Predicting Cooking Times for Recipes
COMP30027 Report

Anonymous

1. Introduction

With the current global COVID-19 pandemic, staying home and learning how to cook from your home has become a necessity. As the number of online recipes becomes more abundant, knowing the time required to complete them is and has always been important for home stayers, new and experienced cooks. This report will firstly aim to explore machine learning models created with functions from Scikit-learn and secondly, explore selected and engineered features to best predict the three cooking times: quick (1.0), medium (2.0) or slow (3.0) of recipes in the data collected from Majumder et al. (2019).

2. Related Work

There have been similar pre-processing approaches for the features of recipes such as MILK (Minimal Instruction Language for the Kitchen) which aims to create a representative set of actions required in making a recipe (Tassie & Smith, 2008). In this language, there are three primitive types, "ingredient", "tool" and "string" that are combined to express a state. Each state represents a recipe step. Interestingly, this pre-processing method was also used in many other recipe processing methods like SIMMR (Simplified Ingredient Merging Map in Recipes), a representation that aims to capture the flow of ingredients (Jermsurawong & Habash, 2017). In SIMMR, MILK was used to construct a database of SIMMR trees using CURD, a MILK language recipe database (Tassie & Smith 2008). In this report, we will similarly attempt to extract and pre-process ingredients and instructions and use them to train our models.

3. Experimental Setup

3.1 Data

The dataset consists of 40,000 labelled instances. Using cross-validation, a data resampling method to assess the generalization ability of predictive models and to prevent overfitting (Hastie et al., 2021; Duda, Hart, Stork, 2007). The data is split into k or 5 folds in this case, where k-1 (80%) partitions are allocated to the training set, and one partition (20%) is allocated to the testing set. This is repeated until each of the k subsets has served as the testing or validation set (Berrar, 2019). Cross validation is the preferred method as our data contains many instances. Other methods like leave-one-out cross validation is computationally expensive as it uses N many partitions.

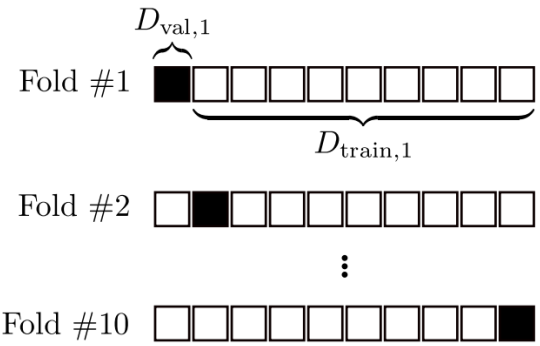

Figure 1- 10-fold cross validation, where each validation set is used exactly once. In each fold, one partition is the validation set, and the rest serves as our training data. (Berrar, 2019)

3.2 Evaluation Metrics

The overall performance of each model is evaluated by calculating the average accuracy over 5 folds. The accuracy per fold is calculated by comparing the model's predicted class labels with the truth labels.

4. Feature Selection

To explore the most effective features for predicting the cooking time, different features were chosen for training. This reduces input size, lowers the computational cost and overfitting chances. Additionally, this can improve the performance of each model if the correct sub features are chosen.

5. Models

Three models were trained with different features as shown below.

Model	Used features
0-R	Truth labels
Decision Trees (Entropy/ Information Gain)	Number of steps, Number of Ingredients
KNN Classifiers	Name, Steps, Ingredients (Separately)

Table 1- Table showing the models and corresponding features used accordingly.

5.1 0-R Model

The 0-R model serves as a baseline for many classification problems. This algorithm finds the majority class of the training set and uses it to predict the test instances, regardless of their features. In the training set, the majority class was "2.0" and an overall average accuracy of 0.5062 was achieved.

5.2 Decision Trees

Decision trees uses a tree-like structure to map all the possible outcomes with decision rules based on the features used. With Scikit-learn's DecisionTreeClassifier, the two decision trees were created with a maximum depth of 3 (to prevent overfitting and improve performance). In addition, entropy and information gain were used as a measure of split quality, the higher the information gain the better. Where entropy is defined as the measure of the disorder of the data and calculated for each split by iterating over all possible values of $\vec{y}$,

$$Entropy(\vec{y}) = -\sum_{j=1}^{n} \frac{|y_j|}{|\vec{y}|} \log \frac{|y_j|}{|\vec{y}|}$$

the conditional entropy is defined as

$$Entropy(j|\vec{y}) = \frac{|y_j|}{|\vec{y}|} \log \frac{|y_j|}{|\vec{y}|}$$

and the information gain is calculated by the formula

$$Gain(\vec{y}, j) = Entropy(\vec{y} - Entropy(j|\vec{y}))$$

(Korting, 2006). The first decision tree was trained based on the number of steps and ingredients, whereas the second used feature engineering by calculating the product of the two selected features. These two features were manually selected because it is expected that as the number of steps or ingredients increases, so does the complexity of the recipe, and hence the time taken.

Decision tree features	Average Accuracy
Number of steps & Number of ingredients	0.6227
Number of steps * Number of ingredients	0.6395

Table 2- Table showing the average accuracies for non-engineered and engineered features.

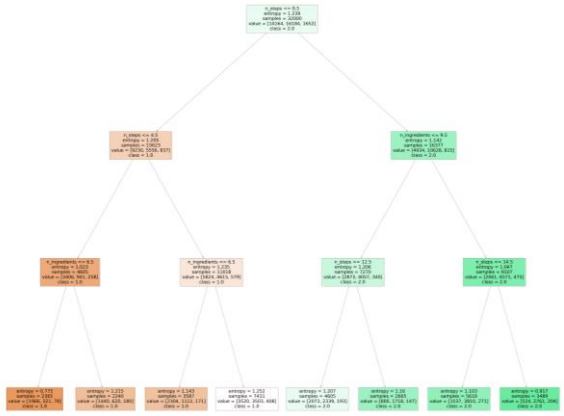

Figure 2- Structure of the 5th fold decision tree classifier trained using the number of steps and ingredients.

As a result, the second decision tree yielded an average accuracy of 0.6359, a small increase from non-engineered features and 13.3% increase in accuracy compared to the 0-R model. This shows that with feature engineering, the right input data can be prepared to improve the performance of various machine learning models.

5.3 K-Nearest Neighbour Classifier

K-Nearest Neighbour (k-NN) is an instance-based learning method which calculates the distance between the test set with the fitted training set. In this model, the Euclidean distance is calculated between each test vector against all the training vectors. These vectors were converted from text using CountVectorizer. The corpus was trained on the training set of each k-fold. Therefore, each vector instance contains the frequency of each appeared word from all the possible words in the training set. The list of words was not pre-processed beforehand as CountVectorizer removes stop words and punctuation. The vector was also not standardised as all the words are frequency based.

To compare the effectiveness of each feature, the same set values of k were used.

Feature	Average Accuracy for k-NN		
	3	5	7
Name	0.6408	0.6560	0.6635
Steps	0.6880	0.6937	0.6921
Ingredients	0.6008	0.6084	0.6113

Table 3- Table showing the average accuracies for different values of k and features used.

As shown in Table 3, the most effective feature in predicting recipe times were the steps. On average 7-NN produced the best accuracy as a larger k is less affected by noise in the data and will therefore produce a smoother boundary between clusters.

6 Result and Error Analysis

From the models, the best accuracy of 0.6937 was achieved with 5-NN using the list of steps. In k-NN, other features did not perform as well since the recipe name and ingredients were not as informative. The name does not tell an individual how much preparation time is needed if they are unfamiliar with the recipe. Similarly, the ingredients list is less informative, as a quick or slow recipe can contain several identical ingredients. An example would be a recipe named “my slow cooked beans” and “spicy pecans with extra spicy lemon pepper seasonings”. These two recipes include “brown sugar”; however, the former recipe is a slow recipe, and the latter is a quick recipe. This is very vague for the model to infer any useful information to predict the cooking time.

Furthermore, both decision tree models did not perform well due to imbalanced data (shown in Table 4), they were not able to classify recipes with the label 3.0.

Class Label	Count	Percentage of total data
1.0	17705	44.26%
2.0	20246	50.62%
3.0	2049	5.12%

Table 4- Table illustrating the imbalance of data with the occurrences of instances with the corresponding class labels

Instead, all the instances were either classified as 1.0 or 2.0. This yielded a better-than-expected accuracy as the combined occurrence of labels 1.0 and 2.0 were 94% of the total data.

7 Improvements

Despite achieving reasonable accuracies, several changes could be made to improve the prediction of cooking times. As the models were trained with an imbalanced training set, under-sampling could be used. Under-sampling is a technique from Imbalanced-learn (a python toolbox) used to reduce the number of instances in the majority class (Lemaître, 2017). If this was used, the decision trees should be capable of predicting all three classes and achieve better accuracies.

Moreover, instead of directly using the frequency of each word in k-NN. The text could have been stemmed beforehand. An example would be transforming the words “cooked” and “cooking” to “cook” as they possess the same meaning, which can improve the accuracy of text-based features. Despite these proposed changes, CountVectorizer has many limitations. It is uncapable of differentiating between word importance since all words have equal weight. Also, it considers the words in the corpus as the most significant words (Saket, 2020), this affects the prediction of test instances with unseen words. Instead of using CountVectorizer, TF-IDF could be used instead as it considers word importance.

8 Conclusion

As shown in the results, these models were able to predict the cooking time for given recipes to an extent. Some models performed worse than others due to data imbalance and lack of pre-processing, and some features were less informative than others, such as the ingredients list compared to the steps. In conclusion, the model and feature shown to be the most effective was 5-NN and steps. With the proposed improvements, these models could predict cooking times more accurately.

9 References

Berrar, D. (2019). Cross-validation. *Encyclopedia of bioinformatics and computational biology*, *1*, 542-545.

Duda R.O., Hart P.E., & Stork D.G., Pattern Classification, New York: John Wiley & Sons, 2001, pp. xx + 654, ISBN: 0–471-05669-3. (2007). *Journal of Classification*, *24*(2), 305–307. https://doi.org/10.1007/s00357-007-0015-9

Hastie, T., Tibshirani, R., & Friedman, J. (2021). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)* (2nd ed.). Springer.

Jermsurawong, J., & Habash, N. (2015, September). Predicting the structure of cooking recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 781-786).

Korting, T. S. (2006). C4. 5 algorithm and multivariate decision trees. *Image Processing Division, National Institute for Space Research–INPE Sao Jose dos Campos–SP, Brazil*.

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, *18*(1), 559-563.

Majumder, B. P., Li, S., Ni, J., & McAuley, J. (2019). Generating personalized recipes from historical user preferences. *arXiv preprint arXiv:1909.00105*.

Saket, S. (2020). *Count Vectorizer vs TFIDF Vectorizer | Natural Language Processing*. Linkedin. https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket?trk=public_profile_article_view#:%7E:text=But%20its%20major%20disadvantages%20are,as%20linguistic%20similarity%20between%20words.

Tasse, D., & Smith, N. A. (2008). SOUR CREAM: Toward semantic processing of recipes. *Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-LTI-08-005*.