Machine Learning Engineer Nanodegree

Capstone project Report

Ahmed Bahgat
February 8th –*9th* 2019

I. Definition

Project Overview

My capstone project is about *sentiment analysis*, i.e. the field that “refers to the use of natural language processing, text analysis, computational linguistics, and bio metrics to systematically identify, extract, quantify, and study effective states and subjective information”.
In particular, this project aims at classifying more than 50,000 real Movies Reviews according to their sentiment (i.e. positive or negative), by only looking at the raw text. For example, if a reviewer says “What an interesting movie!”, then the algorithm should predict this text to be that of a **positive** review.

In general, traditional approaches to sentiment analysis rely on a count-based representation of text data. In this case, the degree of presence (or absence) of certain words - irrespective of their ordering in the text - is the only determinant factor in explaining the sentiment. Recent developments in the field of deep learning, however, allowed for significant changes:

one of these changes is:
- different text data representation by using word vectorization (word embedding) like (Word2Vec, GLoVe);

Such changes aim at providing a more comprehensive representation of both words and their ordered sequence, by potentially unlocking a semantic-like understanding of text. In other words, text data does not become a simple count-based representation of keywords, but an actual - yet numerical - representation of meanings and contexts.
Which means the word not only be affected by its order sequence but also by the words that appear in its context.

Problem Statement

Applying Sentiment Analysis on some text to see how we can teach the Computer to feel the same when he read the same text data, is one of the oldest and earlier necessary task, which the linguists, ML engineers has been facing, And understanding what the human say, what he feels, and what he wants to say with writing those words can lead us to a brand new era, In this project I will work on 26000 positive and 26000 negative processed movies reviews. 2000 of them Introduced in Pang/Lee ACL 2004 Paper. Released June 2004. and with the help of the ML methods we have, that I mentioned above I will try to correctly classify as correctly as possible between those two categories and compare the results I’ll get with what they’ve got and then try to apply newer techniques on the same data.

As input we are provided with 52000 different reviews differs in the length, the polarity and even the style as they’re real data so each review is probably written earlier by different person.

Since our task is to distinguish between two classes, it is essentially a classification task
- first part of the data used in the experiments described in Pang/Lee ACL 2004.

  Specifically:

Within the folder "txt_sentoken" are the 2000 processed down-cased text files used in Pang/Lee ACL 2004; the names of the two sub-directories in that folder, "pos" and "neg", indicate the true
classification (sentiment) of the component files according to our automatic rating classifier.
File names consist of a cross-validation tag plus the name of the original html file. The ten folds used in the Pang/Lee ACL 2004 paper's
experiments were:
fold 1: files tagged cv000 through cv099, in numerical order
fold 2: files tagged cv100 through cv199, in numerical order
...
fold 10: files tagged cv900 through cv999, in numerical order Hence, the file neg/cv114_19501.txt, for example, was labeled as negative, served as a member of fold 2
Each line in each text file corresponds to a single sentence, as determined by Adwait Ratnaparkhi's sentence boundary detector MXTERMINATOR.

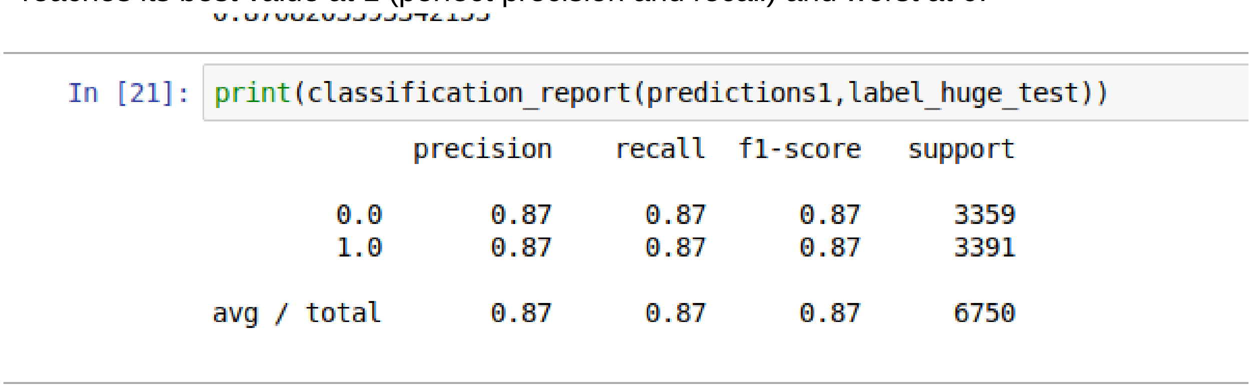
As this was a Real data used earlier in a research paper in the field so it's expected to be accessible. They can be obtained here.

- Second Part of the data I used Is imdb-movie-reviews-dataset provided by kaggle for the same task and it's recognized the same way at the first part which was very easy to merge between those data, this data consisits of 50,000 reviews divided equally into two parts (pos-neg) reviews files, so we can easily say why I decided to use this data also, just to enrich my models with more data to come with reasonable results, They can be obtained https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset.

Metrics

As I said before, we can easily see that this task is a classification task so The model prediction for this problem can be evaluated in several ways. Since I'm not only interested in precision or the recall but both, I think it's more appropriate to use f1-score so our model can keep both values in mind and not bias to just one. I'm also printing the whole classification report to keep an eye on all the values

The F_1 score is the harmonic average of the precision and recall, where an F_1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.



```
In [21]: print(classification_report(predictions1,label_huge_test))

             precision    recall  f1-score   support

        0.0       0.87      0.87      0.87      3359
        1.0       0.87      0.87      0.87      3391

avg / total       0.87      0.87      0.87      6750
```

As I've been advised in the proposal review to use roc_AUC metric, I also did but as expected the result was very close to f1 score as AUC is insensitive to the unbalanced data as AUC represents the actual area under the Receiver Operating Characteristic curve (ROC curve) and since our data is balanced and both the number of reviews for each class are equal so it won't affect that much.

The ROC curve is built by varying the discrimination threshold. For example, suppose that we have a binary classifier, trying to predict whether a certain input sample belongs to one category or another. Such prediction - let's call it *p*- can be thought as a float number between 0 and 1, representing the probability of that sample belonging to one class (while the probability of belonging to the other class would, naturally, be *1 - p*). The **discrimination threshold** is such value that tells at which level such prediction make the data sample belong to one class or the other. For example, if this threshold is 0.5, then if the predicted value *p* is greater than 0.5, the sample is assigned to one class, if below 0.5 to the other. By changing this threshold and by calculating and every time, it is possible to construct a Receiver Operating Characteristic Curve (ROC). The area under such curve is our metric AUC.

II. Analysis

Data Exploration

Dataset Size and Features

The merged dataset contains 52,000 reviews(50,000 for imdb dataset from kaggle - 2,000 for data introduced in polarity review paper).

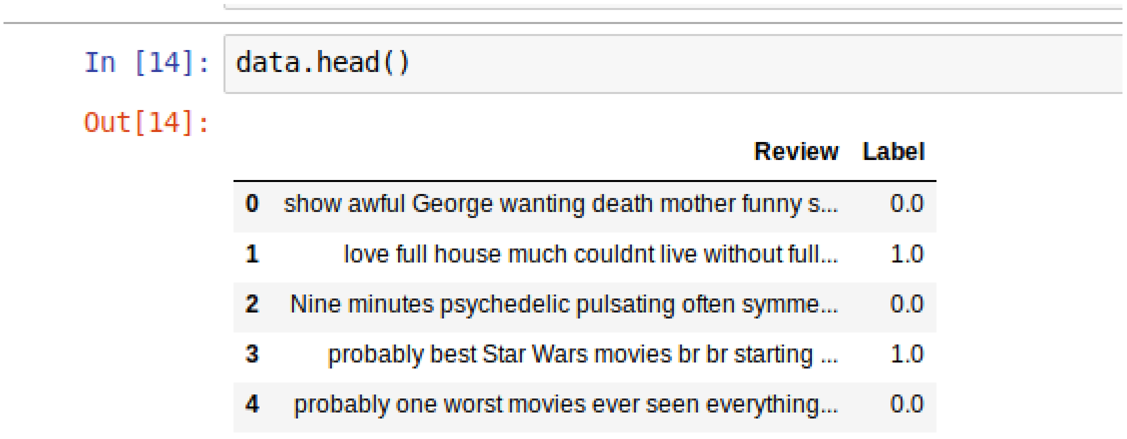
For each one of them, the only information we have is the review text body and which class it belongs to.

So for Text feature:

Given the purpose of my project,*Score* and *Text* are the only important features I am really interested in. In particular:

- Text represents my input variable, i.e. the variable I am going to use to predict the class and apply sentiment analysis;
- Score represents my dependent variable, i.e. the variable I am going to predict.

A sample rows of the reviews' data frame after the reviews being processed is reported below.



```
In [14]: data.head()
Out[14]:
```

	Review	Label
0	show awful George wanting death mother funny s...	0.0
1	love full house much couldnt live without full...	1.0
2	Nine minutes psychedelic pulsating often symme...	0.0
3	probably best Star Wars movies br br starting ...	1.0
4	probably one worst movies ever seen everything...	0.0

where the review with label value is equal to 1.0 is positive value and vice versa and we can sense that by reading the reviews like the third one "probably best ……." it's obvious that this review is positive so it makes sense that its label is 1.0.

The purpose of my model would be able to read such input (i.e. the text) and predict the score associated to this review.

Duplicates

As my data sources (kaggle – review polarity paper) that there's no duplicates in our data and each review is a unique so we don't need to bother ourselves that much in this point.

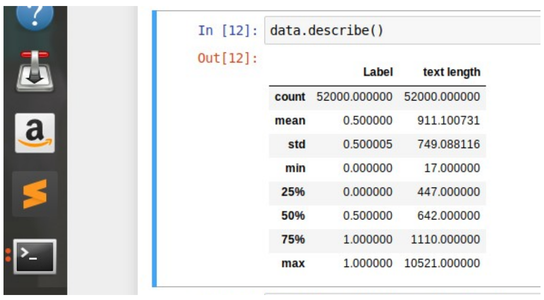
Exploratory Visualisation

Since, as said before, I will only be working with two original features (i.e. Text and Label), I have to see if I can performed exploratory visualization on these two.

Text

The first thing I looked at when analyzing reviews is their length. To keep it simple, I simply use Python built-in `len`-module which counts the **number of characters**. Here are some statistics:

- the **average** length of reviews is 911 characters;
- the **shortest** review is made of 17 characters;
- the **longest** review is made of 10521 characters.



```
In [12]: data.describe()
Out[12]:
```

	Label	text length
count	52000.000000	52000.000000
mean	0.500000	911.100731
std	0.500005	749.088116
min	0.000000	17.000000
25%	0.000000	447.000000
50%	0.500000	642.000000
75%	1.000000	1110.000000
max	1.000000	10521.000000

As shown in the two charts below(a box plot and a histogram), the distribution of reviews' length contains outliers.
The skewed distributions of reviews’ lengths, however, does not really affect my project. Clearly some users like to write longer reviews about the movie they saw.

Label

Label is the dependent variable I am trying to predict. Possible Labels assigned to a movie are either 0 or 1, with 1 being a positive review and 0 is a negative review. As I said before as the data are totally balanced and we know in advance before downloading it that we have equal number of reviews for each label so we can clearly say that we have 26,000 unique review for each one of the two labels we have here, so we don’t really need to spend more time in this section and let’s go to the next section.

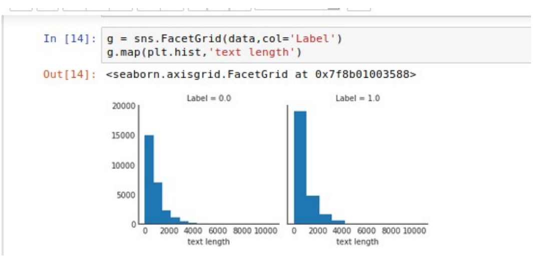
Reviews’ length vs Label

Here I am trying to understand if there is a somehow evident relationship between the length of a review (measured as number of characters) and its associated Label. If that is the case, then it may be a good choice to add length feature as an additional independent variable.

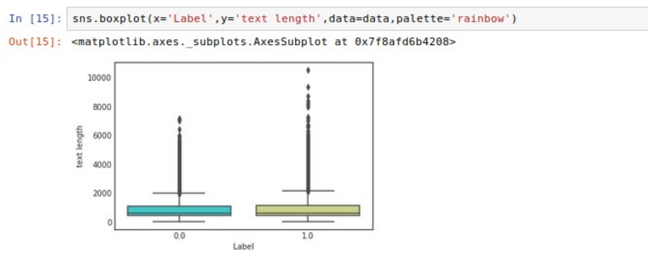
As from my experience sometimes the length of our text data gives us a great intuition like spam-ham mail problem, the bigger length we have, the more likely to be spam mail. So let’s see what I can do here.

A good way to visualize such relationship is to use a box plot, distribution plot where I show the distribution of reviews’ lengths by Label value.

Because of the outliers, it is difficult to see if there are any differences between reviews’ lengths when looking at data by Label value.
It seems that reviews’ length is pretty much similar. This implies that adding a feature like review’s length to my set of independent variables should not add too much value. So I will keep on using Text only.



```
In [14]: g = sns.FacetGrid(data,col='Label')
         g.map(plt.hist,'text length')
Out[14]: <seaborn.axisgrid.FacetGrid at 0x7f8b01003588>
```



```
In [15]: sns.boxplot(x='Label',y='text length',data=data,palette='rainbow')
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8afd6b4208>
```

III. Methodology

Data Preprocessing

When it comes to working with text data, a lot of data preprocessing is usually required. Let me cover this in the following paragraphs.

Text Parsing:

Text Parsing- at least in my own vocabulary-refers to the activity of "formatting"text data so that it is as "clean"as possible. In particular, the parsing options Envisioned in my project are

- removing any HTML text (e.g. HTMLtags)
- make everything lower case
- removing non-letter characters (e.g. numbers, punctuations, etc.)
- removing stop words (i.e. very common words like "and", "or", "I", "about",etc.)

I have then created a function called `text_process` that allows me to - in a kind of functional programming fashion - apply a list of the parsing functions above to a given review.

It is important to have the **flexibility and modularity** to choose which parsing function/s to apply because for some of the activities in this project I would only need a partial text pre-processing. For example, when I will create word vectors I would actually need **not to remove stop words** since they are useful in the defining the "context" around a certain word.
Please see below the example of a review as originally written (left) and after parsing (HTML removal, lower case, non-letter characters removal and stop words removal).
HTML tags have been removed, all text is in lower case, there is no punctuation and stop words (e.g. "These", "I", "they", etc.) have been removed.

Train, Validation and Test Split

An important part of my project is to split sentiment data into consistent train, validation and test sets. Only with a consistently-defined splitting I can truly evaluate the performance of various classification models.

In addition, I want my splitting to maintain the rational proportion of negative-positive reviews across all my data. as shown before, because data is balanced, it's very easy task as after I read my whole data from the files I shuffle it then split it with train_test_split function from sklearn

In the next two paragraphs (*Bag-of-words* and *Word-vectors*), while not going into all the details, I will present an intuition for the BOW and Word2vec models.

Bag-of-words

As reported by Wikipedia in a Bag-of-word model (or BOW) "*[...] a text (such as a sentence or a document) is represented as the bag (multi set) of its words, disregarding grammar and even word order but keeping multiplicity*". In other words, such model operates as following:

- Create a word vocabulary, i.e. a token (an integer typically) is assigned to every word (or to the*n* most frequent words) that appear in a set of documents (or *corpus*);
- Every document is then represented as a vector whose size is the same as the word vocabulary defined where, for every index corresponding to a particular word, the model returns the number of occurrences that word appears in the document itself.

For example, suppose our word vocabulary is the following:

Index	Word
1	i
2	am
3	udacious

Therefore, the model will transform the following reviews as reported in the table below.

OriginalText	Bag-of-wordrepresentation	Explanation
i am udacious	[1,1,1]	Each word in the vocabulary exactly appears one time
i am really udacious	[1,1,1]	Each word in the vocabulary exactly appears one time. Words not in the vocabulary are not counted
udacious udacious udacious	[0,0,3]	The word*udacious*appears three times
you love spaghetti	[0,0,0]	All the words in the original text are not indexed in the vocabulary

Let me comment on a couple of points.
First of all, "*i am good"*and "*i am really good"*are represented by the same BOW vector. This is because the word "*really*" is not in the word vocabulary. Clearly, however, the two sentences have slightly different meaning (with the second being more positive).
Secondly, if I were to change the order of words in the original text, my BOW vector representation would not change. BOW models do not retain any information about the ordering of words in a document.

Apart from the decision on whether to keep or remove stop words in my set of documents, there are a couple of options/parameters I will need to pay attention to. In particular, I am referring to:

- **word count methodology:** in the example above I have used a simple count of words in a document. A smarter - and typically more effective - way is to use the so-called tf-idf vectorization. Such methodology takes into account the frequency of a word in the entire corpus, thus making very frequent word "less important";
- **vocabulary size:**t his is a very important parameter. It is very difficult to have the right answer.Should the maximum size of my vocabulary be a certain percentage of the total number of unique words in my corpus? This is an important choice because having very long vectors may lead us into a problem of very high dimensional data, which, as a consequence, could bring us into the *curse of dimensionality's* trap. I have tested the performance of my classifier (in terms of AUC score, f1 score) on the validation set for different vocabulary sizes and I chose the level that gave me the best result;
- **word definition:** as shown before, documents such as "good" and"really good"are very likely to be represented by the same vector. It is possible, however, to define words as "range of words". For example, "really-good" can be considered as a word itself.

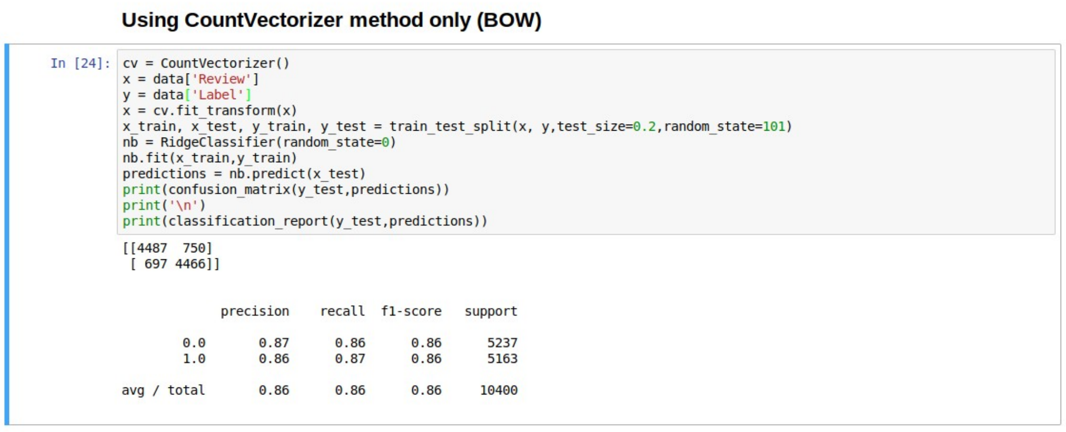
Word vectors

In this project I have used a Word2vec model with the help of gensim python library to represent my words in a vectorized way that carries semantic information. For a complete explanation of the model, originally developed by Tomas Mikolov,

Benchmark

I said before at the project proposal that I will use the results published in ``A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts'' as my data was the same data that published in this paper but since I added more data from kaggle to gain more intuition, using the results in the paper will not be accurate so I will not only use these results as my benchmark but also I will use BOW model as my bench mark and It make sense to me to use it as it’s the oldest technique between those techniques I use. I could’ve used more naive algorithms but I found using BOW to be my benchmark makes more sense.

BOW Benchmark



Using CountVectorizer method only (BOW)

```
In [24]: cv = CountVectorizer()
         x = data['Review']
         y = data['Label']
         x = cv.fit_transform(x)
         x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.2,random_state=101)
         nb = RidgeClassifier(random_state=0)
         nb.fit(x_train,y_train)
         predictions = nb.predict(x_test)
         print(confusion_matrix(y_test,predictions))
         print('\n')
         print(classification_report(y_test,predictions))
```

```
[[4487  750]
 [ 697 4466]]


             precision    recall  f1-score   support

        0.0       0.87      0.86      0.86      5237
        1.0       0.86      0.87      0.86      5163

avg / total       0.86      0.86      0.86     10400
```

To sum, up the BOW benchmark model consists of a simple classifier fitted on data which is been transformed using a BOW-Vectorizer.

Algorithms and Techniques:

The objective of my project is to evaluate whether the adoption and implementation of deep learning algorithms and techniques can improve the performance of my classifier, especially when attempting to retrieve “semantic” information from the reviews’ data.

As said at the beginning, I concentrated my efforts on two techniques:

- data representation using Tf-Idf vectorizer technique.
- a new text data representation using a word vectorization technique called Word2Vec.

Benchmark model

Please refer to the paragraph on benchmark models presented above.

Ridge on averages of Tf-IDF’s vectors

TF-IDF stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.
One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

In this option I am going to represent my text data using tf-idf after creating BOW for the data.

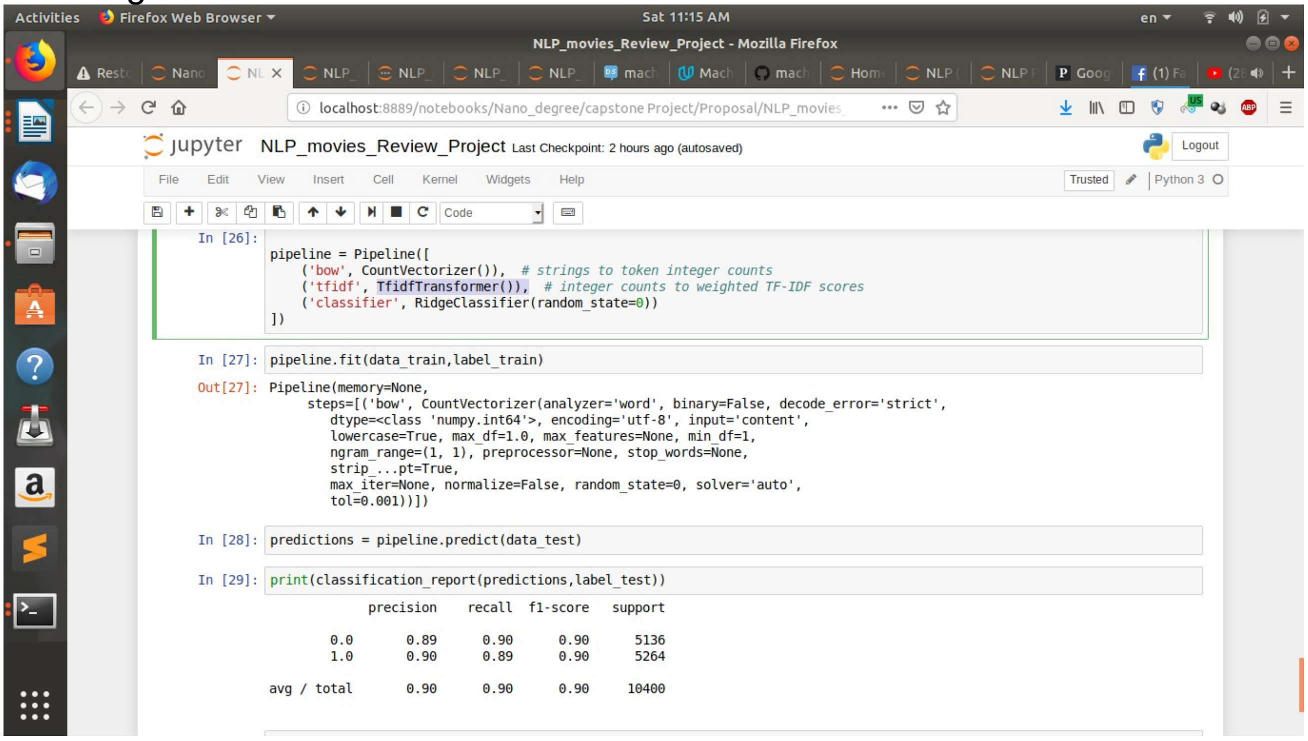
I am going to train my Tfidf model on my training reviews, Once I have my words defined as word vectors, I can “translate” a review into a sequence of word vectors. Then I’ll apply my classifier on the output of the previous step.

Ridge on averages of Word2vec’s vectors

In this option I am going to represent my text data using Word2vec. I find this algorithm to be extremely interesting. I am going to train my Word2vec model on all my reviews. The purpose here is in fact to have as much data as possible in order to learn an accurate representation of word vectors. I am using `gensim` library to train my Word2vec model.
Once I have my words defined as word vectors, I can “translate” a review into a sequence of word vectors. In particular, the review itself can be represented as a single vectors (with the same dimensionality of the word vectors) which is simply the average of all word vectors of the words within it. This approach is quite common and it has also been used in Kaggle’stutorial on NLP *Bag of Words Meets Bags of Popcorn*.

Implementation

Training Tf-IDF model


Activities
Firefox Web Browser
Sat 11:15 AM
NLP_movies_Review_Project - Mozilla Firefox
localhost:8889/notebooks/Nano_degree/capstone Project/Proposal/NLP_movies
jupyter NLP_movies_Review_Project Last Checkpoint: 2 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Code

```
In [26]:
pipeline = Pipeline([
    ('bow', CountVectorizer()),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', RidgeClassifier(random_state=0))
])

In [27]: pipeline.fit(data_train,label_train)

Out[27]: Pipeline(memory=None,
      steps=[('bow', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
         dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
         lowercase=True, max_df=1.0, max_features=None, min_df=1,
         ngram_range=(1, 1), preprocessor=None, stop_words=None,
         strip_...pt=True,
         max_iter=None, normalize=False, random_state=0, solver='auto',
         tol=0.001))])

In [28]: predictions = pipeline.predict(data_test)

In [29]: print(classification_report(predictions,label_test))

             precision    recall  f1-score   support

        0.0       0.89      0.90      0.90      5136
        1.0       0.90      0.89      0.90      5264

avg / total       0.90      0.90      0.90     10400
```

Training Word2vec model

This is probably one of the key steps in the *Implementation* section. I have used gensim Word2vec module in order to transform the words in my reviews into word vectors Reviews fed to the Word2vec model must first be split into sentences. This ensures that vectorial representations of words are only influenced by words around them in the **same sentence**. Splitting a paragraph into sentences is relatively using a punctuation tokeniser available in

gensim :
gensim.utils.simple_preprocess(file_string)
The most important parameters to take into account when training a Word2vec model in`gensim` are :

- **`size`**, .the dimensionality of the feature vectors. I used 100
- **`seed`**, i.e. for the seed for the random numbergenerator;
- **`sample`**, i.e. the threshold for configuring which higher-frequency words are randomly downsampled;
- **`negative`**, i.e. if > 0, negative sampling will be used. The integer value used in`negative`specifies how many “noise words” should be drawn (usually between 5-20). Default is 5. If set to 0, no negative sampling is used;
- **`window`**, i.e is the maximum distance between the current and predicted word within a sentence.

Let me further clarify some of these parameters below.

Word2vec’s parameter - sg
A Word2vec model can be trained using two different architectures:

- skip-gram;
- CBOW,or continuous bag ofwords.

The difference between the two is in the input data and what we are trying to predict.

In the **skip-gram** architecture, the input is a single word (a one-hot encoded vector for the word), and the output is represented by the a vector that contains the probabilities that a randomly chosen word in the vocabulary is "nearby" the input word. In other words, the task of the model is that of identifying words that are likely to be around a given word (i.e. being its context) .

In the **CBOW** architecture, on the contrary, the input to the model are the words around our target word (i.e. we feed the context) and we are trying to predict the most likely word that would fit that context.
I really like the dichotomy between the models as presented by Yanchuan Sim in a Quora's post

- in the skip-gram architecture, the task is that of **predicting the context given a word**;
- in the CBOW architecture, the task is that of **predicting the word given a context**.

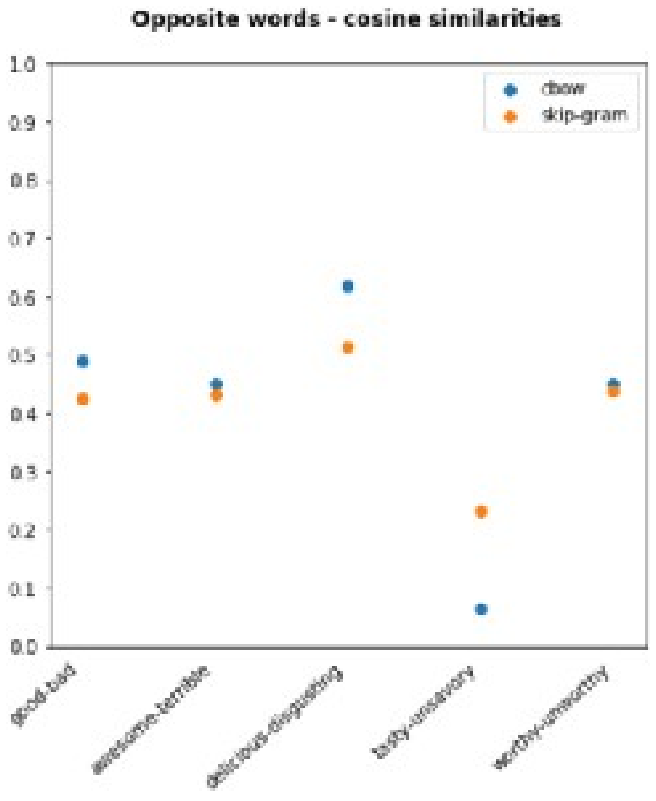
Word2vec’s parameter – sample
In word2vec models, it is possible to randomly down sample words according to their frequency. I will keep the default parameter (i.e. 1e-3, which represents the overall frequency threshold above which a word is considered **highly-frequent** and it is subject to down sample).
However, I would like to give a quick intuition about that. Consider the following three pairs of words:

- the dog
- the house
- the country

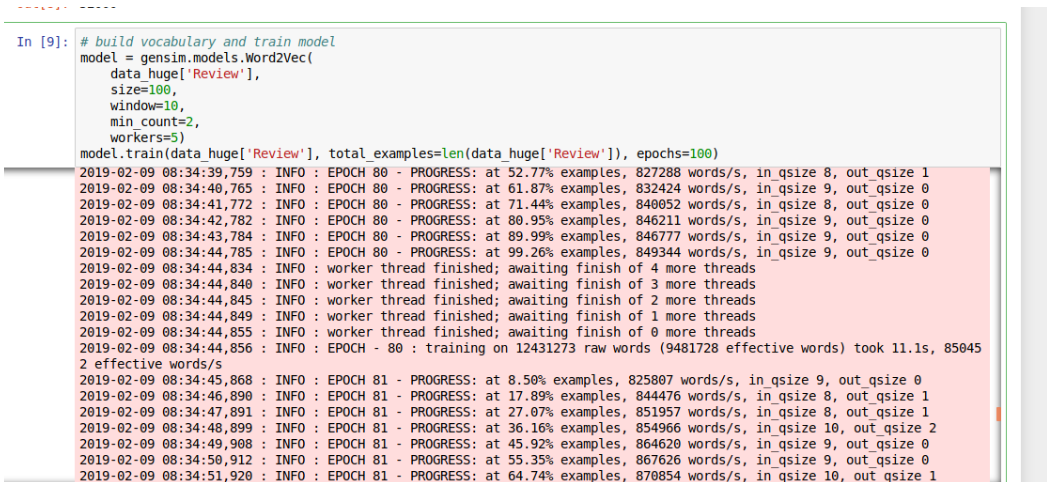
Assuming we are using a skip-gram architecture, the model will learn that the word “the” is very likely to be in the context of all the three words. This may lead to a result where the three words “dog”, “house" and “country" are represented by similar word vectors. In reality, however, we know that such words have a very different meaning. The fact that “the" is in their context, is purely a result given by the fact that “the” is an article and, well, it can be everywhere!
Through sub-sampling the model will randomly remove high-frequency words from the context and let the model:

- be more accurate from a semantic perspective;
- be faster,since we are passing fewer samples.

Word2vec’s parameters - negative
This is one of the coolest things in Word2vec. Typically, the size of a vocabulary will be very large. This means that we will likely have a lot of weights to train in the model and back-propagating the error can be a gigantic task.
Negative sampling solves this problem by letting the model update the weights for only a fraction of negative cases (i.e. the words that should not be in the context, if we are using a skip- gram model).
Conceptually this is similar to **batching** in the sense we are doing small updates a lot of times, rather than a big update a few times.
As you can see from the chart below, the skip-gram model over-performs (i.e. lower cosine similarity between opposite words) the CBOW model in most of the cases. For the “tasty-unsavoury” couple of opposite words, the CBOW model is actually better.



Overall, I have decided to prefer the skip-gram architecture, and those’re my model parameters:

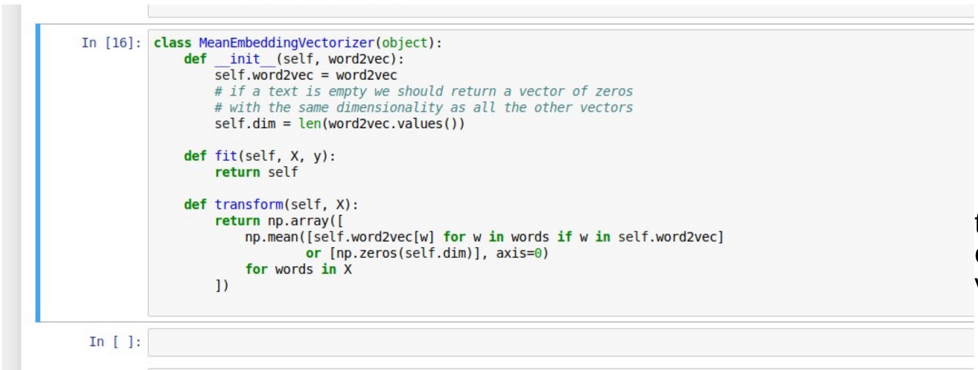


```
In [9]: # build vocabulary and train model
        model = gensim.models.Word2Vec(
            data_huge['Review'],
            size=100,
            window=10,
            min_count=2,
            workers=5)
        model.train(data_huge['Review'], total_examples=len(data_huge['Review']), epochs=100)
```

```
2019-02-09 08:34:39,759 : INFO : EPOCH 80 - PROGRESS: at 52.77% examples, 827288 words/s, in_qsize 8, out_qsize 1
2019-02-09 08:34:40,765 : INFO : EPOCH 80 - PROGRESS: at 61.87% examples, 832424 words/s, in_qsize 9, out_qsize 0
2019-02-09 08:34:41,772 : INFO : EPOCH 80 - PROGRESS: at 71.44% examples, 840052 words/s, in_qsize 8, out_qsize 0
2019-02-09 08:34:42,782 : INFO : EPOCH 80 - PROGRESS: at 80.95% examples, 846211 words/s, in_qsize 9, out_qsize 0
2019-02-09 08:34:43,784 : INFO : EPOCH 80 - PROGRESS: at 89.99% examples, 846777 words/s, in_qsize 9, out_qsize 0
2019-02-09 08:34:44,785 : INFO : EPOCH 80 - PROGRESS: at 99.26% examples, 849344 words/s, in_qsize 9, out_qsize 0
2019-02-09 08:34:44,834 : INFO : worker thread finished; awaiting finish of 4 more threads
2019-02-09 08:34:44,840 : INFO : worker thread finished; awaiting finish of 3 more threads
2019-02-09 08:34:44,845 : INFO : worker thread finished; awaiting finish of 2 more threads
2019-02-09 08:34:44,849 : INFO : worker thread finished; awaiting finish of 1 more threads
2019-02-09 08:34:44,855 : INFO : worker thread finished; awaiting finish of 0 more threads
2019-02-09 08:34:44,856 : INFO : EPOCH - 80 : training on 12431273 raw words (9481728 effective words) took 11.1s, 85045
2 effective words/s
2019-02-09 08:34:45,868 : INFO : EPOCH 81 - PROGRESS: at 8.50% examples, 825807 words/s, in_qsize 9, out_qsize 0
2019-02-09 08:34:46,890 : INFO : EPOCH 81 - PROGRESS: at 17.89% examples, 844476 words/s, in_qsize 8, out_qsize 1
2019-02-09 08:34:47,891 : INFO : EPOCH 81 - PROGRESS: at 27.07% examples, 851957 words/s, in_qsize 8, out_qsize 1
2019-02-09 08:34:48,899 : INFO : EPOCH 81 - PROGRESS: at 36.16% examples, 854966 words/s, in_qsize 10, out_qsize 2
2019-02-09 08:34:49,908 : INFO : EPOCH 81 - PROGRESS: at 45.92% examples, 864620 words/s, in_qsize 9, out_qsize 0
2019-02-09 08:34:50,912 : INFO : EPOCH 81 - PROGRESS: at 55.35% examples, 867626 words/s, in_qsize 9, out_qsize 0
2019-02-09 08:34:51,920 : INFO : EPOCH 81 - PROGRESS: at 64.74% examples, 870854 words/s, in_qsize 10, out_qsize 1
```

Fitting Ridge Regression model on averages of Word2vec’s vectors

Now that I have my word vectors, I can train my first “enhanced” model.
As explained before, for every review, I will initialize a zero vector with the same dimensionality I used to represent my word vectors.



```
In [16]: class MeanEmbeddingVectorizer(object):
             def __init__(self, word2vec):
                 self.word2vec = word2vec
                 # if a text is empty we should return a vector of zeros
                 # with the same dimensionality as all the other vectors
                 self.dim = len(word2vec.values())

             def fit(self, X, y):
                 return self

             def transform(self, X):
                 return np.array([
                     np.mean([self.word2vec[w] for w in words if w in self.word2vec]
                             or [np.zeros(self.dim)], axis=0)
                     for words in X
                 ])

In [ ]:
```

For every word in the review, then, I will add its vector representation to the review vector, Finally, I will take an average of the word vectors).

My new average vectors represent the input data I will feed into the same Ridge classifier architecture defined in the benchmark model. The intuition is that the vector representation of words and - through averaging - reviews should be able to better capture semantic nuances which are not measured by count-based models like BOW.

---

results

The table below reports the AUC , f1-score on the validation set for all the models trained.

Models	Validation accuracy f1-score	AUC-score
BOW benchmark	86%	86.44
Ridge - TFIDF	90%	0.896
Ridge - Word2Vec(CBOW)	88%	0.879
Ridge - Word2Vec(Skip-Gram)	88%	0.880
Ridge - Word2vec(glove.6B.200.txt)	84%	0.835
Ridge – Word2vec(glove.6B.200.txt) then trained it with our reviews text data	88%	0.884

The table above offers numerous insights.

As I wasn't expected and for me it's the best thing I gained from this project is that it's not always the case that the newer techniques always better, and every thing has it's own constraints,

as it was really surprising to see TF-IDF can beat newer techniques like word2vec and I didn't understand why that might happen? Then after done many researching I figured out that the size of the training corpus affect very much word2vec model so to get the most from it we should train it with very large corpus due to the huge number of parameters we have and since 52,000 reviews aren't that much so in this case TF_IDF did better, however using pretrained word2vec model and further training it with our own corpus decrease the gap between it and TF-IDF to just 1%

and I believe If I did use bigger model like google word2vec model, I'm sure I could've beat Tf-IDF but this will cost me much memory and unfortunately my laptop won't be able to do it.

https://datascience.stackexchange.com/questions/19160/why-word2vec-performs-much-worst-than-both-countvectorizer-and-tfidfvectorizer

in the models using pre-trained Word2vec word vectors, the depth of such vectors correspond to the one used while the training the Word2vec model (i.e. in this case this value corresponds to 200).
That is why models that learn their own word vector representations have many more parameters to train.

Refinement

I have spent quite a long time testing different configurations of my models. In general, the two main items I have played with are:

- word vectors representations;
- tuning hyper-parameters of the models;

With regards to **word vectors representation**, I have tested three possible options, i.e.:

- Word2vec embeddings using the model I have trained before;
- Glove.6B.200.txt
- Glove.6B.200.txt then training it more with our own data just to tune its parameter.

For the last case This is somewhat similar to **transfer learning**. In other words, word vector representation is part of the forward/back propagation loop in such a way that even my word vectors are further trained in order to reduce the loss of my sentiment classifier.

In addition to that, I have also experimented with other parameters, in particular:

- sg = 0 or 1 indicated using Skip-Gram or CBOW
- number of epochs

In general, I have seen that the following changes contributed to an improvement in the performance of my classifier (measured as AUC on the validation set):

- **continue training** the word vector embeddings led to a higher AUC on the validation set;

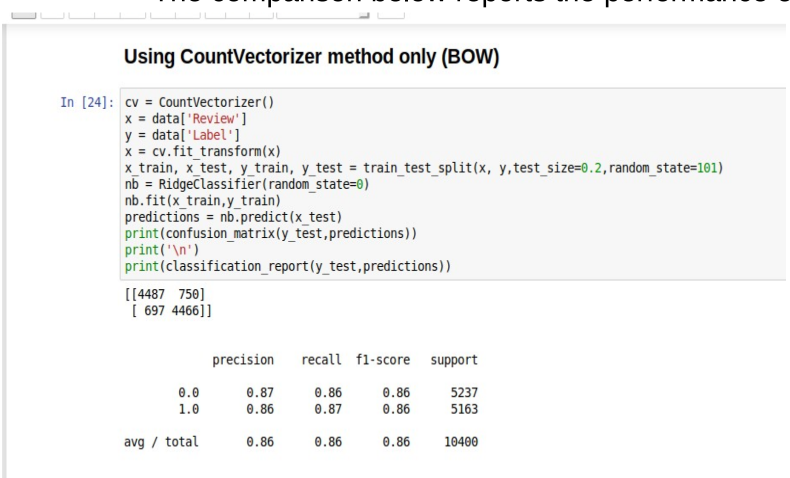
The performance of the refined model on the validation set against the benchmark model is reported below:

IV. Results

Model Evaluation and Validation

The comparison below reports the performance of the benchmark models and the refined models on the test set.

As I described above in Preliminary results , the enhanced glove model still performs better than the benchmark model.



Using CountVectorizer method only (BOW)

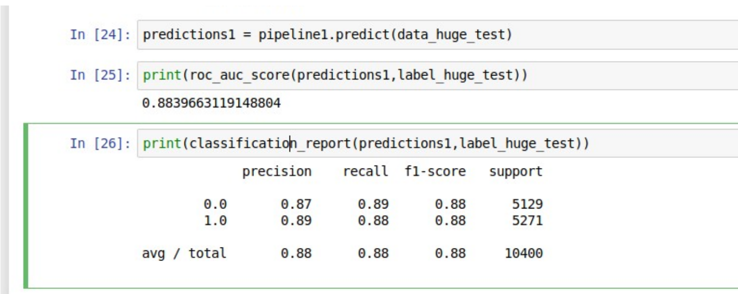
```
In [24]: cv = CountVectorizer()
         x = data['Review']
         y = data['Label']
         x = cv.fit_transform(x)
         x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.2,random_state=101)
         nb = RidgeClassifier(random_state=0)
         nb.fit(x_train,y_train)
         predictions = nb.predict(x_test)
         print(confusion_matrix(y_test,predictions))
         print('\n')
         print(classification_report(y_test,predictions))

[[4487  750]
 [ 697 4466]]


             precision    recall  f1-score   support

        0.0       0.87      0.86      0.86      5237
        1.0       0.86      0.87      0.86      5163

avg / total       0.86      0.86      0.86     10400
```



```
In [24]: predictions1 = pipeline1.predict(data_huge_test)

In [25]: print(roc_auc_score(predictions1,label_huge_test))
         0.8839663119148804

In [26]: print(classification_report(predictions1,label_huge_test))
                      precision    recall  f1-score   support

                 0.0       0.87      0.89      0.88      5129
                 1.0       0.89      0.88      0.88      5271

         avg / total       0.88      0.88      0.88     10400
```

The purpose of the project was to build a model that was capable of capturing semantic nuances. The benchmark model,based on bag-of-words data works pretty well, but should not be able to capture the importance of word ordering.

Justification:

The main differences between my benchmark model and my enhanced models are the word representation (i.e. bag of words vs word vectors) i.e. the word vectors, allows for semantic-like representation of my words. In addition, input data is passed as a sequence of word, thus maintaining the ordering of words themselves. while the AUC of the word vector model is higher, so now the word vector can't only be affected by the word appearance frequent but also with the words in its context so our model now can capture more informations like the similarity and semantic and syntactic information so it makes more sense to see word vectors beats BOW.

Conclusion

Free-Form Visualization

Throughout this report, I have presented a series of visualizations, covering exploratory data analysis, word vectors and performance evaluation.

The visualization before is my coded diagram that represents my final model. This is in line with a previous coded diagram I made in the proposal submission document.

Reflection

The goal of this project is to evaluate the performance of a word embedding-based classifier in predicting the sentiment (measured indirectly as movies' labeled) of a given reviews on movies' reviews data that part of it are published before on here and the rest of the data published on kaggle here

During the project, I have done the following:

- Understand the nature of my dataset (exploratory data analysis);
- Format and parse text data;
- Process text data, either in a bag-of-words , TF-IDF format or as word vectors;
- Identify a metric to evaluate the performance of any model;
- Fit a benchmark model;

If I had to choose two particular aspects of the project that I found interesting and difficult to deal with and accept it , I would choose the following:

- newer techniques not always the right choice just because it's newer, you should fulfill its constraints first to get the best from it
- In a better world, it's fine to think that when we say (base algorithm) you'll have very bad results but unfortunately it's not that easy! As in our project here BOW got 86% ! and if we look at it in a nut shell it's not bad result at all ! But we should know that the more knowledge we learn about our problem the deeper look we will have, as however this optimistic result we should know that this base model won't perform well in real problems as it throws away many information about our data like word ordering.

Improvement

There are some aspects of the implementation that could be improved and potentially may lead to improved performance. Some of them were actually already tested.

First of all, instead of using my Word2vec embeddings (i.e. pre-trained word vectors from the Word2vec model), I could use pre-trained embeddings trained on larger corpus. In my project I have actually used glove.6B.200.txt pre-trained model and checked it twice, once when I used it as I downloaded, second wa when I further trained it with my reviews data but that didn't necessarily achieve the highest performance however its results was very good especially the last one. In general, it would be very easy to implement such change.

Secondly - I think using deep learning models will be satisfying like using RNN or implementing Word2vec with CNN in like using embedding layer in keras. And would give higher results.

Appendix

Paragraph-to-notebook mapping

Paragraph	Notebook
I. Definition	-
II. Analysis	NLP_movies_Review_Project_BOW_tfidf_part.ipynb
III. Methodology - Data preprocessing	NLP_movies_Review_Project_BOW_tfidf_part.ipynb
III. Methodology – Benchmark – TF-IDF model	NLP_movies_Review_Project_BOW_tfidf_part.ipynb
III. Methodology - Implementation - Training Word2vec model	NLP_movies_Review_Project-Word2Vec-main-part.ipynb
III. Methodology - Implementation - Fitting Ridge classifier model on average of word2vec's vectors (CBOW) - using glove pre-trained model in a nut shell	NLP_movies_Review_Project-Word2Vec-main-part.ipynb
III. Methodology - Implementation - Fitting Ridge classifier model on average of word2vec's vectors (Skip-Gram)	NLP_movies_Review_Project-Word2Vec part-skip_gram.ipynb
III. Methodology - Implementation - using glove pre-trained model and train it further with reviews data to get more results	NLP_movies_Review_Project-Word2Vec part-enhanced_glove.ipynb