Sentiment Analysis of Amazon Reviews

Binary Classification with Naïve Bayes and KNN

Author: Ankur Kumar

July 14, 2020

Contents

Chapter 1

Introduction

1.1 Problem Statement

People are using social media increasingly to express their views on the products that they have recently purchased, and companies do not have a structured way of extracting this data and analyzing it for sentiment. Most companies still rely on reviews being written to them or posted to their page to understand the impact of their product.

In this project we will be creating a tool to analyze the sentiments of the reviews whether they are positive or negative labelled as 1 or 2.

1.2 Description of Dataset

The project is based on the set of reviews provided by Amazon. The project aims to use this labelled dataset to develop a tool that can perform textual sentiment analysis for other unlabelled data e.g., Twitter and other social media.

1.3 Data Extraction and Cleaning

The dataset is downloaded from http://jmcauley.ucsd.edu/data/amazon/. Then we process the dataset using the following steps.

- Check for and remove all the empty reviews and ratings
- Remove all punctuations from the reviews
- Change the case of every word to lowercase

Chapter 2

Feature Engineering

Since textual data cannot be fed directly into a model, we first have to transform the textual features into numerical features. This can be done either by Tokenization based on different algorithms or using word embeddings which are learned representations of words where words with the same meaning have similar representations.

2.1 Tokenization

2.1.1 TF-IDF

The Tokenization method used is known as TF-IDF(term frequency-Inverse Document Frequency). The idea behind this algorithm is to weight frequently occurring terms in a sequence highly unless they occur frequently throughout the document. This enables the algorithm to give low importance to commonly used words which provide no meaning at all while capturing keywords that indicate sentiments.

Mathematically, the score for each word in a document is calculated as:

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, d)$$
$$IDF(t, D) = log\ |D| + 1/DF(t, D) + 1$$

where |D| is the total number of documents, TF(t,d) is the number of times in a term , t appears in a sequence, d and $DF(t, d)$ is the number of times the term appears in the document D. We can see an example below which shows how a document of sequences can be transformed using $TF - IDF$.

$$\begin{bmatrix} \text{This was a great product} \\ \text{This was a bad product} \\ \text{This was ok} \end{bmatrix} \overset{\text{TF-IDF}}{\Rightarrow} \begin{bmatrix} 0.15 & 0.15 & 0.15 & 0.30 & 0.20 \\ 0.15 & 0.15 & 0.15 & 0.30 & 0.20 \\ 0.15 & 0.15 & 0.30 & 0 & 0 \end{bmatrix}$$

2.2 Lemmatization

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*. If confronted with the token *saw*, *lemmatization* would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun.

2.3 Stop Words

Stop words are words which are filtered out before or after processing of natural language data (text).[1] Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.

Some of the most common, short function words, such as *the*, *is*, *at*, *which*, and *on* are filtered out after the processing.

Chapter 3

Machine Learning

3.1 Naïve Bayes Classifier

Naive Bayes is a classification algorithm for binary (two-class) and multiclass classification problems. It is called Naive Bayes or idiot Bayes because the calculations of the probabilities for each class are simplified to make their calculations tractable.
It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Above,

- $P(c|x)$ is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of a predictor given *class*.
- $P(x)$ is the prior probability of the predictor.

3.2 k-NN Classifier

In pattern recognition, the ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric method proposed by Thomas Cover used for classification here. The input consists of the *k* closest training examples in the feature space. The output is a class membership.

An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

$$\widehat{f}\,(\overline{x}) = 1/k \sum_{i' \varepsilon k_{\hat{x}}} y_{i'}$$

Chapter 4

Model Evaluation

4.1 Model Implementation

For our dataset, we have chosen the following models for our final discussion.

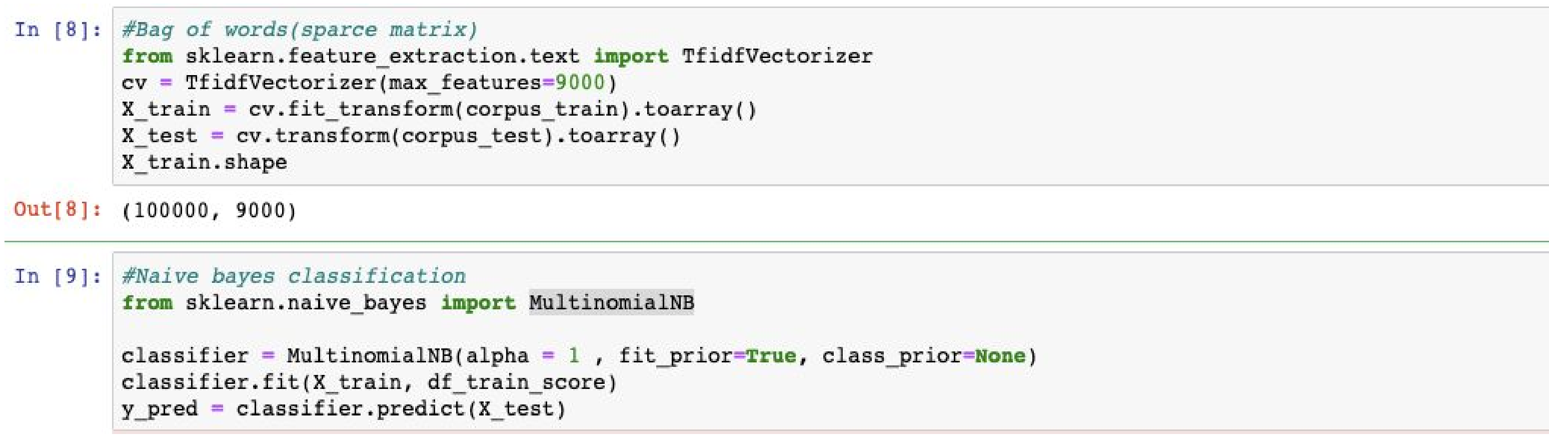
1. TF-IDF Tokenization with Naïve Bayes Classifier
2. TF-IDF Tokenization with KNN Classifier

The training data is huge and hence we have only taken a small chunk of the data and different number of rows in different algorithms due to limited computation power in the laptop. All the code has been executed on the local machine and hence.

- Number of labels: 2
- Maximum words for tokenization: 9k
- Validation metrics: Accuracy

4.2 Model Architecture

Model 1: TF-IDF Tokenization with Naive Bayes



```
In [8]: #Bag of words(sparce matrix)
        from sklearn.feature_extraction.text import TfidfVectorizer
        cv = TfidfVectorizer(max_features=9000)
        X_train = cv.fit_transform(corpus_train).toarray()
        X_test = cv.transform(corpus_test).toarray()
        X_train.shape

Out[8]: (100000, 9000)
```

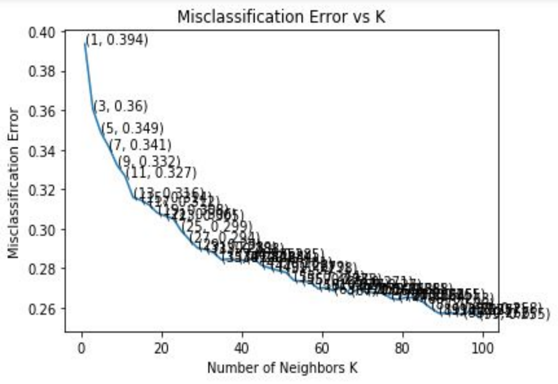
```
In [9]: #Naive bayes classification
        from sklearn.naive_bayes import MultinomialNB

        classifier = MultinomialNB(alpha = 1 , fit_prior=True, class_prior=None)
        classifier.fit(X_train, df_train_score)
        y_pred = classifier.predict(X_test)
```

The first model is implemented using the above mentioned code where the total rows available in the training dataset were 3.6 million. However, we have only used 100000 rows to train the model. We might see an impact on accuracy but due to limited computational power, we could only take a chunk of the large dataset.

Model 2: TF-IDF Tokenization with k-NN Classifiers

```
In [9]: from sklearn.model_selection import cross_val_score
        from sklearn.neighbors import KNeighborsClassifier
        optimal_k_bow = k_classifier_brute(X_train, df_train_score)
        optimal_k_bow
```

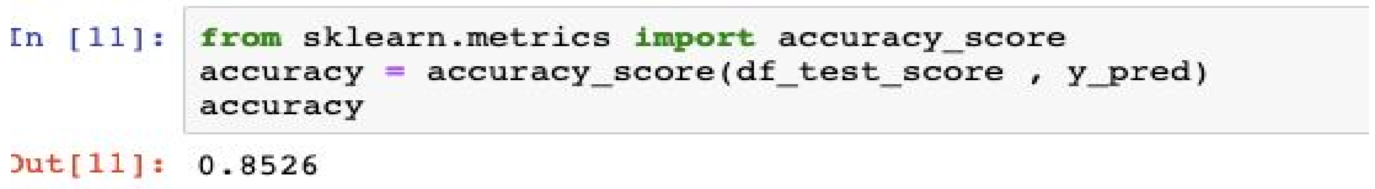


```
the misclassification error for each k value is :  [0.394 0.36  0.349 0.341 0.332 0.327 0.316 0.314 0.312 0.308 0.306
0.305
```

```
In [10]: #K-NN training
         from sklearn.neighbors import KNeighborsClassifier
         classifier = KNeighborsClassifier(n_neighbors= 99, metric = 'minkowski' , p = 2)
         classifier.fit(X_train , df_train_score)
```

This model is implemented using the above mentioned code and in this we could take a very small training dataset of 5000 rows and hence we could not perform the split of training data in training and validation set because the dataset was already small and further smaller dataset would result in not an accurate model to be deployed.

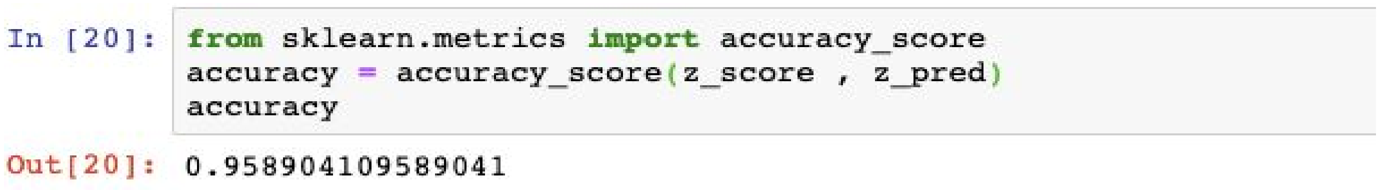
4.3 Results



```
In [11]: from sklearn.metrics import accuracy_score
         accuracy = accuracy_score(df_test_score , y_pred)
         accuracy

Out[11]: 0.8526
```

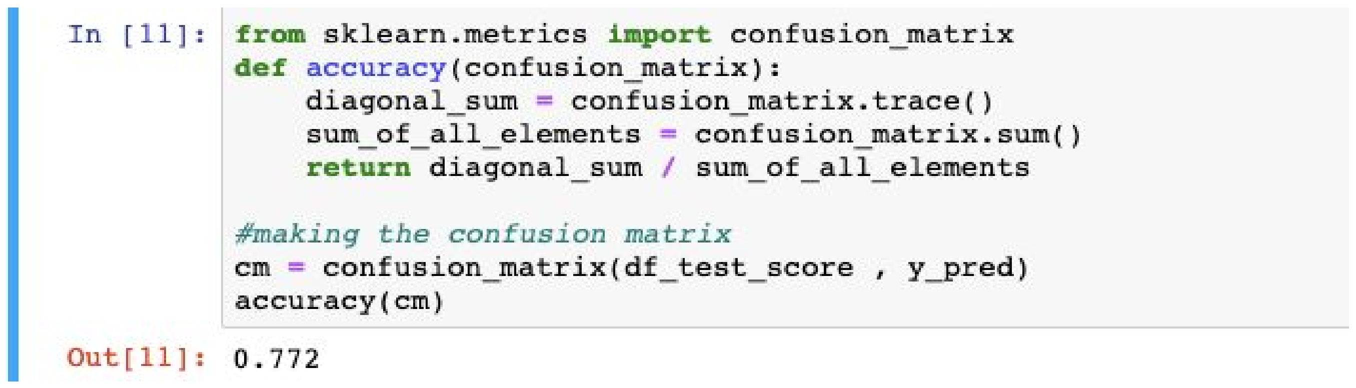
Test results for TF-IDF model with Naive Bayes



```
In [20]: from sklearn.metrics import accuracy_score
         accuracy = accuracy_score(z_score , z_pred)
         accuracy

Out[20]: 0.958904109589041
```

Production results for TF-IDF model with Naive Bayes

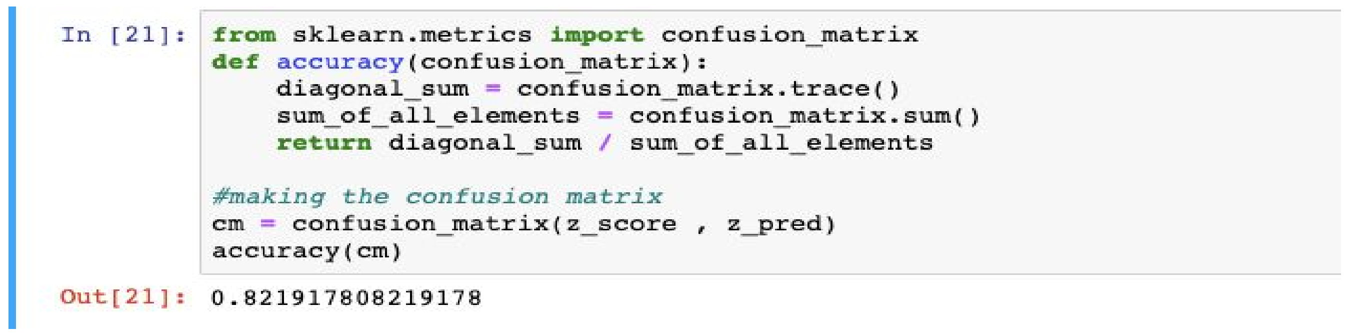


```
In [11]: from sklearn.metrics import confusion_matrix
         def accuracy(confusion_matrix):
             diagonal_sum = confusion_matrix.trace()
             sum_of_all_elements = confusion_matrix.sum()
             return diagonal_sum / sum_of_all_elements

         #making the confusion matrix
         cm = confusion_matrix(df_test_score , y_pred)
         accuracy(cm)

Out[11]: 0.772
```

Test results for TF-IDF model with k-NN classifier



```
In [21]: from sklearn.metrics import confusion_matrix
         def accuracy(confusion_matrix):
             diagonal_sum = confusion_matrix.trace()
             sum_of_all_elements = confusion_matrix.sum()
             return diagonal_sum / sum_of_all_elements

         #making the confusion matrix
         cm = confusion_matrix(z_score , z_pred)
         accuracy(cm)

Out[21]: 0.821917808219178
```

Production results for TF-IDF model with k-NN classifier

Based on the results on the test set, we can conclude that the better performing model is Naive Bayes with 85.26 % accuracy which is higher than k-NN classifier.
We also have seen one more reason for higher accuracy is that the dataset for k-NN was very small as compared to Naive Bayes

Chapter 5

Conclusions & Future Work

The project has shown that the more complex models are not necessarily better performing. It is possible that the more complex model would have performed better with more data and more computational power. It is also possible that not enough has been done to tune the hyper parameters.

Possible future work to undertake:

- Writing a parallel programming code to use all the cores of local machine
- Fine Tuning of existing models
- Using other machine learning models such as Logistic Regression and SVM

References:

[1] Stemming and Lemmatization
https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html#:~:text=Lemmatization%20usually%20refers%20to%20doing,is%20known%20as%20the%20lemma%20.

[2] Naive Bayes Algorithm
https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[3] k-Nearest Neighbours
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[4] TF-IDF
http://www.tfidf.com/

[5] Smokes and Whiskey Data Set
https://www.amazon.com/Smokes-Whiskey-Tejaswini-Divya-Naik-ebook/dp/B07P38QSQ6/ref=tmm_kin_swatch_0?_encoding=UTF8&qid=&sr=