**from** **xgboost** **import** XGBClassifier

**from** **sklearn.neural\_network** **import** MLPClassifier

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.metrics** **import** precision\_score

**from** **sklearn.metrics** **import** recall\_score

**from** **sklearn.metrics** **import** roc\_auc\_score

**from** **sklearn.metrics** **import** f1\_score

**import** **pandas** **as** **pd**

**from** **sklearn** **import** preprocessing

**from** **xgboost** **import** plot\_importance

**import** **numpy** **as** **np**

**from** **sklearn.metrics** **import** confusion\_matrix

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.preprocessing** **import** MinMaxScaler

pd.set\_option('display.max\_columns', 500)

pd.set\_option('display.max\_rows', 500)

df = pd.read\_csv("D:**\\**analyticsvidhyahackathon**\\**train\_LZdllcl.csv")

df.set\_index('employee\_id',inplace=**True**)

dfvalidation = pd.read\_csv("D:**\\**analyticsvidhyahackathon**\\**test\_2umaH9m.csv")

dfvalidation.set\_index('employee\_id',inplace=**True**)

dfvalidation['is\_promoted'] = 5

*#dfvalidation.head()*

dfmerged = pd.concat([df,dfvalidation])

dfmerged.shape

df['previous\_year\_rating'].describe()

df['education'].describe()

*## Filling missing enteris with maximum occuring event*

dfmerged['previous\_year\_rating'].fillna(3.0, inplace=**True**)

dfmerged['education'].fillna('Bachelor**\'**s', inplace=**True**)

*##One hot encoding*

dfmerged = pd.concat([dfmerged[['no\_of\_trainings','age','previous\_year\_rating','length\_of\_service','KPIs\_met >80%','awards\_won?','avg\_training\_score','is\_promoted']],

pd.get\_dummies(dfmerged['gender'],drop\_first = **True**),pd.get\_dummies(dfmerged['education'],drop\_first = **True**),pd.get\_dummies(dfmerged['recruitment\_channel'],drop\_first = **True**),pd.get\_dummies(dfmerged['department'],drop\_first = **True**),pd.get\_dummies(dfmerged['region'],drop\_first = **True**)],axis=1)

*###Generating all possible pair of interactions between 2 pair of columns.*

*##Then removing those columns*

**from** **itertools** **import** combinations

**from** **sklearn.preprocessing** **import** PolynomialFeatures

Y = dfmerged.is\_promoted

X = dfmerged.drop(['is\_promoted'],1)

**def** add\_interactions(df):

combos = list(combinations(list(df.columns), 2))

colnames = list(df.columns)+['\_'.join(x) **for** x **in** combos]

*#scaler = MinMaxScaler()*

*#scaler.fit(df)*

*#df = scaler.transform(df)*

poly = PolynomialFeatures(interaction\_only=**True**, include\_bias=**False**)

df = poly.fit\_transform(df)

df = pd.DataFrame(df)

df.columns = colnames

noint\_indices = [i **for** i,x **in** enumerate(list((df==0).all())) **if** x]

df= df.drop(df.columns[noint\_indices], axis=1)

**return** df

X = add\_interactions(X)

X.shape

*###Seperate out the training and testing dataset*

X.index = Y.index

X['is\_promoted'] = Y

X\_validation = X[54808:]

X = X[0:54808]

X\_validation.shape

Y = X.is\_promoted

X = X.drop(['is\_promoted'],1)

seed = 2

test\_size = 0.2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=test\_size, random\_state=seed)

*###Hypertune the model intensively with a 5 fold cross validation strategy.*

*##first grid search for max\_depth,min\_child\_weight then fix those and search for rest in same fashion*

*### parameter scale\_pos\_weight is quite important in case of imbalanced dataset*

**from** **sklearn.model\_selection** **import** GridSearchCV

param\_test2b = {

*#'min\_child\_weight':[5,6],*

*#'max\_depth': range(3,10,2),*

*#'n\_estimators':[150,200,300,400],*

*#'scale\_pos\_weight':[1,2,3,4],*

*#'colsample\_bytree':[0.7,0.8],*

*#'subsample':[0.7,0.8],*

*#'gamma':[0,0.2.0.4]*

}

gsearch2b = GridSearchCV(estimator = XGBClassifier( learning\_rate=0.1, n\_estimators=150, max\_depth=5,

min\_child\_weight=1, gamma=0, subsample=0.8, colsample\_bytree=0.8,

objective= 'binary:logistic', nthread=4, scale\_pos\_weight=3,seed=27),

param\_grid = param\_test2b, scoring='f1',n\_jobs=4,iid=**False**, cv=5)

gsearch2b.fit(X\_train, y\_train)

print(gsearch2b.grid\_scores\_)

print("gsearch2b.best\_params\_",gsearch2b.best\_params\_)

print("gsearch2b.best\_score\_",gsearch2b.best\_score\_)

*###Train the model with the best params*

modelXg = XGBClassifier(learning\_rate=0.1, n\_estimators=200, max\_depth=4, min\_child\_weight=7,

gamma=0.4,nthread=4, subsample=0.8, colsample\_bytree=0.8, objective= 'binary:logistic',scale\_pos\_weight=3,seed=29)

modelXg.fit(X\_train, y\_train)

y\_xg = modelXg.predict(X\_test)

*## Determine whether your model is overfitting or not , with the help of ROC.*

print(confusion\_matrix(y\_test, y\_xg))

predictions = [value **for** value **in** y\_xg]

accuracy = accuracy\_score(y\_test, predictions)

precision = precision\_score(y\_test, predictions)

recall = recall\_score(y\_test, predictions)

f1 = f1\_score(y\_test, predictions)

print("Accuracy\_score: **%.2f%%** on test dataset" % (accuracy \* 100.0))

print("precision\_score: **%.2f%%** on test dataset" % (precision \* 100.0))

print("recall\_score: **%.2f%%** on test dataset" % (recall \* 100.0))

print("f1\_score: **%.2f%%** on test dataset" % (f1 \* 100.0))

print("roc\_auc test set", roc\_auc\_score(y\_test, model.predict\_proba(X\_test)[:,1]))

print("roc\_auc training set", roc\_auc\_score(y\_train, model.predict\_proba(X\_train)[:,1]))

*##Once the model is ready, train the model on entire dataset.*

modelXg = XGBClassifier(learning\_rate=0.1, n\_estimators=200, max\_depth=4, min\_child\_weight=7,

gamma=0.4,nthread=4, subsample=0.8, colsample\_bytree=0.8, objective= 'binary:logistic',scale\_pos\_weight=3,seed=29)

modelXg.fit(X, Y)

X = X\_validation.drop(['is\_promoted'],1)

dfvalidation = X\_validation.drop(['is\_promoted'],1)

y\_valid = modelXg.predict(dfvalidation)

submission = dfvalidation.copy()

submission['is\_promoted'] = y\_valid

submission['is\_promoted'].value\_counts()

submission = submission[['is\_promoted']]

submission.reset\_index(inplace=**True**)

submission.to\_csv("D:**\\**analyticsvidhyahackathon**\\**solution.csv",index=**False**)s

sub = pd.read\_csv("solution\_18sept.csv")

private = pd.read\_csv("private\_W1BajhK.csv")

private.columns = ['employee\_id', 'actual']

private = private.merge(sub, on = "employee\_id", how="left")

**from** **sklearn.metrics** **import** f1\_score

f1\_score(private['actual'], private['is\_promoted'])