1. Introduction

The goal is to build machine learning algorithms that predict the cooking time of recipes by using some features (recipe name, ingredients, number of steps, etc.) to conduct feature selection and training different models.

Since we are mainly dealing with words and not numerical values in the steps and ingredients, which can be easily categorised, preprocessing is pivotal as it improves classification accuracy (Uysal and Gunal, 2014). After doing this, there are various classification methods such as Naive Bayes, Linear Regression and Zero-R. The most common way of doing this is by tokenising, stemming, removing stopwords, vector representation transformation, feature selection and finally training the algorithm (M, S and V, 2021). Nargesian et al. (2017) found that feature engineering substantially improved the accuracy of their algorithm. They did this by reducing the features to a fixed array while maintaining the characteristics of the data.

In this task, Doc2Vec was implemented to engineer new features for training, along with multiple classification algorithms, for our dataset.

2. Methodology

2.1 Doc2Vec100

Doc2Vec is an unsupervised learning algorithm that numerically represents a document (Li, 2018). It incorporates Word2Vec, which uses a continuous Bag-of-Words algorithm (CBOW) and Skip-Gram algorithm (SG). In essence, it converts words into vector representations by taking into consideration the context they are used in, which is an algorithm that extends from CBOW. This version of Doc2Vec is known as the Distributed Bag of Words version (DBOW) (Naskar, 2019).

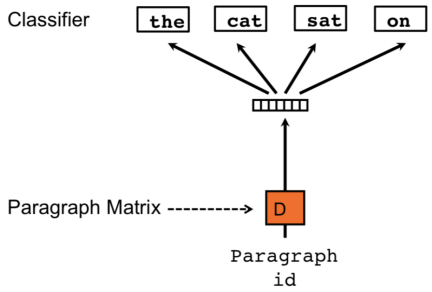

Figure 2.1 - Visual representation of Doc2Vec DBOW (Roy, 2020)

DBOW attempts to predict randomly sampled words from an input from a target word, contextualising the word. This creates a predicted probability distribution of words from the input, using the randomly sampled words (Li, 2018).

2.2 Training and Test Sets

The X-train and X-test sets used for this report both contain: number of steps, number of ingredients and the 300 features obtained via Doc2Vec100. 100 features were engineered from the ingredients, the steps, and the name each.

```
(X_train[0])

array([ 6.00000000e+00,  1.20000000e+01,  3.16286713e-01, -5.13782740e-01,
        2.36653447e-01, -4.30223584e-01, -2.44731635e-01, -3.56350452e-01,
        1.25789046e-01,  5.21478355e-01, -4.01333645e-02,  2.14347363e-01,
       -3.59670579e-04,  6.34366274e-01, -9.01492387e-02,  2.89968699e-01,
        4.60800707e-01, -1.06329329e-01, -4.66606915e-01,  4.80954386e-02,
```

Figure 2.2.1 - First 20 features of the first Instance in the X-train

The y-train set used contains the 40000 ground truth labels of the training data.

```
(y_train[0:5])

array([2., 2., 2., 2., 2.])
```

Figure 2.2.2 - First 5 instances of the y-training set

2.3 Classifiers

In this report, 5 different Machine Learning models were implemented; 4 supervised base classifiers from sklearn (Gaussian Naive Bayes (GNB), Decision Tree, Standardised Linear Support Vector Classifier (SVC), Logistic Regression) and 1 stacking classifier as a combination of the 4 base classifiers.

2.3.1 Gaussian Naive Bayes (GNB):

GNB is a variant of the well-known Naive Bayes classifier that assumes the data is gaussian/normally distributed. It also assumes that all features are independent of each other.

2.3.2 Decision Tree:

The decision tree learning algorithm is a model that utilises roots and nodes. It weighs possible outcomes against each other by mapping them out, starting from (usually) a single node and branching out to multiple possible nodes.

2.3.3 Standardised Linear Support Vector Classifier (SVC):

The SVC classifier that utilises hyperplanes in an N-dimensional space to classify data. These support vectors influence hyperplane positions, which assists in building a classifier. After standardisation, it gives each feature equal sensitivity, so that not only a portion of the features will have an influence, and therefore, search for an optimal hyperplane that classifies between right and wrong (Sotelo, 2017).

2.3.4 Logistic Regression:

The logistic regression model classifies data based on a conceptual probability (Subasi, 2019). It predicts the probability based on a decision boundary, which is set between 0 and 1. When compared to linear regression, the decision boundary is the biggest difference as the predicted value in a linear regression can exceed the 0 and 1 boundary (Pant, 2019).

2.4 Cross Validation

After training, each model was evaluated through cross validation to estimate the overall proficiency of the model. We implemented the k-folds cross validation, as it is known to result in a less biased model (M, 2018).

3. Evaluation

The accuracy for each of these models were deduced through cross validation with k = 10-fold. We first compile the predicted labels for each of the 10 validation sets to make up the 'predicted labels' for the entire training set and use that to compare that to the true labels, using the three multiclass evaluation methods; macro-averaging, micro-averaging, and weighted-averaging. This is then used to find the more in-depth evaluation metrics; precision, recall, and F1-score.

3.1 Accuracy:

```
GNB 0.607675
DecisionTree 0.5556
StandardisedLinearSVC 0.718575
LogisticRegression 0.723925
StackingClassifier 0.726625
```

Figure 3.1 - Accuracy values from the 5 models through cross validation, k = 10-fold.

The stacked model was found to have the highest accuracy amongst the 5 models, with the decision tree classifier having the lowest. The standardised linear SVC and logistic regression models were found to have only slightly lower accuracies to that of the stacked model.

3.2 Precision:

```
Macro Averaging Precision:
GNB: 0.5249709123791567
DecisionTree: 0.4211784594533096
StandardisedLinearSVC: 0.7199050977957403
LogisticRegression: 0.684373287301478
StackingClassifier: 0.7025155470839755

Micro Averaging Precision:
GNB: 0.607625
DecisionTree: 0.55825
StandardisedLinearSVC: 0.719225
LogisticRegression: 0.724125
StackingClassifier: 0.727525

Weighted Averaging Precision:
GNB: 0.6158412035825978
DecisionTree: 0.5609580388840013
StandardisedLinearSVC: 0.7195424207506288
LogisticRegression: 0.7207117864812979
StackingClassifier: 0.7253376634101197
```

Figure 3.2 - Precision values from the 5 models through cross validation, k = 10-fold, for each averaging method.

Amongst the 5 models, the standardised linear SVC model was found to have the highest precision under macro-averaging and the stacked model was found to have the highest under both micro and weighted averaging. The stacked and logistic regression models were more likely to correctly predict a positive case for a duration label under micro-averaging, the decision tree and GNB models under weighted-averaging, and, while similar across all three methods, the standardised linear SVC classifier had its highest precision value under macro-averaging.

3.3 Recall:

```
Macro Averaging Recall:
GNB: 0.5371557233497738
DecisionTree: 0.4223069223437424
StandardisedLinearSVC: 0.5565846795050462
LogisticRegression: 0.5930495640769664
StackingClassifier: 0.6039317386356315

Micro Averaging Recall:
GNB: 0.607625
DecisionTree: 0.55825
StandardisedLinearSVC: 0.719225
LogisticRegression: 0.724125
StackingClassifier: 0.727525

Weighted Averaging Recall:
GNB: 0.607625
DecisionTree: 0.55825
StandardisedLinearSVC: 0.719225
LogisticRegression: 0.724125
StackingClassifier: 0.727525
```

Figure 3.3 - Recall values from the 5 models through cross validation, k = 10-fold, for each averaging method.

For all three methods, the stacked model had the highest recall value. It can be noted that for both micro and weighted averaging the recall values for each of the models were the same, meaning that for each model both methods were equally likely to detect true positives in the dataset. It can also be observed that for each model macro-averaging was much less likely than the other two methods to detect the true positives.

3.4 F1-Score:

```
Macro Averaging F1-Score:
GNB: 0.5309934252483707
DecisionTree: 0.42174193603749194
StandardisedLinearSVC: 0.6277968774304952
LogisticRegression: 0.6354470319081859
StackingClassifier: 0.6495041023186685

Micro Averaging F1-Score:
GNB: 0.607625
DecisionTree: 0.55825
StandardisedLinearSVC: 0.719225
LogisticRegression: 0.724125
StackingClassifier: 0.7275250000000001

Weighted Averaging F1-Score:
GNB: 0.6117055137790134
DecisionTree: 0.5596007432527925
StandardisedLinearSVC: 0.7193836753606444
LogisticRegression: 0.7224143616342305
StackingClassifier: 0.7264296851484108
```

Figure 3.4 - F1-Score values found the 5 models through cross validation, k = 10-fold, for each averaging method.

The F1-score values produced for each method had similar behaviours to that of the recall results obtained above, except that the micro and weighted averaging F1-scores were not the same, but were still similar to each other. This means that the micro and weighted averaging models had higher ‘harmonic’ averages of their precision and recall compared to the macro-averaging models.

3.5 Overall:

It was found that each model under **weighted-averaging** had precision, recall, and F1-score values very **similar** to each other as well as their accuracy. It was also found that the resulting values of precision, recall, and F1-score under **micro-averaging** for each model are **exactly the same**.

Majority of the values for the 3 in-depth metrics are significantly lower under macro-averaging in comparison to the other two. The decision tree classifier consistently performed the worst under all 3 multiclass evaluation methods, followed by the GNB classifier. The stacked model proved to have the highest value for each metric under each method except for macro-averaging precision, with the logistic regression classifier being the second leading model by a relatively small difference.

4. Discussion

4.1 Performance

4.1.1 Decision Tree:

The decision tree classifier's poor fit to the dataset is likely to have been overfitted. This is due to the features being continuous with a lot of decimal places, making test instances highly likely to have attribute value pairings that have yet to be seen. Together with the high number of features being implemented (with many likely being irrelevant) that elongates the height of the tree and there being 40,000 training instances as the number of leaf nodes, this combination of factors is what led to the poor performance on unobserved instances.

4.1.2 GNB:

The Naive Bayes classifier only excels on datasets with independent features as it assumes conditional independence between all the attributes. However, with there being 302 features it is unlikely to be true, resulting in a poor fit. For example, the first two features implemented, number of steps and number of ingredients, for a given recipe are not guaranteed to be independent of each other. This is further backed up by the fact that the data of the 300 features from Doc2Vec100 for the training set follow a Gaussian distribution, considering the low values in the evaluation metrics. This was shown by randomly plotting 25 out of the 300 features by value against frequency.

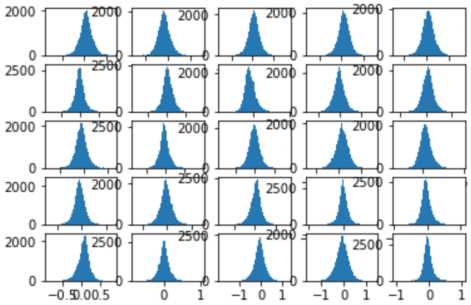

Figure 4.1.2 - 25 of the 300 features from Doc2Vec100 for the training set (value against frequency).

4.1.3 Stacked Model:

The stacked model aims to reduce the errors over its base classifiers which each make different errors, meaning that the more the base classifiers' errors vary from each other the higher the performance of the stacked model. This is because each model will have errors where the other models do not, therefore, increasing the overall stacked performance.

Both the standardised linear SVC and logistic regression classifiers often make errors when the decision boundaries for classification on the data are not linear, as they assume linearity. This may explain why the two models perform similarly to each other (note that while decision trees and GNB classifiers do not assume the same, they have lower performances, which may explain some linearity in the data). Both are less likely to overfit compared to the decision tree classifier and also do not assume conditional independence on the data's features like GNB

does. This all explains the insignificant performance increase between the stacked model and the second best model (logistic regression). While the decision tree and GNB classifiers do not assume linearity, which the dataset is shown to likely have, the standardised linear SVC and logistic regression models both cover their errors and are near indistinguishable in terms of performance and errors.

4.2 Multiclass Evaluation Methods:

4.2.1 Macro-Averaging - Low Performance:

Macro-averaging can seem unreliable as it just takes the average of the classes for the three in-depth evaluation metrics, especially in this case where there are only three possible class labels. The performance worsens when there is a great imbalance of proportions between the class populations. It can be found that the number of instances that actually are of duration label 3 in the training data is almost a tenth of that of duration label 2.

```
Class 1 has 17705 instances
Class 2 has 20246 instances
Class 3 has 2049 instances
```

Figure 4.2.2 - Number of instances for each duration label in the training data.

This explains the low performance of the models under macro-averaging compared to the other two methods.

4.2.2 Micro-Averaging - Same Values for In-Depth Metrics:

Since micro-averaging combines all instances over all classes into one, the only way for a model's precision, recall, and F1-score to be the same is for the data to have the same number of false positives and false negatives, collectively over all classes. This was proven for each of the 5 models by comparing their every predicted label to the ground truth:

```
GNB has 15695 false negatives and 15695 false positives
DecisionTree has 17670 false negatives and 17670 false positives
StandardisedLinearSVC has 11231 false negatives and 11231 false positives
LogisticRegression has 11035 false negatives and 11035 false positives
StackingClassifier has 10899 false negatives and 10899 false positives
```

Figure 4.2.2 - Results for number of false positives and false negatives for each model on the training data.

4.2.3 Micro and Weighted Averaging - Same Values for Recall:

The formula for recall on a model is calculated as: TP/(TP + FN).

Micro-averaging recall calculates the sum of the TP_i's and $(TP_i + FN_i)$'s of the classes (i = 1,2,3) separately then uses those to form a 'stacked' recall formula; $(\sum TP_i)/(\sum TP_i + FN_i)$.
Note that $(TP_i + FN_i)$ for a given class, i, is the number of instances in the i^{th} row of a model's confusion matrix. This means that $\sum TP_i + FN_i$ results in the total number of instances, N.
Therefore, this gives the formula: $(\sum TP_i) / N$.

Weighted-averaging recall first calculates the recall formula then multiplies it according to the class' number of instances, n_i, for a given class, i, with the formula: $\sum (n_i/N)*(TP_i/(TP_i + FN_i))$. Note that $TP_i + FN_i$ for a given class, i, is the number of instances of that class, n_i. This then gives the formula: $\sum (TP_i/N) = (\sum TP_i) / N$, and, therefore, explaining why they have the same values for all models.

	Predicted		
Actual	1	2	3
1	TP	FN	FN
2	FP	TN	TN
3	FP	TN	TN

Figure 4.2.3 - General confusion matrix for the training set to guide the explanation.

5. Error Analysis

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train, pred_y_train['GNB'], labels = [1,2,3])
```

```
array([[12062,  5141,   502],
       [ 7643, 11501,  1102],
       [  774,   533,   742]])
```

```
confusion_matrix(y_train, pred_y_train['DecisionTree'], labels = [1,2,3])
```

```
array([[ 9953,  6846,   906],
       [ 6943, 12164,  1139],
       [  831,  1005,   213]])
```

```
confusion_matrix(y_train, pred_y_train['StandardisedLinearSVC'], labels = [1,2,3])
```

```
array([[13128,  4522,    55],
       [ 4879, 15286,    81],
       [  658,  1036,   355]])
```

```
confusion_matrix(y_train, pred_y_train['LogisticRegression'], labels = [1,2,3])
```

```
array([[13024,  4541,   140],
       [ 4640, 15357,   249],
       [  549,   916,   584]])
```

```
confusion_matrix(y_train, pred_y_train['StackingClassifier'], labels = [1,2,3])
```

```
array([[12969,  4622,   114],
       [ 4525, 15488,   233],
       [  508,   897,   644]])
```

Figure 5 - Confusion matrix for each classifier.

From the results in Figure 5, the class label “2” has the highest false negatives for all classifiers. This could be due to the fact that class 2 has the most number of instances (Figure 4.2.2). Having too many instances causes Doc2Vec to create a large dimensional space, which in return will contain many 0 values. Furthermore, since Doc2Vec samples from the input, this could represent 2 steps similarly, despite their different meanings. For example, the sentences “take out the pasta and add water” and “take out the water and add pasta” would have extremely similar vector representations in the dimensional space even if they have different duration labels.

This idea is further supported by looking at class label “3” as it has the lowest false negatives for all classifiers as well as the least number of instances in the training set. This creates a smaller dimensional space (if class label “3’ was isolated), increasing the overall performance of the Doc2Vec algorithm. This shows a tradeoff between performance and variety; a larger dimensional space will have lower performance but a larger variety of words and vice versa.

REFERENCES

Li, S., 2018. *Multi-Class Text Classification with Doc2Vec & Logistic Regression*. [online] towards data science. Available at: <https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4> [Accessed 19 May 2021].

M, I., S, K. and V, T., 2021. Text Classification Using Machine Learning Techniques. *WSEAS TRANSACTIONS on COMPUTERS*, 4(8), pp.966-974.

M, S., 2018. *Why and how to Cross Validate a Model?*. [online] towards data science. Available at: <https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f> [Accessed 19 May 2021].

Majumder, B. P., Li, S., Ni, J. & McAuley, J. Generating personalized recipes from historical user preferences. [online] Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019.

Nargesian, F., Samulowitz, H., Khurana, U., B.Khalil, E. and Turaga, D., 2017. Learning Feature Engineering for Classification.

Naskar, A., 2019. *Simple: Doc2Vec explained*. [online] ThinkInfi. Available at: <https://thinkinfi.com/simple-doc2vec-explained/> [Accessed 19 May 2021].

Pant, A., 2019. *Introduction to Logistic Regression*. [online] towards data science. Available at: <https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148> [Accessed 19 May 2021].

Roy, A., 2020. *Getting Started with Doc2Vec*. [online] Heartbeat. Available at: <https://heartbeat.fritz.ai/getting-started-with-doc2vec-2645e3e9f137> [Accessed 19 May 2021].

Sotelo, D., 2017. *Effect of Feature Standardization on Linear Support Vector Machines*. [online] towards data science. Available at: <https://towardsdatascience.com/effect-of-feature-standardization-on-linear-support-vector-machines-13213765b812> [Accessed 19 May 2021].

Subasi, C., 2019. *LOGISTIC REGRESSION CLASSIFIER*. [online] towards data science. Available at: <https://towardsdatascience.com/logistic-regression-classifier-8583e0c3cf9> [Accessed 19 May 2021].

Uysal, A. and Gunal, S., 2014. The impact of preprocessing on text classification. *Information Processing & Management*, 50(1), pp.104-112.