Classification of unlabeled LoL match records with "Win/Loss"

Author: 徐淼 Student:201930650116

I. Introduction

1) The Objectives of the Project

League of Legends (LoL) is one of the most played eSports in the world. Arguably, the fun of games lies within the uncertainty of their outcomes. There is nothing more boring than playing or watching a game with a predictable ending. Therefore, the creators of LoL have tried their best to match players with teammates and opponents of as similar skill level as possible.

In this project, I have access to about 3 Million match records of solo games as training set. Each record comprises of all publicly available game statistics of a match played by some gamer, including an important field called "winner". If "winner" is "1", the team 1 won the match, or vice versa.

In sum, the fields include:

- Game ID
- Creation Time (in Epoch format)
- Game Duration (in seconds)
- Season ID
- Winner (1 = team1; 2 = team2)
- First baron, dragon, tower, blood, inhibitor and Rift Herald (1 = team1; 2 = team2)
- The number of tower, inhibitor, Baron, dragon and Rift Herald kills each team has

My goal is to create one or more classifiers that take as inputs from any fields from above (except "winner") such match record, and labels this record as "1" or "2". The test set comprises of ~2 Million of such records.

2) Methodology

In my project, I choose Decision Tree, Knn and Ann model to train the dataset and predict the label for the testing dataset. Finally, using accuracy and running time to measure the training result.

II. Algorithms

1) Import Library and Data Processing

Bring in the required package from the libraries and process the dataset. In order to select the most helpful feature, we need to ignore the features whose variance

is 0. Then divide the given columns into two types of variables target variable and feature variable.

```
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import pydotplus
import os
import time
import numpy as np
```

```
# decision tree
col_names = ['gameId','creationTime','gameDuration','seasonId','winner','firstBlood','firstTower','firstInhibitor','firstBaron',
             'firstDragon','firstRiftHerald','t1_towerKills','t1_inhibitorKills','t1_baronKills',
            't1_dragonKills','t1_riftHeraldKills','t2_towerKills','t2_inhibitorKills','t2_baronKills',
            't2_dragonKills','t2_riftHeraldKills']
# load dataset
train_dataset = pd.read_csv("/Users/changfeng/Desktop/course/BigData/project/new_data.csv", header=None, names=col_names)
train_dataset = train_dataset.iloc[1:] # delete the first row of the dataframe
test_dataset = pd.read_csv("/Users/changfeng/Desktop/course/BigData/project/test_set.csv", header=None, names=col_names)
test_dataset = test_dataset.iloc[1:] # delete the first row of the dataframe

#split dataset in features and target variable
feature_cols = ['gameDuration','firstBlood','firstTower','firstInhibitor','firstBaron',
             'firstDragon','firstRiftHerald','t1_towerKills','t1_inhibitorKills','t1_baronKills',
            't1_dragonKills','t1_riftHeraldKills','t2_towerKills','t2_inhibitorKills','t2_baronKills',
            't2_dragonKills','t2_riftHeraldKills']
X_train = train_dataset[feature_cols] # Features
y_train = train_dataset.winner # Target variable
X_test = test_dataset[feature_cols] # Features
y_test = test_dataset.winner # Target variable
```

2) Classification

A. Decision Tree

♦ Building Decision Tree Model

Parameter	Reason
Criterion = gini	"gini" or "entropy" has little effect on accuracy, but "gini" takes less time.
max_depth = 10	It can not only achieve high accuracy but also avoid overfitting

Splitter = best	"best" is to find the optimal partition point among all partition points of a feature. "random" is to find the local optimal partition point in the partition point randomly. "best" is usually chosen when the data volume is not very large.

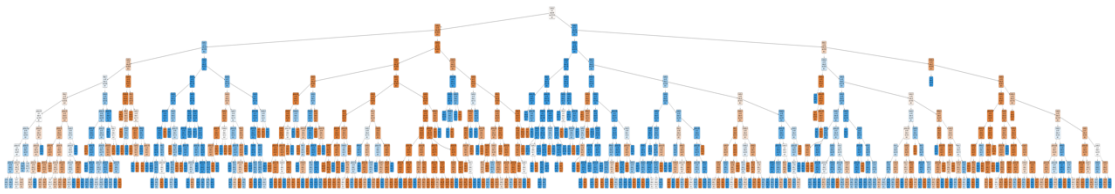
```
# Create Decision Tree classifer object
begin_time=time.time()
tree_clf = DecisionTreeClassifier(criterion="gini", max_depth=10,splitter='best')

# Train Decision Tree Classifer
tree_clf = tree_clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = tree_clf.predict(X_test)
```

- Visualizing

```
#visualizing
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
# Configure environment variables
dot_data = StringIO()
export_graphviz(tree_clf, out_file=dot_data,
filled=True, rounded=True,
special_characters=True,feature_names =
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



B. Knn

In Knn, each sample can be represented by its nearest K neighbors, which is very simple and easy to understand. Knn is more suitable for those datasets with more overlapping class domains to be divided.

Parameter	Reason
n_neighbors	Detectable number of neighbors around

```
knn_cls = KNeighborsClassifier(n_neighbors=5)
knn_cls = knn_cls.fit(X_train, y_train)
knn_predict = knn_cls.predict(X_test)
```

C. Ann

♦ Remap

What I want to do now is to change values from the winner column to 0 and 1.

```
mappings = { 1: 0, 2: 1 }
train_dataset['winner'] = train_dataset['winner'].apply(lambda x: mappings[x])
test_dataset['winner'] = test_dataset['winner'].apply(lambda x: mappings[x])
```

♦ Splitting Data and Convert Data

To start out, I split the training and testing dataset into features and target – or X and y. Then convert the split data from Numpy arrays to PyTorch tensors.

```
X_train = train_dataset.drop(['gameId','gameDuration','creationTime','seasonId','winner'], axis=1).values
y_train = train_dataset['winner'].values
X_test = test_dataset.drop(['gameId','gameDuration','creationTime','seasonId','winner'], axis=1).values
y_test = test_dataset['winner'].values
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
```

♦ Build the architecture of the model

Fully Connected Hidden Layer	16 input features (number of features in X), 16 output features
Output Layer	16 input features, 2 output features (number of distinct classes)
Activation Function	Sigmoid

```
class ANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(in_features=16, out_features=16)
        self.output = nn.Linear(in_features=16, out_features=2)
    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = self.output(x)
        x = F.softmax(x)
        return x
```

♦ Training Preparations

Criterion = CrossRntropyLoss	how we measure loss,

Optimizer = Adam with a learning rate of 0.01	optimization algorithm

```
begin_time3=time.time()
model = ANN()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

- Train the Model

Training the model for 100 epochs and keeping track of time and loss. Every 2 epochs, I will output to the console the current status – indicating on witch epoch are we and what' s the current loss.

```
epochs = 100
loss_arr = []
for i in range(epochs):
    y_hat = model.forward(X_train)
    loss = criterion(y_hat, y_train)
    loss_arr.append(loss)

    if i % 2 == 0:
        print(f'Epoch: {i} Loss: {loss}')

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

- Predict the label

```
predict_out = model(X_test)
_,predict_y = torch.max(predict_out, 1)
```

3) Evaluate the Accuracy and Running Time

Accuracy can be computed by comparing actual test set values and predicted values. Running time can be obtained by subtracting the start time and ending time.

III. Requirements

1) Time

Record the start and end times, and then obtain the running time.

2) Numpy

Store and process large matrices

3) Pandas

For data loading and manipulation

4) Pytorch

For model training

5) Sklearn

For model training and evaluate accuracy.

6) Image、pydotplus、os

For output visualization.

IV. Results

1) Table

	Accuracy	Running Time (s)
Decision Tree	0.9653648110366269	0.19413518905639648
Knn	0.9516661808996405	0.88569211959838887
Ann	0.9594870300204023	1.699631929397583

2) Screenshot

```
Tree classification Accuracy: 0.9653648110366269
Decision Tree running time is : 0.19413518905639648 s
```

```
Ann Accuracy: 0.9516661808996405
Running time for this classifer: 0.8856921195983887 s
```

```
Ann accuracy is  0.9594870300204023
Ann running time is : 1.699631929397583 s
```

V. Comparison and Discussion

1) What I have learnt:

1. How to Process the Dataset

In the project, some features with the same value or low variance are not

very helpful in training process. What' s more, some features are even completely useless such as Game ID and Creation Time. If one feature all has the same value such as Season ID, we need to drop it. For those features with order of magnitude differences, we need to normalize them, otherwise, the accuracy of classification will be greatly affected.

2. Use Classifier and Model to Train and Predict the Dataset

   When the datasets are different, the appropriate classifier is different and its parameters also varies. For example, Decision Tree Classifier behaves not very well when the dataset is very lager. Therefore, we should have a good command of the advantages and disadvantages of each classifier and choose a suitable classifier to do the classification. Then we are expected to use different parameter combination to find the best combination.

3. Vsualize the Output

   To put the result more clearly, we can use the plot package to make the boring numbers and alphabets to be a figure. From it, we can easily understand the classification process of the Decision Tree.

2) How to Improve:

1. Extend Dataset

   Add more data to make the training more accurate. What' s more, data augmentation is also a good way to extend dataset.

2. Assign Different Weight to Different Feature

   In the actual problem, different features have different effects on the result, which requires us assign different weight to different feature. For example, in LoL, the influence of the number of dragons on the victory is not simply "1+1=2" , but when the number reaches 4, it is of great help to the victory. Therefore, we should assign greater weight to the ID whose number of dragons is 4.

3. Use Cross-Validation

   Cross-Validation divides the whole dataset into N parts and take one part as the testing dataset and the other N-1 part as training part. Then, do the loop

for N-1 times. Finally, take the average accuracy for the final result. In this way, we can get a better training model with less dataset.

VI. Summary

From this project, I learned how to process the given dataset simply, train and predict with a variety of classifiers and test the effect of classification. This project give us an opportunity to apply the knowledge I have learned in class into practice, which enables me have a more comprehensive understanding of industrial big data and is very beneficial to my future study and development.