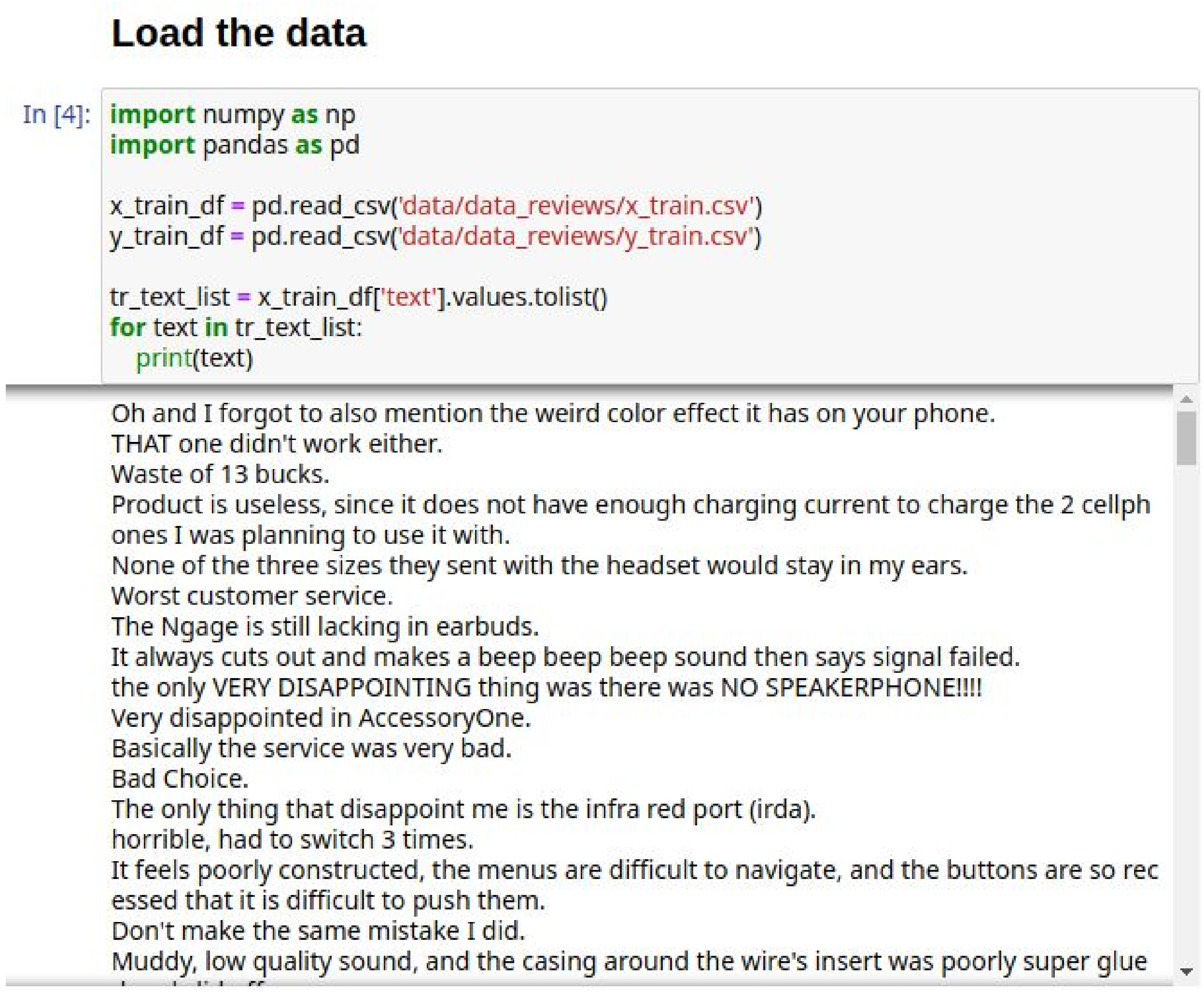
Project 01 Jiawei Wang 4/27/2020

Part One: Classifying Review Sentiment with Bad-of-WOrds Features

---

1.1 Data Preprocessing:

The provided dataset was loaded onto Jupyter Notebook. When preprocessing, Natural Language ToolKit (nltk) was used. But before diving into this procedure, we need to get a general view of the data:



Load the data

In [4]:
```
import numpy as np
import pandas as pd

x_train_df = pd.read_csv('data/data_reviews/x_train.csv')
y_train_df = pd.read_csv('data/data_reviews/y_train.csv')

tr_text_list = x_train_df['text'].values.tolist()
for text in tr_text_list:
    print(text)
```

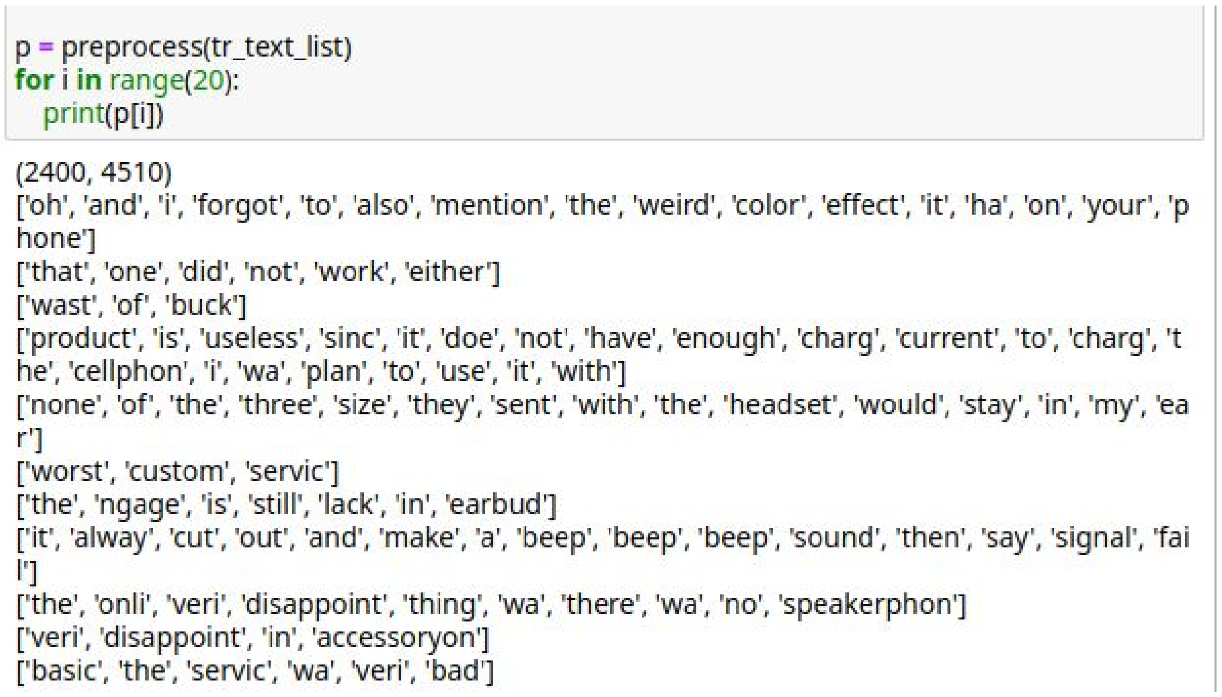
```
Oh and I forgot to also mention the weird color effect it has on your phone.
THAT one didn't work either.
Waste of 13 bucks.
Product is useless, since it does not have enough charging current to charge the 2 cellph
ones I was planning to use it with.
None of the three sizes they sent with the headset would stay in my ears.
Worst customer service.
The Ngage is still lacking in earbuds.
It always cuts out and makes a beep beep beep sound then says signal failed.
the only VERY DISAPPOINTING thing was there was NO SPEAKERPHONE!!!!
Very disappointed in AccessoryOne.
Basically the service was very bad.
Bad Choice.
The only thing that disappoint me is the infra red port (irda).
horrible, had to switch 3 times.
It feels poorly constructed, the menus are difficult to navigate, and the buttons are so rec
essed that it is difficult to push them.
Don't make the same mistake I did.
Muddy, low quality sound, and the casing around the wire's insert was poorly super glue
```

As we can see, there are many obstacles that are hard for nltk to work on, such as grammar mistakes, punctuations, slang vocabularies, mix of upper- and lower-case, etc.

So we want to do the following steps in order to get a ‘pure’ dataset that we can use for analysis.
The steps for preprocessing data are:

1. Turn all sentences to lowercase
2. Delete apostrophes
3. Delete punctuations
4. Remove stop words
5. Lemmatization

And for each step, the data was collected from the previous step. Things that worth mention: 1.For the apostrophes, I used the Apostrophes Word List from spellzone.com (https://www.spellzone.com/word_lists/list-29383.htm). 2.nltk stopword dictionary was used to remove stop words. 3.I used lemmatization to transform sentences into the very basic form.



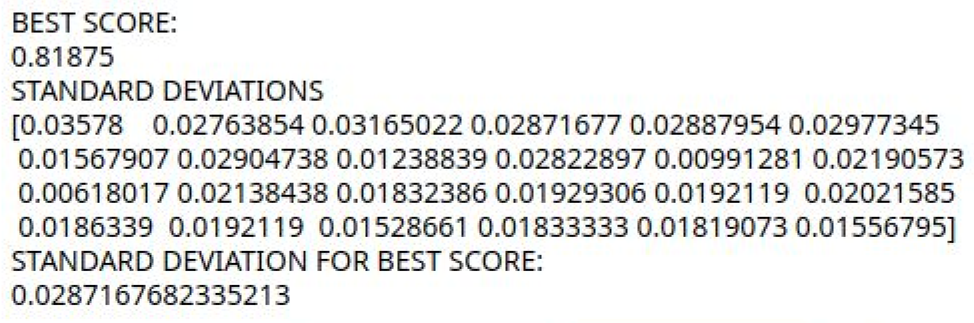
```
p = preprocess(tr_text_list)
for i in range(20):
    print(p[i])
```

```
(2400, 4510)
['oh', 'and', 'i', 'forgot', 'to', 'also', 'mention', 'the', 'weird', 'color', 'effect', 'it', 'ha', 'on', 'your', 'p
hone']
['that', 'one', 'did', 'not', 'work', 'either']
['wast', 'of', 'buck']
['product', 'is', 'useless', 'sinc', 'it', 'doe', 'not', 'have', 'enough', 'charg', 'current', 'to', 'charg', 't
he', 'cellphon', 'i', 'wa', 'plan', 'to', 'use', 'it', 'with']
['none', 'of', 'the', 'three', 'size', 'they', 'sent', 'with', 'the', 'headset', 'would', 'stay', 'in', 'my', 'ea
r']
['worst', 'custom', 'servic']
['the', 'ngage', 'is', 'still', 'lack', 'in', 'earbud']
['it', 'alway', 'cut', 'out', 'and', 'make', 'a', 'beep', 'beep', 'beep', 'sound', 'then', 'say', 'signal', 'fai
l']
['the', 'onli', 'veri', 'disappoint', 'thing', 'wa', 'there', 'wa', 'no', 'speakerphon']
['veri', 'disappoint', 'in', 'accessoryon']
['basic', 'the', 'servic', 'wa', 'veri', 'bad']
```

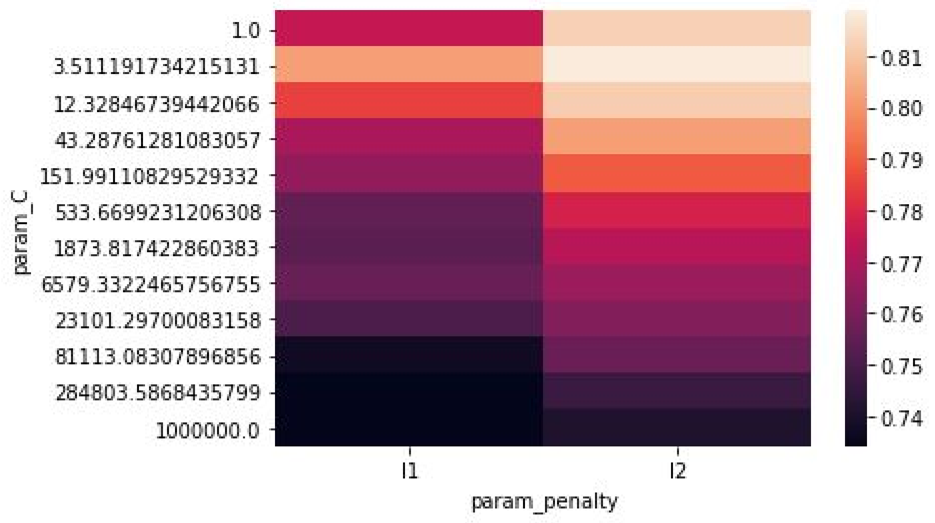
As we can see, every sentence is now presented as a list with no spacing. Next, we need to vectorize the data. For this part, sklearn.feature_extraction.text was used (TfidfTransformer and TfidfVectorizer). Before vectorization, there were 4510 unique words and the number decreased to 3414 after vectorization. All words were kept including common words and rare words. However the words are not ordered as we do not know which part of the dataset were they from. As we have discussed in previous classes, when dealing with large amounts of words, number of the occurrence should be used as weight as different words do not always contribute the same to the result. By using TfidfTransformer and TfidfVectorizer, frequency of words was considered for feature vector and common words gained penalties against rare words. Only single features were used. The method used for this part is TF-IDF Algorithm.

1.2 Logistic Regression Model for Sentiment Analysis

The default Logistic Regression Model from scikit-learn with ‘liblinear’ solver was used since solver ‘lbfgs’ supports only 'l2' or 'none' penalties but I wanted to try ‘l1’ penalty again ‘l2’. Two hyperparameters were used as: the inverse penalty C (C=C) and regularization (penalty=penalty). For C, I used np.logspace(0, 6, 12) and 12 different samples were generated. For penalty, both l1 and l2 regularization were used as a measurement of overfitting. Cross-Validation was used from sklearn.model_selection.GridSearchCV, with 5 folds and 0 verbosity. The model ran till it reached convergence and the result is as follows:



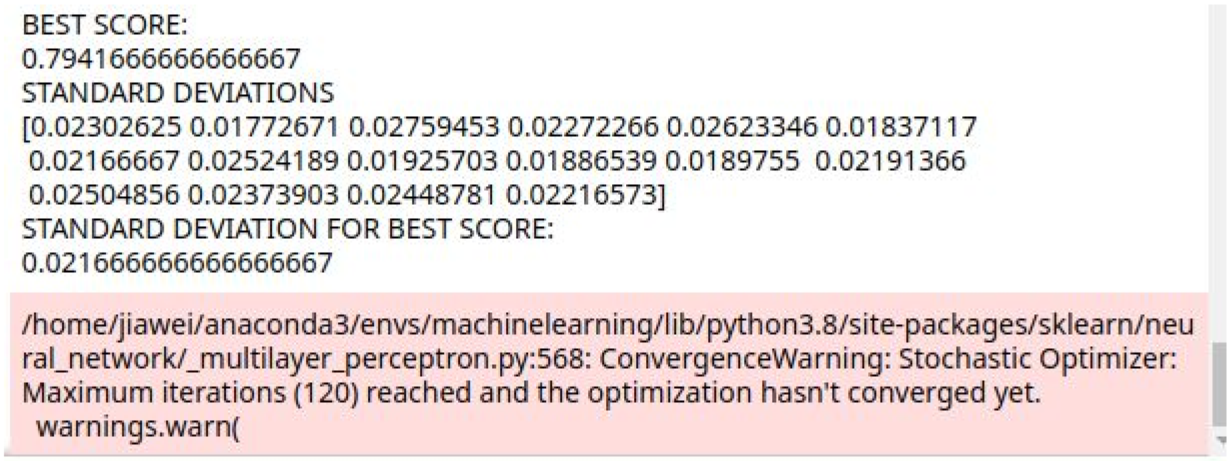
```
BEST SCORE:
0.81875
STANDARD DEVIATIONS
[0.03578    0.02763854 0.03165022 0.02871677 0.02887954 0.02977345
 0.01567907 0.02904738 0.01238839 0.02822897 0.00991281 0.02190573
 0.00618017 0.02138438 0.01832386 0.01929306 0.0192119  0.02021585
 0.0186339  0.0192119  0.01528661 0.01833333 0.01819073 0.01556795]
STANDARD DEVIATION FOR BEST SCORE:
0.0287167682335213
```



As this graph suggests, for C, both smaller and larger values carry bigger losses in accuracy, and this is because C is used to reduce overfiiting by applying penalty to parameter values. For regularizations, l1 and l2 both have the same tendency however, since l1 uses ridge regularization while l2 uses lasso. It appears that l2 with c = 3.5 is the best among all with the highest accuracy over 0.81, while l1 with same c did worse than that. When c grows from 3.5, both l1 and l2 become worse with reducing in accuracy and this suggests overfitting occurs when c is way too big.

1.3 Multilayer Perceptron Model for Sentiment Analysis

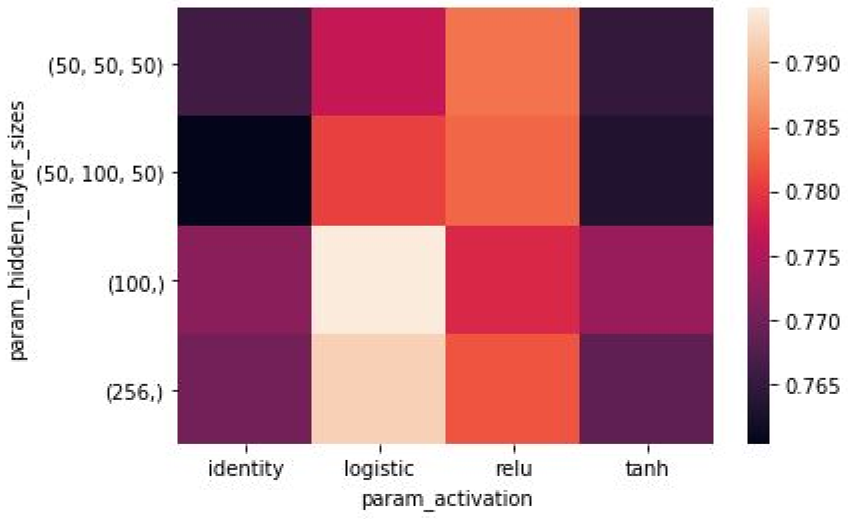
I used sklearn.neural_network.MLPClassifier for this part and several hyperparameters were used. As shown by the graph below, several sets of number of neurons were used alongside the default value in order to test out the best value range for neurons number within every hidden layer. Four most common activation functions were used: linear, sigmoid, relu and tanh. Although Sigmoid is commonly used as logistic activation function, it still have some problems like slow convergence speed and uncentered gradient updates. ReLU converges better than TanH and it also has one advantage over TanH: it does not update weights in a way that some part of the neurons can be set to deactivation. TanH on the other hand, converges better than Sigmoid, however it often makes part or the neurons deactivate, leading to totally uselessness.
Cross-Validation was also used in the same way as the previous question with 5 folds and 0 verbosity.



```
BEST SCORE:
0.7941666666666667
STANDARD DEVIATIONS
[0.02302625 0.01772671 0.02759453 0.02272266 0.02623346 0.01837117
 0.02166667 0.02524189 0.01925703 0.01886539 0.0189755  0.02191366
 0.02504856 0.02373903 0.02448781 0.02216573]
STANDARD DEVIATION FOR BEST SCORE:
0.021666666666666667

/home/jiawei/anaconda3/envs/machinelearning/lib/python3.8/site-packages/sklearn/neu
ral_network/_multilayer_perceptron.py:568: ConvergenceWarning: Stochastic Optimizer:
Maximum iterations (120) reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
ax_mlp = sns.heatmap(pivot_mlp)
```

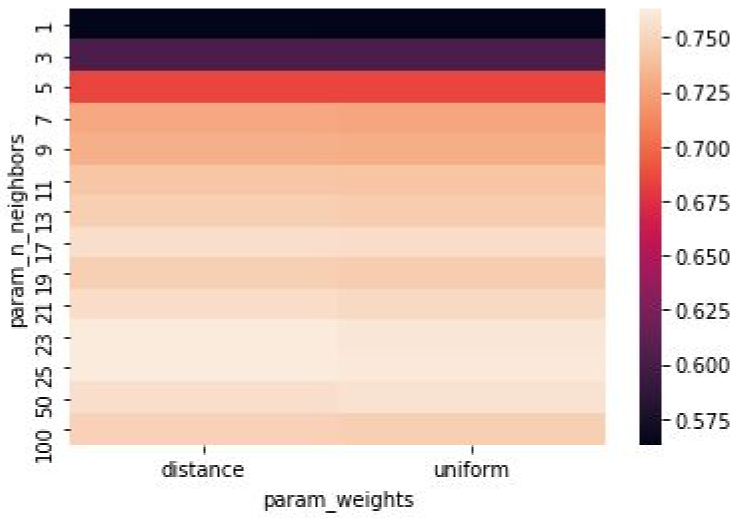


From the graph, we can see that Sigmoid with 100 neurons did the best with accuracy over 0.79. Still, overfitting occurs when the number of neurons is too large. For the other three activation functions, they seem to be underfitted as they have not come to the best result as possible.

1.4 K Nearest Neighbors Model for Sentiment Analysis

As the same in previous assignment, I tested KNN on dataset with different K value. The value range of K is from 1 to 100 with default value of 5. Also, two weight functions were used (distance and uniform). Uniform function marks every point in a cluster as equal while distance function marks every point with a weight inversed against the distance. Cross-Validation was also preformed with 5 folds and 0 verbosity.

```
BEST SCORE:
0.7629166666666667
STANDARD DEVIATIONS
[0.01144128 0.01144128 0.07474958 0.07526112 0.02698379 0.02757565
 0.00805795 0.00870026 0.00618017 0.00552771 0.01333333 0.0124024
 0.01502313 0.01427653 0.01803738 0.01743042 0.0164781  0.0152525
 0.02060711 0.01983438 0.02186607 0.02008662 0.02207154 0.02079162
 0.03080517 0.02225952 0.0209165  0.01902119]
STANDARD DEVIATION FOR BEST SCORE:
0.02008661798865659
```



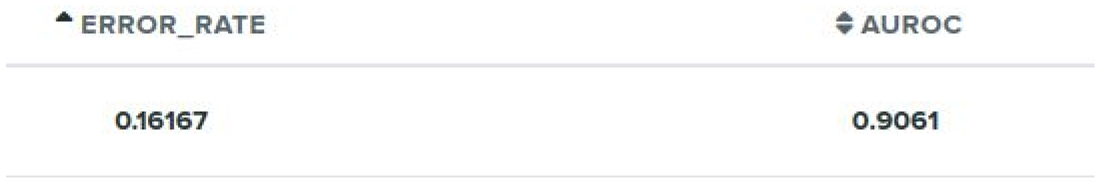
As graph suggests, the result gets better generally when number of neighbors goes up. However, when K is between 19-21 and also bigger than 50, the result gets worse. Best score is over 0.76. However, this is the worst among all three models.

1.5 Summary and Submission

The best model so far is the Logistic Regression Model, compared to KNN and MLP. And I believe it is partly because of the TF-IDF algorithm. It seeks a way to normalize data into feature vectors and performs feature tuning. Vectorization also coms into play as it penalizes closely related attributes while most unrelated attributes have already been removed using stopword dictionary. And the result is two separable clusters that logistic regression can handle.

However this does not necessarily mean KNN and MLP are worse, since in the graph we can see some parameters are not at their best with some models finished underfitted. Future experiments need to be conducted in order to get better KNN and MLP models before we make the conclusion.

My submission result is as below:



ERROR_RATE	AUROC
0.16167	0.9061

Based on GradeScope, my Logistic Regression Model has an error rate of 0.16167 with an AUROC over 0.90. This is close to the accuracy of training dataset. I also uploaded the other two models however error rates of both models are higher than logistic regression while AUROCs lower. This proves my assumption that my Logistic Regression Model is the best among all three.

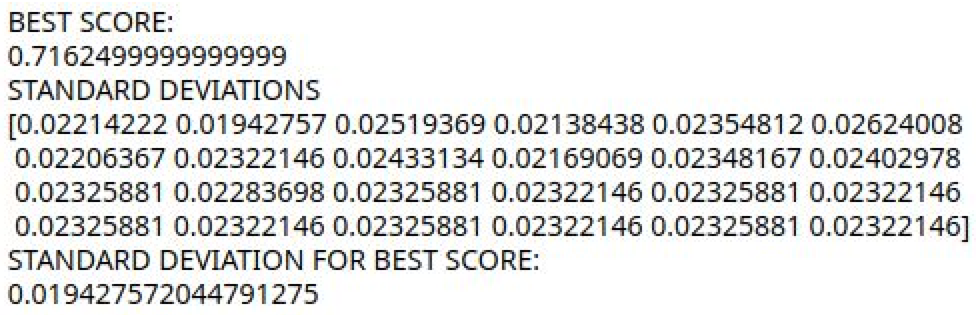
Part Two: Classifying Review Sentiment with Word Embeddings

2.1 Vectorization with GloVe:

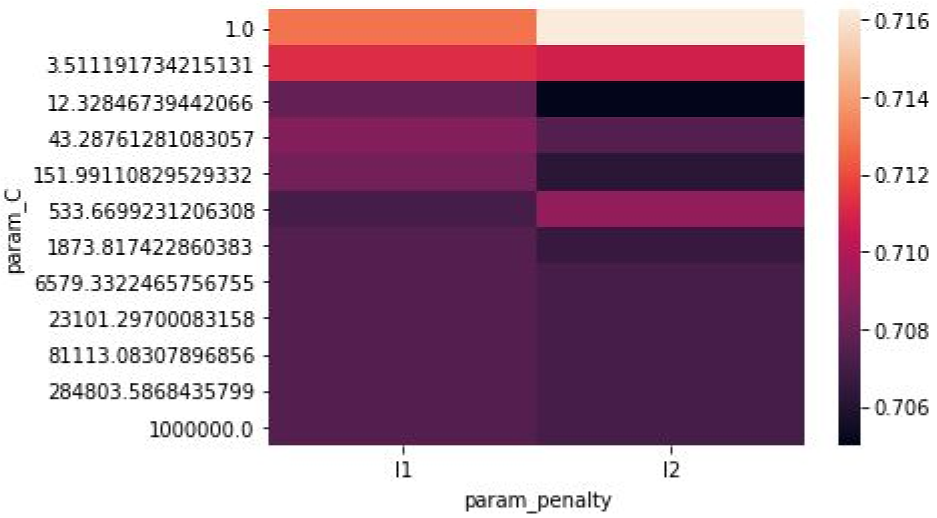
Before vectorizing dataset, first I loaded it and preprocessed it in the same way as 1.1. As stated in the documentation, each line of the given file consists of a word with a 50-value embedding vector, so first I averaged all feature vectors to standardize input data. Summing or concatenating feature vectors will cause problems with different lengths in words. Then weighing was done by multiplying vectors with TF-IDF and calculating using TfidfVectorizer. And the result is 50 dimensions for each feature vector.

2.2 Logistic Regression Model on Word Embeddings

The default Logistic Regression Model from scikit-learn with ‘liblinear’ solver was used since solver ‘lbfgs’ supports only 'l2' or 'none' penalties but I wanted to try ‘l1’ penalty again ‘l2’. Two hyperparameters were used as: the inverse penalty C (C=C) and regularization (penalty=penalty). For C, I used np.logspace(0, 6, 12) and 12 different samples were generated. For penalty, both l1 and l2 regularization were used as a measurement of overfitting. Cross-Validation was used from sklearn.model_selection.GridSearchCV, with 5 folds and 0 verbosity. The model ran till it reached convergence and the result is as follows:



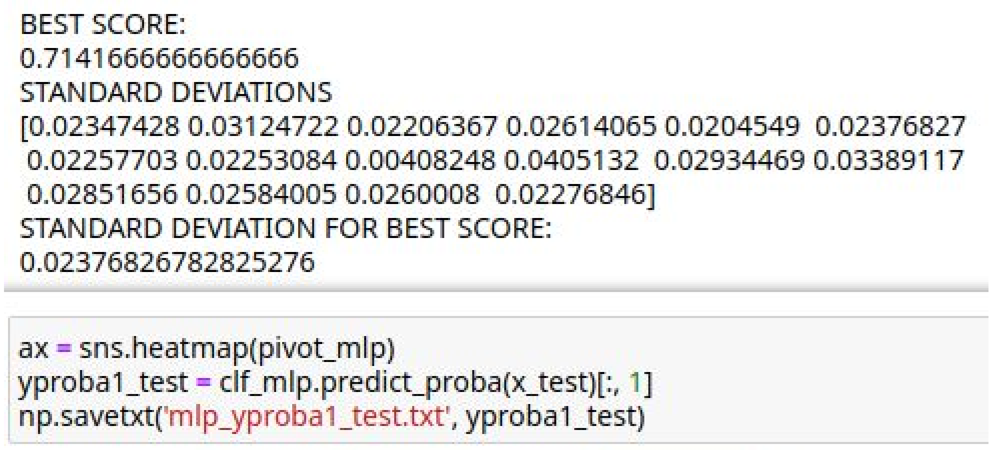
```
BEST SCORE:
0.7162499999999999
STANDARD DEVIATIONS
[0.02214222 0.01942757 0.02519369 0.02138438 0.02354812 0.02624008
 0.02206367 0.02322146 0.02433134 0.02169069 0.02348167 0.02402978
 0.02325881 0.02283698 0.02325881 0.02322146 0.02325881 0.02322146
 0.02325881 0.02322146 0.02325881 0.02322146 0.02325881 0.02322146]
STANDARD DEVIATION FOR BEST SCORE:
0.019427572044791275
```



The two different penalties gave quite different accuracy results. And also we can see that accuracy stopped changing at a relatively smaller C compared to the Logistic Regression Model on Bad of Words in part 1.2. The value range for accuracy is also smaller than the previous one. However, both models share the same thing that both l1 and l2 performed better when C is smaller.
In this graph, we can see that this model is best with l2 regularization with C = 1.0. The highest accuracy is 0.716 and overfitting occurred when C got bigger. One possible explanation for no changing accuracy when C hit 6579 might because of the complex vectorization of the feature inputs.

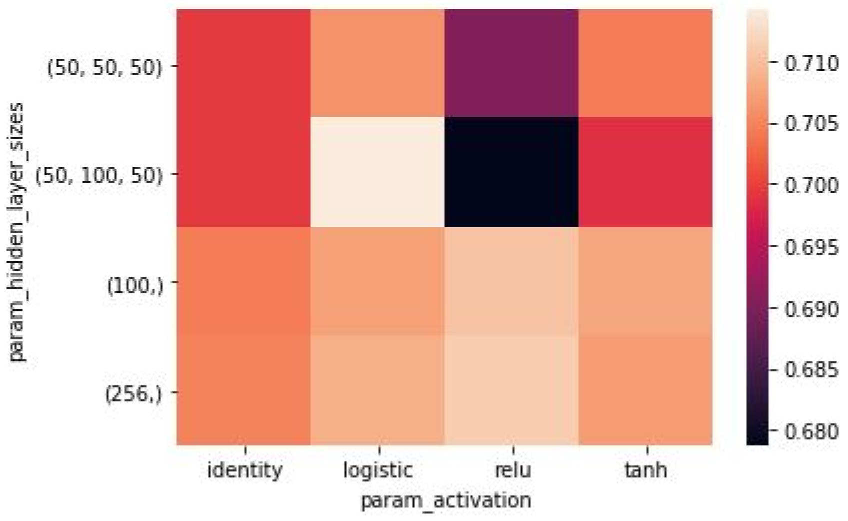
2.3 Multilayer Perceptron Model on Word Embeddings

I used sklearn.neural_network.MLPClassifier for this part and several hyperparameters were used. As shown by the graph below, several sets of number of neurons were used alongside the default value in order to test out the best value range for neurons number within every hidden layer. Four most common activation functions were used: linear, sigmoid, relu and tanh. Although Sigmoid is commonly used as logistic activation function, it still have some problems like slow convergence speed and uncentered gradient updates. ReLU converges better than TanH and it also has one advantage over TanH: it does not update weights in a way that some part of the neurons can be set to deactivation. TanH on the other hand, converges better than Sigmoid, however it often makes part or the neurons deactivate, leading to totally uselessness. Cross-Validation was also used in the same way as the previous question with 5 folds and 0 verbosity.



```
BEST SCORE:
0.7141666666666666
STANDARD DEVIATIONS
[0.02347428 0.03124722 0.02206367 0.02614065 0.0204549  0.02376827
 0.02257703 0.02253084 0.00408248 0.0405132  0.02934469 0.03389117
 0.02851656 0.02584005 0.0260008  0.02276846]
STANDARD DEVIATION FOR BEST SCORE:
0.02376826782825276
```

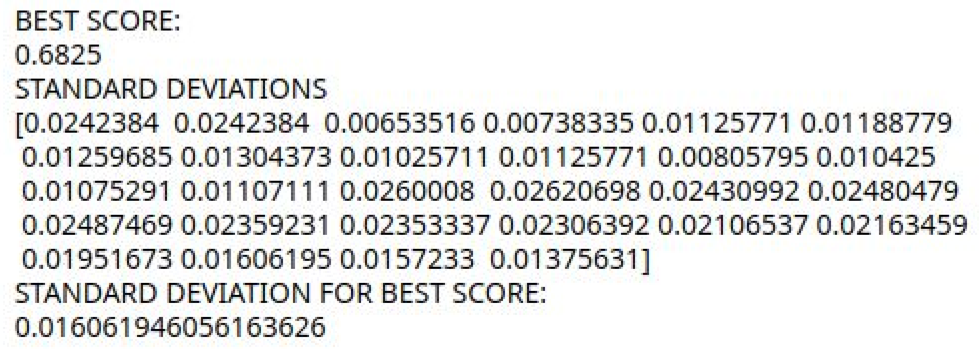
```
ax = sns.heatmap(pivot_mlp)
yproba1_test = clf_mlp.predict_proba(x_test)[:, 1]
np.savetxt('mlp_yproba1_test.txt', yproba1_test)
```



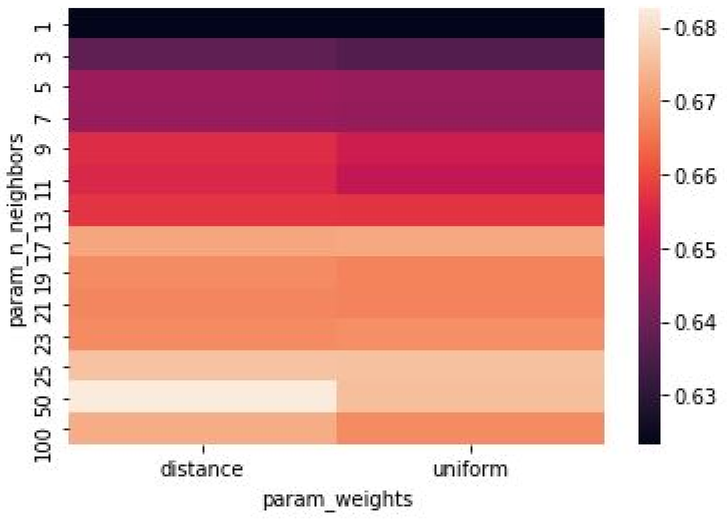
This MLP Model has a quite different result with the MLP Model used in part 1.3. Although the accuracy range for both models are similar, the distributions are totally different. The lowest accuracy occurs when using ReLU with (50, 100, 50) neurons for each hidden layer and the highest accuracy occurs when using Sigmoid with the same amount of neurons. The highest accuracy is 0.714.

2.4.1 K Nearest Neighbors Model on Word Embeddings

As the same in previous assignment, I tested KNN on dataset with different K value. The value range of K is from 1 to 100 with default value of 5. Also, two weight functions were used (distance and uniform). Uniform function marks every point in a cluster as equal while distance function marks every point with a weight inversed against the distance. Cross-Validation was also preformed with 5 folds and 0 verbosity.



```
BEST SCORE:
0.6825
STANDARD DEVIATIONS
[0.0242384  0.0242384  0.00653516 0.00738335 0.01125771 0.01188779
 0.01259685 0.01304373 0.01025711 0.01125771 0.00805795 0.010425
 0.01075291 0.01107111 0.0260008  0.02620698 0.02430992 0.02480479
 0.02487469 0.02359231 0.02353337 0.02306392 0.02106537 0.02163459
 0.01951673 0.01606195 0.0157233  0.01375631]
STANDARD DEVIATION FOR BEST SCORE:
0.016061946056163626
```

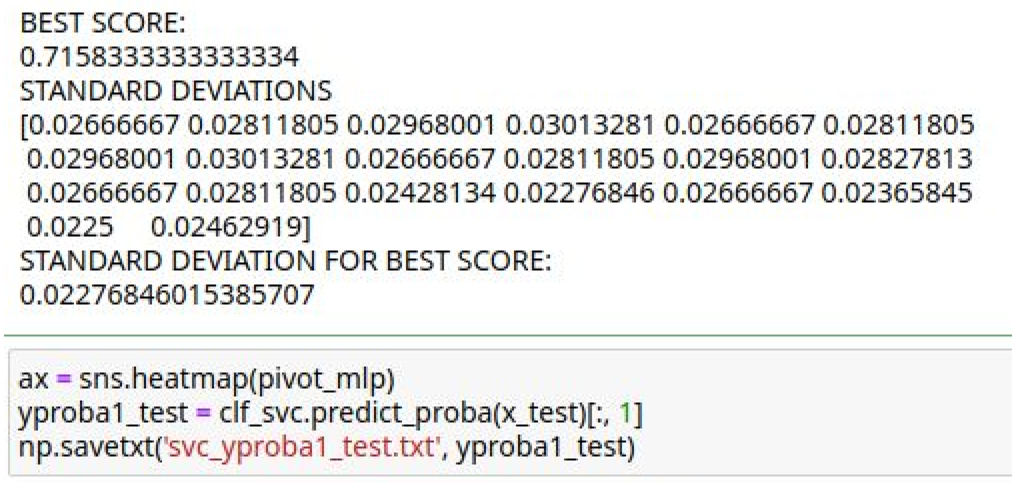


Same as the previous KNN model, this one also demonstrates overfitting when k is relatively small and accuracy starts growing when k becomes bigger. However, one major difference between these two models is that in this one, distance and uniform weight functions distinguish from each other and one possible explanation for this is that since we already averaged the feature vector, a distance function can weight all points by the inverse of their distance thus making them more distinguishable from each other and allow the model to train better.

As graph suggests, the result gets better generally when number of neighbors goes up. However, when K is 13-17 and also bigger than 50, the result gets worse. Best score is over 0.68. However, this is also the worst among all three models.

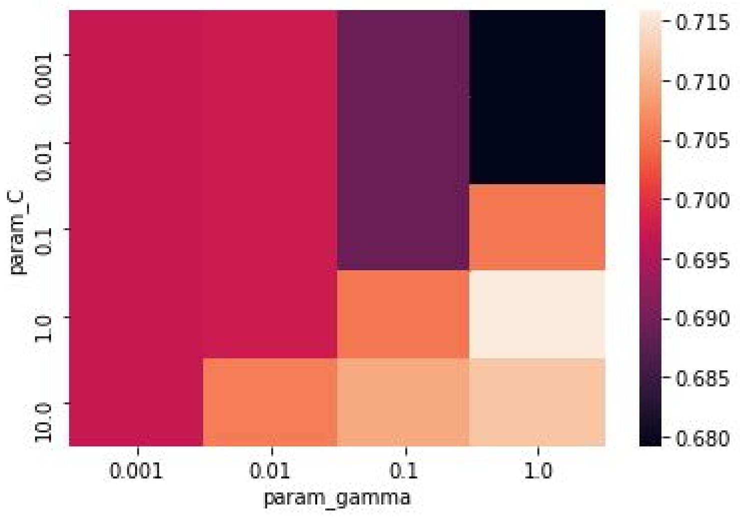
2.4.2 SVM Model on Word Embeddings

sklearn.svm.SVC was used to build a Support Vector Machine Model. Two hyperparameters were used: C and gamma. For C, I chose 5 values: 0.001, 0.01, 0.1, 1 and 10, and for gamma I choose 4 values: 0.001, 0.01, 0.1 and 1. As always, for cross-validation, 5 folds with 0 verbosity are used.



```
BEST SCORE:
0.7158333333333334
STANDARD DEVIATIONS
[0.02666667 0.02811805 0.02968001 0.03013281 0.02666667 0.02811805
 0.02968001 0.03013281 0.02666667 0.02811805 0.02968001 0.02827813
 0.02666667 0.02811805 0.02428134 0.02276846 0.02666667 0.02365845
 0.0225     0.02462919]
STANDARD DEVIATION FOR BEST SCORE:
0.02276846015385707
```

```
ax = sns.heatmap(pivot_mlp)
yproba1_test = clf_svc.predict_proba(x_test)[:, 1]
np.savetxt('svc_yproba1_test.txt', yproba1_test)
```



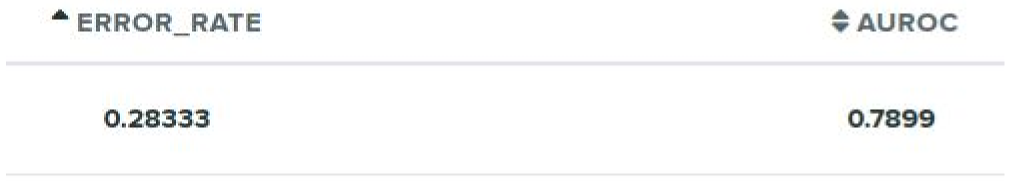
As shown in graph, the accuracy of model does not change with C when gamma is set too small and this is caused by underfitting. As gamma gradually grows, accuracy starts to change with C, suggesting that gamma regularization does have a tightening effect on model. However, there are two tendencies for higher gamma: the accuracy drops when C is too small and the accuracy grows when C is relatively higher, suggesting that increasing in C can help better fit the training dataset.

The highest score for this model is 0.7158, better than KNN but still slightly worse than Logistic Regression Model.

2.5 Summary and Submission

Similar to part one, the best model is still the Logistic Regression Model with accuracy of 0.7162. However, every model in part two gained a lower accuracy compared to the same model in part one and this strongly suggests overfitting. In this case, TF-IDF algorithm and averaging all feature vectors may not be the best method to normalize inputs. And one possible explanation for this might be that every word has context that is tightly related and by using TF-IDF can overfit the data with respect to the testing set and leads to a decrease in accuracy. Averaging may be still a good way for normalization feature vectors but TF-IDF is definitely not a good choice.
Also, this does not necessarily mean KNN, MLP, and SVM are worse, since in the graph we can see some parameters are not at their best with some models finished severely underfitted or overfitted. Future experiments need to be conducted in order to get better models before we make the conclusion.

My submission result is as below:



ERROR_RATE	AUROC
0.28333	0.7899

Based on GradeScope, my Logistic Regression Model has an error rate of 0.28333 with an AUROC of 0.7899. I also uploaded the other three models however error rates of all models are higher than logistic regression while AUROCs lower. This proves my assumption that my Logistic Regression Model is the best among all four.

2.6 Comparison with Part One

As previously stated, the lower accuracy is the major problem. When building models for Word Embeddings, feature vectors were cut down from thousands to only 50. This vectorization resulted in a loss of information. However it still owns some advantages like a more compact input data format and quicker computing time (Of course with the loss in accuracy), and overfitting is another problem that causes this problem, which is related with TF-IDF algorithm itself.