The Task

1. Research:

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import cohen_kappa_score
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
%matplotlib inline
```

Let's take a look at predict_failure data

In [2]:
```
df = pd.read_csv('predict_failure.csv')
df.info()
df.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 124494 entries, 0 to 124493
Data columns (total 12 columns):
date          124494 non-null object
device        124494 non-null object
failure       124494 non-null int64
attribute1    124494 non-null int64
attribute2    124494 non-null int64
attribute3    124494 non-null int64
attribute4    124494 non-null int64
attribute5    124494 non-null int64
attribute6    124494 non-null int64
attribute7    124494 non-null int64
attribute8    124494 non-null int64
attribute9    124494 non-null int64
dtypes: int64(10), object(2)
memory usage: 11.4+ MB
```

Out[2]:

	date	device	failure	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attri
0	2015-01-01	S1F01085	0	215630672	56	0	52	6	407438	
1	2015-01-01	S1F0166B	0	61370680	0	3	0	6	403174	
2	2015-01-01	S1F01E6Y	0	173295968	0	0	0	12	237394	
3	2015-01-01	S1F01JE0	0	79694024	0	0	0	6	410186	
4	2015-01-01	S1F01R2B	0	135970480	0	0	0	15	313173	
5	2015-01-01	S1F01TD5	0	68837488	0	0	41	6	413535	
6	2015-01-01	S1F01XDJ	0	227721632	0	0	0	8	402525	
7	2015-01-01	S1F023H2	0	141503600	0	0	1	19	494462	
8	2015-01-01	S1F02A0J	0	8217840	0	1	0	14	311869	
9	2015-01-01	S1F02DZ2	0	116440096	0	323	9	9	407905	

In order to resolve unbalance data problem, we are increasing number of positive samples by creating 100

copy of each positive samples.

We basicaly read thru entire data set and identify positive samples (failure==1). Then copy each and every of them, generate 100 of them and finally append them to our data frame.

So that, we would add 100*positive samples which is roughly 10,000 positive samples. This approach is called "Oversampling" to handle unbalanced data set problems

In [4]:

```
# This is how we implement to "Oversampling"
j=0
k=1
for i in range(1,len(df)):  # Go thru entire data set
    if df.iloc[i,2]==1:     # Identifying positive samples
        for j in range(101):
            df.loc[len(df)+k]=df.iloc[i,:]  # Copying each samples 100 times an
            k+=1
```

In [9]:
```
# Mixing the data, so that positive and negative samples are randomly distribut
df_new=df.reindex(np.random.permutation(df.index))
df_new.head(10) # Mixed first 10 samples
```

Out[9]:

	date	device	failure	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6
38691	2015-02-20	S1F117PK	0	23337704	0	0	0	12	21527
74232	2015-04-25	S1F0C95J	0	49665864	0	0	0	5	26968
25479	2015-02-01	W1F0X6V0	0	27524104	0	0	0	11	23648
12209	2015-01-14	S1F0CVWK	0	5591504	1928	0	6	7	36532
17790	2015-01-21	Z1F0L3BL	0	53565592	0	0	0	6	30544
53684	2015-03-14	W1F0VA0G	0	75590136	0	56	0	12	31844
128669	2015-01-27	W1F03DP4	1	166313728	0	8	19	17	33161
122620	2015-10-08	W1F18TKX	0	18095904	0	0	0	12	24992
114693	2015-08-20	W1F111N7	0	124630728	0	0	0	8	22467
54014	2015-03-15	S1F0GSD9	0	44318952	0	0	0	12	22501

In [8]:
```
df.head(10) # Original first 10 samples
```

Out[8]:

	date	device	failure	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attri
0	2015-01-01	S1F01085	0	215630672	56	0	52	6	407438	
1	2015-01-01	S1F0166B	0	61370680	0	3	0	6	403174	
2	2015-01-01	S1F01E6Y	0	173295968	0	0	0	12	237394	
3	2015-01-01	S1F01JE0	0	79694024	0	0	0	6	410186	
4	2015-01-01	S1F01R2B	0	135970480	0	0	0	15	313173	
5	2015-01-01	S1F01TD5	0	68837488	0	0	41	6	413535	
6	2015-01-01	S1F01XDJ	0	227721632	0	0	0	8	402525	
7	2015-01-01	S1F023H2	0	141503600	0	0	1	19	494462	
8	2015-01-01	S1F02A0J	0	8217840	0	1	0	14	311869	
9	2015-01-01	S1F02DZ2	0	116440096	0	323	9	9	407905	

Arrange feature_vectors and labels

In [70]:
```
feature_vectors=df_new[['attribute1','attribute2','attribute3','attribute4','at
labels=df_new['failure']
```

In [71]:
```
feature_vectors.head()
```

Out[71]:

	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attribute7	attribute8	attribu
38691	23337704	0	0	0	12	215277	0	0	
74232	49665864	0	0	0	5	269684	0	0	
25479	27524104	0	0	0	11	236484	0	0	
12209	5591504	1928	0	6	7	365325	0	0	
17790	53565592	0	0	0	6	305443	0	0	

In [72]:
```
labels.head()
```

Out[72]:
```
38691    0
74232    0
25479    0
12209    0
17790    0
Name: failure, dtype: int64
```

Now, let's normalize our feature_vectors as follows (or sklearn):

In [73]:
```
feature_vectors=feature_vectors.apply(lambda x: (x-x.min())/(x.max()-x.min()))
```

In [74]:
```
feature_vectors.head()
```

Out[74]:

	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attribute7	attribute8	attribu
38691	0.095591	0.000000	0.0	0.000000	0.113402	0.312368	0.0	0.0	
74232	0.203431	0.000000	0.0	0.000000	0.041237	0.391315	0.0	0.0	
25479	0.112739	0.000000	0.0	0.000000	0.103093	0.343140	0.0	0.0	
12209	0.022903	0.029676	0.0	0.003601	0.061856	0.530096	0.0	0.0	
17790	0.219405	0.000000	0.0	0.000000	0.051546	0.443203	0.0	0.0	

Next, let's split our data into train and test data as follows:

In [75]:
```
X_train, X_test, y_train, y_test = train_test_split(feature_vectors, labels, te
```

In [76]:
```
#X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=
```

In [77]:
```
X_train.shape,  X_test.shape, y_train.shape,  y_test.shape
```

Out[77]:
```
((108160, 9), (27040, 9), (108160,), (27040,))
```

Now, let's explore various ML models for our data:

1-) SVM (Support Vector Machine):

In [78]:
```
support_vector_classifier = SVC(kernel='rbf') # Rbf kernel for nonlinear classi
support_vector_classifier.fit(X_train,y_train)
y_pred_svc = support_vector_classifier.predict(X_test)
```

```
C:\Users\Sertan\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

In [79]:
```
# Generate confusion matrix and accuracy
cm_support_vector_classifier = confusion_matrix(y_test,y_pred_svc) # Let's obta
print(cm_support_vector_classifier,end='\n\n')
numerator = cm_support_vector_classifier[0][0] + cm_support_vector_classifier[1
denominator = sum(cm_support_vector_classifier[0]) + sum(cm_support_vector_clas
acc_svc = (numerator/denominator) * 100
print("Accuracy : ",round(acc_svc,4),"%") # Let's find out the accurarcy
```

```
[[24834    76]
 [ 1683   447]]

Accuracy :  93.4948 %
```

In [80]:
```
# Since when we split the data, it performs operation randomly. Therefore, due
# different accuracy. For that reason, we need to check cross validation. The c
# the accuracy leaving one testing out in different times. And then come out wi
cross_val_svc = cross_val_score(estimator = SVC(kernel = 'rbf'), X = X_train, y
print("Cross Validation Accuracy : ",round(cross_val_svc.mean() * 100 , 4),"%")
```

```
Cross Validation Accuracy :  93.3746 %
```

In [81]:
```
## Since data set is very unbalanced, let's check Cohen's Kappa Score
cohen_score = cohen_kappa_score(y_test,y_pred_svc)
round(cohen_score,4)
```

Out[81]:
```
0.3157
```

2-) Random Forest Model:

In [82]:
```
random_forest_classifier = RandomForestClassifier()
random_forest_classifier.fit(X_train,y_train)
y_pred_rfc = random_forest_classifier.predict(X_test)
```

```
C:\Users\Sertan\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: Fut
ureWarning: The default value of n_estimators will change from 10 in version 0.
20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [83]:
```
# Generate confusion matrix and accuracy
cm_random_forest_classifier = confusion_matrix(y_test,y_pred_rfc)
print(cm_random_forest_classifier,end="\n\n")
numerator = cm_random_forest_classifier[0][0] + cm_random_forest_classifier[1][
denominator = sum(cm_random_forest_classifier[0]) + sum(cm_random_forest_classi
acc_rfc = (numerator/denominator) * 100
print("Accuracy : ",round(acc_rfc,4),"%")
```

```
[[24900    10]
 [    0  2130]]

Accuracy :  99.963 %
```

In [84]:
```
cross_val_rfc = cross_val_score(estimator=RandomForestClassifier(), X=X_train,
print("Cross Validation Accuracy : ",round(cross_val_rfc.mean() * 100 , 4),"%")
```

```
Cross Validation Accuracy :  99.9695 %
```

In [85]:
```
## Since data set is very unbalanced, let's check Cohen's Kappa Score
cohen_score = cohen_kappa_score(y_test,y_pred_rfc)
round(cohen_score,4)
```

Out[85]: 0.9975

3-) XGBoost Model:

In [86]:
```
xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train,y_train)
y_pred_xgb = xgb_classifier.predict(X_test)
```

In [87]:
```
# Generate confusion matrix and accuracy
cm_xgb_classifier = confusion_matrix(y_test,y_pred_xgb)
print(cm_xgb_classifier,end='\n\n')
numerator = cm_xgb_classifier[0][0] + cm_xgb_classifier[1][1]
denominator = sum(cm_xgb_classifier[0]) + sum(cm_xgb_classifier[1])
acc_xgb = (numerator/denominator) * 100
print("Accuracy : ",round(acc_xgb,4),"%")
```

```
[[24755   155]
 [  688  1442]]

Accuracy :  96.8824 %
```

In [88]:
```
cross_val_xgb = cross_val_score(estimator=XGBClassifier(), X=X_train, y=y_trair
print("Cross Validation Accuracy : ",round(cross_val_xgb.mean() * 100 , 4),"%")
```

```
Cross Validation Accuracy :  96.9148 %
```

In [89]:
```
## Since data set is very unbalanced, let's check Cohen's Kappa Score
cohen_score = cohen_kappa_score(y_test,y_pred_xgb)
round(cohen_score,4)
```

Out[89]: 0.7574

4-) Naive Base Model:

In [90]:
```
# fit a Naive Bayes model to the data
modelnb = GaussianNB()
modelnb.fit(X_train, y_train)
```

Out[90]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [91]:
```
y_pred_nb = modelnb.predict(X_test) # Prediction
```

In [92]:
```
# Generate confusion matrix and accuracy
cm_nb_classifier = confusion_matrix(y_test,y_pred_nb)
print(cm_nb_classifier,end='\n\n')
numerator = cm_nb_classifier[0][0] + cm_nb_classifier[1][1]
denominator = sum(cm_nb_classifier[0]) + sum(cm_nb_classifier[1])
acc_nb = (numerator/denominator) * 100
print("Accuracy : ",round(acc_nb,4),"%")
```

```
[[24663   247]
 [ 1414   716]]

Accuracy :  93.8572 %
```

In [93]:
```
cross_val_nb = cross_val_score(estimator=GaussianNB(), X=X_train, y=y_train, cv
print("Cross Validation Accuracy : ",round(cross_val_nb.mean() * 100 , 4),"%")
```

```
Cross Validation Accuracy :  93.7824 %
```

In [94]:
```
## Since data set is very unbalanced, let's check Cohen's Kappa Score
cohen_score = cohen_kappa_score(y_test,y_pred_nb)
round(cohen_score,4)
```

Out[94]:
```
0.4353
```

5-) k-Nearest Neighbor:

In [95]:
```
modelknn = KNeighborsClassifier()
modelknn.fit(X_train, y_train)
```

Out[95]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [96]:
```
y_pred_knn = modelknn.predict(X_test) # Prediction
```

In [97]:
```
# Generate confusion matrix and accuracy
cm_knn_classifier = confusion_matrix(y_test,y_pred_knn)
print(cm_knn_classifier,end='\n\n')
numerator = cm_knn_classifier[0][0] + cm_knn_classifier[1][1]
denominator = sum(cm_knn_classifier[0]) + sum(cm_knn_classifier[1])
acc_knn = (numerator/denominator) * 100
print("Accuracy : ",round(acc_knn,4),"%")
```

```
[[24856    54]
 [    0  2130]]

Accuracy :  99.8003 %
```

In [98]:
```
cross_val_knn = cross_val_score(estimator=KNeighborsClassifier(), X=X_train, y=
print("Cross Validation Accuracy : ",round(cross_val_knn.mean() * 100 , 4),"%")
```

```
Cross Validation Accuracy :  99.8105 %
```

In [99]:
```
## Since data set is very unbalanced, let's check Cohen's Kappa Score
cohen_score = cohen_kappa_score(y_test,y_pred_knn)
round(cohen_score,4)
```

Out[99]:
```
0.9864
```

Deep Learning Models, in particular TensorFlow Estimator API model for our data:

In [100]:
```
import tensorflow as tf
```

In [101]:
```
# For TF Estimator API, we need to define feature columns(Since all of them are
attr1=tf.feature_column.numeric_column('attribute1')
attr2=tf.feature_column.numeric_column('attribute2')
attr3=tf.feature_column.numeric_column('attribute3')
attr4=tf.feature_column.numeric_column('attribute4')
attr5=tf.feature_column.numeric_column('attribute5')
attr6=tf.feature_column.numeric_column('attribute6')
attr7=tf.feature_column.numeric_column('attribute7')
attr8=tf.feature_column.numeric_column('attribute8')
attr9=tf.feature_column.numeric_column('attribute9')
```

In [102]:
```
feat_cols=[attr1, attr2, attr3, attr4, attr5, attr6, attr7, attr8, attr9]
```

In [103]:
```
labels.head()
```

Out[103]:
```
38691    0
74232    0
25479    0
12209    0
17790    0
Name: failure, dtype: int64
```

In [104]:
```
feature_vectors.head()
```

Out[104]:

	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attribute7	attribute8	attribı
38691	0.095591	0.000000	0.0	0.000000	0.113402	0.312368	0.0	0.0	
74232	0.203431	0.000000	0.0	0.000000	0.041237	0.391315	0.0	0.0	
25479	0.112739	0.000000	0.0	0.000000	0.103093	0.343140	0.0	0.0	
12209	0.022903	0.029676	0.0	0.003601	0.061856	0.530096	0.0	0.0	
17790	0.219405	0.000000	0.0	0.000000	0.051546	0.443203	0.0	0.0	

Next, let's split our data into train and test data as follows:

In [105]:
```
## X_train, X_test, y_train, y_test = train_test_split(feature_vectors, labels,
# Since we have already split our data in the previous classification operation
```

In [106]:
```
input_func=tf.estimator.inputs.pandas_input_fn(x=X_train, y=y_train, batch_size
```

In [107]:
```
model=tf.estimator.LinearClassifier(feature_columns=feat_cols, n_classes=2)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\Sertan\A
ppData\Local\Temp\tmpuauuy6ha
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Users\\Sertan\\AppData\\Local
\\Temp\\tmpuauuy6ha', '_tf_random_seed': None, '_save_summary_steps': 100, '_sa
ve_checkpoints_steps': None, '_save_checkpoints_secs': 600, '_session_config':
allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_ste
p_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_protoco
l': None, '_eval_distribute': None, '_experimental_distribute': None, '_servic
e': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec o
bject at 0x000001940CC3EE80>, '_task_type': 'worker', '_task_id': 0, '_global_i
d_in_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': True,
'_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

In [108]:
```
model.train(input_fn=input_func, steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into C:\Users\Sertan\AppData\Local\Tem
p\tmpuauuy6ha\model.ckpt.
INFO:tensorflow:loss = 6.931472, step = 1
INFO:tensorflow:global_step/sec: 235.735
INFO:tensorflow:loss = 6.1775684, step = 101 (0.430 sec)
INFO:tensorflow:global_step/sec: 353.383
INFO:tensorflow:loss = 1.1133516, step = 201 (0.287 sec)
INFO:tensorflow:global_step/sec: 342.535
INFO:tensorflow:loss = 0.78092825, step = 301 (0.292 sec)
INFO:tensorflow:global_step/sec: 339.973
INFO:tensorflow:loss = 5.7018795, step = 401 (0.291 sec)
INFO:tensorflow:global_step/sec: 359.714
INFO:tensorflow:loss = 0.853746, step = 501 (0.280 sec)
INFO:tensorflow:global_step/sec: 356.763
INFO:tensorflow:loss = 3.1515746, step = 601 (0.283 sec)
INFO:tensorflow:global_step/sec: 330.079
INFO:tensorflow:loss = 2.4890015, step = 701 (0.303 sec)
INFO:tensorflow:global_step/sec: 328.031
INFO:tensorflow:loss = 3.18041, step = 801 (0.300 sec)
INFO:tensorflow:global_step/sec: 370.676
INFO:tensorflow:loss = 0.83171225, step = 901 (0.273 sec)
INFO:tensorflow:Saving checkpoints for 1000 into C:\Users\Sertan\AppData\Local
\Temp\tmpuauuy6ha\model.ckpt.
INFO:tensorflow:Loss for final step: 0.79522204.
```

Out[108]:
```
<tensorflow.python.estimator.canned.linear.LinearClassifier at 0x1940cc3e1d0>
```

In [109]:
```
eval_input_func=tf.estimator.inputs.pandas_input_fn(x=X_test, y=y_test, batch_s
```

In [110]:
```
results=model.evaluate(eval_input_func)
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; plea
se switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; plea
se switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2019-01-23-21:19:58
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\Sertan\AppData\Local\Temp\tm
puauuy6ha\model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2019-01-23-21:20:06
INFO:tensorflow:Saving dict for global step 1000: accuracy = 0.92659026, accura
cy_baseline = 0.9212278, auc = 0.6303042, auc_precision_recall = 0.28776234, av
erage_loss = 0.25509268, global_step = 1000, label/mean = 0.07877219, loss = 2.
5509267, precision = 0.93939394, prediction/mean = 0.08627975, recall = 0.07276
9955
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 1000: C:\Users
\Sertan\AppData\Local\Temp\tmpuauuy6ha\model.ckpt-1000
```

In [111]:
```
results
```

Out[111]:
```
{'accuracy': 0.92659026,
 'accuracy_baseline': 0.9212278,
 'auc': 0.6303042,
 'auc_precision_recall': 0.28776234,
 'average_loss': 0.25509268,
 'label/mean': 0.07877219,
 'loss': 2.5509267,
 'precision': 0.93939394,
 'prediction/mean': 0.08627975,
 'recall': 0.072769955,
 'global_step': 1000}
```

In [112]:
```
pred_input_func=tf.estimator.inputs.pandas_input_fn(x=X_test, batch_size=10, nu
```

In [113]:
```
predictions=model.predict(pred_input_func)
```

In [114]:
```
my_pred=list(predictions)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\Sertan\AppData\Local\Temp\tm
puauuy6ha\model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

In [115]:
```
# Now, let's change the classifier as Deeply Connected Neurel Network
dnn_model=tf.estimator.DNNClassifier(hidden_units=[8, 8], feature_columns=feat_
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\Sertan\A
ppData\Local\Temp\tmpadxh36so
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Users\\Sertan\\AppData\\Local
\\Temp\\tmpadxh36so', '_tf_random_seed': None, '_save_summary_steps': 100, '_sa
ve_checkpoints_steps': None, '_save_checkpoints_secs': 600, '_session_config':
allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_ste
p_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_protoco
l': None, '_eval_distribute': None, '_experimental_distribute': None, '_servic
e': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec o
bject at 0x00000194127D1400>, '_task_type': 'worker', '_task_id': 0, '_global_i
d_in_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': True,
'_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

In [116]:
```
dnn_model.train(input_fn=input_func, steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into C:\Users\Sertan\AppData\Local\Tem
p\tmpadxh36so\model.ckpt.
INFO:tensorflow:loss = 6.5674324, step = 1
INFO:tensorflow:global_step/sec: 246.064
INFO:tensorflow:loss = 0.3429582, step = 101 (0.409 sec)
INFO:tensorflow:global_step/sec: 356.841
INFO:tensorflow:loss = 3.236828, step = 201 (0.288 sec)
INFO:tensorflow:global_step/sec: 342.723
INFO:tensorflow:loss = 0.7914312, step = 301 (0.290 sec)
INFO:tensorflow:global_step/sec: 347.778
INFO:tensorflow:loss = 0.6893047, step = 401 (0.289 sec)
INFO:tensorflow:global_step/sec: 309.894
INFO:tensorflow:loss = 1.1385949, step = 501 (0.326 sec)
INFO:tensorflow:global_step/sec: 351.211
INFO:tensorflow:loss = 0.95407104, step = 601 (0.283 sec)
INFO:tensorflow:global_step/sec: 353.744
INFO:tensorflow:loss = 3.2633197, step = 701 (0.281 sec)
INFO:tensorflow:global_step/sec: 328.181
INFO:tensorflow:loss = 0.55023897, step = 801 (0.303 sec)
INFO:tensorflow:global_step/sec: 325.845
INFO:tensorflow:loss = 2.4902527, step = 901 (0.310 sec)
INFO:tensorflow:Saving checkpoints for 1000 into C:\Users\Sertan\AppData\Local
\Temp\tmpadxh36so\model.ckpt.
INFO:tensorflow:Loss for final step: 1.3085368.
```

Out[116]:
```
<tensorflow.python.estimator.canned.dnn.DNNClassifier at 0x1940fca2d68>
```

In [119]:
```
# DNN predictions:
predictions2=dnn_model.predict(pred_input_func)
```

In [120]:
```
# mydnn_predict
mydnn_pred=list(predictions2)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\Sertan\AppData\Local\Temp\tm
padxh36so\model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

In [121]:
```
y_pred_deepnn=[pred['class_ids'][0] for pred in mydnn_pred]
```

In [122]:
```
# Generate confusion matrix and accuracy
deepnn_classifier = confusion_matrix(y_test,y_pred_deepnn)
print(deepnn_classifier,end='\n\n')
numerator = deepnn_classifier[0][0] + deepnn_classifier[1][1]
denominator = sum(deepnn_classifier[0]) + sum(deepnn_classifier[1])
acc_dnn = (numerator/denominator) * 100
print("Accuracy : ",round(acc_dnn,4),"%")
```

```
[[24910     0]
 [ 2130     0]]

Accuracy :  92.1228 %
```

In [123]:
```
## Since data set is very unbalanced, let's check Cohen's Kappa Score
cohen_score = cohen_kappa_score(y_test,y_pred_xgb)
round(cohen_score,4)
```

Out[123]:
```
0.7574
```

We have resolved unbalanced data problem by oversampling positive data. ## Thus, this is what we get as a result:

##

Classifier: Accuracy: Cross validation accuracy: Cohen's Kappa Score:

SVM 93.4948% 93.3746% 0.3157

Random Forest Model 99.963% 99.9695% 0.9975

XGBoost Model 96.8824% 96.9148% 0.7574

Naive Base Model 93.8572% 93.7824% 0.4353

k-Nearest Neighbor 99.8003% 99.8105% 0.9864

TF Estimator - DNN 92.7959% 0.7606

As you can see the table above, Random Forest Model has not only best classification accuracy, but also it has higher Cohen's Kappa Score (close to 1) which shows perfect agreement with our data set. Therefore, in the coding part, I will be using Random Forest Model in the class package.

2nd best is K-Nearest Neighbor is classifier

In [ ]:

```

```