Loading Data

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_curve, auc, f1_score, accuracy_score
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import OneHotEncoder
from scipy.sparse import hstack

from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization, Activation, Flatten
from tensorflow.keras.layers import Input, Dense, Activation, Dropout, Embedding, concatenate
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
import bisect
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import pickle
```

In [ ]:

```
# loading train data
train = pd.read_csv('/content/drive/MyDrive/train.csv')
```

In [ ]:

```
Y = train['ACTION']
X = train[train.columns.difference(['ACTION'])]
```

In [ ]:

```
X.shape,Y.shape
```

Out[ ]:

```
((32769, 9), (32769,))
```

In [ ]:

```
# loading test data
test = pd.read_csv('/content/drive/MyDrive/test.csv')
```

In [ ]:

```
X_test=test.drop(columns=['id'],axis=1)
X_test.shape
```

Out[ ]:

```
(58921, 9)
```

Set-1

One_hot Encoding

RESOURCE

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['RESOURCE'].values.reshape(-1,1))

tr_RESOURCE = ohe.transform(train['RESOURCE'].values.reshape(-1,1))
te_RESOURCE = ohe.transform(test['RESOURCE'].values.reshape(-1,1))

print(tr_RESOURCE.shape)
print(te_RESOURCE.shape)
print(ohe.get_feature_names())
```

```
(32769, 7518)
(58921, 7518)
['x0_0' 'x0_38' 'x0_136' ... 'x0_312140' 'x0_312152' 'x0_312153']
```

MGR_ID

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['MGR_ID'].values.reshape(-1,1))

tr_MGR_ID = ohe.transform(train['MGR_ID'].values.reshape(-1,1))
te_MGR_ID = ohe.transform(test['MGR_ID'].values.reshape(-1,1))

print(tr_MGR_ID.shape)
print(te_MGR_ID.shape)
print(ohe.get_feature_names())
```

```
(32769, 4243)
(58921, 4243)
['x0_25' 'x0_27' 'x0_30' ... 'x0_311682' 'x0_311683' 'x0_311696']
```

ROLE_ROLLUP_1

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['ROLE_ROLLUP_1'].values.reshape(-1,1))

tr_ROLE_ROLLUP_1 = ohe.transform(train['ROLE_ROLLUP_1'].values.reshape(-1,1))
te_ROLE_ROLLUP_1 = ohe.transform(test['ROLE_ROLLUP_1'].values.reshape(-1,1))

print(tr_ROLE_ROLLUP_1.shape)
print(te_ROLE_ROLLUP_1.shape)
print(ohe.get_feature_names())
```

```
(32769, 128)
(58921, 128)
['x0_4292' 'x0_5110' 'x0_11146' 'x0_91261' 'x0_117876' 'x0_117882'
 'x0_117887' 'x0_117890' 'x0_117893' 'x0_117902' 'x0_117910' 'x0_117916'
 'x0_117918' 'x0_117922' 'x0_117926' 'x0_117929' 'x0_117932' 'x0_117935'
 'x0_117943' 'x0_117951' 'x0_117959' 'x0_117961' 'x0_117975' 'x0_117978'
 'x0_117980' 'x0_117983' 'x0_117989' 'x0_117993' 'x0_118000' 'x0_118003'
 'x0_118006' 'x0_118023' 'x0_118074' 'x0_118079' 'x0_118084' 'x0_118090'
 'x0_118095' 'x0_118106' 'x0_118114' 'x0_118120' 'x0_118126' 'x0_118138'
 'x0_118163' 'x0_118169' 'x0_118181' 'x0_118185' 'x0_118192' 'x0_118200'
 'x0_118212' 'x0_118216' 'x0_118219' 'x0_118256' 'x0_118269' 'x0_118290'
 'x0_118315' 'x0_118349' 'x0_118358' 'x0_118441' 'x0_118541' 'x0_118550'
 'x0_118555' 'x0_118573' 'x0_118582' 'x0_118595' 'x0_118602' 'x0_118658'
 'x0_118670' 'x0_118717' 'x0_118725' 'x0_118742' 'x0_118752' 'x0_118774'
 'x0_118887' 'x0_118953' 'x0_118976' 'x0_118990' 'x0_119027' 'x0_119062'
 'x0_119134' 'x0_119170' 'x0_119178' 'x0_119280' 'x0_119301' 'x0_119343'
 'x0_119370' 'x0_119402' 'x0_119596' 'x0_119615' 'x0_119665' 'x0_119691'
 'x0_119740' 'x0_119828' 'x0_119920' 'x0_120140' 'x0_120268' 'x0_120342'
 'x0_120354' 'x0_120810' 'x0_120864' 'x0_120883' 'x0_121005' 'x0_121411'
 'x0_121518' 'x0_121785' 'x0_122532' 'x0_122880' 'x0_124034' 'x0_125714'
```

```
 'x0_126918' 'x0_126974' 'x0_127044' 'x0_127616' 'x0_130570' 'x0_130684'
 'x0_131853' 'x0_132839' 'x0_133430' 'x0_138798' 'x0_141221' 'x0_143008'
 'x0_147236' 'x0_183723' 'x0_192441' 'x0_203209' 'x0_209434' 'x0_216705'
 'x0_247952' 'x0_311178']
```

ROLE_ROLLUP_2

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['ROLE_ROLLUP_2'].values.reshape(-1,1))

tr_ROLE_ROLLUP_2 = ohe.transform(train['ROLE_ROLLUP_2'].values.reshape(-1,1))
te_ROLE_ROLLUP_2 = ohe.transform(test['ROLE_ROLLUP_2'].values.reshape(-1,1))

print(tr_ROLE_ROLLUP_2.shape)
print(te_ROLE_ROLLUP_2.shape)
print(ohe.get_feature_names())
```

```
(32769, 177)
(58921, 177)
['x0_23779' 'x0_31010' 'x0_32137' 'x0_117877' 'x0_117883' 'x0_117891'
 'x0_117894' 'x0_117903' 'x0_117911' 'x0_117917' 'x0_117919' 'x0_117923'
 'x0_117927' 'x0_117930' 'x0_117933' 'x0_117936' 'x0_117940' 'x0_117944'
 'x0_117952' 'x0_117954' 'x0_117960' 'x0_117962' 'x0_117969' 'x0_117976'
 'x0_117979' 'x0_117981' 'x0_117984' 'x0_117990' 'x0_117994' 'x0_118001'
 'x0_118004' 'x0_118007' 'x0_118011' 'x0_118024' 'x0_118026' 'x0_118041'
 'x0_118052' 'x0_118076' 'x0_118080' 'x0_118085' 'x0_118091' 'x0_118096'
 'x0_118102' 'x0_118107' 'x0_118115' 'x0_118121' 'x0_118124' 'x0_118139'
 'x0_118150' 'x0_118164' 'x0_118170' 'x0_118178' 'x0_118182' 'x0_118193'
 'x0_118201' 'x0_118213' 'x0_118217' 'x0_118220' 'x0_118225' 'x0_118237'
 'x0_118257' 'x0_118266' 'x0_118270' 'x0_118291' 'x0_118300' 'x0_118316'
 'x0_118327' 'x0_118340' 'x0_118343' 'x0_118350' 'x0_118359' 'x0_118386'
 'x0_118413' 'x0_118442' 'x0_118446' 'x0_118463' 'x0_118491' 'x0_118542'
 'x0_118551' 'x0_118574' 'x0_118580' 'x0_118583' 'x0_118587' 'x0_118596'
 'x0_118603' 'x0_118659' 'x0_118671' 'x0_118718' 'x0_118726' 'x0_118743'
 'x0_118753' 'x0_118775' 'x0_118855' 'x0_118888' 'x0_118907' 'x0_118954'
 'x0_118977' 'x0_118991' 'x0_119028' 'x0_119063' 'x0_119070' 'x0_119075'
 'x0_119091' 'x0_119135' 'x0_119171' 'x0_119179' 'x0_119216' 'x0_119256'
 'x0_119281' 'x0_119302' 'x0_119344' 'x0_119370' 'x0_119403' 'x0_119428'
 'x0_119597' 'x0_119616' 'x0_119623' 'x0_119666' 'x0_119692' 'x0_119715'
 'x0_119741' 'x0_119762' 'x0_119763' 'x0_119829' 'x0_119836' 'x0_119883'
 'x0_119921' 'x0_120018' 'x0_120141' 'x0_120216' 'x0_120269' 'x0_120343'
 'x0_120355' 'x0_120811' 'x0_120846' 'x0_120862' 'x0_120865' 'x0_120884'
 'x0_121006' 'x0_121013' 'x0_121019' 'x0_121519' 'x0_121602' 'x0_121786'
 'x0_122533' 'x0_122974' 'x0_123330' 'x0_123999' 'x0_124035' 'x0_124157'
 'x0_124335' 'x0_125018' 'x0_125100' 'x0_125715' 'x0_126095' 'x0_126102'
 'x0_126919' 'x0_126975' 'x0_127045' 'x0_130600' 'x0_130685' 'x0_131390'
 'x0_131854' 'x0_132564' 'x0_132840' 'x0_138799' 'x0_140550' 'x0_141176'
 'x0_141222' 'x0_143009' 'x0_145248' 'x0_147237' 'x0_151110' 'x0_159716'
 'x0_176316' 'x0_185842' 'x0_286791']
```

ROLE_DEPTNAME

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['ROLE_DEPTNAME'].values.reshape(-1,1))

tr_ROLE_DEPTNAME = ohe.transform(train['ROLE_DEPTNAME'].values.reshape(-1,1))
te_ROLE_DEPTNAME = ohe.transform(test['ROLE_DEPTNAME'].values.reshape(-1,1))

print(tr_ROLE_DEPTNAME.shape)
print(te_ROLE_DEPTNAME.shape)
print(ohe.get_feature_names())
```

```
(32769, 449)
(58921, 449)
['x0_4674' 'x0_5488' 'x0_5606' 'x0_6104' 'x0_6725' 'x0_7646' 'x0_16232'
 'x0_19666' 'x0_19772' 'x0_20807' 'x0_28618' 'x0_29113' 'x0_81476'
 'x0_117878' 'x0_117884' 'x0_117895' 'x0_117904' 'x0_117912' 'x0_117920'
```

```
 'x0_117941' 'x0_117945' 'x0_117963' 'x0_117970' 'x0_118008' 'x0_118027'
 'x0_118035' 'x0_118042' 'x0_118053' 'x0_118063' 'x0_118066' 'x0_118128'
 'x0_118171' 'x0_118179' 'x0_118202' 'x0_118214' 'x0_118221' 'x0_118229'
 'x0_118246' 'x0_118292' 'x0_118301' 'x0_118317' 'x0_118320' 'x0_118328'
 'x0_118341' 'x0_118344' 'x0_118352' 'x0_118360' 'x0_118367' 'x0_118378'
 'x0_118387' 'x0_118391' 'x0_118395' 'x0_118403' 'x0_118404' 'x0_118409'
 'x0_118414' 'x0_118416' 'x0_118421' 'x0_118433' 'x0_118437' 'x0_118447'
 'x0_118450' 'x0_118458' 'x0_118464' 'x0_118471' 'x0_118481' 'x0_118483'
 'x0_118492' 'x0_118501' 'x0_118507' 'x0_118514' 'x0_118518' 'x0_118522'
 'x0_118529' 'x0_118535' 'x0_118543' 'x0_118546' 'x0_118552' 'x0_118556'
 'x0_118560' 'x0_118562' 'x0_118575' 'x0_118597' 'x0_118599' 'x0_118609'
 'x0_118616' 'x0_118623' 'x0_118631' 'x0_118635' 'x0_118660' 'x0_118673'
 'x0_118684' 'x0_118692' 'x0_118700' 'x0_118701' 'x0_118706' 'x0_118727'
 'x0_118733' 'x0_118744' 'x0_118746' 'x0_118754' 'x0_118783' 'x0_118791'
 'x0_118800' 'x0_118810' 'x0_118816' 'x0_118821' 'x0_118825' 'x0_118833'
 'x0_118840' 'x0_118846' 'x0_118856' 'x0_118862' 'x0_118867' 'x0_118881'
 'x0_118889' 'x0_118896' 'x0_118910' 'x0_118911' 'x0_118921' 'x0_118929'
 'x0_118933' 'x0_118940' 'x0_118957' 'x0_118963' 'x0_118970' 'x0_118979'
 'x0_118984' 'x0_118992' 'x0_119019' 'x0_119031' 'x0_119064' 'x0_119076'
 'x0_119092' 'x0_119107' 'x0_119121' 'x0_119136' 'x0_119142' 'x0_119181'
 'x0_119195' 'x0_119214' 'x0_119218' 'x0_119223' 'x0_119238' 'x0_119243'
 'x0_119257' 'x0_119262' 'x0_119279' 'x0_119303' 'x0_119362' 'x0_119386'
 'x0_119408' 'x0_119424' 'x0_119488' 'x0_119496' 'x0_119507' 'x0_119565'
 'x0_119569' 'x0_119598' 'x0_119703' 'x0_119734' 'x0_119742' 'x0_119781'
 'x0_119791' 'x0_119796' 'x0_119824' 'x0_119830' 'x0_119837' 'x0_119890'
 'x0_119898' 'x0_119922' 'x0_119924' 'x0_119945' 'x0_119954' 'x0_119961'
 'x0_119968' 'x0_119969' 'x0_119972' 'x0_119984' 'x0_119986' 'x0_119987'
 'x0_119993' 'x0_119995' 'x0_120016' 'x0_120026' 'x0_120041' 'x0_120050'
 'x0_120054' 'x0_120059' 'x0_120096' 'x0_120126' 'x0_120142' 'x0_120144'
 'x0_120171' 'x0_120201' 'x0_120211' 'x0_120270' 'x0_120283' 'x0_120291'
 'x0_120297' 'x0_120299' 'x0_120304' 'x0_120312' 'x0_120317' 'x0_120318'
 'x0_120323' 'x0_120347' 'x0_120356' 'x0_120361' 'x0_120368' 'x0_120370'
 'x0_120383' 'x0_120398' 'x0_120410' 'x0_120417' 'x0_120428' 'x0_120526'
 'x0_120535' 'x0_120539' 'x0_120551' 'x0_120559' 'x0_120574' 'x0_120584'
 'x0_120620' 'x0_120624' 'x0_120663' 'x0_120666' 'x0_120671' 'x0_120677'
 'x0_120685' 'x0_120694' 'x0_120709' 'x0_120722' 'x0_120764' 'x0_120823'
 'x0_120924' 'x0_120943' 'x0_120995' 'x0_121014' 'x0_121023' 'x0_121030'
 'x0_121097' 'x0_121108' 'x0_121169' 'x0_121176' 'x0_121216' 'x0_121220'
 'x0_121305' 'x0_121363' 'x0_121405' 'x0_121458' 'x0_121533' 'x0_121574'
 'x0_121589' 'x0_121617' 'x0_121639' 'x0_121645' 'x0_121667' 'x0_121668'
 'x0_121678' 'x0_121694' 'x0_121710' 'x0_121716' 'x0_121747' 'x0_121787'
 'x0_121820' 'x0_121883' 'x0_121949' 'x0_121951' 'x0_121961' 'x0_121977'
 'x0_121979' 'x0_122001' 'x0_122007' 'x0_122012' 'x0_122059' 'x0_122070'
 'x0_122109' 'x0_122215' 'x0_122224' 'x0_122273' 'x0_122298' 'x0_122299'
 'x0_122358' 'x0_122392' 'x0_122453' 'x0_122550' 'x0_122587' 'x0_122636'
 'x0_122672' 'x0_122722' 'x0_122870' 'x0_122938' 'x0_122963' 'x0_123003'
 'x0_123007' 'x0_123055' 'x0_123072' 'x0_123089' 'x0_123125' 'x0_123144'
 'x0_123173' 'x0_123175' 'x0_123195' 'x0_123201' 'x0_123279' 'x0_123454'
 'x0_123472' 'x0_123476' 'x0_123494' 'x0_123519' 'x0_123606' 'x0_123614'
 'x0_123631' 'x0_123656' 'x0_123675' 'x0_123719' 'x0_123749' 'x0_123757'
 'x0_123766' 'x0_123844' 'x0_123858' 'x0_123901' 'x0_124051' 'x0_124130'
 'x0_124133' 'x0_124170' 'x0_124211' 'x0_124266' 'x0_124380' 'x0_124449'
 'x0_124656' 'x0_124668' 'x0_124725' 'x0_124816' 'x0_124921' 'x0_124942'
 'x0_124948' 'x0_125004' 'x0_125016' 'x0_125101' 'x0_125133' 'x0_125139'
 'x0_125144' 'x0_125178' 'x0_125316' 'x0_125440' 'x0_125821' 'x0_125857'
 'x0_125872' 'x0_125884' 'x0_125919' 'x0_126137' 'x0_126229' 'x0_126310'
 'x0_126352' 'x0_126574' 'x0_126745' 'x0_126785' 'x0_126930' 'x0_126955'
 'x0_127155' 'x0_127168' 'x0_127284' 'x0_127470' 'x0_127491' 'x0_127522'
 'x0_127705' 'x0_127812' 'x0_127849' 'x0_128113' 'x0_128350' 'x0_128516'
 'x0_128639' 'x0_128742' 'x0_128801' 'x0_128823' 'x0_128830' 'x0_128935'
 'x0_129120' 'x0_129128' 'x0_129526' 'x0_129578' 'x0_129617' 'x0_129972'
 'x0_130192' 'x0_130859' 'x0_131067' 'x0_131159' 'x0_131274' 'x0_131303'
 'x0_131461' 'x0_131868' 'x0_132427' 'x0_132480' 'x0_132530' 'x0_132647'
 'x0_134257' 'x0_134848' 'x0_135245' 'x0_137107' 'x0_137996' 'x0_138789'
 'x0_139001' 'x0_139677' 'x0_139759' 'x0_139876' 'x0_139897' 'x0_140453'
 'x0_141383' 'x0_142038' 'x0_142145' 'x0_142493' 'x0_142540' 'x0_143531'
 'x0_145424' 'x0_145774' 'x0_146387' 'x0_147019' 'x0_147589' 'x0_148436'
 'x0_148450' 'x0_149210' 'x0_149666' 'x0_151108' 'x0_164199' 'x0_168533'
 'x0_169899' 'x0_171098' 'x0_176153' 'x0_179069' 'x0_181065' 'x0_184402'
 'x0_185576' 'x0_186536' 'x0_189629' 'x0_196823' 'x0_204054' 'x0_215920'
 'x0_223958' 'x0_225010' 'x0_240766' 'x0_253965' 'x0_255696' 'x0_272283'
 'x0_274241' 'x0_275600' 'x0_277693' 'x0_286792']
```

ROLE_TITLE

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['ROLE_TITLE'].values.reshape(-1,1))

tr_ROLE_TITLE = ohe.transform(train['ROLE_TITLE'].values.reshape(-1,1))
te_ROLE_TITLE = ohe.transform(test['ROLE_TITLE'].values.reshape(-1,1))

print(tr_ROLE_TITLE.shape)
print(te_ROLE_TITLE.shape)
print(ohe.get_feature_names())
```

```
(32769, 343)
(58921, 343)
['x0_117879' 'x0_117885' 'x0_117896' 'x0_117899' 'x0_117905' 'x0_117906'
 'x0_117946' 'x0_117985' 'x0_118028' 'x0_118043' 'x0_118047' 'x0_118054'
 'x0_118129' 'x0_118172' 'x0_118194' 'x0_118203' 'x0_118207' 'x0_118259'
 'x0_118274' 'x0_118278' 'x0_118293' 'x0_118318' 'x0_118321' 'x0_118361'
 'x0_118368' 'x0_118370' 'x0_118396' 'x0_118422' 'x0_118451' 'x0_118459'
 'x0_118465' 'x0_118502' 'x0_118523' 'x0_118530' 'x0_118536' 'x0_118563'
 'x0_118568' 'x0_118636' 'x0_118641' 'x0_118674' 'x0_118685' 'x0_118702'
 'x0_118728' 'x0_118734' 'x0_118747' 'x0_118760' 'x0_118777' 'x0_118784'
 'x0_118792' 'x0_118801' 'x0_118805' 'x0_118811' 'x0_118826' 'x0_118834'
 'x0_118841' 'x0_118863' 'x0_118890' 'x0_118912' 'x0_118924' 'x0_118958'
 'x0_118980' 'x0_118995' 'x0_119004' 'x0_119065' 'x0_119077' 'x0_119093'
 'x0_119137' 'x0_119172' 'x0_119192' 'x0_119219' 'x0_119323' 'x0_119346'
 'x0_119351' 'x0_119363' 'x0_119409' 'x0_119433' 'x0_119502' 'x0_119529'
 'x0_119587' 'x0_119743' 'x0_119778' 'x0_119782' 'x0_119786' 'x0_119849'
 'x0_119885' 'x0_119899' 'x0_119928' 'x0_119949' 'x0_119962' 'x0_119976'
 'x0_119997' 'x0_120001' 'x0_120006' 'x0_120033' 'x0_120056' 'x0_120069'
 'x0_120097' 'x0_120115' 'x0_120132' 'x0_120172' 'x0_120284' 'x0_120300'
 'x0_120313' 'x0_120344' 'x0_120348' 'x0_120357' 'x0_120418' 'x0_120497'
 'x0_120516' 'x0_120527' 'x0_120560' 'x0_120575' 'x0_120578' 'x0_120591'
 'x0_120611' 'x0_120618' 'x0_120621' 'x0_120628' 'x0_120632' 'x0_120647'
 'x0_120690' 'x0_120702' 'x0_120765' 'x0_120773' 'x0_120789' 'x0_120812'
 'x0_120903' 'x0_120952' 'x0_120988' 'x0_120990' 'x0_121015' 'x0_121067'
 'x0_121122' 'x0_121143' 'x0_121246' 'x0_121364' 'x0_121372' 'x0_121414'
 'x0_121469' 'x0_121527' 'x0_121594' 'x0_121618' 'x0_121915' 'x0_122022'
 'x0_122030' 'x0_122060' 'x0_122067' 'x0_122129' 'x0_122142' 'x0_122188'
 'x0_122269' 'x0_122274' 'x0_122290' 'x0_122297' 'x0_122345' 'x0_122551'
 'x0_122645' 'x0_122849' 'x0_122860' 'x0_122927' 'x0_122952' 'x0_122967'
 'x0_122989' 'x0_123045' 'x0_123067' 'x0_123073' 'x0_123082' 'x0_123131'
 'x0_123178' 'x0_123191' 'x0_123400' 'x0_123408' 'x0_123609' 'x0_123615'
 'x0_123648' 'x0_123651' 'x0_123670' 'x0_123684' 'x0_123737' 'x0_123850'
 'x0_124000' 'x0_124134' 'x0_124144' 'x0_124152' 'x0_124194' 'x0_124246'
 'x0_124305' 'x0_124313' 'x0_124419' 'x0_124435' 'x0_124486' 'x0_124537'
 'x0_124576' 'x0_124775' 'x0_124799' 'x0_124810' 'x0_124886' 'x0_124922'
 'x0_125010' 'x0_125171' 'x0_125405' 'x0_125687' 'x0_125751' 'x0_125793'
 'x0_125798' 'x0_126078' 'x0_126085' 'x0_126110' 'x0_126138' 'x0_126184'
 'x0_126264' 'x0_126293' 'x0_126418' 'x0_126502' 'x0_126516' 'x0_126538'
 'x0_126547' 'x0_126684' 'x0_126746' 'x0_126820' 'x0_126869' 'x0_126931'
 'x0_127031' 'x0_127108' 'x0_127389' 'x0_127589' 'x0_127657' 'x0_127700'
 'x0_127723' 'x0_127782' 'x0_127847' 'x0_127850' 'x0_127955' 'x0_128093'
 'x0_128197' 'x0_128230' 'x0_128351' 'x0_128422' 'x0_128764' 'x0_128903'
 'x0_129229' 'x0_129561' 'x0_129909' 'x0_130060' 'x0_130284' 'x0_130362'
 'x0_130479' 'x0_130528' 'x0_130606' 'x0_130633' 'x0_130637' 'x0_130857'
 'x0_131252' 'x0_131336' 'x0_131795' 'x0_131849' 'x0_131997' 'x0_132096'
 'x0_132103' 'x0_132583' 'x0_132671' 'x0_132692' 'x0_132723' 'x0_132737'
 'x0_133111' 'x0_133306' 'x0_133646' 'x0_133718' 'x0_134067' 'x0_134095'
 'x0_134118' 'x0_134655' 'x0_135123' 'x0_135740' 'x0_135809' 'x0_136115'
 'x0_136701' 'x0_137370' 'x0_137969' 'x0_138019' 'x0_138137' 'x0_139965'
 'x0_140847' 'x0_143183' 'x0_144353' 'x0_145648' 'x0_146249' 'x0_146951'
 'x0_147122' 'x0_149228' 'x0_149337' 'x0_149351' 'x0_149916' 'x0_150074'
 'x0_150752' 'x0_152268' 'x0_152308' 'x0_153248' 'x0_153893' 'x0_153957'
 'x0_155110' 'x0_157300' 'x0_157347' 'x0_157359' 'x0_157799' 'x0_158289'
 'x0_159116' 'x0_159677' 'x0_159787' 'x0_161098' 'x0_162860' 'x0_166592'
 'x0_166800' 'x0_169634' 'x0_174391' 'x0_179731' 'x0_180927' 'x0_184274'
 'x0_187168' 'x0_188046' 'x0_192867' 'x0_208126' 'x0_208565' 'x0_209874'
 'x0_212192' 'x0_216825' 'x0_235351' 'x0_239003' 'x0_240103' 'x0_247659'
 'x0_258434' 'x0_259173' 'x0_266862' 'x0_268608' 'x0_270690' 'x0_273308'
 'x0_279482' 'x0_280788' 'x0_297560' 'x0_299559' 'x0_307024' 'x0_310825'
 'x0_311867']
```

ROLE_FAMILY_DESC

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['ROLE_FAMILY_DESC'].values.reshape(-1,1))

tr_ROLE_FAMILY_DESC = ohe.transform(train['ROLE_FAMILY_DESC'].values.reshape(-1,1))
te_ROLE_FAMILY_DESC = ohe.transform(test['ROLE_FAMILY_DESC'].values.reshape(-1,1))

print(tr_ROLE_FAMILY_DESC.shape)
print(te_ROLE_FAMILY_DESC.shape)
print(ohe.get_feature_names())
```

```
(32769, 2358)
(58921, 2358)
['x0_4673' 'x0_62587' 'x0_117879' ... 'x0_311834' 'x0_311839' 'x0_311867']
```

ROLE_FAMILY

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['ROLE_FAMILY'].values.reshape(-1,1))

tr_ROLE_FAMILY = ohe.transform(train['ROLE_FAMILY'].values.reshape(-1,1))
te_ROLE_FAMILY = ohe.transform(test['ROLE_FAMILY'].values.reshape(-1,1))

print(tr_ROLE_FAMILY.shape)
print(te_ROLE_FAMILY.shape)
print(ohe.get_feature_names())
```

```
(32769, 67)
(58921, 67)
['x0_3130' 'x0_4673' 'x0_6725' 'x0_19721' 'x0_19793' 'x0_117887'
 'x0_118131' 'x0_118205' 'x0_118295' 'x0_118331' 'x0_118347' 'x0_118363'
 'x0_118372' 'x0_118398' 'x0_118424' 'x0_118453' 'x0_118467' 'x0_118474'
 'x0_118478' 'x0_118504' 'x0_118612' 'x0_118638' 'x0_118643' 'x0_118667'
 'x0_118704' 'x0_118736' 'x0_118762' 'x0_118870' 'x0_118960' 'x0_119006'
 'x0_119095' 'x0_119184' 'x0_119221' 'x0_119695' 'x0_119772' 'x0_119784'
 'x0_119788' 'x0_120134' 'x0_120302' 'x0_120518' 'x0_121069' 'x0_121620'
 'x0_121916' 'x0_122032' 'x0_123611' 'x0_123689' 'x0_124136' 'x0_124145'
 'x0_124487' 'x0_125407' 'x0_127957' 'x0_130364' 'x0_131999' 'x0_132725'
 'x0_136398' 'x0_143398' 'x0_149353' 'x0_151277' 'x0_155173' 'x0_159679'
 'x0_161100' 'x0_249618' 'x0_254395' 'x0_270488' 'x0_290919' 'x0_292795'
 'x0_308574']
```

ROLE_CODE

In [ ]:

```
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(train['ROLE_CODE'].values.reshape(-1,1))

tr_ROLE_CODE = ohe.transform(train['ROLE_CODE'].values.reshape(-1,1))
te_ROLE_CODE = ohe.transform(test['ROLE_CODE'].values.reshape(-1,1))

print(tr_ROLE_CODE.shape)
print(te_ROLE_CODE.shape)
print(ohe.get_feature_names())
```

```
(32769, 343)
(58921, 343)
['x0_117880' 'x0_117888' 'x0_117898' 'x0_117900' 'x0_117908' 'x0_117948'
 'x0_117973' 'x0_117987' 'x0_118030' 'x0_118046' 'x0_118049' 'x0_118055'
 'x0_118132' 'x0_118175' 'x0_118196' 'x0_118206' 'x0_118209' 'x0_118232'
 'x0_118261' 'x0_118276' 'x0_118279' 'x0_118296' 'x0_118319' 'x0_118322'
 'x0_118332' 'x0_118364' 'x0_118373' 'x0_118399' 'x0_118425' 'x0_118454'
 'x0_118461' 'x0_118468' 'x0_118475' 'x0_118479' 'x0_118486' 'x0_118505'
 'x0_118525' 'x0_118532' 'x0_118539' 'x0_118565' 'x0_118570' 'x0_118639'
 'x0_118644' 'x0_118676' 'x0_118687' 'x0_118705' 'x0_118730' 'x0_118737'
```

```
 'x0_118749' 'x0_118763' 'x0_118779' 'x0_118786' 'x0_118794' 'x0_118803'
 'x0_118807' 'x0_118813' 'x0_118828' 'x0_118836' 'x0_118843' 'x0_118865'
 'x0_118892' 'x0_118899' 'x0_118914' 'x0_118926' 'x0_118943' 'x0_118961'
 'x0_118982' 'x0_118997' 'x0_119007' 'x0_119067' 'x0_119079' 'x0_119082'
 'x0_119096' 'x0_119139' 'x0_119174' 'x0_119194' 'x0_119222' 'x0_119325'
 'x0_119348' 'x0_119353' 'x0_119365' 'x0_119411' 'x0_119435' 'x0_119503'
 'x0_119531' 'x0_119589' 'x0_119745' 'x0_119779' 'x0_119785' 'x0_119789'
 'x0_119817' 'x0_119851' 'x0_119887' 'x0_119900' 'x0_119929' 'x0_119951'
 'x0_119964' 'x0_119978' 'x0_119998' 'x0_120003' 'x0_120008' 'x0_120035'
 'x0_120058' 'x0_120071' 'x0_120099' 'x0_120117' 'x0_120135' 'x0_120173'
 'x0_120285' 'x0_120303' 'x0_120315' 'x0_120346' 'x0_120350' 'x0_120359'
 'x0_120364' 'x0_120419' 'x0_120499' 'x0_120519' 'x0_120529' 'x0_120562'
 'x0_120577' 'x0_120580' 'x0_120593' 'x0_120613' 'x0_120619' 'x0_120623'
 'x0_120629' 'x0_120634' 'x0_120649' 'x0_120692' 'x0_120704' 'x0_120767'
 'x0_120774' 'x0_120791' 'x0_120814' 'x0_120904' 'x0_120954' 'x0_120989'
 'x0_120992' 'x0_121017' 'x0_121070' 'x0_121124' 'x0_121145' 'x0_121248'
 'x0_121366' 'x0_121374' 'x0_121395' 'x0_121416' 'x0_121471' 'x0_121529'
 'x0_121596' 'x0_121621' 'x0_121917' 'x0_122024' 'x0_122033' 'x0_122062'
 'x0_122069' 'x0_122131' 'x0_122143' 'x0_122190' 'x0_122271' 'x0_122275'
 'x0_122292' 'x0_122346' 'x0_122552' 'x0_122647' 'x0_122850' 'x0_122862'
 'x0_122929' 'x0_122954' 'x0_122969' 'x0_122991' 'x0_123047' 'x0_123068'
 'x0_123075' 'x0_123084' 'x0_123133' 'x0_123180' 'x0_123192' 'x0_123402'
 'x0_123410' 'x0_123612' 'x0_123617' 'x0_123650' 'x0_123652' 'x0_123657'
 'x0_123672' 'x0_123686' 'x0_123738' 'x0_123851' 'x0_124002' 'x0_124137'
 'x0_124146' 'x0_124154' 'x0_124196' 'x0_124247' 'x0_124307' 'x0_124315'
 'x0_124421' 'x0_124436' 'x0_124488' 'x0_124539' 'x0_124578' 'x0_124777'
 'x0_124801' 'x0_124812' 'x0_124888' 'x0_124924' 'x0_125012' 'x0_125173'
 'x0_125408' 'x0_125689' 'x0_125753' 'x0_125795' 'x0_125800' 'x0_126080'
 'x0_126087' 'x0_126112' 'x0_126140' 'x0_126186' 'x0_126266' 'x0_126295'
 'x0_126420' 'x0_126504' 'x0_126518' 'x0_126540' 'x0_126549' 'x0_126685'
 'x0_126748' 'x0_126822' 'x0_126870' 'x0_126933' 'x0_127032' 'x0_127110'
 'x0_127391' 'x0_127590' 'x0_127659' 'x0_127702' 'x0_127725' 'x0_127783'
 'x0_127848' 'x0_127851' 'x0_127958' 'x0_128095' 'x0_128199' 'x0_128231'
 'x0_128353' 'x0_128424' 'x0_128765' 'x0_128905' 'x0_129231' 'x0_129563'
 'x0_129911' 'x0_130062' 'x0_130285' 'x0_130365' 'x0_130481' 'x0_130607'
 'x0_130635' 'x0_130638' 'x0_130858' 'x0_131254' 'x0_131338' 'x0_131797'
 'x0_131851' 'x0_132000' 'x0_132098' 'x0_132105' 'x0_132585' 'x0_132673'
 'x0_132694' 'x0_132726' 'x0_132739' 'x0_133113' 'x0_133308' 'x0_133648'
 'x0_133719' 'x0_134069' 'x0_134120' 'x0_134657' 'x0_135125' 'x0_135742'
 'x0_135811' 'x0_136061' 'x0_136117' 'x0_136702' 'x0_137371' 'x0_137970'
 'x0_138021' 'x0_138139' 'x0_139967' 'x0_140849' 'x0_143185' 'x0_144355'
 'x0_146251' 'x0_146952' 'x0_147124' 'x0_149230' 'x0_149339' 'x0_149354'
 'x0_149918' 'x0_150076' 'x0_150754' 'x0_152270' 'x0_152310' 'x0_153249'
 'x0_153895' 'x0_153959' 'x0_155111' 'x0_157301' 'x0_157348' 'x0_157361'
 'x0_157801' 'x0_158291' 'x0_159118' 'x0_159680' 'x0_159789' 'x0_161101'
 'x0_162862' 'x0_163313' 'x0_163732' 'x0_166594' 'x0_166801' 'x0_169635'
 'x0_174393' 'x0_180928' 'x0_184276' 'x0_187169' 'x0_188048' 'x0_192869'
 'x0_208127' 'x0_208567' 'x0_209875' 'x0_212194' 'x0_216827' 'x0_239004'
 'x0_240105' 'x0_247660' 'x0_254396' 'x0_258436' 'x0_266863' 'x0_268610'
 'x0_270691']
```

Combining all encoded features

In [ ]:

```
x_train_ohe = hstack((tr_RESOURCE,tr_MGR_ID,tr_ROLE_CODE,tr_ROLE_DEPTNAME,tr_ROLE_FAMILY,tr_ROLE_FAMILY_DESC,tr_ROLE_ROLLUP_1,tr_ROLE_ROLLUP_2,tr_ROLE_TITLE))
x_test_ohe = hstack((te_RESOURCE,te_MGR_ID,te_ROLE_CODE,te_ROLE_DEPTNAME,te_ROLE_FAMILY,te_ROLE_FAMILY_DESC,te_ROLE_ROLLUP_1,te_ROLE_ROLLUP_2,te_ROLE_TITLE))
```

In [ ]:

```
x_train_ohe.shape
```

Out[ ]:

```
(32769, 15626)
```

In [ ]:

```
x_test_ohe.shape
```

```
x_test_one.shape
```

Out[ ]:

```
(58921, 15626)
```

In [ ]:

```
# storing in a pickle file
f = open('1_hot_enc.pckl','wb')
pickle.dump([x_train_ohe,x_test_ohe],f)
f.close()
```

In [ ]:

```
# loading pickle file
f = open('/content/drive/MyDrive/1_hot_enc.pckl','rb')
x_train_ohe,x_test_ohe = pickle.load(f)
f.close()
```

Set-2

Using Learned Embedding

A learned embedding, or simply an “embedding,” is a distributed representation for categorical data.

Each category is mapped to a distinct vector, and the properties of the vector are adapted or learned while training a neural network.

Unlike one hot encoding, the input vectors are not sparse (do not have lots of zeros).

https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/

In [ ]:

```
from tensorflow.keras.optimizers import *
```

In [ ]:

```
# the embedding expects the categories to be ordinal encoded, so label_encoding all features.

X_train_enc, X_test_enc = list(), list()

# label encode each column
for i in X.columns.values.tolist():
    le = LabelEncoder()
    le.fit(X[i])

    #https://stackoverflow.com/q/21057621/13401359                                    # for categorie:
in test_data that are not present in train_data
    X_test[i] = X_test[i].map(lambda s:0 if s not in le.classes_ else s)            # encoding all th
ose categories to 0.

    le_classes = le.classes_.tolist()
    bisect.insort_left(le_classes, 0)                                                # inserting '0'
class in labelencoder_classes
    le.classes_ = le_classes

    train_enc = le.transform(X[i])
    test_enc = le.transform(X_test[i])

    X_train_enc.append(train_enc)
    X_test_enc.append(test_enc)
```

In [ ]:

```
# label_encoding train class_label

le = LabelEncoder()
le.fit(Y)
```

```
le.fit(Y)
y_train_enc = le.transform(Y)
```

In [ ]:

```
# creating embedding layer for each column

in_layers = list()
em_layers = list()

for i in range(len(X_train_enc)):
 # calculate the number of unique inputs
 n_labels = len(np.unique(X_train_enc[i]))

 # define input layer
 in_layer = Input(shape=(1,))

 # define embedding layer
 em_layer = Embedding(n_labels+1, 100)(in_layer)

 # store layers
 in_layers.append(in_layer)
 em_layers.append(em_layer)

# concat all embeddings
merge = concatenate(em_layers)
```

In [ ]:

```
tf.keras.backend.clear_session()
```

In [ ]:

```
model = None
```

In [ ]:

```
# creating a NN model

e1 = Dense(units=128,activation='relu')(merge)
d1 = Dropout(0.4)(e1)
e2 = Dense(units=32,activation='relu')(d1)
d2 = Dropout(0.2)(e2)
e3 = Dense(units=16,activation='relu')(d2)
#b1 = BatchNormalization()(e2)
out = Dense(1, activation='sigmoid')(e3)
model = Model(inputs=in_layers, outputs=out)
```

In [ ]:

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_8 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
```

```
 input_9 (InputLayer)           [(None, 1)]          0

 embedding (Embedding)          (None, 1, 100)       424400      input_1[0][0]

 embedding_1 (Embedding)        (None, 1, 100)       751900      input_2[0][0]

 embedding_2 (Embedding)        (None, 1, 100)       34400       input_3[0][0]

 embedding_3 (Embedding)        (None, 1, 100)       45000       input_4[0][0]

 embedding_4 (Embedding)        (None, 1, 100)       6800        input_5[0][0]

 embedding_5 (Embedding)        (None, 1, 100)       235900      input_6[0][0]

 embedding_6 (Embedding)        (None, 1, 100)       12900       input_7[0][0]

 embedding_7 (Embedding)        (None, 1, 100)       17800       input_8[0][0]

 embedding_8 (Embedding)        (None, 1, 100)       34400       input_9[0][0]

 concatenate (Concatenate)      (None, 1, 900)       0           embedding[0][0]
                                                                 embedding_1[0][0]
                                                                 embedding_2[0][0]
                                                                 embedding_3[0][0]
                                                                 embedding_4[0][0]
                                                                 embedding_5[0][0]
                                                                 embedding_6[0][0]
                                                                 embedding_7[0][0]
                                                                 embedding_8[0][0]

 dense (Dense)                  (None, 1, 64)        57664       concatenate[0][0]

 dropout (Dropout)              (None, 1, 64)        0           dense[0][0]

 dense_1 (Dense)                (None, 1, 32)        2080        dropout[0][0]

 dropout_1 (Dropout)            (None, 1, 32)        0           dense_1[0][0]

 dense_2 (Dense)                (None, 1, 16)        528         dropout_1[0][0]

 dense_3 (Dense)                (None, 1, 1)         17          dense_2[0][0]
==================================================================================================
Total params: 1,623,789
Trainable params: 1,623,789
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [ ]:

```python
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,patience=1,verbose=1)
earlystop = EarlyStopping(monitor='val_loss', patience=2, verbose=1)
```

In [ ]:

```python
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.00001), metrics=tf.keras.metrics.AUC
())
```

In [ ]:

```python
model.fit(X_train_enc, y_train_enc, epochs=100, batch_size=1024, verbose=2,validation_split=0.25,ca
llbacks=[earlystop])
```

```
Epoch 1/100
24/24 - 2s - loss: 0.6744 - auc: 0.4617 - val_loss: 0.6711 - val_auc: 0.5215
Epoch 2/100
24/24 - 1s - loss: 0.6637 - auc: 0.5267 - val_loss: 0.6597 - val_auc: 0.5808
Epoch 3/100
24/24 - 1s - loss: 0.6512 - auc: 0.5896 - val_loss: 0.6453 - val_auc: 0.6437
Epoch 4/100
24/24 - 1s - loss: 0.6352 - auc: 0.6735 - val_loss: 0.6266 - val_auc: 0.7043
```

```
Epoch 5/100
24/24 - 1s - loss: 0.6150 - auc: 0.7324 - val_loss: 0.6043 - val_auc: 0.7528
Epoch 6/100
24/24 - 1s - loss: 0.5923 - auc: 0.7747 - val_loss: 0.5811 - val_auc: 0.7852
Epoch 7/100
24/24 - 1s - loss: 0.5688 - auc: 0.8080 - val_loss: 0.5580 - val_auc: 0.8010
Epoch 8/100
24/24 - 1s - loss: 0.5451 - auc: 0.8316 - val_loss: 0.5355 - val_auc: 0.8119
Epoch 9/100
24/24 - 1s - loss: 0.5223 - auc: 0.8397 - val_loss: 0.5134 - val_auc: 0.8192
Epoch 10/100
24/24 - 1s - loss: 0.4999 - auc: 0.8529 - val_loss: 0.4918 - val_auc: 0.8241
Epoch 11/100
24/24 - 1s - loss: 0.4786 - auc: 0.8614 - val_loss: 0.4708 - val_auc: 0.8291
Epoch 12/100
24/24 - 1s - loss: 0.4580 - auc: 0.8682 - val_loss: 0.4506 - val_auc: 0.8321
Epoch 13/100
24/24 - 1s - loss: 0.4383 - auc: 0.8715 - val_loss: 0.4313 - val_auc: 0.8357
Epoch 14/100
24/24 - 1s - loss: 0.4187 - auc: 0.8766 - val_loss: 0.4131 - val_auc: 0.8380
Epoch 15/100
24/24 - 1s - loss: 0.4019 - auc: 0.8830 - val_loss: 0.3960 - val_auc: 0.8391
Epoch 16/100
24/24 - 1s - loss: 0.3849 - auc: 0.8860 - val_loss: 0.3800 - val_auc: 0.8411
Epoch 17/100
24/24 - 1s - loss: 0.3684 - auc: 0.8938 - val_loss: 0.3650 - val_auc: 0.8423
Epoch 18/100
24/24 - 1s - loss: 0.3551 - auc: 0.8942 - val_loss: 0.3509 - val_auc: 0.8432
Epoch 19/100
24/24 - 1s - loss: 0.3409 - auc: 0.8993 - val_loss: 0.3378 - val_auc: 0.8444
Epoch 20/100
24/24 - 1s - loss: 0.3279 - auc: 0.9004 - val_loss: 0.3255 - val_auc: 0.8455
Epoch 21/100
24/24 - 1s - loss: 0.3158 - auc: 0.9051 - val_loss: 0.3139 - val_auc: 0.8454
Epoch 22/100
24/24 - 1s - loss: 0.3051 - auc: 0.9059 - val_loss: 0.3032 - val_auc: 0.8468
Epoch 23/100
24/24 - 1s - loss: 0.2921 - auc: 0.9113 - val_loss: 0.2931 - val_auc: 0.8475
Epoch 24/100
24/24 - 1s - loss: 0.2833 - auc: 0.9097 - val_loss: 0.2836 - val_auc: 0.8481
Epoch 25/100
24/24 - 1s - loss: 0.2741 - auc: 0.9147 - val_loss: 0.2748 - val_auc: 0.8483
Epoch 26/100
24/24 - 1s - loss: 0.2643 - auc: 0.9176 - val_loss: 0.2666 - val_auc: 0.8488
Epoch 27/100
24/24 - 1s - loss: 0.2554 - auc: 0.9221 - val_loss: 0.2590 - val_auc: 0.8492
Epoch 28/100
24/24 - 1s - loss: 0.2473 - auc: 0.9225 - val_loss: 0.2519 - val_auc: 0.8494
Epoch 29/100
24/24 - 1s - loss: 0.2402 - auc: 0.9239 - val_loss: 0.2453 - val_auc: 0.8503
Epoch 30/100
24/24 - 1s - loss: 0.2332 - auc: 0.9259 - val_loss: 0.2392 - val_auc: 0.8507
Epoch 31/100
24/24 - 1s - loss: 0.2260 - auc: 0.9303 - val_loss: 0.2335 - val_auc: 0.8510
Epoch 32/100
24/24 - 1s - loss: 0.2201 - auc: 0.9329 - val_loss: 0.2283 - val_auc: 0.8511
Epoch 33/100
24/24 - 1s - loss: 0.2130 - auc: 0.9362 - val_loss: 0.2233 - val_auc: 0.8511
Epoch 34/100
24/24 - 1s - loss: 0.2085 - auc: 0.9369 - val_loss: 0.2188 - val_auc: 0.8513
Epoch 35/100
24/24 - 1s - loss: 0.2020 - auc: 0.9406 - val_loss: 0.2146 - val_auc: 0.8516
Epoch 36/100
24/24 - 1s - loss: 0.1978 - auc: 0.9369 - val_loss: 0.2107 - val_auc: 0.8519
Epoch 37/100
24/24 - 1s - loss: 0.1920 - auc: 0.9437 - val_loss: 0.2070 - val_auc: 0.8520
Epoch 38/100
24/24 - 1s - loss: 0.1889 - auc: 0.9413 - val_loss: 0.2037 - val_auc: 0.8526
Epoch 39/100
24/24 - 1s - loss: 0.1842 - auc: 0.9443 - val_loss: 0.2006 - val_auc: 0.8528
Epoch 40/100
24/24 - 1s - loss: 0.1797 - auc: 0.9462 - val_loss: 0.1977 - val_auc: 0.8534
Epoch 41/100
24/24 - 1s - loss: 0.1753 - auc: 0.9492 - val_loss: 0.1950 - val_auc: 0.8532
Epoch 42/100
24/24 - 1s - loss: 0.1718 - auc: 0.9497 - val_loss: 0.1926 - val_auc: 0.8537
Epoch 43/100
```

```
24/24 - 1s - loss: 0.1690 - auc: 0.9515 - val_loss: 0.1904 - val_auc: 0.8539
Epoch 44/100
24/24 - 1s - loss: 0.1660 - auc: 0.9507 - val_loss: 0.1883 - val_auc: 0.8537
Epoch 45/100
24/24 - 1s - loss: 0.1616 - auc: 0.9547 - val_loss: 0.1864 - val_auc: 0.8543
Epoch 46/100
24/24 - 1s - loss: 0.1589 - auc: 0.9547 - val_loss: 0.1847 - val_auc: 0.8545
Epoch 47/100
24/24 - 1s - loss: 0.1571 - auc: 0.9548 - val_loss: 0.1830 - val_auc: 0.8547
Epoch 48/100
24/24 - 1s - loss: 0.1522 - auc: 0.9608 - val_loss: 0.1815 - val_auc: 0.8543
Epoch 49/100
24/24 - 1s - loss: 0.1498 - auc: 0.9616 - val_loss: 0.1801 - val_auc: 0.8548
Epoch 50/100
24/24 - 1s - loss: 0.1484 - auc: 0.9592 - val_loss: 0.1788 - val_auc: 0.8552
Epoch 51/100
24/24 - 1s - loss: 0.1456 - auc: 0.9614 - val_loss: 0.1776 - val_auc: 0.8556
Epoch 52/100
24/24 - 1s - loss: 0.1425 - auc: 0.9651 - val_loss: 0.1765 - val_auc: 0.8547
Epoch 53/100
24/24 - 1s - loss: 0.1398 - auc: 0.9670 - val_loss: 0.1755 - val_auc: 0.8558
Epoch 54/100
24/24 - 1s - loss: 0.1380 - auc: 0.9678 - val_loss: 0.1745 - val_auc: 0.8561
Epoch 55/100
24/24 - 1s - loss: 0.1368 - auc: 0.9653 - val_loss: 0.1736 - val_auc: 0.8559
Epoch 56/100
24/24 - 1s - loss: 0.1349 - auc: 0.9661 - val_loss: 0.1727 - val_auc: 0.8558
Epoch 57/100
24/24 - 1s - loss: 0.1320 - auc: 0.9696 - val_loss: 0.1720 - val_auc: 0.8561
Epoch 58/100
24/24 - 1s - loss: 0.1300 - auc: 0.9702 - val_loss: 0.1712 - val_auc: 0.8557
Epoch 59/100
24/24 - 1s - loss: 0.1285 - auc: 0.9696 - val_loss: 0.1706 - val_auc: 0.8569
Epoch 60/100
24/24 - 1s - loss: 0.1264 - auc: 0.9711 - val_loss: 0.1700 - val_auc: 0.8562
Epoch 61/100
24/24 - 1s - loss: 0.1256 - auc: 0.9701 - val_loss: 0.1694 - val_auc: 0.8562
Epoch 62/100
24/24 - 1s - loss: 0.1221 - auc: 0.9748 - val_loss: 0.1689 - val_auc: 0.8570
Epoch 63/100
24/24 - 1s - loss: 0.1216 - auc: 0.9745 - val_loss: 0.1684 - val_auc: 0.8567
Epoch 64/100
24/24 - 1s - loss: 0.1199 - auc: 0.9737 - val_loss: 0.1680 - val_auc: 0.8560
Epoch 65/100
24/24 - 1s - loss: 0.1184 - auc: 0.9744 - val_loss: 0.1676 - val_auc: 0.8566
Epoch 66/100
24/24 - 1s - loss: 0.1172 - auc: 0.9748 - val_loss: 0.1673 - val_auc: 0.8571
Epoch 67/100
24/24 - 1s - loss: 0.1150 - auc: 0.9758 - val_loss: 0.1669 - val_auc: 0.8577
Epoch 68/100
24/24 - 1s - loss: 0.1139 - auc: 0.9774 - val_loss: 0.1667 - val_auc: 0.8569
Epoch 69/100
24/24 - 1s - loss: 0.1124 - auc: 0.9765 - val_loss: 0.1664 - val_auc: 0.8567
Epoch 70/100
24/24 - 1s - loss: 0.1110 - auc: 0.9786 - val_loss: 0.1662 - val_auc: 0.8565
Epoch 71/100
24/24 - 1s - loss: 0.1100 - auc: 0.9787 - val_loss: 0.1660 - val_auc: 0.8559
Epoch 72/100
24/24 - 1s - loss: 0.1078 - auc: 0.9800 - val_loss: 0.1658 - val_auc: 0.8568
Epoch 73/100
24/24 - 1s - loss: 0.1074 - auc: 0.9792 - val_loss: 0.1656 - val_auc: 0.8566
Epoch 74/100
24/24 - 1s - loss: 0.1061 - auc: 0.9795 - val_loss: 0.1655 - val_auc: 0.8566
Epoch 75/100
24/24 - 1s - loss: 0.1050 - auc: 0.9809 - val_loss: 0.1654 - val_auc: 0.8581
Epoch 76/100
24/24 - 1s - loss: 0.1043 - auc: 0.9799 - val_loss: 0.1653 - val_auc: 0.8576
Epoch 77/100
24/24 - 1s - loss: 0.1021 - auc: 0.9813 - val_loss: 0.1652 - val_auc: 0.8581
Epoch 78/100
24/24 - 1s - loss: 0.1012 - auc: 0.9798 - val_loss: 0.1652 - val_auc: 0.8568
Epoch 79/100
24/24 - 1s - loss: 0.1008 - auc: 0.9807 - val_loss: 0.1651 - val_auc: 0.8572
Epoch 80/100
24/24 - 1s - loss: 0.0991 - auc: 0.9818 - val_loss: 0.1651 - val_auc: 0.8569
Epoch 81/100
24/24 - 1s - loss: 0.0982 - auc: 0.9812 - val_loss: 0.1651 - val_auc: 0.8564
```

```
Epoch 82/100
24/24 - 1s - loss: 0.0975 - auc: 0.9807 - val_loss: 0.1651 - val_auc: 0.8576
Epoch 00082: early stopping
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7ff21c206c50>
```

In [ ]:

```python
# predictions on test data
predictions = model.predict(X_test_enc)
```

In [ ]:

```python
# Kaggle score

res = pd.DataFrame()
res["Id"] = test["id"]
res["ACTION"] = predictions.ravel()

res.to_csv("embedding_demo_2.csv", index = False)
```

Submission and Description	Private Score	Public Score
embedding_demo_2.csv a few seconds ago by ankit chandrakar	0.86334	0.86825

- Seems like model is overfitting a bit with train auc of 0.98 and kaggle private score of 0.86.

Modeling

Logistic Regression

In [ ]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
```

On Original Data

In [ ]:

```python
lr = LogisticRegression(random_state=0,class_weight='balanced')
parameter = {'C':[10**i for i in range(-7,1)]}
#parameter = {'C':[10**i for i in range(-5,5)]}

clf = GridSearchCV(lr, parameter, scoring='roc_auc',return_train_score=True, n_jobs=-1,verbose=1)
#hyperparameter tuning using gridsearch
grid_result = clf.fit(X,Y)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))  #printing best accuracy score for best alpha

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):       #printing accuracy score for all values of alpha
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    5.2s finished
```

```
Best: 0.523268 using {'C': 1e-06}
0.523233 (0.016559) with: {'C': 1e-07}
0.523268 (0.016583) with: {'C': 1e-06}
0.523231 (0.016601) with: {'C': 1e-05}
0.523236 (0.016599) with: {'C': 0.0001}
0.523226 (0.016602) with: {'C': 0.001}
0.523218 (0.016604) with: {'C': 0.01}
0.523217 (0.016604) with: {'C': 0.1}
0.523217 (0.016604) with: {'C': 1}
```

- Results are not good on original data.

On one_hot encoded data

In [ ]:

```
lr = LogisticRegression(random_state=0,class_weight='balanced',max_iter=1000)
parameter = {'C':[10**i for i in range(-5,5)]}
#parameter = {'C':[10**i for i in range(-5,5)]}

clf = GridSearchCV(lr, parameter, scoring='roc_auc',return_train_score=True, n_jobs=-1,verbose=1)
#hyperparameter tuning using gridsearch
grid_result = clf.fit(x_train_ohe,Y)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))  #printing best ac
curacy score for best alpha

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):       #printing accuracy score for all values o
f alpha
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   49.5s finished
```

```
Best: 0.866904 using {'C': 1}
0.669739 (0.015195) with: {'C': 1e-05}
0.676097 (0.016066) with: {'C': 0.0001}
0.714192 (0.018530) with: {'C': 0.001}
0.794784 (0.016075) with: {'C': 0.01}
0.854633 (0.009944) with: {'C': 0.1}
0.866904 (0.007791) with: {'C': 1}
0.858383 (0.008332) with: {'C': 10}
0.840493 (0.009218) with: {'C': 100}
0.819472 (0.011656) with: {'C': 1000}
0.814840 (0.015574) with: {'C': 10000}
```

In [ ]:

```
# best C
C = grid_result.best_params_['C']
C
```

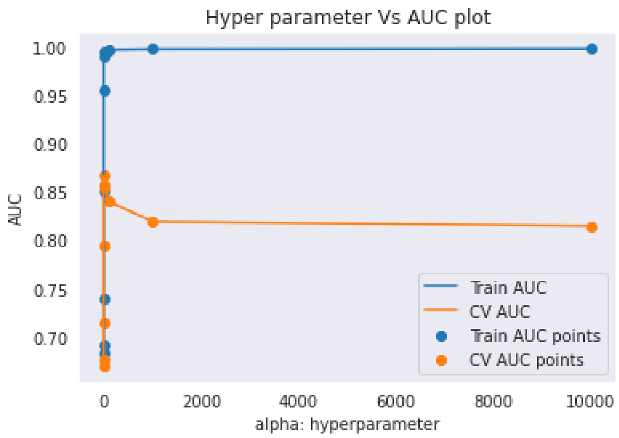
Out[ ]:

```
1
```

In [ ]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_C'])
```

```
#print(results)

train_auc= results['mean_train_score']
cv_auc = results['mean_test_score']

K =  results['param_C']

sns.set_style('darkgrid')
plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



In [ ]:

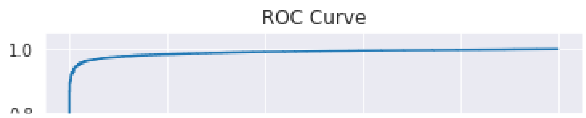
```
lr_model = LogisticRegression(random_state=0,class_weight='balanced',max_iter=1000,C=1)
lr_model.fit(x_train_ohe,Y)
```

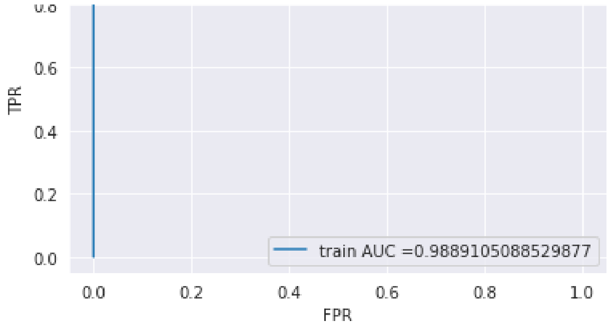
Out[ ]:

```
LogisticRegression(C=1, class_weight='balanced', dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [ ]:

```
y_train_pred = lr_model.predict_proba(x_train_ohe)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y, y_train_pred[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))

plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.show()
```





In [ ]:

```
predictions = lr_model.predict_proba(x_test_ohe)[:,1]
data = {'ID':test["id"],
        'Action':predictions}
res = pd.DataFrame(data)
res.to_csv("logistic_regression_ohe.csv", index = False)
```

Kaggle Score

Submission and Description	Private Score	Public Score
logistic_regression_ohe.csv just now by ankit chandrakar	0.88176	0.88815

- Logistic regression model didn't performed well on original data.
- Performance on one_hot encoded data is really good with private score of 0.88.

SVM

In [ ]:

```
from sklearn.svm import LinearSVC
from scipy.stats import uniform
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC
```

On original data

In [ ]:

```
svm = LinearSVC(verbose=10,class_weight='balanced')
parameters={'C':[10**i for i in range(-5,5)]}
clf = GridSearchCV(svm,parameters,cv=5,verbose=10,scoring='roc_auc',n_jobs=-1)
grid_result = clf.fit(X,Y)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):    #printing accuracy score for all values of
alpha
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    8.0s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   18.7s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   25.8s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   40.0s
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   50.9s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.4min
```

```
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  1.5min finished
```

```
[LibLinear]0.495758 (0.017641) with: {'C': 1e-05}
0.512644 (0.009984) with: {'C': 0.0001}
0.516022 (0.010340) with: {'C': 0.001}
0.504906 (0.016218) with: {'C': 0.01}
0.504706 (0.007861) with: {'C': 0.1}
0.517333 (0.015246) with: {'C': 1}
0.512128 (0.013522) with: {'C': 10}
0.493152 (0.017856) with: {'C': 100}
0.494395 (0.023553) with: {'C': 1000}
0.500625 (0.014493) with: {'C': 10000}
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear fai
led to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

- Best auc of 0.51 on training data, not good.

On one_hot encoded data

In [ ]:

```
svm = LinearSVC(class_weight='balanced',random_state=0)
parameters={'C':[10**i for i in range(-5,5)]}

clf = GridSearchCV(svm,parameters,cv=5,verbose=10,scoring='roc_auc',n_jobs=-1)
grid_result = clf.fit(x_train_ohe, Y)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):   #printing accuracy score for all values of
alpha
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    1.0s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:    1.4s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.1998s.) Setting batch_size=2.
[Parallel(n_jobs=-1)]: Done  22 tasks      | elapsed:    2.4s
[Parallel(n_jobs=-1)]: Batch computation too slow (2.1200s.) Setting batch_size=1.
[Parallel(n_jobs=-1)]: Done  36 tasks      | elapsed:    9.4s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:   14.4s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   17.3s finished
```

```
[LibLinear]0.675464 (0.016407) with: {'C': 1e-05}
0.708540 (0.018482) with: {'C': 0.0001}
0.789363 (0.016375) with: {'C': 0.001}
0.855621 (0.009346) with: {'C': 0.01}
0.867433 (0.007313) with: {'C': 0.1}
0.851004 (0.008209) with: {'C': 1}
0.825095 (0.009918) with: {'C': 10}
0.802364 (0.013876) with: {'C': 100}
0.795965 (0.014582) with: {'C': 1000}
0.793403 (0.014247) with: {'C': 10000}
```

In [ ]:

```
C = grid_result.best_params_['C']
C
```

Out[ ]:

```
0.1
```

In [ ]:

```
svm_model = LinearSVC(C=C,random_state=0,class_weight='balanced')
svm_model = CalibratedClassifierCV(svm_model)
svm_model.fit(x_train_ohe,Y)
```

```
[LibLinear][LibLinear][LibLinear][LibLinear][LibLinear]
```

Out[ ]:

```
CalibratedClassifierCV(base_estimator=LinearSVC(C=0.1, class_weight='balanced',
                                                dual=True, fit_intercept=True,
                                                intercept_scaling=1,
                                                loss='squared_hinge',
                                                max_iter=1000,
                                                multi_class='ovr', penalty='l2',
                                                random_state=None, tol=0.0001,
                                                verbose=1),
                       cv=None, method='sigmoid')
```

In [ ]:

```
predictions = svm_model.predict_proba(x_test_ohe)[:,1]
data = {'ID':test["id"],
        'Action':predictions}
res = pd.DataFrame(data)
res.to_csv("linear_svm_ohe.csv", index = False)
```

Submission and Description	Private Score	Public Score
linear_svm_ohe.csv a few seconds ago by ankit chandrakar linear svc ohe	0.88236	0.88858

- Private score = 0.88 on one_hot encoded data.

XGBoost

In [ ]:

```
import xgboost as xgb
from scipy import stats
```

On Original Data

In [ ]:

```
params = {'n_estimators': stats.randint(150, 1000),
              'learning_rate': stats.uniform(0.01, 0.6),
              'subsample': stats.uniform(),
              'max_depth': [3,5,10,20],
              'colsample_bytree': stats.uniform(),
              'min_child_weight': [1, 2, 3, 4]
             }
x = xgb.XGBClassifier(objective = 'binary:logistic',random_state=0)
clf = RandomizedSearchCV(x,params,random_state=0,cv=3,verbose=10,scoring='roc_auc',n_jobs=-1,n_iter
=50)
best_model=clf.fit(X,Y)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   29.3s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   59.9s
```

```
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:  3.2min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  4.8min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  6.2min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:  7.7min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed: 12.2min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed: 14.3min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed: 16.4min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed: 17.2min finished
```

In [ ]:

```
# printing results of training
data = pd.DataFrame.from_dict(best_model.cv_results_).sort_values(by = "mean_test_score", 
ascending=False)
data.head(10)
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_colsample_bytree	param_learning_rate	param_max_depth	pa
36	38.444025	0.265638	1.553602	0.048692	0.855803	0.0170285	20	
47	7.447735	0.058304	0.435113	0.009814	0.229219	0.538951	20	
6	36.799969	0.114985	1.445446	0.030425	0.758616	0.0735446	20	
43	26.189451	0.037725	1.248993	0.014694	0.680545	0.0611773	10	
1	28.624212	0.169257	1.217180	0.010872	0.623564	0.240629	20	
0	26.890684	0.608804	1.261433	0.031535	0.548814	0.439114	20	
40	22.676457	0.222880	1.781196	0.027588	0.237893	0.570528	20	
33	46.990056	0.329874	1.890684	0.012423	0.868126	0.107496	20	
25	31.080992	0.238144	1.313645	0.016209	0.767024	0.257092	20	
45	30.164230	0.303719	1.269791	0.017651	0.743835	0.297158	20	

In [ ]:

```
best_model.best_params_
```

Out[ ]:

```
{'colsample_bytree': 0.855803342392611,
 'learning_rate': 0.017028450511001183,
 'max_depth': 20,
 'min_child_weight': 2,
 'n_estimators': 491,
 'subsample': 0.7499992487701004}
```

In [ ]:

```
colsample_bytree = best_model.best_params_['colsample_bytree']
learning_rate=best_model.best_params_['learning_rate']
max_depth=best_model.best_params_['max_depth']
min_child_weight=best_model.best_params_['min_child_weight']
n_estimators=best_model.best_params_['n_estimators']
subsample=best_model.best_params_['subsample']
```

In [ ]:

```
xgb_model =
xgb.XGBClassifier(colsample_bytree=colsample_bytree,learning_rate=learning_rate,max_depth=max_depth
,

min_child_weight=min_child_weight,n_estimators=n_estimators,subsample=subsample,random_state=0,obj
ective='binary:logistic')

xgb_model.fit(X,Y)
```

Out[ ]:

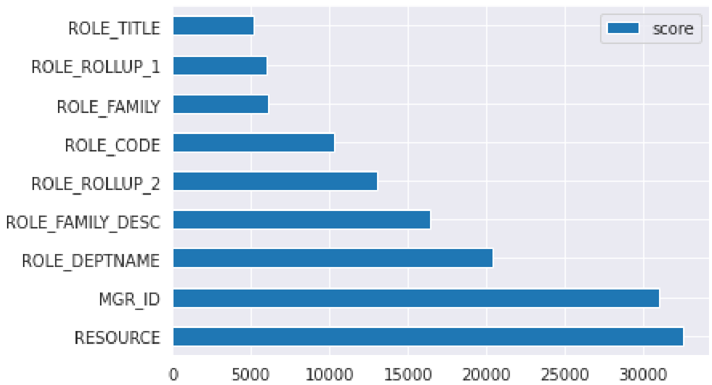
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.855803342392611, gamma=0,
              learning_rate=0.017028450511001183, max_delta_step=0,
              max_depth=20, min_child_weight=2, missing=None, n_estimators=491,
              n_jobs=1, nthread=None, objective='binary:logistic',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=None, silent=None, subsample=0.7499992487701004,
              verbosity=1)
```

In [ ]:

```
#https://stackoverflow.com/a/52777909/13401359
# plotting feature importance

feature_important = xgb_model.get_booster().get_score(importance_type='weight')
keys = list(feature_important.keys())
values = list(feature_important.values())

data = pd.DataFrame(data=values, index=keys, columns=["score"]).sort_values(by = "score", ascending
=False)
sns.set_style('darkgrid')
data.plot(kind='barh')
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a4512da58>
```



In [ ]:

```
t = X_test[model.get_booster().feature_names]
```

In [ ]:

```
predictions = xgb_model.predict_proba(t)[:,1]
data = {'ID':test["id"],
        'Action':predictions}
```

```
res = pd.DataFrame(data)
res.to_csv("xgboost.csv", index = False)
```

Submission and Description	Private Score	Public Score
xgboost.csv just now by ankit chandrakar	0.86882	0.87329

- private score of 0.86 on not-encoded original data.

On one_hot encoded data

In [ ]:

```
params = {'n_estimators': stats.randint(150, 1000),
              'learning_rate': stats.uniform(0.01, 0.5),
              'subsample': stats.uniform(),
              'max_depth': [3,5,10,20],
              'colsample_bytree': stats.uniform(),
              'min_child_weight': [1, 2, 3, 4]
             }
x = xgb.XGBClassifier(objective = 'binary:logistic',random_state=0)
clf = RandomizedSearchCV(x,params,random_state=0,cv=3,verbose=10,scoring='roc_auc',n_jobs=-1,n_iter
=50)
best_model=clf.fit(x_train_ohe,Y)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  6.7min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  7.7min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 10.1min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 10.8min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 11.8min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 14.4min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed: 16.3min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed: 17.5min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed: 22.7min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed: 27.2min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed: 37.2min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed: 42.4min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed: 48.8min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed: 51.4min finished
```

In [ ]:

```
data = pd.DataFrame.from_dict(best_model.cv_results_).sort_values(by = "mean_test_score",
ascending=False)
```

In [ ]:

```
data.head(10)
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_colsample_bytree	param_learning_rate	param_max_depth	pa
47	14.637066	0.029733	0.350040	0.000475	0.229219	0.450793	20	
40	58.755151	0.287809	1.820313	0.036030	0.237893	0.477107	20	
45	113.264374	0.107808	1.378625	0.023527	0.743835	0.249298	20	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_colsample_bytree	param_learning_rate	param_max_depth	pa
39	70.868121	0.065669	0.934240	0.013633	0.596655	0.401822	20	
25	120.561068	0.104147	1.403155	0.014518	0.767024	0.21591	20	
33	157.463008	0.071701	1.624585	0.025410	0.868126	0.0912465	20	
15	103.280871	0.053857	1.248568	0.004949	0.67366	0.495973	20	
1	85.444303	0.219927	1.037348	0.055905	0.623564	0.202191	20	
4	14.322339	0.033516	0.404431	0.003785	0.140351	0.445044	20	
35	71.109990	0.036788	0.627242	0.008610	0.697429	0.236771	10	

In [ ]:

```
best_model.best_params_
```

Out[ ]:

```
{'colsample_bytree': 0.22921932308657944,
 'learning_rate': 0.4507926996052904,
 'max_depth': 20,
 'min_child_weight': 1,
 'n_estimators': 243,
 'subsample': 0.8310484552361904}
```

In [ ]:

```
colsample_bytree = best_model.best_params_['colsample_bytree']
learning_rate=best_model.best_params_['learning_rate']
max_depth=best_model.best_params_['max_depth']
min_child_weight=best_model.best_params_['min_child_weight']
n_estimators=best_model.best_params_['n_estimators']
subsample=best_model.best_params_['subsample']
```

In [ ]:

```
xgb_model =
xgb.XGBClassifier(colsample_bytree=colsample_bytree,learning_rate=learning_rate,max_depth=max_depth
,

min_child_weight=min_child_weight,n_estimators=n_estimators,subsample=subsample,random_state=0,obj
ective='binary:logistic')

xgb_model.fit(x_train_ohe,Y)
```

Out[ ]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.22921932308657944, gamma=0,
              learning_rate=0.4507926996052904, max_delta_step=0, max_depth=20,
              min_child_weight=1, missing=None, n_estimators=243, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=0.8310484552361904, verbosity=1)
```

In [ ]:

```
predictions = xgb_model.predict_proba(x_test_ohe)[:,1]
data = {'ID':test["id"],
        'Action':predictions}
```

```
res = pd.DataFrame(data)
res.to_csv("xgboost_ohe.csv", index = False)
```

Submission and Description	Private Score	Public Score
xgboost_ohe.csv just now by ankit chandrakar	0.85701	0.86212

- Performace on original data is better for xgboost classifier than on one_hot_encoded data.
- 'RESOURCE' & 'MGR_ID' were the most important features.

CatBoost

ref = https://www.kaggle.com/mitribunskiy/tutorial-catboost-overview

In [ ]:

```
! pip3 install catboost
```

```
Collecting catboost
  Downloading
https://files.pythonhosted.org/packages/20/37/bc4e0ddc30c07a96482abf1de7ed1ca54e59bba2026a33bca6d2e
e5b/catboost-0.24.4-cp36-none-manylinux1_x86_64.whl (65.7MB)
     |████████████████████████████████| 65.8MB 49kB/s
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from
catboost) (1.19.5)
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from catboost)
(0.10.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from catboost)
(4.4.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from
catboost) (3.2.2)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catboost)
(1.15.0)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packages (from
catboost) (1.1.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from catboost)
(1.4.1)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from
plotly->catboost) (1.3.3)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages
(from matplotlib->catboost) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from
matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from
matplotlib->catboost) (0.10.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from
pandas>=0.24.0->catboost) (2018.9)
Installing collected packages: catboost
Successfully installed catboost-0.24.4
```

In [ ]:

```
from catboost import CatBoostClassifier
```

In [ ]:

```
# splitting data into train/validation set
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.25,stratify=y_t
rain)
```

In [ ]:

```
# creating a list of features which we want catboost model to treat as categorical_feature
features = list(range(X_train.shape[1]))
```

```python
print(features)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [ ]:

```python
params = {'loss_function':'Logloss',
          'eval_metric':'AUC',
          'cat_features':features,
          'verbose':200,
          'early_stopping_rounds': 200,
          'random_seed':1
          }

clf = CatBoostClassifier(**params,use_best_model=True)

clf.fit(X_train, y_train,
          eval_set=(X_valid, y_valid),
          use_best_model=True,
          plot=True);
```

```
Learning rate set to 0.069882
0: test: 0.5477370 best: 0.5477370 (0) total: 85ms remaining: 1m 24s
200: test: 0.8918628 best: 0.8918628 (200) total: 10.6s remaining: 42.2s
400: test: 0.8960038 best: 0.8960494 (393) total: 22s remaining: 32.9s
600: test: 0.8980542 best: 0.8982337 (593) total: 33.7s remaining: 22.4s
800: test: 0.8982004 best: 0.8985327 (671) total: 45.7s remaining: 11.4s
Stopped by overfitting detector  (200 iterations wait)

bestTest = 0.8985327454
bestIteration = 671

Shrink model to first 672 iterations.
```

In [ ]:

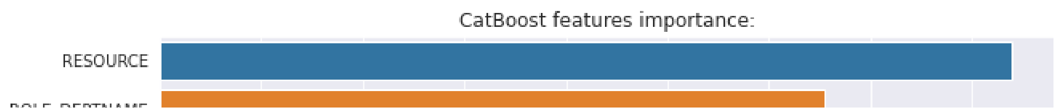
```python
feature_imp = clf.get_feature_importance(prettified=True)
feature_imp
```

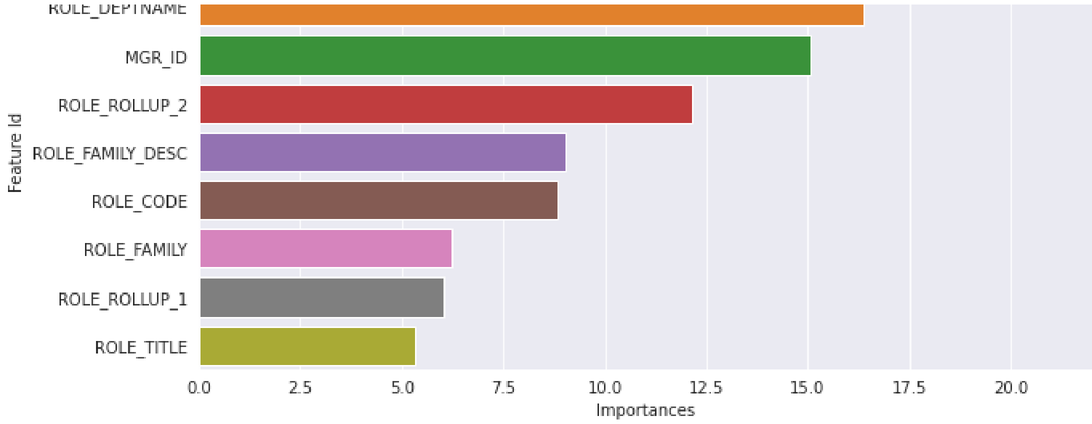
Out[ ]:

	Feature Id	Importances
0	RESOURCE	20.963298
1	ROLE_DEPTNAME	16.378010
2	MGR_ID	15.058625
3	ROLE_ROLLUP_2	12.156828
4	ROLE_FAMILY_DESC	9.015432
5	ROLE_CODE	8.850208
6	ROLE_FAMILY	6.226168
7	ROLE_ROLLUP_1	6.042361
8	ROLE_TITLE	5.309069

In [ ]:

```python
# plotting feature importance

sns.set_style("darkgrid")
plt.figure(figsize=(10, 5));
sns.barplot(y=feature_imp['Feature Id'], x=feature_imp['Importances'], data=feature_imp);
plt.title('CatBoost features importance:');
```





Submission and Description	Private Score	Public Score
catboost.csv a few seconds ago by ankit chandrakar	0.90865	0.91328

- Catboost model performed best out of all models.
- 'RESOURCE' was most important feature

Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
```

On Original data

In [ ]:

```
r = RandomForestClassifier(random_state=0,class_weight='balanced')

n_estimators = [10,20,100,200,500]
max_depth = [1,10,20,50]
max_features=[1,2,3,4,5]
min_samples_split=[2,5,7,10,20]

params = {'n_estimators':n_estimators,
         'max_depth':max_depth,
         'max_features':max_features,
         'min_samples_split':min_samples_split}

clf = RandomizedSearchCV(r,params,verbose=10,scoring='roc_auc',n_jobs=-1,return_train_score=True,ra
ndom_state=0,n_iter=50,cv=3)
clf = clf.fit(X,Y)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    1.3s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   14.9s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   29.5s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   33.8s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   36.3s
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed:  5.7min
```

```
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed:  6.8min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:  7.1min finished
```

In [ ]:

```
data = pd.DataFrame.from_dict(clf.cv_results_).sort_values(by = "mean_test_score", ascending=False)
data.head(10)
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_min_samples_split	param_max_features
25	7.781401	0.104589	0.397354	0.007121	200	7	3
7	28.883591	0.347014	0.979427	0.012759	500	7	5
16	9.319182	0.125402	0.382355	0.003274	200	10	4
48	5.648349	0.096797	0.397080	0.012846	200	10	2
35	5.569864	0.068283	0.188302	0.002782	100	10	5
13	14.300058	0.195125	1.005710	0.000723	500	7	2
27	5.595344	0.065892	0.204321	0.010846	100	10	5
34	27.087074	0.328897	0.890475	0.014150	500	20	5
40	3.605768	0.055935	0.186792	0.002742	100	20	3
2	5.884104	0.098252	0.411510	0.005664	200	5	2

In [ ]:

```
clf.best_params_
```

Out[ ]:

```
{'max_depth': 50,
 'max_features': 3,
 'min_samples_split': 7,
 'n_estimators': 200}
```

In [ ]:

```
rf_model =
RandomForestClassifier(max_depth=50,n_estimators=200,class_weight='balanced',max_features=3,min_sam
ples_split=7,random_state=0)
rf_model.fit(X,Y)
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                       criterion='gini', max_depth=50, max_features=3,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=7,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [ ]:

```
predictions = rf_model.predict_proba(X_test)[:,1]
data = {'ID':test["id"],
        'Action':predictions}
res = pd.DataFrame(data)
res.to_csv("random_forest.csv", index = False)
```

random_forest.csv 36 minutes ago by ankit chandrakar Random Forest	0.56254	0.57455

- Altough model performed well while training, but on kaggle test data, the score was too low.

On one_hot encoded data

In [ ]:

```
r = RandomForestClassifier(random_state=0,class_weight='balanced')

n_estimators = [10,20,100,200,500]
max_depth = [1,10,20,50]
max_features=[1,2,3,4,5]
min_samples_split=[2,5,7,10,20]

params = {'n_estimators':n_estimators,
          'max_depth':max_depth,
          'max_features':max_features,
          'min_samples_split':min_samples_split}

clf = RandomizedSearchCV(r,params,verbose=10,scoring='roc_auc',n_jobs=-1,return_train_score=True,ra
ndom_state=0,n_iter=50,cv=3)
clf = clf.fit(x_train_ohe,Y)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    1.3s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   10.9s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   18.2s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   37.2s
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   44.9s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   57.8s
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:  2.6min finished
```

In [ ]:

```
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
```

```
params = clf.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):   #printing auc score for all values of alpha
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
0.705869 (0.017158) with: {'n_estimators': 20, 'min_samples_split': 7, 'max_features': 5,
'max_depth': 20}
0.624025 (0.010340) with: {'n_estimators': 200, 'min_samples_split': 20, 'max_features': 3,
'max_depth': 1}
0.835375 (0.005977) with: {'n_estimators': 500, 'min_samples_split': 20, 'max_features': 5,
'max_depth': 20}
0.554142 (0.004631) with: {'n_estimators': 10, 'min_samples_split': 5, 'max_features': 2,
'max_depth': 10}
0.740317 (0.017305) with: {'n_estimators': 500, 'min_samples_split': 2, 'max_features': 5,
'max_depth': 1}
0.849402 (0.015398) with: {'n_estimators': 500, 'min_samples_split': 10, 'max_features': 1,
'max_depth': 50}
0.741614 (0.024224) with: {'n_estimators': 100, 'min_samples_split': 2, 'max_features': 1,
'max_depth': 50}
0.740317 (0.017305) with: {'n_estimators': 500, 'min_samples_split': 20, 'max_features': 5,
'max_depth': 1}
0.624025 (0.010340) with: {'n_estimators': 200, 'min_samples_split': 10, 'max_features': 3,
'max_depth': 1}
0.661110 (0.016864) with: {'n_estimators': 10, 'min_samples_split': 2, 'max_features': 4,
'max_depth': 50}
0.610408 (0.009568) with: {'n_estimators': 500, 'min_samples_split': 5, 'max_features': 1,
'max_depth': 1}
0.822109 (0.006932) with: {'n_estimators': 500, 'min_samples_split': 10, 'max_features': 3,
'max_depth': 10}
0.752374 (0.011634) with: {'n_estimators': 20, 'min_samples_split': 5, 'max_features': 2,
'max_depth': 50}
0.738362 (0.013402) with: {'n_estimators': 500, 'min_samples_split': 5, 'max_features': 4,
'max_depth': 1}
0.712174 (0.019139) with: {'n_estimators': 20, 'min_samples_split': 20, 'max_features': 5,
'max_depth': 20}
0.827436 (0.015604) with: {'n_estimators': 200, 'min_samples_split': 7, 'max_features': 1,
'max_depth': 50}
0.750107 (0.014350) with: {'n_estimators': 10, 'min_samples_split': 20, 'max_features': 5,
'max_depth': 50}
0.505009 (0.000364) with: {'n_estimators': 10, 'min_samples_split': 5, 'max_features': 2,
'max_depth': 1}
0.713607 (0.009723) with: {'n_estimators': 20, 'min_samples_split': 10, 'max_features': 3,
'max_depth': 20}
0.845721 (0.008427) with: {'n_estimators': 200, 'min_samples_split': 5, 'max_features': 2,
'max_depth': 50}
0.739195 (0.009148) with: {'n_estimators': 10, 'min_samples_split': 10, 'max_features': 5,
'max_depth': 50}
0.788036 (0.013251) with: {'n_estimators': 20, 'min_samples_split': 10, 'max_features': 5,
'max_depth': 50}
0.617917 (0.008910) with: {'n_estimators': 10, 'min_samples_split': 5, 'max_features': 2,
'max_depth': 20}
0.705560 (0.018530) with: {'n_estimators': 20, 'min_samples_split': 5, 'max_features': 5,
'max_depth': 20}
0.522085 (0.004857) with: {'n_estimators': 20, 'min_samples_split': 2, 'max_features': 4,
'max_depth': 1}
0.785272 (0.009166) with: {'n_estimators': 20, 'min_samples_split': 7, 'max_features': 4,
'max_depth': 50}
0.844543 (0.004149) with: {'n_estimators': 100, 'min_samples_split': 20, 'max_features': 5,
'max_depth': 50}
0.642227 (0.009305) with: {'n_estimators': 20, 'min_samples_split': 7, 'max_features': 4,
'max_depth': 10}
0.547859 (0.003851) with: {'n_estimators': 20, 'min_samples_split': 2, 'max_features': 5,
'max_depth': 1}
0.843446 (0.007885) with: {'n_estimators': 500, 'min_samples_split': 5, 'max_features': 4,
'max_depth': 20}
0.676447 (0.008339) with: {'n_estimators': 10, 'min_samples_split': 2, 'max_features': 5,
'max_depth': 50}
0.715535 (0.007824) with: {'n_estimators': 20, 'min_samples_split': 7, 'max_features': 4,
'max_depth': 20}
0.730011 (0.021860) with: {'n_estimators': 10, 'min_samples_split': 10, 'max_features': 3,
'max_depth': 50}
0.790838 (0.006165) with: {'n_estimators': 200, 'min_samples_split': 20, 'max_features': 2,
'max_depth': 10}
0.531018 (0.002289) with: {'n_estimators': 100, 'min_samples_split': 2, 'max_features': 1,
'max_depth': 1}
0.831866 (0.004346) with: {'n_estimators': 200, 'min_samples_split': 5, 'max_features': 4,
'max_depth': 20}
```

```
'max_depth': 20}
0.853299 (0.010303) with: {'n_estimators': 500, 'min_samples_split': 5, 'max_features': 2,
'max_depth': 50}
0.504268 (0.000907) with: {'n_estimators': 10, 'min_samples_split': 20, 'max_features': 3,
'max_depth': 1}
0.826817 (0.010526) with: {'n_estimators': 500, 'min_samples_split': 5, 'max_features': 4,
'max_depth': 10}
0.624025 (0.010340) with: {'n_estimators': 200, 'min_samples_split': 7, 'max_features': 3,
'max_depth': 1}
0.837724 (0.016100) with: {'n_estimators': 500, 'min_samples_split': 5, 'max_features': 1,
'max_depth': 50}
0.646419 (0.012500) with: {'n_estimators': 200, 'min_samples_split': 10, 'max_features': 4,
'max_depth': 1}
0.730052 (0.003244) with: {'n_estimators': 10, 'min_samples_split': 7, 'max_features': 5,
'max_depth': 50}
0.577598 (0.008147) with: {'n_estimators': 10, 'min_samples_split': 7, 'max_features': 3,
'max_depth': 10}
0.604148 (0.010635) with: {'n_estimators': 200, 'min_samples_split': 5, 'max_features': 2,
'max_depth': 1}
0.589401 (0.008674) with: {'n_estimators': 100, 'min_samples_split': 2, 'max_features': 4,
'max_depth': 1}
0.505014 (0.000265) with: {'n_estimators': 10, 'min_samples_split': 2, 'max_features': 1,
'max_depth': 1}
0.511253 (0.001665) with: {'n_estimators': 20, 'min_samples_split': 7, 'max_features': 1,
'max_depth': 1}
0.706349 (0.027565) with: {'n_estimators': 10, 'min_samples_split': 10, 'max_features': 2,
'max_depth': 50}
0.531018 (0.002289) with: {'n_estimators': 100, 'min_samples_split': 20, 'max_features': 1,
'max_depth': 1}
```

In [ ]:

```
clf.best_params_
```

Out[ ]:

```
{'max_depth': 50,
 'max_features': 2,
 'min_samples_split': 5,
 'n_estimators': 500}
```

In [ ]:

```
rf_model = RandomForestClassifier(max_depth=50,n_estimators=500,max_features=2,min_samples_split=5
,class_weight='balanced')
rf_model.fit(x_train_ohe,Y)
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                       criterion='gini', max_depth=50, max_features=2,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
predictions = rf_model.predict_proba(x_test_ohe)[:,1]
data = {'ID':test["id"],
        'Action':predictions}
res = pd.DataFrame(data)
res.to_csv("logistic_regression_ohe.csv", index = False)
```

Submission and Description	Private Score	Public Score
random_forest_ohe_demo.csv just now by ankit chandrakar add submission details	0.85686	0.86124

- Much better performance on one_hot encoded data. But not better than other classifiers.

Conclusion

Model	Encoding	Public Score	Private Score
CatBoost	None	0.913	0.908
SVM	one_hot	0.888	0.882
Log Reg	one_hot	0.888	0.881
XGBoost	None	0.873	0.868
Neural Network	embedding	0.868	0.863
XGBoost	one_hot	0.862	0.857
Random Forest	one_hot	0.861	0.856
Random Forest	None	0.57	0.56

- CatBoost was best model.
- 'RESOURCE' feature was most important.