Credit Risk-Random forest, NaiveBayes, LDA

April 29, 2019

In [28]:
```
# numpy and pandas for data manipulation
import numpy as np
import pandas as pd

# sklearn preprocessing for dealing with categorical variables
from sklearn.preprocessing import LabelEncoder

# File system manangement
import os

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt
import seaborn as sns
```

In [29]:
```
# Training data
app_train = pd.read_csv(r'C:\Users\roshn\Desktop\all\application_train.csv')
print('Training data shape: ', app_train.shape)
app_train.head()
```

```
Training data shape:  (307511, 225)
```

Out[29]:
```
   SK_ID_CURR  TARGET  NAME_CONTRACT_TYPE.Cash_loans  \
0    100002.0       1                              1
1    100003.0       0                              1
2    100004.0       0                              0
3    100006.0       0                              1
4    100007.0       0                              1

   NAME_CONTRACT_TYPE.Revolving_loans  CODE_GENDER.F  CODE_GENDER.M  \
0                                   0              0              1
1                                   0              1              0
2                                   1              0              1
3                                   0              1              0
```

```
4                                  0               0               1

   FLAG_OWN_CAR.N  FLAG_OWN_CAR.Y  CNT_CHILDREN  AMT_CREDIT  \
0               1               0             0    406597.5
1               1               0             0   1293502.0
2               0               1             0    135000.0
3               1               0             0    312682.5
4               1               0             0    513000.0

                 ...                 PA_CNT_DAYS_FIRST_DUE  \
0                ...                                -565.0
1                ...                               -3823.0
2                ...                                -784.0
3                ...                              364266.0
4                ...                               -6316.0

   PA_CNT_DAYS_LAST_DUE_1ST_VERSION  PA_AVG_DAYS_LAST_DUE_1ST_VERSION  \
0                             125.0                        125.000000
1                           -3013.0                      -1004.333333
2                            -694.0                       -694.000000
3                          366336.0                      91584.000000
4                           -4186.0                       -837.200000

   PA_AVG_DAYS_LAST_DUE  PA_AVG_DAYS_TERMINATION  \
0            -25.000000               -17.000000
1          -1054.333333             -1047.333333
2           -724.000000              -714.000000
3         182477.500000            182481.750000
4          72136.200000             72143.800000

   PA_CNT_HOUR_APPR_PROCESS_START  PA_AVG_HOUR_APPR_PROCESS_START  \
0                             1.0                        9.000000
1                             3.0                       14.666667
2                             1.0                        5.000000
3                             9.0                       14.666667
4                             6.0                       12.333333

   PA_CNT_CNT_PAYMENT  PA_AVG_CNT_PAYMENT  PA_CNT_NFLAG_INSURED_ON_APPROVAL
0                24.0           24.000000                               0.0
1                30.0           10.000000                               2.0
2                 4.0            4.000000                               0.0
3               138.0           23.000000                               0.0
4               124.0           20.666667                               3.0

[5 rows x 225 columns]
```

In [30]:
```
# Testing data features
app_test = pd.read_csv(r'C:\Users\roshn\Desktop\all\application_test.csv')
```

```python
print('Testing data shape: ', app_test.shape)
app_test.head()
```

```
Testing data shape:  (48744, 224)
```

Out[30]:

```
   SK_ID_CURR  NAME_CONTRACT_TYPE.Cash_loans  \
0    100001.0                              1
1    100005.0                              1
2    100013.0                              1
3    100028.0                              1
4    100038.0                              1

   NAME_CONTRACT_TYPE.Revolving_loans  CODE_GENDER.F  CODE_GENDER.M  \
0                                   0              1              0
1                                   0              0              1
2                                   0              0              1
3                                   0              1              0
4                                   0              0              1

   FLAG_OWN_CAR.N  FLAG_OWN_CAR.Y  CNT_CHILDREN  AMT_CREDIT  AMT_GOODS_PRICE  \
0               1               0             0    568800.0         450000.0
1               1               0             0    222768.0         180000.0
2               0               1             0    663264.0         630000.0
3               1               0             2   1575000.0        1575000.0
4               0               1             1    625500.0         625500.0

                ...                PA_CNT_DAYS_FIRST_DUE  \
0               ...                              -1709.0
1               ...                               -706.0
2               ...                              -3017.0
3               ...                              -3813.0
4               ...                               -787.0

   PA_CNT_DAYS_LAST_DUE_1ST_VERSION  PA_AVG_DAYS_LAST_DUE_1ST_VERSION  \
0                           -1499.0                      -1499.000000
1                            -376.0                       -376.000000
2                           -1547.0                       -515.666667
3                          363664.0                     121221.333300
4                            -457.0                       -457.000000

   PA_AVG_DAYS_LAST_DUE  PA_AVG_DAYS_TERMINATION  \
0          -1619.000000             -1612.000000
1           -466.000000              -460.000000
2           -715.666667              -710.333333
3         121171.333300            121182.666700
4           -457.000000              -449.000000
```

```
   PA_CNT_HOUR_APPR_PROCESS_START  PA_AVG_HOUR_APPR_PROCESS_START  \
0                             1.0                            13.0
1                             2.0                            10.5
2                             4.0                            14.5
3                             5.0                            10.8
4                             2.0                             5.5

   PA_CNT_CNT_PAYMENT  PA_AVG_CNT_PAYMENT  PA_CNT_NFLAG_INSURED_ON_APPROVAL
0                 8.0            8.000000                               0.0
1                12.0           12.000000                               0.0
2                52.0           17.333333                               1.0
3                34.0           11.333333                               0.0
4                48.0           24.000000                               0.0

[5 rows x 224 columns]
```

In [31]:
```
# Create a label encoder object
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in app_train:
    if app_train[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(app_train[col].unique())) <= 2:
            # Train on the training data
            le.fit(app_train[col])
            # Transform both training and testing data
            app_train[col] = le.transform(app_train[col])
            app_test[col] = le.transform(app_test[col])

            # Keep track of how many columns were label encoded
            le_count += 1

print('%d columns were label encoded.' % le_count)
```

```
0 columns were label encoded.
```

In [32]:
```
# one-hot encoding of categorical variables
app_train = pd.get_dummies(app_train)
app_test = pd.get_dummies(app_test)

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

```
---------------------------------------------------------------------------
```

```
MemoryError                               Traceback (most recent call last)

<ipython-input-32-4c395bbbbf95> in <module>()
      1 # one-hot encoding of categorical variables
----> 2 app_train = pd.get_dummies(app_train)
      3 app_test = pd.get_dummies(app_test)
      4
      5 print('Training Features shape: ', app_train.shape)


~\Anaconda3\lib\site-packages\pandas\core\reshape\reshape.py in get_dummies(data, prefi
    878             dummy = _get_dummies_1d(col[1], prefix=pre, prefix_sep=sep,
    879                                     dummy_na=dummy_na, sparse=sparse,
--> 880                                     drop_first=drop_first, dtype=dtype)
    881             with_dummies.append(dummy)
    882         result = concat(with_dummies, axis=1)


~\Anaconda3\lib\site-packages\pandas\core\reshape\reshape.py in _get_dummies_1d(data, p
    966
    967     else:
--> 968         dummy_mat = np.eye(number_of_cols, dtype=dtype).take(codes, axis=0)
    969
    970         if not dummy_na:


~\Anaconda3\lib\site-packages\numpy\lib\twodim_base.py in eye(N, M, k, dtype, order)
    184     if M is None:
    185         M = N
--> 186     m = zeros((N, M), dtype=dtype, order=order)
    187     if k >= M:
    188         return m


MemoryError:
```

In [26]:
```python
train_labels = app_train['TARGET']

# Align the training and testing data, keep only columns present in both dataframes
app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)

# Add the target back in
app_train['TARGET'] = train_labels

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)# Create an anomalous flag column
app_train['DAYS_EMPLOYED_ANOM'] = app_train["DAYS_EMPLOYED"] == 365243
```

```
# Replace the anomalous values with nan
app_train['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)

app_train['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
plt.xlabel('Days Employment');
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method
   3062             try:
-> 3063                 return self._engine.get_loc(key)
   3064             except KeyError:


pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.ge


pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.ge


KeyError: 'TARGET'


During handling of the above exception, another exception occurred:


KeyError                                  Traceback (most recent call last)

<ipython-input-26-b8da99b4956e> in <module>()
----> 1 train_labels = app_train['TARGET']
      2
      3 # Align the training and testing data, keep only columns present in both dataframe
      4 app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)
      5


~\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   2683             return self._getitem_multilevel(key)
```

```
   2684         else:
-> 2685             return self._getitem_column(key)
   2686 
   2687     def _getitem_column(self, key):


    ~\Anaconda3\lib\site-packages\pandas\core\frame.py in _getitem_column(self, key)
   2690         # get column
   2691         if self.columns.is_unique:
-> 2692             return self._get_item_cache(key)
   2693 
   2694         # duplicate columns & possible reduce dimensionality


    ~\Anaconda3\lib\site-packages\pandas\core\generic.py in _get_item_cache(self, item)
   2484         res = cache.get(item)
   2485         if res is None:
-> 2486             values = self._data.get(item)
   2487             res = self._box_item_values(item, values)
   2488             cache[item] = res


    ~\Anaconda3\lib\site-packages\pandas\core\internals.py in get(self, item, fastpath)
   4113 
   4114             if not isna(item):
-> 4115                 loc = self.items.get_loc(item)
   4116             else:
   4117                 indexer = np.arange(len(self.items))[isna(self.items)]


    ~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method
   3063                 return self._engine.get_loc(key)
   3064             except KeyError:
-> 3065                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   3066 
   3067         indexer = self.get_indexer([key], method=method, tolerance=tolerance)


    pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.ge


    pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.ge
```

```
KeyError: 'TARGET'
```

In [7]:
```
anom = app_train[app_train['DAYS_EMPLOYED'] == 365243]
non_anom = app_train[app_train['DAYS_EMPLOYED'] != 365243]
print('The non-anomalies default on %0.2f%% of loans' % (100 * non_anom['TARGET'].mean
print('The anomalies default on %0.2f%% of loans' % (100 * anom['TARGET'].mean()))
print('There are %d anomalous days of employment' % len(anom))
```

```
The non-anomalies default on 8.07% of loans
The anomalies default on nan% of loans
There are 0 anomalous days of employment
```

In [8]:
```
app_test['DAYS_EMPLOYED_ANOM'] = app_test["DAYS_EMPLOYED"] == 365243
app_test["DAYS_EMPLOYED"].replace({365243: np.nan}, inplace = True)

print('There are %d anomalies in the test data out of %d entries' % (app_test["DAYS_EM
```

```
There are 9274 anomalies in the test data out of 48744 entries
```

In [9]:
```
# Find correlations with the target and sort
correlations = app_train.corr()['TARGET'].sort_values()

# Display correlations
print('Most Positive Correlations:\n', correlations.tail(15))
print('\nMost Negative Correlations:\n', correlations.head(15))
```

```
Most Positive Correlations:
 OCCUPATION_TYPE_Laborers                             0.043019
FLAG_DOCUMENT_3                                      0.044346
REG_CITY_NOT_LIVE_CITY                               0.044395
FLAG_EMP_PHONE                                       0.045982
NAME_EDUCATION_TYPE_Secondary / secondary special    0.049824
REG_CITY_NOT_WORK_CITY                               0.050994
DAYS_ID_PUBLISH                                      0.051457
CODE_GENDER_M                                        0.054713
DAYS_LAST_PHONE_CHANGE                               0.055218
NAME_INCOME_TYPE_Working                             0.057481
REGION_RATING_CLIENT                                 0.058899
REGION_RATING_CLIENT_W_CITY                          0.060893
DAYS_EMPLOYED                                        0.074958
DAYS_BIRTH                                           0.078239
TARGET                                               1.000000
Name: TARGET, dtype: float64
```

```
Most Negative Correlations:
 EXT_SOURCE_3                           -0.178919
EXT_SOURCE_2                           -0.160472
EXT_SOURCE_1                           -0.155317
NAME_EDUCATION_TYPE_Higher education   -0.056593
CODE_GENDER_F                          -0.054704
NAME_INCOME_TYPE_Pensioner             -0.046209
DAYS_EMPLOYED_ANOM                     -0.045987
ORGANIZATION_TYPE_XNA                  -0.045987
FLOORSMAX_AVG                          -0.044003
FLOORSMAX_MEDI                         -0.043768
FLOORSMAX_MODE                         -0.043226
EMERGENCYSTATE_MODE_No                 -0.042201
HOUSETYPE_MODE_block of flats          -0.040594
AMT_GOODS_PRICE                        -0.039645
REGION_POPULATION_RELATIVE             -0.037227
Name: TARGET, dtype: float64
```

In [10]:
```
# Find the correlation of the positive days since birth and target
app_train['DAYS_BIRTH'] = abs(app_train['DAYS_BIRTH'])
app_train['DAYS_BIRTH'].corr(app_train['TARGET'])
```

Out[10]: -0.07823930830982712

In [11]:
```
# Age information into a separate dataframe
age_data = app_train[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365

# Bin the age data
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20, 70,
age_data.head(10)
```

Out[11]:

	TARGET	DAYS_BIRTH	YEARS_BIRTH	YEARS_BINNED
0	1	9461	25.920548	(25.0, 30.0]
1	0	16765	45.931507	(45.0, 50.0]
2	0	19046	52.180822	(50.0, 55.0]
3	0	19005	52.068493	(50.0, 55.0]
4	0	19932	54.608219	(50.0, 55.0]
5	0	16941	46.413699	(45.0, 50.0]
6	0	13778	37.747945	(35.0, 40.0]
7	0	18850	51.643836	(50.0, 55.0]
8	0	20099	55.065753	(55.0, 60.0]
9	0	14469	39.641096	(35.0, 40.0]

In [12]:
```
# Group by the bin and calculate averages
age_groups  = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

Out[12]:

	TARGET	DAYS_BIRTH	YEARS_BIRTH
YEARS_BINNED			
(20.0, 25.0]	0.123036	8532.795625	23.377522
(25.0, 30.0]	0.111436	10155.219250	27.822518
(30.0, 35.0]	0.102814	11854.848377	32.479037
(35.0, 40.0]	0.089414	13707.908253	37.555913
(40.0, 45.0]	0.078491	15497.661233	42.459346
(45.0, 50.0]	0.074171	17323.900441	47.462741
(50.0, 55.0]	0.066968	19196.494791	52.593136
(55.0, 60.0]	0.055314	20984.262742	57.491131
(60.0, 65.0]	0.052737	22780.547460	62.412459
(65.0, 70.0]	0.037270	24292.614340	66.555108

In [13]:

```
# Extract the EXT_SOURCE variables and show correlations
ext_data = app_train[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_
ext_data_corrs = ext_data.corr()
ext_data_corrs
```

Out[13]:

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155317	-0.160472	-0.178919	-0.078239
EXT_SOURCE_1	-0.155317	1.000000	0.213982	0.186846	0.600610
EXT_SOURCE_2	-0.160472	0.213982	1.000000	0.109167	0.091996
EXT_SOURCE_3	-0.178919	0.186846	0.109167	1.000000	0.205478
DAYS_BIRTH	-0.078239	0.600610	0.091996	0.205478	1.000000

In [14]:

```
# Make a new dataframe for polynomial features
poly_features = app_train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRT
poly_features_test = app_test[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_

# imputer for handling missing values
from sklearn.preprocessing import Imputer
imputer = Imputer(strategy = 'median')

poly_target = poly_features['TARGET']

poly_features = poly_features.drop(columns = ['TARGET'])

# Need to impute missing values
poly_features = imputer.fit_transform(poly_features)
poly_features_test = imputer.transform(poly_features_test)

from sklearn.preprocessing import PolynomialFeatures

# Create the polynomial object with specified degree
poly_transformer = PolynomialFeatures(degree = 3)
```

In [15]:

```
# Train the polynomial features
poly_transformer.fit(poly_features)
```

```
# Transform the features
poly_features = poly_transformer.transform(poly_features)
poly_features_test = poly_transformer.transform(poly_features_test)
print('Polynomial Features shape: ', poly_features.shape)
```

```
Polynomial Features shape:  (307511, 35)
```

In [16]:
```
poly_transformer.get_feature_names(input_features = ['EXT_SOURCE_1', 'EXT_SOURCE_2',
```

Out[16]:
```
['1',
 'EXT_SOURCE_1',
 'EXT_SOURCE_2',
 'EXT_SOURCE_3',
 'DAYS_BIRTH',
 'EXT_SOURCE_1^2',
 'EXT_SOURCE_1 EXT_SOURCE_2',
 'EXT_SOURCE_1 EXT_SOURCE_3',
 'EXT_SOURCE_1 DAYS_BIRTH',
 'EXT_SOURCE_2^2',
 'EXT_SOURCE_2 EXT_SOURCE_3',
 'EXT_SOURCE_2 DAYS_BIRTH',
 'EXT_SOURCE_3^2',
 'EXT_SOURCE_3 DAYS_BIRTH',
 'DAYS_BIRTH^2']
```

In [17]:
```
# Create a dataframe of the features
poly_features = pd.DataFrame(poly_features,
                             columns = poly_transformer.get_feature_names(['EXT_SOURCE
                                                                           'EXT_SOURCE

# Add in the target
poly_features['TARGET'] = poly_target

# Find the correlations with the target
poly_corrs = poly_features.corr()['TARGET'].sort_values()

# Display most negative and most positive
print(poly_corrs.head(10))
print(poly_corrs.tail(5))
```

```
EXT_SOURCE_2 EXT_SOURCE_3                -0.193939
EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3   -0.189605
EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH     -0.181283
EXT_SOURCE_2^2 EXT_SOURCE_3              -0.176428
EXT_SOURCE_2 EXT_SOURCE_3^2              -0.172282
EXT_SOURCE_1 EXT_SOURCE_2                -0.166625
EXT_SOURCE_1 EXT_SOURCE_3                -0.164065
EXT_SOURCE_2                             -0.160295
```

```
EXT_SOURCE_2 DAYS_BIRTH                  -0.156873
EXT_SOURCE_1 EXT_SOURCE_2^2              -0.156867
Name: TARGET, dtype: float64
DAYS_BIRTH     -0.078239
DAYS_BIRTH^2   -0.076672
DAYS_BIRTH^3   -0.074273
TARGET          1.000000
1                    NaN
Name: TARGET, dtype: float64
```

In [18]:
```
# Put test features into dataframe
poly_features_test = pd.DataFrame(poly_features_test,
                                  columns = poly_transformer.get_feature_names(['EXT_S
                                                                               'EXT_S

# Merge polynomial features into training dataframe
poly_features['SK_ID_CURR'] = app_train['SK_ID_CURR']
app_train_poly = app_train.merge(poly_features, on = 'SK_ID_CURR', how = 'left')

# Merge polnomial features into testing dataframe
poly_features_test['SK_ID_CURR'] = app_test['SK_ID_CURR']
app_test_poly = app_test.merge(poly_features_test, on = 'SK_ID_CURR', how = 'left')

# Align the dataframes
app_train_poly, app_test_poly = app_train_poly.align(app_test_poly, join = 'inner', ax

# Print out the new shapes
print('Training data with polynomial features shape: ', app_train_poly.shape)
print('Testing data with polynomial features shape:  ', app_test_poly.shape)
```

```
Training data with polynomial features shape:  (307511, 275)
Testing data with polynomial features shape:   (48744, 275)
```

In [19]:
```
app_train_domain = app_train.copy()
app_test_domain = app_test.copy()

app_train_domain['CREDIT_INCOME_PERCENT'] = app_train_domain['AMT_CREDIT'] / app_train
app_train_domain['ANNUITY_INCOME_PERCENT'] = app_train_domain['AMT_ANNUITY'] / app_tra
app_train_domain['CREDIT_TERM'] = app_train_domain['AMT_ANNUITY'] / app_train_domain[
app_train_domain['DAYS_EMPLOYED_PERCENT'] = app_train_domain['DAYS_EMPLOYED'] / app_tr

app_test_domain['CREDIT_INCOME_PERCENT'] = app_test_domain['AMT_CREDIT'] / app_test_do
app_test_domain['ANNUITY_INCOME_PERCENT'] = app_test_domain['AMT_ANNUITY'] / app_test_
app_test_domain['CREDIT_TERM'] = app_test_domain['AMT_ANNUITY'] / app_test_domain['AMT
app_test_domain['DAYS_EMPLOYED_PERCENT'] = app_test_domain['DAYS_EMPLOYED'] / app_test
```

In [20]:
```
from sklearn.ensemble import RandomForestClassifier
```

```
# Make the random forest classifier
random_forest = RandomForestClassifier(n_estimators = 100, random_state = 50, verbose
```

In [13]:
```
from sklearn.preprocessing import MinMaxScaler, Imputer

# Drop the target from the training data
if 'SK_ID_CURR' in app_train:
    train = app_train.drop(columns = ['SK_ID_CURR'])
else:
    train = app_train.copy()

# Feature names
features = list(train.columns)

# Copy of the testing data
test = app_test.copy()

# Median imputation of missing values
imputer = Imputer(strategy = 'median')

# Scale each feature to 0-1
scaler = MinMaxScaler(feature_range = (0, 1))

# Fit on the training data
imputer.fit(train)

# Transform both training and testing data
train = imputer.transform(train)
test = imputer.transform(app_test)

# Repeat with the scaler
scaler.fit(train)
train = scaler.transform(train)
test = scaler.transform(test)

print('Training data shape: ', train.shape)
print('Testing data shape: ', test.shape)
```

```
Training data shape:  (356255, 46)
Testing data shape:  (48744, 46)
```

In [49]:
```
# Train on the training data
random_forest.fit(train, train_labels)

# Extract feature importances
feature_importance_values = random_forest.feature_importances_
feature_importances = pd.DataFrame({'feature': features, 'importance': feature_importa
```

```
# Make predictions on the test data
predictions = random_forest.predict_proba(test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   32.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.4min finished
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.9s finished
```

In [50]:
```
# Make a submission dataframe
submit = app_test[['SK_ID_CURR']]
submit['SK_ID_CURR'] = predictions
```

In [51]:
```
submit.head(10)
```

Out[51]:

	SK_ID_CURR	TARGET
0	100001	0.13
1	100005	0.21
2	100013	0.05
3	100028	0.14
4	100038	0.19
5	100042	0.15
6	100057	0.12
7	100065	0.14
8	100066	0.11
9	100067	0.18

In [26]:
```
from sklearn.naive_bayes import GaussianNB


# Perform naive Bayes classification
nb = GaussianNB()
nb.fit(train.data, train_labels.data)

#feature_importance_values = nb.feature_importances_
#feature_importances = pd.DataFrame({'feature': features, 'importance': feature_impor

nb_res  = nb.predict_proba(test.data)[:, 1]
```

In [27]:
```
# Make a submission dataframe
submit1 = app_test[['SK_ID_CURR']]
submit1['TARGET'] = nb_res
```

In [28]:
```
submit1.head(10)
```

Out[28]:

	SK_ID_CURR	TARGET
0	100001	1.000000e+00
1	100005	1.000000e+00

```
2       100013  1.000000e+00
3       100028  1.935577e-08
4       100038  1.000000e+00
5       100042  9.401605e-01
6       100057  1.000000e+00
7       100065  1.000000e+00
8       100066  1.000000e+00
9       100067  1.000000e+00
```

In [52]:
```python
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

In [63]:
```python
clf_gini = DecisionTreeClassifier(random_state = 50, min_samples_leaf=1, min_samples_s
clf_gini.fit(train, train_labels)
```

Out[63]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=50,
            splitter='best')
```

In [64]:
```python
feature_importance_values = clf_gini.feature_importances_
feature_importances = pd.DataFrame({'feature': features, 'importance': feature_importa

# Make predictions on the test data
prediction1 = clf_gini.predict_proba(test)[:, 1]
```

In [65]:
```python
# Make a submission dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = prediction1
```

In [66]:
```python
submit.head(20)
```

Out[66]:

	SK_ID_CURR	TARGET
0	100001	0.0
1	100005	1.0
2	100013	0.0
3	100028	0.0
4	100038	1.0
5	100042	1.0
6	100057	1.0
7	100065	0.0
8	100066	0.0
9	100067	0.0
10	100074	0.0
11	100090	0.0

```
12      100091     0.0
13      100092     0.0
14      100106     0.0
15      100107     1.0
16      100109     0.0
17      100117     0.0
18      100128     0.0
19      100141     0.0
```

In [ ]:
```
submit.to_csv("first.csv", index=False)
```

In [33]:
```
# Applying Linear Discriminant Analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components = 2)
lda.fit(train, train_labels)

X_test = lda.predict_proba(test)[:, 1]
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-33-762a387016f4> in <module>()
      2 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
      3 lda = LinearDiscriminantAnalysis(n_components = 2)
----> 4 lda.fit(train, train_labels)
      5
      6 X_test = lda.predict_proba(test)[:, 1]


~\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py in fit(self, X, y)
    427             Target values.
    428         """
--> 429         X, y = check_X_y(X, y, ensure_min_samples=2, estimator=self)
    430         self.classes_ = unique_labels(y)
    431


~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_spa
    581         y = y.astype(np.float64)
    582
--> 583     check_consistent_length(X, y)
    584
    585     return X, y


~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_consistent_length(*a
```

```
    202     if len(uniques) > 1:
    203         raise ValueError("Found input variables with inconsistent numbers of"
--> 204                          " samples: %r" % [int(l) for l in lengths])
    205
    206


    ValueError: Found input variables with inconsistent numbers of samples: [356255, 30751
```

```
In [34]: submit = app_test[['SK_ID_CURR']]
         submit['TARGET'] = X_test
```

```
    ---------------------------------------------------------------------------

    NameError                                 Traceback (most recent call last)

    <ipython-input-34-9a95cc9a64ea> in <module>()
      1 submit = app_test[['SK_ID_CURR']]
----> 2 submit['TARGET'] = X_test


    NameError: name 'X_test' is not defined
```

```
In [35]: submit.head()
```

Out[35]:

	SK_ID_CURR
0	100001.0
1	100005.0
2	100013.0
3	100028.0
4	100038.0

```
In [23]: app_train_domain = app_train_domain.drop(columns = 'TARGET')

         domain_features_names = list(app_train_domain.columns)

         # Impute the domainnomial features
         imputer = Imputer(strategy = 'median')

         domain_features = imputer.fit_transform(app_train_domain)
         domain_features_test = imputer.transform(app_test_domain)

         # Scale the domainnomial features
         scaler = MinMaxScaler(feature_range = (0, 1))

         domain_features = scaler.fit_transform(domain_features)
```

```
domain_features_test = scaler.transform(domain_features_test)

#random_forest_domain = RandomForestClassifier(n_estimators = 100, random_state = 50,
XGB_params = {'num_round':200}
xgb_domain = XGBClassifier(max_depth = 6, learning_rate=0.2, estimator =100, **XGB_pa

# Train on the training data
#random_forest_domain.fit(domain_features, train_labels)
xgb_domain.fit(domain_features, train_labels)

# Extract feature importances
#feature_importance_values_domain = random_forest_domain.feature_importances_
#feature_importances_domain = pd.DataFrame({'feature': domain_features_names, 'import

# Make predictions on the test data
#predictions = random_forest_domain.predict_proba(domain_features_test)[:, 1]
X_test = xgb_domain.predict_proba(domain_features_test)[:, 1]
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-23-d86b07a564f8> in <module>()
----> 1 app_train_domain = app_train_domain.drop(columns = 'TARGET')
      2
      3 domain_features_names = list(app_train_domain.columns)
      4
      5 # Impute the domainnomial features


~\Anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axis, index,
   3692                                            index=index, columns=columns,
   3693                                            level=level, inplace=inplace,
-> 3694                                            errors=errors)
   3695
   3696     @rewrite_axis_style_signature('mapper', [('copy', True),


~\Anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, axis, index
   3106         for axis, labels in axes.items():
   3107             if labels is not None:
-> 3108                 obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   3109
   3110         if inplace:


~\Anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, labels, axis,
```

```
   3138                 new_axis = axis.drop(labels, level=level, errors=errors)
   3139             else:
-> 3140                 new_axis = axis.drop(labels, errors=errors)
   3141             dropped = self.reindex(**{axis_name: new_axis})
   3142             try:


    ~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, errors)
   4385             if errors != 'ignore':
   4386                 raise KeyError(
-> 4387                     'labels %s not contained in axis' % labels[mask])
   4388             indexer = indexer[~mask]
   4389         return self.delete(indexer)


    KeyError: "labels ['TARGET'] not contained in axis"
```

In [32]:
```
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = X_test
```

In [ ]:
```
submit.head()
```

In [1]:
```
# Applying XGB
from xgboost import XGBClassifier
XGB_params = {'num_round':200}
xgb = XGBClassifier(max_depth = 6, learning_rate=0.2, estimator =100, **XGB_params)
xgb.fit(train, train_labels)

X_test = xgb.predict_proba(test)[:, 1]
```

```
    ---------------------------------------------------------------------------

    NameError                                 Traceback (most recent call last)

    <ipython-input-1-1d451edc2c0f> in <module>()
      3 XGB_params = {'num_round':200}
      4 xgb = XGBClassifier(max_depth = 6, learning_rate=0.2, estimator =100, **XGB_params)
----> 5 xgb.fit(train, train_labels)
      6 
      7 X_test = xgb.predict_proba(test)[:, 1]


    NameError: name 'train' is not defined
```

In [28]:
```
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = X_test
```

In [29]: submit.head()

Out[29]:

	SK_ID_CURR	TARGET
0	100001	0.028452
1	100005	0.091167
2	100013	0.015969
3	100028	0.012882
4	100038	0.069109