Use Multiple Classifiers to Predict the Winning Teams of LoL

1. Introduction:

1.1. Description of Objectives：

In this project, my objective is to create a number of classifiers that take as inputs from any fields (except “label”) and labels this record as a “1” or a “2” based on classification of labeled LoL match records of solo gamers of training set.

1.2. Description of Requirements：

Firstly, implementing at least one classification method we learn and successfully evaluate the test data to finish the project.

Secondly, the evaluation result should at least be higher than 50%.

Thirdly, be free to use Python or the libraries including we learnt.

Finally, getting a distinction unless you use more than one classification methods and try to compare their pros and cons, accuracy and training time and so on.

1.3. Description of the Methodology：

In this project, I mainly apply supervised machine learning including Decision Tree, Support Vector Machine, Art Neural Network, K-Nearest-Neighbour and Ensemble. Supervised machine learning firstly experiments with a dataset from given training set to update its algorithms and explores optimal classification results of given test set.

2. Algorithms:

2.1. Data Preprocessing:

2.1.1. Load Dataset

At the beginning of everything, I load given training and test dataset by pandas, a data analysis package and assign them to variables “df”, “df_target”.

And then I divide training dataset into features and results and mark them as “X”, “Y”. For test dataset, they are divided into “X_test” and “Y_test”.

2.1.2. Normalize the Dataset

Normalization is a technique often applied as part of data preparation for machine learning. As we can see, the order of magnitudes of former three columns are greater than data of other columns.

Fortunately, normalization can change the values of numeric columns to a common scale, without

distorting differences in the range of values.

To be specific, I apply “MinMaxscaler” function to normalize all data.

```
mm = MinMaxScaler()
X = mm.fit_transform(X) #Normalization
```

To implement the normalization, the function transform the value of a sample, **x** into a value between the maximum value and the minimum value of corresponding attribute. And new value of sample, **x**’ can be calculated as following formula:

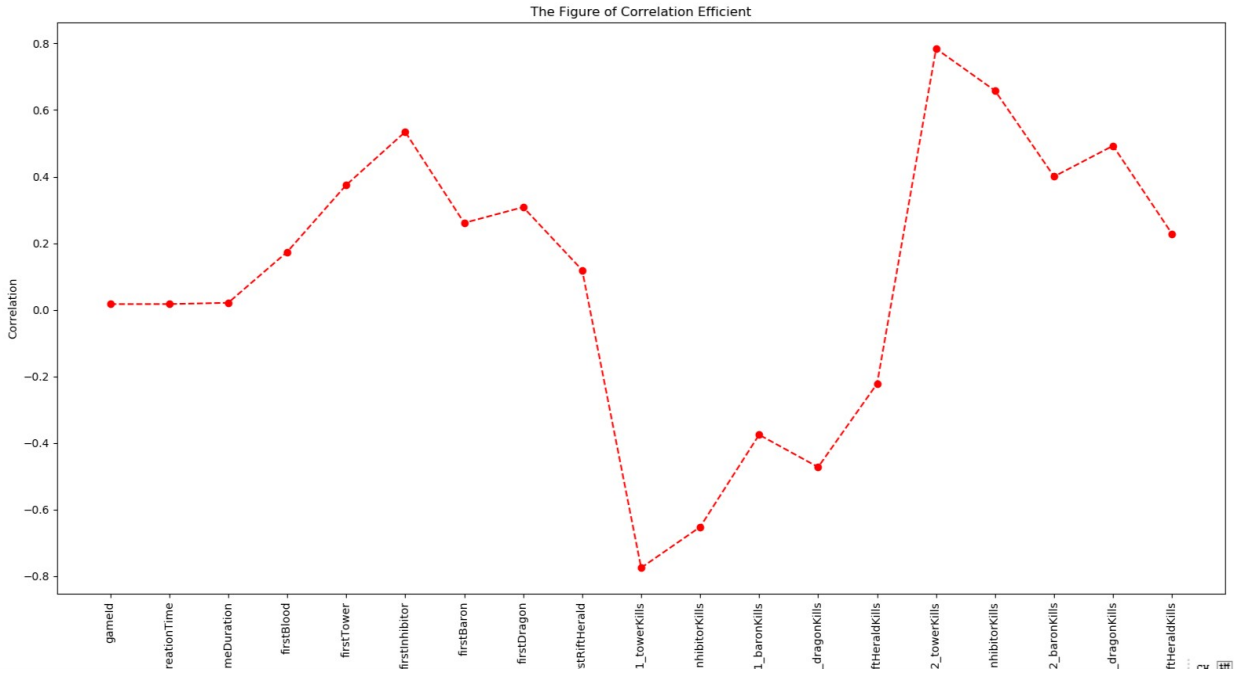
$$\mathbf{X'}=\frac{x - x_{min}}{x_{max} - x_{min}}$$

where x_{min} represent the minimum value of corresponding attribute, x_{max} represent the maximum value. X’ and x respectively represent updated sample value and original value.

2.1.3. Data Cleaning:

Real-world data tend to be noisy and inconsistent which also contribute to inaccurate results. Thus, data cleaning is necessary for removing noisy data to acquire better results.

The first step in data learning is discrepancy detection. To implement the function, I utilize python to make a line chart about the correlation between attributes and the labels.



As is known to everyone, the correlation coefficient is between [-1,1]. Nevertheless, data of the former 3 columns must be removed due to 0 values. And operation shows as following:

```
X_train = X_train.drop(['gameId', 'creationTime', 'gameDuration'], axis=1).values
X_test = X_test.drop(['gameId', 'creationTime', 'gameDuration'], axis=1).values
```

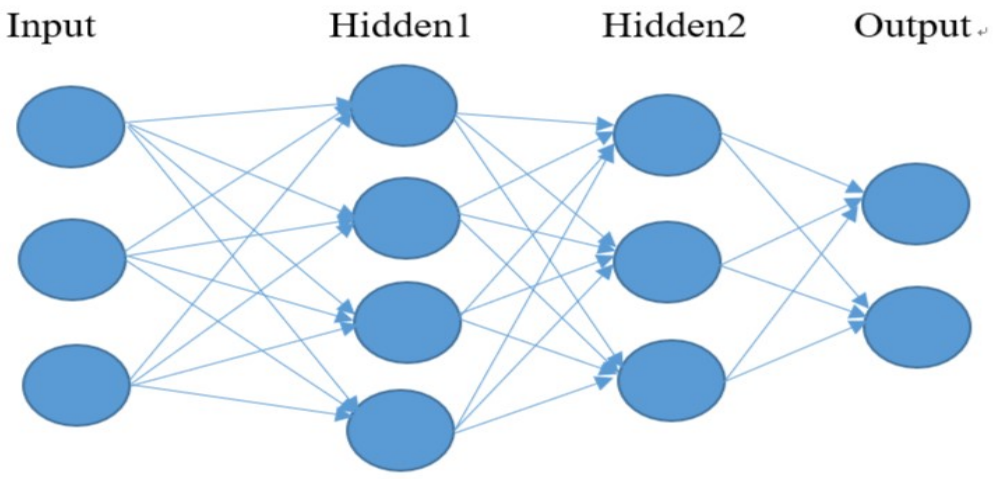
2.2. Classification Methods

In the process of implementing the classification, I apply ANN, SVM, DT, KNN and Ensemble classifiers.

2.2.1. ANN:

2.2.1.1. ANN Theory:

Artificial Neural Networks(ANNs) are a typical model of machine learning inspired by biological neural networks and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. ANNs are generally presented as systems of interconnected "neurons" which exchange message with each other. The connections have numeric weights that can be turned based on experience, making nets adaptive to inputs and capable of learning.

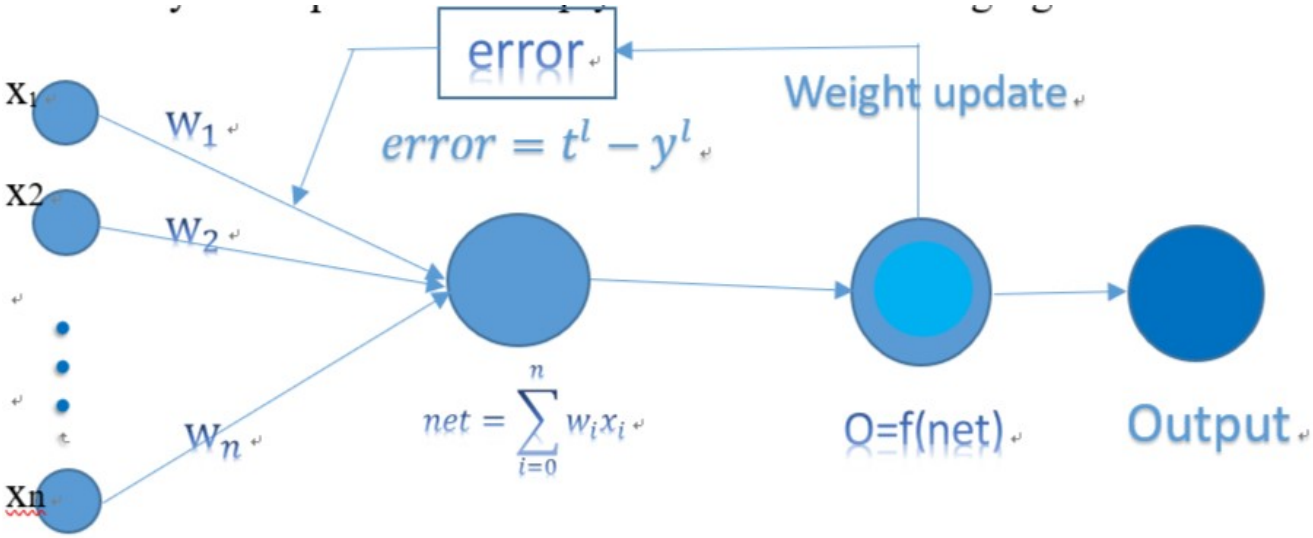


As above simple figure showing, ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data.

Specifically speaking, backpropagation algorithm plays a critical role in the whole ANN model. It learns by iteratively processing a data set of training tuples, comparing the network's prediction for each tuple with the actual known target value. For each training tuple, the weights are modified so as to minimize the mean-squared error between the network's prediction and the actual target value. These

modification are made in the “backwards” direction through each hidden layer back to the former layer.

All process is simply showed as following figure:



Thee figure take a neuron as example and steps are described concretely next:

Initialize the weights: The weight in the network are initialized to small random numbers, such as 1, 0, -1 and so on.

Propagate the inputs forward: First of all, the training tuple is fed to the network’s input layer. Then input value, I_j, will multiply their corresponding weights to form a weighted sum.

$$\mathbf{Net}=\sum_i w_{ij} I_j+\theta_j$$

where I_j is equal to its input value, W_{ij} is corresponding weight of I_j, $\boldsymbol{\theta_j}$ is the bias of the unit.

And then the net input will apply a nonlinear activation function to become a new output, noted O_j.

$$\mathbf{O_j=f(net)}$$

where f is corresponding function of net.

Backpropagate the error: After computing the output values, O_j, it will be compared with the actual target value so that error is produced. Then the error is propagated backward by updating the weights and biases to reflect the error of the network’s prediction. For a unit j in the output layer, the error will have a specific expression.

$$\mathbf{E_{rrj}=H(O_j,T_j)}$$

where T_j is the known target value of the given training tuple. E_{rrj} is the j^{th} error of the node. H is a function which is depended on the activation function.

Next, weight begin to update and iterate. The process can also be called epoch updating, where one

iteration through the training set is an epoch.

$$\Delta w_{ij} = (l)\, E_{rrj}\, O_j$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

where the l is learning rate, which helps learning avoid getting stuck at a local minimum in decision space.

2.2.1.2. Build Appropriate Ann Model:

To implement mentioned model, I create a class named "ANN" in python. The class is made up of two functions, of which part 1 is initializing the number of layers and number of nodes that each layer has and the other is define a forward function which converts the value of each node to the final output through sigmoid function. In this way, I build a four layers feed-forward neural network which contains two hidden layers.

In the class, I assign the number of nodes for each layer to n1, n2 and n3 variable so as to change them flexibly.

```
#Define ANN Function
class ANN(nn.Module):
    def __init__(self,n1,n2,n3):
        super().__init__()
        self.fc1 = nn.Linear(in_features=n1, out_features=n2)
        self.fc2 = nn.Linear(in_features=n2, out_features=n3)
        self.output = nn.Linear(in_features=n3, out_features=2)
    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.output(x)
        x = F.softmax(x)
        return x
```

Sigmoid Function: $f(x) = \frac{1}{1+e^{-x}}$

And then, I utilize two built-in functions, one is "criterion" function, the other is "optimizer function", which both play a vital role in the propagation. The former function is used to calculate the error the model needs. The latter function is applied to recall initial model and introduce some parameters like learning rate and so on. The model define error as variable named "loss".

Loss=criterion(y_hat, y_train)

where y_hat represents known labels of data and y_train are the predicted labels of data.

In this way, I set the number of epoch is 50 so that the model can modify its weight adequately and

take as little as possible time to finish the task.

Finally, after the process of training model is over, the test data can apply it so as to predict classification labels.

2.2.1.3. Adjust Appropriate Parameters for Model:

What's more, building a perfect ANN model should focus on the parameters of ANN. In particular, the number of hidden layers, the number of nodes of each layer and the number of epochs all make a difference on the final accuracy.

The process of determining the values of parameter is described concretely next:

Determine the number of hidden layers:

As we all known, the reason why BP network can approximate any nonlinear mapping relationship is that the networks possess adequate hidden-layers and nodes of each layer. However, excess quantity will complicate the network so that the tendency of "overfitting" increases. Thus, it is of importance to determine the relatively accurate layers.

Yin Nian-dong[1] suggested a method to determine the number of hidden-layer and nodes of each layer in his article "Design of BP neural network in engineering application". He applies a multi-dimensional unit hypercube to simulate networks with multi-dimensional input.

Ultimately, he concludes that one hidden-layer is adequate for a continuous activation function and at most two layers are adequate for a discrete function.

The result of these operations is passed to other neurons. The output at each node is called its activation or node value.

Therefore, I follow these principles that only applying one layer for any practical problem and use as few as possible layers. If it doesn't turn out well, I will apply two hidden-layer.

When all other conditions are same, the former model with hidden-layer 20 and the latter model with hidden-layer 20 and 10 both are applied to experiment test data.

And the comparison results show as following:

Test Order	Accuracy of one hidden-layer	Accuracy of two hidden-layer
1	0.96376	0.96506
2	0.95869	0.96462

[1]

3	0.96365	0.96409
4	0.96506	0.96516
5	0.96084	0.96258

Obviously, the accuracy of two hidden-layer is a little higher than the other.

Thus, I apply two hidden-layer for my model finally.

Determine the number of neurons of each hidden layers:

Compared to the previous parameter, the number of neurons for each layer has more significant meaning. If the number is small, the network only get too little information. However, if the number is big, it is difficult to do it in a time that people can accept training and also causes mentioned over-fitting.

Refer to the research results of domestic scholars, I will utilize their algorithms to acquire the parameter.

Golden Section Method:

Xia Ke-wen[2] presents a golden section method to solve the problem well.

I apply the method to build a model to get parameters and concrete steps show as following:

Determine Boundary Value:

At present, for the nodes of input-layer, hidden-layer and output-layer n_i,n_h, and n_o.

In terms of a large number of existing experiments at home and abroad, perfect number of nodes usually occurs frequently in the following range:

$$a=\frac{n_i+n_o}{2} \le n_h \le (n_i+n_o)+10=b$$

So I get a initial range [9,28].

The Key of Golden Section:

In order to satisfy the requirement of high precision approximation, it continue to expanded the search space based on the golden section principle. In this way, we get a more precise range [b,c] to search best parameter.

b=0.618*(c-a)+a

So the range updates to [28,39].

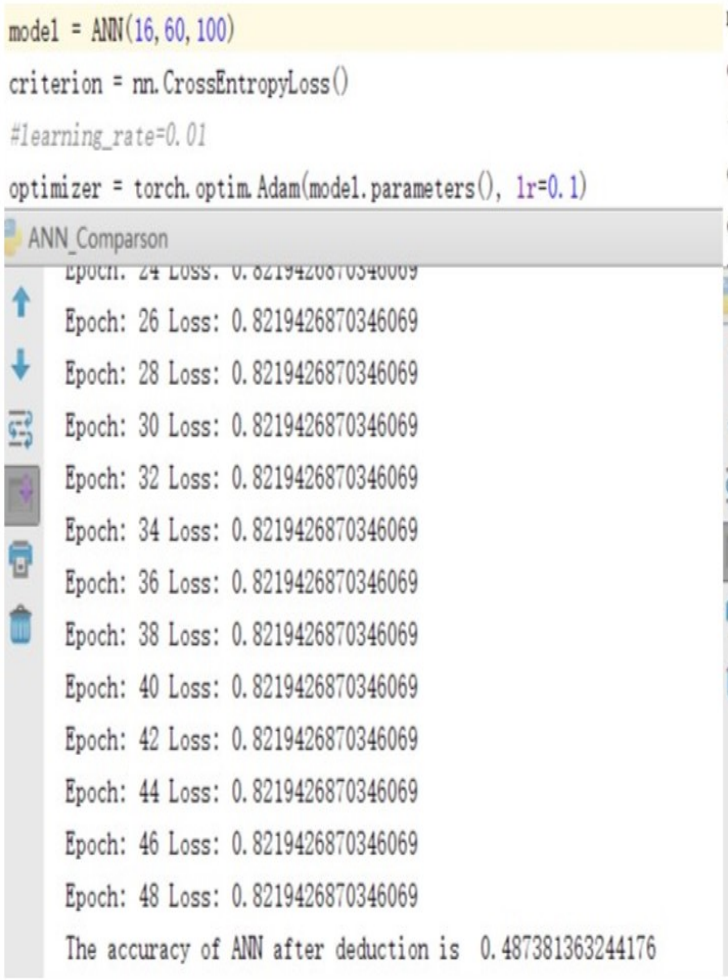
2

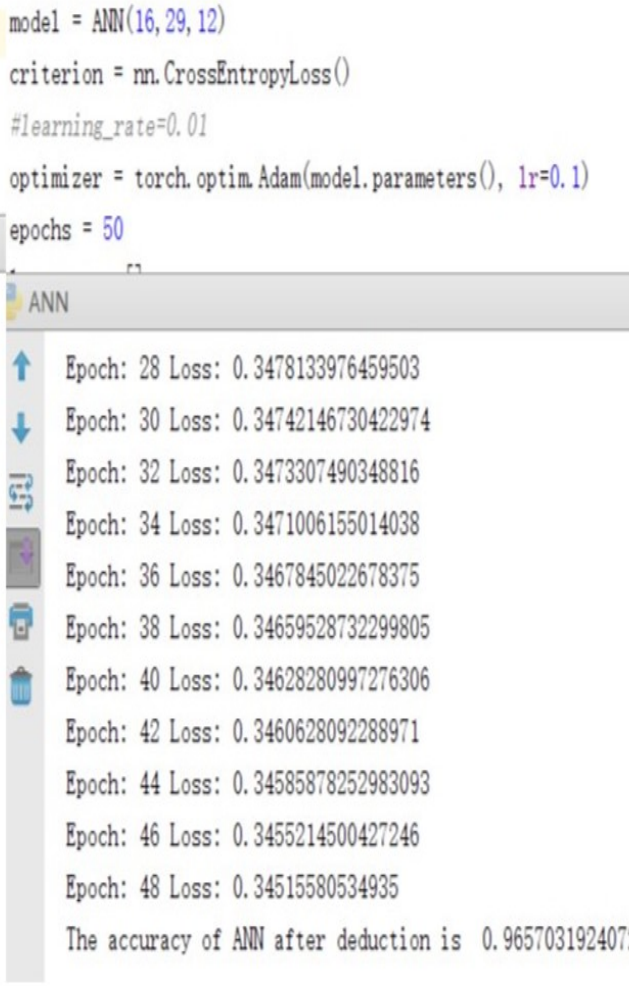
Different values are tested:

Test Order	Accuracy of hidden-layer(28-39,9-28)	Accuracy of hidden-layer(9-28,28-37)	Accuracy of hidden-layer(random, random)
1	**0.96452 (16, 28, 10)**	**0.96409 (16, 20, 37)**	**0.48738 (16, 3, 50)**
2	**0.96570（16，35，11）**	**0.96398（16，9，29）**	**0.96591（16，60，60）**
3	**0.96300（16，38，15）**	**0.95837（16，12，30）**	**0.48738 （16，60， 100）**
4	**0.96570（16，29，12）**	**0.96047（16，22，30）**	**0.96247（16，12，4）**
5	**0.96441（16，31，9）**	**0.96139（16，19，35）**	**0.48738（16，4，52）**

where the numbers in bracket respectively represent input layer, first hidden-layer and second hidden-layer.

Some of the program results are shown below:





Choose the number of nodes:

In terms of the test results, if we randomly select nodes of each layer, the bad accuracy will appear as column 4. Instead, the accuracy of column 2 and column 3 both are higher than 95%. Besides, the results of column 2 are better than the column 3.

Thus, the number of neurons of first hidden-layer will be in range [b,c], and next number will be selected from range [a,b].

Determine the number of epoch:

Test Order	**Epoch=50 (Time, Accuracy)**		**Epoch=100 (Time, Accuracy)**		**Epoch=200 (Time, Accuracy)**	
1	**1.727s**	**96.29%**	**3.803s**	**96.76%**	**7.207s**	**96.76%**
2	**1.762s**	**96.58%**	**3.581s**	**96.92%**	**7.105s**	**96.81%**
3	**1.744s**	**96.49%**	**3.500s**	**96.87%**	**7.146s**	**96.95**
4	**1.769s**	**96.09%**	**3.677s**	**96.95%**	**7.192s**	**97.00%**

From above table, it has a tendency that the accuracy will be higher as the number of training epochs are more. However, it presents a decline of rate of rise. For the case with epoch=100 and the case with epoch=200, the former only use half time and approximate accuracy, so I select epoch=100 as one parameter for ANNs model.

2.2.2. SVM:

2.2.2.1. SVM Theory:

Support Vector Machine(SVM), the algorithm uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane. The SVM finds this hyperplane using support vectors("essential" training tuples) and margins. Then it can classify successfully.

2.2.2.2. The Parameter of SVM:

After 30 training, I select penality coefficient C as 1 and kernel function as "rbf", also called "Radial basis function" based their best accuracy. As the final result table shows, it get a accuracy higher than 96%.

2.2.3. KNN:

2.2.3.1. KNN Theory:

Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. As the training tuples are described by n attributes, all the training tuples are stored in an n-dimensional pattern space.

For a unknown tuple, KNN classifier search the pattern space for the k training tuples that are closest to the unknown tuple.

"Closeness" is defined in terms of a distance metric, such as Euclidean distance. For example, two tuples' distance shows as following:

$$\mathrm{Dist}(X_1,X_2)=\sqrt{\sum_{i=1}^{n}(x_{1i}-x_{2i})^2}$$

2.2.3.2. The parameter of KNN:

After many trainings, I select n=5 as initial value for KNN to run. As the final result table shows, it get a accuracy higher than 95%.

2.2.4. Decision Tree:

2.2.4.1. DT Theory:

Decision Tree mainly selects best attributes which satisfy next entropy reaches minimum and utilize the attribute to classify. In this way, it starts this process steps and steps until the data is completely classified.

In the whole process, it mainly applies Gini index to evaluate the entropy:

$$\text{Gini Index}=1-\sum_{i=0}^{c-1} p_i(t)^2$$

where i represents i^{th} class based current attribute, c is the total number of classes and $P_i(t)$ represents the frequency of the i class.

2.2.4.2. Parameter of DT:

The function "DecisionTreeClassifier" in Python can finish the whole process with no need for providing any parameters.

2.3. Fusion:

I finally make a fusion for above all classifiers. Actually, it is also called "ensemble methods".

2.3.1. Ensemble Function:

An ensemble combines a series of learned models and returns a class prediction based on the votes of the base classifiers. By majority voting, the ensemble will avoid some mistakes when the base classifiers may make. Thus, it will acquire significant effect when different classifiers is difficult to distinguish data of different kinds.

2.3.2. Build the Fusion Model:

As function mentions, I apply majority voting method to combine all base classifiers.

```
#Apply Fusion Method
# Create Parameter number of neighbour of Ensemble classifer object
#Record the time of running process
start=time.time()
voting_clf=VotingClassifier(
    estimators=[('SVM',clf2),('DT',clf1),('KNN',clf3),('MLP',clf4)],voting='hard'
)
voting_clf.fit(X_train,Y_train)
```

3. Requirements:

To implement the whole project efficiently, I use some packages which are prerequisite conditions for my code and they are showed as following:

3.1. Prerequisite Package:

Name	Function
Numpy	A package can convert other types data into array type.
Pandas	A package help read data from the csv. document
Sklearn	A important library for machine learning
torch	A package is used to build ANNs model.
time	A package can record time that process takes.
matplotlib	A package serves as draw tool of python.
pylab	A package serves as table tool of python.

3.2. Concrete Functions for Specific Package:

Name	Package		
Torch.nn	Torch	Build and improve ANNs model	
Torch.nn.function			
MinMaxScaler	Sklearn	Normalization	
Accuracy_score		Measure accuracy	
DecisionTreeClassifier		Introduce corresponding function	
KNeighborsClassifier			
SVC			
MLPClassifier			
VotingClassifier			

4. Results:

Our task is to predict labels. In terms of known labels of test labels, so I present result as accuracy.

In order to avoid contingency, I calculate the mean value of ten results.

Methods (time, accuracy)	SVM		DT		ANNs		KNN		Fusion(ensemble)	
1	14.09s	96.56%	0.0668s	96.16%	3.590s	96.62%	4.730s	95.96%	47.850s	96.66%
2	15.39s	96.56%	0.0622s	96.08%	3.557s	96.64%	2.879s	95.96%	52.747s	96.67%
3	9.178s	96.56%	0.0609s	96.06%	3.548s	96.94%	6.549s	95.96%	42.054s	96.69%
4	18.555s	96.56%	0.0597s	96.13%	3.572s	96.82%	6.227s	95.96%	53.098s	96.62%
5	18.072s	96.56%	0.1540s	96.06%	3.550s	96.67%	6.484s	95.96%	52.565s	96.65%
6	17.946s	96.56%	0.171s	96.08%	3.564s	96.69%	6.722s	95.96%	40.038s	96.62%
7	18.371s	96.56%	0.104s	96.05%	3.609s	96.97%	6.293s	95.96%	51.405s	96.62%
8	14.337s	96.56%	0.0647s	96.12%	3.548s	96.77%	5.440s	95.96%	49.227s	96.68%
9	17.786s	96.56%	0.0678s	96.07%	3.734s	96.88%	5.879s	95.96%	53.988s	96.67%
10	16.523s	96.56%	0.0597	96.17%	3.550s	96.85%	5.972s	95.96%	45.942s	96.72%
Mean Value	**16.025s**	**96.56%**	**0.0871s**	**96.10%**	**3.582**	**96.79%**	**5.718s**	**95.96%**	**48.891s**	**96.50%**

Photos Captured from the Computer:

```
#Apply Fusion Method
# Create Parameter number of neighbour of Ensemble classifer object
#Record the time of running process
start=time.time()
voting_clf=VotingClassifier(
    estimators=[('SVM',clf2),('DT',clf1),('KNN',clf3),('MLP',clf4)],voting='hard'
)
voting_clf.fit(X_train,Y_train)
Y_pred = voting_clf.predict(X_test)
b=accuracy_score(Y_test, Y_pred)
```

```
Project_01
E:\Anaconda3\python.exe "E:/International Lessons/Intenational Lessons/Big Data Techn
The Accuracy of DT for tast data is 0.961284368017099
The time for DT running test data is 0.23231768608093262 s
The Accuracy of SVM for test data is: 0.965607694549694
The time for SVM running test data is 7.500975847244263 s
The Accuracy of KNN for test data is: 0.9596327601282425
The time for KNN running test data is 2.3456852436065674 s
E:\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
  % self.max_iter, ConvergenceWarning)
E:\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
  % self.max_iter, ConvergenceWarning)
The Accuracy of Voting Fusion for test data is: 0.9663363450888953
The time for Ensemble running test data is 22.44476819038391 s
```

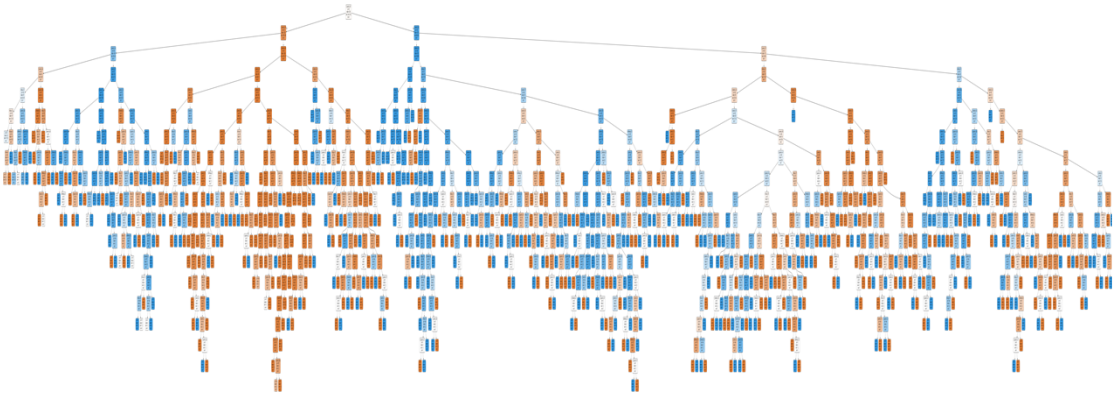
```
The accuracy of ANN after deduction is  0.9665792286019625
The time for ANN running test data is 4.02023720741272 s
```

Where my ANN model and other methods have different building methods, so I separate it alone.

5. Comparison and Discussion:

As above result table showing, different methods perform very different.

For DT algorithm, it always gets good results and takes least time. Besides, I can also distinguish the influence between different attributes and labels based DT figure. But for continuous data and data with numerous classes, it will cause some errors.



For ANN algorithm, it performs highest accuracy although takes relatively long time. Thus, it's obvious that ANN can perform excellent encountering a large sample even with complex features. However, its parameters need much time to determine even not best because its parameter like neurons of each layer has significant impact. Besides, from its principle, it depends on vast training datasets.

For KNN, it seems to operate simply and takes relatively little time but it's difficult to determine initial parameter K particularly in vast dataset. Thus, if we can get the algorithm to determine K, it will perform better.

For SVM, I still need to determine a number of parameters. Although it performs better, it takes a relatively time.

For the fusion, it always produces a stable and good accuracy but not best. Based the analysis of ensemble's theory, I speculate that it misclassify about 4% training set which also makes over half of base classifiers make mistakes.

6. Summary:

In conclusion, utilizing machine learning to predict the winning teams based on the LOL data is an excellent method with an accuracy at least 95.9%. From my results, I think ANNs model is best when vast and complicate data need to be analyzed.

Indeed, if I have more time, the results will be better. On the one hand, I will be more cautious to determine the parameters of KNN and SVM. At the same time, I will think more fully on select

ensemble method like Random Forest and so on.

It's worthy mentioning that I gain a lot of experience from the project. First of all, I understand machine learning with a greater extent. Second, I also try my best efforts to make results as better as possible. In particularly, I read a lot of literatures concerning relevant studies and get better results after applying their methods. I find great joy when exploring the process. Finally, I understand the accuracy of science because I finally choose ANN but not KNN only due to the difference in accuracy of 1%.

7. Reference:

[1] Yin Nian-Dong, Design of BP neural network in engineering application[J], INFORMATION TECHNOLOGY, 2003.

[2] Xia Ke-Wen, Li Chang-Biao and Shen Jun-Yi. An Optimization Algorithm on the Number of Hidden Layer Nodes in Feed-forward Neural Network[J], COMPUTER SCIENCE, 2005(10):143-145.