

Textual Analysis of Movie Reviews

DS 3010 - Case Study 3 Report

Team 9: Abigail Albuquerque, Aria Yan, Isabel Herrero Estrada, Megan Sin, Sandra Phan

Problem 1:

We were tasked with writing a text classification pipeline to classify movie reviews as either positive or negative, finding a good set of parameters using grid search and evaluating the performance on a held out test set. To achieve this, we first downloaded the movie review dataset. We then split the dataset into a training set and test set. We then built a vectoriser/classifier pipeline that filtered out tokens that are too rare or too frequent using the Pipeline function. Then, a grid search was built to figure out if unigrams or bigrams were more useful. We found the mean and standard deviation for every candidate along with the parameters settings for all candidates explored by the grid search. We predicted the outcome on the test set. Lastly we printed the classification report and printed and plotted the confusion matrix. The classification report and confusion matrix can be found in figure 1.

| | | | | |
|--|-----------|--------|----------|---------|
| n_samples: 2000 | | | | |
| 0 params - {'vect_ngram_range': (1, 1)}; mean - 0.84; std - 0.02 | | | | |
| 1 params - {'vect_ngram_range': (1, 2)}; mean - 0.86; std - 0.02 | | | | |
| | precision | recall | f1-score | support |
| neg | 0.87 | 0.87 | 0.87 | 255 |
| pos | 0.87 | 0.87 | 0.87 | 245 |
| accuracy | | | 0.87 | 500 |
| macro avg | 0.87 | 0.87 | 0.87 | 500 |
| weighted avg | 0.87 | 0.87 | 0.87 | 500 |
| [[222 33] | | | | |
| [33 212]] | | | | |

Figure 1: The number of samples, mean and standard deviation for candidate followed by parameter settings, predicted outcome on tester, classification report and confusion matrix.

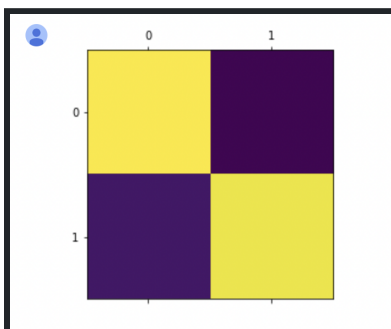
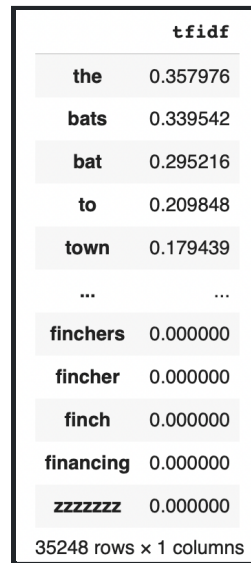


Figure 2: A plot of the confusion matrix.

Problem 2:

The goal of problem 2 was to explore the sci-kit learn Tfidf class. TF-IDF is a statistic that reflects how important a word is to a group of documents. We used TfidfVectorizer to plot words with their tfidf values.



| | tfidf |
|-----------|----------|
| the | 0.357976 |
| bats | 0.339542 |
| bat | 0.295216 |
| to | 0.209848 |
| town | 0.179439 |
| ... | ... |
| finchers | 0.000000 |
| fincher | 0.000000 |
| finch | 0.000000 |
| financing | 0.000000 |
| zzzzzzz | 0.000000 |

35248 rows x 1 columns

Figure 3: Plot of tfidf values using tfidfVectorizer with no parameters.

We then played around with the parameters of the TfidfVectorizer - specifically min_df, max_df and ngram_range.

- min_df - ignores terms that have a document frequency strictly lower than the given threshold when building vocabulary. The value is also called cut-off in literature. If float in range of [0.0, 1.0], the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None. The default value of min_df is 1.0. We chose a min_df value of 0.5 and the output is shown in figure 4 below.

| tfidf | |
|---------------------|----------|
| the | 0.638067 |
| to | 0.374039 |
| in | 0.264556 |
| of | 0.242187 |
| that | 0.190034 |
| ... | ... |
| two | 0.000000 |
| which | 0.000000 |
| much | 0.000000 |
| make | 0.000000 |
| its | 0.000000 |
| 78 rows × 1 columns | |

Figure 4: Plot of tfidf values using tfidf vectorizer with parameter $\text{min_df} = 0.5$.

- max_df : ignores terms that have a document frequency strictly higher than the given threshold. (corpus-specific stop words). If float in range $[0.0, 1.0]$, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None. The default value of max_df is 1.0. We chose a max_df of 2 and the output is shown in figure 5 below.

| tfidf | |
|------------------------|----------|
| phosphorus | 0.247094 |
| multiply | 0.247094 |
| cooling | 0.247094 |
| jacott | 0.247094 |
| omnivorous | 0.247094 |
| ... | ... |
| fittest | 0.000000 |
| fitchner | 0.000000 |
| fisted | 0.000000 |
| fission | 0.000000 |
| zzzzzz | 0.000000 |
| 19288 rows × 1 columns | |

Figure 5: Plot of tfidf values using tfidf vectorizer with parameter $\text{max_df} = 0.5$.

- `ngram_range`: It sets the lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $\min_n \leq n \leq \max_n$ will be used. For example an `ngram_range` of (1,1) means only unigrams, (1,2) means unigrams and bigrams, and (2, 2) means only bigrams. Only applies if the analyzer is not callable. The default ngram range is (1,1). We chose an `ngram_range` of (1,2) and the output is shown in Figure 6 below.

| tfidf | |
|------------------------|----------|
| coordinators | 0.423118 |
| katsulas | 0.423118 |
| pigeonholing | 0.400606 |
| romanticized | 0.400606 |
| overcoat | 0.400606 |
| ... | ... |
| fleshes | 0.000000 |
| flemish | 0.000000 |
| fleiss | 0.000000 |
| fleischer | 0.000000 |
| zzzzzzz | 0.000000 |
| 19323 rows x 1 columns | |

Figure 6: Tfidf values using `tfidfVectorizer` with parameter `ngram_range = (1,2)`

Problem 3:

The goal was to implement and understand machine learning algorithms. The parameters we picked for `tfidfVectorizer` were `max_df` set to 1 and `ngram_range` set to (1,2). We then computed "Xtrain", a Tf-idf-weighted document-term matrix using the transform function on `docs_train` and "Xtest", a Tf-idf-weighted document-term matrix using the transform function on `docs_test`. We then examined two classifiers - LinearSVC and KNeighborsClassifier. Linear Support Vector Classifier (SVC) applies a linear kernel function to perform classification. The KNeighbors classifier classified new data points based on the similarity measure of earlier data points. Both these classifiers work well with a larger number of data points. The two are working better as they are more refined algorithms. The LinearSVC seems to have a better performance than KNeighbors because it had lower numbers of false positives and false negatives combined and a more even percentage of corrected positives and negatives. KNeighbors is heavily skewed towards predicting negatives.

```

Xtrain Shape
(1500, 325236)
Xtest Shape
(500, 325236)
Y test Prediction 1 - LinearSVC
[[163  88]
 [ 55 194]]
Y test Prediction 2 - KNeighbors classifier
[[ 86 165]
 [ 31 218]]

```

Figure 7: Shows output for Xtrain, Xtest and Ytest prediction with LinearSVC and KNeighbors.

Problem 4:

The goal was to use a multi-layer perceptron to classify reviews and explore parameters for the MLP. Then, we would compare the accuracies against the baseline algorithms from Problem 1. We tried 2 different values for “hidden_layer_sizes” which were 50 and 100, and 4 different values for “activation” which were “identity”, “logistic”, “tanh” and “relu” and observed the accuracy.

Based on the previous confusion matrix, we observed the accuracies based on layer size and activation string. To know which layer size and activation string case has the accuracy, we looked at the ratio between true positives and negatives versus false positives and negatives since this comparison would tell us if the prediction were mainly accurate or not. From the log below, the layer size and activation string with most true positives and negatives was 50 and “relu”, meaning it had a high accuracy in predicted values that turned out to be actual. They also were the case with least false positives and negatives again proving to be most accurate in predictions.

| | Predicted 0 | Predicted 1 |
|--------------------|-----------------------|-----------------------|
| Actual 0 | TN | FP |
| Actual 1 | FN | TP |

Figure 8: Shows plot of confusion matrix.

```

Layer size: 50, Activation String: identity
[[215 40]
 [ 42 203]]
Layer size: 50, Activation String: logistic
[[213 42]
 [ 41 204]]
Layer size: 50, Activation String: tanh
[[215 40]
 [ 43 202]]
Layer size: 50, Activation String: relu
[[216 39]
 [ 41 204]]
Layer size: 100, Activation String: identity
[[215 40]
 [ 43 202]]
Layer size: 100, Activation String: logistic
[[214 41]
 [ 41 204]]
Layer size: 100, Activation String: tanh
[[215 40]
 [ 43 202]]
Layer size: 100, Activation String: relu
[[213 42]
 [ 43 202]]

```

Figure 9: Shows accuracy for different layer sizes and activation strings.

Problem 5:

The goal was to see how fast the algorithms are versus their accuracy. We compared the baseline algorithms to the runtime of the MLPClassifier. We tried 2 different values for “hidden_layer_sizes” which were 50 and 100, and 4 different values for “activation” which were “identity”, “logistic”, “tanh” and “relu” and observed the runtime.

```

Layer size: 50, Activation String: identity
1 loop, best of 5: 17.1 s per loop
100 loops, best of 5: 10.7 ms per loop
Layer size: 50, Activation String: logistic
1 loop, best of 5: 32.4 s per loop
100 loops, best of 5: 11.3 ms per loop
Layer size: 50, Activation String: tanh
1 loop, best of 5: 18.3 s per loop
100 loops, best of 5: 11.9 ms per loop
Layer size: 50, Activation String: relu
1 loop, best of 5: 18.3 s per loop
100 loops, best of 5: 11 ms per loop
Layer size: 100, Activation String: identity
1 loop, best of 5: 29.8 s per loop
10 loops, best of 5: 20.1 ms per loop
Layer size: 100, Activation String: logistic
1 loop, best of 5: 53.8 s per loop
10 loops, best of 5: 22.3 ms per loop
Layer size: 100, Activation String: tanh
1 loop, best of 5: 29 s per loop
10 loops, best of 5: 23.7 ms per loop
Layer size: 100, Activation String: relu
1 loop, best of 5: 28.5 s per loop
10 loops, best of 5: 20.1 ms per loop

```

Figure 10: Shows runtime for different layer sizes and activation strings.

Trying different values for "hidden_layer_sizes" showed us that the 50 had a much faster runtime than the value of 100 in the 4 different "activation" cases since 50 is a half the layer size of 100. Changing the options for "activation", we observed that "logistic" has a longer time than other "activation" values in the layer size of 50 and 100. The fit function consistently takes longer than the prediction function. This is because we use the fit function on our training set. The predict function uses the learned parameters by the fit function to perform predictions on new test data points. The training set is much bigger than the test set, thus making the fit function take longer than the prediction function.

Problem 6:

In a world ruled by technology and social media, we decided to propose a business product using a machine learning algorithm that could detect the sentiment or feeling of a tweet with high accuracy. You may wonder, what kind of business could you build around that. Well the answer is fairly simple. Information holds the key to unlocking any type of success. If our product were to be successful in identifying a positive or negative response within the audience's tweets we would also be able to have a product that is able to monitor distinct feelings about certain entertainment pieces directly from social media. Not only would this be useful for the customer to understand the public opinion on their work, but it would also provide useful information for future works and lead to a better understanding of what the spectator wants. Furthermore, customers would be able to know what their target audience should be based on this product ability of classifying reactions. This product would give an edge to companies amongst their rivals because it would push the boundaries to increase customer satisfaction, maintain a good public image and even optimize brand marketing.

Our competitors would be companies with SaaS tools like MonkeyLearn (28 employees, \$2.88 M in funding, \$5+M in revenue per annum), IBM Watson (5290+ companies using, 7000+ employees, \$1B+ valuation), Lexalytics (25-100 employees, 350K+funding), Amazon Comprehend(\$1B+ in revenue, 10000+ employees) among others. Our competitors are well-established companies.

The market for our product would be any company/organization/individual with a decently sized twitter following and even more specifically if they were working in the entertainment industry and depended on the public's eye opinion to further their career.

The model is one of your most important assets when developing machine learning. Because the model can grow to be somewhat large, you must take the time to consider how to handle it, and you must check that the models you build can properly remain on your targeted device. This can affect our business plan in a negative and positive way. Negatively, the entertainment industry is fast-changing and it's always important for businesses to keep up with social media and current views and trends. Although it is slow to train, we can make it so it increases the training time. There are a few factors to consider that can make the training time run faster such as changing the optimization algorithm and testing data structures. We can also take advantage of the fact that it is fast in making predictions on new data. In addition, it is always better to have a machine learning algorithm that takes slow time to train but makes fast, and most importantly, highly accurate predictions on new data, therefore affecting us in a positive manner because our product would have a small margin of error therefore making the slow training time worth it.

The cloud helps make machine learning more affordable. They provide cheap data storage as well as computational power. Since we will be working with real-time data and the need for computational power will vary depending on the traffic on the internet, the time of the day etc, using the cloud will be more beneficial and efficient than trying to set up our own servers.

References

Brownlee, J. (2019, August 6). *How to improve deep learning performance*. Machine Learning Mastery. Retrieved February 22, 2022, from <https://machinelearningmastery.com/improve-deep-learning-performance/>

Myriantous, G. (2021, March 9). *fit() vs predict() vs fit_predict in Python scikit-learn*. Towards Data Science. Retrieved February 21, 2022, from <https://towardsdatascience.com/fit-vs-predict-vs-fit-predict-in-python-scikit-learn-f15a34a8d39f>