Sharrey Suhendra

Professor Allen

Comp 135: Intro to Machine Learning - Project 02

Part One: Classifying Review Sentiment with Bag-of-Words Features

1)

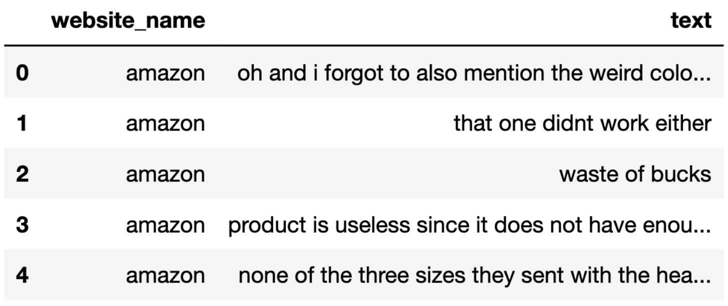
Pre-Processing

Figure 1.1 shows the data after the 3 Basic techniques that were done, that are:

1. Setting everything to Lower Case
   Ensures that all texts are in the same case. Otherwise, the machine will interpret and treat words for example, ‘good’ and ‘Good’, differently which is a problem to avoid.

2. Removing Punctuation
   This reduces unwanted features by removing unwanted hyphens, modulus signs, commas, and full stops that are not important for machines to evaluate.

3. Removing Digits and Words that Include Digits (e.g. hello23, he2llo, 23)
   There are cases of a mistype or username that may confuse the machine later on. As for digits alone, it does not play many factors in the reviews; Unless we are dealing with rating review data, in this case, a machine cannot distinguish if a user wrote a rating or not.



	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
3	amazon	product is useless since it does not have enou...
4	amazon	none of the three sizes they sent with the hea...

Figure 1.1 Data After Basic Pre-Processing

I also did other pre-processings that require more evaluations:

1. Removing Stop Words
   Stop words are highly common occurring words in a text that are invaluable to a machine to evaluate. There are several libraries that have pre-made stop words, such as the NLKT library. However, there are issues, that is the filtering is too aggressive: stop-words may contain words we believe are important; For example in NLKT, ‘not’ is considered a stop word so we would result in ‘good’ for both ‘good’ and ‘not good’ reviews.

   A better approach would be to customize a list of stop words. I decided to research the most commonly used words that are invaluable/of low relevance features, as well as rare words. Rare words are important to consider because they can lead to models overfitting to that feature.

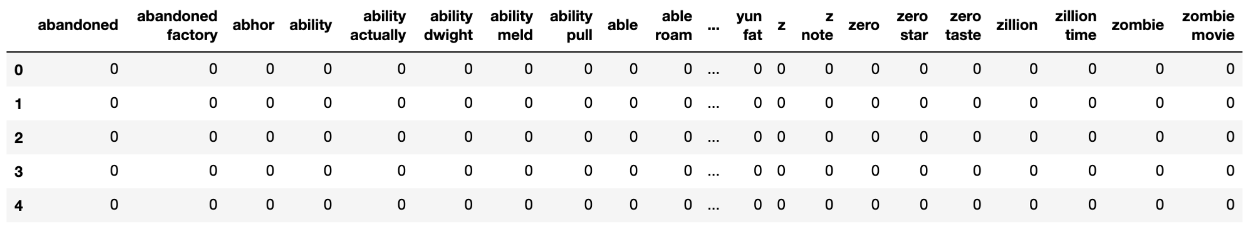
2. Bigrams: Single + Pair of words
   I built a logistic regression model on data that contained only single word features vs both single and pair of words that are next to each other (ngram_range to (1,2)). After evaluating the accuracy, I concluded that the latter led to better performance. Some words are important as a pair, for example in 'not happy', 'happy' is positive but we will need both together 'not happy' to know that it's negative.

3. Stemming & Lemmatization
   In stemming and lemmatization, words are unified and reduced to their root by ignoring plural endings and different tense endings. Lemmatization is generally considered to be the better option; Stemming removes the last few characters leading to wrong spellings and meanings whereas Lemmatization looks at the context leading to real dictionary words. I created two different data frames for the two approaches, to be analyzed later on.

4. Inverse Document Frequency (IDF)
   An IDF is a statistical measure that evaluates how relevant a word is, such that the more documents in which a term shows up, the less important it is. This is done by applying sklearn's TfidfTransformer. I created two different data frames, performing IDF on stemmed data, another on lemmatized data.

Choosing Best Pre-Processed Data

I evaluated 4 pre-processed data by building a logistic regression model on the training data. The 4 data are 1) Lemmatized, 2) Stemmed, 3) Lemmatized + IDF, 4) Stemmed + IDF. I evaluated the accuracy of each model and concluded that the first, data that is lemmatized (accuracy = 0.99125), was the best data to be used from now on.



	abandoned	abandoned factory	abhor	ability	ability actually	ability dwight	ability meld	ability pull	able	able roam	...	yun fat	z	z note	zero	zero star	zero taste	zillion	zillion time	zombie	zombie movie
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 1.2 Final Pre-Processed Features

2)

From 1), I utilized logistic regression to find the best final-feature vectors. Now, I explored the best logistic regression model by training the model through k-fold cross-validations and hyperparameter tunings. The 2 hyperparameters chosen are 1) Regularization Parameter C and 2) Elastic-net mixing parameter. Since we do not have the output testing data, I decided to perform 5-fold cross-validations through the cross_validatate function from sklearn, and obtain the mean accuracies of the 5 models. This way, the model will not be subjected to overfitting the training data and we can get a better view of the model's performance on unknown inputs. Furthermore, after several experiments, I decided to use the 'saga' solver for all logistic regressions models.

Parameter C Regularization

Different values of the C parameter were explored using a logarithmic scale grid of values ranging from small decimals to a million. The purpose of this C value is to reduce overfitting, as a higher regularization will give a greater penalty on overfitting. For each C-value, I performed

5-fold cross-validation on the training input data and obtain the mean accuracies. Figure 2.1 shows the accuracy of the models depending on the C-value. Moreover, as the C-value regularization penalty increases, so does the uncertainty (standard deviation) of the models. C = 3.09 led to the highest testing accuracy of 0.819. This makes sense because the C value measures the inverse of regularization; A lower C value increases regularization, uncertainty is still low, and it is not too small such that it might lead to heavy regularization and an underfit model.

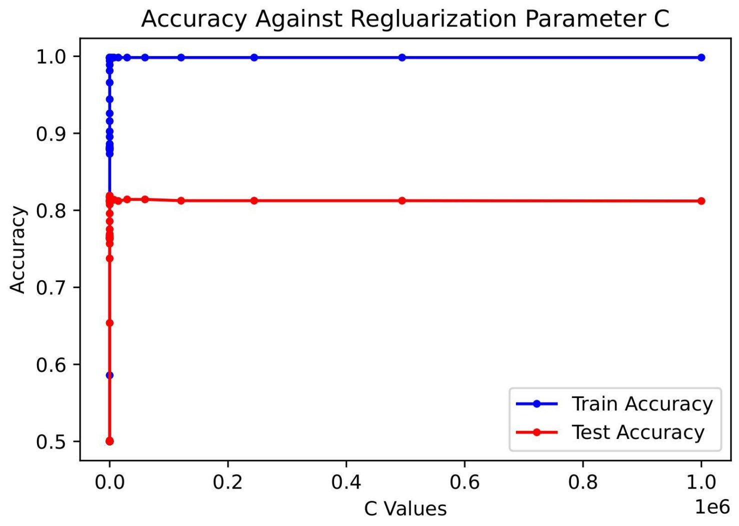


Figure 2.1 Change in Mean Accuracy for Increasing Regularization Parameter C

Elastic-Net Penalty

I experimented with the elastic-net mixing parameter; This is done by setting penalty to 'elasticnet' and varying the l1_ratio parameter. Setting l1_ratio = 0 is equivalent to using L2 penalty (Ridge Regression), while l1_ratio = 1 is equivalent to L1 penalty (Lasso Regression). For 0 < l1_ratio <1, the penalty is a combination of L1 and L2. As before, for each mixing parameter ratio, I performed 5-fold cross-validation and obtain mean accuracies. Figure 2.2 shows the accuracy of the models depending on the mixing parameter. Moreover, the uncertainty (standard deviation) is somewhat parabolic, with smaller uncertainties for 0 < l1_ratio <1. Looking at the testing accuracies, there is an overall decreasing trend as the ratio increased and l1_ratio = 0.1 led to the highest testing accuracy of 0.770.

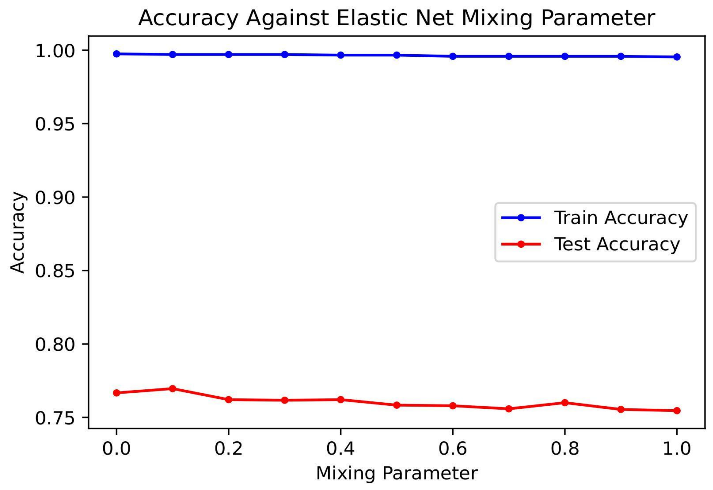


Figure 2.2 Change in Mean Accuracy for Increasing Mixing Parameter (l1_ratio)

3)

Next, I experimented with the Multilayer Perceptron Model that is a feedforward neural network characterized by several layers of input nodes connected as a directed graph. I mainly tested the MLP with 2 hyperparameters: 1) Batch Size and 2) L2 Regularization Penalty.

Batch Size

The batch size refers to the size of minibatches for stochastic optimizers, that is the number of samples processed before the model is updated. The greater the batch size, the more memory space is needed. Figure 3.1 shows the mean accuracy of the 5 fold cross-validated models for every batch size tested. The batch size also affects the likelihood of underfitting and overfitting such that smaller batch sizes provide a regularization effect; And we can see this by the training accuracy increasing as batch size increases, indicating a possibility of overfitting. Furthermore, as the batch size increases so does the standard deviation, and it was concluded that the best batch is 200 with a testing accuracy of 0.7725, which follows the logic of what a batch size does.

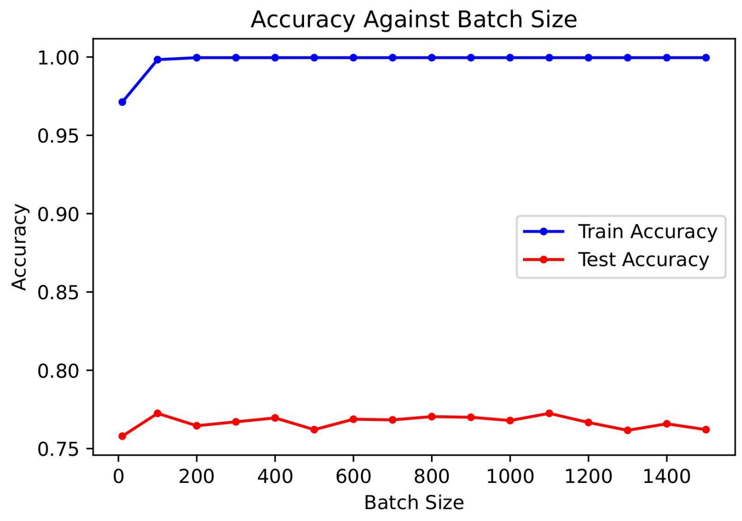


Figure 3.1 Change in Mean Accuracy for Increasing Batch Size

L2 Regularization Penalty

Next, I looked at the L2 Ridge Regression penalty regularization, which is the alpha parameter, by performing cross-validation with alpha ranging on a logarithmic scale. The alpha parameter combats overfitting by constraining weight sizes such that increasing the alpha can fix high variance (possible overfitting) by encouraging smaller weights, and a decreasing alpha can fix high bias (possible underfitting) by encouraging larger weights. Figure 3.2 shows the accuracies for each alpha, and I found that the best alpha value = 0.184 with a 0.771 accuracy on testing data. At this alpha value, the model has a training accuracy of 0.999 which clearly indicates overfitting of the training data; furthermore, the uncertainty that is the standard deviation of the testing data is greater than the training data. This makes it difficult to conclude which alpha value is optimal for the L2 penalty, and I decided to follow the one that resulted in the highest mean accuracy that is 0.184.

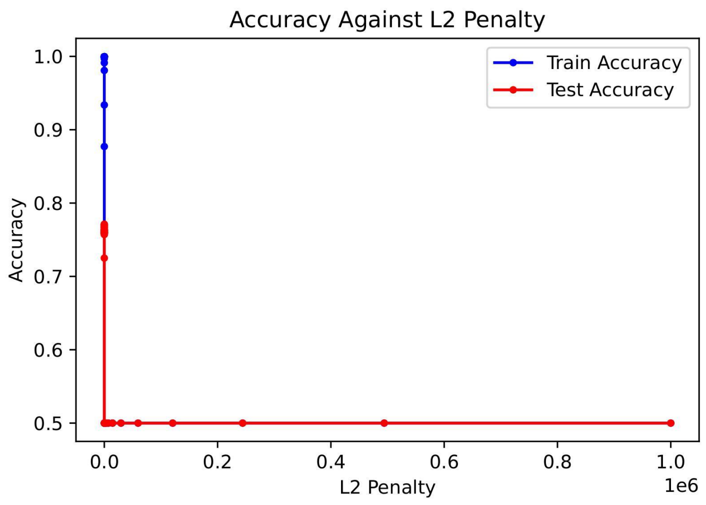


Figure 3.2 Change in Mean Accuracy for Increasing L2 Regularization Penalty

4)

For the last model, I decided to do a tree model and decided on sklearn's Random Forest Classifier. I chose a random forest over a decision tree model because random forests consist of multiple single trees that are individually based on a random sample of training data, making it more accurate than single decision trees. Overall, it adds additional randomness to the model and thus will reduce the likelihood of overfitting the training data. In this experiment, the 2 hyperparameters are 1) Number of Trees and 2) Minimum Number of Sample Splits.

Number of Trees in Forest

Using sklearn's Random Forest, I looked at varying the number of trees in the forest, that is the n_estimators parameter that has a default value of 100. A higher number of trees tend to improve performance, though it will take a longer time and more memory. Figure 4.1 shows the accuracy of the models for every number of trees tested, that is a range between 0 (excluding 0) to 1000 in increments of 100. Furthermore, when the number of trees increased, the standard deviation decreased, though very slightly. As shown in the graph, there is not much fluctuation in the trends of accuracy but there is a small gradual increase in the testing accuracy, such that the best n_estimaters = 800 with an accuracy of 0.732 on testing accuracy.

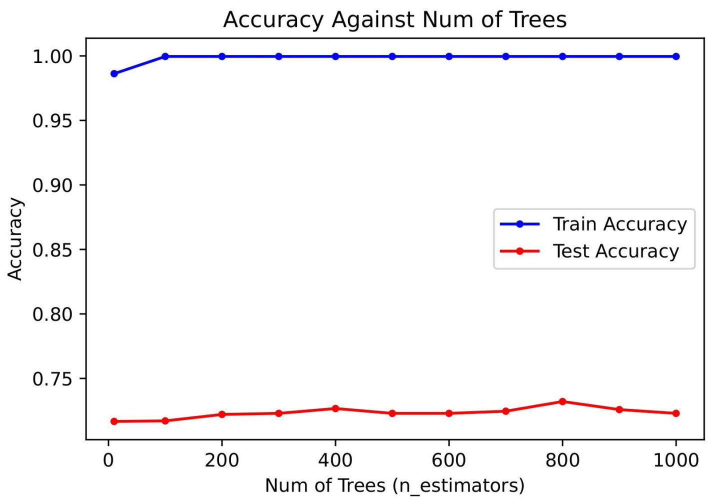


Figure 4.1 Change in Mean Accuracy for Increasing Number of Trees in Forest

Minimum Samples to Split

The last hyperparameter I looked into was the minimum samples to split, that is the minimum number of samples required to split an internal node, varying the min_samples_split parameter that has a default value of 2. Figure 4.2 shows the change in the accuracy of the models for the number of minimum samples to split. Furthermore, the trend in the standard deviation of training data is fluctuating whereas it is just slightly upward sloping for testing data. Based on the figure, there is a clear downward trend of the accuracy for the training data that heavily implies the reduction of overfitting, and a small fluctuation but overall gradual upward trend for the testing data. Simply changing the number from 2 to 4 clearly improved the testing accuracy and it is found that the best minimum number of samples = 6 with a 0.758 accuracy on the testing data.

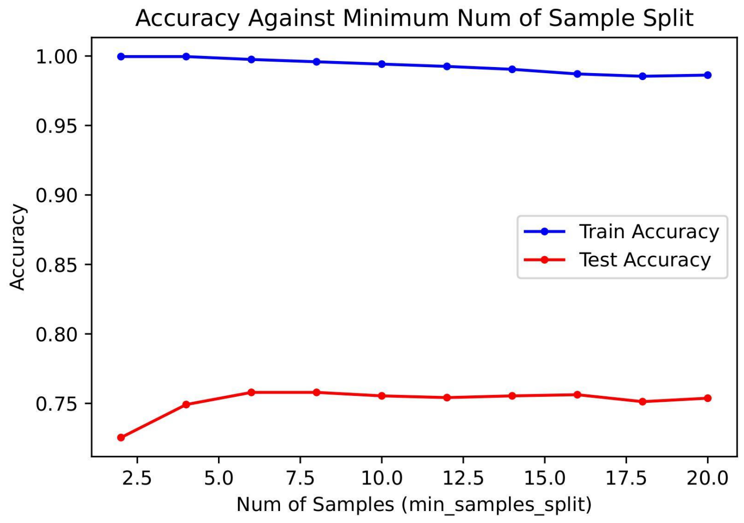


Figure 4.2 Change in Mean Accuracy for Increasing Minimum Number of Samples

5)

Figure 5.1 Summarizes the mean accuracy and standard deviation of the 5 testing models from the 5-cross validations. Based on the summary, the logistic regression model performed best in terms of accuracy and the Random Forest Classifier had the lowest standard deviation. Based on the past experimentations, it showed that the logistic regression model was more flexible in terms that it is the model that least overfits the training data in the cross-validation.

Model	**Accuracy**	**Standard Deviation**
Logistic Regression	0.819	0.0348
Multilayer Perceptron	0.7725	0.0313
Random Forest Classifier	0.758	0.0256

Figure 5.1 Table Performance Summary - Logistic Regression, Multilayer Perceptron, Random Forest

Performance of Best Classifier - Logistic Regression Model

Now focusing on the best model, that is the logistic regression, I decided to evaluate the performance by the domain source of reviews that are Amazon, IMDB, and Yelp. I trained the model on 2160 reviews, with equal distribution such that 720 randomized reviews are taken from each source. I then tested the models on the remaining 240 reviews, again 80 from each source. Figure 5.2 shows the result of the model's prediction, showing the percentage that the model correctly and incorrectly classified for each domain.

The first thing to note about this project's sentiment analysis is that the 3 domains have different purposes. Reviews on Amazon are mostly on products, IMDB on movies/shows/video content, and Yelp on food/restaurants. This makes it much more difficult for the model to correctly classify reviews as distinguishing the source is very important; A word contained in one domain review may mean the complete opposite to another domain. For example, a review on IMDB may say a movie is 'sick' (slang for cool) while another on yelp may say that a food made them 'sick', which to a human can easily distinguish that one's positive and the other negative.

Furthermore, I performed a confusion matrix on each of the source to further evaluate what the model could have done wrong. Since IMDB had the worst classifications, I decided to look into it. For example, the model classified 'A film not easily forgotten' as a false negative. I believe that the phrase of the last 3 words 'not easily forgotten' should be evaluated all three together, and the model I had only took single and pair words; So the words 'not, 'not easily', 'forgotten' and 'easily forgotten' possibly seemed like a negative review to a computer. So, there may be instances where it is necessary to have more than just a two-word pairing.

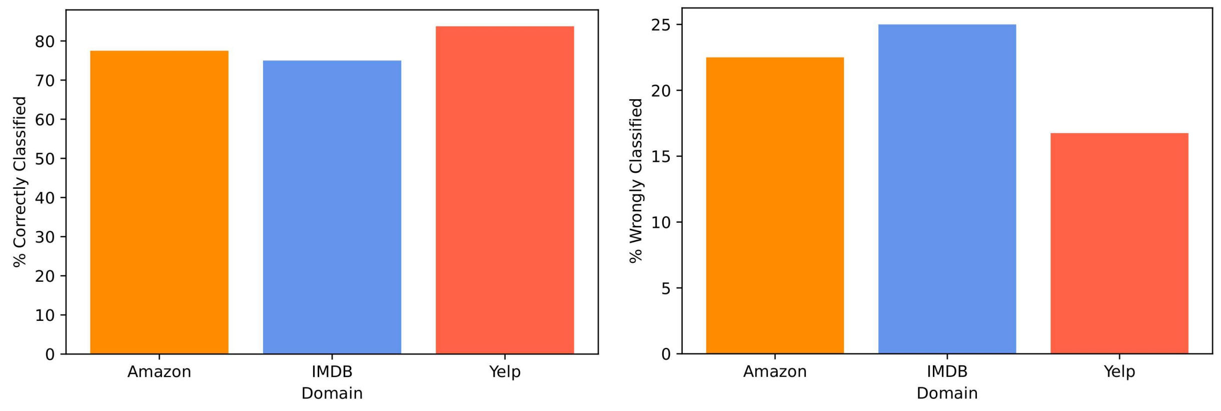


Figure 5.2 Performance of Best Model According to Source Domain - Amazon, IMDB, Yelp

6)

Below is the result of the performance based on the probabilities of the testing data submitted to the leaderboard on Gradescope.

Error Rate = 0.17667
Accuracy = 1 - Error Rate = 0.82333
AUROC = 0.89642

The AUROC indicates the area below the ROC curve such that a value closer to 1 means better model performance. In this case, the AUROC value is high which is considered pretty good, though not anywhere near an excellent model.

The accuracy of my model on this testing data is higher than when the model is being tested on the training and cross-validated data. This accuracy is expected, as as the model needs to evaluate the 600 reviews on the testing dataset that is trained on 2400 reviews on the training dataset; Which compared to the testing in the 5-fold cross-validation, the model was evaluating 480 that is trained on 1920 reviews. Thus, there's a possible correlation between the number of data being trained and the accuracy of the model performance on unknown testing data; That is, if the model has more data, in this case, more reviews, to learn and train from along with optimal hyperparameter regularization, the better the performance it will become. Another reason is that this is just down to pure luck such that the testing data is very similar to the training dataset, which is why the model was able to get decently good accuracy.

Overall, I believe that I was able to explore different models and evaluate relevant hyperparameters that can improve regularizations. Since we are looking at very different types of reviews from different domains, more experimentations would be needed to individually test each domains and as a whole.