Report on Review Classification

ID: math.random12 **Accuracy:** 0.82 **Rank:** 8

Problem Statement:

Given a data set of 18000 reviews with their sentiment classified as positive or negative. Expectation is to build a K-Nearest Neighbor classifier which will classify reviews with unknown sentiment to one of these categories.

Approach Followed to solve the problem:

There are several steps performed before classifying reviews with unknown sentiment. Those are broadly categorized into the following.

1. Data cleaning
2. Feature extraction
3. Feature subset selection
4. Training the model
5. Cross validation
6. Classifying unknown documents

Details of each step are discussed in the following sections.

1. Data Cleaning:

The goal of this process is to reduce as much noise as ***possible***. So that during the following steps we are assured that we are dealing with cleaned data.

The training data set contains a lot of punctuation, digits and escape sequence. During the implementation, I tried to remove as many of those as possible. Also, most of the reviews contain capital and small letters which could be treated as different words during the feature extraction process. This will unnecessarily increase the number of features but they represent the same word. To avoid this problem, all of them are converted into small letter characters before any further processing.

There are a lot of stop words in almost all reviews. These are words like “the”, “in”, “a”, “an” etc. They don’t represent anything relevant about the documents class. We don’t need those words for further processing. In the code I’m using NLTK to tokenize and then remove the stop words.

We represent the same word differently. Like “do” and “doing”, they generally refer to the same thing and have similar influence on the final sentiment. Computer does not understand grammar so it will try to treat “do” and “doing” as two different words. To get rid of this issue I am using **lemmatization**.

I have used lemmatization over stemming intentionally. The thought process behind choosing lemmatization is that it would find more accurate root words. Finding correct root words helps in representing documents more accurately. For example, if we stem “studies” it will return “studi” and stemming on study will return “study”. On the other hand, lemmatization will return “study” for both of them. With part of speech tagging lemmatization could choose root words according to the context.

Because of all the above reasons I am using lemmatization with part of speech tagging to generate root words. After performing all the steps I am joining all the words using a single space. These are the reviews that will go for further processing.

2. Feature Extraction

The next step is to extract meaningful information which will be used to represent the document. Before extracting features let's split the training data set into two different parts. One part we are going to use for training our model and the other part will be used for checking how well we are predicting the sentiment labels. During splitting I have used a function parameter named ***stratify,*** value of this is set to the ***target*** variable. Stratify is a sampling technique to select a ***balanced data set*** based on a column variable. In our case the column variable is the sentiment. We do not want to endup with a sample which has a bias towards a particular type of sentiment. Also, the ***train_size*** is set to ***0.85*** which means ***85%*** of the data will go for training and ***15%*** data will be used for cross validation later.

As the first step I'm using Tf-idf vectorizer to create our document representation with features (basically the frequency of words). Note that in the first step I am **only trying to build a dictionary** of words present across our training split. So, in the function ***use_idf*** is set to ***false***. After forming the table we could see all the features, essentially the words that are used to represent all the documents. There is a parameter ***min_df*** present in the tf-idf vectorizer, details about this are discussed in the next section. Another thing to highlight here is that the word count here is ***normalized*** to represent all documents on the same scale.

During the process of creating our document representation I am using both ***unigram*** and ***bigram.*** Unigrams are just single words. Bigram contains two words. Bigram helps presenting documents more accurately, for example if there is a review having phrase "***Not Good***" the unigram may get trapped by considering good as a potential feature for positive sentiment. But in the ***Bigram*** model it would know that if something is closer to this axis that means it has some negative influence.

3. Feature Subset Selection

In this step I have tried to select features that best represent and influence the classification of documents. To select features I have mainly used two different approaches.

1. Drop features that are very infrequent.
2. Find the features that are correlated more with the target variable and could be a potential influencer in prediction.

To drop infrequent features, with *tf-idf* I have used ***min_df*** in the function parameter, the value of which is set to ***.0095.*** This means that only select terms that appear at least in ***0.95%*** of the documents in the training split. Because if the world is very infrequent it makes no sense to add them in the feature table.

The second one is a bit complicated and uses statistics to find the correlation between features and categorical variables. The test is known as ***Chi-Square Test of Independence****.* In this test the null hypothesis says there is no correlation between two variables. In our case it is features and sentiment. The alternative hypothesis says there exists a correlation between them. Based on the test result we reject one hypothesis. But my motive here is ***not to do hypothesis testing*** but to use *Chi-Square values*. As ***higher*** value corresponds to ***higher correlation*** between feature and target variable. Mathematical expression for calculating the value is the following

Chi-Square Test Statistic

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

Observed value is just the frequency count of each term. Expected value is calculated considering both feature and target values are independent of each other. It can be calculated as the row count multiplied by column count divided by the total sum of occurrence of all the words.
I used the sklearn provided library which would select ***K-best features*** out of the dictionary based on their ***chi square value***. Also, I am setting ***K*** to ***(length of dictionary) * 0.104 .*** Which means it would only take the top ***10.4%*** of the features having highest chi square value out of the total features we have selected.

All these K-features are the final features that we are going to use for prediction. A new tf-idf vectorizer is built out of these K-specific terms with ***use_idf*** set to ***true***. This time we are using inverse document frequency to prioritize local words.

4. Training the Model

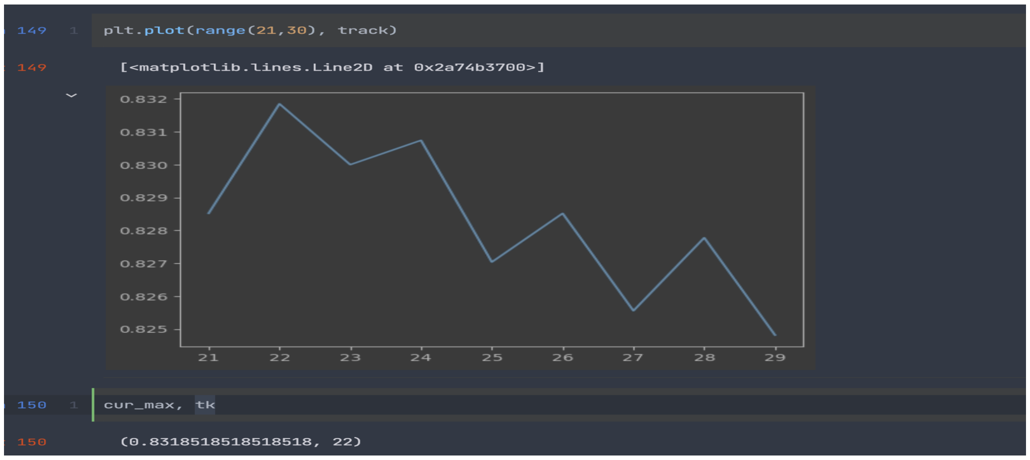
K-nn is a supervised model that maps all the documents based on their features in a hyper space. Then it calculates the distance of the test data from each of the training points. Majority class of the first K training point sentiment will be the predicted class of the test document.

In the code for distance measure I am using ***euclidean distance.*** Also, the distance is ***cubed and then it is inverse*** , this value is the ***weight*** of the test data from a training data point. Inversing the distance will give ***higher weightage to closer data points*** as compared to data points having more distance.

5. Cross Validation

I kept all the predicted sentiments for each of the data points in the test split in an array and compared it with the actual sentiment to calculate the accuracy. This process is repeated several times

for different K-values and optimal K is chosen based on its accuracy. I tested this and the diagram represents a line graph for different k values and their respective accuracy on the vertical axis.



```
149   plt.plot(range(21,30), track)
149   [<matplotlib.lines.Line2D at 0x2a74b3700>]
```

```
150   cur_max, tk
150   (0.8318518518518518, 22)
```

The K-value with highest accuracy is chosen to predict the sentiment of unlabeled test data.