Shanney Suhendra
CS135 P2

Part 1.1_Pre-process Data

Step 1	Lower-case, remove punctuation, remove numbers	
Step 2	Remove stop-words	
Step 3	Lemmatization tokenizer	Stemming tokenizer
	0.970	0.965
Step 4	Lemmatization tokenizer	Lemmatization tokenizer + word-pair
	0.970	0.9933

Table 1: summary of pre-process steps undergone

	website_name	text
0	amazon	Oh and I forgot to also mention the weird colo...
1	amazon	THAT one didn't work either.
2	amazon	Waste of 13 bucks.

Fig 1.1: original input data

	website_name	text
0	amazon	oh and i forgot to also mention the weird colo...
1	amazon	that one didnt work either
2	amazon	waste of bucks

Fig 1.2: input data after step 1

Step 1: The general pre-processing steps done to the data was setting everything to lower-case, removing punctuations, and removing numbers (fig 1.1 → fig 1.2).
Step 2: The second step was removing stop-words (common words). Instead of using the 'english' stop-word list, I made my own list. Reason for this is that the 'english' list consists of specific words that are crucial in creating the algorithm. An example of these words is 'not'. If 'not' was eliminated, sentences, for example, consisting of 'not good' will turn to 'good', thus eliminating the nuance that we're looking for in a sentence. Additionally, since punctuations were removed, words in the list like 'they're' should be changed to 'theyre'.

	abandoned	abhor	ability	able	abound	abovepretty	abroad	absolute	absolutel	absolutely	...	youtube
0	0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	0	...	0

Fig 1.3: count vectorizer, document-term matrix

For steps 3 and 4, count vectorizer was utilized to convert the text data into input token-counts vectors (fig 1.3), then logistic regression was created with the input vectors as the input, and accuracy on the training data was determined. **Step 3:** I compared utilizing lemmatization vs stemming tokenizer. Purpose of both tokenizer is to further unify word-tokens ignoring things like plural endings, different tense endings, etc. As shown on table 1, utilizing lemmatization tokenizer produces a better accuracy than utilizing stemming (0.970>0.965). This is expected as opposed to stemming, lemmatization looks beyond word reduction and apply morphological analysis to the word. For example, for stemming, 'studies' → 'studi' and 'studying' → 'study', whereas for lemmatization, 'studies' → 'study' and 'studying' → 'study'.

Step 4: I compared, using the lemmatization tokenizer for both, counting single word vs word-pair as the input token-counts vectors. As shown on table 1, word-pair generated a better accuracy than single word (0.9933>0.970). Hypothetically this makes sense as words that appear next to each other generally correlate with one another. For example, single words 'not' is negative and 'good' is positive, but if you pair them together, 'not good' is negative, thus it helps increase the performance of the algorithm generated by the word-pair input vectors as the input.

Additionally, I compared using count vectorizer vs TFIDF vectorizer, and count vectorizer generated a better accuracy (0.9933>0.947)

Part 1.2_Logistic Regression

Note: cross validation is used to evaluate machine learning models generated by training the model on subsets of the input data and then evaluating the data using the complementary subset. In fig 1.4 and fig 1.5, 'testing data' refers to the complementary subset used to test the model and 'training data' refers to the data used to train the model. In this case, 5-fold cross validation was used to generate the models, with the graph values calculated from mean, and the hyperparameter values chosen were determined by the value in testing data that resulted in the best accuracy.

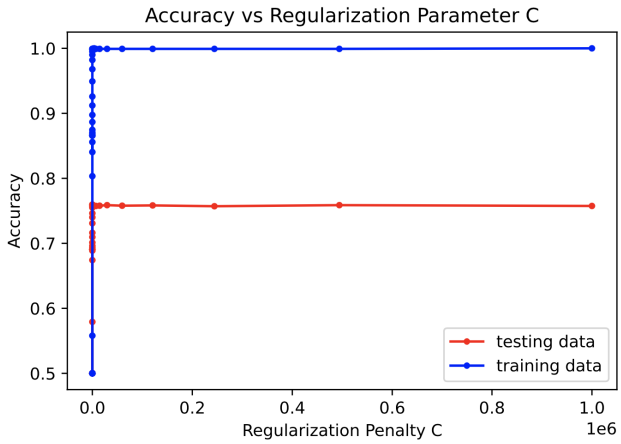


Fig 1.4: change in accuracy as regularization penalty increases

Hyperparameter 1: regularization penalty c (fig 1.4)

Regularization penalty is used to prevent overfitting and underfitting when creating the logistic model, such that the accuracy of prediction when predicting other/new data sets improves, by blocking out noise data and prioritizing important data. Regularization penalty was set to a range of values c = np.logspace(-9,6,50) and the corresponding c value that gives the best accuracy was used. It was found that c = 3.089 resulted in the best accuracy of 0.760 for testing data.

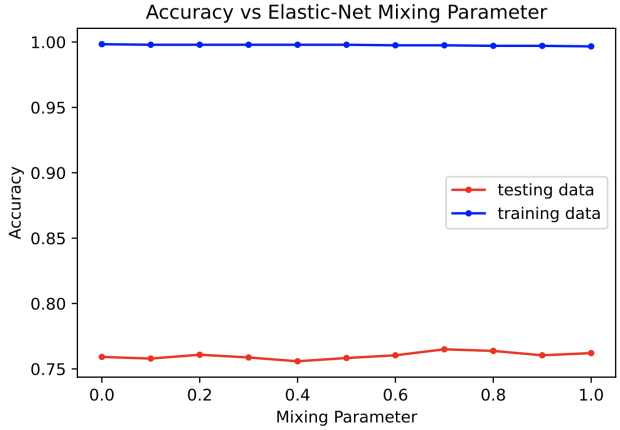


Fig 1.5: change in accuracy as elastic-net mixing parameter increases

Hyperparameter 2: elastic-net mixing parameter (fig 1.5)

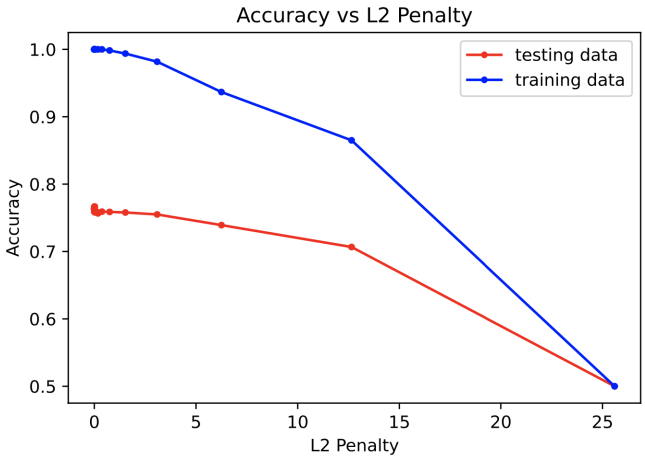
L1 and L2 penalty both prevent overfitting and underfitting. Elastic-net mixing parameter is a combination of L1 and L2 penalty, with the penalty being equivalent to L2 if the parameter is 0 and L1 if the parameter is 1. Reason why a combination of L1 and L2 was used is to take advantage of what both individually have to offer, where L2 gives accuracy while L1 gives sparsity, thus we would be able to gain both accuracy and sparsity. The parameter was set to a range of values p from 0 to 1, with increments 0.1. elastic-net penalty was used and regularization penalty c = 3.089 was used. If L1 penalty was used, p = 1, accuracy is 0.762. Likewise, if L2 penalty was used, p = 0, accuracy is 0.759. We can conclude that a mix of both L1 and L2 generates better accuracy. It was found that p = 0.7 resulted in the best accuracy of 0.765 for testing data.

Final logistic regression model

c = 3.088843596477485, solver = 'saga', penalty = "elasticnet", l1_ratio = 0.7, max_iter = 300

Part 1.3_Neural Network (MLP) Model

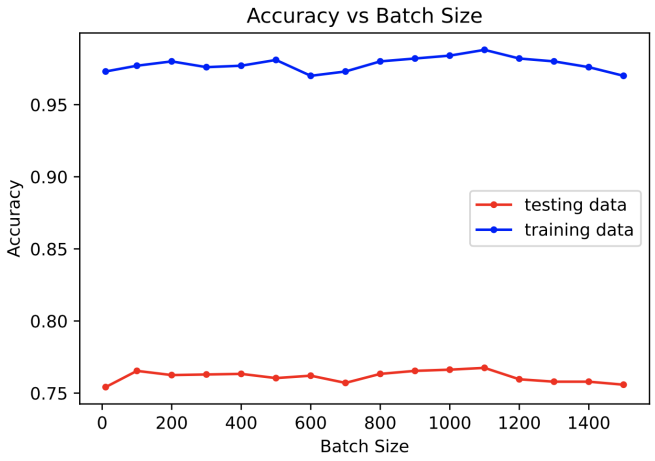
Note: same as part 1.2 logistic regression, 5-fold cross validation was used to generate the models, with the graph values calculated from mean, and the hyperparameter values chosen were determined by the value in testing data that resulted in the best accuracy.



Hyperparameter 1: L2 penalty (fig 1.6)

L2 penalty was set to a range of values alpha = np.logspace(-9,6,50) and it was found that alpha = 0.00133 resulted in the best accuracy of 0.767 for testing data. We can see that the general trend is that as the L2 penalty increases, the accuracy of the model decreases. (graph: datas to the right were cut to enlarge data points on the left)

Fig 1.6: change in accuracy as elastic-net mixing parameter increases



Hyperparameter 2: batch size (fig 1.7)

Batch size is the number of samples that is processed by the model. Here, batch size was set to a range of values b = 10,100,200,..,1500 with increments of 100, with elastic-net mixing parameter alpha set at 0.00133. It was found that b = 1100 resulted in the best accuracy of 0.768. In theory, a large batch size number will cause the model generated to overfit the training data, we can see this in fig 1.7, as batch size is over 1100, the general trend is that the accuracy decreases as batch size increases.

Fig 1.7: change in accuracy as batch size increases

Final MLP model

alpha = 0.00133, batch_size = 1100, max_iter = 100

Part 1.4_Third Model-Random Forest Classifier Model

For the third model, I decided to generate a random forest (RF) classifier model. Reason behind this is that RF classifiers are suitable for dealing with highly dimensional noisy data in text classification. The algorithm is able to handle large datasets efficiently and generally predicts outcomes more accurately than other algorithms such as decision trees. Same as before, 5-fold cross validation was used to generate the models, with the graph values calculated from mean, and the hyperparameter values chosen were determined by the value in testing data that resulted in the best accuracy.

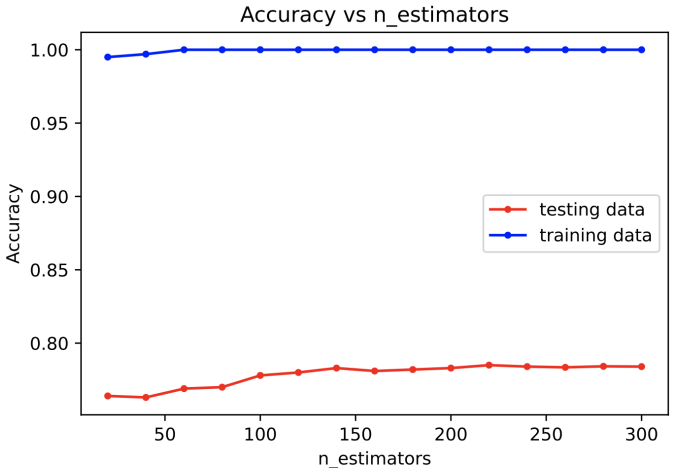


Fig 1.8: change in accuracy as n_estimators increases

Hyperparameter 1: n_estimators (fig 1.8)

N_estimators correspond to the number of trees in the forest. In theory, as the number of trees increases, overfitting should decrease when generating the model. Here, n_estimators was set to range e = 20,40,..,300 with increments of 30. It was found that e = 220 resulted in the best accuracy of 0.785. Typically, the model performs low when n_estimators are less than 100, we can see this on the training data at fig 1.8. Reason why it performs lower is because the less the number of trees, the closer the model is to a decision tree, which is not what we're aiming for as that defeats the whole purpose of trying to generate a random forest classifier model.

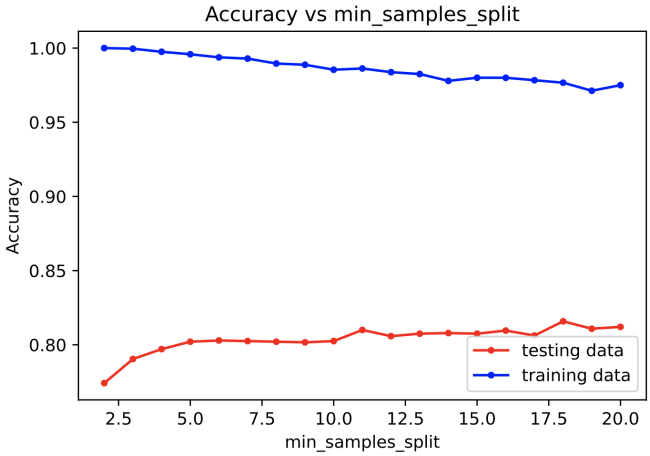


Fig 1.9: change in accuracy as min_samples_split increases

Hyperparameter 2: min_samples_split (fig 1.9)

min_samples_split correspond to the minimum number of samples required to split an internal node. Here, min_samples split was set to range min = 2,3,..20 and it was found that min = 18 resulted in the best accuracy of 0.816. Analyzing the testing data graph, we can see that the general trend is as the min_samples_split increase, the accuracy increases, with the accuracy increasing significantly on the first portions of the graph when min_samples_split increase from 2.5 to 5.

Analysis on all the training-testing data graphs: why accuracy is greater on training data than testing data:

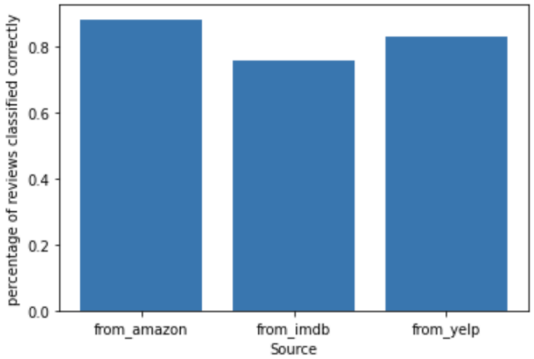
If we see fig 1.4-1.9, all the training data line is above the testing data. This is expected as the models were generated by training them using the training data. Thus, when training data is inputted for the model to predict, it resulted in better performance than when testing data is inputted.

Part 1.5_Classifier Summary

Model	Accuracy
Logistic Regression	0.768
Neural Network (MLP)	0.785
Random Forest Classifier	0.816

Table 2: accuracy of models generated

Based on the accuracy, the random forest classifier model performs best overall, telling us that it was the best in not overfitting the training data. Reasons for this could be because random forest uses the ensemble learning technique, which helps reduce overfitting and also reduce variance. Additionally, random forest works well with high dimensional data sets. We can also see that the logistic regression and neural network both have similar accuracy, we can think of logistic regression as a one-layer neural network.



Looking more into random forest classifier, the model did not perform the same across the different sources (amazon, imdb, yelp) if the data set were separated by different sources. As in, the model did not predict at 0.816 accuracy for each source. Fig 2 shows the percentage (in decimals) of reviews that were classified correctly based on the source. To elaborate, the logistic model was generated three times, by separating the data into three based on the sources. Then cross validation was done on each of the data sets and then the model was generated, just as how it was in previous models. We can see that imdb_source has the least accuracy. This may be because it is more difficult to analyze movie reviews than other reviews such as products or restaurants, as there are many more factors that come into play when critiquing movies. In a larger perspective, the reason why the accuracies are different for different sources may be due to misinformation of context. For example, words such as “cheap” may be positive for restaurant reviews but may also be negative for product reviews, as in the products were poorly made.

Part 1.6_Leaderboard

Error rate = 0.16833
Accuracy = 0.91414
The model performed better on the x_test.csv than during training and cross validation. It is not expected as in usual cases, accuracy on training data should be higher than testing data since the model was trained using the training data. However, this does tell us a few things and there may be reasons why the testing did better. For one, it tells us that the model generated did not overfit the training data, because if it did overfit then the accuracy should decrease when applying the model on the testing data. Additionally, a reason why the testing did better than training may be because there are more reviews on the testing data that are similar to reviews on training data for which the model classified correctly.