Final Practical Report

MLMI13 - Natural Language Processing

December 7, 2021

Candidate no.: J902C

Word count: 1998

UNIVERSITY OF CAMBRIDGE
Department of Engineering

Trumpington Street
Cambridge CB2 1PZ
Tlf.: +44 01223 332600
http://www.eng.cam.ac.uk/

Table of contents

1. Part I: Baseline and Essentials

The task in Part One is to implement and evaluate methods to automatically classify movie reviews according to their sentiment.

1.1. Question 0: Symbolic approach

The first studied method is quite simple, but a sentiment lexicon is needed. A review r can be classified by counting how many words match either a positive (1) or a negative (-1) word entry in the sentiment lexicon:

$$S(r) = 1 \Leftrightarrow \sum_{w \in r} sgn(SLex[w]) >= t \ \ , \ \ t \in \mathbb{R} := \text{threshold}.$$

The information about the magnitude of sentiment can be used to try improve this method:

$$S(r) = 1 \Leftrightarrow \sum_{w \in r} weight[w] * sgn(SLex[w]) >= t \ \ , \ \ t \in \mathbb{R} := \text{threshold}.$$

Both symbolic methods are tested using the sentiment lexicon provided, considering that a *strong* positive word has twice the weight of a *weak* one (analogously in the negative case). The **resulting accuracy is 67%** using **just word sentiment**, and **68%** if the **magnitude of the sentiment** is used. A *significance test* is executed and it shows that magnitude lexicon results are **not** significant with respect to token-only. Results are shown in figure 1.1.

```
# retrieve corpus
corpus=MovieReviewCorpus(stemming=False,pos=False)

# use sign test for all significance testing
signTest=SignTest()

print("--- classifying reviews using sentiment lexicon  ---")

# read in lexicon
lexicon=SentimentLexicon()

# on average there are more positive than negative words per review
# (~7.13 more positive than negative per review)
# to take this bias into account will use threshold (roughly the bias itself)
# to make it harder to classify as positive
threshold=8

# question 0.1
lexicon.classify(corpus.reviews,threshold,magnitude=False)
token_preds=lexicon.predictions
print(f"token-only results: {lexicon.getAccuracy():.2f}")

lexicon.classify(corpus.reviews,threshold,magnitude=True)
magnitude_preds=lexicon.predictions
print(f"magnitude results: {lexicon.getAccuracy():.2f}")

# question 0.2
p_value=signTest.getSignificance(token_preds,magnitude_preds)
significance = "significant" if p_value < 0.05 else "not significant"
print(f"magnitude lexicon results are {significance} with respect to token-only")
```

```
--- classifying reviews using sentiment lexicon  ---
token-only results: 0.67
magnitude results: 0.68
magnitude lexicon results are not significant with respect to token-only
```

Figure 1.1: Code to execute Q0 - symbolic approach results

1.2. Question 1: Base Naive-Bayes

The first machine learning method implemented is Naive-Bayes, that operates on a simple Bag-of-Words (BoW) representation of the text data. The Naive-Bayes classifier works according to the equation 1.1.

$$\hat{c} = \underset{c \in C}{argmax} P(c|\mathbf{f}) = \underset{c \in C}{argmax} P(c) \prod_{i=1}^{n} P(f_i|c) = \underset{c \in C}{argmax} \{logP(c) + \sum_{i=1}^{n} logP(f_i|c)\}, \tag{1.1}$$

where $c \in C = \{POS, NEG\}$, $\hat{c}$ is the best possible class and **f** is the feature vector.

Naive-Bayes is a probabilistic classifier based on applying Bayes' rule with strong **independence assumptions between the features**.

After training our Naive-Bayes on files cv000–cv899 and testing on the remaining files cv900–cv999, the **resulting accuracy is 46%** (figure 1.2), i.e., worse than a random classifier. The **base version of Naive-Bayes is not effective**.

```
# question 1.0
print("--- classifying reviews using Naive Bayes on held-out test set ---")
NB=NaiveBayesText(smoothing=False,bigrams=False,trigrams=False,discard_closed_class=False)
NB.train(corpus.train)
NB.test(corpus.test)
# store predictions from classifier
non_smoothed_preds=NB.predictions
print(f"Accuracy without smoothing: {NB.getAccuracy():.2f}")
```

```
--- classifying reviews using Naive Bayes on held-out test set ---
Accuracy without smoothing: 0.46
```

Figure 1.2: Code to execute Q1 - base Naive-Bayes results

1.3. Question 2: Laplace smoothing in Naive-Bayes

The presence of words in the test dataset that haven't been seen during training can cause **probabilities** in the Naive-Bayes classifier to be **0**, thus making that particular test instance **undecidable**. In order to fix this and improve the base version of Naive-Bayes classifier, **Laplace smoothing** technique is incorporated, that consists of adding a small constant κ (in our implementation $\kappa = 1$) when counting word/token frequencies, as shown in equation 1.2.

$$P(w_i|c) = \frac{frec(w_i) + \kappa}{N_{words} + N_{words}\kappa} \tag{1.2}$$

After implementing **Laplace smoothing** and running the improved Naive-Bayes classifier, the **resulting accuracy increases to 82%**, showing that the results using smoothing **are significant** with respect to no smoothing. The reported performance can be seen in figure 1.3.

```
# question 2.0
# use smoothing
NB=NaiveBayesText(smoothing=True,bigrams=False,trigrams=False,discard_closed_class=False)
NB.train(corpus.train)
NB.test(corpus.test)
smoothed_preds=NB.predictions
# saving this for use later
num_non_stemmed_features=len(NB.vocabulary)
print(f"Accuracy using smoothing: {NB.getAccuracy():.2f}")


# question 2.1
# see if smoothing significantly improves results
p_value=signTest.getSignificance(non_smoothed_preds,smoothed_preds)
significance = "significant" if p_value < 0.05 else "not significant"
print(f"results using smoothing are {significance} with respect to no smoothing")
```

```
Accuracy using smoothing: 0.82
results using smoothing are significant with respect to no smoothing
```

Figure 1.3: Code to execute Q2 - Laplace smoothing results

1.4. Question 3: Cross-validation in Naive-Bayes

A common practice in Machine Learning is to use Cross-Validation as an evaluation technique. This technique **reduces the risk of producing type III errors** ("testing hypotheses suggested by the data" errors) and to obtain a **better estimation of the generalization error**. In N-fold cross-validation, data is divided into several chunks/-folds, training in all chunks but one, that it's used for testing and evaluating. Then, the experiment is repeated N times.

Cross-validation is implemented and Question 2 (section 1.3) is repeated, obtaining an **accuracy of 81.0% +- 2.07%**, visible in figure 1.4. The fact that the **standard deviation is so low** (about 2%) means that the classification algorithm **performs equally well in all folds**, which is a very good sign.

```
# question 3.0
print("--- classifying reviews using 10-fold cross-evaluation ---")
# using previous instantiated object
NB.crossValidate(corpus)
# using cross-eval for smoothed predictions from now on
smoothed_preds=NB.predictions
print(f"Accuracy: {NB.getAccuracy():.3f}")
print(f"Std. Dev: {NB.getStdDeviation()}")
```

```
--- classifying reviews using 10-fold cross-evaluation ---
Accuracy: 0.810
Std. Dev: 0.020736441353327688
```

Figure 1.4: Code to execute Q3 - Naive-Bayes cross-validation results

1.5. Question 4: Stemming

When using Word2Vec, we would like that every word is linked with its unique vector representation. However, what is a word? The answer is not easy due to capital letters, plurals, accents, etc. That's why NLTK Stemmer [1] is used. This software **removes morphological affixes** from words, **leaving only the word stem**.

```python
# question 4.0
print("--- stemming corpus ---")
# retrieve corpus with tokenized text and stemming (using porter)
stemmed_corpus=MovieReviewCorpus(stemming=True,pos=False)
print("--- cross-validating NB using stemming ---")
NB.crossValidate(stemmed_corpus)
stemmed_preds=NB.predictions
print(f"Accuracy: {NB.getAccuracy():.3f}")
print(f"Std. Dev: {NB.getStdDeviation():.3f}")

# TODO Q4.1
# see if stemming significantly improves results on smoothed NB
p_value=signTest.getSignificance(stemmed_preds,smoothed_preds)
significance = "significant" if p_value < 0.05 else "not significant"
print(f"results using stemming are {significance} with respect to non-stemmed corpus")

# TODO Q4.2
print("--- determining the number of features before/after stemming ---")
print("Number of features before stemming:", num_non_stemmed_features)
NB.train(stemmed_corpus.train)
print("Number of features after stemming:", len(NB.vocabulary))
```

```
--- stemming corpus ---
--- cross-validating NB using stemming ---
Accuracy: 0.813
Std. Dev: 0.026
results using stemming are not significant with respect to non-stemmed corpus
--- determining the number of features before/after stemming ---
Number of features before stemming: 52555
Number of features after stemming: 32561
```

Figure 1.5: Code to execute Q4 - stemming results

After using stemming in the corpus and training it with Naive-Bayes classifier, the resulting accuracy is **81.3%** $\pm$ **2.6%**. Theese results are **not significant** with respect to non-stemmed corpus trained on smoothed Naive-Bayes. This is shown in figure 1.5, where the number of features before and after stemming is also printed:
- No. words **before stemming: 52555**.
- No. words **after stemming: 32561**.
That's the expected result because stemming reduces the number of unique tokens in the corpus.

1.6. Question 5: Bigrams and Trigrams

N-grams Language Models are commonly used in the computational linguistics field. An N-gram is a contiguous sequence of N items from a given sample of text or speech. The items can be phonemes, syllables, letters, words, etc. according to the problem.

In this practical, **bigrams** and **trigrams** (N-grams of length 2 and 3 respectively) are going to be tested, using words as items, so **tokens will be pairs or triples of words**. Bigrams and trigrams can be easily implemented using `nltk` Python library, the function `ngrams(review,2)` will return a list of bigrams given a review, and analogously `ngrams(review,3)` will preprocess a review with trigrams.

Naive-Bayes can be executed using bigrams and trigrams as features and the results (using Cross-evaluation as evaluation technique) are shown in figure 1.6. Employing **bigrams** as features **improves slightly** the performance of Naive-Bayes, obtaining

an accuracy about **83%** $\pm$ **2%**, which is **not** a **significant** improvement over using just monograms. In addiction, **trigram features** perform **equally** well as monograms with an accuracy of **82%** $\pm$ **2%**. The reason why bigrams are improving the results a bit is that they **add useful context information** for each word. However, trigrams seem not to work as good as bigrams, probably because they **increase considerably the number of features**, making it impossible to improve the performance with the amount of data available. Bigrams and trigrams are not making a significant difference since they are **good at capturing relantionships** between previous, current and/or next token, but they are **ineffective at infering a general sentiment** in a whole document.

Focusing in the number of features, bigrams increase this number to **about half a million**, and trigrams **above a million**. Since there are 52555 monograms features, using **bigrams increases** the total amount of features **x10**, and trigrams **x20**. This might **raise** feature **sparcity** and problem **dimensionality**, and the Naive-Bayes classifier is not good at handling that.

```
# question Q5.0 and TODO Q5.1
# cross-validate model using smoothing and bigrams
print("--- cross-validating naive bayes using smoothing and bigrams ---")
NB=NaiveBayesText(smoothing=True,bigrams=True,trigrams=False,discard_closed_class=False)
NB.crossValidate(corpus)
smoothed_and_bigram_preds=NB.predictions
print(f"Accuracy: {NB.getAccuracy():.3f}")
print(f"Std. Dev: {NB.getStdDeviation():.3f}")
# counting features using bigrams
NB.train(corpus.train)
num_bigram_features = len(NB.vocabulary)
# see if bigrams significantly improves results on smoothed NB only
p_value=signTest.getSignificance(smoothed_preds,smoothed_and_bigram_preds)
signifance = "significant" if p_value < 0.05 else "not significant"
print(f"results using smoothing and bigrams are {signifance} with respect to smoothing only")
print("--- cross-validating naive bayes using smoothing and trigrams ---")
NB=NaiveBayesText(smoothing=True,bigrams=False,trigrams=True,discard_closed_class=False)
NB.crossValidate(corpus)
smoothed_and_trigram_preds=NB.predictions
print(f"Accuracy: {NB.getAccuracy():.3f}")
print(f"Std. Dev: {NB.getStdDeviation():.3f}")
# counting features using trigrams
NB.train(corpus.train)
num_trigram_features = len(NB.vocabulary)
# see if bigrams significantly improves results on smoothed NB only
p_value=signTest.getSignificance(smoothed_preds,smoothed_and_trigram_preds)
signifance = "significant" if p_value < 0.05 else "not significant"
print(f"results using smoothing and trigrams are {signifance} with respect to smoothing only")
print("--- determining the number of features before/after using bigrams ---")
print("Number of features before using bigrams (Q3):", num_non_stemmed_features)
print("Number of features after using bigrams:", num_bigram_features)
print("Number of features after using trigrams:", num_trigram_features)
```

```
--- cross-validating naive bayes using smoothing and bigrams ---
Accuracy: 0.828
Std. Dev: 0.025
results using smoothing and bigrams are not significant with respect to smoothing only
--- cross-validating naive bayes using smoothing and trigrams ---
Accuracy: 0.819
Std. Dev: 0.024
results using smoothing and trigrams are not significant with respect to smoothing only
--- determining the number of features before/after using bigrams ---
Number of features before using bigrams (Q3): 52555
Number of features after using bigrams: 500086
Number of features after using trigrams: 1015074
```

Figure 1.6: Code to execute Q5 - bigrams and trigrams results

1.7. Question 6: Using Support Vector Machines

The most important characteristic of Naive-Bayes algorithm is its assumption about feature independence. In a document, tokens (words, phonemes, ...) are usually **correleted** in order to transmit the desired message. That's why Naive-Bayes might be an inefficient ML method for this problem, and other methods are going to be tested.

Support Vector Machines (SVMs) is another popular supervised learning algorithm that, despite having other cons such as expensive execution or hard debugging, it **doesn't assume feature independence**.

The **Scikit-Learn implementation** of **SVM** [2] is going to be used for this assignment. Since SVM works with **numerical vectors**, reviews (set of words or tokens) are required to be **preprocessed**. The first approach was to convert all reviews into vectors of the **vocabulary size**, and each entry would be the number of **occurrences** of that word in the review. However, this approach might not be the best, since we end up with **sparse vectors** and it doesn't take into account the **review length**, so occurrences should be **normalized by review length**. To sum up, a review is encoded as a **frequency word vector** of the vocabulary's length, as explained in this Scikit-learn tutorial [3].

```
# TODO Q6 and 6.1

# retrieve corpus
corpus=MovieReviewCorpus(stemming=False,pos=False)

print("--- classifying reviews using SVM 10-fold cross-eval ---")
SVM=SVMText(bigrams=False,trigrams=False,discard_closed_class=False)
SVM.crossValidate(corpus)
svm_preds=SVM.predictions
print(f"Accuracy: {SVM.getAccuracy():.4f}")
print(f"Std. Dev: {SVM.getStdDeviation():.3f}")

# see if stemming significantly improves results on smoothed NB
p_value=signTest.getSignificance(svm_preds, smoothed_preds)
significance = "significant" if p_value < 0.05 else "not significant"
print(f"results using SVMs are {significance} with respect to smoothed Naive-Bayes")
```

```
--- classifying reviews using SVM 10-fold cross-eval ---
Accuracy: 0.8380
Std. Dev: 0.020
results using SVMs are not significant with respect to smoothed Naive-Bayes
```

Figure 1.7: Code to execute Q6 - Support Vector Machine results

The results for this section can be found in figure 1.7. The resulting **accuracy has slightly increased to about 84% $\pm$ 2%**, although this results are not significant with respect to smoothed Naive-Bayes of Question 3 (section 1.4).

1.8. Question 7: Part-of-speech features

Several words have different meaning depending on the context. For example, ”will” can be a verb or a noun. That’s why it is common to use Part-of-Speech (POS) tags when tackling a linguistic problem. Theese tags give syntax information about the word (or token) and can make the difference when it comes down to analyze sentiments.

In this section reviews are going to be classified **using as tokens/features tuples of word + POS tag**.

Additionally, closed-class words (prepositions, pronouns, determiners) tend to be **irrelevant**. After testing SVMs with word+POS features, **closed-class words are discarded and a SVM is trained** again in this trimmed corpus keeping just nouns, verbs, adjectives and adverbs.

```
# TODO Q7

print("--- adding in POS information to corpus ---")
# retrieve corpus with POS information
corpus_pos = MovieReviewCorpus(stemming=False,pos=True)

print("--- training svm on word+pos features ----")
SVM=SVMText(bigrams=False,trigrams=False,discard_closed_class=False)
SVM.crossValidate(corpus_pos)
svm_wordpos_preds = SVM.predictions
print(f"Accuracy: {SVM.getAccuracy():.4f}")
print(f"Std. Dev: {SVM.getStdDeviation():.3f}")

# see if stemming significantly improves results on smoothed NB
p_value=signTest.getSignificance(svm_wordpos_preds, smoothed_preds)
significance = "significant" if p_value < 0.05 else "not significant"
print(f"results using Word+POS features with SVMs are {significance} with respect to smoothed Naive-Bayes")

print("--- training svm discarding closed-class words ---")
SVM_dcc=SVMText(bigrams=False,trigrams=False,discard_closed_class=True)
SVM_dcc.crossValidate(corpus_pos)
svm_dcc_preds = SVM_dcc.predictions
print(f"Accuracy: {SVM_dcc.getAccuracy():.4f}")
print(f"Std. Dev: {SVM_dcc.getStdDeviation():.3f}")

# see if stemming significantly improves results on smoothed NB
p_value=signTest.getSignificance(svm_dcc_preds, smoothed_preds)
significance = "significant" if p_value < 0.05 else "not significant"
print(f"results discarding all closed-class words are {significance} with respect to smoothed Naive-Bayes")
```

```
--- adding in POS information to corpus ---
--- training svm on word+pos features ----
Accuracy: 0.8345
Std. Dev: 0.026
results using Word+POS features with SVMs are not significant with respect to smoothed Naive-Bayes
--- training svm discarding closed-class words ---
Accuracy: 0.8360
Std. Dev: 0.023
results discarding all closed-class words are not significant with respect to smoothed Naive-Bayes
```

Figure 1.8: Code to execute Q7 - POS features results

In figure 1.8 we can see the **accuracy for the SVM trained on word+POS features: 83.5% $\pm$ 2.5%**. In addition, after discarding **closed-class words**, the resulting **accuracy** is about **84% $\pm$ 2%**. In both cases the **results are not significant** with respect to **smoothed Naive-Bayes classifier**.

1.9. Additional analysis: optimized Laplace smoothing for Naive-Bayes, lowercase and punctuation

In section 1.3, the smoothing constant κ of equation 1.2 was fixed to 1. This constant can be optimized in order to get better results, as shown in figure 1.9. Different values have been tested for Naive-Bayes classifier using smoothing, smoothing with bigrams and smoothing with trigrams. The best values for the Laplace constant are $\kappa = 2$, $\kappa = 0.6$ and $\kappa = 0.2$ respectively.

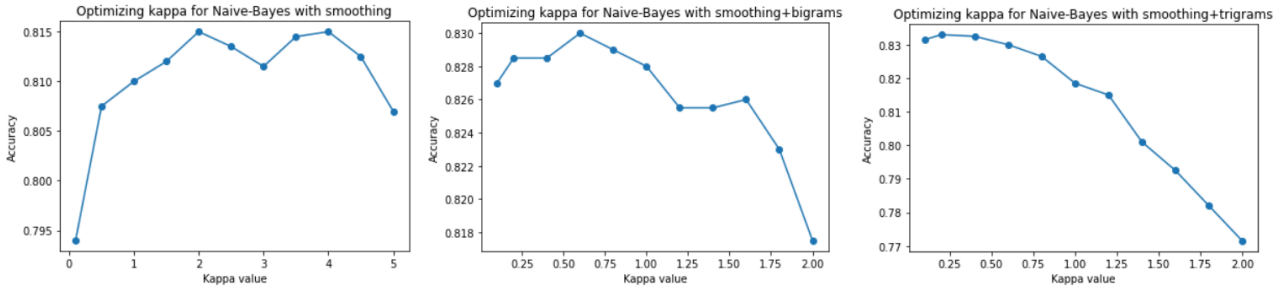


Figure 1.9: Tuning κ in Naive-Bayes using smoothing, bigrams and trigrams

Once the optimized values are set, the results from previous sections can be further improved using lowercased test (LC) and deleting common punctuation symbols (DP) that can be not useful as tokens. These results are gathered in table 1.1. Also, the number of unique features can be seen in brackets.

MODEL	LC	DP	LC + DP
NB+Smoothing	$81.9\% \pm 2.2\%$ [45642]	$81.5\% \pm 2.6\%$ [52914]	$81.7\% \pm 2.2\%$ [45635]
NB+S+Bigrams	$83.7\% \pm 2.3\%$ [473084]	$82.8\% \pm 2.4\%$ [538620]	$83.8\% \pm 2.2\%$ [502432]
NB+S+Trigrams	$83.6\% \pm 2.7\%$ [995432]	$82.9\% \pm 2.7\%$ [1006255]	$83.4\% \pm 2.4\%$ [979171]
NB+S+B+T	$83.5\% \pm 2.2\%$ [1422874]	$82.6\% \pm 2.3\%$ [1491961]	$83.7\% \pm 1.8\%$ [1435968]
SVM	$83.9\% \pm 2.1\%$ [45642]	$83.8\% \pm 2.0\%$ [52914]	$83.9\% \pm 2.1\%$ [45635]

Table 1.1: Comparing different extra approaches for Naive-Bayes and SVMs

In general, *some improvement is achieved in all models*. Naive-Bayes using **unigram tokens keeps an accuracy of 82%**, but **bigram and trigram models have been improved**, with an accuracy about **80%**. SVM performance is **not** considerably enhanced, having an accuracy of **84%**.

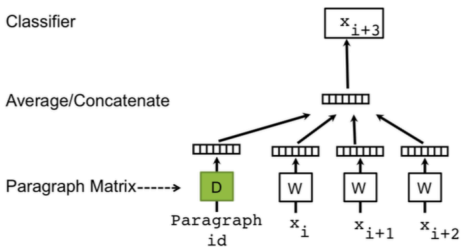
2. Part II: Extension using Doc2Vec

Many text classification and clustering algorithms require the text input to be represented as a fixed-length vector. Perhaps the most common fixed-length vector representation for texts is the bag-of-words (used so far) due to its simplicity and efficiency. However, the bag-of-words (BOW) has many disadvantages: the word order is lost, and thus different sentences can have exactly the same representation as long as the same words are used; they also have very little sense about the semantics of the words or more formally the distances between the words, i.e., “powerful” should be closer to “strong” than “Paris”.

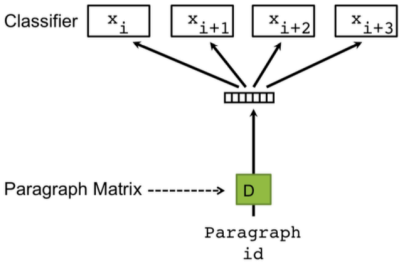
A recent paradigm in machine intelligence is to use a **distributed representation for documents**. Even though these representations are less human-interpretable than previous representations, they seem to work well in practice. In particular, Le and Mikolov [5] show that their method, *Paragraph Vectors*, capture many **document semantics in dense vectors** and that they can be used in **classifying movie reviews**. In this final part of the report, a little piece of its work is going to be studied.

Paragraph Vector is introduced using two different models: **“Distributed Memory”**, represented in figure 2.1a; and **“Distributed Bag of Words”** model, shown in figure 2.1b.

The experiments in this report focuses on both implementation of the Paragraph Vector using the Python library `gensim`, that implements **doc2vec model**, a generalization of Paragraph Vector.



(a) The distributed memory model of Paragraph Vector for an input sentence.



(b) The distributed bag of words model of Paragraph Vector.

Figure 2.1: Models of Paragraph Vector. The first model inserts a memory vector to the standard language model which aims at capturing the topics of the document. The authors named this model **“Distributed Memory”**. The paragraph vector is concatenated or averaged with local context word vectors to predict the next word. The second model simplifies Paragraph Vector using no local context in the prediction task. It is called **“Distributed Bag of Words”** model.

We are training various doc2vec models using the IMDB movie review database. Several sets of hyperparameters are going to be tuned: distributed memory model (`dm=1`) vs distributed BoW (`dm=0`) or hierarchical softmax for model training (`hs=1`) vs negative sampling (`hs=0`). Additionally, we are going to ignore all words with total frequency lower than `min_count={1,2,5}`, and the embedding space is going to be 50-dimensional or 100-dimensional. Looping through all theese hyperparameter options, we end up testing 24 doc2vec models, that transfom each document into a embedding (numerical vector representing the document). After all documents have been encoded into their corresponding embedding, they are splitted into training and testing set. Finally, SVM classifier is trained and tested on those vectors, getting the results shown in table 2.1.

MODEL	Accuracy	Std. deviation
D2V-DM-hs-mc1-vs50	80.72%	0.456%
D2V-DM-hs-mc1-vs100	81.37%	0.719%
D2V-DM-hs-mc2-vs50	79.65%	0.5%
D2V-DM-hs-mc2-vs100	81.26%	0.466%
D2V-DM-hs-mc5-vs50	79.60%	0.445%
D2V-DM-hs-mc5-vs100	80.98%	0.592%
D2V-DM-ns-mc1-vs50	83.86%	0.541%
D2V-DM-ns-mc1-vs100	85.72%	0.516%
D2V-DM-ns-mc2-vs50	83.64%	0.467%
D2V-DM-ns-mc2-vs100	85.83%	0.409%
D2V-DM-ns-mc5-vs50	84.20%	0.415%
D2V-DM-ns-mc5-vs100	85.90%	0.528%
D2V-DBOW-hs-mc1-vs50	87.72%	0.467%
D2V-DBOW-hs-mc1-vs100	87.13%	0.665%
D2V-DBOW-hs-mc2-vs50	87.62%	0.635%
D2V-DBOW-hs-mc2-vs100	87.02%	0.617%
D2V-DBOW-hs-mc5-vs50	87.79%	0.424%
D2V-DBOW-hs-mc5-vs100	86.89%	0.655%
D2V-DBOW-ns-mc1-vs50	89.54%	0.39%
D2V-DBOW-ns-mc1-vs100	**89.77%**	**0.693%**
D2V-DBOW-ns-mc2-vs50	89.66%	0.628%
D2V-DBOW-ns-mc2-vs100	89.6%	0.702%
D2V-DBOW-ns-mc5-vs50	89.72%	0.505%
D2V-DBOW-ns-mc5-vs100	**89.77%**	**0.74%**

Table 2.1: Results of different doc2vec models trained on SVM

Some results are quite promising compared to word2vec models (chapter 1). Analyzing table 2.1 we can infer the best hyperparameters for doc2vec algorithm. First, it is clear that **Distributed BoW** (DBOW) is a better approach than Distributed Memory (DM). Additionally, **negative sampling (ns)** seems to perform better than hierarchical

softmax (hs). The parameter that defines the minimum number of appearances of a word to be taken into account is **not important**, since there are not significant changes in he accuracy. Finally, in general the best results are obtained when the embedding space is **100-dimensional**. Models using Distributed BoW, negative sampling and 100-dimensional vector space are performing remarkably, with **accuracies of 90%** $\pm$ **0.7%**.

The **implemented Naive-Bayes classifier** (with Laplace smoothing, $\kappa = 1$) can be tested in **IMDB database**. The results are shown in figure 2.2, obtaining an **accuracy close to 82%**. These results **are significant** with respect to **Doc2Vec+SVM model**, since the latter had a performance of 90%.

```
NB_doc2vec = NaiveBayesText(smoothing=True,bigrams=False,trigrams=False,discard_closed_class=False)
NB_doc2vec.train(train_reviews)
NB_doc2vec.test(test_reviews)
NB_doc2vec_preds=NB_doc2vec.predictions
print(f"Accuracy: {NB_doc2vec.getAccuracy():.3f}")
print(f"Std. Dev: {NB_doc2vec.getStdDeviation():.3f}")
```

```
Accuracy: 0.816
Std. Dev: 0.007
```

```
p_value=signTest.getSignificance(NB_doc2vec_preds, SVM_doc2vec_preds)
significance = "significant" if p_value < 0.05 else "not significant"
print(f"results using Naive-Bayes are {significance} with respect to Doc2Vec-SVM in IMDB")
```

```
results using Naive-Bayes are significant with respect to Doc2Vec-SVM in IMDB
```

Figure 2.2: Trying IMDB in our implementation of Naive-Bayes (with smoothing)

2.1. Doc2Vec visual analysis: TSNE

The best model can be further studied using similar techniques as [6]. First, `tsne` is run in order to calculate the **two principal components** (using **PCA**) to analyze the **embeddings in a 2-dimensional space**. This can be seen in figure 2.3.

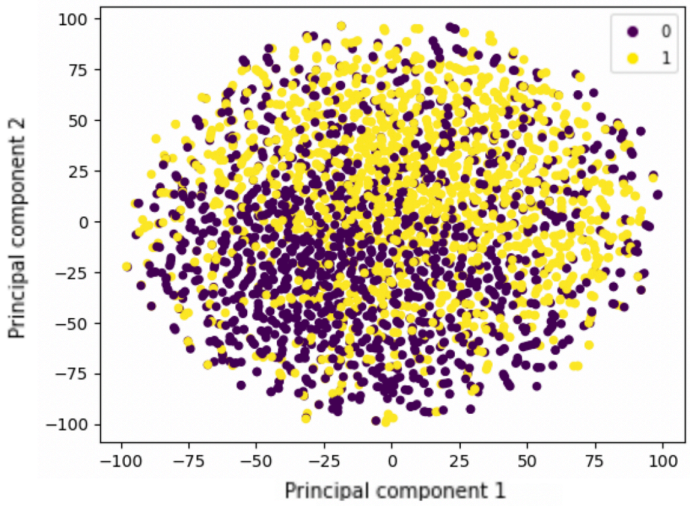
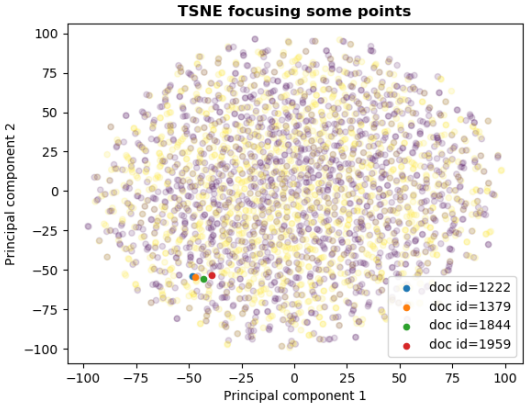
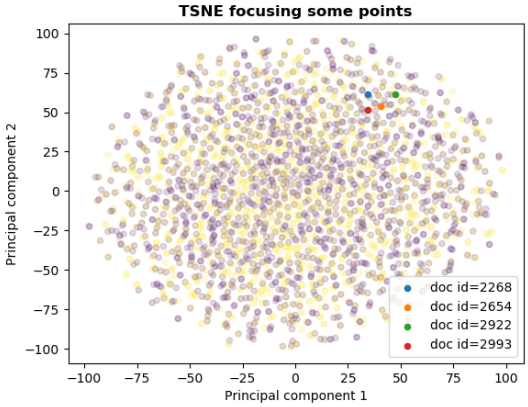


Figure 2.3: 3000 Embeddings in 2-D after using TSNE



(a) Some positive reviews embeddings projected in 2D



(b) Some negative reviews embeddings projected in 2D

Figure 2.4: Some reviews embeddings projected in 2D

The separation between classes is not really well defined, but we have to take into account that this might not be he best visual projection to discriminate the classes, since **100-dimensional vectors are being projected into a 2-dimensional space**.

Anyway, **positive reviews** tend to appear in the **bottom-left corner** (check figure 2.4a) and the **negative reviews** tend to appear in the **top-right** (figure 2.4b). For example, the review with id=1379 (figure 2.5) is correctly classified as positive since there are positive sentences such as ”it’s worth watching” or ”I would love to see this film again”, despite having other neative wods like ”insipid” or ”vular”. Review number 1844 (figure 2.6) is also correctly classified as positive thank to sentences like ”extraordinary scenes” or ”this highly moving film is a must see for those who love a good romance”. On the other hand, negative reviews (e.g. id=2268 in figure 2.7 or id=264 in figure 2.7) are correctly classified as negative with sentences like ”this comedy was a very cruel joke”, ”this is one oof the worst films we’ve ever seen” or ”this is bad enough”.

```
docs[25000 + 1379]
```

Python

```
TaggedDocument(words=['<START>', 'i', 'rarely', 'even', 'bother', 'to', 'watch', 'comedic', 'movies', 'or',
'television', 'these', 'days', "they're", 'insipid', 'vulgar', 'and', 'most', 'importantly', 'not', 'funny', 'this',
'one', 'could', 'be', 'seen', 'as', 'a', 'refreshing', 'blast', 'from', 'the', 'past', "it's", 'worth', 'watching',
'and', 'i', "don't", 'believe', 'it', 'would', 'be', 'dated', 'in', 'any', 'significant', 'way', 'classic', 'humor',
'is', 'classic', 'humor', 'and', 'good', 'writing', 'is', 'good', 'writing', 'regardless', 'of', 'the', 'era', 'in',
'which', 'created', 'br', 'br', 'i', 'would', 'love', 'to', 'see', 'this', 'film', 'again', 'it', 'came', 'to', 'mind',
'after', 'having', 'seen', 'the', 'somewhat', 'similar', 'summer', 'school', 'on', 'television', 'recently', 'like',
'that', 'slightly', 'newer', 'film', '<UNK>', 'is', 'funny', 'without', 'being', 'cruel', 'overly', 'sexual', 'or',
'<UNK>', 'in', 'bathroom', 'humor', 'the', 'key', 'of', 'course', 'is', 'how', 'the', 'adult', 'character', 'makes',
'such', 'a', 'difference', 'in', 'the', 'life', 'of', 'the', 'teen', 'character', 'maybe', 'even', 'a', 'virtually',
'life', 'saving', 'change', 'and', 'how', 'they', 'both', 'grow', 'up', 'in', 'the', 'space', 'of', 'a', 'summer'],
tags=[26379])
```

Figure 2.5: Review at test set with id=1379

```
docs[25000 + 1844]
```

Python

```
TaggedDocument(words=['<START>', 'between', 'sweeping', 'extraordinary', 'scenes', 'within', 'a', '<UNK>', 'of',
'indian', 'locations', 'and', 'a', 'plot', 'which', 'is', 'bollywood', 'inspired', 'yet', 'grounded', 'in', 'reality',
'this', 'highly', 'moving', 'film', 'is', 'a', 'must', 'see', 'for', 'those', 'who', 'love', 'a', 'good', 'romance',
'adventure', 'and', 'by', 'this', 'i', 'mean', 'good', 'not', 'a', 'hollywood', 'or', 'bollywood', 'easy', 'and',
'contrived', 'happy', 'ending', 'east', 'meets', 'west', 'in', 'a', 'different', 'take', 'on', 'the', 'buddy', 'film',
'the', 'performances', 'are', 'inspired', 'and', 'brilliant', 'the', 'cinematography', 'epic', 'and', 'the', 'plot',
'entertaining', 'engaging', 'and', 'poignant', 'without', 'a', 'trace', 'of', '<UNK>', 'or', 'cheap', '<UNK>', 'at',
'<UNK>', '<UNK>', '<UNK>', 'is', 'a', 'sweet', 'film', 'that', "won't", '<UNK>', 'your', 'teeth', 'see', 'it'], tags=
[26844])
```

Figure 2.6: Review at test set with id=1844

```
docs[25000 + 2268]
```

Python

```
TaggedDocument(words=['<START>', 'i', 'bought', 'this', 'video', 'on', 'a', 'throw', 'out', 'table', 'at', 'the',
'video', 'store', 'expecting', 'a', 'good', 'cast', 'in', 'what', 'was', '<UNK>', 'as', 'an', 'award', 'winning',
'brit', 'sex', 'comedy', 'i', 'guess', 'i', 'should', 'have', 'read', 'the', '<UNK>', 'print', 'i', 'rarely', 'write',
'a', '<UNK>', 'review', 'but', 'here', 'goes', 'br', 'br', 'these', 'actors', 'in', 'gay', 'roles', 'really', 'play',
'games', 'with', 'your', 'memories', 'of', 'a', 'lot', 'of', 'far', 'more', 'worthy', 'films', 'this', 'comedy', 'was',
'a', 'very', 'cruel', 'joke', 'at', 'the', 'expense', 'of', 'the', 'actors', 'the', 'theatre', 'going', 'public', 'and',
'of', 'all', 'the', 'nice', 'films', 'that', 'have', 'contributed', 'to', 'their', '<UNK>', 'br', 'br', 'i', 'repeat',
'is', 'the', 'joke', 'about', '<UNK>', 'the', "actors'", 'other', 'highly', 'respectable', 'on', 'screen', '<UNK>',
'with', 'this', 'trashy', 'flick', 'can', 'the', 'reference', 'to', 'the', 'austen', 'classics', '<UNK>', 'and', 'and',
'<UNK>', 'and', '<UNK>', 'be', 'anything', 'else', 'how', 'much', 'of', 'a', 'political', 'statement', 'was', 'it',
'to', 'produce', 'this', 'melodrama', 'using', 'these', 'stars', 'are', 'we', 'meant', 'to', 'simply', 'take', 'it',
'as', 'a', 'lay', 'down', 'that', 'all', 'actors', 'are', 'gay', 'and', 'thus', 'letting', 'their', 'on', 'screen',
'affect', 'our', '<UNK>', 'is', 'accepting', 'their', 'private', 'homosexual', '<UNK>', 'in', 'our', 'faces', 'too',
"i'm", 'sorry', 'but', 'i', "don't", 'think', 'so', 'i', 'say', 'no', 'to', 'this', 'one'], tags=[27268])
```

Figure 2.7: Review at test set with id=2268

```
docs[25000 + 2654]
```

Python

```
TaggedDocument(words=['<START>', '<UNK>', 'there', 'was', 'you', 'is', 'one', 'of', 'the', 'worst', 'films', "we've",
'ever', 'seen', 'it', 'fails', 'in', 'every', 'respect', '<UNK>', '<UNK>', 'was', 'better', 'as', 'an', 'actress', 'in',
'<UNK>', 'in', 'comparison', 'this', 'film', 'is', 'when', 'a', 'character', 'stumbles', 'once', 'or', 'even', 'twice',
'in', 'the', 'course', 'of', 'a', 'film', 'one', 'can', 'understand', 'it', 'but', 'character', 'falls', '<UNK>',
'stumbles', 'so', 'often', 'that', 'she', 'might', 'have', 'a', 'bit', 'of', 'jerry', 'lewis', 'in', 'her', 'in', 'her',
'<UNK>', 'each', '<UNK>', 'fall', 'was', 'probably', '<UNK>', 'choreographed', 'and', '<UNK>', 'and', '<UNK>',
'although', 'this', 'is', 'bad', 'enough', 'for', 'a', 'film', 'the', 'actors', 'dylan', 'mcdermott', 'and', '<UNK>',
'<UNK>', 'seem', 'to', 'spend', 'most', 'of', 'the', 'plot', 'going', 'out', 'for', 'a', 'smoke', 'or', 'trying', 'to',
'find', 'a', 'place', 'to', 'smoke', 'if', 'the', 'film', 'was', 'a', '<UNK>', 'on', 'having', 'no', 'place', 'to',
'smoke', 'ok', 'but', 'it', "isn't", 'br', 'br', 'however', 'long', 'this', 'film', 'runs', 'it', 'is', 'too', 'long',
'by', '10', 'minutes', 'past', 'the', 'running', 'time', 'br', 'br', 'oh', '<UNK>', '<UNK>', 'almost', 'acts', 'in',
'a', 'public', 'forum', 'meeting', 'you', 'almost', 'see', 'her', 'break', 'life', 'into', 'the', 'character', 'oh',
"it's", 'almost', 'as', 'convincing', 'as', 'her', 'scene', 'yelling', 'at', 'michael', 'douglas', 'in', 'basic',
'instinct', 'hmm', 'on', 'second', 'thought', 'not', 'really', 'br', 'br', 'this', 'is', 'a', 'film', 'to', 'avoid',
'at', 'all', 'costs', 'unless', 'you', 'need', 'a', 'cigarette', 'and', 'are', 'trapped', 'in', '<UNK>', '<UNK>',
'anonymous', 'or', 'forced', 'to', 'watch', '<UNK>', 'of', 'hooper', 'burt', 'reynolds', 'and', 'even', 'then', 'toss',
'a', '<UNK>', 'or', 'go', 'to', 'sleep'], tags=[27654])
```

Figure 2.8: Review at test set with id=2654

In figure 2.9 are represented the **well-classified reviews** (translucent points) and the **misclassified** ones, specifying whether they are false negatives or false positives.

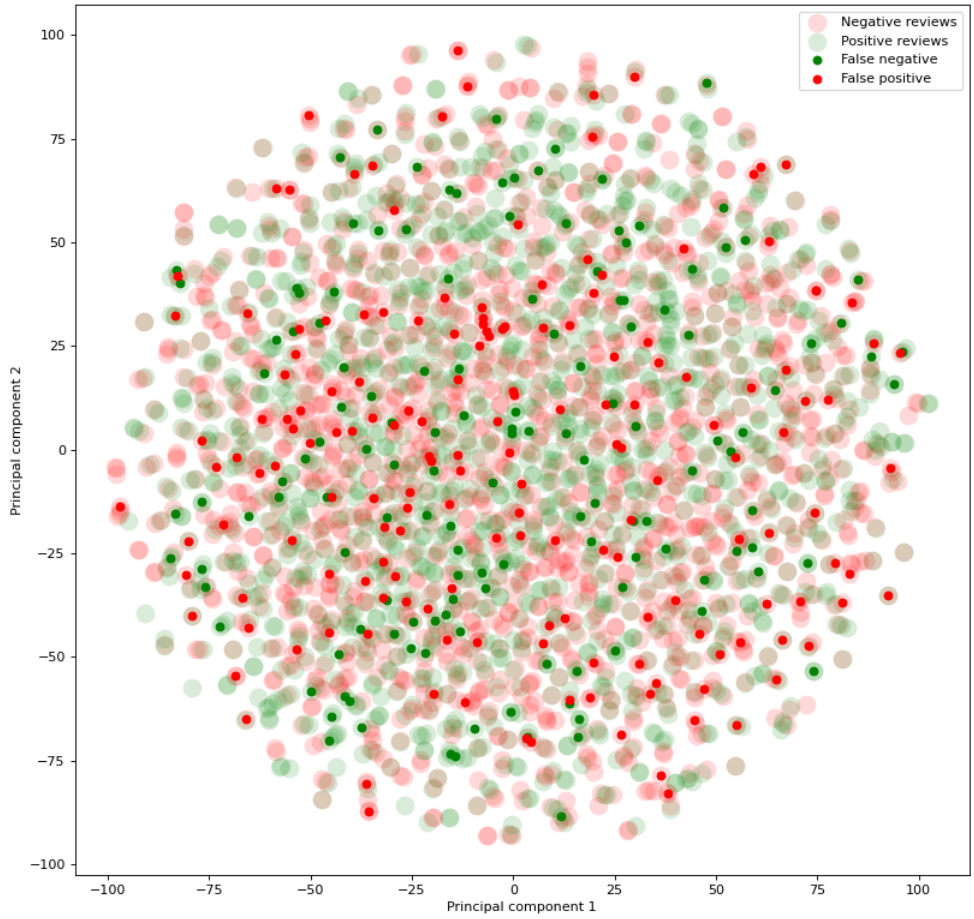


Figure 2.9: Well-classified reviews vs False positive vs False negatives

Finally, `tsne` can be also executed to get a reduced representation of the **embeddings in 3D**, as it is shown in figure 2.10 where **300 data points** are plotted. In the 3D case, **negative reviews** tend to be **higher** in the z-axis than **positive reviews**, defining a **clearer separation** than in the 2-dimensional case.

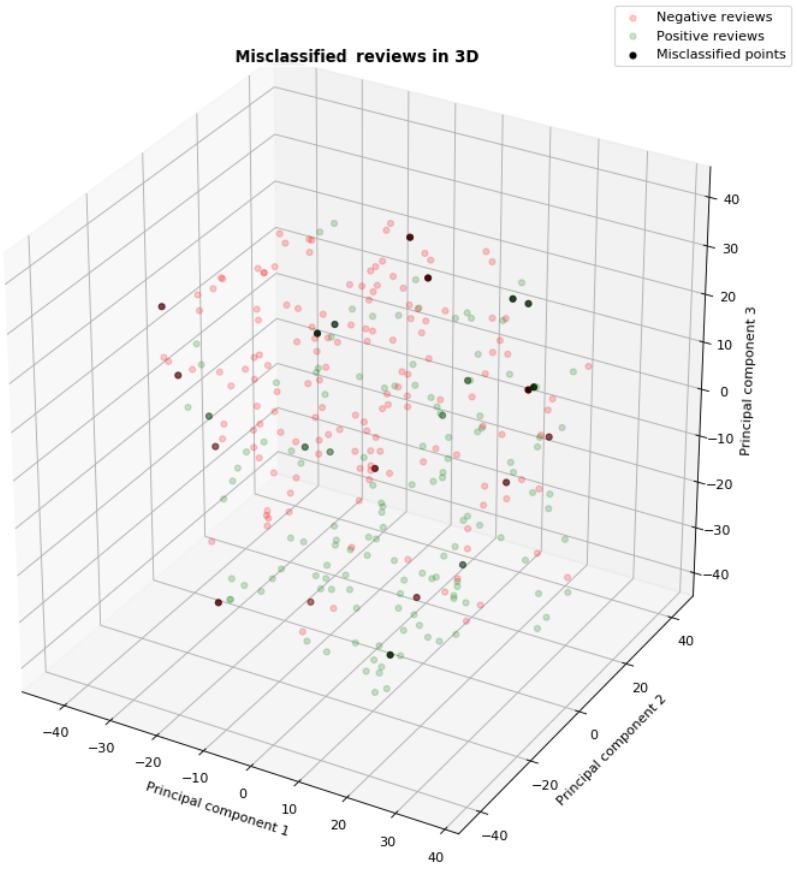


Figure 2.10: Well-classified reviews vs False positive vs False negatives in 3D

Bibliography

[1] Documentation for NLTK Stemer (2021). NLTK Project. Version 3.6.5 https://www.nltk.org/howto/stem.html

[2] Documentation for SVM (2021). Scikit-learn Project. Version 1.0.1. https://scikit-learn.org/stable/modules/svm.html

[3] Working With Text Data (2021). Scikit-learn tutorial. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

[4] Documentation for Doc2Vec (2021). Gensim Project. Version 3.8.3. https://radimrehurek.com/gensim/models/doc2vec.html

[5] Le, Q. and Mikolov, T. (2014). *Distributed representations of sentences and documents*. Proceedings of ICML.

[6] Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2015). *Visualizing and Understanding Neural Models in NLP*. Proceedings of NAACL.