COMP30027 Machine Learning - Project 2

How Long Does It Take To Cook This?

Anonymous Group of 2
Word Count: 2496 (Excluding Tables, Figures and References)

May 21, 2021

Introduction

This project aims to detect how long a recipe takes to reach completion, in three possible numeric duration levels. This is a popular sub-problem of **culinary prediction**, with numerous articles investigating on how factors affect the outcome or cooking complexity of a dish [1, 2, 3, 4].

Through experimenting, implementation has proven efficiency in providing highly accurate outcomes, by successively incorporating **Natural Language Processing (NLP)** techniques and self-developed method **Näive Keyword Assign (NKA)** into Feature Engineering, as well as finding optimal percentile for feature selection on a combination set of multiple features. Effectiveness is shown through Kaggle, where implementation results are within top percentile. Further hyperparameter tuning is also carried out to investigate its effect on stacking models.

Data Investigation

At first glance, datasets have been well-processed, having no missing values, with all string values in lowercase. Training dataset from `recipe_train.csv` consists of the following attribute types:

- Numerical: number of steps (`n_steps`), number of ingredients (`n_ingredients`).
- Strings: `name`, `ingredients` and `steps`.
- Categorizer: `duration_label`, in the form of three levels 1.0, 2.0, 3.0.

For string features, preprocessing is needed. Two precomputed and provided methods for the problem are:

- `CountVectorizer`: converts document to counts representing word frequencies.
- `doc2vec`: converts document to high-dimensional vector space such that words in similiar context are close to each other. Datasets with 50 and 100 dimensional vector spaces were provided.

Visualisation

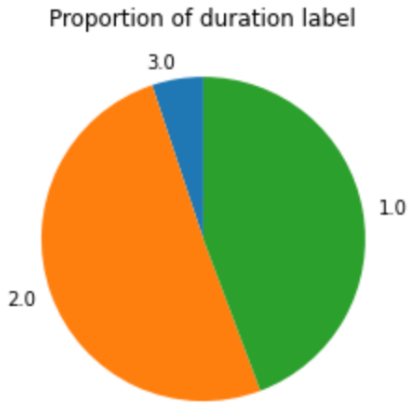

Figure 1: Pie chart of `duration_label`

Proportion of duration label, shown in Figure 1, illustrates that Label 3 instances comprise noticeably small percentage of data compared to Labels 1 and 2. This should be kept in mind when building classifier as model tend to predict towards Label 1 and 2 more, causing bias in prediction process.

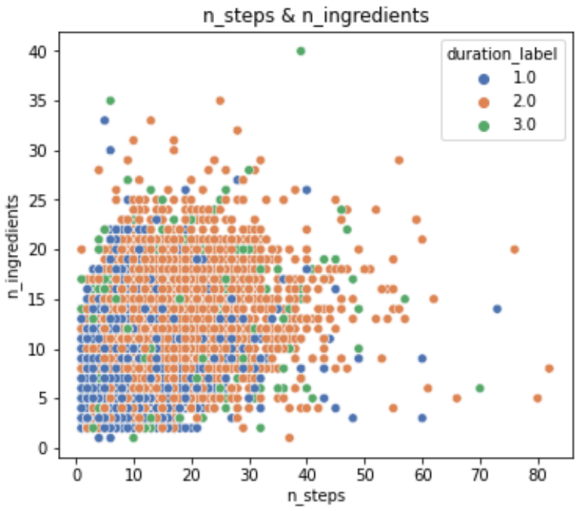

Figure 2: Scatterplot of `n_steps` and `n_ingredients`

Furthermore, Figure 2 indicates while Label 1 and Label 2 has dense and recognizable clusters, recipes with Label 3 are sparsely scattered across the plot. This shows that while using `n_steps` and `n_ingredients` is sufficient to distinguish between Labels 1 & 2 at hand, being able to detect Label 3 recipes will require further improvement.

Text Feature Evaluation & Selection

Since each pre-computed text feature is fairly large, it would be computationally expensive to train on all features. Having too many features can affect model accuracy tremendously due to increasing variance from **The curse of Dimensionality** [5]. Therefore, for this project, only most correlated text feature is chosen.

From background knowledge, `steps` is speculated to tell most information about cooking time. It contains detailed instructions i.e. duration required for each step, verbs describing step. On the other hand, `name` and `ingredients` are also expected to slightly help in prediction in specific cases, given that ingredients can have different preparation duration, while name can contain nouns of dish types which also has different cooking time ranges [1, 2].

By training on different features, accuracy can imply how much information stored in predicting classifiers and whether one feature should be included in training data. Pre-computed data is split into training and testing data using Random Hold-out. List of testing classifiers includes basic classification models: Zero-R, Linear SVC, Decision Tree, K-Nearest Neighbor (k=5), Logistic Regression [1].

Figure 3 shows, for all processing methods, `steps` has highest accuracy. This confirms the background guess of `steps` being the best performer among text features. Additionally, among text preprocessing methods, `CountVectorizer` gives highest accuracy, while `doc2vec` methods have similar results.

Model Training

Model Choices

During evaluation, the following criteria are taken into consideration:

- With three possible labels, classification problem becomes multi-class. Therefore, basic binary classifiers requires extension to multi-label approach.
- With up to 40000 instances, time and space complexity are main problems of concern when choosing algorithm, making square complexity models such as Support Vector Machine [6] infeasible.
- Data are in wide variety of structures and forms, including sparse matrix and negative vector values, which restricts Naive Bayes and Multinomial Naive Bayes, as the former requires dense data, and the latter requires positive data [7].

With these factors, five models are chosen for evaluation process.

Decision Tree

Decision Tree gives flow-chart structure where each branch node represents split in training attributes, with loglinear complexity [8]. Splitting is called recursively and chosen based on gain ratio. In dataset where numeric features are placed equally with text-derived features, gain ratio helps reducing bias towards highly-branching attributes by normalising split information, in exchange for high variance.

For evaluation, Decision Tree has hyperparameters set to fully expands to pure leaves only, catching the most dataset information. This makes it prone to overfitting

[1] This is not the final choice of models for evaluation. These models are only used in this section

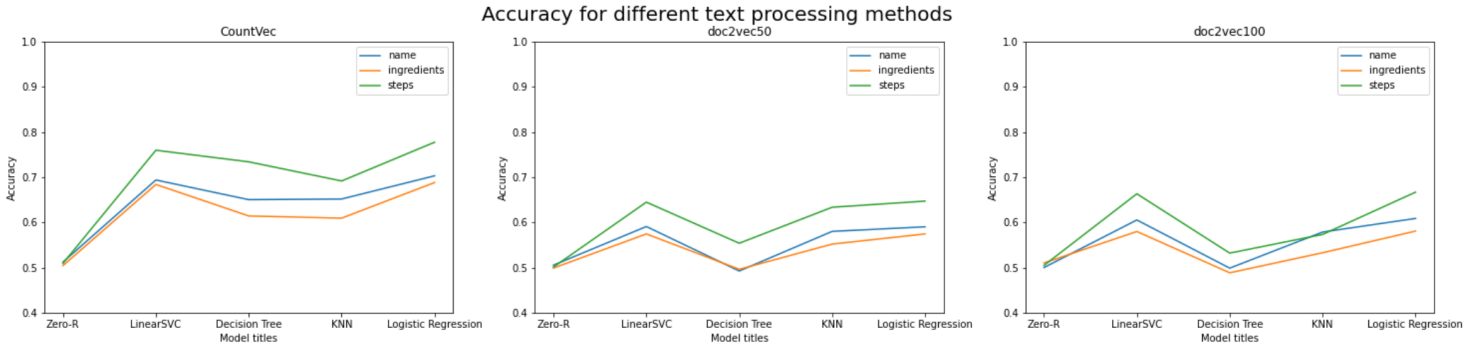

Figure 3: Accuracy for different text processing methods

and sampling error, hence is used as benchmark classifier to assess data generalization.

K-Nearest Neighbor (KNN, k=5)

KNN classifies instances based on majority label of their nearest instances, producing flexible decision boundary with linear-growth complexity [6]. KNN is expected to perform well with integer-typed features. At low k, without weighting, model is prone to local noise and redundant attributes that are correlated [9].

Therefore, KNN with k=5 and no weighting is used as second benchmark classifier to estimate usability of features and noise risk.

Multinomial Logistic Regression (MLR)

Logistic Regression uses logistic function to predict conditional likelihood of binary label to instance. MLR extends from binary to multiple classes, using multinomial distribution to estimate loss functions of probabilities [10]. By assigning weights from learning, MLR helps eliminating noise in data. However, for large dimensional data with irrelevant attributes, MLR is more prone to overfitting and require more instances to remain effective [11, 5], as a result from **The curse of Dimensionality** [12]

The model is expected to perform well in text analysis [13], and therefore is suitable when evaluating texts from `steps`.

Random Forest

Random Forest combines decision trees where each is built using a different bagged training dataset and use voting for final result [9]. With each tree being built and combined, the model reduces variance, hence more robust to overfitting. However, bootstrapping in general and random forest specifically will become more prone to bias, due to this selection preference [14].

For implementation, hyperparameters are also taken into consideration:

- Number of trees = 100: More trees will keep or improve model performance [15]. However, as complexity grows linear with it, balance between time and accuracy is needed.
- Minimum samples to split = 50: The larger the parameter, fewer splits are operated, which helps reducing overfitting, but may underfit when it goes higher.
- Minimum samples in a leaf = 1: Same characteristics as minimum samples split.

Stacked Ensemble with eXtreme Gradient Boosting (XGBoost)

XGBoost Gradient Boosting (GB) is similar to Random Forest, where model is built up on collection of different Decision Trees to minimize variance. But rather than bagging, GB uses boosting so that hard-to-classify trees are assigned more weights to focus on [16, 9]. XGBoost improves on GB, by differentiating loss function to second order rather than first order, and penalizing high complexity trees to further regularize the objectives and reduce variance [17]. It also allows parallelization, which helps training efficient in time.

For now, maximum depth of each tree will be set low at 6, as parallel-run trees require more memory at further depths [17]. Further hyperparameter tuning will be implemented at the end to assess effectiveness.

Stacking Stacking classifier uses a meta-classifier to train results from different base models, giving more ac-

Model/Combined Attribute	CountVec (name)	CountVec (ingredients)	CountVec (steps)	doc2vec50 (steps)	doc2vec100 (steps)
Decision Tree	66.98	61.92	73.58	57.61	56.25
K Nearest Neighbor	66.29	66.19	72.28	67.7	66
Logistic Regression	73.19	70.46	78.52	70.84	71.59
Random Forest	73.58	70.69	77.71	70.02	68.73
Stacking Ensemble	73.93	71.79	80.61	71.89	72.19

Evaluation Method: 10-Fold Cross-Validation

Table 1: Result table (n_steps, n_ingredients, seconds and Combined Attribute)

curate result. In order to create diverse scenarios for learning, the following variety of base models are chosen:

- Random Forest: Reduce variance significantly, prone to bias.
- XGBoost: Reduce the most data variance.
- MLR: Can handle irrelevant values, but prone to redundant features.

With 3 base models, MLR is chosen as meta-classifier, due to its ability to assign weights during training, which helps favoring the most correct model, but still taking others into account when the model make mistakes itself.

Results & Analysis

Table 1 shows that steps is dominant for text features and CountVectorizer is dominant for processing methods, which are consistent with observations in Data Investigation part. Another significant point is Stacking Classifier performs better than its base classifiers in all cases, which performed better than the benchmark classifiers themselves.

Label	Presicion	Recall	f1-score
1	0.80	0.80	0.80
2	0.81	0.83	0.82
3	0.84	0.58	0.69

Table 2: Performance metrics of Stacking Classifier - CountVectorizer(steps) input

From Table 2, all labels have high precision, proving that model is sufficient. However, recall shows that nearly half of Label 3 are predicted incorrectly. This confirms the observation of lacking representative data for Label 3, making it hard to detect among other labels.
Furthermore, consider the following steps:

```
510 (1.0): "mix...chill overnight"
544 (3.0): "refrigerate...overnight is best"
471 (2.0): "pour...per...units' instructions"
```

Both 510 and 544 has the word '*overnight*', but is assigned differently. While 544 takes it into account and labelled 3.0, 510's label is 1.0, not counting in refrigerating time. For 471, the step is vague with no clear information other than 'follow instructions'. This shows that for steps like above, even human label will also **yield different possibilities for same instances**, creating more variance. Therefore, accuracy of around 70-80% is reasonable. For improvement, either more data is required, or number of features should be reduced to an optimal balance. With minimal training data, the latter is the only feasible choice for investigation.

Feature Engineering: How Long & How?

As mentioned in previous section, with number of steps and ingredients being only numerical variables available, detecting labels with low occurrence will require further helping variables. Viewing on steps provides potential information for investigation.

RegEx Time Retrieval

```
'few minutes'
'couple secs'
'2 hours'
'30s'
'overnight'
```

Figure 4: Examples of timing keywords

In reality, element for predicting duration type is the duration itself. Most recipe steps include the duration themselves, in the form of **timing keywords**.

$$\text{time}(\mathcal{S}) = \sum_{i \in \mathcal{S}} \text{numeric}(c_i) \times \text{seconds}(u_i) \quad (1)$$

where $\mathcal{S}$: the steps of a recipe

c_i: count keyword of keyword i

u_i: time unit of keyword i

To retrieve these keywords, numeral or quantitative words (e.g. 'few') are converted to numeric values. Then, time unit keywords with a number at the front will be retrieved using `RegEx` (Regular Expression). Each recipe will have their timing keywords converted to **numerical time** in seconds, and added up together as a result.
However, not all recipes has timing. Counting the missing values for each label returns the following result:

```
Counter({1.0: 1969, 2.0: 728, 3.0: 174})
```

This means there are nearly 3000 instances without cooking time out of 40000, up to 10% of the dataset. Further view into missing data shows that they **only contains cooking verbs** rather than specific duration:

```
'mix all ingredients together and serve'
'slow cook until meat is soft'
```

NLP Keyword Assign

Verb Cloud for Cooking Steps

Figure 5: Cloud of verbs from dataset

As defined in Oxford English Dictionary, step is *'a sequence of movements'* [18]. It can be seen from dataset's cloud of verbs in Figure 5 that they are either instantaneous actions (e.g. 'add', 'pour'), or processing methods (e.g. 'bake', 'tender').

$$\text{time}(\mathcal{S}) = \max_{w_i \in \text{verbs}(\mathcal{S})} \mathbf{K}(w_i) \quad (2)$$

Keyword Assign method will assign to each verb a duration, either by averaging from train dataset $\mathcal{D}$ (**Trained Keyword Assign - TKA**), or self-assume a specific duration (**Näive Keyword Assign - NKA**). The most time-dominant action will have its duration as output for the instance, as shown in Equation 2.

$$\text{TKA: } \mathbf{K}(w) = \frac{1}{N} \sum_{j=1}^{N} \text{seconds}(r_j) \quad (3)$$

$$\text{where } r_j \in \mathcal{D}, \quad w \in r_j$$

$$\text{NKA: } \mathbf{K}(w) = \mathcal{C}, \quad \mathcal{C} \text{ self-defined} \quad (4)$$

With 40000 training instances, the dataset might be insufficient to retrieve its own estimation of verb duration. Many verbs only occur in data at low rate. Furthermore, consider the following step:

```
'boil the meat, then leave it for 8 hours'
```

The step has the word 'boil', but is followed by another 8-hour step. This will result in an instance with 'boil' word but **takes up to 8 hours**. With low occurrence rate and noise risk, it is evident that insufficient data size will make TKA prone to bias.

To reduce the risk of bias, NKA is used for 80 most common verbs in the dataset. Instantaneous actions are assigned 36 seconds (0.01 hour), while for cooking methods, research is carried in food engineering articles to determine average time of different heat treatments [1, 2]. Applying NKA to instances with missing values, and recounting the number of missing values left for each label returned the following:

```
Counter({1.0: 69, 2.0: 8, 3.0: 1})
```

It can be seen that there are now only 78 missing instances remaining out of 40000, with most low-occurrence labels have their time estimated.

Feature Evaluation

Compared to the sparse distribution in Figure 2, Label 3 instances can be seen in Figure 6 with more dense clusters in the scatterplots, with addition of `seconds`. This shows that the newly engineered feature is capable of detecting rarer instances when combined with initial numeric features.

From the correlation heatmap in Figure 7, `n_steps`, `n_ingredients` and `seconds` have similar correlation to `duration_label`. Compared to the two original features, `seconds` are independent, which means it can tell

Model/Combined Attribute	`CountVectorizer(steps)`	`doc2vec50(steps)`	`doc2vec100(steps)`
Decision Tree	74.67	69.88	69.3
K Nearest Neighbor	77.94/**68.866**	77.37/**68.233**	76.98/**67.066**
Logistic Regression	78.97	70.88	71.1
Random Forest	79.38	77.16	75
Stacking Ensemble	81.98/**81.4**	79.45/**77.633**	79.26/**76.066**

0: 10-Fold Cross-Validation **0**: Kaggle Submission

Table 3: Result table (`n_steps`, `n_ingredients`, `seconds` and Combined Attribute)

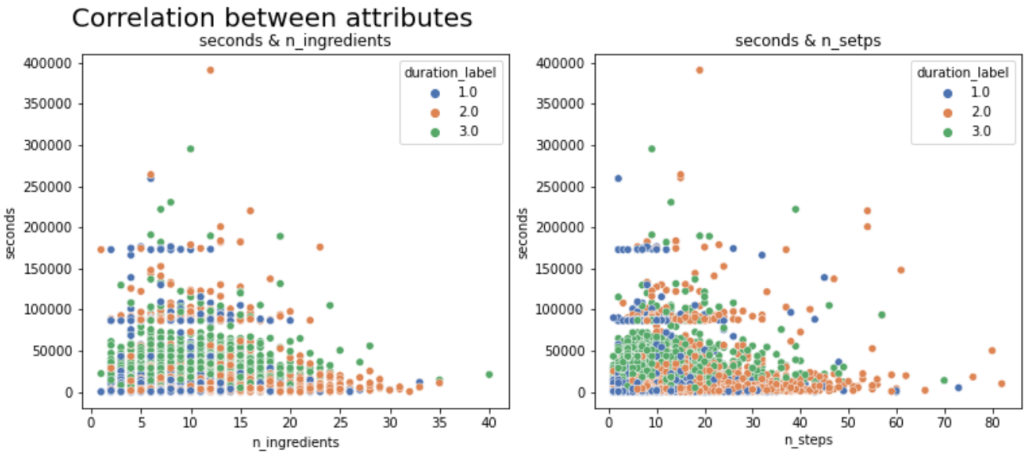

Figure 6: Scatterplot of engineered feature `seconds` with `n_steps` and `n_ingredients` of data points

the same amount of additional information in predicting label, without increasing overfitting risk.

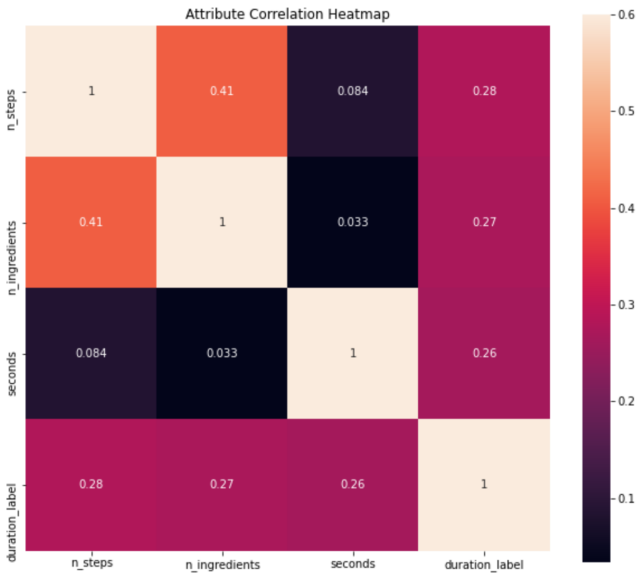

Figure 7: Numeric features correlation

Results Analysis

Label	Presicion	Recall	f1-score
1	0.81	0.83	0.82
2	0.84	0.83	0.84
3	0.84	0.66	0.74

Table 4: Performance metrics of Stacking Classifier - `CountVectorizer(steps)` and `seconds` input

From Table 4, performance evaluation on the same stacking model shows 10% increase in Label 3's recall. This shows that the engineered feature successfully improved Label 3 representation as expected.

Results from Table 3 shows high increase in most models, with the exception of KNN and MLR.

When Time Retrieval is applied, MLR does not significantly improve the accuracy, and even obtained lower results for `doc2vec100`. This confirms the fact that more features can negatively affect the performance of MLR.

Same problem also occurred for KNN, which makes it

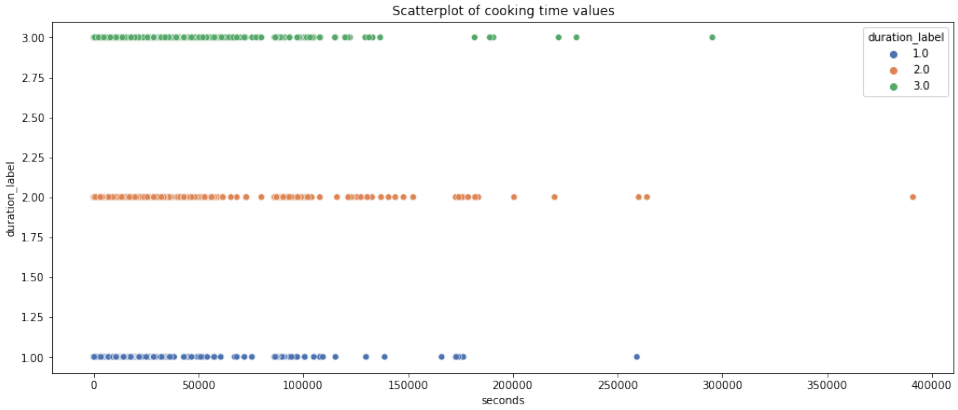

Figure 8: Scatterplot of `seconds` attribute for each label

overfit the data, shown through Kaggle submissions having 10% lower accuracy. This indicates that `seconds` has noise, as well as features are being redundant.

Figure 8 confirms the precense of outliers in `seconds` feature. Extracting the rightmost outlier in Figure 8 returns the following step:

```
33090 (2.0):"refrigerate for 24 to 36 hours"
```

While it takes up to 36 hours to complete, the recipe was labelled as **2.0**. This is because the user does not count refrigeration time as preparation time. This once again reflects and supports the error analysis of initial predictions.

Top Percentile Feature Selection

Now that `seconds` has been successfully engineered, it is implemented in training model. More `steps` information can be used for improvement and `doc2vec50(steps)` is combined to the data.

However, feature quantity can become excessive, where features are redundant and uncorrelated, reducing model performance [5, 12] and costing expensive computational time.

One solution to this problem is Feature Selection, where only most correlated features are selected, maintaining substantial information in dataset. For this project, mutual information is used as comparison criteria.

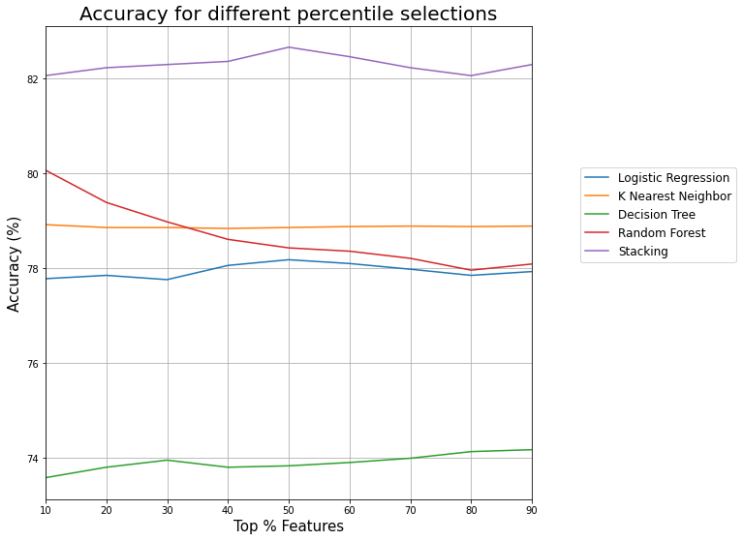

Figure 9: Accuracy of Top Feature Selection for different Percentiles

Results from Figure 9 shows that top 50% features is best for Stacking model, as well as being a safe choice for the rest, given that it will improve or not differ significantly from the rest of percentiles.

With all information mentioned, final model is trained and results are recorded in Table 5.

	All Features	Top 50%
Decision Tree	74.06	73.9
K Nearest Neighbor	77.99	77.98
Logistic Regression	78.87	79.17
Random Forest	77.8	78.37
Stacking Ensemble	81.64/**82.6**	81.66/**82.666**

0: 10-Fold Cross-Validation **0**: Kaggle Submission

Table 5: Feature Selection Result Comparison

Overall, Feature Selection's accuracy improved significantly, compared to original training. Having the highest overall accuracy, this data combination process is selected for Kaggle submission.

Random Search Hyperparameter

As an attempt investigating how hyperparameter tuning affect models, Random Search is carried for Random Forest model. Six hyperparameters of Random Forest were chosen and given a range of values, resulting in around 4000 combinations. 100 combinations are picked randomly and run a 3-fold Cross Validation each for accuracy. Best combination in the set will be chosen to replace original model as base classifier in the stack.

	Original	Tuned
Random Forest	78.37	77.34
Stacked Ensemble	**82.66**	**82.4**

0: 10-Fold Cross-Validation **0**: Kaggle Submission

Table 6: Result comparison for hyperparameter tuning

Contrary to prediction, tuned model resulted in worse performance, in comparison to both original model and stacking. This is due to the fact that only partial subset of parameters is chosen randomly, which has chance of not including any better models. Furthermore, tuning result also depends on dataset, which may not generalize well, and as a result, causes overfitting [19].

Overall, hyperparameter tuning was not successful with the current resources. When conditions are met, potential improvements include Grid Search for each combination, or evaluation on multiple datasets.

Conclusion

Comparing to other results on Kaggle, the model's accuracy lies in the top percentile. Similar results on the leaderboard at the range of 75-80% shows that training on the data can return plausible results, indicating that the dataset contains useful information with minimal variance and noise.

Through feature engineering using NLP techniques and feature selection, results on the same model showed an improvement of 2%. Further trial of combinations, neural networks or hyperparameter tuning methods may result in better results. However, these potentials are currently impossible given the time constraint. Therefore, the current model is kept, and selected as the best method.

References

[1] C. Bejerholm and M.D. Aaslyng. COOKING OF MEAT. In *Encyclopedia of Meat Sciences*, pages 343–349. Elsevier, 2004.

[2] Pankaj B. Pathare and Anthony Paul Roskilly. Quality and energy evaluation in meat cooking. *Food Engineering Reviews*, 8(4):435–447, April 2016.

[3] Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and 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.

[4] Jiatong Li, Ricardo Guerrero, and Vladimir Pavlovic. Deep cooking: Predicting relative food ingredient amounts from images. In *Proceedings of the 5th International Workshop on Multimedia Assisted Dietary Management - MADiMa '19*. ACM Press, 2019.

[5] G. V. Trunk. A problem of dimensionality: A simple example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3):306–307, July 1979.

[6] Computational complexity of machine learning algorithms, Apr 2018.

[7] Merran Evans. *Statistical distributions*. Wiley, New York, 2000.

[8] Habiba Muhammad Sani, Ci Lei, and Daniel Neagu. Computational complexity analysis of decision tree algorithms. In *Lecture Notes in Computer Science*, pages 191–197. Springer International Publishing, 2018.

[9] Tan et al. *Introduction to data mining*. Pearson Education, Inc, New York, NY, 2019.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] C. R. Taylor. *Applications of dynamic programming to agricultural decision problems*. Routledge, Abingdon, Oxon New York, NY, 2018.

[12] Cognitive (Neuro)Scientist. Why does logistic regression overfit in high-dimensions?, Aug 2020.

[13] Tomas Pranckevicius and Virginijus Marcinkevicius. Application of logistic regression with part-of-the-speech tagging for multi-class text classification. In *2016 IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. IEEE, November 2016.

[14] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(1), January 2007.

[15] Sharoon Saxena. Random forest hyperparameter tuning in python: Machine learning, Apr 2020.

[16] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5), October 2001.

[17] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[18] *Oxford Dictionary of English*. Oxford University Press, January 2010.

[19] Ben Gorman. Guide to model stacking (i.e. meta ensembling), Dec 2016.