UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and Machine Intelligence 2019-2020

Module Coursework Feedback

Module Title: Natural Language Processing

Module Code: MLMI13

Candidate Number: G709

Coursework Number: Practial Report

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			**Pass**			**Fail (C+ - marginal fail)**		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

Part One: Naive Bayes Baseline

Let $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ be our documents and $\mathcal{Y} = \{y_1, y_2, ..., y_n\}$ the target classes, for $n = 1 \rightarrow 2000$ and $y_i = \{0, 1\}\ \forall i$. The Naive Bayes classifier can be derived from Bayes' rule:

$$\begin{aligned} P(y_i = 1|d_i) &= \frac{P(d_i|y_i = 1)P(y_i = 1)}{P(d_i)} \\ &\propto P(d_i|y_i = 1)P(y_i = 1) \end{aligned}$$

where $P(y_i = 1)$ is our prior knowledge of the proportion of documents with class 1. Since we have 1000 positive and 1000 negative classes, I set $P(y_i = 1) = 0.5$. We can decompose the probability of a document given a class as:

$$P(d_i|y_i = 1) = \prod_{k=1}^{n_{d_i}} P(t_k^{(i)}|y_i = 1)$$

where $t_k^{(i)}$ are the features associated with d_i and n_{d_i} is the number of them in d_i. These are simply the words (unigrams) or 2-sequences of words (bigrams), occurring in the document. The 'naiveness' of the Naive Bayes classifier comes from the estimation:

$$P(t|y_i = 1) = \frac{\sum_{d_i \in \mathcal{T}} \sum_{k=1}^{n_{d_i}} \mathbb{1}(t = t_k^{(i)})}{\sum_{t \in V_{y_i}} \sum_{d_i \in \mathcal{T}} \sum_{k=1}^{n_{d_i}} \mathbb{1}(t = t_k^{(i)})} = \frac{T_{y_i t}}{\sum_{t \in V_{y_i}} T_{y_i t}}$$

where $\mathcal{T}$ represents our training data, $\mathbb{1}$ is an indicator function and V_{y_i} is the entire vocabulary (list of distinct features) in the training data of class y_i. It means that for a feature t, $P(t|y_i = 1)$ is the proportion of the number of times t appears in the positive training documents to the number of total appearances of all features in the positive training documents. The final decision of the classifier, y_j^*, for a document d_j is given by

$$y_j^*|d_j = \underset{y \in \{0,1\}}{\text{argmax}} \big[\sum_{k=1}^{n_{d_j}} \log(P(t_k^{(j)}|y_i = y)) \big]$$

The problem with the setup above is that at test time, we are likely to encounter features which we have not seen in $\mathcal{T}$. In my implementation, if a feature t was not seen before, I set $T_{yt} = 1$ to avoid the logarithm being undefined, which implies that $\log P(t|y = 1) = -\log\left(\sum_{t \in V_y} T_{yt}\right)$. It is very important that for every feature we take away the term $\log\left(\sum_{t \in V_y} T_{yt}\right)$, as not doing so will favour the class in which that feature did not appear (simply skipping that feature will result in higher probability of the class that the feature does not appear in!).

Frequency cutoff

An important implementation detail, is to introduce thresholds on the frequency of appearance of certain features. It means that if a certain feature t appears in $\mathcal{T}$ less than η times, then it is removed from the vocabulary. As suggested, I will be using two different cutoffs, η_1 and η_2, representing different thresholds for unigrams and bigrams. This makes sense as unique bigrams will generally appear less often than unigrams.

Laplace smoothing

In order to deal with the undefined log when a feature was not seen in $\mathcal{T}$, and do avoid having to arbitrarily set $T_{yt} = 1$ for these instances, we introduce Laplace smoothing. The probability of feature given a class becomes:

$$P(t|y_i = 1) = \frac{T_{y_i t} + \kappa}{\sum_{t \in V_{y_i}} (T_{y_i t} + \kappa)}$$

for κ a constant. Laplace smoothing has a feature of stabilizing the algorithm, as it evens out probabilities between common and rare words.

For the first part of the results, I will use $\mathcal{T}$ = `cv000-cv899` files, from the `POS/` and `NEG/` directories. From now on until the remainder of this work, I always process (tokenize) documents with the same pipeline involving removing all punctuation and lowering all words. I will report the accuracy obtained by testing on the remainder of the data (`cv900-cv999`).

The accuracies of 3 models (only unigram, only bigram and unigram and bigram jointly) for various settings of hyperparameters η_1, η_2 and κ can be seen in Tables 1,2,3 below.

	$\boldsymbol{\kappa = 0}$	$\boldsymbol{\kappa = 0.5}$	$\boldsymbol{\kappa = 1.5}$	$\boldsymbol{\kappa = 5}$
$\boldsymbol{\eta_1 = 0}$	0.810	0.825	0.830	0.835
$\boldsymbol{\eta_1 = 4}$	0.830	0.830	0.815	0.815
$\boldsymbol{\eta_1 = 8}$	0.830	0.805	0.835	0.790
$\boldsymbol{\eta_1 = 12}$	**0.850**	0.840	0.845	0.810

Table 1: Table showing accuracy on test set of a only **unigram model**, for various settings of hyperparameters.

	$\boldsymbol{\kappa = 0}$	$\boldsymbol{\kappa = 0.5}$	$\boldsymbol{\kappa = 1.5}$	$\boldsymbol{\kappa = 5}$
$\boldsymbol{\eta_2 = 0}$	0.800	**0.860**	0.850	0.815
$\boldsymbol{\eta_2 = 4}$	0.785	0.815	0.755	0.565
$\boldsymbol{\eta_2 = 8}$	0.755	0.780	0.735	0.575
$\boldsymbol{\eta_2 = 12}$	0.740	0.765	0.720	0.570

Table 2: Table showing accuracy on test set of a only **bigram model**, for various settings of hyperparameters.

	$\boldsymbol{\kappa = 0}$	$\boldsymbol{\kappa = 0.5}$	$\boldsymbol{\kappa = 1.5}$	$\boldsymbol{\kappa = 5}$
$\boldsymbol{(\eta_1, \eta_2) = (0, 0)}$	0.835	**0.865**	**0.865**	0.775
$\boldsymbol{(\eta_1, \eta_2) = (0, 4)}$	0.810	0.815	0.820	0.790
$\boldsymbol{(\eta_1, \eta_2) = (4, 8)}$	0.825	0.835	0.815	0.695
$\boldsymbol{(\eta_1, \eta_2) = (6, 12)}$	0.800	0.820	0.795	0.695

Table 3: Table showing accuracy on test set of a joint **unigram and bigram** model, for various settings of hyperparameters.

Although the results in Tables 1,2,3 are not very reliable as they were obtained by testing on a arbitrary subset of the data, we can observe some general trends. We tend to get high accuracy for models with $\eta_1 = \eta_2 = 0$ and $\kappa = 1.5$ and generally (with the exception of only

uni-gram model), accuracy drops as we increase the frequency cut-off, which makes sense as this way we are reducing the features in our dictionary.

Sign test

In order to see whether the above differences in accuracy are statistically significant, I perform the sign test. Let $\mathcal{M}(\eta_1, \eta_2, \kappa)$ represent the joint uni- and bi-gram Naive Bayes model with parameters η_1, η_2 and κ.
Let also $H_0(\kappa, \eta_1, \eta_2)$: $[\mathcal{M}_1(\eta_1, \eta_2, 0)$ performs identically to $\mathcal{M}_2(\eta_1, \eta_2, \kappa)]$. The sign test p-values for various models are given in Table 4 below.

	$P(H_0(0.5, \eta_1, \eta_2))$	$P(H_0(1.5, \eta_1, \eta_2))$	$P(H_0(5, \eta_1, \eta_2))$
$\boldsymbol{(\eta_1, \eta_2) = (0, 0)}$	0.103	0.103	0.079
$\boldsymbol{(\eta_1, \eta_2) = (0, 4)}$	0.112	0.111	0.108
$\boldsymbol{(\eta_1, \eta_2) = (4, 8)}$	0.111	0.111	**0.021**
$\boldsymbol{(\eta_1, \eta_2) = (6, 12)}$	0.108	0.112	**0.038**

Table 4: Table showing the probabilities of $H_0(0.5, \eta_1, \eta_2)$, $H_0(1.5, \eta_1, \eta_2)$ and $H_0(5, \eta_1, \eta_2)$ for various hyperparameters of the joint Naive Bayes model.

We can see that there is no statistical evidence that small smoothing increases the performance of the Naive Bayes model. There is however a significant drop in classifier performance when $\kappa = 5$.

Cross-validation

In order to avoid overfitting the test set, we can use k-fold cross-validation to estimate how the accuracy of our model would fair in an unseen dataset. I will use 10-fold Round Robin cross validation. In order to use statistical significance testing on cross-validated data, we can perform a t-test on the mean and variance of the accuracies from the 10 folds. Let $\boldsymbol{m}(\mathcal{M}_1) = (m_1(\mathcal{M}_1), m_2(\mathcal{M}_1), ..., m_{10}(\mathcal{M}_1))^T$ be the cross-validated accuracies of $\mathcal{M}_1$. In

the t-test, the null hypothesis is:

$$H_0 : \boldsymbol{m}(\mathcal{M}_1) \text{ and } \boldsymbol{m}(\mathcal{M}_2) \text{ are sampled from a Gaussian with the same mean}$$

and the t-statistic is:

$$t = \frac{\bar{\boldsymbol{m}}(\mathcal{M}_1) - \bar{\boldsymbol{m}}(\mathcal{M}_2)}{\sqrt{[\mathrm{Var}(\boldsymbol{m}(\mathcal{M}_1)) + \mathrm{Var}(\boldsymbol{m}(\mathcal{M}_2))]/10}}$$

where $\bar{\boldsymbol{m}}(\cdot)$ and $\mathrm{Var}(\boldsymbol{m}(\cdot))$ are the sample mean and variances. We can compute the p-value by looking at the cdf of the t-distribution at the value of t.

The cross-validated accuracies (average accuracy out of all folds) for the joint uni- and bigram model for various hyperparameters are given in Table 5 below.

	$\boldsymbol{\kappa = 0}$	$\boldsymbol{\kappa = 0.5}$	$\boldsymbol{\kappa = 1.5}$	$\boldsymbol{\kappa = 5}$
$\boldsymbol{(\eta_1, \eta_2) = (0, 0)}$	0.815±0.05	0.816±0.06	**0.821**±0.05	0.768±0.03
$\boldsymbol{(\eta_1, \eta_2) = (0, 4)}$	0.815±0.04	0.819±0.05	0.817±0.05	0.812±0.06
$\boldsymbol{(\eta_1, \eta_2) = (4, 8)}$	0.815±0.06	0.82±0.05	0.81±0.07	0.778±0.06
$\boldsymbol{(\eta_1, \eta_2) = (6, 12)}$	0.812±0.07	0.818±0.06	0.806±0.07	0.772±0.06

Table 5: Table showing cross-validated accuracy of a joint **unigram and bigram** model, for various settings of hyperparameters. Each number is reported with $\pm 2\sqrt{\mathrm{Var}}$.

In accordance to what we found with the sign test, we get that $\mathcal{M}(0, 0, 1.5)$ is not significantly better than $\mathcal{M}(0, 0, 0)$ (p-value=0.15). We do however get that $\mathcal{M}(0, 0, 1.5)$ is significantly better than $\mathcal{M}(6, 12, 1.5)$, suggesting that higher η_1 and η_2 give worse performance for $\kappa = 1.5$. It is also clear that increasing the frequency cutoffs results in higher prediction variance.

Although the smoothed version of Naive Bayes is not significantly better, it performs considerably better and hence, I would choose $\mathcal{M}(0, 0, 1.5)$ as the best model.

Part Two: SVM-based Sentiment Detection of Reviews

Blind test set

I have now delegated the first fold of the 10-fold Round Robin cross validation as the blind test set. Table 6 below shows the cross-validated accuracies using the rest of the data to train several NB models. I denote a uni-gram only model by $\mathcal{M}_{\text{uni}}(\eta_1, \kappa)$, a bi-gram only by $\mathcal{M}_{\text{bi}}(\eta_2, \kappa)$ and a joint uni and bi-gram model by $\mathcal{M}_{\text{joint}}(\eta_1, \eta_2, \kappa)$. The full grid search (not shown here) confirms that highest CV-accuracy is obtained by using $\mathcal{M}_{\text{joint}}(0, 0, 1.5)$.

Model	Accuracy
$\mathcal{M}_{\text{uni}}(0, 1.5)$	0.822 ± 0.04
$\mathcal{M}_{\text{uni}}(4, 1.5)$	0.801 ± 0.05
$\mathcal{M}_{\text{bi}}(0, 1.5)$	0.839 ± 0.03
$\mathcal{M}_{\text{bi}}(8, 1.5)$	0.632 ± 0.04
$\mathcal{M}_{\text{joint}}(0, 0, 1.5)$	$\mathbf{0.847 \pm 0.06}$
$\mathcal{M}_{\text{joint}}(4, 8, 1.5)$	0.741 ± 0.05

Table 6: Table showing cross-validated accuracies for several Naive Bayes models.

Bag of Words SVM

In order to train an SVM model, we need to have a vectorized representation of a document. One natural way of doing this is creating a vector $v^{d_i} = (v_1^{d_i}, ..., v_n^{d_i})^T \in \mathbb{R}^M$ per each document d_i, where $V = \{f_1, ..., f_M\}$ is our vocabulary and $v_j^{d_i}$ represents the count of feature f_j in d_i. I will use the same method of frequency cut-off as for Naive Bayes. For $\eta_1 = \eta_2 = 0$, the vectors will be very sparse, and `scipy` sparse-matrices need to be used. I first try different frequency cut-offs and test the cross-validated performance on a simple linear SVM.

	$\eta_2 = 0$	$\eta_2 = 4$	$\eta_2 = 8$	$\eta_2 = 12$
$\eta_1 = 0$	0.839 ± 0.03	0.852 ± 0.05	0.852 ± 0.05	0.850 ± 0.04
$\eta_1 = 4$	0.841 ± 0.03	**0.853 ± 0.05**	0.849 ± 0.04	0.849 ± 0.04
$\eta_1 = 8$	0.844 ± 0.03	0.851 ± 0.04	0.850 ± 0.04	0.847 ± 0.04

Table 7: Table showing cross-validated accuracies on the blind train set, using different frequency cutoffs, for the vectorized BOW model trained on a linear SVM.

In the table above we can see that SVM performs almost identically using different cutoffs η_1 and η_2. There is no significant difference in performance on any particular setting, but a considerable jump from $\eta_1 = \eta_2 = 0$ to $(\eta_1, \eta_2) = (4, 4)$, with p-value 0.15. This may be because for $(\eta_1, \eta_2) = (0, 0)$ the vectors are very sparse and long (around 500k entries), and relevant information gets dwarfed by the sea of 0's. I will thus use the BOW vectors with $(\eta_1, \eta_2) = (4, 4)$ to optimize SVM parameters.

Model	**Accuracy**
$\mathcal{SVM}(\text{linear}, -, -)$	**0.853 ± 0.05**
$\mathcal{SVM}(\text{poly}, 0.001, 2)$	0.701 ± 0.06
$\mathcal{SVM}(\text{poly}, 0.001, 5)$	0.622 ± 0.09
$\mathcal{SVM}(\text{poly}, \text{'scale'}, 2)$	0.739 ± 0.06
$\mathcal{SVM}(\text{poly}, \text{'scale'}, 5)$	0.652 ± 0.08
$\mathcal{SVM}(\text{rbf}, 0.001, -)$	0.737 ± 0.05
$\mathcal{SVM}(\text{rbf}, 0.002, -)$	0.627 ± 0.07
$\mathcal{SVM}(\text{rbf}, 0.005, -)$	0.524 ± 0.04
$\mathcal{SVM}(\text{rbf}, \text{'scale'}, -)$	0.767 ± 0.04

Table 8: Table showing the cross-validated accuracies on the blind train set, using different SVM models trained on vectors obtained from a BOW with $(\eta_1, \eta_2) = (4, 4)$.

Let $\mathcal{SVM}(\text{kernel}, \gamma, \text{d})$ represent a kernel SVM model, with γ parameter (only applicable for the Gaussian radial basis function and polynomial kernels) and a d-degree polynomial (only applicable for a polynomial kernel). A selection of $\mathcal{SVM}$ models and their cross-validated accuracy is given in Table 8 above. A $\gamma =$ 'scale' represents $\gamma = \frac{1}{\text{no\_features} \times \text{Var}(X)}$,

which in our case is ≈ 0.00016. The simple linear kernel achieves significantly better accuracy on the BOW vector representation, than other models (all p-values < 0.001).

Doc2vec SVM

Another way to vectorize the documents is by using a Doc2Vec model. There are 2 main architectures, the distributed memory (DM) and distributed bag of words (DBOW). In the DM architecture, every document and every word are mapped to unique one-hot id vectors. These vectors are then used to train a single-layer neural network in predicting a word given some context window around it, using the concatenated document and context word vectors as parameters. The DBOW architecture uses a similar method, but at each iteration samples a text window and tries to predict a randomly chosen context word in it, using only the document vector as an input. In both architectures, the trained weights of the neural network corresponding to the document vectors are used as the vector representations of a document.

Model	**Accuracy**
$\mathcal{DV}(0,0,1,100,1)$	0.838 ± 0.03
$\mathcal{DV}(0,0,10,50,2)$	0.852 ± 0.04
$\mathcal{DV}(0,0,5,100,2)$	0.866 ± 0.05
$\mathcal{DV}(0,1,1,50,2)$	0.877 ± 0.06
$\mathcal{DV}(0,1,10,50,5)$	$\mathbf{0.887 \pm 0.04}$
$\mathcal{DV}(1,0,1,100,1)$	0.819 ± 0.04
$\mathcal{DV}(1,0,10,50,2)$	0.825 ± 0.03
$\mathcal{DV}(1,0,5,100,2)$	0.871 ± 0.05
$\mathcal{DV}(1,1,1,50,2)$	0.861 ± 0.04
$\mathcal{DV}(1,1,10,50,5)$	0.871 ± 0.04

Table 9: Table showing the cross-validated accuracies on the blind train set, using different Doc2Vec models (all trained for 30 epochs) trained on a linear SVM.

To select the best hyperparameters for the Doc2Vec model, I keep the SVM model fixed

(linear kernel) and cross-validate on the blind train set. The parameters I tune are `dm`=$\{0, 1\}$ (0 is DBOW and 1 is DM), `hs`=$\{0, 1\}$ (1 indicates using hierarchical softmax, a faster version which traverses only the most probable nodes in the tree), `min_count`=$\{1, 5, 10\}$ (frequency cutoff for the words), `vector_size`=$\{50, 100\}$ (the size of the output document vectors) and `window` = $\{1, 2, 5\}$ (the size of the contex window).

Let $\mathcal{DV}(\text{dm}, \text{hs}, \text{min\_count}, \text{vector\_size}, \text{window})$ represent a Doc2Vec model with specified parameters. The cross-validated accuracies for a selection of trained $\mathcal{DV}$ models are given in Table 8 above. We can see that generally the DBOW architecture performs better. This might be because the DBOW architecture does not actually use context words as prediction aid, allowing it to grasp a general direction of a review, without relying on seeing words appearing in certain contexts. The optimized Doc2Vec embeddings perform significantly better than the best BOW embeddings (on the same, linear SVM, p-value=0.01).

In Table 10 below, we can see the results of optimizing the SVM parameters for the Doc2Vec embeddings.

Model	**Accuracy**
$\mathcal{SVM}(\text{linear}, -, -)$	0.887 ± 0.04
$\mathcal{SVM}(\text{poly}, 0.001, 2)$	0.832 ± 0.06
$\mathcal{SVM}(\text{poly}, 0.001, 5)$	0.558 ± 0.09
$\mathcal{SVM}(\text{poly}, \text{'scale'}, 2)$	0.882 ± 0.06
$\mathcal{SVM}(\text{poly}, \text{'scale'}, 5)$	0.871 ± 0.08
$\mathcal{SVM}(\text{rbf}, 0.001, -)$	0.848 ± 0.05
$\mathcal{SVM}(\text{rbf}, 0.1, -)$	0.892 ± 0.07
$\mathcal{SVM}(\text{rbf}, 0.2, -)$	0.873 ± 0.04
$\mathcal{SVM}(\text{rbf}, \text{'scale'}, -)$	**0.893** ± 0.04

Table 10: Table showing the cross-validated accuracies on the blind train set, using different SVM models trained on vectors obtained from a Doc2Vec model $\mathcal{DV}(0, 1, 10, 50, 5)$.

Embedding space analysis

But what do the document embeddings actually mean? We can look at the TSNE components of our movie reviews in Doc2Vec vector form and see whether some words or reviews tend to cluster together.

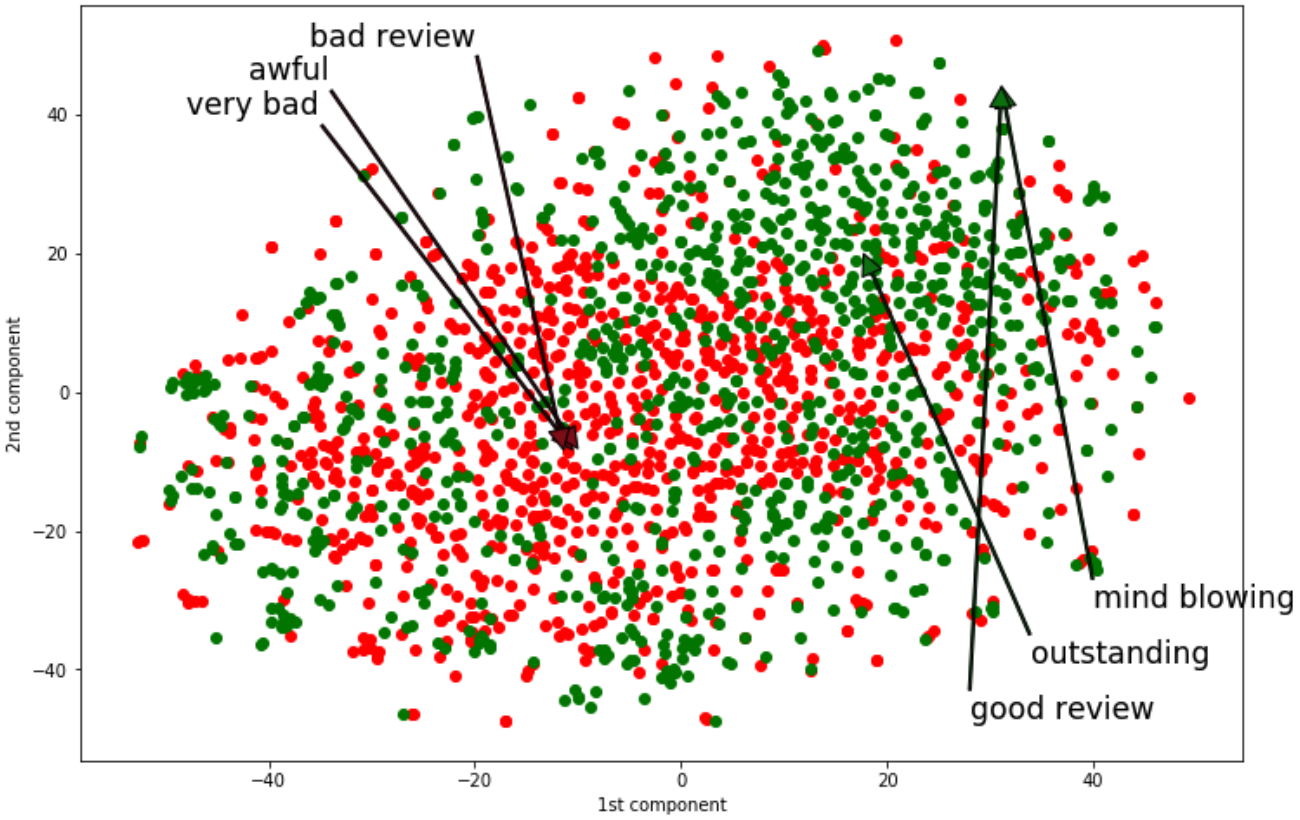

Figure 1: A plot of the first 2 components of TSNE, calculated on Doc2Vec vectors (using $\mathcal{DV}(0, 1, 10, 50, 5)$). The red dots represent negative reviews and green the positive reviews. The arrows point to positions on the TSNE component space of various words and made up reviews.

On Figure 1 above we can see a plot of 2 TSNE components calculated from the entire dataset, transformed to vectors by our best Doc2Vec model ($\mathcal{DV}(0, 1, 10, 50, 5)$). On the plot, we can also see the vector form of 4 features: 'mind blowing', 'outstanding', 'awful' and 'very bad'.

Additionally, I plot (on Figure 1) Doc2Vec vector components of a made up good review:

"Great! Wow! Amazing! The best movie I've seen, this is probably the best movie ever, fantastic acting, beautiful lighting"

and a made up bad review:

"Absolutely terrible movie I hated every moment of it! Stupid and dull and it sucks. So unfunny and boring, the worst, I barely survived, never watching it again"

Unsurprisingly the positive reviews (green) are not easily separable from the negative ones (red) in the space of the first 2 components, as the Doc2Vec embeddings are high-dimensional. We can see a slight trend of negative reviews being pushed to the left side of the graph - this is confirmed by the made up bad review and bad words being a lot more to the left of the made up good review and good words. This proves that Doc2Vec is actually useful in separating the concepts of 'negative' and 'positive' sentiments in a document.

Results

I will now present final results of all the models trained on the blind train set and tested on the blind test set. The final models I will compare are:

$\mathcal{M}_1$: A joint unigram and bigram Naive Bayes model, $(\eta_1, \eta_2) = (0, 0), \kappa = 1.5$

$\mathcal{M}_2$: A BOW embedding with $(\eta_1, \eta_2) = (4, 4)$, evaluated on a linear SVM

$\mathcal{M}_3$: A BOW embedding with $(\eta_1, \eta_2) = (4, 4)$, evaluated on a rbf SVM, gamma='scale'

$\mathcal{M}_4$: A DBOW Doc2Vec embedding with (hs,min_count,vector_size,window) $= (1, 10, 50, 5)$ evaluated on a linear SVM

$\mathcal{M}_5$: A DBOW Doc2Vec embedding with (hs,min_count,vector_size,window) $= (1, 10, 50, 5)$ evaluated on a rbf SVM, gamma = 'scale'

Additionally to comparing the prediction significance with the sign test, I will also compare it with the Monte Carlo permutation test, with $R = 5000$. Table 11 below contains the final accuracies of the above models:

Model	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
Accuracy	0.785	0.810	0.755	0.820	**0.825**

Table 11: Table showing blind test set accuracies for final set of optimized models defined above.

Table 11 below shows the results of running the sign and permutation tests on the predictions from the above models.

Model	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
$\mathcal{M}_1$	-	0.11/0.55	0.10/0.45	0.07/0.08	0.06/**0.05**
$\mathcal{M}_2$	-	-	0.08/**0.05**	0.10/0.38	0.11/0.66
$\mathcal{M}_3$	-	-	-	**0.05**/**0.02**	0.06/**0.04**
$\mathcal{M}_4$	-	-	-	-	0.11/0.37
$\mathcal{M}_5$	-	-	-	-	-

Table 12: Table showing the p-values for testing whether the model in the row performs better than the model in the column.

From Table 11 we can see that the $\mathcal{M}_5$ performs significantly better than $\mathcal{M}_1$ and $\mathcal{M}_3$ and that $\mathcal{M}_3$ is significantly worse than all the other models. Thus, the Doc2Vec embedding space model does better than the Naive Bayes one, but we cannot with significant confidence confirm that the BOW embedding is worse than the Doc2Vec one.

Error analysis

Looking at reviews that all 5 models misclassified (7 such reviews in the blind test set), reveals that their authors tried to express very complicated and often sarcastic ideas, thus fooling the models by containing a mixture of words typical for both positive and negative classes. One such negative review contained:

"...I was truly disappointed in Fincher who's previous two films Se7en and The Game put him in a very slim category of truly brilliant young filmmakers of the next generation. This is not to imply by any means that he is no longer a genius or capable of bringing more masterpieces to the table, but with fight club he has allowed his unique talent for extraordinary vision to be distorted..."

We can see that there are many positive words like 'extraordinary', 'brilliant' and 'genius' but the review actually turns out negative. The problem with our methods is that they cannot understand and classify sarcasm or sudden negation (when someone speaks highly of something only to refute it with a single sentence).
Another class of errors that we are getting is from the reviews being too descriptive.

"...one of the most respected names in American independent filmmaking is John Sayles. He has built a strong reputation with films like ... and one of his best was his last film Lone Star after Sayles gave us this complex and unconventional look at ethnic tensions... Expectation ran high for his next film, unfortunately his 'Men with Guns' is not the film anyone was hoping for..."

This review is negative, but before the author gets to the gist of it, he spends some time praising the director for his previous movies. This is something we could potentially solve by identifying which movie a particular bit of text is referring to (in this case we are interested in 'Men with Guns'). We could maybe use an LDA model to classify which movie is which word referring to, and then only consider the BOW referring to the movie the review is actually about.