In [1]:

```
from sklearn.metrics import roc_auc_score
import pandas as pd
import numpy as np
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
```

From all the experimentation that was done using various models, the most that gave the best performance(high Test AUC score) was **CatBoost**

Using Catboost Model

Function 1 : Predicting the class label

In [2]:

```
def predict_class_catboost(input):
    """this function predicts the final o/p class label"""
    train=pd.read_csv('train.csv')
    test=input
    train_data=train.drop(columns=['ACTION'],axis=1)
    test_data=test.drop(columns=['id'],axis=1)
    y_true = train['ACTION']
    train_data.shape,test_data.shape,y_true.shape
    categorical_features = list(range(train_data.shape[1]))

    params = {'loss_function':'Logloss',
             'eval_metric':'AUC',
              'cat_features':categorical_features,
              'verbose':200,
              'random_seed':42}

    model = CatBoostClassifier(**params)
    model.fit(train_data,y_true)
    output = model.predict(test_data)
    return output
```

In [3]:

```
input_data=pd.read_csv('test.csv')
predict_class_catboost(input_data[:10])
```

```
Learning rate set to 0.045713
0:      total: 186ms    remaining: 3m 5s
200:    total: 23.5s    remaining: 1m 33s
400:    total: 52s      remaining: 1m 17s
600:    total: 1m 19s   remaining: 52.9s
800:    total: 1m 47s   remaining: 26.7s
999:    total: 2m 15s   remaining: 0us
```

Out[3]:

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

Function 2 : predicting the performance metric

In this problem, we were given two csv files, train.csv and test.csv . While the train.csv file contains the output class label for all data points, the test.csv file doesn't contain the output class label which means we can't calculate the AUC metric manually . We need to know the class labels for the data to calculate the AUC score manually.

So we have only two options.

1. Split the train.csv into train and test data , fit the model using training data and calculate the performance metric using test data
2. Use all the data in train.csv to train the model, use the data in test.csv and get the class probabilities , store it in a csv file and upload it in the kaggle comptetition to get the AUC score.

I am choosing option 1 here

In [4]:

```
def predict_metric_catboost(X_train,X_test,y_train,y_test):
    '''this function calculates the performance metric for given train and test data'''
    categorical_features = list(range(train_data.shape[1]))
    params = {'loss_function':'Logloss',
             'eval_metric':'AUC',
              'cat_features':categorical_features,
              'verbose':200,
              'random_seed':42}
    model = CatBoostClassifier(**params)
    model.fit(X_train,y_train)
    pred = model.predict_proba(X_test)[:,1]
    return roc_auc_score(y_test,pred)
```

In [5]:

```
train=pd.read_csv('train.csv')
train_data=train.drop(columns=['ACTION'],axis=1)
y_true = train['ACTION']
X_train, X_test, y_train, y_test = train_test_split(train_data,y_true, test_size=0.25,stratify=y_true)
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
print('Performance metric: AUC :',predict_metric_catboost(X_train,X_test,y_train,y_test))
```

```
(24576, 9) (8193, 9) (24576,) (8193,)
Learning rate set to 0.040428
0:      total: 95.8ms   remaining: 1m 35s
200:    total: 17.8s    remaining: 1m 10s
400:    total: 39.4s    remaining: 58.9s
600:    total: 1m 2s    remaining: 41.3s
800:    total: 1m 23s   remaining: 20.8s
999:    total: 1m 46s   remaining: 0us
Performance metric: AUC : 0.8995517936725808
```

Since the CatBoost model, doesnt require any feature transformations, the task was much simpler as you need to just fit the model and predict the class label/calculate the AUC score.

Therefore I am trying to write the same functionality using **random forest model that also includes frequency encoded features**

Using Random Forest + Frequency Encoding model

In [6]:

```
train_dict ={}
test_dict={}
def freq_encoding(train,test,each):
    values=train[each].value_counts()/len(train)
    v1 =train[each].map(values).fillna(0)
    v2 =test[each].map(values).fillna(0)
    return v1,v2


def preprocessing(train,test):
    for each in train.columns:
        v1,v2 = freq_encoding(train,test,each)
        train_dict[each+'_train']=v1
        test_dict[each+'_test']=v2

    return train_dict,test_dict
```

Function 1 : Predicting the class label

In [7]:

```
def pred_output(input):
    #get train data and test data
    train=pd.read_csv('train.csv')
    train_data=train.drop(columns=['ACTION'],axis=1)
    test=input
    test_data=test.drop(columns=['id'],axis=1)
    print('Shape of train and test data:',train_data.shape,test_data.shape)
    print('#'*50)

    #performing data preprocessing
    train_fc,test_fc = preprocessing(train_data,test_data)
    train_fc=pd.DataFrame(train_fc)
    test_fc=pd.DataFrame(test_fc)
    print('Shape of frequency coded data:',train_fc.shape,test_fc.shape)
    print('#'*50)

    #Concatenate data
    mod_train = pd.concat((train_data,train_fc),axis=1)
    mod_test = pd.concat((test_data,test_fc),axis=1)
    print('Concatenated data shape:',mod_train.shape,mod_test.shape)
    print('#'*50)

    from sklearn.ensemble import RandomForestClassifier
    model=RandomForestClassifier(n_estimators=1000,max_depth=25,max_features=5,
                             min_samples_split=2,
                             random_state=42,class_weight='balanced',n_jobs=-1)
    model.fit(mod_train,y_true)
    print('Model fitted using training data')
    #predicting the labels
    pred = model.predict(mod_test)
    return pred
```

In [8]:

```
test=pd.read_csv('test.csv')
print('Predicted class for the provided input is ', pred_output(test[:10]))
```

```
Shape of train and test data: (32769, 9) (10, 9)
##################################################
Shape of frequency coded data: (32769, 9) (10, 9)
##################################################
Concatenated data shape: (32769, 18) (10, 18)
##################################################
Model fitted using training data
Predicted class for the provided input is  [1 1 1 1 1 1 1 1 1 1]
```

Function 2 : predicting the performance metric

As explained earlier, i am splitting the train data into train and test since the given test data doesn't contain any class labels

In [9]:

```
def pred_metric(X_train,X_test,y_train,y_test):
    #perform data preprocessing
    train_fc,test_fc = preprocessing(X_train,X_test)
    train_fc=pd.DataFrame(train_fc)
    test_fc=pd.DataFrame(test_fc)
    print('Shape of frequency coded data:',train_fc.shape,test_fc.shape)
    print('#'*50)


    #Concatenate data
    mod_train = pd.concat((X_train,train_fc),axis=1)
    mod_test = pd.concat((X_test,test_fc),axis=1)
    print('Concatenated data shape:',mod_train.shape,mod_test.shape)
    print('#'*50)


    from sklearn.ensemble import RandomForestClassifier
    model=RandomForestClassifier(n_estimators=1000,max_depth=25,max_features=5,
                             min_samples_split=2,
                             random_state=42,class_weight='balanced',n_jobs=-1)
    model.fit(mod_train,y_train)
    print('Model fitted using training data')
    #predicting the labels
    pred = model.predict_proba(mod_test)[:,1]
    return roc_auc_score(y_test,pred)
```

In [10]:

```
train=pd.read_csv('train.csv')
train_data=train.drop(columns=['ACTION'],axis=1)
y_true = train['ACTION']
X_train, X_test, y_train, y_test = train_test_split(train_data,y_true, test_size=0.25,stratify=y_true)
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
print('Performance metric: AUC :',pred_metric(X_train,X_test,y_train,y_test))
```

```
(24576, 9) (8193, 9) (24576,) (8193,)
Shape of frequency coded data: (24576, 9) (8193, 9)
##################################################
Concatenated data shape: (24576, 18) (8193, 18)
##################################################
Model fitted using training data
Performance metric: AUC : 0.8598218927158204
```