# Data Mining Project

970357

December 29, 2017

## 1 Design of the data mining pipeline

### 1.1 Type of data

Film discussions is a textual data, which is an instance of dependency-oriented type of data. This is because there exist implicit inner dependencies of data points such as word order. Text data can also be considered a single long string, making it a univariate discrete sequence data type.

### 1.2 Type of super-problem

We are presented with a classification super-problem. This is a supervised method where are to train our classifier on labelled data items and later use it to classify new data instances.

### 1.3 Observations from data exploration

Our positive class contains two folders with 3290 and 3463 discussions, respectively. This amounts to a total of 6753 text samples with a label of 'good film' review. Similarly, our negative dataset contains two folders of 1428 and 1537 items and totals to a sum of 2965 bad film discussions. The size of total training set is then 9718 which is quite large and potentially inefficient to process in its entirety. Finally, our training dataset contains 1719 data items which need to be classified.

Each file contains 42 different entries which specify web review's details. The majority of the entries (such number of Facebook shares or publishing date) are irrelevant for the purpose of classification and would have to be discarded. Therefore, we settle on "text" and "title" as the most relevant attributes for our study. The files also contain "performance score" entry which was initially expected to be extremely useful for classification. Later it was revealed, however, that the entry has a value of 0 in majority of reviews which makes it irrelevant too.

We calculate distribution and measures of dispersion on the lengths of all film reviews. An average (mean) length of all positive class reviews is 254.93, with the shortest review having only 13 words and the longest one 993. The

mean of negative review lengths is 218.13 with shortest and longest discussions having 29 and 996 words respectively. Histograms showing the distribution of discussion lengths for both classes are shown on figures 1 and 2 below:

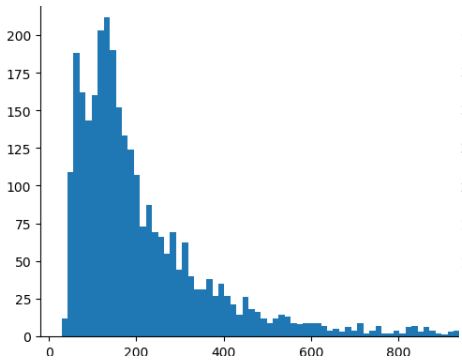


Figure 1: Negative class distribution

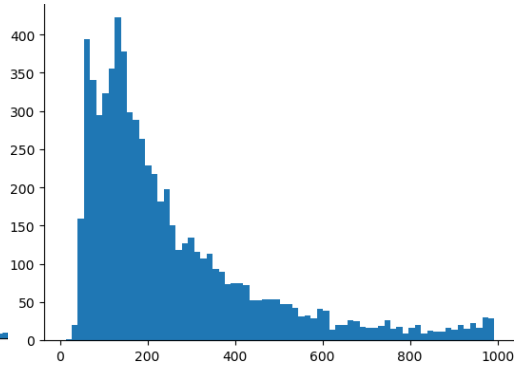


Figure 2: Positive class distribution

As we can see the word length distributions for our positive and negative classes look quite similar. Most lengths appear to be centred around the value of 180 words which is below their respective means. As the document length approaches 990 the number of texts rapidly decreases in both classes. This observation suggests that our training samples vary quite significantly in their lengths which will need to be accounted for.

Our testing set has 1719 film reviews with a mean word length of 260.96, shortest review of 78 words and longest of 989 words. Its distribution is shown on figure 3 below. Even though data samples across all three datasets (positive, negative, training) vary greatly in word length, none of them contain any empty text samples, which means in our data preparation step we will not have to address missing values.

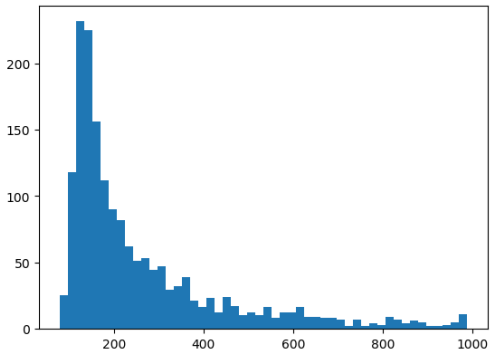


Figure 3: Testing set word length distribution

After removing punctuation symbols and extracting word stems (two procedures discussed later) from the full dataset of both classes we are presented with a lexicon of length 54,772 words. The figure 4 below shows how the length of lexicon varies with the number of discussions loaded. As we can see the rate of increase of lexicon size reduces gradually. This graph might help to choose an optimal sample size to use in our pipeline. For example, doubling the number of parsed discussions from 4,000 to 8,000 gives an increase of only about 10,000 words to our lexicon, but might take significantly longer to process. At this stage, however, we do not yet know if the length of our lexicon is positively related to our classifier performance and might be something to analyse in the next sections.

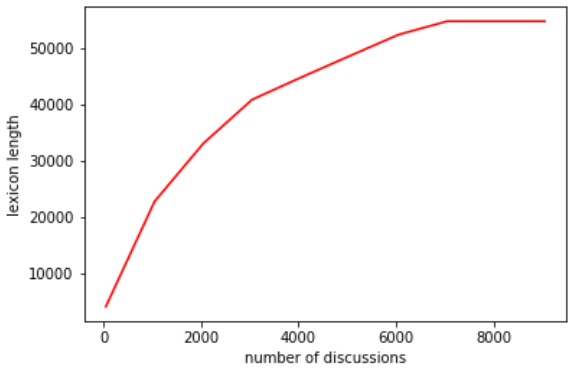


Figure 4: Variation of lexicon length with amount of data

## 1.4 Candidate methods for the data preparation stage

In our data preparation stage we propose to implement data cleaning, data reduction and dimensionality reduction. As mentioned above, our complete training dataset contains 9718 film reviews which is quite large. Therefore we propose to implement stratified sampling as a data reduction measure. We would parse an equal number of samples from each class, for example 500 from positive and 500 from negative discussions. We expect the sampling technique to greatly reduce computational times of our analytical stage. We choose stratified sampling specifically, to make sure both data classes are equally represented and avoid bias during classification.

To follow sampling we would implement three data cleaning steps. We propose to remove all punctuation symbols with regular expression module as well as make all words lower-case. We expect these steps to reduce length of our lexicon significantly. This is because same words written with and without apostrophe, or capitalised vs non-capitalised words would now count as one word. For example, “Wouldn’t”, “Wouldnt”, “wouldn’t” and “wouldnt” would be counted as a single word instead of 4 different ones. A lower lexicon is, sub-

sequently expected to increase accuracy of our classifier by making it focus on occurrence of words themselves instead of their variations.

As a next data cleaning stage we propose replacing all words in our data with their extracted word stems. This can also be viewed as a dimensionality reduction step, because in a vector space model (VSM) such as bag of words each word of a lexicon represents a dimension. We expect that fewer dimensions will improve our computational times and reduce a risk of curse of dimensionality via overfitting. Finally, we would need to filter out 'stop words' from our dataset. These are very frequent words such as 'the', 'a', 'to' which carry little semantic meaning on their own. This could be achieved by either importing a popular list of English stop words, or by counting the frequency of all words in our dataset and removing an X amount of terms with the highest count.

## 1.5 Candidate methods for the analytical stage

As a first analytical step we are going to create a lexicon of all words which are left after data cleaning and use it to fit our data into a bag-of-words model. Bag of words transforms dependency-oriented data into multidimensional data type, because word order is lost. It is also characterised by being very sparse across attributes, where the vector sparsity will vary directly with document length. This sparsity and its dependency on text length have a very adverse effect on Lp-norm metrics. This will lead to texts with similar lengths having shorter distances (and being more similar) according to Lp-norm. Therefore, we propose to use cosine similarity to classify data, which is insensitive to document length. We use unweighed cosine similarity according to the formula below and will assign a more a specific normalisation weight shortly.

$$cos(X, Y) = \frac{\sum_{i=1}^{d} x_i \times y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \times \sqrt{\sum_{i=1}^{d} y_i^2}}$$

As a next analytical step we propose to calculate term frequency-inverse document frequency (tf-idf) for each attribute. Tf-idf can be found by dividing total number of documents by number of samples where term t appears, followed by taking a natural logarithm of the result. The new bag of words matrix could then be calculated by multiplying tf-idf of each term with each value in that column. As an outcome of this technique we expect the new matrix to reflect importance of words in a text, as opposed to simply their presence. Thereby, frequent terms will be scaled down while infrequent words assigned a higher value.

As a final step before classification we propose to normalise each document-vector by converting it into its unit-vector form. Vectors in their unit form have the same direction as before, but have a magnitude (length) of 1. This can be achieved by dividing each value in a vector with its current magnitude $m$ calculated as follows:

$$m = \sqrt{x_1^2 + x_2^2 + x_3^2 + ... + x_d^2}$$

This way, each document-vector will be of magnitude 1 and its length will no longer affect the effectiveness of Lp-norm metrics. Even though we are going to use cosine similarity which is insensitive to document length, transforming our bag of words into a matrix of unit vectors will allow us to use other metrics in our classifiers and compare the results.

Finally, in our classification stage we propose to test stochastic gradient descent (SGD) and nearest centroid (NC) classifiers. Our motivation behind choosing SGD is that it is known to be effective with sparse data (as is in our case) as well as, widely used in natural language processing. The nearest centroid is a popular method choice for binary text classification too. It works by computing a centre vector (centroid) for each data class and assigns new data instances to the centroid they are the closest to. In our film classification we expect the direction of both centroid vectors to represent the positive and negative semantic patterns for each class respectively, which could be then compared to vector directions of new documents. As a choice of distance metrics we are going to test L2-norm and cosine similarity, as discussed previously. Again, transforming each document-vector to its unit form will make L2-norm metric effective, because text length will no longer affect the result. To measure the performance of our trained model we propose to use k-fold cross validation (kNN) with $k = 5$. The cross validation will split the training dataset into 5 bins of equal size. The method will use 4 partitions for training and 1 for testing with each iteration, altogether performing 5 learning experiments and using new testing bin each time. Then we propose to take the mean of the five test results which would represent an overall performance of a particular classifier. As an outcome, we expect the cross validation technique to help us assess the learning algorithm more accurately. In addition, it will allow each data point to participate in both training and testing cycles which is expected to yield a more generalisable result.

# 2 Implementation

In addition to our main.py script, we provide a jupyter notebook file as an alternative. The notebook executes our data mining pipeline in the same order as main.py but adds jupyter functionality. It allows us to run the pipeline in a step-by-step manner and is expected to help us understand computational times of each particular part.

# 3 Validation, tests, and discussions

Out of the proposed methodologies our pipeline implemented the following methods. First, stratified sampling was used during the data loading phase to extract 500 texts from each class. Next we carried out the three data cleaning steps in the following order: remove all punctuation including “ ’ ” from each document, make all words lower-case, extract stem from each word in our

dataset using SnowballStemmer from nltk.stem module. These three steps have reduced the size of our lexicon (in the later stage) to 15,249 words. This is contrasted to the initial lexicon that would be constructed from unclean data and would contain 34,808 terms. Following this, we created a lexicon of all words present in our total data sample (positive and negative). We, then, use the lexicon to create bag-of-words (BOW) model and fit our entire data sample into it. Our BOW can now be describes as a matrix with 1,000 data items and 15,249 attributes. During the word stem extraction different word variations get transformed into a single stem-attribute which can now also be viewed a dimensionality reduction method. We then compute term frequency-inverse document frequency for each word and multiply these values with respective columns of each vector. After this, we normalise each document-vector by converting it into its unit vector form. This is achieved by dividing every value in each row by its vector magnitude (length).

In our next stage we use nearest centroid and stochastic gradient descent classifiers and compare their performance. We also use k-fold cross validation with $k = 5$ with each performance test and report the mean of the 5 learning experiments. Our nearest centroid (NC) method was tested using cosine similarity as well as L2-norm distance metrics. Given the sample size of 1000 texts (500 positive, 500 negative) the classifier successfully predicts if a film is good or bad with 85.2% accuracy when we use cosine similarity. This is increased to 85.7% with L2-norm metric, which is 0.5% improvement.

We use the same sample size and cross validation method for our stochastic gradient descent (SGD) classifier. For the parameters we choose 'l2' penalty and experiment with the random_state parameter. Random_state affects initialisation of random number generator used by SGD. Testing showed that the classifier performs best when random_state is initialised to 46. The figure 5 below shows how this result was obtained where x represents random_state value and y - our SGD score. The classifier, then, outputs a score of 88.5% which is 2.8% higher than the nearest centroid algorithm with its optimal parameters. After increasing our sample size to a total of 3,000 the two classifiers perform slightly better, each. The score of our NC algorithm increases to 87.37% which is an improvement of 1.67%. The performance of SGD increases to 90.07% and is 1.57% difference from the previous result. This suggests that increasing sample size of our training dataset will return much diminished gains to our classifiers' accuracies. In other words, increasing the sample size by 300% results in the improvement of accuracy of 1.67% to 1.57% depending on classifier. This finding might, for example, help us to choose sample size in the future if computational time will be a constraint.

We chose stochastic gradient descent algorithm with random_state = 46 to carry out classification of the new data items. Our pipeline described above was implemented to process the new film reviews with the exception of bag-of-words creation step. Instead of creating BOW from the lexicon of new data we use our old lexicon and fit new data instances according to the presence of corresponding terms. We then perform classification using our trained SGD model which results in a list of 1719 items of 0 and 1 and represents the classifier prediction.

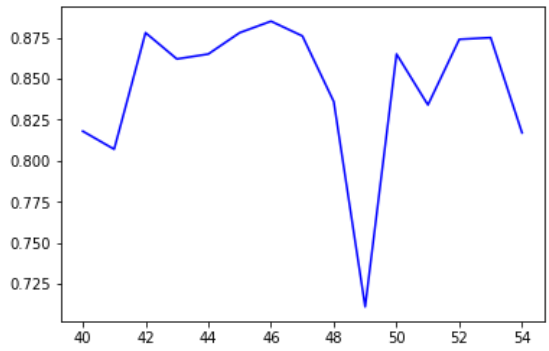


Figure 5: Variation of SGD performance with random_state parameter

Out of 1719 films our SGD classifies 1041 films as good and remaining 678 as bad. For the sake of an experiment we also conduct classification with our nearest centroid method using L2-norm. This results in a separation of 1054 and 665 films to good and bad classes respectively, which is a very similar division as with SGD. We, then, compare the classification of each particular data instance between our two algorithms. With the sample size 1000 the methods give a case-by-case match percentage of 90.29%, which is quite similar.

To analyse failure cases we need to present our SGD with data that was not used for training, yet has a class label. Therefore, we load data instances from our negative class with document numbers ranging from 501 to 520. Within the range SGD wrongly predicted the outcome for films including "La La Land", "The Wailing" and "Freddy vs. Jason". We notice that, to an extent, these film reviews use a rhetoric of negating positive statements. For example the review of "La La Land" contains the sentence "Nice to look at, pleasant but nothing special". The phrase uses highly positive words such as 'nice', 'pleasant' and 'special'. On their own the words may appear to be positive which is possibly why our classifier labelled them as good. In the "Freddy vs. Jason" example the review uses words such as 'boring' and 'uninspiring' which are negative, but also uses 'not epic' which can be considered a heavy weight towards positive class by SGD. From this we can conclude that classification of discussions bases on single-word attributes is somewhat limited in its potential accuracy. If we were to improve our pipeline, we would need to transform the data in a way which can capture negation of positive words and recognise word combinations. A promising methodology to achieve this is the latent semantic analysis classifier which maps word combinations of particular length to newly defined concepts.