02462 Final project
Text classification by word embeddings

Rasmus Aagaard (s164419), Andreas Lau Hansen (s194235)

Abstract

We wish to investigate and evaluate the performance of the *fastText* [2] model compared to a baseline model produced with GloVe embeddings [3]. This tells us how much of a benefit a more complex model such as fastText has compared to a pre-trained GloVe embedding which is relatively simple. We score the prediction accuracy of both the models with two data sets, spam/not spam-texts and categorical news headlines, and see which performs better. We performed a number of experiments on the trained classifiers to test their robustness in non-ideal situations. fastText seemed to generally perform better than the pre-trained GloVe embeddings, which is to be expected given that the fastText models were trained on the specific data and would not perform well in general applications like GloVe.

1 Introduction

In order to do any meaningful work with text, it is necessary to turn the text into to some form and shape that a computer can work with. The usability and accuracy of a classifier is dependent on a strong representation of its input. Similarly, to make a sentiment analysis it is ideal to have good basis to transform the text on, since it will likely achieve a better result. Across all use cases, the goal is to gain as much meaning of a word as possible in as little time and space as possible.

In this study we compare the use of different representations for texts for later classification. We compare pre-trained GloVe embedding vectors and the fastText model from Facebook AI trained on data we're given as part of the project.

The primary difference between the two methods is that fastText is capable of using n-grams rather than unigrams for its word representation. But it is maybe more significant that the GloVe embedding vectors are trained on a much larger corpus than what we have available - in turn making it more generalised.

We are using two data sets for our experiments. `spam` and `news` (Pytorch `AG_NEWS`). The `spam`-corpus consists of 4457 SMS texts of which 661 are spam texts and 3846 are not spam. The `news`-corpus consists of 120,000 short news reports each with a label corresponding to one of four: *sports, world, business or science & technology* and is evenly divided.

1.1 Notation

Bold small letters e.g. $\mathbf{z}$ represent column vectors. Accordingly bold capital letters are matrices e.g. $\mathbf{A}$. $\mathbf{A}^\top$ defines the transpose of $\mathbf{A}$. $p(y_n|x_n)$ indicates the probability of y given x.

2 Methods

2.1 Pre-processing of texts

We did not perform any further pre-processing of the text other from what was already done. Both data sets were all in lower case. For the news data set, unknown words were replaced with a generic unknown token (<UNK>).

2.2 Baseline model

For the baseline model we're using a pre-trained collection of word embeddings, GloVe, which is of dimension $D = 50$. For the nth text we mean over all the word embeddings, $\mathbf{w}_i$ with i going from 1 to N_n, (Number of words in nth text):

$$\mathbf{z}_n^{\text{GloVe}} = \frac{1}{N_n} \sum_{i=1}^{N_n} \mathbf{w}_i$$

Using a simple Gaussian Naive Bayes (GNB) classifier we score the test accuracy of both data sets which can be seen in Table 1. The GNB classifier was chosen for its simplicity while also performing decently for the purpose of a baseline. We considered implementing it ourselves but it ultimately seemed redundant to spend too much time on the baseline causing us to use `sklearn`s implementation.

	Test accuracy
`spam`	$\approx 93\%$
`news`	$\approx 84\%$

Table 1: Test accuracies for baseline model obtained with a GNB classifier

The idea behind a Naive Bayes classifier is that you have some vector $\mathbf{x} = (x_1, x_2, \dots x_n)$ and wish to express the probability of $\mathbf{x}$ being in class y_i which is the conditional probability of y_i given $\mathbf{x}$

$$p(y_i|x_1, x_2, \dots x_n)$$

The above expression runs into problems, when n becomes large - it becomes computationally unattainable to work out the probability. To work around this we *naively* assume that each point x_i is independent thus creating a product of probabilities which is way more manageable. Since our data is continuous, we use the *Gaussian* Naive Bayes which simply assumes that the values for each class are normally distributed.

2.3 fastText

This model takes a text as input and outputs a probability distribution in accordance to the labels. The model utilises the mean embedding of all ngrams up to the selected hyperparameter `ngrams` as seen in Equation (1)

$$\mathbf{z}_n = \frac{1}{N_n} \sum_{i \in \mathcal{D}_n} \mathbf{b}_i \tag{1}$$

This latent representation of a given text can then be classified using a softmax layer in the model such that

$$p(y_n|\mathbf{A}, \mathbf{z}_n) = \text{softmax}(\mathbf{A}\mathbf{z}_n)$$

where $\mathbf{A} \in \mathbb{R}^{K \times D}$ are the weights obtained during training, K is the number of labels and D is the dimension of the hidden representation. D is how many dimensions the embedding vectors are represented as. This can also be interpreted as the a parameter for the complexity spectrum. Too low dimensionality will cause under-fitting and too high will cause over-fitting.

2.4 Cross validation

The *fastText*-method has the hyperparameters: `embed_dim`, `ngrams` and `num_epochs`.

We employ the strategy of cross validation to find the best suitable number of dimensions D by performing 5 splits on the training data thus holding $1/5$ as a test set to test the accuracy of the model trained on the remainding $4/5$th and note the mean accuracy for each number of dimensions. The hyperparamter `ngrams` is also varied from 1 to 3 such that it tries uni-, bi-, and trigrams. Lastly, the number of epochs during training are set to 5 during all folds. The cross-validations are plotted on Figure 1 and those yielding the optimal results are found in Table 2.

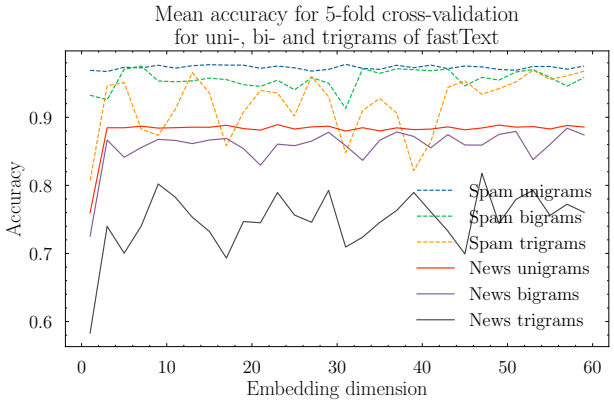

Figure 1: Mean accuracies for both data sets obtained during cross validation with varying dimension D and n-grams

	`ngrams`	`embed_dims`	Accuracy
`spam`	1	31	$\approx 97\%$
`news`	1	23	$\approx 91\%$

Table 2: Optimal hyperparameters and respective accuracy given by Figure 1.

2.5 Principal Component Analysis

We used principal component analysis using eigenvalue decomposition in order to maximise variance in

the least amount of dimensions as possible. This is beneficial for visualising the data since it's much easier to comprehend a two- or three-dimensional space rather than a larger one. The general idea is that we can form a matrix with each vector as a row

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_1} \\ \vdots \\ \mathbf{x_m} \end{bmatrix}$$

We can then form the covariance matrix

$$\mathbf{C_x} = \frac{1}{n}\mathbf{X}\mathbf{X}^\top = \mathbf{V}\mathbf{D}\mathbf{V}^\top$$

Then we need to find an orthonormal matrix $\mathbf{P} = \mathbf{V}^\top$ such that the matrix equation is satisfied

$$\mathbf{Y} = \mathbf{P} \cdot \mathbf{X} \tag{2}$$

By doing this the rows of $\mathbf{P}$ will be the ordered set of principal components $\mathbf{p_i}$. In practice you subtract the mean of the data $\mathbf{X}$ and calculate the eigenvectors of $\mathbf{C_x}$. For a more in depth guide see [4].

3 Results

From Figure 1 it is immediately noticeable that the accuracy in the cross validation is consistently higher for a lower number of n-grams. Our interpretation of this is that given the representation of the n'th document in fastText is defined as Equation (1) and with the assumption that an n-gram only carries valuable information to the model if that n-gram is present somewhat often, and the lengthier the n-grams the more unlikely they are to be present often, we'll get more and more n-grams which are not valuable for representing the n-gram, thus the mean embedding of those n-grams will ultimately end up not being valuable for representing that text.

The found hyperparameters for both data sets used for fastText as seen in Table 2 are used to train a classifier on the entire training set which we test on the test set and get the accuracies as shown in Table 3. These show an increase in performance in both cases.

	fastText accuracy	GloVe accuracy
`spam`	$\approx 97\%$	$\approx 93\%$
`news`	$\approx 91\%$	$\approx 84\%$

Table 3: fastText accuracy found when training the classifiers with hyperparameters as found in Table 2 on the respective test sets compared to GloVe

The following sections are split up for the `news` and `spam` data set and presents and evaluates experiments done on each.

3.1 News

3.1.1 Comparison with similar news reports

In order to give a better evaluation of the models, we have included some additional articles to make a qualitative assessment of the performance. The article are in some cases ambiguous to classify, but also included are articles that we definitely thought belonged to one category. We gathered a selection of 14 articles from the front page of BBC.com, on 08-05-2020. From the articles we extracted the introductory paragraph, to make them more similar to the articles in the news corpus.

Take for example the article: *"Facebook and Google have said they will let employees continue working from home for the rest of the year. The tech giants have announced plans to reopen their offices soon but are allowing more home working flexibility"*. We originally labelled it to the business category, which BBC also did. However, it could quite easily be Sci/Tec. The fastText model gave it the softmax scores in Table 4, this corresponds to our labelling which was Sci-Tec/Business. It is about 0.3 higher for Sci-Tec than Business which is a much smaller difference between the categories than what we see for some of the texts where we only gave it one label (see [1]).

Softmax scores

	Sports	World	Business	Sci/Tec
BBC art.	0.1757	0.0201	0.2365	0.5678
NN 1	0.2027	0.0223	0.5053	0.2698
NN 2	0.1067	0.0107	0.0753	0.8073
NN 3	0.1957	0.0185	0.6601	0.1256

Table 4: Softmax scores for articles with fastText embedding. NN are the nearest neighbours.

By finding the three closest points (articles) in the PCA plot with GloVe vectors, we can investigate their original labels, which turned out to be [3, 3, 1]. The 1st article classified as category 3 is: *"house passes second spyware bill the us house of representatives passed a second spyware bill on Thursday adding more teeth to a previously passed spyware bill that*

included fines for offender". The content of the article is very similar to the content of the BBC Facebook/Google article. This demonstrates that the embeddings represent the content of the articles in a very direct manner. When using the fastText, we more regularly see articles in the vicinity which are of the correct label. With the same BBC article fastText gave the labels [2, 3, 1]. While this is only one case, it seems to be better overall. It's a metric that is quite difficult to measure. The three most similar articles have very similar softmax scores to the BBC article see Table 4. The scores reflect that the articles are very similar, and it is easy to understand why the classifier is having trouble with them.

3.1.2 Visualisation using PCA

Here we have visualised each article as point in 2D space, coloured according to their class. For the news data set, we found that the second and third eigenvalues and their corresponding eigenvectors gave the optimal result at showing as much possible of each category of articles. The representation was at least more balanced across the categories using these eigenvectors.

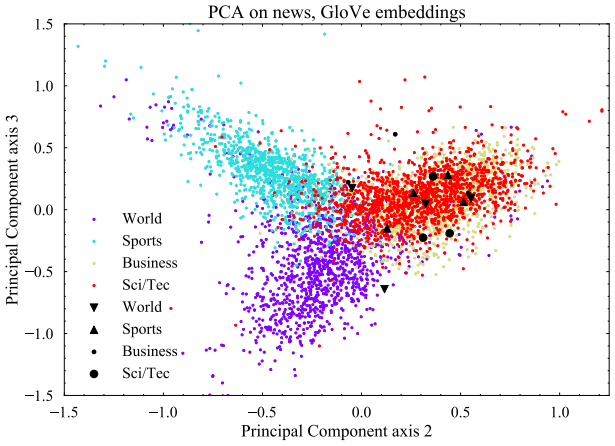

Figure 2: PCA plot for news data set with baseline embeddings. Shown with 5000 points in order to see the yellow points. Black symbols are BBC articles.

Figure 2 shows the PCA representation of the news data set with the 2nd and 3rd eigenvector. 10 principal components are needed to explain 80% variance. It is clear that the Business and Sci/Tec categories overlap to such a high degree that they are nearly identical. Contrasting this is Figure 3, which has a much better separation of all the categories, albeit the sports category has been muted a bit.

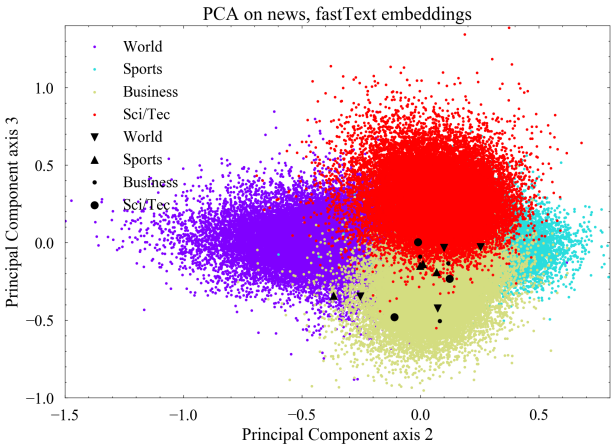

Figure 3: PCA plot with fastText embeddings. All points. Black symbols are BBC articles

Overall, the overlap between the categories seem to suggest that the articles themselves contain some overlapping features. Signifying a similarity in the content of the articles.

3.2 Spam

A similar PC analysis was done for the spam data set.

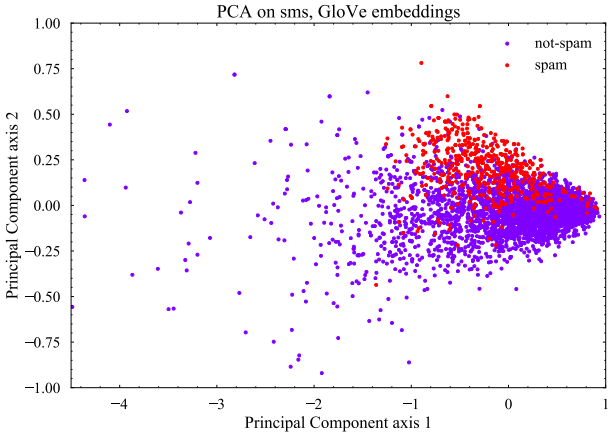

Figure 4: PCA plot for baseline embeddings with 2 most significant directions

Both Figure 4 and 5 reflect an inherently better separation, since there only are two classes. Accordingly,

around 88% variance is preserved with just two eigenvectors for Figure 4. Strangely, Figure 5 only displays a 40% variance with the first two eigenvectors.

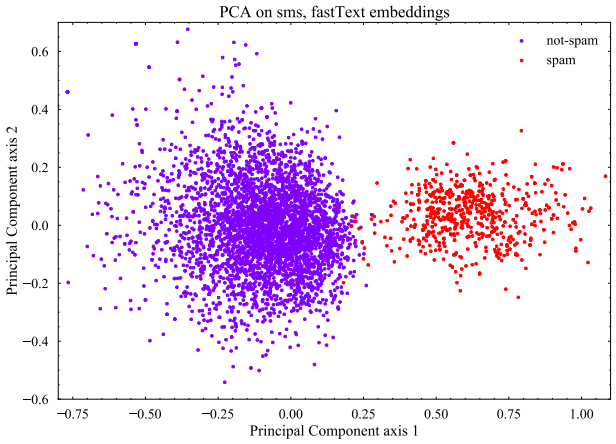

Figure 5: PCA plot with fastText embeddings with 2 most significant directions

Based on the visualisations of both the data sets, it is clear that the fastText embeddings do a much better job at representing the data. The axes of the PCA plots signify most general characteristic features of the text. Each axis is a kind of weighted linear combination of each of the components of the original word vectors that maximises the variance. The (x, y) coordinates of the points then represents the covariance between the original text and the principal components.

3.2.1 Most significant words for text classification

To get an understanding of what causes a text to be classified as either spam or not spam, we processed the dataset such that we could get all n-grams for the entirety of the corpus. These n-grams were then classified using fastText trained on the entire training set. From this, we gathered softmax scores for all n-grams which were either spam or not spam. The resulting sorted top 5 of uni-, bi- and bi-grams can be seen in Table 5. Do note that most of these simply had a score of "0" or "1" indicating not spam and spam meaning that the order might be somewhat arbitrary.

Unigram

Spam	Not spam
smart	fine:)when
ltd	much
16	still
www.txttowin.co.uk	oh.
hello	father

Bigram

Spam	Not spam
txt bak	me last
wallpaper text	that night
stop 4	not that
stop by	very soon
txt dis	probably going

Trigram

Spam	Not spam
incredible txts! reply	diddy is my
& double txt	know where my
unsubscribe text stop	anything i will
& 1000 txts	month hence my
txt xxx slo(4msgs)	my phones having

Table 5: Top 5 spam and not spam for uni-, bi- and trigrams respectively according to the softmax scores of our spam fastText classifier.

There are clear trends in the spam portion of the n-grams. In the bigram portion we see the string "txt" come up twice, while also appearing multiple times under trigrams. Curiously the unigram "Hello" is classified strongly as spam. Perhaps this is too formal of a greeting for the texts in the data set leading it to be considered spam.

From the "Not spam" columns, we are seeing more coherent parts of sentences as expected. One can easily imagine those n-grams being part of a human-structured text message not intended to sell or advertise.

3.2.2 Fooling the classifier

When considering the spam-labelled texts we wish to see if we can add an n-gram from the the entire data set such that the classification will switch from spam to not spam. The experiment does 100 simulations for each n and samples 100 random spam-labelled text and classifies it with and without a random n-gram. The resulting mean accuracies for each n is plotted as seen in Figure 6. Here it is seen that there is a clear trend in decreasing classification accuracy

when adding more random n-grams to an otherwise spam-labelled text.

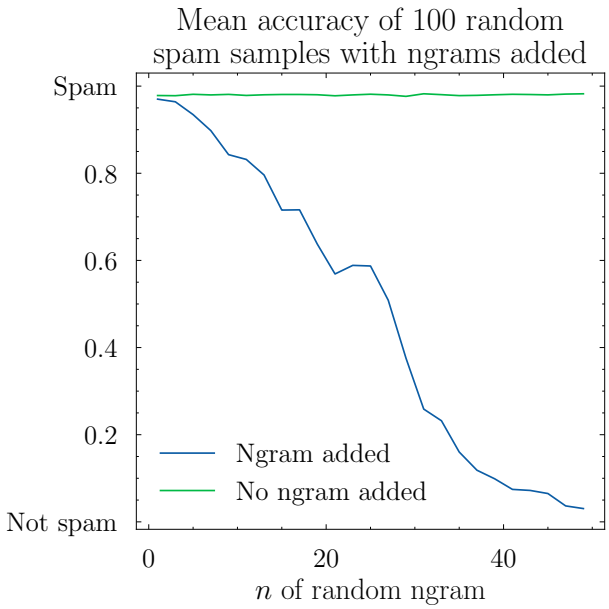

Figure 6: Mean fastText classification accuracy over 100 samples of random spam-labeled text with and without a randomly sampled n-grams.

3.2.3 Other applications

We investigated how well the classifier would perform with classifying emails after being trained on the sms dataset. We used roughly 5500 emails from the "enron" email dataset in the experiment.

	Test accuracy
`GloVe`	$\approx 66\%$
`fastText`	$\approx 68\%$

Table 6: Test accuracy for `GloVe` and `fastText` found when training the classifiers with hyperparameters as found in Table 2 on the test set

As seen in the classification accuracies in Table 6, the email data-set the classifier trained on the `spam` data set performs quite poorly. This is likely attributed to the style of language often used in text messages in contrast to that of emails which is often in a more formal setting.

4 Discussion

Using the fastText embeddings it seemed as though the similar news articles were much more specific with the content being more similar or just more specific. Though, there is no doubt that both types of embeddings elicit a high degree of similarity between the articles. When doing PCA on the fastText embeddings, we achieve a much better spread. This is likely due to fact that it is trained directly on the dataset, so it internally maximises the vectors - whereas the GloVe embeddings originally had to account for a much wider range of words. It could also be attributed to the fact that the fastText embeddings need fewer dimensions to achieve the same accuracy, so when reducing the dimensions less is lost. The end result is a much better separation when visualised in 2D

A thing to note is that the data is likely to be inherently bias at least in the case of the `news` data set where a text has been subjectively categorised and labeled to obtain the data set.

In conclusion, the benefit of using pre-trained embedding vectors is that we can save a lot of computation time, if we are willing to sacrifice a bit of accuracy. However, the fastText model is speedy after we found the optimal parameters; requiring only around 2 minutes of training time on an ordinary laptop. The small amount of training time is well worth it to gain extra performance in a classification use case. An improvement of roughly 7-8% is significant.

References

[1] URL: `https://github.com/Aagaarrd/02462-text_classification`.

[2] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *CoRR* abs/1607.01759 (2016). arXiv: `1607.01759`. URL: `http://arxiv.org/abs/1607.01759`.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: `http://www.aclweb.org/anthology/D14-1162`.

[4] Jonathon Shlens. "A Tutorial on Principal Component Analysis". In: *Educational* 51 (Apr. 2014).