Credit Card Approval Prediction Using Sklearn

Table of Content

In [1]:
```python
# %matplotlib inline
# %config InlineBackend.figure_format = 'svg'

import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
```

```
Using TensorFlow backend.
```

In [2]:
```python
# plt.rcParams['figure.facecolor'] = 'white'
```

Binary Features

In [3]:
```
# Calculate information value
def calc_iv(df, feature, target, pr=False):
    lst = []
    df[feature] = df[feature].fillna("NULL")

    for i in range(df[feature].nunique()):
        val = list(df[feature].unique())[i]
        lst.append([feature,                                                # Variable
                    val,                                                    # Value
                    df[df[feature] == val].count()[feature],                # All
                    df[(df[feature] == val) & (df[target] == 0)].count()[feature],  # Good (think: Fraud == 0)
                    df[(df[feature] == val) & (df[target] == 1)].count()[feature]]) # Bad (think: Fraud == 1)

    data = pd.DataFrame(lst, columns=['Variable', 'Value', 'All', 'Good', 'Bad'])
    data['Share'] = data['All'] / data['All'].sum()
    data['Bad Rate'] = data['Bad'] / data['All']
    data['Distribution Good'] = (data['All'] - data['Bad']) / (data['All'].sum() - data['Bad'].sum())
    data['Distribution Bad'] = data['Bad'] / data['Bad'].sum()
    data['WoE'] = np.log(data['Distribution Good'] / data['Distribution Bad'])

    data = data.replace({'WoE': {np.inf: 0, -np.inf: 0}})

    data['IV'] = data['WoE'] * (data['Distribution Good'] - data['Distribution Bad'])

    data = data.sort_values(by=['Variable', 'Value'], ascending=[True, True])
    data.index = range(len(data.index))

    if pr:
        print(data)
        print('IV = ', data['IV'].sum())

    iv = data['IV'].sum()
    print('This variable\'s IV is:',iv)
    print(df[feature].value_counts())
    return iv, data

def convert_dummy(df, feature,rank=0):
    pos = pd.get_dummies(df[feature], prefix=feature)
    mode = df[feature].value_counts().index[rank]
    biggest = feature + '_' + str(mode)
    pos.drop([biggest],axis=1,inplace=True)
```

```
    df.drop([feature],axis=1,inplace=True)
    df=df.join(pos)
    return df

def get_category(df, col, binsnum, labels, qcut = False):
    if qcut:
        localdf = pd.qcut(df[col], q = binsnum, labels = labels) # quantile cut
    else:
        localdf = pd.cut(df[col], bins = binsnum, labels = labels) # equal-length cut

    localdf = pd.DataFrame(localdf)
    name = 'gp' + '_' + col
    localdf[name] = localdf[col]
    df = df.join(localdf[name])
    df[name] = df[name].astype(object)
    return df

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
```

```
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Feature Engineering

In [4]:
```
def return_no(x):
    if x==None:
        return "No"

def data_preprocessing(data):

    ## Feature Engineering
    # find all users' account open month.
    begin_month=pd.DataFrame(data.groupby(["ID"])["Vintage"].agg(min))
    begin_month=begin_month.rename(columns={'Vintage':'begin_month'})
    new_data=pd.merge(data,begin_month,how="left",on="ID") #merge to record data

    del begin_month

    ##Generally, users in risk should be in 3%, thus I choose users who overdue for more than 60 days as target risk u
sers. Those samples are marked as '1', else are '0'.
    new_data['target']=data['Is_Active']
    new_data.loc[new_data['target']=='Yes','target']=1
    new_data.loc[new_data['target']=='No','target']=0
    print(new_data['target'].value_counts())
    new_data['target'].value_counts(normalize=True)

    #features
#     new_data.dropna()
#     new_data = new_data.mask(new_data == 'NULL').dropna()

    ivtable=pd.DataFrame(new_data.columns,columns=['variable'])
    ivtable['IV']=None


#     for col in columns:
#         try:
# #             new_data[col].replace(np.NaN, new_data[col].mean())
#             new_data[col].replace(None, new_data[col].mean())
#         except:pass

    #No      144357, Yes      72043
    new_data['Credit_Product'] = new_data['Credit_Product'].apply(lambda x: return_no(x))


    from sklearn.preprocessing import LabelEncoder
```

```
    label_encoder = LabelEncoder()

    l_encoder = label_encoder.fit(new_data['Gender'])
    new_data['Gender'] = l_encoder.transform(new_data['Gender'])

    l_encoder = label_encoder.fit(new_data['Region_Code'])
    new_data['Region_Code'] = l_encoder.transform(new_data['Region_Code'])

    l_encoder = label_encoder.fit(new_data['Occupation'])
    new_data['Occupation'] = l_encoder.transform(new_data['Occupation'])

    l_encoder = label_encoder.fit(new_data['Channel_Code'])
    new_data['Channel_Code'] = l_encoder.transform(new_data['Channel_Code'])

    l_encoder = label_encoder.fit(new_data['Credit_Product'])
    new_data['Credit_Product'] = l_encoder.transform(new_data['Credit_Product'])

#     l_encoder = label_encoder.fit(new_data['Is_Active'])
#     new_data['Is_Active'] = l_encoder.transform(new_data['Is_Active'])

    del label_encoder, l_encoder


#     print(data['Is_Active'].value_counts())
#     data['Is_Active'].value_counts(normalize=True)

    #Gender
    print(new_data['Gender'].value_counts())
    iv, data = calc_iv(new_data,'Gender','target')
    ivtable.loc[ivtable['variable']=='Gender','IV']=iv

    #Avg_Account_Balance
#     new_data['Avg_Account_Balance']=new_data['Avg_Account_Balance'].astype(object)
#     new_data['Avg_Account_Balance'] = new_data['Avg_Account_Balance']/10000
    new_data['Avg_Account_Balance'] = new_data['Avg_Account_Balance'].apply(lambda x: x/10000)

    print(new_data['Avg_Account_Balance'].value_counts(bins=10,sort=False))
    new_data['Avg_Account_Balance'].plot(kind='hist',bins=50,density=True)

    return new_data
```

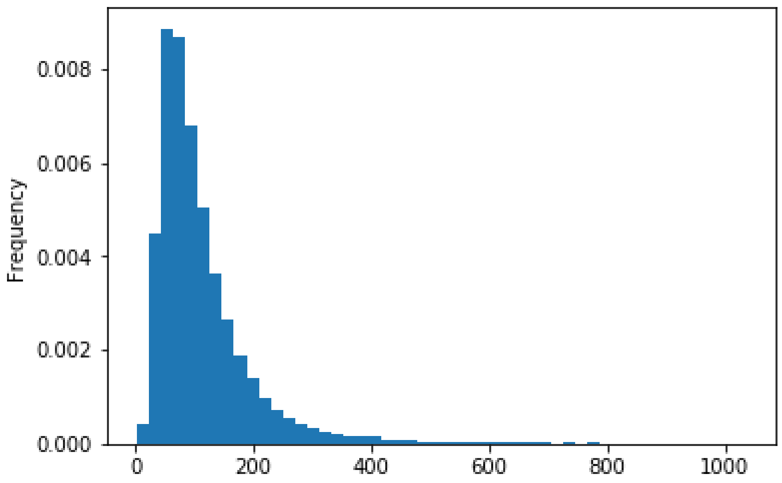
In [5]:
```
new_data = data_preprocessing(pd.read_csv("../input/jobathon-may-2021/train.csv", encoding = 'utf-8') )

print(new_data.columns)
new_data.head()
```

```
0    150290
1     95435
Name: target, dtype: int64
1    134197
0    111528
Name: Gender, dtype: int64
This variable's IV is: 0.017432813544O5051
1    134197
0    111528
Name: Gender, dtype: int64
(1.0450000000000002, 105.391]    148577
(105.391, 208.703]                73815
(208.703, 312.016]                15340
(312.016, 415.328]                 4834
(415.328, 518.64]                  1585
(518.64, 621.952]                   738
(621.952, 725.264]                  418
(725.264, 828.577]                  282
(828.577, 931.889]                  125
(931.889, 1035.201]                  11
Name: Avg_Account_Balance, dtype: int64
Index(['ID', 'Gender', 'Age', 'Region_Code', 'Occupation', 'Channel_Code',
       'Vintage', 'Credit_Product', 'Avg_Account_Balance', 'Is_Active',
       'Is_Lead', 'begin_month', 'target'],
      dtype='object')
```

Out[5]:

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead	begin
0	NNVBBKZB	0	73	18	1	2	43	0	104.5696	No	0	
1	IDD62UNG	0	30	27	2	0	32	0	58.1988	No	0	
2	HD3DSEMC	0	56	18	3	2	26	0	148.4315	Yes	0	
3	BF3NC7KV	1	34	20	2	0	19	0	47.0454	No	0	
4	TEASRWXV	0	30	32	2	0	33	0	88.6787	No	0	

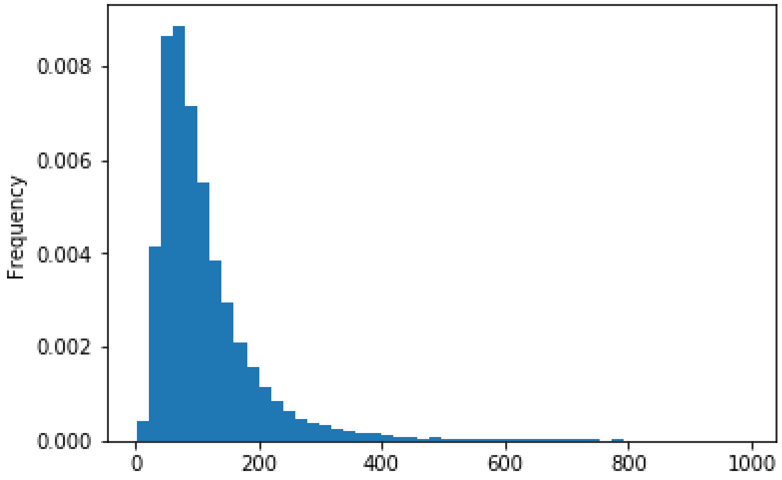

Frequency
0.008
0.006
0.004
0.002
0.000
0
200
400
600
800
1000

In [6]:
```
test_data = data_preprocessing(pd.read_csv("../input/jobathon-may-2021/test.csv", encoding = 'utf-8') )
test_data.head()
```

```
0    63797
1    41515
Name: target, dtype: int64
1    57705
0    47607
Name: Gender, dtype: int64
This variable's IV is: 0.020052086601170444
1    57705
0    47607
Name: Gender, dtype: int64
(1.27, 101.122]       60798
(101.122, 199.985]    33196
(199.985, 298.848]     7263
(298.848, 397.71]      2343
(397.71, 496.573]       872
(496.573, 595.435]      396
(595.435, 694.298]      201
(694.298, 793.161]      152
(793.161, 892.023]       63
(892.023, 990.886]       28
Name: Avg_Account_Balance, dtype: int64
```

Out[6]:

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	begin_month
0	VBENBARO	1	29	4	1	0	25	0	74.2366	No	25
1	CCMEWNKY	1	43	18	1	1	49	0	92.5537	No	49
2	VK3KGA9M	1	31	20	2	0	14	0	21.5949	No	14
3	TT8RPZVC	1	29	22	1	0	33	0	86.8070	No	33
4	SHQZEYTZ	0	29	20	1	0	19	0	65.7087	No	19



Algorithms

In [7]:
```
Y = new_data['Is_Lead']
X = new_data[['Gender', 'Age', 'Region_Code', 'Occupation', 'Channel_Code',
    'Credit_Product', 'Avg_Account_Balance', 'target', 'begin_month']]
X.head()
```

Out[7]:

	Gender	Age	Region_Code	Occupation	Channel_Code	Credit_Product	Avg_Account_Balance	target	begin_month
0	0	73	18	1	2	0	104.5696	0	43
1	0	30	27	2	0	0	58.1988	0	32
2	0	56	18	3	2	0	148.4315	1	26
3	1	34	20	2	0	0	47.0454	0	19
4	0	30	32	2	0	0	88.6787	0	33

- Using Synthetic Minority Over-Sampling Technique(`SMOTE`) to overcome sample imbalance problem.

In [8]:
```
Y = Y.astype('int')
X_balance,Y_balance = SMOTE().fit_sample(X,Y)
X_balance = pd.DataFrame(X_balance, columns = X.columns)
```

- After over sampling, the number between 1 and 0 is balanced. It can be seen from the confusion matrix.

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X_balance,Y_balance, test_size=0.2,
                                                    random_state = 10086)
print(X_train.size, X_test.size, y_train.size, y_test.size)
```

```
2699091 674775 299899 74975
```

Logistic Regression

$$\log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_q x_q$$

In [10]:
```
# model = LogisticRegression(C=0.8,
#                            random_state=0,
#                            solver='lbfgs')
# model.fit(X_train, y_train)
# y_predict = model.predict(X_test)

# print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

# sns.set_style('white')
# class_names = ['0','1']
# plot_confusion_matrix(confusion_matrix(y_test,y_predict),
#                       classes= class_names, normalize = True,
#                       title='Normalized Confusion Matrix: Logistic Regression')
```

Decision Tree

In [11]:
```
# model = DecisionTreeClassifier(max_depth=12,
#                                 min_samples_split=8,
#                                 random_state=1024)
# model.fit(X_train, y_train)
# y_predict = model.predict(X_test)

# print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

# plot_confusion_matrix(confusion_matrix(y_test,y_predict),
#                       classes=class_names, normalize = True,
#                       title='Normalized Confusion Matrix: CART')
```

Random Forest

In [12]:
```
# model = RandomForestClassifier(n_estimators=250,
#                                max_depth=12,
#                                min_samples_leaf=16
#                                )
# model.fit(X_train, y_train)
# y_predict = model.predict(X_test)

# print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

# plot_confusion_matrix(confusion_matrix(y_test,y_predict),
#                       classes=class_names, normalize = True,
#                       title='Normalized Confusion Matrix: Ramdom Forests')
```

SVM

In [13]:
```
# model = svm.SVC(C = 0.8,
#                 kernel='linear')
# model.fit(X_train, y_train)
# y_predict = model.predict(X_test)

# print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

# plot_confusion_matrix(confusion_matrix(y_test,y_predict),
#                       classes=class_names, normalize = True,
#                       title='Normalized Confusion Matrix: SVM')
```

LightGBM

In [14]:
```
# model = LGBMClassifier(num_leaves=31,
#                        max_depth=8,
#                        learning_rate=0.02,
#                        n_estimators=250,
#                        subsample = 0.8,
#                        colsample_bytree =0.8
#                       )
# model.fit(X_train, y_train)
# y_predict = model.predict(X_test)
# print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# print(pd.DataFrame(confusion_matrix(y_test,y_predict)))
```

In [15]:
```
# submisson = pd.DataFrame()

# submisson['ID'] = test_data['ID']
# test_data_temp = test_data[['Gender', 'Age', 'Region_Code', 'Occupation', 'Channel_Code',
#        'Credit_Product', 'Avg_Account_Balance', 'Is_Active', 'begin_month', 'target']]

# submisson['Is_Lead'] = model.predict(test_data_temp)
# submisson.to_csv('sample_submission.csv')
# submisson.head()
```

Showing important features:

In [16]:
```
def plot_importance(classifer, x_train, point_size = 25):
    '''plot feature importance'''
    values = sorted(zip(x_train.columns, classifer.feature_importances_), key = lambda x: x[1] * -1)
    imp = pd.DataFrame(values,columns = ["Name", "Score"])
    imp.sort_values(by = 'Score',inplace = True)
    sns.scatterplot(x = 'Score',y='Name', linewidth = 0,
                data = imp,s = point_size, color='red').set(
    xlabel='importance',
    ylabel='features')

# plot_importance(model, X_train,20)
```

In [17]:
```
# print(model.booster_.feature_importance(importance_type='gain'))
```

Xgboost

In [18]:
```
model = XGBClassifier(max_depth=12,
                      n_estimators=400,
                      min_child_weight=8,
                      subsample=0.8,
                      learning_rate =0.01,
                      seed=42)

eval_set = [(X_test, y_test)]
model.fit(X_train, y_train, early_stopping_rounds=25, eval_metric="auc", eval_set=eval_set, verbose=True)

y_predict = model.predict(X_test)
print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

plot_importance(model, X_train, 20)
```

```
[0]	validation_0-auc:0.906367
Will train until validation_0-auc hasn't improved in 25 rounds.
[1]	validation_0-auc:0.908923
[2]	validation_0-auc:0.909655
[3]	validation_0-auc:0.909875
[4]	validation_0-auc:0.910589
[5]	validation_0-auc:0.910542
[6]	validation_0-auc:0.910711
[7]	validation_0-auc:0.910832
[8]	validation_0-auc:0.911081
[9]	validation_0-auc:0.911471
[10]	validation_0-auc:0.911584
[11]	validation_0-auc:0.911648
[12]	validation_0-auc:0.911733
[13]	validation_0-auc:0.91202
[14]	validation_0-auc:0.911995
[15]	validation_0-auc:0.911937
[16]	validation_0-auc:0.912135
[17]	validation_0-auc:0.9122
[18]	validation_0-auc:0.912219
[19]	validation_0-auc:0.912248
[20]	validation_0-auc:0.912326
[21]	validation_0-auc:0.912389
[22]	validation_0-auc:0.912448
[23]	validation_0-auc:0.912509
[24]	validation_0-auc:0.912508
[25]	validation_0-auc:0.912541
[26]	validation_0-auc:0.912561
[27]	validation_0-auc:0.912595
[28]	validation_0-auc:0.912642
[29]	validation_0-auc:0.912702
[30]	validation_0-auc:0.912771
[31]	validation_0-auc:0.912855
[32]	validation_0-auc:0.912985
[33]	validation_0-auc:0.913005
[34]	validation_0-auc:0.913002
[35]	validation_0-auc:0.913024
[36]	validation_0-auc:0.913085
[37]	validation_0-auc:0.913128
[38]	validation_0-auc:0.913202
[39]	validation_0-auc:0.913249
```

```
[40]	validation_0-auc:0.913274
[41]	validation_0-auc:0.913274
[42]	validation_0-auc:0.913506
[43]	validation_0-auc:0.913564
[44]	validation_0-auc:0.913604
[45]	validation_0-auc:0.913708
[46]	validation_0-auc:0.913734
[47]	validation_0-auc:0.913774
[48]	validation_0-auc:0.913801
[49]	validation_0-auc:0.913878
[50]	validation_0-auc:0.913917
[51]	validation_0-auc:0.913964
[52]	validation_0-auc:0.914049
[53]	validation_0-auc:0.914104
[54]	validation_0-auc:0.914189
[55]	validation_0-auc:0.914241
[56]	validation_0-auc:0.914305
[57]	validation_0-auc:0.914366
[58]	validation_0-auc:0.914411
[59]	validation_0-auc:0.914484
[60]	validation_0-auc:0.914509
[61]	validation_0-auc:0.914512
[62]	validation_0-auc:0.914563
[63]	validation_0-auc:0.914595
[64]	validation_0-auc:0.914613
[65]	validation_0-auc:0.91465
[66]	validation_0-auc:0.914723
[67]	validation_0-auc:0.914765
[68]	validation_0-auc:0.914801
[69]	validation_0-auc:0.914901
[70]	validation_0-auc:0.914941
[71]	validation_0-auc:0.914963
[72]	validation_0-auc:0.915005
[73]	validation_0-auc:0.915153
[74]	validation_0-auc:0.915283
[75]	validation_0-auc:0.915298
[76]	validation_0-auc:0.915339
[77]	validation_0-auc:0.91538
[78]	validation_0-auc:0.91541
[79]	validation_0-auc:0.915421
[80]	validation_0-auc:0.915443
[81]	validation_0-auc:0.91546
```

```
[82]	validation_0-auc:0.915518
[83]	validation_0-auc:0.915649
[84]	validation_0-auc:0.915686
[85]	validation_0-auc:0.91572
[86]	validation_0-auc:0.915742
[87]	validation_0-auc:0.91582
[88]	validation_0-auc:0.915834
[89]	validation_0-auc:0.915856
[90]	validation_0-auc:0.915875
[91]	validation_0-auc:0.915897
[92]	validation_0-auc:0.916027
[93]	validation_0-auc:0.916046
[94]	validation_0-auc:0.916074
[95]	validation_0-auc:0.916088
[96]	validation_0-auc:0.916226
[97]	validation_0-auc:0.916303
[98]	validation_0-auc:0.916331
[99]	validation_0-auc:0.916402
[100]	validation_0-auc:0.916519
[101]	validation_0-auc:0.916571
[102]	validation_0-auc:0.916642
[103]	validation_0-auc:0.916707
[104]	validation_0-auc:0.916732
[105]	validation_0-auc:0.916795
[106]	validation_0-auc:0.916852
[107]	validation_0-auc:0.916887
[108]	validation_0-auc:0.91697
[109]	validation_0-auc:0.917024
[110]	validation_0-auc:0.917112
[111]	validation_0-auc:0.9172
[112]	validation_0-auc:0.917222
[113]	validation_0-auc:0.917267
[114]	validation_0-auc:0.917368
[115]	validation_0-auc:0.917404
[116]	validation_0-auc:0.917456
[117]	validation_0-auc:0.91755
[118]	validation_0-auc:0.917592
[119]	validation_0-auc:0.917646
[120]	validation_0-auc:0.917737
[121]	validation_0-auc:0.917758
[122]	validation_0-auc:0.917833
[123]	validation_0-auc:0.917917
```

```
[124]	validation_0-auc:0.917992
[125]	validation_0-auc:0.918056
[126]	validation_0-auc:0.918097
[127]	validation_0-auc:0.918128
[128]	validation_0-auc:0.918198
[129]	validation_0-auc:0.918244
[130]	validation_0-auc:0.918302
[131]	validation_0-auc:0.918334
[132]	validation_0-auc:0.918357
[133]	validation_0-auc:0.918453
[134]	validation_0-auc:0.918483
[135]	validation_0-auc:0.91854
[136]	validation_0-auc:0.918584
[137]	validation_0-auc:0.918625
[138]	validation_0-auc:0.918664
[139]	validation_0-auc:0.918701
[140]	validation_0-auc:0.918711
[141]	validation_0-auc:0.91876
[142]	validation_0-auc:0.9188
[143]	validation_0-auc:0.918847
[144]	validation_0-auc:0.918864
[145]	validation_0-auc:0.918887
[146]	validation_0-auc:0.918934
[147]	validation_0-auc:0.918969
[148]	validation_0-auc:0.919
[149]	validation_0-auc:0.919034
[150]	validation_0-auc:0.919059
[151]	validation_0-auc:0.919094
[152]	validation_0-auc:0.919123
[153]	validation_0-auc:0.919177
[154]	validation_0-auc:0.919193
[155]	validation_0-auc:0.91923
[156]	validation_0-auc:0.919268
[157]	validation_0-auc:0.919292
[158]	validation_0-auc:0.919364
[159]	validation_0-auc:0.919389
[160]	validation_0-auc:0.91941
[161]	validation_0-auc:0.919438
[162]	validation_0-auc:0.91946
[163]	validation_0-auc:0.919493
[164]	validation_0-auc:0.919517
[165]	validation_0-auc:0.919535
```

```
[166]	validation_0-auc:0.919552
[167]	validation_0-auc:0.919579
[168]	validation_0-auc:0.919592
[169]	validation_0-auc:0.919605
[170]	validation_0-auc:0.919629
[171]	validation_0-auc:0.919645
[172]	validation_0-auc:0.919661
[173]	validation_0-auc:0.919697
[174]	validation_0-auc:0.919727
[175]	validation_0-auc:0.919744
[176]	validation_0-auc:0.91977
[177]	validation_0-auc:0.919799
[178]	validation_0-auc:0.919833
[179]	validation_0-auc:0.919862
[180]	validation_0-auc:0.919893
[181]	validation_0-auc:0.919906
[182]	validation_0-auc:0.919938
[183]	validation_0-auc:0.919976
[184]	validation_0-auc:0.919994
[185]	validation_0-auc:0.920027
[186]	validation_0-auc:0.920066
[187]	validation_0-auc:0.920092
[188]	validation_0-auc:0.920116
[189]	validation_0-auc:0.920146
[190]	validation_0-auc:0.920171
[191]	validation_0-auc:0.920212
[192]	validation_0-auc:0.920247
[193]	validation_0-auc:0.920273
[194]	validation_0-auc:0.92029
[195]	validation_0-auc:0.92032
[196]	validation_0-auc:0.920343
[197]	validation_0-auc:0.920362
[198]	validation_0-auc:0.920401
[199]	validation_0-auc:0.92043
[200]	validation_0-auc:0.920442
[201]	validation_0-auc:0.920469
[202]	validation_0-auc:0.92054
[203]	validation_0-auc:0.920575
[204]	validation_0-auc:0.920613
[205]	validation_0-auc:0.920643
[206]	validation_0-auc:0.920682
[207]	validation_0-auc:0.920707
```

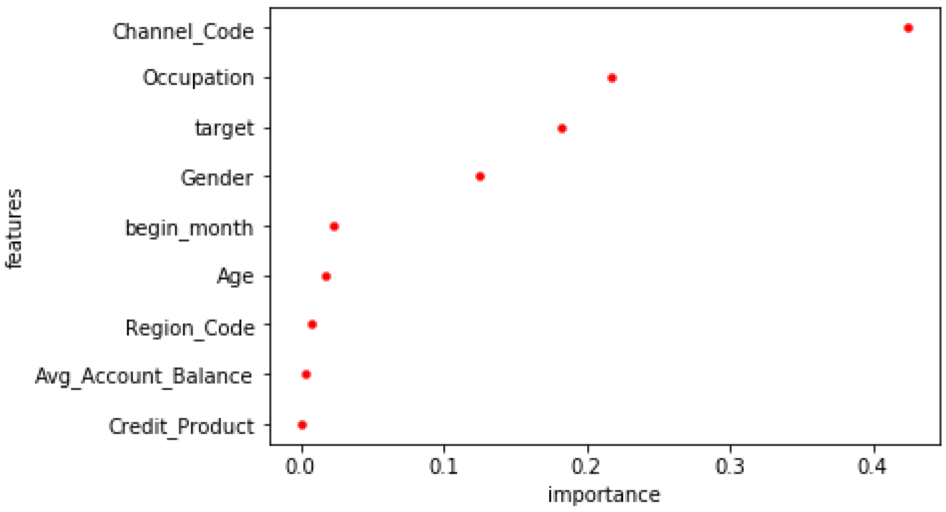
```
[208]	validation_0-auc:0.920737
[209]	validation_0-auc:0.920767
[210]	validation_0-auc:0.920806
[211]	validation_0-auc:0.920832
[212]	validation_0-auc:0.920875
[213]	validation_0-auc:0.920886
[214]	validation_0-auc:0.92092
[215]	validation_0-auc:0.920969
[216]	validation_0-auc:0.921015
[217]	validation_0-auc:0.921065
[218]	validation_0-auc:0.921125
[219]	validation_0-auc:0.921147
[220]	validation_0-auc:0.921178
[221]	validation_0-auc:0.921205
[222]	validation_0-auc:0.921243
[223]	validation_0-auc:0.921273
[224]	validation_0-auc:0.921326
[225]	validation_0-auc:0.921341
[226]	validation_0-auc:0.921358
[227]	validation_0-auc:0.921417
[228]	validation_0-auc:0.921437
[229]	validation_0-auc:0.921469
[230]	validation_0-auc:0.921522
[231]	validation_0-auc:0.921552
[232]	validation_0-auc:0.921599
[233]	validation_0-auc:0.921608
[234]	validation_0-auc:0.921654
[235]	validation_0-auc:0.921669
[236]	validation_0-auc:0.921705
[237]	validation_0-auc:0.921713
[238]	validation_0-auc:0.921726
[239]	validation_0-auc:0.921747
[240]	validation_0-auc:0.921759
[241]	validation_0-auc:0.921813
[242]	validation_0-auc:0.921834
[243]	validation_0-auc:0.921853
[244]	validation_0-auc:0.921899
[245]	validation_0-auc:0.921932
[246]	validation_0-auc:0.921979
[247]	validation_0-auc:0.92202
[248]	validation_0-auc:0.922056
[249]	validation_0-auc:0.922076
```

```
[250]	validation_0-auc:0.922093
[251]	validation_0-auc:0.922133
[252]	validation_0-auc:0.922165
[253]	validation_0-auc:0.922196
[254]	validation_0-auc:0.922238
[255]	validation_0-auc:0.922272
[256]	validation_0-auc:0.922307
[257]	validation_0-auc:0.922348
[258]	validation_0-auc:0.922399
[259]	validation_0-auc:0.922422
[260]	validation_0-auc:0.922464
[261]	validation_0-auc:0.922477
[262]	validation_0-auc:0.922525
[263]	validation_0-auc:0.922553
[264]	validation_0-auc:0.922582
[265]	validation_0-auc:0.922604
[266]	validation_0-auc:0.922641
[267]	validation_0-auc:0.922688
[268]	validation_0-auc:0.92273
[269]	validation_0-auc:0.922747
[270]	validation_0-auc:0.922763
[271]	validation_0-auc:0.922774
[272]	validation_0-auc:0.922788
[273]	validation_0-auc:0.922835
[274]	validation_0-auc:0.922855
[275]	validation_0-auc:0.922876
[276]	validation_0-auc:0.922891
[277]	validation_0-auc:0.922909
[278]	validation_0-auc:0.922949
[279]	validation_0-auc:0.922972
[280]	validation_0-auc:0.92299
[281]	validation_0-auc:0.923029
[282]	validation_0-auc:0.923042
[283]	validation_0-auc:0.923058
[284]	validation_0-auc:0.923102
[285]	validation_0-auc:0.923138
[286]	validation_0-auc:0.923181
[287]	validation_0-auc:0.9232
[288]	validation_0-auc:0.923215
[289]	validation_0-auc:0.923245
[290]	validation_0-auc:0.923287
[291]	validation_0-auc:0.92332
```

```
[292]	validation_0-auc:0.92335
[293]	validation_0-auc:0.923386
[294]	validation_0-auc:0.923433
[295]	validation_0-auc:0.923463
[296]	validation_0-auc:0.923498
[297]	validation_0-auc:0.923529
[298]	validation_0-auc:0.923553
[299]	validation_0-auc:0.923563
[300]	validation_0-auc:0.923592
[301]	validation_0-auc:0.923614
[302]	validation_0-auc:0.923631
[303]	validation_0-auc:0.923649
[304]	validation_0-auc:0.923686
[305]	validation_0-auc:0.923706
[306]	validation_0-auc:0.923725
[307]	validation_0-auc:0.923753
[308]	validation_0-auc:0.923778
[309]	validation_0-auc:0.923796
[310]	validation_0-auc:0.923824
[311]	validation_0-auc:0.923848
[312]	validation_0-auc:0.923861
[313]	validation_0-auc:0.92389
[314]	validation_0-auc:0.923925
[315]	validation_0-auc:0.923946
[316]	validation_0-auc:0.923964
[317]	validation_0-auc:0.923988
[318]	validation_0-auc:0.924016
[319]	validation_0-auc:0.924026
[320]	validation_0-auc:0.924039
[321]	validation_0-auc:0.924071
[322]	validation_0-auc:0.924088
[323]	validation_0-auc:0.924117
[324]	validation_0-auc:0.924151
[325]	validation_0-auc:0.924177
[326]	validation_0-auc:0.924205
[327]	validation_0-auc:0.924214
[328]	validation_0-auc:0.924227
[329]	validation_0-auc:0.924257
[330]	validation_0-auc:0.924287
[331]	validation_0-auc:0.924306
[332]	validation_0-auc:0.924336
[333]	validation_0-auc:0.924367
```

```
[334]	validation_0-auc:0.924394
[335]	validation_0-auc:0.924419
[336]	validation_0-auc:0.924442
[337]	validation_0-auc:0.92446
[338]	validation_0-auc:0.924494
[339]	validation_0-auc:0.924508
[340]	validation_0-auc:0.924524
[341]	validation_0-auc:0.924545
[342]	validation_0-auc:0.924575
[343]	validation_0-auc:0.924601
[344]	validation_0-auc:0.924622
[345]	validation_0-auc:0.924648
[346]	validation_0-auc:0.924671
[347]	validation_0-auc:0.92469
[348]	validation_0-auc:0.92471
[349]	validation_0-auc:0.924731
[350]	validation_0-auc:0.924763
[351]	validation_0-auc:0.924786
[352]	validation_0-auc:0.924815
[353]	validation_0-auc:0.924836
[354]	validation_0-auc:0.92485
[355]	validation_0-auc:0.924871
[356]	validation_0-auc:0.9249
[357]	validation_0-auc:0.924917
[358]	validation_0-auc:0.924933
[359]	validation_0-auc:0.924946
[360]	validation_0-auc:0.924964
[361]	validation_0-auc:0.924982
[362]	validation_0-auc:0.924996
[363]	validation_0-auc:0.925013
[364]	validation_0-auc:0.925034
[365]	validation_0-auc:0.925068
[366]	validation_0-auc:0.925091
[367]	validation_0-auc:0.925124
[368]	validation_0-auc:0.925156
[369]	validation_0-auc:0.925176
[370]	validation_0-auc:0.925186
[371]	validation_0-auc:0.925212
[372]	validation_0-auc:0.925231
[373]	validation_0-auc:0.925258
[374]	validation_0-auc:0.925276
[375]	validation_0-auc:0.925298
```

```
[376]	validation_0-auc:0.925324
[377]	validation_0-auc:0.925361
[378]	validation_0-auc:0.925389
[379]	validation_0-auc:0.925392
[380]	validation_0-auc:0.925406
[381]	validation_0-auc:0.925437
[382]	validation_0-auc:0.925461
[383]	validation_0-auc:0.925484
[384]	validation_0-auc:0.925506
[385]	validation_0-auc:0.925523
[386]	validation_0-auc:0.925544
[387]	validation_0-auc:0.925571
[388]	validation_0-auc:0.925585
[389]	validation_0-auc:0.925598
[390]	validation_0-auc:0.925613
[391]	validation_0-auc:0.925626
[392]	validation_0-auc:0.925649
[393]	validation_0-auc:0.92567
[394]	validation_0-auc:0.925688
[395]	validation_0-auc:0.925698
[396]	validation_0-auc:0.925727
[397]	validation_0-auc:0.925736
[398]	validation_0-auc:0.925755
[399]	validation_0-auc:0.925772
Accuracy Score is 0.84758
       0      1
0  34971   2429
1   8999  28576
```



CatBoost

In [19]:
```
# model = CatBoostClassifier(iterations=250,
#                            learning_rate=0.2,
#                            od_type='Iter',
#                            verbose=25,
#                            depth=16,
#                            random_seed=42)

# model.fit(X_train, y_train)
# y_predict = model.predict(X_test)
# print('CatBoost Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# print(pd.DataFrame(confusion_matrix(y_test,y_predict)))
```

>>> roc_auc_score(y, clf.predict_proba(X)[:, 1])

In [24]:
```
submisson = pd.DataFrame()

submisson['ID'] = test_data['ID']
test_data_temp = test_data[['Gender', 'Age', 'Region_Code', 'Occupation', 'Channel_Code',
       'Credit_Product', 'Avg_Account_Balance', 'target', 'begin_month']]

submisson['Is_Lead'] = model.predict(test_data_temp)

submisson.to_csv('sample_submission.csv', index=False)
submisson.head()
```

Out[24]:

	ID	Is_Lead
0	VBENBARO	0
1	CCMEWNKY	0
2	VK3KGA9M	0
3	TT8RPZVC	0
4	SHQZEYTZ	0

Building the Keras neural networks

After a good deal of trial and error, I found that a network architecture with three hidden layers, each followed by a dropout layer of rate 0.3, was as good as I could find. I used ReLU activation in those hidden layers, and adam optimization and a loss metric of mean squared error in the model as a whole. I also settled on a mean squared logarithmic error loss function, since it performed better than mean absolute error, mean squared error, and mean absolute percentage error.

The dataset being so large, I had great results increasing the batch size for the first couple models.

In [21]:
```
# from sklearn.model_selection import train_test_split
# from sklearn_pandas import DataFrameMapper
# from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
# from tensorflow.keras import Sequential, Input
# from tensorflow.keras.layers import Dense, Dropout
# from tensorflow.keras.callbacks import EarlyStopping



# def run_pipeline(batch_size):

#     input_nodes = X_train.shape[1]
#     output_nodes = 1

#     model = Sequential()
#     model.add(Input((input_nodes,)))
#     model.add(Dense(32, activation="sigmoid"))
#     model.add(Dropout(0.4, seed=0))
#     model.add(Dense(16, activation="sigmoid"))
#     model.add(Dropout(0.4, seed=1))
#     model.add(Dense(8, activation="sigmoid"))
#     model.add(Dropout(0.4, seed=2))
#     model.add(Dense(output_nodes, activation='sigmoid'))
#     model.compile(optimizer="adam", loss="mean_squared_error", metrics=['accuracy'])

#     es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=40)

#     history = model.fit(
#         X_train,
#         y_train,
#         batch_size=batch_size,
#         epochs=500,
#         validation_data=(X_test, y_test),
#         verbose=2,
#         callbacks=[es]
#     )

#     return history.history, model
```

In [22]:
```
# print("Model 1:")
# history_1, model1 = run_pipeline(batch_size=1024*10)
```

In [23]:
```
# submisson = pd.DataFrame()

# submisson['ID'] = test_data['ID']
# test_data_temp = test_data[['Gender', 'Age', 'Region_Code', 'Occupation', 'Channel_Code',
#        'Credit_Product', 'Avg_Account_Balance', 'target', 'begin_month']]

# submisson['Is_Lead'] = model1.predict(test_data_temp)

# submisson.to_csv('sample_submission.csv', index=False)
# submisson.head()
```