Machine Learning Assignment

Ensemble Methods

Importing

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn. ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
```

In [2]:

```
data = pd.read_csv('games.csv')
```

In [3]:

```
data
```

Out[3]:

	gameId	creationTime	gameDuration	seasonId	winner	firstBlood	firstTower	firs
0	3326086514	1504279457970	1949	9	1	2	1	
1	3229566029	1497848803862	1851	9	1	1	1	
2	3327363504	1504360103310	1493	9	1	2	1	
3	3326856598	1504348503996	1758	9	1	1	1	
4	3330080762	1504554410899	2094	9	1	2	1	
...	...	...	...	...	...	...	...	
51485	3308904636	1503076540231	1944	9	2	1	2	
51486	3215685759	1496957179355	3304	9	2	1	1	
51487	3322765040	1504029863961	2156	9	2	2	2	
51488	3256675373	1499562036246	1475	9	2	2	2	
51489	3317333020	1503612754059	1445	9	1	1	1	

51490 rows × 61 columns

As our code was taking very long time so we reduced the dataset from 51489 to 5000 rows

In [4]:

```
data = pd.read_csv('games.csv', nrows=5000)
```

In [5]:

```
data
```

Out[5]:

	gameId	creationTime	gameDuration	seasonId	winner	firstBlood	firstTower	firstI
0	3326086514	1504279457970	1949	9	1	2	1	
1	3229566029	1497848803862	1851	9	1	1	1	
2	3327363504	1504360103310	1493	9	1	2	1	
3	3326856598	1504348503996	1758	9	1	1	1	
4	3330080762	1504554410899	2094	9	1	2	1	
...	...	...	...	...	...	...	...	
4995	3321230613	1503926907856	1849	9	2	2	1	
4996	3329225554	1504472840709	1512	9	2	2	2	
4997	3324575521	1504178461333	1694	9	2	1	1	
4998	3322395687	1504007920743	2292	9	2	1	2	
4999	3329269426	1504477286180	1852	9	2	1	2	

5000 rows × 61 columns

Since in our data winner 1 is for team 1 and 2 is for team 2. So we will make it to 0 and 1 for logistic regression.

In [6]:

```
data['winner'] = data['winner'].replace(1, 0)
data['winner'] = data['winner'].replace(2, 1)
data['winner'].unique()
```

Out[6]:

```
array([0, 1], dtype=int64)
```

In [7]:

```
data
```

Out[7]:

	gameId	creationTime	gameDuration	seasonId	winner	firstBlood	firstTower	firstI
0	3326086514	1504279457970	1949	9	0	2	1	
1	3229566029	1497848803862	1851	9	0	1	1	
2	3327363504	1504360103310	1493	9	0	2	1	
3	3326856598	1504348503996	1758	9	0	1	1	
4	3330080762	1504554410899	2094	9	0	2	1	
...	...	...	...	...	...	...	...	
4995	3321230613	1503926907856	1849	9	1	2	1	
4996	3329225554	1504472840709	1512	9	1	2	2	
4997	3324575521	1504178461333	1694	9	1	1	1	
4998	3322395687	1504007920743	2292	9	1	1	2	
4999	3329269426	1504477286180	1852	9	1	1	2	

5000 rows × 61 columns

Spliting data into training and testing

In [8]:

```
X = data[['t1_baronKills', 't1_dragonKills', 't2_baronKills', 't2_dragonKills',
          't1_towerKills','t1_inhibitorKills','t2_towerKills','t2_inhibitorKills']]
y = data['winner']
```

In [9]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

Support Vector Machine SVM

Support Vector Machine SVM is a supervised learning algorithm

- We used linear and radial basis function (RBF) kernal to check their accuracy
- We know that in case of linear data SVM with linear kernal performs better
- Based on our dataset we can use soft SVM
- So in case of our data SVM with kernal RBF have more accuracy than SVM with linear kernal

In [10]:

```
model = SVC(kernel='rbf')
```

In [11]:

```
model.fit(X_train, y_train)
prediction = model.predict(X_test)
```

In [12]:

```
print(classification_report(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       770
           1       0.97      0.96      0.96       730

    accuracy                           0.96      1500
   macro avg       0.96      0.96      0.96      1500
weighted avg       0.96      0.96      0.96      1500
```

In [13]:

```
model.score(X_test, y_test)
```

Out[13]:

```
0.9633333333333334
```

In [14]:

```
model = SVC(kernel='linear')
model.fit(X_train, y_train)
prediction = model.predict(X_test)
print(classification_report(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       770
           1       0.97      0.95      0.96       730

    accuracy                           0.96      1500
   macro avg       0.96      0.96      0.96      1500
weighted avg       0.96      0.96      0.96      1500
```

In [15]:

```
model.score(X_test, y_test)
```

Out[15]:

```
0.9566666666666667
```

Bagging

It combines multiple learners in a way to reduce the variance of estimates.

- We applied bagging with n_estimator 5, 20 and 150
- Since our accuracy was already very high so their was small increase with increasing value of n_estimator
- We saw that for n_estimators = 5 our accuracy is 0.959, it increased for n_estimators = 20,150 to 0.96

With subset 20

In [16]:

```
bg = BaggingClassifier(SVC(kernel='rbf'), max_samples= 0.5, max_features = 1.0, n_estimator
bg.fit(X_train,y_train)
```

Out[16]:

```
BaggingClassifier(base_estimator=SVC(C=1.0, break_ties=False, cache_size=20
0,
                                     class_weight=None, coef0=0.0,
                                     decision_function_shape='ovr', degree=
3,
                                     gamma='scale', kernel='rbf', max_iter=-
1,
                                     probability=False, random_state=None,
                                     shrinking=True, tol=0.001, verbose=Fals
e),
                  bootstrap=True, bootstrap_features=False, max_features=1.
0,
                  max_samples=0.5, n_estimators=20, n_jobs=None,
                  oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

In [17]:

```
bg.score(X_test,y_test)
```

Out[17]:

```
0.96
```

With subset 5

In [18]:

```
bg = BaggingClassifier(SVC(kernel='rbf'), max_samples= 0.5, max_features = 1.0, n_estimator
bg.fit(X_train,y_train)
bg.score(X_test,y_test)
```

Out[18]:

```
0.96
```

With subset 150

In [19]:

```
bg = BaggingClassifier(SVC(kernel='rbf'), max_samples= 0.5, max_features = 1.0, n_estimator
bg.fit(X_train,y_train)
bg.score(X_test,y_test)
```

Out[19]:

```
0.96
```

Boosting

Ada-boost or Adaptive Boosting is one of ensemble boosting classifier. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier.

- n_estimators is Number of weak learners to train iteratively.
- learning_rate contributes to the weights of weak learners. It uses 1 as a default value.

we can see that adaboost is trying to overfit the training dataset but still we got approx. same accuracy

With n_estimator 50

In [20]:

```
adb = AdaBoostClassifier(SVC(probability=True, kernel='rbf'),n_estimators = 50, learning_ra
adb.fit(X_train,y_train)
adb.score(X_test,y_test)
```

Out[20]:

```
0.9306666666666666
```

With n_estimator 100

In [21]:

```
adb = AdaBoostClassifier(SVC(probability=True, kernel='rbf'),n_estimators = 150, learning_r
adb.fit(X_train,y_train)
adb.score(X_test,y_test)
```

Out[21]:

```
0.9426666666666667
```

End