



About the competition

I joined the competition as team "COMP4432_Bor". During The final score for 2 submissions are 0.733 for base logistic regression model and 0.718 for stacking model.

 logR.csv Complete (after deadline) · 1d ago	0.733	0.734	<input type="checkbox"/>
 stack_2.csv Complete (after deadline) · now	0.718	0.744	<input type="checkbox"/>

About the code

I had implemented several class related to the competition. The Python source code can be found in "src" folder, which shares the same directory with this report. Classes are implemented for the sake of observing their performance and constructing ensemble models.

Step 1: Data Evaluation

Retrieved from Kaggle, I have 250 records * 300 features training dataset and a 19750 records * 300 features testing dataset. During the evaluation, I noticed that:

1. Data is well-constructed:

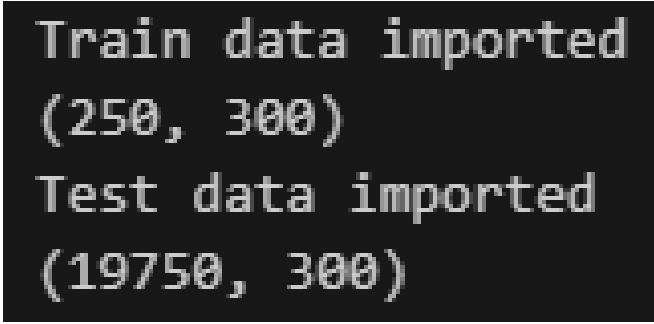
The dataset appears to be well-constructed, indicating that it is organized, consistent, and without missing values or errors.

2. Limited training set:

The training set consists of only 250 records. A small training set may pose challenges for machine learning models, particularly complex ones, as they may struggle to generalize well with limited data. Techniques such as cross-validation would be necessary to mitigate overfitting.

3. High dimensionality:

The dataset exhibits 300 features. We consider this as high-dimensional data as it is larger than the number of records (i.e. 250). Such a phenomenon can lead to several issues, including the curse of dimensionality, increased computational complexity, and overfitting. Dimensionality reduction would be required for the training.

A terminal window with a black background and white text. The text displays the results of data import: 'Train data imported (250, 300)' and 'Test data imported (19750, 300)'.

```
Train data imported  
(250, 300)  
Test data imported  
(19750, 300)
```

Fig.1 Data is imported as Pandas Dataframe

Step 2: Data Preprocessing

For boosting performance, we tend to preprocess data before using them to train a model. In general, we perform standardization for each column in the data. This can reduce the bias of each record, retrieving a more generalized result.

Step 3: Model Selection

To facilitate my study in the competition, I proposed a variety of base models, including k-nearest neighbours, support vector machine, and logistic regression. In addition, a few ensemble models with different base models are proposed. Under the sense of collective wisdom, these models genuinely perform better than the base models. Therefore, experiments around the hypothesis are conducted.

This study employs five-fold cross-validation to assess model accuracy and fine-tunes hyperparameters using grid search, guided by recommendations from ChatGPT [1]. The depicted plot showcases the maximum accuracy achieved by models utilizing the specified hyperparameter configurations. This approach ensures a robust evaluation of model performance for comparative analysis.

Step 3.1: Base Model

1. K-Nearest Neighbors

kNN is a simple and intuitive algorithm used for classification and regression tasks. It classifies data points based on the majority class of their k nearest neighbours in the feature space, making it non-parametric and robust to noisy data. It is known that the curse of dimensionality often occurs when using the kNN model due to its sensitivity to dimensionality. This results in a reduction of model accuracy. Therefore, it is suggested to perform PCA on the dataset before training [2].

Through cross-validation, the KNN model attained its highest accuracy of 0.676 with a value of k set to 13. On the other hand, the PCA-kNN model has a 0.636 accuracy

score with the best parameter $k=13$. It is interesting to observe the equivalence of both fine-tuned hyperparameters.

```
param_grid = {  
    'n_neighbors': [i for i in range(1, 32, 2)],  
}
```

```
Best parameters found: {'n_neighbors': 13}  
Best accuracy on validation set: 0.6759999999999999
```

```
Best parameters found: {'n_neighbors': 13}  
Best accuracy on validation set: 0.636
```

Fig.2 Cross-validation result on kNN

2. Support Vector Machine

SVM optimize a hyperplane to effectively classify data points in feature space, leveraging support vectors to define this boundary's position and orientation. By maximizing the margin between support vectors of distinct classes and utilizing kernel functions for nonlinear relationships, SVM achieves robustness against overfitting and enhances its generalization capabilities.

After fine-tuning the SVM model using cross-validation, we achieved the highest accuracy of 0.712. This optimal performance was obtained with the following hyperparameters: C set to 10, gamma set to 'scale', and kernel set to 'rbf'.

```
param_grid = {  
    'C': [0.1, 1, 10, 100],  
    'kernel': ['linear', 'rbf', 'poly'],  
    'gamma': ['scale', 'auto'],  
}
```

```
Best parameters found: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}  
Best accuracy on validation set: 0.7120000000000001
```

Fig.3 Cross-validation result on SVM

3. Logistic Regression

In essence, Logistic Regression (LogR) offers a straightforward yet effective approach to binary classification, estimating the probability of a sample belonging to a specific class using the sigmoid function. By iteratively refining model parameters through optimization techniques like maximum likelihood estimation or gradient descent, Logistic Regression achieves accurate predictions by adjusting to the underlying data distribution and applying a threshold to delineate class boundaries based on calculated probabilities.

Following cross-validation, the Logistic Regression model achieved its highest accuracy of 0.768. This optimal performance was attained using the following hyperparameters: C value set to 0.1, penalty set to 'l1', and solver set to 'liblinear'.

```
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga'],
}
```

```
Best parameters found: {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
Best accuracy on validation set: 0.768
```

Fig.4 Cross-validation result on logistic regression

Test Case Accuracy

After submitting the predictions of each basic model, the following result is gained:

Model	Validation Score	Private Score	Difference to val.
kNN	0.676	0.546	-0.130
PCA-kNN	0.636	0.519	-0.117
SVM	0.712	0.590	-0.112
LogR	0.768	0.733	-0.035

Fig.5 Accuracy of submitted predictions of base models

Based on the observations, we found that:

1. PCA indeed helps reduce overfitting issues. However, the precision also drops. Therefore, PCA should not be used in further experiments.
2. LogR model has both the highest private score and the lowest score difference. It is persuasive to conclude that it is the most reliable algorithm among these base models.

Step 3.2: Ensemble Models

Upon obtaining the optimized base models, we proceed to construct ensemble models, encompassing both bagging and boosting techniques. Each ensemble model will explore various base models, aiming to identify the most effective configuration. With the inspiration provided by Sarafanov, we applied a simultaneous tuning strategy by concatenating the parameter grid of an ensemble model with a base model [3]. The previous parameter grids will be reused for concatenation.

Bagging Models

1. Custom Bagging Classifiers

We'll experiment with different base models, such as kNN, SVM, and Logistic Regression. Through rigorous evaluation, we'll select the best-performing base model for bagging, optimizing parameters for maximum efficacy.

```
Best parameters found: {'bootstrap': True, 'estimator': KNeighborsClassifier(),
'estimator__n_neighbors': 9, 'max_features': 0.5, 'max_samples': 1.0, 'n_estimators': 10, 'n_jobs':
-1, 'random_state': 42}
Best accuracy on validation set: 0.6880000000000001
```

Fig.6 The best score and hyperparameter of the kNN-bagging model

```
Best parameters found: {'bootstrap': True, 'estimator': SVC(), 'estimator__C': 0.1, 'estimator__gamma': 'scale',
'estimator__kernel': 'linear', 'max_features': 0.7, 'max_samples': 1.0, 'n_estimators': 10, 'n_jobs': -1,
'random_state': 42}
Best accuracy on validation set: 0.724
```

Fig.7 The best score and hyperparameter of the SVM-bagging model

```
Best parameters found: {'bootstrap': True, 'estimator': LogisticRegression(), 'estimator__C': 0.1, 'estimator__penalty':
'l1', 'estimator__solver': 'liblinear', 'max_features': 1.0, 'max_samples': 0.7, 'n_estimators': 100, 'n_jobs': -1,
'random_state': 42}
Best accuracy on validation set: 0.7719999999999999
```

Fig.8 The best score and hyperparameter of the LogR-bagging model

Compared with hyperparameters from the base models, the hyperparameters of the bagging models are varied. In addition, each corresponding bagging model has a better performance than the base model itself.

However, the cost of grid search is realized. For grid searching these ensemble models, it took at least 40 minutes for computation. Most of the computational cost comes from the LogR-based bagging model, approximately 3 times compared with other models.

2. Random Forest Classifier

Expanding our ensemble comparisons, we incorporated the RandomForestClassifier() alongside other bagging classifiers. Similar to prior endeavours, we provided a parameter grid for fine-tuning to optimize model performance. Upon validation, the RandomForestClassifier achieved the best accuracy of 0.668, with the following hyperparameters identified as optimal: bootstrap set to True, max_depth as None, max_features using 'sqrt', max_samples set to 1.0, min_samples_leaf of 1, min_samples_split of 2, n_estimators at 50, n_jobs set to -1, and random_state at 42.

Performance result of bagging models

By submitting the prediction on Kaggle, the following accuracy scores are given:

Model	Validation Score	Private Score	Difference to val.	Improvement from base private	Improvement from base val. diff.
kNN Bagging	0.688	0.576	-0.112	+0.030	+0.018
SVM Bagging	0.724	0.625	-0.099	+0.035	+0.013
LogR Bagging	0.772	0.606	-0.166	-0.127	-0.131
Random Forest	0.668	0.542	-0.126	NA	NA

Fig.9 Accuracy of submitted predictions of bagging models

Upon the observation of bagging and base models' performance, we conclude that:

1. For kNN and SVM models, using bagging helps improve precision and reduce overfitting.
2. Random forest has a slightly worse performance than kNN and SVM bagging models.
3. LogR bagging has a much worse performance than the base one, in both precision and overfitting. Therefore, LogR is not suitable as a base model candidate for bagging models.

Boosting Models

Similar to bagging, we'll explore various base models to determine the optimal choice for boosting. By iteratively refining weak learners' performance, we aim to construct a powerful ensemble model that enhances predictive accuracy.

We are using only SVM as the base model. Since the kNN classifier does not support the sample weighting feature, it is excluded from the base model candidates. Also, it is evaluated that LogR is less likely to improve the performance meanwhile bearing a numerous computational cost. Thus, it is excluded from the candidates.

```
ValueError: BaseClassifier in AdaBoostClassifier ensemble is worse than random, ensemble can not be fit.
[Done] exited with code=1 in 4.483 seconds
```

Fig.10 Error message when applying LogR as base model of AdaBoost

1. AdaBoost

AdaBoost (Adaptive Boosting) is an ensemble learning method that combines multiple weak learners to create a strong classifier. It sequentially trains weak learners by adjusting the weights of misclassified instances, focusing more on difficult cases in subsequent iterations. The final prediction is made by a weighted combination of weak learners, where those with higher accuracy contribute more to

the final decision.

```
Best parameters found: {'algorithm': 'SAMME', 'estimator': SVC(), 'estimator__C': 1, 'estimator__gamma': 'scale',
'estimator__kernel': 'linear', 'learning_rate': 1, 'n_estimators': 50, 'random_state': 42}
Best accuracy on validation set: 0.712
```

Fig.11 The best score and hyperparameter of the SVM-boosting model

Performance result of SVM-related models

Model	Validation Score	Private Score	Difference to val.	Improvement from base private	Improvement from base val. diff.
SVM base	0.712	0.590	-0.112	NA	NA
SVM Bagging	0.724	0.625	-0.099	+0.035	+0.013
SVM AdaBoost	0.712	0.631	-0.081	+0.041	+0.031

Fig.12 Accuracy of submitted predictions of SVM-related models

From the figure, we found that the performance of AdaBoost is better than both basic SVM and SVM Bagging model, in terms of both precision and overfitting mitigation.

2. XGBoost

Apart from other ensemble models, XGBoost requires no base models. Instead, each decision tree is added to the ensemble to correct the errors made by the previous trees. The trees are grown iteratively, with each subsequent tree focusing on the instances that were misclassified by the previous ones. XGBoost uses a gradient boosting framework, where the trees are trained to minimize a differentiable loss function.

```
Best parameters found: {'colsample_bytree': 0.6, 'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 7,
'n_estimators': 200, 'reg_alpha': 0.5, 'reg_lambda': 0.5, 'subsample': 0.6}
Best accuracy on validation set: 0.756
```

Fig.13 The best score and hyperparameter of the XGBoost model

Performance result of XGBoost

Model	Validation Score	Private Score	Difference to val.
XGBoost	0.756	0.629	-0.127

Fig.14 Accuracy of submitted predictions of XGBoost model

Evaluating with previous scores, we found that XGBoost is slightly better than SVM Bagging, yet worse than SVMBoost model. Also, the overfitting problem is more obvious than all the SVM-related models and base models.

Stacking

To implement a stack model that includes base, bagging, and boosting models, we'll follow a structured approach. The stack model will consist of two levels: level-0 models (base models) and a level-1 model (meta-model). The base models are the previously developed models. The meta-model is often a simple model that combines the predictions of the base models. We are using logistic regression as the meta model.

```
Best parameters found: {'final_estimator__C': 100, 'stack_method': 'auto'}
Best accuracy on validation set: 0.7479999999999999
```

Fig.15 The best score and hyperparameter of the stacking model

Performance result of stacking model

Model	Validation Score	Private Score	Difference to val.
Untuned Stacking	NA	0.696	NA
Tuned Stacking	0.748	0.718	0.030

It is observed that, with tuned stacking model, its accuracy is better than the majority of models, except the base logistic regression model.

Summary

It is interesting to verify that a base logistic regression model has the best result among others. It is argued that the simplicity of dataset leads to such result. When the dataset grows larger and contains more features, the ensemble models are likely performing better than base logistic model. Overall, this was a great opportunity to boost my knowledge in machine learning.

Reference

- [1] ChatGPT, response to author query. OpenAI [Online].
- [2] H. Zhai, "Improving KNN Algorithm Efficiency Based on PCA and KD-tree," presented at the 2022 International Conference on Machine Learning and Knowledge Engineering. [Online]. Available: <https://ieeexplore.ieee.org/document/9763596>
- [3] M. Sarafanov, "Hyperparameters Tuning for Machine Learning Model Ensembles," *Medium*, Jun 28, 2022. [Online]. Available: <https://towardsdatascience.com/hyperparameters-tuning-for-machine-learning-model-ensembles-8051782b538b>