COMP30027 Machine Learning Report
Project 2 : How Long Will It Take To Cook This?

1. Introduction

Natural Language Processing (NLP) has always been a challenging problem in machine learning. It requires “a computer to understand, analyze, manipulate and potentially generate human language” (Shetty, 2018). It has a wide range of applications in real life such as the information retrieval in Google search, the spam filter in Gmail. ***Food.com*** is a recipe website that allows users to view food-making online. In this project, I will be implementing classification techniques to predict the cook time for recipes from ***Food*** website and evaluate these models’ performances. We aim to build and analyze some supervised Machine Learning models with high accuracy. LinearSVC will be focused mainly, and DecisionTree is compared regarding to our task.

2. Dataset Characteristics

The dataset I used for training and testing was kindly provided by ***Recipes, Easy Dinners And Meal Ideas -Food.com***. A recipe train dataset, with 40000 rows of data. Each row of data is corresponding to a Food Recipes information(see Table 1). The information consists of 6 features, which are {name, n_steps, n_ingredients, steps, ingredients, duration_label}. The class label is “***duration_label***”.

Dataset	Features
recipe_train.csv	name, n_steps, n_ingredients, steps, ingredients, duration_label
recipe_test.csv	name, n_steps, n_ingredients, steps, ingredients,

Table 1- Dataset document and features inside

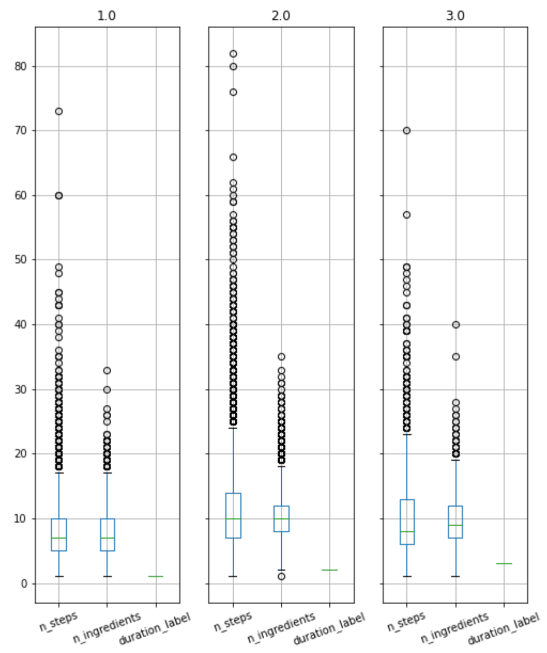


Figure 1- The distribution of each duration

These features over different duration in the Figure 1 seems balance, but there is a significant difference among the distribution of these features, so I decide ***“steps”*** for my later training process.

3. Data Preprocessing

Because of the particularity of the dataset, before training the model, I did some preprocessing to the raw data. I try to rebalance the dataset using oversampling and under-sampling methods. But both over-fit and under-fit the data so much. So, I abandoned these methods, just removed the punctuations and numbers in the column ***"steps"***

4. Model Selection

4.1 Discussion

1. LinearSVC
2. DecisionTreeClassifier

4.2 Procedure

For each model, it is trained using the recipe data extracted from ***"recipe_train.csv".***

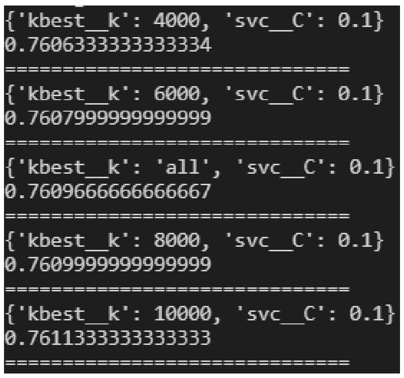
To make the data more precise and let training performance much better, I used ***"TfidfVectorizer"*** to transform the column ***"steps"*** and it did increase overall accuracy in the predictions, I will discuss more in the later page.

The classifier is tested in "***grid_search"*** validating function and the output accuracy are the changing ***"C"*** in "***LinearSVC(svc)"*** and ***"max_depth"*** in "***DecisionTreeClassifier (tree)".*** Their performances and behaviours will be evaluated separately regarding the choice of the parameters.

5. Discussion

5.1 LinearSVC – C

Combined with "***SelectKBest"*** : in my pipeline, I changed ***K*** in "***SelectKBest"*** function and ***C*** in ***"LinearSVC"*** function and the result after using "***grid_search"*** are displayed below(see Figure 2)



```
{'kbest__k': 4000, 'svc__C': 0.1}
0.7606333333333334
===============================
{'kbest__k': 6000, 'svc__C': 0.1}
0.7607999999999999
===============================
{'kbest__k': 'all', 'svc__C': 0.1}
0.7609666666666667
===============================
{'kbest__k': 8000, 'svc__C': 0.1}
0.7609999999999999
===============================
{'kbest__k': 10000, 'svc__C': 0.1}
0.7611333333333333
===============================
```

Figure 2- Top 5 result for LinearSVC

Also, I drew a diagram by "Validate Curve", we could see there will be an overfitting situation when C gets larger, And the train score would go down. So, 0.1 is my best parameter C value (see Figure3)

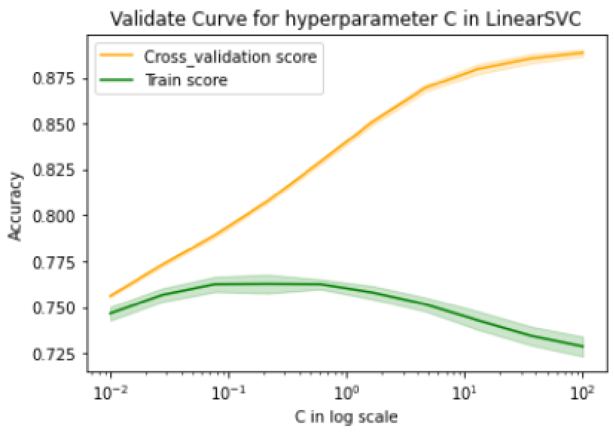
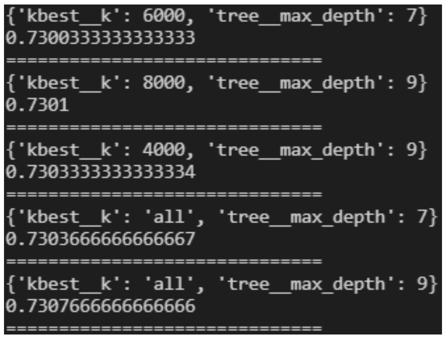


Figure 3- Validate Curve for LinearSVC

5.2 DecisionTree – Max_depth

I use same method in ***"DecisionTree"*** classifier, and just changed the ***"max_depth"*** to see how it goes in the result, and to compare to ***"LinearSVC"***. (see Figure 4)



```
{'kbest__k': 6000, 'tree__max_depth': 7}
0.7300333333333333
==============================
{'kbest__k': 8000, 'tree__max_depth': 9}
0.7301
==============================
{'kbest__k': 4000, 'tree__max_depth': 9}
0.7303333333333334
==============================
{'kbest__k': 'all', 'tree__max_depth': 7}
0.7303666666666667
==============================
{'kbest__k': 'all', 'tree__max_depth': 9}
0.7307666666666666
==============================
```

Figure 4- Top 5 result for DecisionTree

I drew a ***"Learning Curve"***, which also shows an overfitting situation when ***"max_depth"*** has really low value. The train score will go higher and the result will be more reasonable (see Figure 5)

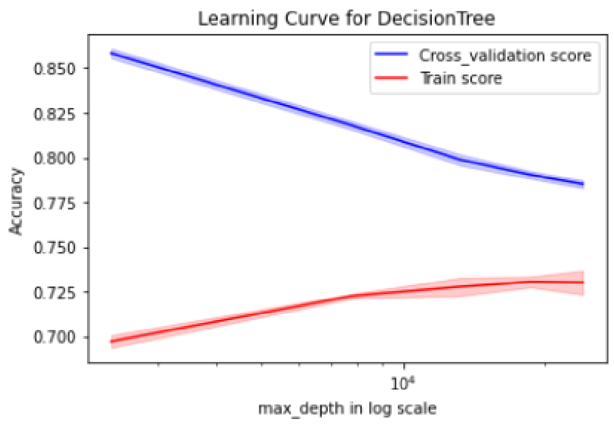


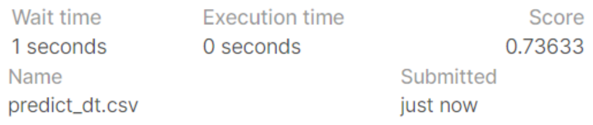
Figure 5- Learning Curve for DecisionTree

5.2 SelectBestK – K

Compared to Figure 2 and Figure 4, we can see the largest ***"K"*** is not the best choice at all, but overall, higher ***"K"*** is performing better than the other.

6. Performance on Kaggle

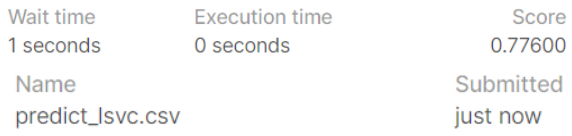
For my "predict_dt.csv", it has lower score(see Figure 6) when perform on the test data while "predict_lsvc.csv" has better result on the site(see Figure 7). And the scores has no much different to my local testing, (see Figure 2 and 4)



Wait time	Execution time	Score
1 seconds	0 seconds	0.73633

Name	Submitted
predict_dt.csv	just now

Figure 6- Results on Kaggle(DecisionTree)



Wait time	Execution time	Score
1 seconds	0 seconds	0.77600

Name	Submitted
predict_lsvc.csv	just now

Figure 6- Results on Kaggle(LinearSVC)

7. Conclusions

In this project where each duration needs to be classified into three different labels, I found out ***"LinearSVC"*** classifier performs the better than ***"DecisionTree"***. The underlying ***"TFIDF"*** model fit with the assumption of the data. On the other hand, the limitation of the ***"C"*** implementation to restrict the improvement on the performance. In general, ***"LinearSVC"*** model is a decent classifier for text classification that is intuitive and easy to implement.

Second, I have implemented several ways of preprocessing the data to make data more precise and easier to be trained and introduce 2 models, I experimented with several classification algorithms to predict duration from recipe text, and mainly using the ***"steps"*** column which contains the most useful content to predict the duration time in the results. They all give reasonable results.

Thirdly, I carry out learning curves on the process of comparing parameters setting for DecisionTreeClassifier model to tune the performance and to control the model complexity. For LinearSVC model, I use Validate Curve for comparing the parameters. Finally, I evaluate each model, analyze its performance by using the grid_search. As a result, the justified classifiers with best parameters in the arguments setting , outperforms any other results by initial value in the setting or the default setting when import classifiers. Because it combines the knowledge from different spaces of the data, and trained with preprocessed data.

9. Future Improvement

In my case, I can see my result is in a pretty good performance in Kaggle when comparing to the other people. But if I want to make improvement for my outcome, I may need to use bag-of-words in the training state, and implement more preprocess when collecting the training data at the beginning. Also, using stacking classifiers and change more parameters in each model will be a better way to get better accuracy score.

8. References

Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni & Julian McAuley, Generating Personalized Recipes from Historical User Preferences. in 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

G. Salton & M. J. McGill., Introduction to modern information retrieval. 1983

H. Wu, R. Luk, K. Wong & K. Kwok., Interpreting TF-IDF term weights as making relevance decisions". ACM Transactions on Information Systems. 2008.

T. Hastie, R. Tibshirani & J. Friedman. Elements of Statistical learning, Springer, 2009.

Pedregosa *et al.*, Scikit-learn: Machine Learning in Python https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.htmledu) JMLR 12, pp. 2825-2830, 2011