Home Credit Default Risk Using BigMl

In [204]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set(style = "whitegrid",
        color_codes = True,
        font_scale = 1.5)
```

In [205]:
```
#Save our BigMl Username and Api Key to our environment to access the AP
I
import os
os.environ['BIGML_USERNAME'] = "efetoros"
os.environ['BIGML_API_KEY'] = "471ae5485d74ceeb2e911e0c1d37edda58cf79d3"
```

In [206]:
```
#Access our project in our BigMl account in odrer to keep track of our r
esources created
from bigml.api import BigML
API = BigML(project="project/5b17b75c92fb560173000387")
```

In [207]:
```
#Download each CSV provided into a pandas DataFrame
application_train = pd.read_csv("application_train.csv")
application_test = pd.read_csv("application_test.csv")
bureau = pd.read_csv("bureau.csv")
bureau_balance = pd.read_csv("bureau_balance.csv")
credit_card_balance = pd.read_csv("credit_card_balance.csv")
installments_payments = pd.read_csv("installments_payments.csv")
previous_application = pd.read_csv("previous_application.csv")
POS_CASH_balance = pd.read_csv("POS_CASH_balance.csv")
```

In [217]:
```
#This chart enables us to see the common IDs we will need to know for me
rging
```

home_credit.png

In [208]:
```
#Turn any catergorical variables into dummy variables for each DataFrame
application_test_dummies = pd.get_dummies(application_test)

application_train_dummies = pd.get_dummies(application_train)

full_bureau = bureau.merge(bureau_balance, on= "SK_ID_BUREAU")
full_bureau_dummies = pd.get_dummies(full_bureau)

previous_application_dummies = pd.get_dummies(previous_application)

credit_card_balance_dummies = pd.get_dummies(credit_card_balance)

installments_payments_dummies = pd.get_dummies(installments_payments)

POS_CASH_balance_dummies = pd.get_dummies(POS_CASH_balance)
```

In [209]:
```
#Groupby SK_ID_CURR on each DataDrame using the function mean.
application_test_avg = application_test_dummies.groupby("SK_ID_CURR").me
an()

app_train_avg = application_train_dummies.groupby("SK_ID_CURR").mean()

full_bureau_avg = full_bureau_dummies.groupby("SK_ID_CURR").mean()

previous_application_avg=previous_application_dummies.groupby("SK_ID_CUR
R").mean()

credit_card_balance_avg = credit_card_balance_dummies.groupby("SK_ID_CUR
R").mean()

installments_payments_avg = installments_payments_dummies.groupby("SK_ID
_CURR").mean()

POS_CASH_balance_avg = POS_CASH_balance_dummies.groupby("SK_ID_CURR").me
an()
```

In [210]:
```
from functools import reduce
```

In [211]:
```
# Merge all tables that will be used for training on their indices creat
ed by the Groupby
dfs = [app_train_avg,previous_application_avg,full_bureau_avg, credit_ca
rd_balance_avg, installments_payments_avg, POS_CASH_balance]
df_final = reduce(lambda left,right: pd.merge(left,right,left_index=True
, right_index=True, how="left"), dfs)
```

In [212]:
```
# Merge all tables that will be used for our final batch prediction on t
heir indices created by the Groupby
dfs = [application_test_avg,previous_application_avg,full_bureau_avg, cr
edit_card_balance_avg, installments_payments_avg, POS_CASH_balance]
df_test_final = reduce(lambda left,right: pd.merge(left,right,left_index
=True, right_index=True, how="left"), dfs)
```

In [213]:
```
#Create BigMl sources from API
source = API.create_source("Avg_info_deafult_loans.csv")
test_source = API.create_source("Modified_Test_Set.csv")

full_dataset = API.create_dataset(source)
full_test_dataset = API.create_dataset(test_source)

#Wait to see if Datasets are created
API.ok(full_dataset)
API.ok(full_test_dataset)
```

In [215]:
```
#Create a test train split
train_dataset = API.create_dataset(
    full_dataset, {"name": "Dataset Name | Training",
                   "sample_rate": 0.8, "seed": "my seed"})
train_test_dataset = API.create_dataset(
    full_dataset, {"name": "Dataset Name | Test",
                   "sample_rate": 0.8, "seed": "my seed",
                   "out_of_bag": True})

#Wait to see if Datasets are created
API.ok(train_dataset)
API.ok(train_test_dataset)
```

Out[215]: True

In [188]:
```
#Create a logstic regression model using BigMl
logistic_regression = API.create_logistic_regression(train_dataset, {"na
me": "credit_default_logistic regression",
                                                                          "objective_fiel
d": "TARGET"})
#Wait till the model is created
API.ok(logistic_regression)
```

Out[188]: True

In [216]:
```
#Create an evaluation
evaluation = API.create_evaluation(logistic_regression, train_test_datas
et)
API.ok(evaluation)
```

Out[216]: True

As seen in the BigMl Interface below, at first glance it appears if our model did amazing since the accuracy is 91.9%. However, looking into the confusion matrix, we can see that the false cases had a very low recall percentage, implying that our accuracy is only high because there is an unbalance between the positive and negative cases.

info.png

In order to account for this unbalance, we will have to set find probability threshold that will create a more appropriate model. This threshold can be often times set to the Max. phi coefficient.

Screen%20Shot%202018-06-08%20at%2010.18.31%20AM.png

In [200]:
```
#Create final batch prediction with the arguments that you prefer, a ful
l list of arguments to choose form
#can be found on BigMl's Api manual
batch_prediction = API.create_batch_prediction(logistic_regression, full
_test_dataset, {
    "name": "my batch prediction", "all_fields": True,
    "header": True,
    "confidence": True,
"operating_point": {
    "kind": "probability",
    "positive_class": "0",
    "threshold": 0.83
}})
API.ok(batch_prediction)
```

Out[200]: True

In [202]:
```
#This small workflow comes out with a .703 prediction.
#More improvement can be made by further feature engineering and finding
 the optimal probability threshold.
API.download_batch_prediction(batch_prediction,
    filename='my_prediction.csv')
```

Out[202]: 'my_prediction.csv'