, it would be very useful to have some sort of ordering of significance between the topics to make visualization easier. For this, a metric was created from the EnsembleLda algorithm results, that shows how stable a topic was by looking at the cluster size and the amount of cores and border points. A new function called “rateTopicStability” was added to the module that divides the number of topics in each cluster by the average distance. To calculate the average distance, the minimum of the forward and return distance between each pair of topics was used. The result is a density like metric.

4.1.2 Increasing the Ensemble Size

Any topic model that learns the topics as topic word distributions can be used to create this ensemble. For this, a function called “add_model” was added that takes a list of topic term distributions (ttd) as parameter and appends it to the current ttds in the ensemble. Afterwards, the asymmetric distance matrix is regenerated and they are are analyzed for clusters again. So previously trained models, even unstable ones as they will either support other topics in becoming cores or will have no influence as noise points, can be recycled to extract learned knowledge from them. Even when the models had different hyper parameters to start with. They all have to have the exact same id2-word mapping in common though.

4.1.3 Multiprocessing

As the training of the individual models don’t influence each other, the ensemble could easily be parallelized. A new parameter was added to the existing EnsembleLDA module, that takes the number of processes that are being spawned and sends the workload to those. The “multiprocessing” package in python proved to be easy to use for this purpose. The ensemble has the option to store all the models in memory and to process the topics from those models afterwards, however, as the models cannot be pickled so easily, they can’t be sent over the pipe to the parent process. The attributes of each model (for example sstats, eta) would have to be sent and the models reconstructed. However, for the ensemble it is sufficient to send the topics (topic word distribution, ttda) of each model over the pipe to the parent process. All following steps, including the computation of the distance matrix, were done sequentially in the parent process. To avoid overflowing of the pipes buffer, an individual pipe for each worker was created.

Ideally, the number of models should be a multiple of the number of workers, so that each worker has the same workload. For example, when workers is set to 2 and models set to 3, one worker will train 1 model and the other one 2. This means that the first worker will be finished earlier and the second worker will bottleneck the training.

4.2 API

The usage of the API is similar to Gensim. The model is trained using the EnsembleLda constructor. New documents can be classified by using the __getitem__ method through square braces:

```
from ensemble_lda import EnsembleLda
model = EnsembleLda(corpus=corpus, num_models=3, num_topics=10,
                    topic_model_kind="ldamulticore")
model[new_document]
```

The constructor introduces a variety of new keyword arguments additionally to those of the classic Gensim model:

topic_model_kind decides about which model in the Gensim package to use. LdaMulticore is recommended, as explained in the chapter “Lda or LdaMulticore“. Possible values are "lda", "ldamulticore", "lsi" and "hdp".

num_models is the number of classic models in the ensemble. Increasing this has a significant drawback on performance.

min_cores decides about how many cores a cluster from the ADBSCAN step has to have to be considered “stable”.

epsilon, the hyper parameter that represents the threshold for topic clustering distances in the ADBSCAN algorithm. This and min_cores is explained in “3.2 How the Algorithm Works“

ensemble_workers is a number and causes that many processes to be spawned, that distribute the model training between them. num_models should be a multiple of ensemble_workers, so that each worker gets the same workload.

5. Testing the Stability of Ensemble LDA

Various model setups will be tested to find out how the stability depends on the given hyper parameters[v]. Theoretically the ensemble LDA model is less prone to vary in its results after each training. After a given time as stopping criterion the results will be written down and the test will be restarted. Also, a comparison to the classic model will be shown that has the same time to learn the topics.

5.1 Validation

5.1.1 Perplexity

As real world data is not labeled and the LDA Topic Models are unsupervised, cross validation techniques can’t be used. Blei proposed the perplexity as a way to measure the quality of the models. [9]

Perplexity is used for showing the progressing convergence of a model, even though no labeled data is available. During experiments, perplexity often showed a relative quality between multiple model setups similar to other validation methods, that are going to be presented in this chapter. But it produced conflicting results when comparing classic and ensemble models, as will be seen in “5.2.3 Comparison with the Classic Model“. Perplexity estimates the log-likelihood of the hold-out test data given the models parameters.

v Parameters with values that the developer or user needs to define. They are not optimized using machine learning algorithms.

$$perplexity(D_{test}) = \exp\left\{-\frac{\sum_{d=1}^{M} \log p(\boldsymbol{w}_d)}{\sum_{d=1}^{M} N_d}\right\}$$

M is the number of documents in the corpus D, N_d is the number of tokens/words in a document d. D_{test} *is the hold-out test corpus.* $\boldsymbol{w}$ is a document. Note that the nomenclature makes a difference between a bold "$\boldsymbol{w}$" and a normal "w", which is a token/word. The perplexity is decreasing for a high likelihood[vi] of the hold-out test corpus given the trained model. [9] p(**w**) is the likelihood of a document given the model.

5.1.2 Compare Topics

The LDA Topic Model will not learn the topics in the same order each time. It depends on the random initialization. So the resulting matrix of topic-word distributions will have the rows, that represent the topics, in a random order. This however makes validating the model harder, as the expected topic-word distributions can't be compared row by row. A method proposed in [2] is able to compare the topics of various models and see which topics are identical. It therefore can be used to compare the true topics from synthetic data and learned topics. For this thesis it was modified to return a floating point value opposed to a binary classification, that is low for matching topics and gets higher for topics that are more different, by using the mean squared error (MSE). Using other distance metrics like cosine distance might also be possible. MSE proved to be sufficient.

This quality measuring algorithm iterates over the true topics (T_T) that were generated synthetically and selects the closest matching learned topic (T_L) from the model by the sum of squared errors. The distance between those two topics is added to the total quality. It then continues to iterate over all the other topics. Afterwards it is averaged by the number of true topics $|T_T|$ and by the dictionary length V. This step is optional, but it would make it easier to compare model quality on different synthetic corpus with a varying number of true topics by doing so, because the result will be the average error per word and per document.

$$\frac{\sum_{A \in T_T} \min_{B \in T_L} [(A-B)^2]}{|T_T| \cdot V}$$

This method even works, when $dim\ T_L\ !=\ dim\ T_T$, which makes this method of evaluation possible for comparing topic models that have different hyperparameters for the number of topics than the number of true topics.

vi The probability, that a specific sample is generated by a – continuous or discrete – probability distribution or model.

A and B can also be switched to show the quality of those topics, that were detected, which will ignore those topics that were not detected by the ensemble. This can be seen in Figure 16. For this, another topic is added to the true topics, which is an uniform distribution, to allow for empty topics that do not represent a topic but rather appears because the num_topics parameter was chosen too high in a classic model.

5.1.3 Cross Validate

Distances

As already mentioned, the order of the topics in the classified topics will be different for each model. That means one model might classify a document as *[0, 1, 0]* and another one will classify it as *[1, 0, 0]*, with both being correct. This makes classic cross validation impossible however, as this classification would have to be compared to a expected topic probability distribution element by element.

However, the distance between each of those probability distribution vectors will remain the same, now matter in which random order the topics are. The classification of a document (that means, to which topics a certain document belongs based on probability) will now be treated as the in dimensionality reduced[vii] representation of the documents. A symmetric pairwise euclidean-distance matrix between each classified document can be created, as well as a symmetric pairwise distance matrix between the expected classification. The difference between those two matrices is calculated and each cell is squared and summed together, which returns the sum of squared errors for the distances. As this is on a per document basis, the permutation of topics will not play a role anymore.

A downside of this could be, that the predicted vectors of the documents can be transformed to turn in a circle, which would still result in the same distances between them.

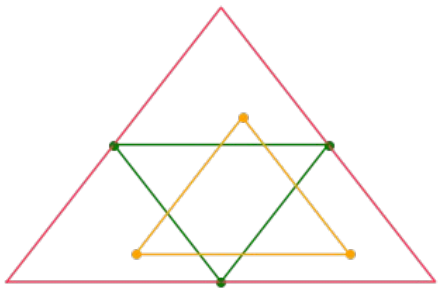

Figure 8: The green and the orange triangle result in the same distance matrix between data points (documents).

vii Reducing or compressing the amount of information in a way that possibly useful information is still preserved.

Entropies

This can be tackled to some degree by doing the same cross validation steps with numbers, that gives measure about how widely spread out the probability distribution is. *[0, 1, 0]* is sharp, as opposed to *[⅓, ⅓, ⅓]*, which is uniform. There are various methods of measuring this sharpness: magnitude, entropy, and variance. They all have slightly different characteristics, but all of them proved to be sufficient to show how close a predicted document-topic distribution is to the edges in the topic simplex.

However, experiments on the synthetic corpus have showed that both – the cross validated distances, and the cross validated entropies magnitudes and variances – show very similar results when using them to compare the quality of various models. This is actually expected for the tests with the synthetic data, because each documents is supposed to belong to mostly one true topic, in which case the entropy is low. The model is expected to converge to a prediction of low entropies, that means document-topic distributions like *[0, 1, 0]*. So anything that is closer to an uniform distribution shows that the model has a bad quality. And since this low entropy is expected, the vectorized documents do not have space to rotate in the topic simplex while maintaining their distance in the best case.

Conclusion

So either of both cross-validation methods is a valid way to validate the quality of models. They also showed to correlate strongly to the quality that was calculated from the other methods from “5.1.1 Perplexity“ and “5.1.2 Compare Topics“, as long as the models are not too different.

5.1.4 On Real World Data

Those ways of measuring quality proved to be not applicable to real world data. Some datasets provide categories or labels, but those might be split into multiple topics or a topic might me learned across multiple categories by LDA. So true topics cannot be extracted from the datasets. The quality can be examined in terms of perplexity, or by a human observer.

5.2 Experiments on Synthetic Data

The experiments on synthetic data in this paper are easy to visualize, validate and fast to learn, which makes it easy to make experiments to show certain characteristics of the ensemble and to prove if it works at all. Synthetic data was also used to quickly test new features of the code.

5.2.1 Generating the Data

Synthetic topics can be generated by making a list of words that belong to each topic. Then a topic-word distribution is generated from that. This can be made in such a way, that the topics have words in common. It is also possible to generate Topics that don’t have any words in common.

The method that is used for the experiments here creates a table, with each cell being a specific word. This table gets divided into multiple chunks of equal size. Each of those chunks represents a topic. Furthermore, each of the topics overlaps with other topics.

Additionally, unique words are added to each topic that only appear in that single topic. Experiments have showed that without those unique words, the prediction of both the ensemble and the classic model hardly converge to something good. A small number of unique words is already sufficient to make both models converge much better. Adding too many of them should also be avoided though, as training should not become too easy for the models because otherwise the instabilities might not appear in the results.

Also, new methods were written to plot synthetic data that was generated like that. Those were also used to plot the clusters in models, as seen in Figure 3, and to plot the results of the trained models.

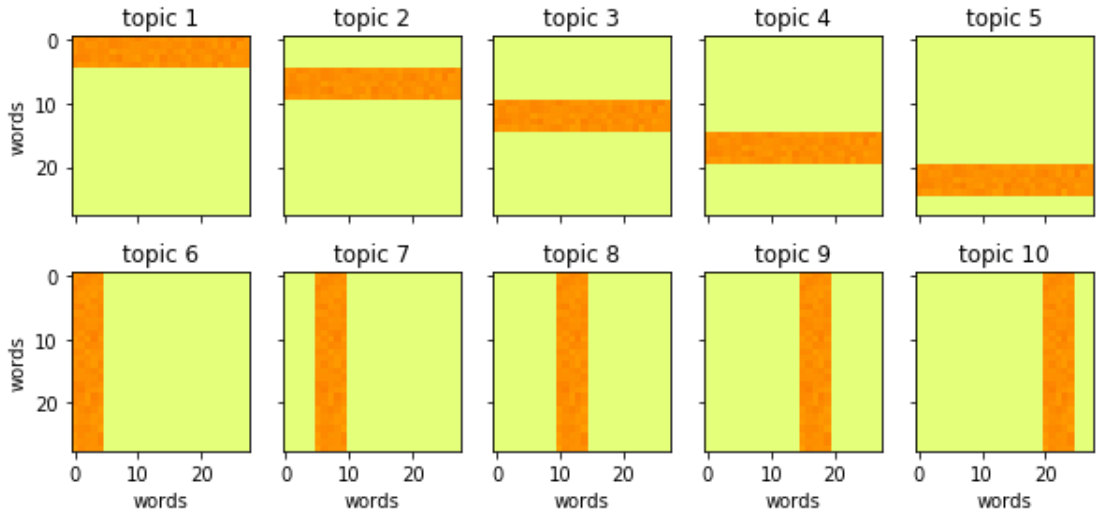

Figure 9: Each pixel represents one specific word, which is represented in each of those 10 heatmaps. Each heatmap represents one topic. The darker the color, the more likely a specific word is for this topic, which makes up an artificial topic-word distribution. The sum over the probabilities for each word is 1 in each topic.

This approach of generating data is inspired by [5].

Methods were written to make generation and visualization of this data as easy as possible.

For the experiments here, the number of true topics is set to 10. Each document has a length of 200 words, the dictionary (also called id2word) is 775 words long. 250 Documents are generated by chaining tokens that are drawn from the pool of possible tokens for a specific topic. Additionally, 25 documents per topic are generated as test data for testing the generalization of the models in the same fashion. True topics for validating the model as discussed in “5.1.2 Compare Topics“ are generated by counting the number of tokens in the generated documents, so that they properly contain the noise inside the training data which is created by the random sampling.

5.2.2 Parameters

Lda or LdaMulticore

As the ensemble could easily be parallelized without the need for a special implementation of the LDA algorithm by training the individual models in individual processes, it should be evaluated if the existing LdaMulticore algorithm is still the best performing one.

During the training of a regular LdaMulticore model, the usage of the cores was inspected using htop. It appeared that the algorithm is not able to fully use all the 4 available cores on the computer using this corpus. When using the newly implemented multiprocessing functionality on any kind of ensemble, the CPU usage of every core can be increased to be much closer to 100%, which results in faster training. The following figure shows how various configurations of the ensemble compete. The performance gain becomes negligible on larger corpus, as seen in “5.3.3 Amazon“.

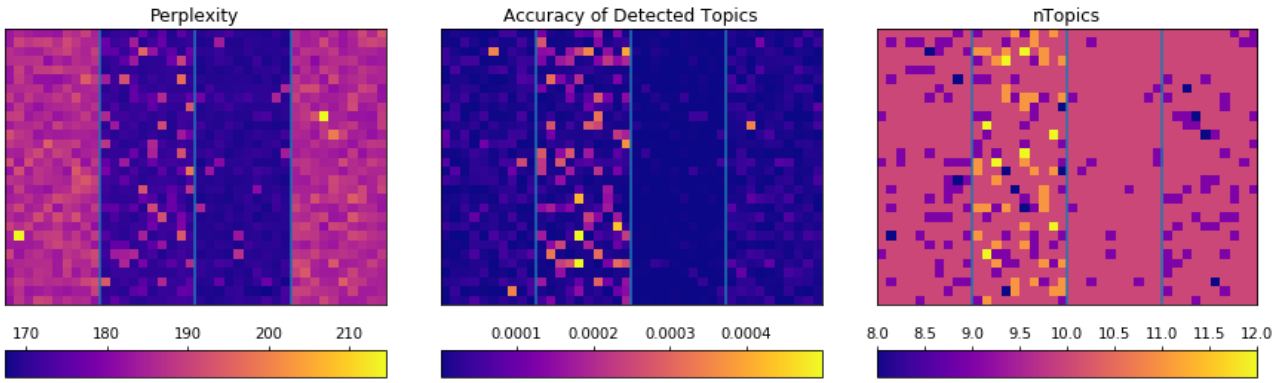

Figure 10: 4 different configurations on ensembles are compared in each plot. The first one uses LdaMulticore models with 3 workers, the second one uses LdaModels distributed to 4 processes, the third one a hybrid solution of LdaMulticore models distributed to 2 processes with 3 workers each, and the last one uses LdaMulticore models with 7 workers. All of them trained on the same corpus and have iterations set to equal values. The number of passes each model does over the training data was set in such a way, that they train approximately the same amount of time. Each of them was trained multiple times to show how noisy the results are, as each of them is initialized with a different random seed. The corpus was 10 topics large, so any number larger than 10 on the “num_topics” plot is caused by false positives.

According to the Gensim documentation, the ideal configuration for an LdaMulticore model is the number of cores minus one [10]. In the experiment, no performance gain was observed when increasing the number of workers over that value.

It can be seen, that the LdaMulticore ensembles are more reliable as they don’t detect too many topics for the synthetic corpus, which can only be the case with false positives. The quality of those topics, that were detected, also seems to be higher for the LdaMulticore ensembles. Therefore the LdaMulticore ensemble is chosen for the experiments. It can also be seen, that the number of detected topics does not correlate with the topics quality.

Furthermore, the most stable configuration appeared to be the LdaMulticore ensemble with 3 workers for each model, distributed to 2 processes (the third configuration in Figure 10). That model managed to do 17 passes in 31.6 seconds. Without distributing is to 2 processes, it only managed to do 10 passes in 32.1 seconds. Therefore, the performance of the ensemble was considerably increased. Note, that the corpus is very small and this performance increase is negligible on large corpus. Interestingly, the second configuration, which is an LdaModel distributed to 4 processes, even managed to do 36 passes in 32.4 seconds but still is the worst performing model in terms of “Accuracy of Detected Topics”. What is also very interesting is, that this and “Perplexity” show different ratings of the quality. The computer on which the benchmark was performed is powered by an Intel i5-3470 CPU @ 3.20GHz.

It is very important to note, that the RAM usage increases with an increasing number of parallel processes. So special care should be taken when a large number of workers is spawned.

When training the classic models in the ensemble sequentially, by default only the new topic word distributions (ttda) are stored, so the memory usage does increase with an increasing size of the ttda matrix. The ttda used in a later experiment on the English Wikipedia corpus is 100000 x 400 elements large and consumes 320 MB of memory.

The number of topics for the models that make up the ensemble can be higher than the number of actual detectable topics. When the algorithm converges, empty topics will show up in the model results, in which the topic-word distribution is either uniform or noisy. This depends on whether LdaMulticore or single core LdaModels were used:

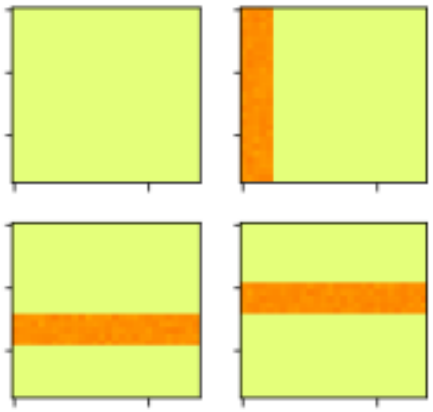

Figure 11: The 3 visible learned topics resemble true topics from synthetic training-data. The empty topic on the top left is uniform. This picture is taken from a non-ensemble single core **LdaModel**.

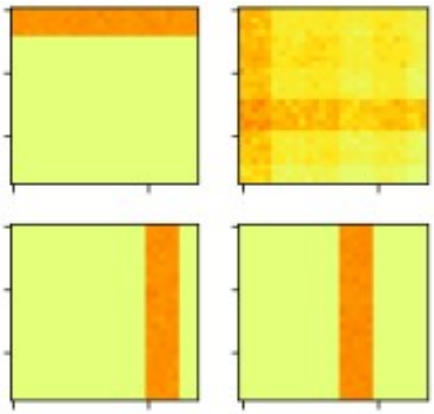

Figure 12: The same situation as in Figure 11, however, this model configuration filled the extra topic with a mixture of many other topics, which is very noisy. This picture is taken from a non-ensemble L**daMulticore** model.

This characteristic could be the reason why the ensemble performs so much better for the LdaMulticore model, as “empty” topics are topic mixtures of many other topics. Those topic mixtures are used to identify the stable topics.

Iterations and num_topics

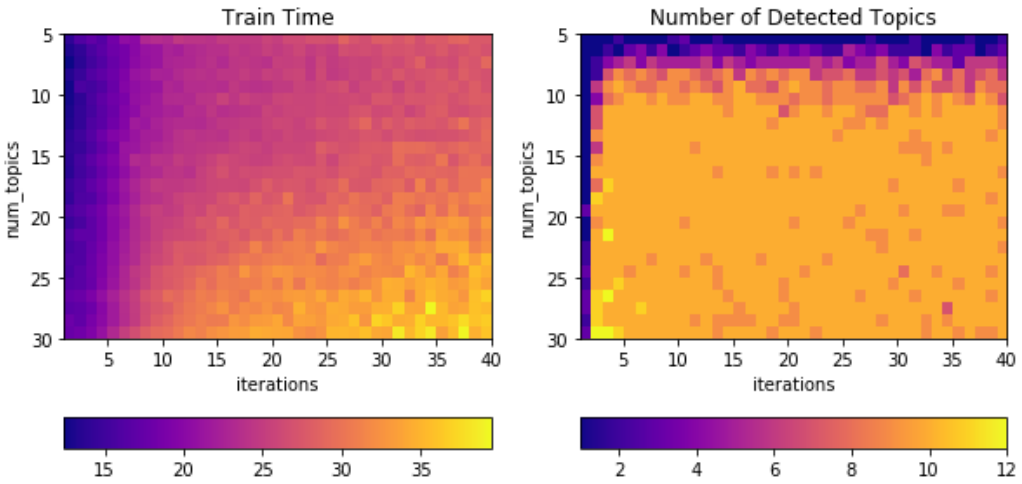

Figure 13: To the left, the time each model needs to train in seconds, to the right the number of detected topics. It can be seen that the model is mostly converged after 10 iterations, as iterations stop when the model converges.

No relevant improvement of the model quality is observed when num_topics is higher than the number of true topics. On real world data, this number is unknown, but a very high parameter of num_topics can be chosen with the only trade off being the models performance. Choosing the number of iterations is a matter of how fast the model is able to converge.

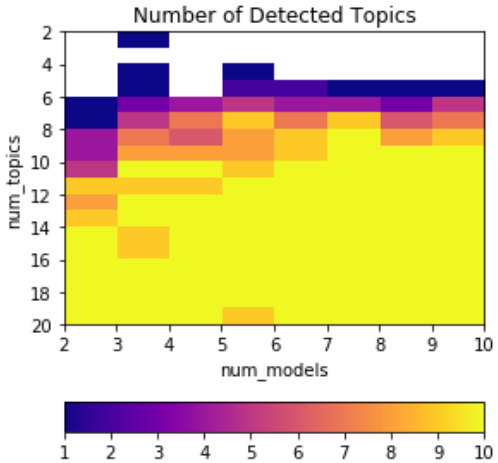

Figure 14: Test about the model stability given various values for num_models and num_topics. The number of true topics is 10.

A larger number for num_models seems to be able to compensate for only a slightly too small num_topics. Having the num_topics hyperparameter for the ensemble too low should be avoided, whereas it appears that it is not a bad thing to have num_topics set too high.

passes and num_models

iterations was set to 40, num_topics was set to 20. Ensembles were trained for every combination of passes and num_models. Instabilities were visible in the form, that a too low number of topics was detected.

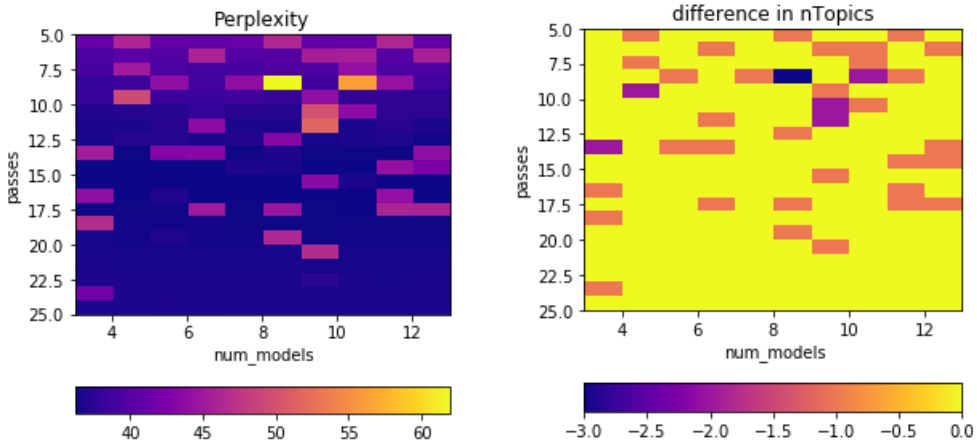

Figure 15: Typical instability of the ensemble: too few topics are detected. The difference between the expected number of topics (10) and the detected number can be seen on the right. It can also be seen, that the perplexity gets higher when the number of detected topics is not correct.

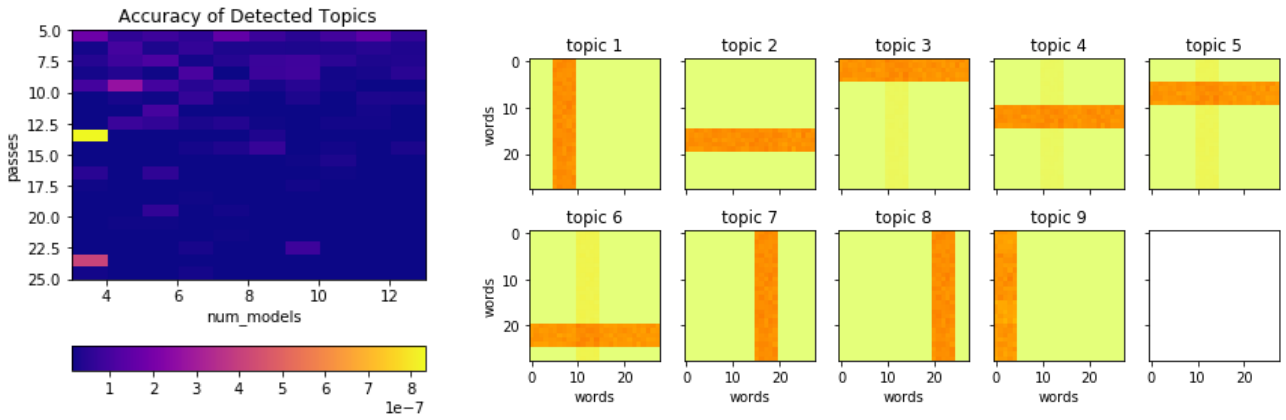

Figure 16: A bad ensemble that was able to detect 9 of 10 topics is shown on the right. Even though the model was rather unstable, the topics look very crisp and correct, besides that the shape of the missing topic can be faintly seen in topics 3, 4, 5 and 6. To the left can be seen, that the quality of those topics, that were detected, is similarly crisp for many other ensembles.

Passes for comparing two models

To compare two models that might pass faster or slower over the data, the number of passes for both models had to be set in such a way, that they approximately take the same amount of time to train. Since the goal is to show, if possible, if the ensemble works more robust than the classic model, the classic model was given slightly more time to train than the ensemble. The number of passes on the classic model was increased until it takes at least the same or up to 5% more time to train.

The time to train the ensemble rises approximately linearly – just like with the number of passes – with the number of models used in the ensemble, as can be seen here:

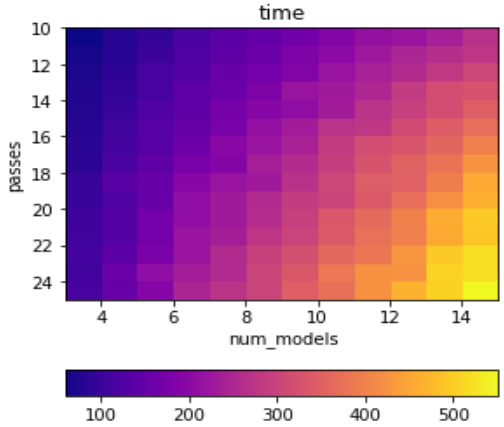

Figure 17: The time the training of a few ensemble LDA models took in seconds. The iterations and num_topics hyper parameters were set to 40 and 20.

This characteristic makes it very easy to figure out the number of passes as described. The ratio between the time the second model takes to the time the first model took can be multiplied onto the number of passes for the second model. If there is no overhead at all and the correlation between passes and training time taken is perfectly linear, it would find the best setup for passes in one single step. However, in the experiments this step had to be repeated a few times, often about 2 to 5 times, to find a setup that was good enough. For a very small corpus that can be trained very quickly, incrementing or decrementing in the number of passes can lead to a rather large change in the training time. Afterwards, the best possible configuration was chosen. As the performance of the model is not perfectly consistent and can vary, the models were trained multiple times before measuring the time to calculate the ratio.

5.2.3 Comparison with the Classic Model

For this experiment, the corpus was altered to be more difficult to train. It contains 14 true topics now. Documents are 200 words long. Each true topic generates 250 documents for training and 25 test documents. The dictionary is 1295 tokens large. There is a total of 770000 tokens in the whole corpus. Each topic contains 5 individual words that are not present in other topics.

The classic model was trained using the LdaMulticore algorithm with 3 workers. The ensemble had 4 LdaMulticore children with 3 workers each, spread to two processes. Both were doing 20 iterations. Being able to utilize more of the CPU than the classic model is an advantage, still, the ensemble has to train multiple models which is a major influence on the overall performance. In about 53 seconds of training for a model, the classic model can do 77 passes and each of the models in the ensemble can do 25 passes. When adding the number of passes in the ensemble together, it does a total of 100 passes therefore.

The ensemble can use approximately 94.25% per core and the classic model uses 67.5% per core on average on this corpus.

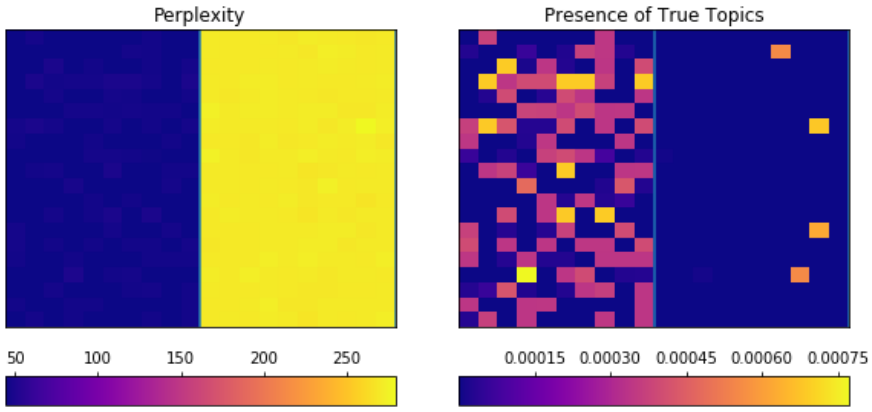

Figure 18: Comparison of two good configurations for synthetic data, both for the classic (to the left of the blue center line) and the ensemble model (to the right). The cooler the color the better. Even though perplexity is higher, the true topics were much more present in the ensemble. There, most ensembles, which are the models to the right of the blue center line, managed to detect all of the 14 true topics. The noise on the left side of the blue line suggests that this was harder for the classic model. This quality measurement is described in “5.1.2 Compare Topics“. It iterates over the true topics and compares them with the closest match in the models results.

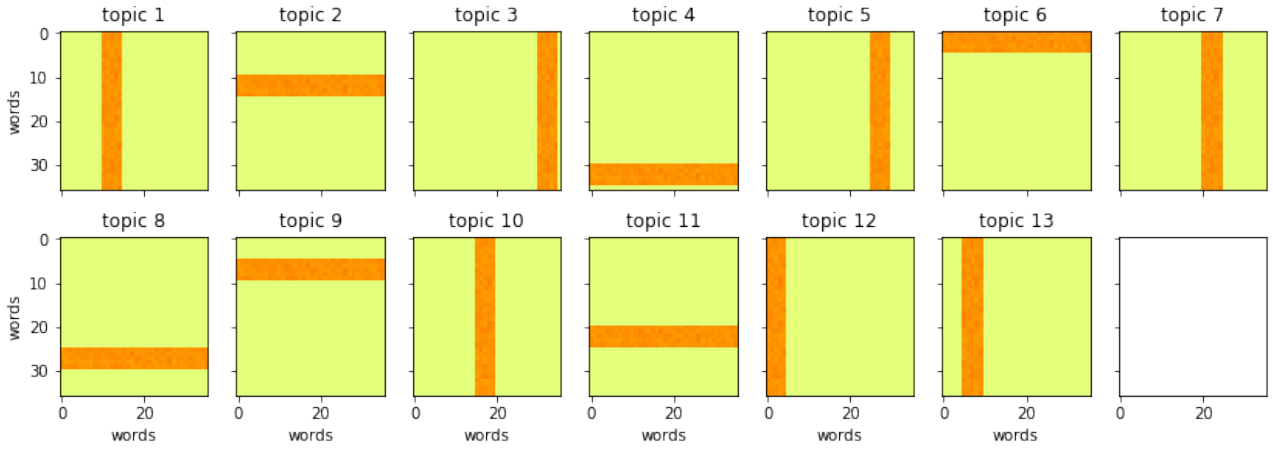

Figure 19: The worst ensemble. There are no noisy topics or topic mixtures visible at all. The perplexity is 279.8. This ensemble was able to detect 13 of 14 topics.

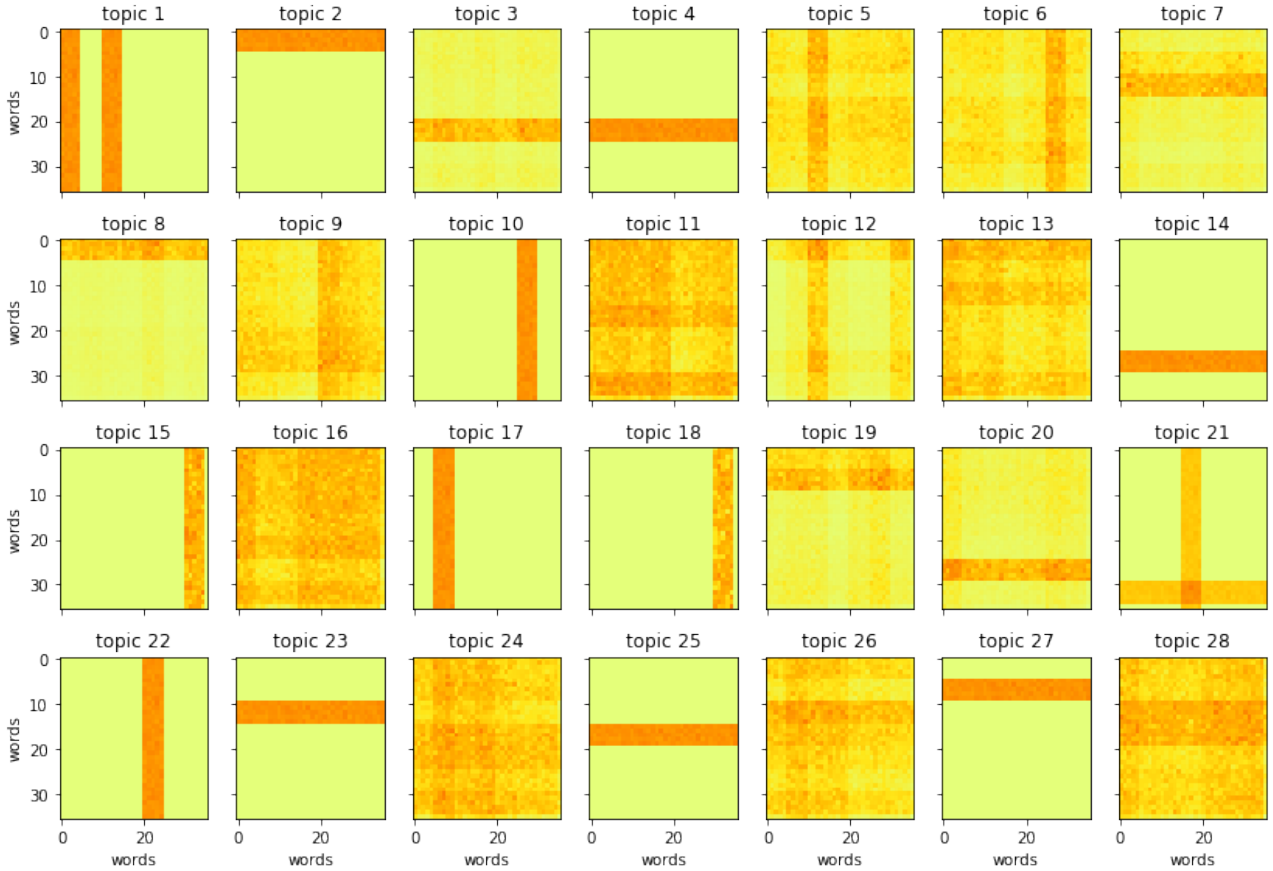

Figure 20: The classic model from the comparison with the highest perplexity. A lot of noise and topic mixtures are visible. It has a perplexity of 50.27

This also shows, that perplexity might not always work as expected and that it can be misleading when using it to compare two different kinds of models.

5.3 Experiments on Real World Data

The experiments here were done on an Intel i5-3470 CPU @ 3.20GHz.

5.3.1 Preprocessing

To classify the correct topics, preprocessing of the text should be done. Words, that are very likely to be in every text (for example “the”, “a”, “or”) – also called stopwords – can be removed, as well as words that have very few occurrences. By doing so the amount of data to be analyzed is reduced and the documents, that are meant to be of different topics, have less common words. Another way to reduce the amount of data is to throw away everything except nouns and verbs. The grammar is not important for the LDA model, only the probability of a word to occur in that text is important. In [9], a standard list of 50 stopwords was removed from the corpus as well as words that only occurred once. For the experiments here, a list of 179 stopwords from the nltk python package was used. They are in lowercase, so the corpus was lowercased first. Also, punctuation was removed because they are a matter of writing style and grammar and not of content. In a bag of words model like the LDA topic model, grammar is not important.

Decreasing the size of the corpus will speed the algorithm up as less words are there to be iterated over. In an amazon review corpus about electronics [4], about 44.2% of all the words are one of the 179 stopwords according to the nltk package, not counting those that are misspelled and therefore not detected.

The stopwords can be obtained like this:

```
import nltk
nltk.download('stopwords')
nltk.corpus.stopwords.words('english')
```

Then, the words are stemmed using the PorterStemmer from the nltk package, which makes sure that "*running*" and "*run*" are treated as the same word, so that documents from similar topics appear more similar, because otherwise "*running*" and "*run*" are treated as different tokens by the LDA topic model. The results are often words that appear chopped off, however, the goal is not to maintain the readability of the text corpus.

Obtaining a list of stopwords that are specific to the corpus and not included in nltk involved counting the number of occurrences of the terms throughout the corpus. The Gensim dictionary automatically counts in how many documents a term is apparent. The following code snippet extracts those term ids, that are visible in only 2 documents or less, and 1% of the most frequent terms but not more than 50. In the case of amazon review data, a very dominant term has been "use", which is not an nltk stopword, but which will be detected here:

```
n = min(int(len(dictionary)/100)+1, 50)
a = np.array(sorted(dictionary.dfs.items(), key=lambda x: -x[1]))[:n, 0]


k = 2
b = np.array([id[0] for id in dictionary.dfs.items() if id[1] < k])
```

As this relies on a finished dictionary, this was done after the corpus has been completely preprocessed. To remove words from the corpus, it was iterated over the documents, which have the shape of *[(1, 10), (5, 7), (token_id, count) …]* sorted by the token_id, and count was set to 0 for those special ids. Creating the corpus from scratch without the tokens in questions would also work.

So the token was not actually removed from the corpus or the dictionary, which means that measurements like the cosine distance should show a closer distance because suddenly some of the probabilities in the topic word distributions are set to 0, which means they are equal in the topics. This is not the case for the ensemble though, as tokens with low probabilities, and 0 is the lowest probability possible, are not taken into account in the calculation of the distance, because they get masked first.

Original	Preprocessed
Deep in the shady sadness of a vale \nFar sunken from the healthy breath of morn, \nFar from the fiery noon, and eve's one star, \nSat gray-hair'd Saturn, quiet as a stone, \nStill as the silence round	deep shadi sad far breath far fieri one star sat hair saturn stone still silenc round

Table 1: To the left is a small part of John Keats peom Hyperion. To the right, the preprocessed version of the left text. Paragraphs of the complete corpus were treated as documents.

5.3.2 English Wikipedia

The English Wikipedia corpus has been used to test LDA before and results are available online at https://radimrehurek.com/Gensim/wiki.html [10]. The current version as of June the 21st contains 4489460 documents and the doc bow is 10.5 GB large on the disk. The doc bow is a mapping of token id to number of occurrences in each document and is the input of Gensim models and called corpus. The documents are in average 284 terms long and range from a few dozen to a few thousand terms. This corpus was not preprocessed like the other real world corpus in this thesis.

Similar to how the LdaModel constructor was used on the Gensim documentation about the wiki corpus, the EnsembleLda constructor can be used to train it:

```
elda = ensemble_lda.EnsembleLda(corpus=mm, id2word=id2word, num_models=4,
                                passes=1, num_topics=100,
                                topic_model_kind="ldamulticore",
                                workers=3, initial_random_state=0,
                                chunksize=10000)
```

The performance could not be significantly increased using the multiprocessing ensemble as shown in “Lda or LdaMulticore“. The performance gain was only at 2.37%, most likely this is due to the LdaMulticore algorithm, that is used for the models in the ensemble, performing better on larger a corpus, because the CPU was already almost fully utilized.

An ensemble was trained that contains 4 models and does one pass over the corpus. Each model in the ensemble was trained sequentially after the other. This took 14 hours and 25 minutes. There was an additional classic LdaMulticore model trained to compare the results, that also did one pass over the corpus. This took 3 hours and 17 minutes. Both models, the classic and the ensemble, had a random state of 0. Note that the classic model could not be added to the ensemble to increase the size therefore, as it would just result in one of the ensemble children models to have a duplicate, because one of the ensembles children has a random seed of 0 as well.

One difference that was noticeable during working with the trained models was, that classifying documents with the classic model would have a zero probability for various topics usually. This was never observed in the ensemble. Furthermore, those zero probabilities are automatically removed

from the vector which influences the shape of the vector and therefore how easily those vectors can be used in further processing. There is a threshold hyper parameter involved in this which is called “minimum_probability” in the models constructor. The threshold can be set to 0, but will be 10^{-8}. By doing so, no probability was removed from the vector anymore.

The docstring of the show_topics function in Gensim says “there is no natural ordering between the topics in LDA” [10]. To sort the results and create such a natural ordering to make the visualization more meaningful, each document in the corpus was classified and then the average probability of a document being from one of the classes was calculated. For this, minimum_probability had to be set to 0 so that the vectors have the same dimensions. The topic that was most often classified is then considered the most relevant.

This way of rating topic relevance is rather slow, because classification takes some time. So another way was tried out. From the corpus the total word frequency per word was calculated and normalized in a way, that all the frequencies sums up to 1. Afterwards, each topic-word distribution of the models results was convoluted over that, that means first the vectors were element wise multiplied and then summed up. The idea was, that those topics with a high convoluted value would also be those that were the most relevant, as they heavily influenced the word distribution in the whole corpus. Convolution should result in a high value for similar distributions. But they turned out to be very dissimilar to the approach by classification that was explained before.

The Result from the Ensemble

Those are the 5 highest rated topics this way:

1. 0.014 ship, 0.012 navy, 0.010 ships, 0.007 naval, 0.006 boat, 0.005 vessel, 0.005 fleet, 0.005 submarine, 0.004 hms, 0.004 vessels
2. 0.009 protein, 0.006 gene, 0.006 cell, 0.005 acid, 0.005 cells, 0.004 proteins, 0.004 dna, 0.003 chemical, 0.003 bacteria, 0.003 enzyme
3. 0.009 indian, 0.009 tamil, 0.006 films, 0.006 kumar, 0.006 malayalam, 0.006 directed, 0.006 rao, 0.006 cast, 0.006 hindi, 0.006 telugu
4. 0.005 court, 0.004 law, 0.004 police, 0.003 rights, 0.002 act, 0.002 women, 0.002 justice, 0.002 case, 0.002 prison, 0.002 political
5. 0.020 historic, 0.012 building, 0.012 register, 0.010 places, 0.008 listed, 0.006 buildings, 0.006 story, 0.006 brick, 0.005 roof, 0.005 street

Examination of those topics by a human shows that they make sense, which means that the ensemble does not prevent the learning of meaningful topics. A total of 47 topics was detected, which shows that the thresholds that are used at the moment to mark clusters and cores as valid and that are used in the masking step of the algorithm as well as epsilon in ADBSCAN, are working, though it could be tried to do further optimization of those, especially when considering that the overhead of playing around with those after the ensemble is trained is rather small compared to the total train time.

Classic Model Results

Those are the 5 highest rated results from the classic model, rated in the same way:

1. 0.016 album, 0.011 song, 0.008 chart, 0.007 band, 0.006 track, 0.005 vocals, 0.005 guitar, 0.004 albums, 0.004 label, 0.004 listing
2. 0.011 league, 0.010 cup, 0.009 club, 0.008 goals, 0.008 championships, 0.007 apps, 0.007 round, 0.007 football, 0.006 tournament, 0.006 fc
3. 0.003 episode, 0.002 plot, 0.002 story, 0.002 police, 0.002 man, 0.002 reception, 0.002 cast, 0.002 mother, 0.001 said, 0.001 directed
4. 0.003 women, 0.003 law, 0.003 rights, 0.003 act, 0.002 social, 0.002 court, 0.002 organization, 0.002 policy, 0.002 health, 0.002 political
5. 0.012 football, 0.010 basketball, 0.009 coach, 0.008 conference, 0.008 league, 0.008 ncaa, 0.007 tournament, 0.006 baseball, 0.006 game, 0.006 games

So unfortunately the sorting was different.

Conclusion

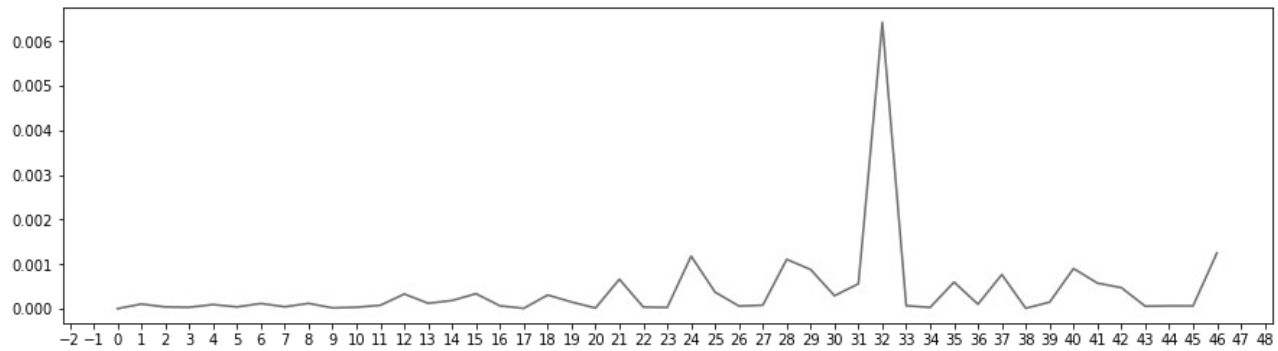

Figure 21: This shows how well the topics from the ensemble are visible in the classic model. Except for one topic those learned by the ensemble seem to be present in the classic model as well.

Topic 32 of the ensemble is the following:

- 0.044 australian, 0.040 australia, 0.030 queensland, 0.021 melbourne, 0.021 sydney, 0.015 adelaide, 0.015 wales, 0.015 brisbane, 0.010 locality, 0.010 afl

It seems to be about australian geography mostly, but also the australian football league is mentioned. The closest topic in the classic model seems to be more about football, but also about australia:

- 0.022 discal, 0.012 queensland, 0.011 sydney, 0.011 rifle, 0.010 tackles, 0.008 nsw, 0.008 brisbane, 0.008 scout, 0.007 wales, 0.007 halfback

It is very hard to tell if the quality is good or bad or if there are topic mixtures. The english wikipedia corpus is so big, it probably contains many more topics that are not yet detected.

5.3.3 Amazon

A corpus consisting of reviews has the great advantage, that they are labeled. This way, the complexity of the corpus can be controlled by allowing for more or less categories and therefore potential topics. It should be noted here, that the categories don't translate to true topics. Consider review data from laptops and from smartphones. They both will contain information about the speed, the screen, the WiFi connection and the storage. Also, a single category can contain multiple topics. The food corpus contains among others "coffee", "tea" and "bread" as keywords that might be assigned to individual topics.

Amazon review data is easily available to download online. Compressed Archive in the .gz format can be obtained here: http://jmcauley.ucsd.edu/data/amazon/, in the section of the small subsets. The download labeled with "5-core" is the one that was used. The whole corpus contains 18.186.733 documents. Once decompressed, there is one .json file for each category that contains one rating per line. Each line is a JSON representation of the review. [4]

The relevant key in the JSON objects is "reviewText". The other keys like "asin" or "reviewTime" are not needed. Not all categories were downloaded, it was decided to only use "Musical Instruments", "Pet Supplies", "Grocery and Gourmet Food", "Baby", "Beauty", "Health and Personal Care", "Home and Kitchen" and "Electronics" to reduce the needed time for training and preprocessing and to make the examination of the results by a human easier. The idea is, that more experiments can be done that are easier to interpret. Furthermore, the number of documents in each category was restricted to a maximum of 100.000 documents.

To speed the processing of the JSON files up, they were not parsed, but rather a substring based on the key "reviewText" and the following key "overall" was done. This was 7.56 times faster.

```
reviewText = line[ line.find('"reviewText":')+12 : line.rfind('"overall":') ]
```

While iterating over the lines and the files, an id2word mapping was extended to contain new tokens. This could easily be done using methods provided by Gensim:

```
doc = preprocess(reviewText.split())
dictionary.add_documents([doc])
```

with "dictionary" being a gensim.corpora.Dictionary object. The "preprocess" method that was written for this in the experiment setup removes stopwords, exclamation and and stemms the words using the nltk.stem.porter.PorterStemmer class, as described in "5.3.1 Preprocessing".

This makes for a corpus of 710268 tokens. 10% of those were held out of the training process as test documents. The average document length in the train set was 45.3 tokens. Depending on the selected size of the corpus, it might not fit into the RAM. Luckily, Gensim provides solutions for this using the gensim.corpora.MmCorpus class, which can store corpus and dictionary to a matrix market file that can be used to supply the model with data without loading the whole corpus at once.

Complete Corpus

2 ensembles were trained. Both with 4 models, one of them multiprocessed to 2 workers. The performance gain of 3.6% for the multiprocessed one was negligible again because they already utilized almost the complete CPU. Both models can be combined to form an ensemble of 8 models, as they both have different random seeds. It is important to note that the random seeds of the models in the ensemble are currently set to 0, 1, 2 and 3, when the random seed of the ensemble is set to 0. So the second ensemble has the random seed set to something > 3, namely 1000. The training of both ensembles took 51.3 minutes. On this corpus, frequent words like “use” were not removed.

The models used were LdaMulticore models, 1 pass, 100 topics, 50 iterations. The combined ensemble that contains the 8 classic models learned 11 topics, and they look rather different and unique, so no duplicates are visible in the results. One of the classic models in the ensemble, that was closer examined, appeared to actually learn a hundred different topics. Note, that the words are stemmed so they appear chopped off sometimes.

1. 0.339 len, 0.025 lens, 0.021 use, 0.016 eye, 0.010 shot, 0.010 view, 0.009 imag, 0.009 get
2. 0.292 tea, 0.026 oil, 0.021 organ, 0.016 use, 0.014 tree, 0.014 coconut, 0.013 season
3. 0.296 mm, 0.017 use, 0.014 memori, 0.014 optic, 0.011 juic, 0.010 fm, 0.009 bodi
4. 0.248 coffe, 0.034 brew, 0.024 tast, 0.014 ground, 0.013 drink, 0.013 amp, 0.012 caffein
5. 0.088 size, 0.045 small, 0.036 fit, 0.027 larg, 0.024 one, 0.023 smaller, 0.020 larger
6. 0.355 cabl, 0.049 f, 0.038 usb, 0.027 cover, 0.012 disk, 0.011 printer, 0.010 use, 0.009 work
7. 0.240 filter, 0.019 use, 0.014 dust, 0.012 food, 0.012 air, 0.011 water, 0.010 appl
8. 0.059 flavor, 0.057 tast, 0.038 like, 0.025 chocol, 0.023 good, 0.021 sweet, 0.017 bar
9. 0.286 router, 0.030 transfer, 0.030 recept, 0.016 block, 0.012 flake, 0.012 password
10. 0.160 bag, 0.023 pocket, 0.021 carri, 0.019 case, 0.016 use, 0.011 great, 0.010 one, 0.010 fit
11. 0.149 sound, 0.025 nois, 0.022 music, 0.021 listen, 0.018 ear, 0.017 volum, 0.017 hear

Another attempt was done with num_topics set to 20 and an otherwise similar setup. Now, only one topic is detected:

- 0.026 tast, 0.023 flavor, 0.019 like, 0.013 good, 0.010 make, 0.009 eat, 0.008 food, 0.008 coffe, 0.008 chocol, 0.008 sweet,

Creating clusters for a specific configuration without evaluating the quality takes only 0.451 seconds on average on the used computer, without the creation of the asymmetric distance matrix, which remained constant throughout the tests. So it was very easy to try various settings and find a setup that detected more topics, but that might not even be the goal of the ensemble, as it should detect the stable topics and not as many topics as possible.

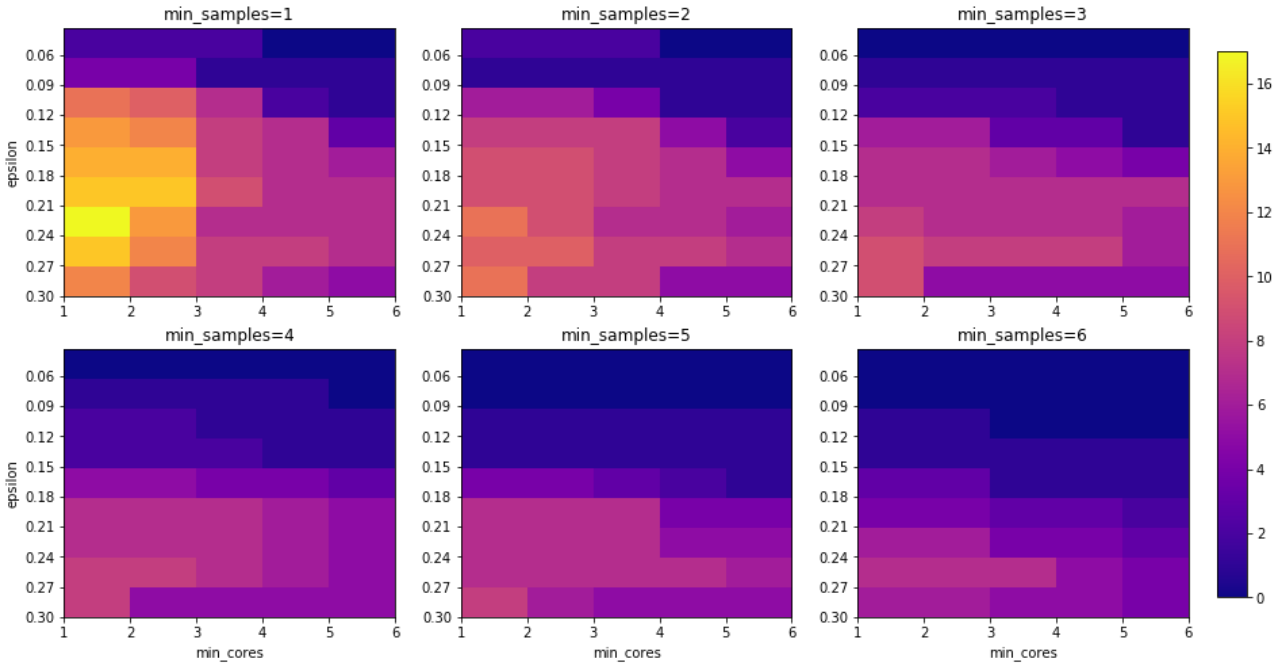

Figure 22: How the number of detected topics is affected by 3 of the clustering parameters for this corpus. This shows, that the ensemble results are strongly influenced by the choice of those parameters.

For this ensemble with num_topics=20 which detected only one topic, Epsilon was then set to 0.21, min_samples to 3 and min_cores to 4 and after clustering the ensemble again, 7 topics were detected:

1. 0.026 tast, 0.023 flavor, 0.018 like, 0.015 coffe, 0.012 good, 0.010 make, 0.008 eat, 0.008 tri
2. 0.086 len, 0.018 use, 0.009 lens, 0.008 one, 0.007 get, 0.007 good, 0.007 like, 0.006 look
3. 0.018 use, 0.014 work, 0.013 cabl, 0.009 one, 0.009 camera, 0.007 get, 0.007 need
4. 0.079 tea, 0.023 oil, 0.014 use, 0.010 coconut, 0.010 like, 0.009 organ, 0.007 product
5. 0.025 price, 0.018 product, 0.018 good, 0.016 amazon, 0.015 buy, 0.014 great, 0.013 qualiti
6. 0.019 day, 0.017 time, 0.017 use, 0.013 get, 0.013 tri, 0.013 work, 0.012 year, 0.012 month
7. 0.016 fit, 0.013 use, 0.009 one, 0.009 like, 0.008 get, 0.008 strap, 0.008 would, 0.008 well

The already mentioned words like “use” that are very frequent for this individual corpus were not removed, so they are visible in many of the topics.

The number of detected topics is strongly affected by the hyperparameters, so the question that arises is, how many topics are needed. More restrictive the hyperparameters mean, that the number of topics in a cluster needs to be higher and/or the distances to each other need to be closer, which also means that less topics are identified.

Smaller Corpus

The complexity of the corpus was further reduced to only contain the categories "Health and Personal Care", "Beauty", "Baby", "Pet Supplies", "Musical Instruments". It contains 45000 documents for training purposes and 5005 documents for testing. The average document is 42.19 tokens long.

This shorter corpus makes training faster and examination of the results by a human easier. This time, the 50 most frequent words and 15126 words that were present in only 2 documents or less were removed. This is actually about half of all tokens in the dictionary, still, the documents got modified only little by removing them, because their mass is so low.

16 ensembles that contain 16 children were trained. num_topics was 20, iterations 30 and passes 1. From that, 8 ensembles with 32 children were created and 4 ensembles with 64 children, by recombining them.

None of them detected more than one topic, which was about “guitar”, using the default configuration.

Epsilon was then set to 0.15, min_samples to 3, 6 and 12 depending on the ensemble size and min_cores to 3. The ensembles with 16 children were rather unstable, they detected between one and 5 topics. The ensembles with 32 children were rather stable and detected between 3 and 5 topics and always including “guitar”, “skin” and “sound equipment” topics. The four 64 children ensembles were the most stable and detected similar topics to:

1. 0.069 guitar, 0.005 tuner, 0.005 play, 0.005 sound, 0.005 pick, 0.004 acoust, 0.003 review
2. 0.050 pedal, 0.009 gate, 0.007 sound, 0.005 effect, 0.004 metal, 0.004 tone, 0.003 tuner
3. 0.029 skin, 0.011 dri, 0.009 face, 0.008 smell, 0.006 moistur, 0.005 oil, 0.005 wash
4. 0.050 amp, 0.006 sound, 0.005 bottl, 0.004 say, 0.004 effect, 0.004 skin, 0.004 set
5. 0.045 string, 0.009 sound, 0.008 play, 0.005 strap, 0.005 pick, 0.004 differ, 0.004 toy

And one of them additionally detected the following topic:

6. 0.037 bottl, 0.010 water, 0.006 dog, 0.005 pump, 0.004 purchas, 0.004 problem, 0.004 lot

Indicating, that an increased number of children makes the ensemble more stable. Topic 1 and 5 look very similar, it could be tried to increase epsilon so that they cluster together.

Reclustering is fast, as long as the asymmetric distance matrix is not created again. With that, it takes 16.2 seconds to recluster one of the 16-children ensembles. Without it, only 0.28 seconds. Speeding this up is discussed in “6 Optimizing the Asymmetric Distance Matrix“.

For comparison, 8 classic LdaMulticore models were trained with num_topics = 20, passes = 19 and iterations = 40. By using a small script that groups topics based on a cosine distance threshold of 0.25 up, the following topics were extracted from those 8 classic models. Counting them up by hand would be a rather difficult job as they don’t always look obviously similar, because the order of the significant tokens in each topic are randomized a little bit.

8x 0.032 guitar, 0.024 sound, 0.021 string, 0.018 pedal, 0.014 play, 0.012 amp, 0.012 pick
8x 0.020 skin, 0.017 hair, 0.011 smell, 0.010 dri, 0.009 color, 0.008 face, 0.006 oil
7x 0.015 dog, 0.008 toy, 0.007 strap, 0.007 fit, 0.006 old, 0.006 size, 0.006 brush
3x 0.009 cabl, 0.008 batteri, 0.007 power, 0.007 amazon, 0.006 qualiti, 0.006 box, 0.006 mic
6x 0.018 water, 0.018 bottl, 0.011 food, 0.011 tank, 0.010 cat, 0.009 fish, 0.008 filter
4x 0.030 dog, 0.015 cat, 0.013 toy, 0.012 brush, 0.010 tub, 0.008 food, 0.007 old
2x 0.014 tank, 0.014 water, 0.012 filter, 0.011 fish, 0.011 food, 0.007 test, 0.007 amazon

1x 0.010 test, 0.007 help, 0.006 read, 0.005 review, 0.005 head, 0.005 replac, 0.005 give
1x 0.019 hair, 0.013 water, 0.012 color, 0.012 tank, 0.012 fish, 0.012 food, 0.010 smell

So there is much more variety than in the ensemble. The same technique can also be used for the classic models with 20 topics each, the result is a list of 77 different topics. 66 of them were not present more than 4 times in the 8 classic models. 7 managed to be in 7 or 8 models. This shows how varying the results of the classic models can be, because each of them contains individual and unique unstable topics besides those 7 stable ones.

5.3.3 Opiniosis

Opiniosis is an extremely small corpus that contains 289 product reviews for 51 products [6]. 8 ensembles have been trained with 128 LdaMulticore children each. num_topics was 20, passes 20 and iterations 100. To make visualization easier, they were grouped based on a cosine distance of 0.3 again:

8x 0.179 clean, 0.074 bathroom, 0.033 issu, 0.017 gener, 0.017 mani, 0.014 food, 0.013 great
8x 0.002 clear, 0.002 voic, 0.002 food, 0.002 keyboard, 0.002 sit, 0.002 stylish, 0.002 pretti
8x 0.193 free, 0.055 park, 0.045 even, 0.024 internet, 0.021 wine, 0.021 coffe, 0.015 use
8x 0.137 friendli, 0.119 staff, 0.101 help, 0.014 hotel, 0.008 long, 0.008 front, 0.006 make
6x 0.098 keyboard, 0.067 function, 0.047 use, 0.042 get, 0.037 expect, 0.032 smaller, 0.029 touch
2x 0.158 mileag, 0.106 ga, 0.039 expens, 0.035 high, 0.015 front, 0.013 food, 0.011 car
2x 0.167 price, 0.042 hotel, 0.035 high, 0.034 kindl, 0.025 howev, 0.024 expens, 0.023 book
7x 0.189 speed, 0.067 limit, 0.060 map, 0.044 updat, 0.037 fast, 0.015 great, 0.013 navig
1x 0.065 time, 0.060 bright, 0.055 batteri, 0.054 use, 0.049 mani, 0.046 drive, 0.046 run

6 of 9 topics could be detected very reliably. The ensembles detected between 5 and 8 topics.

To compare this to non-ensembled models, 8 classic LdaMulticore models were trained with 40 passes and 100 iterations that are set to detect 7 topics each. The following topics were detected 3 or more times:

4x 0.046 seat, 0.041 great, 0.041 perform, 0.036 video, 0.036 front, 0.026 uncomfort
3x 0.060 clean, 0.051 small, 0.046 well, 0.037 mani, 0.033 bright, 0.024 bathroom, 0.019 nice
3x 0.056 friendli, 0.043 staff, 0.039 help, 0.031 make, 0.031 mileag, 0.026 kindl, 0.022 new

It can be seen, that this corpus is very hard to learn and the results between each model are very different, because otherwise some topics would be much more often visible. Afterwards they werwe trained with num_topics set to 20. They yielded some topics that were present in almost all models. About 100 different topics were detected, however, this number is hard to make out as they are sometimes quite similar and sometimes just a little bit, so there is no definite answer to that. There is a huge list of topics that look very different and that are visible in only few models. The threshold for counting those topics up was set to 0.35.

8x 0.075 keyboard, 0.054 function, 0.043 use, 0.033 touch, 0.033 get, 0.033 smaller, 0.023 hotel
8x 0.139 friendli, 0.117 staff, 0.088 help, 0.063 well, 0.038 font, 0.025 adjust, 0.025 except
7x 0.169 clean, 0.062 bathroom, 0.047 larg, 0.032 life, 0.032 batteri, 0.032 great, 0.032 problem

7x 0.161 free, 0.081 food, 0.047 park, 0.035 perform, 0.034 even, 0.024 poor, 0.024 overal
6x 0.172 speed, 0.098 mileag, 0.062 limit, 0.062 ga, 0.037 road, 0.037 map, 0.025 model
5x 0.142 seat, 0.083 front, 0.072 uncomfort, 0.054 howev, 0.019 back, 0.019 hard, 0.019 find
5x 0.082 mani, 0.068 bright, 0.055 batteri, 0.041 window, 0.041 run, 0.041 time, 0.040 use
5x 0.106 interior, 0.049 roomi, 0.039 updat, 0.029 map, 0.029 much, 0.029 use, 0.029 hard

So about 5 topics can be detected quite reliably out of about 100 different topics. These include topic mixtures and empty topics as well, as the classic models don't have a way to remove those from the results. The way Ensemble LDA works is actually quite similar to grouping the topics up like that. The difference is, that topic mixtures are used to identify stable topics, that ADBSCAN clustering is done and that stable topics are the average of the DBSCAN cores.

The speedup that can be achieved by multiprocessing the ensemble children was very dominant. The LdaMulticore algorithm did not manage to fully utilize the cores. 3 workers need to be created to almost fully utilize the CPUs capacity. The ensembles here were trained on 4 workers.

Without the creation of the asymmetric distance matrix, an ensemble without multiprocessing needed 287 seconds. With 4 workers, only 83 seconds. The creation of the asymmetric distance matrix takes more time than to train the 128 children of the ensemble. 503 seconds are needed only to do that, as it is a 2560 x 2560 matrix. Using the multiprocessing method described in "6.2 Multiprocessing with Workers", it can be reduced to 135 seconds.

6 Optimizing the Asymmetric Distance Matrix

The generation of the asymmetric distance matrix was a major bottleneck for one of the advantages of ensembling: the fast reevaluation of it. There are two possibilities in how the algorithm can be changed to be faster in python: difference distance metric and different masking method. Then there is also the possibility of parallelizing. The following tests were done on one of the 16-model ensembles of "5.3.3 Amazon", which were trained on the amazon corpus.

6.1 Methods in the Algorithm

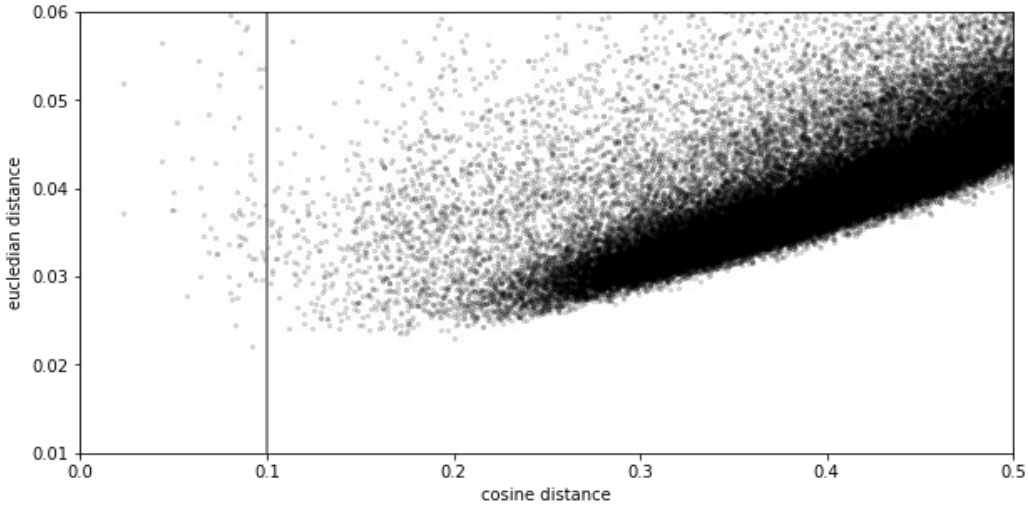

Figure 23: Dots (topic pairs) to the the left of the vertical line, which is the default **epsilon** parameter, are considered close in ADBSCAN using cosine distance measurements and default parameters. Each dot represents one vector of (euclidean distance, cosine distance) in the asymmetric distance matrix, which means it is the distance of a single topic pair. Return distances are included.

The masked topic was normalized to sum to 1. Normalization does not make a difference for cosine distance, but for euclidean it does. An important question here would be, if selecting an euclidean threshold of for example 0.025 produces stable results or not. The euclidean distance speeds up the creation of the distance matrix from 15.65 seconds to 9.80 seconds, which is 1.60 times as fast.

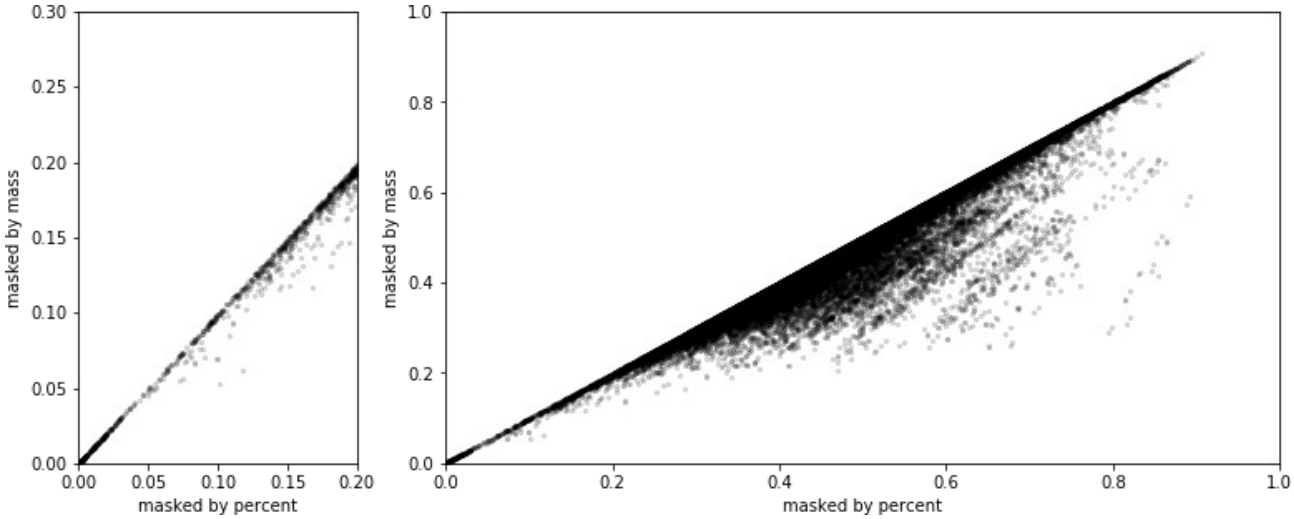

Figure 24: selecting the mask by 95% of mass (keeps 3389 elements per topic) and 5% of largest elements (keeps 1533 elements per topic) plotted against each other. The left plot shows the plot in the area that is relevant for epsilon in higher detail, the right shows the complete plot to the maximum distance of 1. The data points to the bottom right of the diagonal mean, that they are considered closer when masked by 95% of mass. Each data point is the distance of a single topic pair.

In the previous Figure can be seen, that the threshold in masking is rather flexible, because it is very close to a linear relationship for epsilon < 0.2. Masking by 11 % of the largest elements, which cuts away about the same amount of tokens, is 1.28 times as fast. The result of masking by mass is, that a certain amount of elements is removed. So it is just a matter of the right threshold.

Using both approaches creates the distance matrix in 6.54 seconds, which is 2.39 times as fast. An even faster approach for calculating the euclidean distance can be achieved by not normalizing the topics and squaring the distance, which removes the square root.

6.2 Multiprocessing with Workers

The calculation of this can easily be distributed by passing the complete topic term distribution to each worker, and telling every worker how many rows to calculate. Afterwards the results are concatenated. On a 4-Core CPU this is 3.46 times as fast and gets calculated in 4.52 seconds, when using the cosine distance and the masking by mass. When doing the same and masking by percent, it is 4.62 times as fast and finishes in 3.39 seconds.

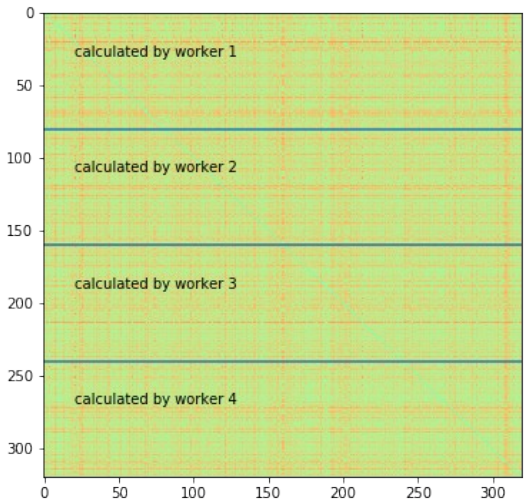

Figure 25: How distributing the calculation of the asymmetric distance matrix works

The package was modified in a way, that a parameter can be supplied that uses the faster masking method. The multi processed matrix calculation was also added to the module as a function.

```
model.generate_asymmetric_distance_matrix(workers=1, method="percent")
model.generate_asymmetric_distance_matrix(workers=None, method="mass")
```

workers=1 is the default value and will cause single core calculation. *workers=None* will detect the number of CPU cores. *Workers=4* can be used to create 4 workers. *method=”mass”* is the default value and uses the original masking method. After using the above method, the following has to be done to create stable topics from the distance matrix:

```
model.generate_topic_clusters()
model.generate_stable_topics()
model.create_Gensim_model()
```

Alternatively, new parameters were added to the constructor so that it can be controlled right from the beginning:

```
EnsembleLda(corpus=mm, masking_method="percent", distance_workers=3)
```

7. Discussion, Conclusion and Outlook

The EnsembleLda model showed that it produces more stable results on synthetic data. Results were less prone to be false positives or topic mixtures, which indicates that those topics that were detected are more likely to be correct. However, it is very hard to evaluate the quality of the models on real world data. The more models are used as children, the more stable the results of the ensemble become.

The parameters for clustering the models should be further optimized after training the ensemble, as this can easily be done quickly, and it might strongly affect the ensemble quality.

The package will be open sourced at some point in the future, but some development is still needed to clean experimental methods and classes up, that were only used during the development and that are not needed anymore.

8. References

[1]: A. FRIGYIK, Bela, Amol KAPILA and Maya R. GUPTA, 2010. Introduction to the Dirichlet Distribution and Related Processes. In: UWEE Technical Report [online]. December 2010 [Accessed on: 14/03/2018]. Washington: University of Washington. Available under: http://mayagupta.org/publications/FrigyikKapilaGuptaIntroToDirichlet.pdf

[2]: BELFORD, Mark, Brian MAC NAMEE and Derek GREENE, 2017. Stability of topic modeling via matrix factorization. In: Expert Systems With Applications [online]. 2018(91), pp. 159-169 [Accessed on: 11/07/2018]. ScienceDirect.com. Available under: DOI:10.1016/j.eswa.2017.08.047

[3]: EL-ARINI, Khalid, 2008. Dirichlet Processes: A gentle tutorial [Presentation]. In: SELECT Lab Meeting [online]. 14/10/2008 [Accessed on: 14/03/2018]. Available under: https://www.cs.cmu.edu/~kbe/dp_tutorial.pdf

[4]: HE, Ruining and Julian MCAULEY, 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In: WWW '16 Proceedings of the 25th International Conference on World Wide Web. Montréal, Québec, Canada, April 11 - 15. pp. 507-517. ISBN 978-1-4503-4143-1

[5]: HUGHES, Michael C. and others, 2016. Supervised topic models for clinical interpretability [online]. Massachusetts: Harvard Univsersity. [Accessed on: 10/07/1028]. Available under: https://www.michaelchughes.com/papers/HughesElibolMcCoyPerlisDoshi_MLHCWorkshopAtNIPS2016.pdf

[6]: KAVITA, Ganesan, ChengXiang ZHAI and jiawei HAN, 2010. Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions. In: Proceedings of the 23rd International Conference on Computational Linguistics [online]. Association for Computational Linguistics. 2018. pp. 340-348. [Accessed on: 22/07/2018]. Available from: http://kavita-ganesan.com/opinosis

[7]: KNISPELIS, Andrius, 2016. LDA Topic Models. In: youtube.com [online]. 08/07/2016 [Accessed on: 12/03/2018]. Available under: https://www.youtube.com/watch?v=3mHy4OSyRf0

[8]: LOOSLEY, J. Alex, Alexandre D. SALLES, Stephan SAHM and Juan BERNABE-MORENO, 2017. Extracting Reliable Topics Using Topic Model Ensembles. Munich: Data Reply GmbH.

[9]: M. BLEI, David, Andrew Y. NG and Michael I. JORDAN, 2003. Latent Dirichlet Allocation. In: Journal of Machine Laerning Research [online]. 2003(3), pp. 993-1022 [Accessed on: 16/06/2018]. Available under: http://www.cs.columbia.edu/~blei/publications.html

[10]: REHUREK, Radim and Sojka, PETR, 2010. Software framework for topic modelling with large corpora. In : The LREC 2010 Workshop on New Challenges for NLP Frameworks [online]. Msida: University of Malta. 2010. pp. 45-50 [Accessed on: 29/06/2018]. Available under: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.695.4595

[11]: STEVEN, Ford, 2013, Non-Negative Matrix Factorization. In: YoutTube [online]. Pittsburgh, Pennsylvania: Carnegie Mellon University, 11/10/2013 [Accessed on: 18/07/2018]. Available under: https://www.youtube.com/watch?v=UQGEB3Q5-fQ

9. Declaration

I hereby declare that this thesis is my own work, that I have not presented it elsewhere for examination purposes and that I have not used any sources or aids other than those stated. I have marked verbatim and indirect quotations as such.

Munich, 02/08/2018

Tobias Brigl