Project 2

1. The Pipeline for Generating Bag-of-Words Feature

In the given training dataset, there are 2400 samples, including three websites: Amazon, IMDB, and Yelp, each of them has 800 samples. The testing dataset has 600 samples and each of the websites has 200 samples. In training samples, there are 33046 words and 5268 of them are unique. Our goal is to analyze these sentences and classify the corresponding sentiments: positive or negative. To achieve this goal, we should convert these words to a form that computer can understand. The pipeline is described as below:

(a) Convert all characters to lowercase
(b) Remove all the numbers in the sentence (number contributes little meaning to classifying sentiment)
(c) Remove stop words in English (stop words are commonly used words like 'a', 'the' etc., which have little contribution to the sentiment. The library NLTK has a corpus of stop words). However, we should keep the negative words like 'not', 'couldn't', 'don't' etc. because they are critical to sentiment.
(d) Remove the punctation
(e) Tokenize (`nltk.word_tokenize`): separate the sentence into words and put them back to the list
(f) Use Porter Stemmer (`nltk.stem`) to remove morphological affixes from words and leave only the word stem
(g) Check the frequency of each word and remove the low-frequency words (the words appear only once). Low-frequency words may have spelling errors.

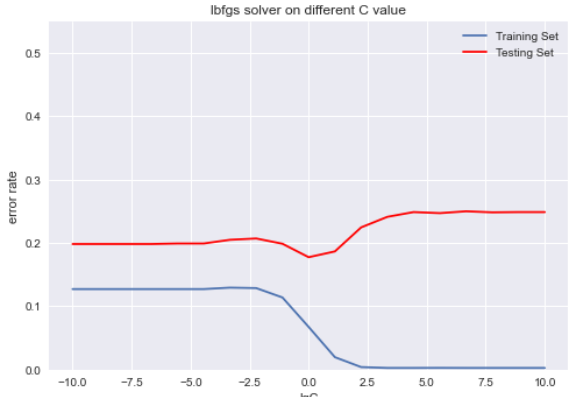
When creating the feature representation, instead of simply counting the occurrences of the words, I use TF-IDF (Term Frequency- Inverse Document Frequency) to represent the words. TF is counting the frequency of the words, and IDF can help to reduce the effect of useless high-frequency words. Combing TF and IDF can be more effective at representing the relevance of words compared with using TF or IDF independently. In Scikit Learn, the method `feature_extraction.text.TfidfVectorizer` can achieve this. Finally, the 3000 reviews including training and testing dataset convert into a 3000*1835 matrix.

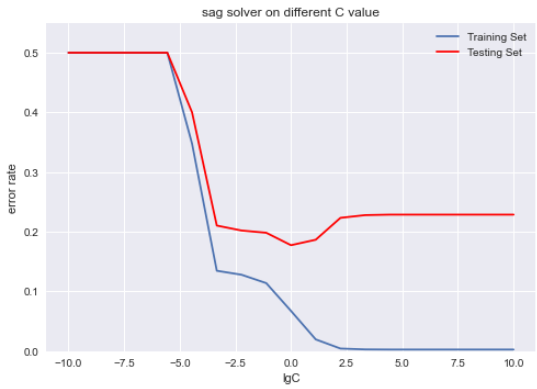
2. Logistic Regression

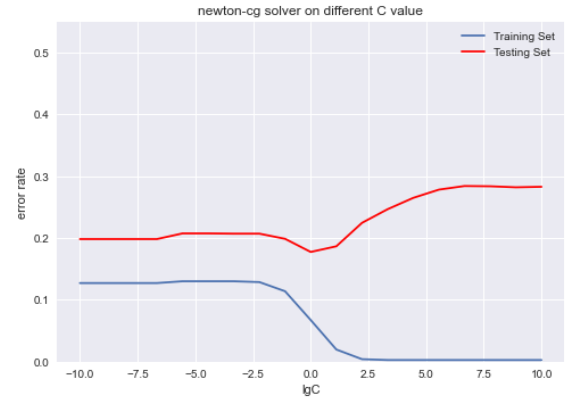
Since the given datasets only include x_train, y_train, x_test but without y_test, in the following models, I use 5-fold cross validation to fit the model to avoid overfitting and get a higher accuracy score and error rate for each model.

In the scikit learn Logistic Regression classifier, I choose the hyperparameters: penalty C and solver. The penalty I choose is L2, and I will compare the solvers of 'newton-cg', 'sag' and 'lbfgs' because they support L2 regularization. I set penalty C value in log space(-10, 10) using `numpy.logspace(-10, 10, 19).` I set 5-fold cross validation using `cross_validate` in the model and calculate the average score. Error rate equal to 1 minus score. Then, I plot the figures as below.

It's clear to see the how training set becoming overfitting as the penalty increasing in these three different solvers.'lbfgs' and 'newton-cg' solvers share similar pattern, but 'lbfgs' have better performance. As for 'sag' solver, both training and testing dataset start from poor performance when penalty is small and the error rate decreases as the penalty increasing. Although I use different solvers, the models always hit the best performance when lgC=0, that is C=1. Finally, I use grid search and find out the best hyperparameter combination for logistic regression model is C=1, solver='sag', and the best score is 0.8225.





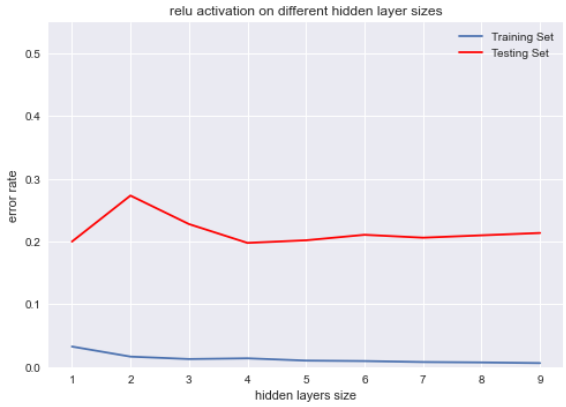


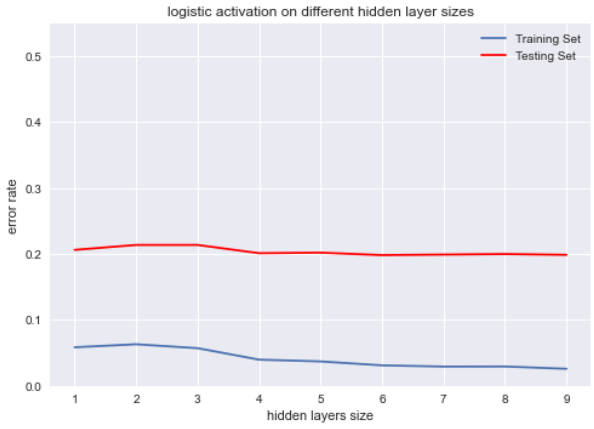
3. Neural Network (MLP)

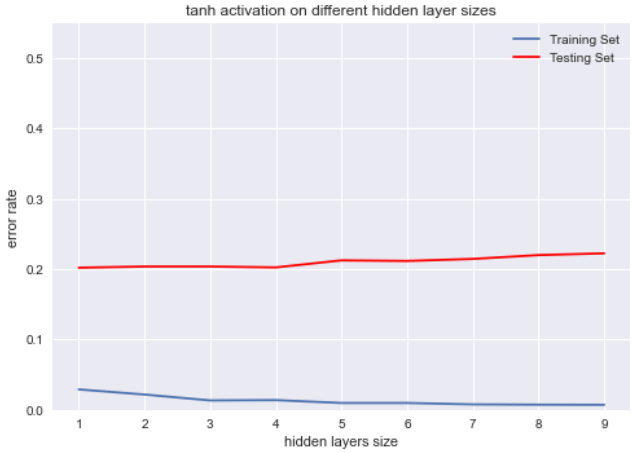
In MLP model, I am going to choose the activation function and the size of hidden layer as my research hyperparameters, and the other parameters will stay in default setting in scikit learn MLP model.

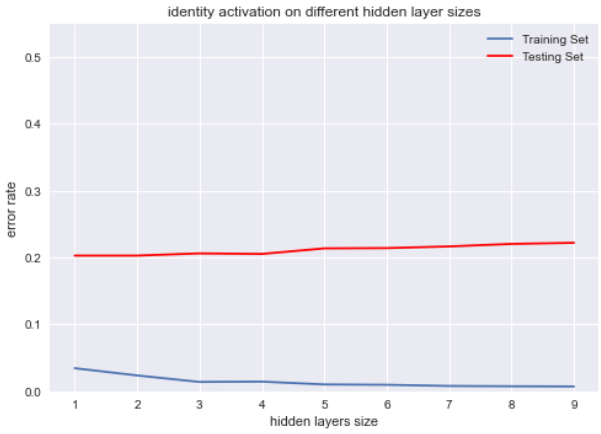
There are four activation functions in MLP module: 'identity' (returns f(x) = x), 'logistic' (returns f(x) = 1 / (1 + exp(-x)).), 'tanh' (returns f(x) = tanh(x)), 'relu': (returns f(x) = max (0, x)). I will compare these four activation functions to check which function has the best performance. There is no certain function to calculate the best hidden layer size in neural network and it is worth seeing how different hidden layer sizes affect the model performance. However, limited by the computing power of my own laptop, I test the hidden layer size from 1 to 10 in case long time waiting.

From these figures, the error rate lines are stable in different hidden layer sizes in these four different activation functions. There is a slight overfitting problem when hidden layer size increases. It is interesting to see there is little difference in error rate in different activation functions, except for the testing dataset using 'relu' when hidden layer size is 2. As in the previous step, I use grid search method and find that the best parameter combination is logistic activation function and 7 hidden layers. The final best score is 0.8025.





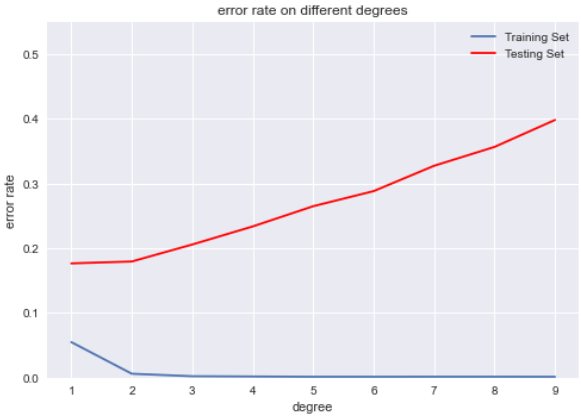


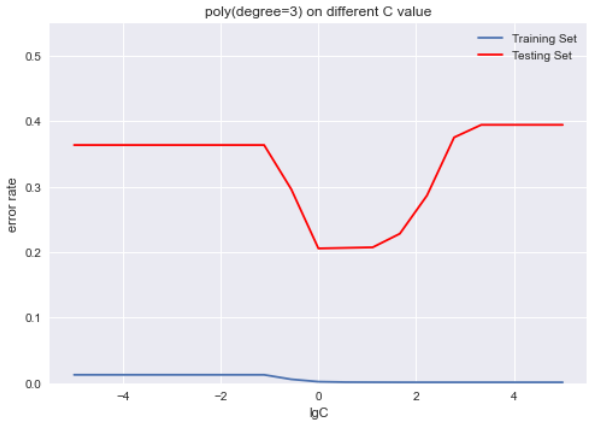


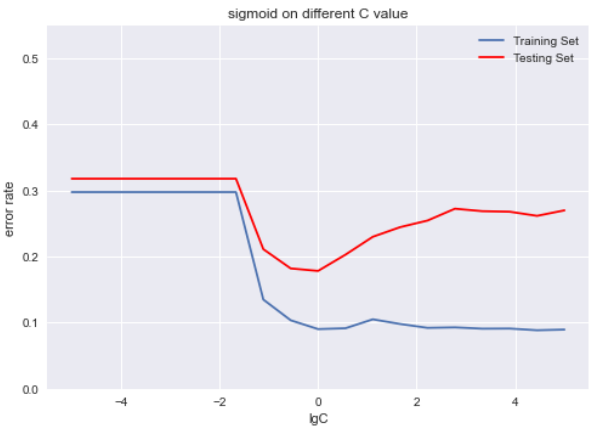
4. SVM

For Support Vector Machine model, I will focus on two hyperparameters: penalty C and kernel function. There are four candidate kernels: 'linear', 'poly', 'rbf', and 'sigmoid'. The penalty C will be set between 10^{-5} to 10^{5} using the code `numpy.logspace(-5, 5, 19)`. From these figures, we can see SVM model has serious overfitting problem when penalty increases in training dataset. Almost all of the testing set achieved the lowest error when C=1 except using 'rbf' kernel (C=3.6, error rate=0.1688).

When using 'poly' kernel, the default degree setting is 3. Thus, I continue to find the best degree from 1 to 10 using penalty C=1 and plot the figure. The testing set has better performance when degree is 1 (score: 0.8233) or 2 (score: 0.8204), then the error rate increases with increasing degrees. After using grid search, it turns out that when C=3.6, using 'rbf' kernel function has the best score: 0.8313.







5. Summarize & Hypothesis

The three classifiers I use are Logistic Regression (score: 0.8225), MLP (score: 0.8025), SVM (score: 0.8313). The best performance classifier is SVM while the difference of score between SVM and Logistic Regression is only 0.0088. Neural Network model does not perform as well as machine learning algorithms even though MLP has more flexibility. I think the reason for this is the problem we are dealing with is binary classification, Logistic Regression is born for binary classification problem and SVM can also achieve multi separate in high dimension space. Also, when we transform the words into matrix, the matrix is sparse. When it feed into a neural network model, most neurons are zero and the model may learn nothing.

Since the given dataset only includes x_train, y_train, and x_test. If I want to see the details of the wrong classified samples, I have to use the x_train and y_train and do the split again. Thus, I choose train test splitting (6:4), use training set to feed SVM model, and predict the testing set. There are 165 samples get wrong. I look into the source of these wrong classified samples, there are 47 from Amazon, 55 from IMDB, 63 from Yelp. It seems that Yelp has a higher number of wrong samples. In IMDB reviews, the expressions are not direct and include some descriptions that contain not common words. It is hard for the model to learn these expressions. What's more, if a sentence has positive and negative words at the same, for example, "*it was a pale color instead of nice and char and has no flavor*", it seems hard for machine to understand and make a wrong prediction.

After submitting to the leaderboard, the error rate is 0.1767 and AUCROC is 0.9087. The result is close to the score cross validation that I use as expected. Thus, using cross validation is an important way to avoid overfitting and inaccuracy from picking certain parts of data.