ML Final Report

Mr. O'clock insanely practices

December 22, 2024

1 Introduction

This final project is aim to use machine learning to predict HTMLB outcome. We have explore through many algorithms and data preprocess, and result in 0.58989 accuracy in predicting same season outcome(stage1) and 0.55473 in predicting a whole new season(stage2). We are going to break it down what have we done with every model and preprocess and what are the results.

2 Data Preprocess

Dealing with NAN

There are many NAN in all kinds of data. To deal with them, we can either fill in 0, fill in mean of other non-NAN rows, or fill in mean of the data of same team.

Team Attribute and Pitcher Attribute

We try to add 60 boolean columns: is_home_team_{team name} and is_away_team_{team name} which indicate that if home team or away team equal to team name. Same method can be used on pitcher name.

However we try it on random forest an observed that these attribute may have low importance. We guess that perhaps this kind of data is too scatter to make it important

Alternatively, we can labelize string columns, this may work in algorithm based on decision tree.

Drop std and skew

Since we think std and skew has little relationship with winning or lose. We try to drop all of those column.

The result is that there are no difference in accuracy. However, training speed is faster using this data.

3 Models

KNN

PLA

Linear Regression

Logistic Regression

Logistic regression is a machine learning technique that predicts the probability of an outcome based on the weighted contribution of predictors. The sigmoid function ensures the probabilities are valid, and the coefficients are adjusted during training to maximize how well the model explains the observed data.

This is one of the very first models we have tried on, and the best one among all trained logitic regression models used the parameters as follows:

python model = LogisticRegression(solver='liblinear', $\max_iter = 1000, random_state = 42$)
where the logistic regression model is included in the package *sklearn.linear_model*.

We split the provided training data into 80% for training and 20% for validation. The above model could reach a training accuracy of 63.42% and validation accuracy 54.52%. The public score we got on kaggle is 56.9% for stage 1, and 57.64% for stage 2. For private, we got an accuracy of 57.1% for stage 1 and 53.85% for stage 2. In stage 2, a larger gap emerges between the public and private scores, which we attribute to using the same training technique for both stages. We did not account for the influence of time, an especially critical factor in stage 2, which should have been incorporated into our model. Overall, it's an efficient model but assumes a linear relationship which might not always be the case especially in complex problems like this.

SVM

Choosing the best model in SVM

In this section, we implement SVM model to predict the MLB result. SVM a.k.a. support vector machine can select multiple parameters to decide the data transformation method and hyperplane to seperate the data. To choose the best model, we use grid search by following parameters

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

After the bruteforce searching, the result turns out that (kernel : linear, C=1.0, gamma : scale) and (kernel : rbf, C=1.0, gamma : scale) have the best performance which results in training data accuracy around 0.58 and validation data accuracy about 0.56 with 5-folds validations. After submit the test results to the kaggle in stage 1 and stage 2, the first parameters combination turns out to be (stage 1 : , stage 2 :), and the second combinations results in (stage 1 : , stage 2 :).

SVM with bagging and blending

Since both parameters combinations have the similar prediction accuracy, we come up with an idea to combine those parameters combinations by bagging, blending with SVM. When it comes to bagging, we use bootstrapping to generate different model with kernel=linear and kernel=rbf by selecting different values in the maximum number of features (abbreviated as max_feature which can increase the diverity of models), the maximum number of samples (maximum_number of train samples abbreviated as max_samples) and the number of estimator (the number of different hypothesis generated from bagging).

This time we utilize optuna package to help us find the best parameters by following commands (Since optuna can tune the parameters in more precision ways, we also take C and gamma into considerations to expect better outcomes).

```
C = trial.suggest_loguniform('C', 0.1, 100.0)
gamma = trial.suggest_categorical('gamma', ['scale', 'auto'])
n_estimators = trial.suggest_int('n_estimators', 10, 50)
max_samples = trial.suggest_float('max_samples', 0.5, 1.0)
max_features = trial.suggest_float('max_features', 0.3, 1.0)
```

After optimization with Optuna, the selected best parameters are as follows:

```
C = 31.318108635885725
gamma = 'auto'
n_estimators = 25
max_samples = 0.9099476134699473
max_features = 0.806228041021773
```

And the submission result on stage 1 and stage 2 is corresponding to and .

Package references

```
from sklearn.linear_model import LogisticRegression  # Logistic Regression model
from sklearn.model_selection import cross_val_score  # Cross-validation
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report  #
    Evaluation metrics
rom sklearn.feature_selection import RFE
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC, LinearSVC
import optuna
```

Random Forest

XGboost

XGboost is a machine learning algorithm package that based on GBDT, gradient boosting deciesion tree. In this package we can self define multiple parameters.

- max depth : The maximum depth in each decision tree.
- learning rate : Also known as eta which scales the contribution of each tree.
- n_estimators : The number of boosting trees.
- subsample : Fraction of the training data used for growing each tree.
- colsample_bytree : Fraction of features sampled for each tree.
- gamma : Minimum loss reduction required to make a split.
- lambda : L2 regularization term on leaf weights.

In order to select the best combination form above parameters, we use optuna to search in following ranges

```
    param = {
        "verbosity": 0,
        "objective": "multi:softmax",
        "num_class": 2,
        "eval_metric": "mlogloss",
        "max_depth": trial.suggest_int("max_depth", 1, 10),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3),
        "n_estimators": trial.suggest_int("n_estimators", 30, 200),
        "subsample": trial.suggest_float("subsample", 0.3, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 10),
        "lambda": trial.suggest_float("lambda", 0, 10),
    }
```

The validation method implemented in this case is first sort the date of given training data and then take the first 80% of the data in each each season as training data set and the remaining ones as validation test data. (which is designed for stage 1 prediction). The best parameters is represented as follows :

```
    param = {
        "verbosity": 0,
        "objective": "multi:softmax",
        "num_class": 2,
        "eval_metric": "mlogloss",
        "max_depth": 5,
        "learning_rate": 0.06878298628903712,
        "n_estimators": 53 ,
        "subsample": 0.31490312644463636,
        "colsample_bytree": 0.6050549554091992,
        "gamma": 5.0214264964575595,
        "lambda": 9.517586328422444
    }
```

Submit the prediction results by this parameters can get the accuracy in stage 1 as and in stage 2 as .

3.0.1 Package references

```
    import optuna
    import xgboost as xgb
```

CATboost

Catboost is an algorithm based on Gradient Boosting Tree, which is similar to XGboost and LightGBM. However, Catboost can deal with string data type directly, which is a very good characteristic for this problem since we think string attribute like team name or pitcher name are important for training and cannot be dropped.

In all of the following experiment, i use mean of column to replace numerical column's NAN, and use 'NAN' to replace all NANs in string column.

In the initial tries, we use parameters as following

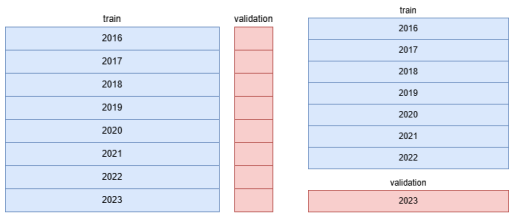
```
iterations = 500, depth = 5, learning_rate = 0.05, loss_function = 'Logloss'
```

This is our first algorithm that reaches 0.58 in public test. However, it is still not good enough to pass benchmark.

To find a better set of parameters, we use optuna to help us find it. And to mimic to stage1 / stage2 situation we will met, we use following different strategy to cut training and validation test for optuna.

stage1 Since we are predicting the last half season given each year's first half season, we use the last 0.2 of each season's data as validation data, and other as training data, and feed these to optuna to find the best parameters.

stage2 We are predicting a whole new year. Hence we take 2023's data as validation data, and other as training data, and feed these to optuna to find best parameters.



When using optuna, we search in following range

```
params = {
    "iterations": trial.suggest_int("iterations", 100, 1000),
    "depth": trial.suggest_int("depth", 4, 10),
    "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.3, log=True),
    "l2_leaf_reg": trial.suggest_float("l2_leaf_reg", 1, 10),
    "bagging_temperature": trial.suggest_float("bagging_temperature", 0, 1),
    "random_strength": trial.suggest_float("random_strength", 0.1, 10),
    "loss_function": "Logloss",
}
```

The best parameters lead to a 0.57985 accuracy in stage1. And 0.54493 in stage2.

4 Conclusion