Machine Learning Models and Predictive Analysis

Laura Lazarescou, John Rodgers, Maysam Mansor and Mel Schwan

March 1, 2021

Introduction

All machine learning models are categorized as either supervised or unsupervised. Supervised learning involves learning a function that maps an input to an output based on example input-output pairs. Supervised models are then sub-categorized as a regression or classification model. In regression models, the result is continuous. Linear regression is merely finding a line that best fits the data. Decision trees are a popular model used in operations research, strategic planning, and machine learning. The last node of the decision tree, where a decision is made, is called the tree leaves. (Terence Shin - towards data science, 2020)

Extreme Gradient Boosting is a model that creates a partition tree to make predictions on class-level outcomes using data subsets. New, subsequent partition trees are applied to the remaining batches of the dataset until residual error is minimized. The weight of each sample batch is adaptively changed after each round of boosting (new tree). The model focuses on building trees to correctly explain data contributing to incorrect classifications. This is repeated until optimal performance is obtained. However, XGBoost is prone to over-fitting.

Support Vector Machine, is a supervised classification technique that can get pretty complicated but is intuitive at the most fundamental level. A support vector machine will find a hyperplane or a boundary between the the classes of data that maximizes the margin between the the classes. Many planes can separate the classes, but only one plane can maximize the margin or distance between the classes. This plane becomes the optimal solution for the model.

The third model type, Random Forest, is an ensemble learning technique that builds off of decision trees. Random forests involve creating multiple decision trees using bootstrapped datasets of the original data and randomly selecting a subset of variables at each decision tree step. Relying on a "majority wins" model reduces the risk of error from an individual tree.

We will train all three models in this study and vary the hyperparameters to maximize the models' accuracy values and minimize log-loss while considering the runtime requirements.

Methods

Data Preparation

Financial data, containing 538 features and a binary target are the corpus used for this study. There were no feature descriptions included. The features were labeled with non-descriptive letters like v1, v2 etc.

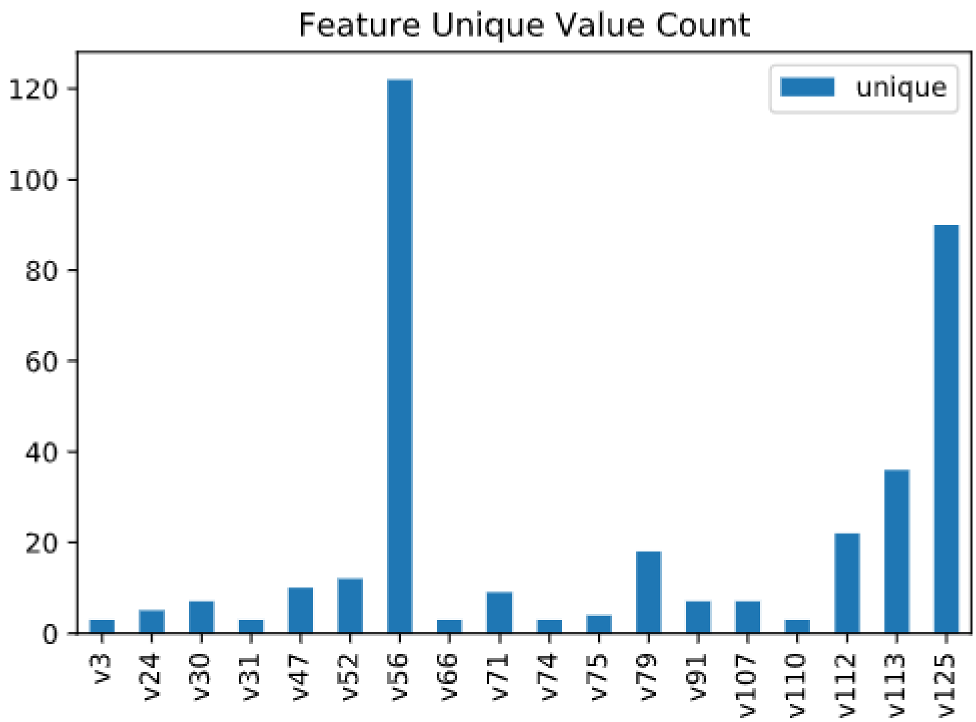
Our prepared dataset included 538 features once we applied one-hot encoding and categorical transformation to all non-numeric factors.

Factor v22 received exceptional treatment since it included over 18000 unique character values. Using a grouping approach, we categorized v22 into 53 factors based on the rationale that a value should have at least 125 occurrences to be represented in the dataset. If we had one-hot encoded v22 without this transformation, our final dataset would have included more than 19000 factors and it would have caused the feature space to grow intractably.

Sparce matrices can lead to poor models and results. Thus we have choosen to reduce the model feature set to the most impactful factors.

Data Types and Data Distribution

The majority of our original factors were numeric. These factors are naturally compatible with machine learning models that we have used in this study. Figure 1 shows the unique values counts of non-numeric features, except for factor v22. It was removed from this plot because of its large number of unique values (18,210). v56 feature has the next greatest number of unique values.(125)



Unique Value Count per Non-Numeric Feature (excluding v22) (Figure 1)

Model Validation

The models were validated using five(5)-fold cross-validation. The steps for model validation are given below. Hyperameter tuning procedures and choices are discussed in each Model Discussion.

Model Validation Procedure

- Split data into 67% training and 33% test sets. The train dataset consisted of 76595 rows and the test dataset included 37726 rows. Both datasets included 538 factors.
- Use the train dataset with cross-validation to build multiple models of each type.
- Estimate the log-loss of XGBoost and Random Forest models, and the accuracy for all models by using the test dataset as a validation dataset, then comparing the predicted target values to the known target values.
- Statistics and performance of each model are listed in Results.

Models

In this case study, we use XGBoost, Support Vector Machines, and Random Forest to model the data. In each section we will desrcibe how these models work, and what steps were taking to tune each model.

Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is laser focused on computational speed and model performance. (Jason Brownlee; 2016, machine learning mastery)

Model Features

Three main forms of gradient boosting are supported:

- Gradient Boosting algorithm also called gradient boosting machine including the learning rate.
- Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.
- Regularized Gradient Boosting with both L1 and L2 regularization.

Algorithm Features

Some key algorithm implementation features include:

- Sparse Aware implementation with automatic handling of missing data values.
- Block Structure to support the parallelization of tree construction.
- Continued Training so that you can further boost an already fitted model on new data.

XGBoost Hyperparameter Tuning

To find an optimal combination of hyperparameters for an XGBoost model, a randomized search of combinations was performed to identify the best performing model based on the value of log loss. Each of these hyperparameter combinations was evaluated using 5-fold cross validation of the training data set. The following hyper-paramaters and values were incorporated into the randomized grid search. (Table 1)

Hyperparameter	Values
max_depth	6, 10, 15, 20
learning_rate	0.001, 0.01, 0.1, 0.2, 0.3
subsample	0.5, 0.6, 0.7, 0.8, 0.9, 1.0
colsample_bytree	0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
colsample_bylevel	0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
min_child_weight	0.5, 1.0, 3.0, 5.0, 7.0, 10.0
gamma	0, 0.25, 0.5, 1.0
reg_lambda	0.1, 1.0, 5.0, 10.0, 50.0, 100.0

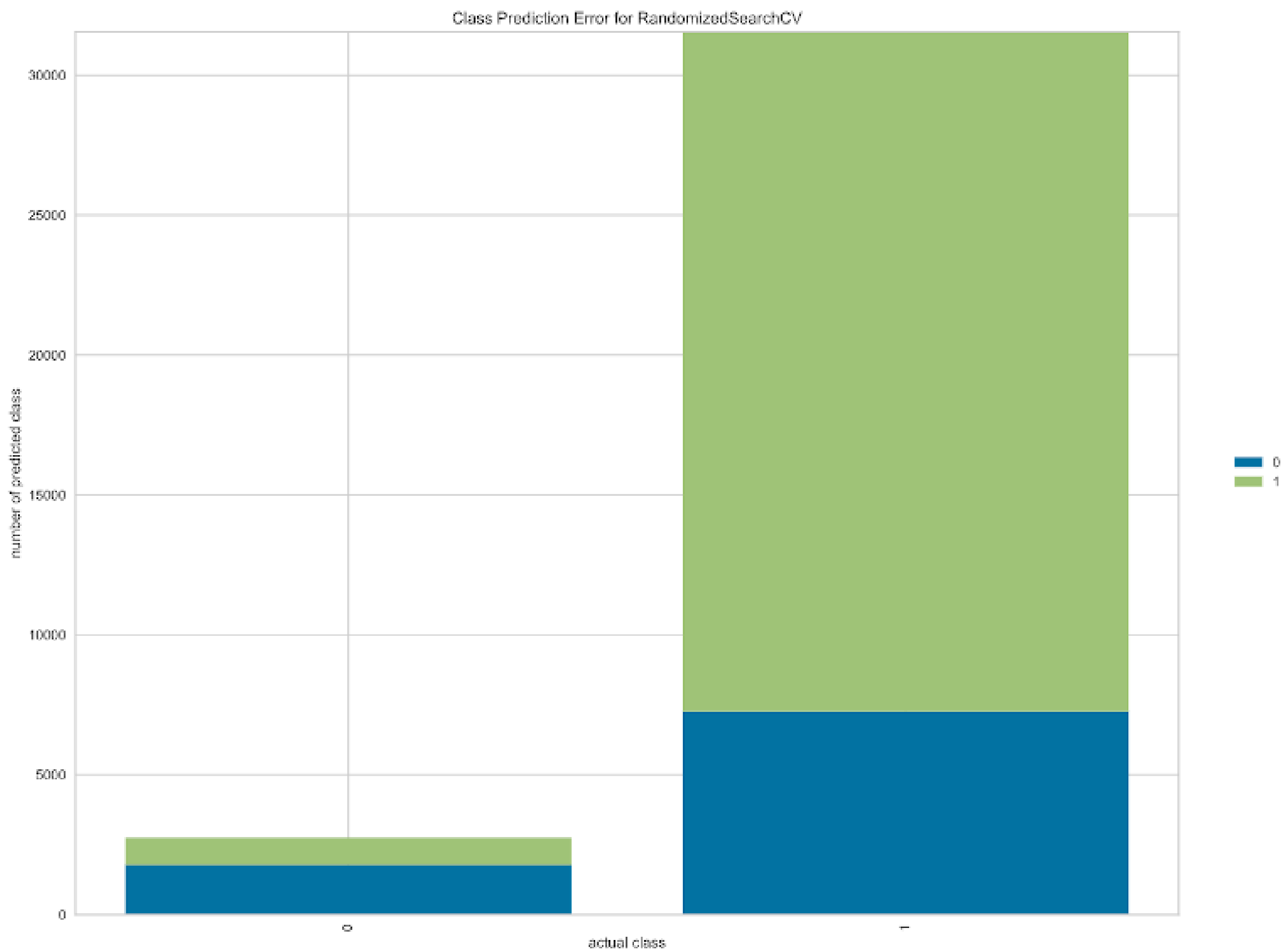
Table 1

The search model selected 5 hyperparameter combinations at random from the list above. With each of these 5 models being evaluated with a 5 cross-fold cross-validation, a total of 25 models were evaluated to determine the best-performing combination of hyperparameters. Log loss was used to identify the best-performing model, with the following combination of hyperparameters returning a log-loss value of 0.4691 and an accuracy score of 0.7683. (Table 2)

Hyperparameter	Value
max_depth	10
learning_rate	0.1
subsample	0.9
colsample_bytree	0.5
colsample_bylevel	0.4
min_child_weight	7.0
gamma	0.5
reg_lambda	10.0

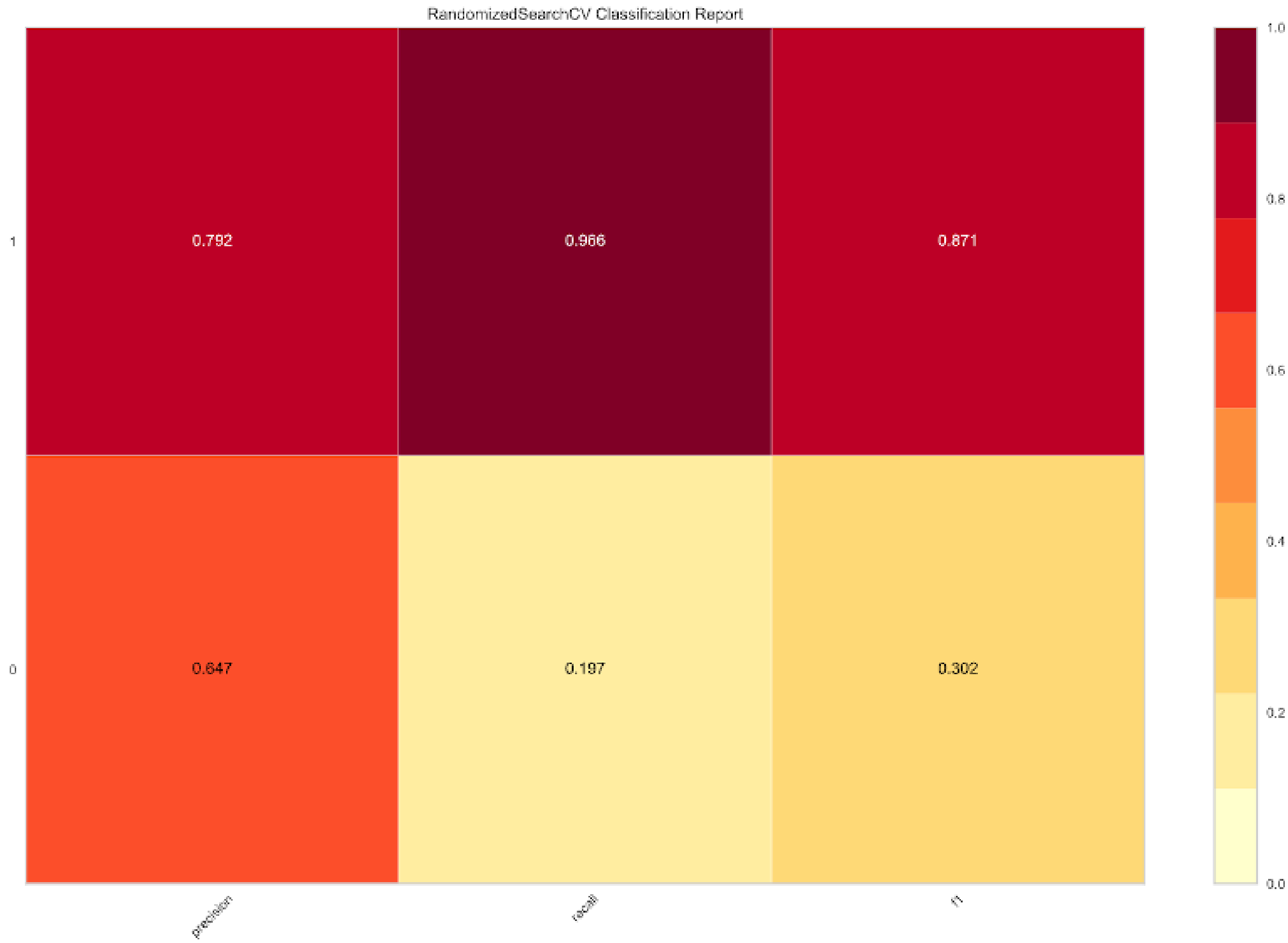
Table 2

Additional metrics for the tuned XGBoost model were also evaluated. The following figure shows the actual classes of the test data compared to the value that the model predicted. (Figure 2)



XGB Class Prediction (Figure 2)

The figure below shows the precision, recall, and f1 score for each of the two classes. (Figure 3)



XGB Classification Report (Figure 3)

Support Vector Machine

TA support vector machine is a supervised learning algorithm that sorts data into two categories. It is trained with a series of data already classified into two categories, building the model as it is initially trained. The task of an SVM algorithm is to determine which category a new data point belongs in. This makes SVM a kind of non-binary linear classifier.

An SVM algorithm should not only place objects into categories, but have the margins between them on a graph as wide as possible.

Advantages:

- Works relatively well when there is a clear margin of separation between classes
- More effective in high dimensional spaces
- Effective in cases where the number of dimensions is greater than the number of samples
- Relatively memory efficient

Disadvantages:

- Algorithm is not suitable for large data sets
- Does not perform very well when the data set has more noise i.e. target classes are overlapping
- In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform
- Works by putting data points, above and below the classifying hyperplane there is no probabilistic explanation for the classification

Estimating the SVM in these high-dimensional spaces is considerably computationally expensive. Consider the model complexity when determining whether SVM should be implemented.

SVM Hyperparameter Tuning

Hyperparameters were selected with a randomized search.GridSearchCV is a library functions that is a member of sklearn's model_selection package. It loop through predefined Hyperparameter and fit the model on the training set.In the end best parameteres from the list of hyperparameteres can be selected. In our LinearSVC we evaluated hyperparameteres(C,loss,penalty,dual and tol,max_iter) and set a range for each and at the end we got accuracy of 0.77 with best parameteres were selected and tabulated in the table which gained 0.001 for C, squared_hinge for param_loss and with param_dual become False, tol =1e-05 with 100 max_iter with 872 sec long and finally with accuracy of 0.771137. We also hypertuned LinearSVC for sample size 1000, 2000, 5000, 10000 and find out accuracy for each and time of fiting the model was calculated and reported respectively. We showed results of parameter tuning and accuracy in a coresponsing tables.

Random Forest

A Random Forest is an emsemble model created from a collection of decision trees and bootstrapped aggregated (bagged) data (Breiman, 1996; James et al, 2013). The following steps are used to create bagged trees:

- bootstrap sample (repeated sampling with replacement) the dataset to create B separate datasets.
- fit a model $f^b(x)$ on each B dataset.

The bagged decision tree model is the majority vote of the classifiers resulting in the class prediction. Generally an ensemble should consist of a large number of decision trees. The number of decision trees was used as a hyperparameter and we tuned it with cross-validation.

Random Forest Hyperparameter Tuning

For Random Forest we also used GridSearchCV to randomly select combinations of hyper parameters. Outside of the Grid, the value of n_iterations is one of the most influential parameters in Random Forest. In our base model we chose n_iterations = 10 and ran it on the complete train dataset.

Below in figure 4 is a list of the default or base parameters used in our RF Base Model:

```
Parameters Used by Base Model:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 10,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 123,
 'verbose': 0,
 'warm_start': False}
```

Random Forest Base Parameters (Figure 4)

Grid Parameter Options

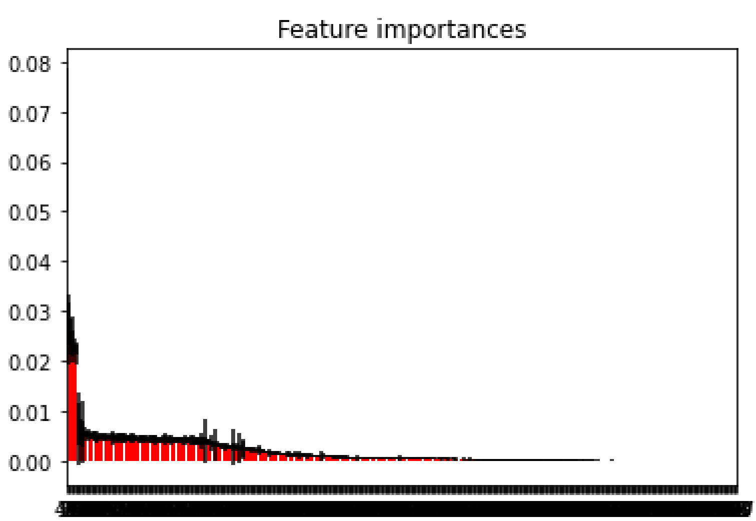
We did not include all possible parameters that could be tuned in our grid definition. We also limited n_iterations to 5000 and in our results, the algorithm only chose 1000. From this we learned that if one wants to test some extreme conditions or specific grid configurations, a random grid search may not be the best approach. Figure 4 shows the grid parameters that could be chosen at random by the GridSearchCV function (Figure 5).

```
Random Grid Parameters:

{'bootstrap': [True, False],
 'max_depth': [10, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [100, 500, 1000, 1500]}
```

Random Forest Grid Parameters (Figure 5)

Top Features

We will not share the top features from all models that were run, however we did notice that the top ten features from different models were not consistent. The feature importance graph in Figure 4 is difficult to read because there are so many features. However, what it does show well is that no features has a significant percent of importance. The top 10 features out of 508 features represent approximately 30% of the influence in the model. This says that the variability of the model is high, and we have seen that in our different scenarios (Figure 6)



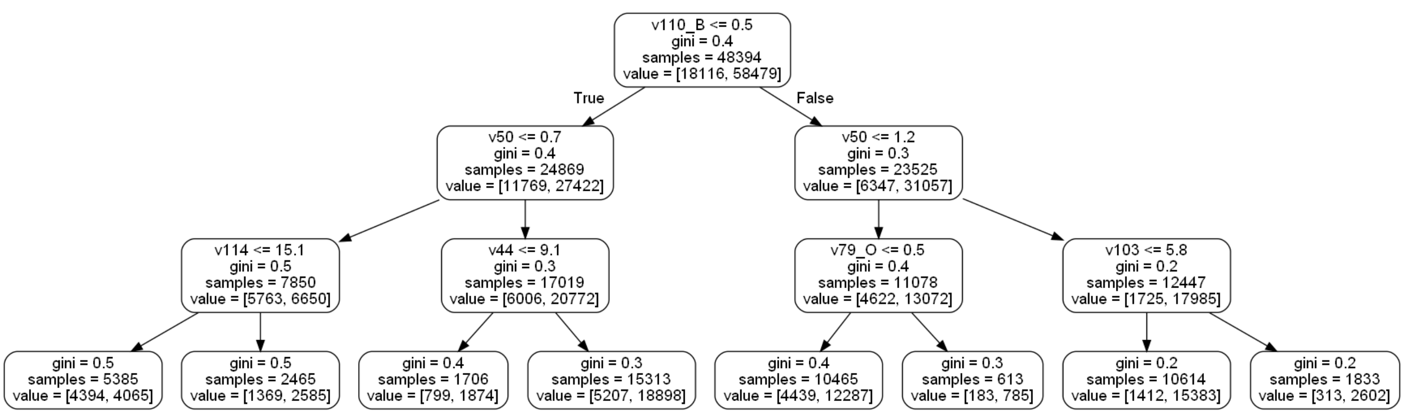
Random Forest Feature Importance - All Features (Figure 6)

Top Ten Features

Rank	Feature ID	Importances
1.	feature 44	(0.060483)
2.	feature 11	(0.027440)
3.	feature 9	(0.025312)
4.	feature 96	(0.024937)
5.	feature 35	(0.023831)
6.	feature 20	(0.023625)
7.	feature 29	(0.023525)
8.	feature 13	(0.022473)
9.	feature 0	(0.021958)
10.	feature 338	(0.006959)

****Example Tree from our Random Forest Base Model"**

The Figure below demonstrates a very small portion of the total Random Forest model. In this example, we intentionally limited the max_depth of the tree to be able to visualize each element. (Figure 7)



Random Forest Tree with max_depth=3 and n_iterations=10 (Figure 7)

Results

Validation Results

Model results are provided in Table 4. Base models were constructed with the parameters that were provided by Dr. Slater. Hypertuned models are listed by type and variation. We were able to provide log-loss and accuracy for XGBoost and Random Forest. SVM does not provide log-loss so it is excluded from the table. In addition to log-loss and accuracy, we have provided timing or estimated timing for some models. This allows us to evaluate the cost/accuracy trade-offs.

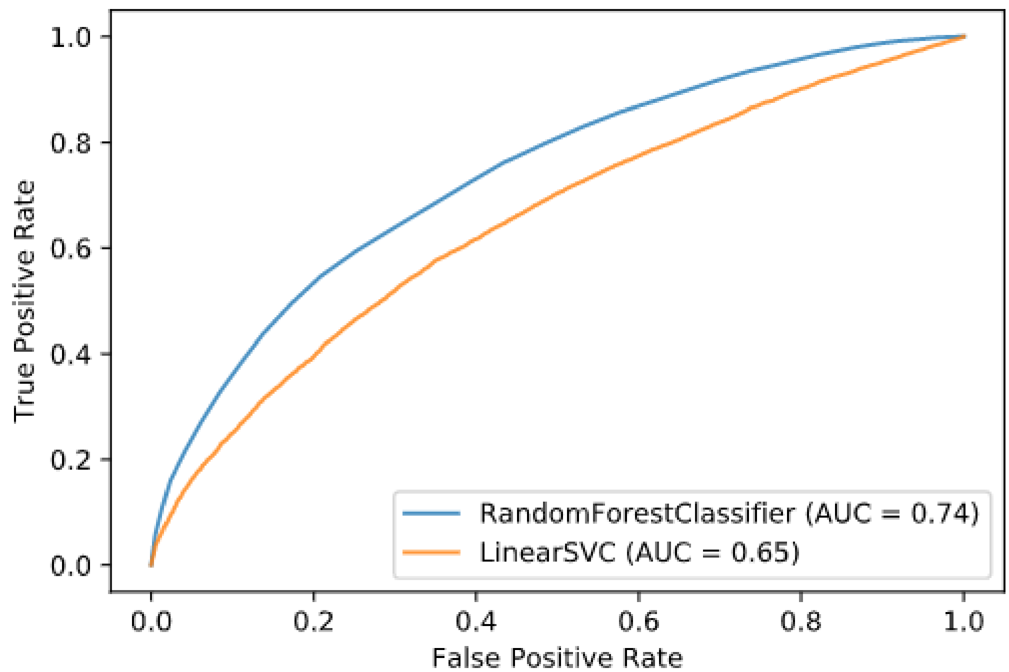
Of the Random Forest models, our best performer was also the most simple and took the least amount of processing time because we did not use RandomizedSearchCV. The number of iterations for all randomized models was n_iter=5 and the best_params were all very similar between the different cases. It's very possible that the default parameters just happened to be the best performing combination of tunable parameters for this dataset. Given the relatively small amount of feature importance and the large number of features, it's also possible that the test dataset was very similar to the train dataset in its lack of correlation or internal trend. Whatever the reason, we find that random forest outperforms the XGBoost and SVM models with this data. (Table 3)

Model	Log-Loss	Accuracy	Wall Time (Seconds)
XGBoost (Pre-Tuned Model)	0.585068	0.768330	8
XGBoost (RandomizedSearchCV)	0.469189	0.781848	1534
SVM, Entire DSet	NA	0.7608	1179.79
SVM, For_1000	NA	0.779	13.2
SVM, For_2000	NA	0.4075	28.39
SVM, For_5000	NA	0.7662	71.83
SVM, For_10000	NA	0.7599	161.57
Random Forest Base	0.248184	0.927400	41
RF Tuned, 1000 Entries	0.483830	0.786000	120
RF Tuned, 5000 Entries	0.258795	0.917200	420
RF Tuned, Full Dataset	0.264958	0.913960	540

Table of Model Performance and Results (Table 3)

AUC Comparisions for LinearSVC and Random Forest

Below in figure 1, is the comparision of the AUC performance for the LinearSVC and Random Forest model. This plot indicates that the Random Forest model trends towards more true positives than LinearSVC.decision_function (Figure 8)



LinearSVC and Random Forest Model Comparisions (Figure 8)

Conclusion

The best accuracy and log-loss values of our Random Forest (RF), Support Vector Machine (SVM), and XGBoost tuned models varied greatly depending on the hyperparameter tuning. Random Forest accuracy results were significantly higher than the other models. XGBoost and SVM were similar in their accuracy, but their execution time was significantly different. The main difference between all models appeared in the compute time required to ensure these results. The log-loss values for XGBoost were also higher than Random Forest. This difference in the predicted probability from the actual value in XGBoost and Random Forest makes our choice less difficult if you have the computing power. Random Forest appears to be the best model for this type of data set.

SVM is a useful model for small data sets that are highly dimensional. If you have a Big Data corpus, then XGBoost would be a good model. Random Forest works with categorical features very well and can handle high dimensional spaces and large numbers of training examples.

References

Terence Shin (2020), towards data science - All Machine Learning Models Explained in 6 Minutes

Breiman, L. (1996). Bagging Predictors. Machine Learning, 24, 123-140.

CJason Brownlee (2016), machine learning mastery - A Gentle Introduction to XGBoost for Applied Machine Learning.

Appendix

Code

In [3]:
```
import pandas as pd
import xgboost as xgb
import os
import time
import numpy as np
from sklearn.metrics import log_loss, accuracy_score
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from tabulate import tabulate
import pickle

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

from sklearn.tree import export_graphviz
from pprint import pprint

import math
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
from sklearn import model_selection

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
import sklearn.feature_selection as fs
from sklearn.model_selection import cross_val_score
```

In [7]:
```
# Set Directory for image files - Comment out if you are not LL

import os
os.chdir('C:\\SMU_Local\\SMU_T5_QTW_CapA\\QTW_7333\\Unit 7 and 8\\case_s
tudy_81_2\\CS8')
```

In [8]:
```
os.getcwd()
```

Out[8]:
```
'C:\\SMU_Local\\SMU_T5_QTW_CapA\\QTW_7333\\Unit 7 and 8\\case_study_81_
2\\CS8'
```

Load Data and Prepare for Modeling

In [4]:
```
# Load Data
# load data and separate target variable from dataset
train = pd.read_csv('Data/case_8.csv')
target = train['target']
train.drop(['target'],inplace=True, axis=1)
```

In [ ]:
```
pickle.dump(target, open("Pickle/target.pkl", "wb"))
```

In [5]:
```
# evaluate data types
train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114321 entries, 0 to 114320
Data columns (total 132 columns):
 #   Column  Dtype  
---  ------  -----  
 0   ID      int64  
 1   v1      float64
 2   v2      float64
 3   v3      object 
 4   v4      float64
 5   v5      float64
 6   v6      float64
 7   v7      float64
 8   v8      float64
 9   v9      float64
 10  v10     float64
 11  v11     float64
 12  v12     float64
 13  v13     float64
 14  v14     float64
 15  v15     float64
 16  v16     float64
 17  v17     float64
 18  v18     float64
 19  v19     float64
 20  v20     float64
 21  v21     float64
 22  v22     object 
 23  v23     float64
 24  v24     object 
 25  v25     float64
 26  v26     float64
 27  v27     float64
 28  v28     float64
 29  v29     float64
 30  v30     object 
 31  v31     object 
 32  v32     float64
 33  v33     float64
 34  v34     float64
 35  v35     float64
 36  v36     float64
 37  v37     float64
 38  v38     int64  
 39  v39     float64
 40  v40     float64
 41  v41     float64
 42  v42     float64
 43  v43     float64
 44  v44     float64
 45  v45     float64
 46  v46     float64
 47  v47     object 
 48  v48     float64
 49  v49     float64
 50  v50     float64
 51  v51     float64
```

52	v52	object
53	v53	float64
54	v54	float64
55	v55	float64
56	v56	object
57	v57	float64
58	v58	float64
59	v59	float64
60	v60	float64
61	v61	float64
62	v62	int64
63	v63	float64
64	v64	float64
65	v65	float64
66	v66	object
67	v67	float64
68	v68	float64
69	v69	float64
70	v70	float64
71	v71	object
72	v72	int64
73	v73	float64
74	v74	object
75	v75	object
76	v76	float64
77	v77	float64
78	v78	float64
79	v79	object
80	v80	float64
81	v81	float64
82	v82	float64
83	v83	float64
84	v84	float64
85	v85	float64
86	v86	float64
87	v87	float64
88	v88	float64
89	v89	float64
90	v90	float64
91	v91	object
92	v92	float64
93	v93	float64
94	v94	float64
95	v95	float64
96	v96	float64
97	v97	float64
98	v98	float64
99	v99	float64
100	v100	float64
101	v101	float64
102	v102	float64
103	v103	float64
104	v104	float64
105	v105	float64
106	v106	float64
107	v107	object
108	v108	float64

```
 109 v109    float64
 110 v110    object
 111 v111    float64
 112 v112    object
 113 v113    object
 114 v114    float64
 115 v115    float64
 116 v116    float64
 117 v117    float64
 118 v118    float64
 119 v119    float64
 120 v120    float64
 121 v121    float64
 122 v122    float64
 123 v123    float64
 124 v124    float64
 125 v125    object
 126 v126    float64
 127 v127    float64
 128 v128    float64
 129 v129    int64
 130 v130    float64
 131 v131    float64
dtypes: float64(108), int64(5), object(19)
memory usage: 115.1+ MB
```

In [6]:
```
# isolate object data type columns
train_object_dtype_cols = train.select_dtypes(include='object')
```

In [7]:
```
# review head of object columns
train_object_dtype_cols.head()
```

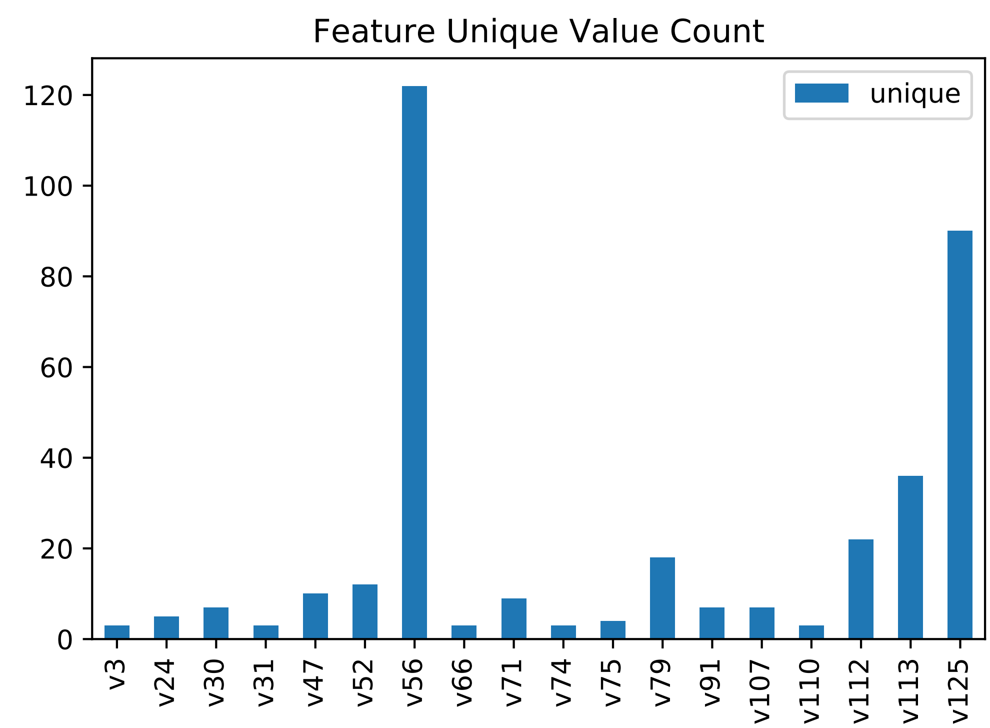
Out[7]:

	v3	v22	v24	v30	v31	v47	v52	v56	v66	v71	v74	v75	v79	v91	v107	v110	v112	v1
0	C	XDX	C	C	A	C	G	DI	C	F	B	D	E	A	E	B	O	
1	C	GUV	C	C	A	E	G	DY	A	F	B	D	D	B	B	A	U	
2	C	FQ	E	C	A	C	F	AS	A	B	B	B	E	G	C	B	S	
3	C	ACUE	D	C	B	C	H	BW	A	F	B	D	B	B	B	B	J	
4	C	HIT	E	C	A	I	H	BW	C	F	B	D	C	G	C	A	T	

In [8]:
```
# count unique values in object columns
train_object_dtype_cols.describe(include='all').loc['unique', :]
```

Out[8]:
```
v3          3
v22     18210
v24         5
v30         7
v31         3
v47        10
v52        12
v56       122
v66         3
v71         9
v74         3
v75         4
v79        18
v91         7
v107        7
v110        3
v112       22
v113       36
v125       90
Name: unique, dtype: object
```

In [9]:
```
# count unique values without v22 in object columns and plot
unique_series = train_object_dtype_cols.describe(include='all').loc['uni
que', :]
unique_df = pd.DataFrame(unique_series)
unique_df = unique_df.drop(['v22'])
ax = unique_df.plot.bar(title='Feature Unique Value Count')
```



In [52]:
```
# EDA on v22 feature

v22_counts = train_object_dtype_cols.groupby('v22').v22.count()
v22_counts.describe()
v22_counts.median()
v22_counts[v22_counts>125].describe()
```

Out[52]:
```
count      61.000000
mean      281.704918
std       431.560670
min       126.000000
25%       147.000000
50%       167.000000
75%       226.000000
max      2886.000000
Name: v22, dtype: float64
```

In [53]:

```
# Get counts of unique values in v22
val = train_object_dtype_cols['v22'].value_counts()
# identify values with counts > 125
y = val[val < 125].index
# replace values with count < 125 with NaN
train_object_dtype_cols['v22'] = train_object_dtype_cols['v22'].replace
({x:math.nan for x in y})
# Output v22 information after removing values less than 125
train_object_dtype_cols.info()
train_object_dtype_cols.groupby('v22').v22.count().describe()
```

```
2021-02-28 22:04:04,974 [35816] WARNING  py.warnings:110: [JupyterRequi
re] C:\Anaconda\lib\site-packages\ipykernel_launcher.py:6: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy



<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114321 entries, 0 to 114320
Data columns (total 19 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   v3      114321 non-null  object
 1   v22     17184 non-null   object
 2   v24     114321 non-null  object
 3   v30     114321 non-null  object
 4   v31     114321 non-null  object
 5   v47     114321 non-null  object
 6   v52     114321 non-null  object
 7   v56     114321 non-null  object
 8   v66     114321 non-null  object
 9   v71     114321 non-null  object
 10  v74     114321 non-null  object
 11  v75     114321 non-null  object
 12  v79     114321 non-null  object
 13  v91     114321 non-null  object
 14  v107    114321 non-null  object
 15  v110    114321 non-null  object
 16  v112    114321 non-null  object
 17  v113    114321 non-null  object
 18  v125    114321 non-null  object
dtypes: object(19)
memory usage: 16.6+ MB
```

Out[53]:

```
count      61.000000
mean      281.704918
std       431.560670
min       126.000000
25%       147.000000
50%       167.000000
75%       226.000000
max      2886.000000
Name: v22, dtype: float64
```

In [54]:

```
# one-hot encode remaining object columns
object_one_hot_df = pd.get_dummies(data=train_object_dtype_cols)
# Output one-hot encodeing results
object_one_hot_df.info(verbose=True)
# get list of columns that were one-hot encoded
drop_cols = train_object_dtype_cols.columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114321 entries, 0 to 114320
Data columns (total 425 columns):
 #   Column    Dtype
---  ------    -----
 0   v3_A      uint8
 1   v3_B      uint8
 2   v3_C      uint8
 3   v22_AAPP  uint8
 4   v22_ABF   uint8
 5   v22_ABOF  uint8
 6   v22_ACHJ  uint8
 7   v22_ACWE  uint8
 8   v22_ACXD  uint8
 9   v22_ADDF  uint8
 10  v22_ADGN  uint8
 11  v22_ADMI  uint8
 12  v22_ADMP  uint8
 13  v22_AFOZ  uint8
 14  v22_AFYU  uint8
 15  v22_AGDF  uint8
 16  v22_AGON  uint8
 17  v22_AGZT  uint8
 18  v22_AHE   uint8
 19  v22_AJQ   uint8
 20  v22_AMR   uint8
 21  v22_AWT   uint8
 22  v22_AXH   uint8
 23  v22_BLE   uint8
 24  v22_DJU   uint8
 25  v22_EJC   uint8
 26  v22_GBS   uint8
 27  v22_GEB   uint8
 28  v22_GEJ   uint8
 29  v22_HDD   uint8
 30  v22_HUU   uint8
 31  v22_HZE   uint8
 32  v22_JGY   uint8
 33  v22_KLZ   uint8
 34  v22_LIP   uint8
 35  v22_MNZ   uint8
 36  v22_MQE   uint8
 37  v22_NGS   uint8
 38  v22_NRT   uint8
 39  v22_NWG   uint8
 40  v22_NXE   uint8
 41  v22_OFD   uint8
 42  v22_PBC   uint8
 43  v22_PFR   uint8
 44  v22_PSE   uint8
 45  v22_PTJ   uint8
 46  v22_PTO   uint8
 47  v22_PWR   uint8
 48  v22_QKI   uint8
 49  v22_QKP   uint8
 50  v22_QVR   uint8
 51  v22_RIC   uint8
```

52	v22_ROZ	uint8
53	v22_TG	uint8
54	v22_TVR	uint8
55	v22_UAG	uint8
56	v22_VVI	uint8
57	v22_VZF	uint8
58	v22_WFT	uint8
59	v22_WNI	uint8
60	v22_WRI	uint8
61	v22_YEP	uint8
62	v22_YGJ	uint8
63	v22_YOD	uint8
64	v24_A	uint8
65	v24_B	uint8
66	v24_C	uint8
67	v24_D	uint8
68	v24_E	uint8
69	v30_A	uint8
70	v30_B	uint8
71	v30_C	uint8
72	v30_D	uint8
73	v30_E	uint8
74	v30_F	uint8
75	v30_G	uint8
76	v31_A	uint8
77	v31_B	uint8
78	v31_C	uint8
79	v47_A	uint8
80	v47_B	uint8
81	v47_C	uint8
82	v47_D	uint8
83	v47_E	uint8
84	v47_F	uint8
85	v47_G	uint8
86	v47_H	uint8
87	v47_I	uint8
88	v47_J	uint8
89	v52_A	uint8
90	v52_B	uint8
91	v52_C	uint8
92	v52_D	uint8
93	v52_E	uint8
94	v52_F	uint8
95	v52_G	uint8
96	v52_H	uint8
97	v52_I	uint8
98	v52_J	uint8
99	v52_K	uint8
100	v52_L	uint8
101	v56_A	uint8
102	v56_AA	uint8
103	v56_AB	uint8
104	v56_AC	uint8
105	v56_AE	uint8
106	v56_AF	uint8
107	v56_AG	uint8
108	v56_AH	uint8

109	v56_AI	uint8
110	v56_AJ	uint8
111	v56_AK	uint8
112	v56_AL	uint8
113	v56_AM	uint8
114	v56_AN	uint8
115	v56_AO	uint8
116	v56_AP	uint8
117	v56_AR	uint8
118	v56_AS	uint8
119	v56_AT	uint8
120	v56_AU	uint8
121	v56_AV	uint8
122	v56_AW	uint8
123	v56_AX	uint8
124	v56_AY	uint8
125	v56_AZ	uint8
126	v56_B	uint8
127	v56_BA	uint8
128	v56_BC	uint8
129	v56_BD	uint8
130	v56_BE	uint8
131	v56_BF	uint8
132	v56_BG	uint8
133	v56_BH	uint8
134	v56_BI	uint8
135	v56_BJ	uint8
136	v56_BK	uint8
137	v56_BL	uint8
138	v56_BM	uint8
139	v56_BN	uint8
140	v56_BO	uint8
141	v56_BP	uint8
142	v56_BQ	uint8
143	v56_BR	uint8
144	v56_BS	uint8
145	v56_BT	uint8
146	v56_BU	uint8
147	v56_BV	uint8
148	v56_BW	uint8
149	v56_BX	uint8
150	v56_BY	uint8
151	v56_BZ	uint8
152	v56_C	uint8
153	v56_CA	uint8
154	v56_CB	uint8
155	v56_CC	uint8
156	v56_CD	uint8
157	v56_CE	uint8
158	v56_CF	uint8
159	v56_CG	uint8
160	v56_CH	uint8
161	v56_CI	uint8
162	v56_CJ	uint8
163	v56_CK	uint8
164	v56_CL	uint8
165	v56_CM	uint8

166	v56_CN	uint8
167	v56_CO	uint8
168	v56_CP	uint8
169	v56_CQ	uint8
170	v56_CS	uint8
171	v56_CT	uint8
172	v56_CV	uint8
173	v56_CW	uint8
174	v56_CX	uint8
175	v56_CY	uint8
176	v56_CZ	uint8
177	v56_D	uint8
178	v56_DA	uint8
179	v56_DB	uint8
180	v56_DC	uint8
181	v56_DD	uint8
182	v56_DE	uint8
183	v56_DF	uint8
184	v56_DG	uint8
185	v56_DH	uint8
186	v56_DI	uint8
187	v56_DJ	uint8
188	v56_DK	uint8
189	v56_DL	uint8
190	v56_DM	uint8
191	v56_DN	uint8
192	v56_DO	uint8
193	v56_DP	uint8
194	v56_DQ	uint8
195	v56_DR	uint8
196	v56_DS	uint8
197	v56_DT	uint8
198	v56_DU	uint8
199	v56_DV	uint8
200	v56_DW	uint8
201	v56_DX	uint8
202	v56_DY	uint8
203	v56_DZ	uint8
204	v56_E	uint8
205	v56_F	uint8
206	v56_G	uint8
207	v56_H	uint8
208	v56_I	uint8
209	v56_L	uint8
210	v56_M	uint8
211	v56_N	uint8
212	v56_O	uint8
213	v56_P	uint8
214	v56_Q	uint8
215	v56_R	uint8
216	v56_T	uint8
217	v56_U	uint8
218	v56_V	uint8
219	v56_W	uint8
220	v56_X	uint8
221	v56_Y	uint8
222	v56_Z	uint8

223	v66_A	uint8
224	v66_B	uint8
225	v66_C	uint8
226	v71_A	uint8
227	v71_B	uint8
228	v71_C	uint8
229	v71_D	uint8
230	v71_F	uint8
231	v71_G	uint8
232	v71_I	uint8
233	v71_K	uint8
234	v71_L	uint8
235	v74_A	uint8
236	v74_B	uint8
237	v74_C	uint8
238	v75_A	uint8
239	v75_B	uint8
240	v75_C	uint8
241	v75_D	uint8
242	v79_A	uint8
243	v79_B	uint8
244	v79_C	uint8
245	v79_D	uint8
246	v79_E	uint8
247	v79_F	uint8
248	v79_G	uint8
249	v79_H	uint8
250	v79_I	uint8
251	v79_J	uint8
252	v79_K	uint8
253	v79_L	uint8
254	v79_M	uint8
255	v79_N	uint8
256	v79_O	uint8
257	v79_P	uint8
258	v79_Q	uint8
259	v79_R	uint8
260	v91_A	uint8
261	v91_B	uint8
262	v91_C	uint8
263	v91_D	uint8
264	v91_E	uint8
265	v91_F	uint8
266	v91_G	uint8
267	v107_A	uint8
268	v107_B	uint8
269	v107_C	uint8
270	v107_D	uint8
271	v107_E	uint8
272	v107_F	uint8
273	v107_G	uint8
274	v110_A	uint8
275	v110_B	uint8
276	v110_C	uint8
277	v112_A	uint8
278	v112_B	uint8
279	v112_C	uint8

280	v112_D	uint8
281	v112_E	uint8
282	v112_F	uint8
283	v112_G	uint8
284	v112_H	uint8
285	v112_I	uint8
286	v112_J	uint8
287	v112_K	uint8
288	v112_L	uint8
289	v112_M	uint8
290	v112_N	uint8
291	v112_O	uint8
292	v112_P	uint8
293	v112_Q	uint8
294	v112_R	uint8
295	v112_S	uint8
296	v112_T	uint8
297	v112_U	uint8
298	v112_V	uint8
299	v113_A	uint8
300	v113_AA	uint8
301	v113_AB	uint8
302	v113_AC	uint8
303	v113_AD	uint8
304	v113_AE	uint8
305	v113_AF	uint8
306	v113_AG	uint8
307	v113_AH	uint8
308	v113_AI	uint8
309	v113_AJ	uint8
310	v113_AK	uint8
311	v113_B	uint8
312	v113_C	uint8
313	v113_D	uint8
314	v113_E	uint8
315	v113_F	uint8
316	v113_G	uint8
317	v113_H	uint8
318	v113_I	uint8
319	v113_J	uint8
320	v113_L	uint8
321	v113_M	uint8
322	v113_N	uint8
323	v113_O	uint8
324	v113_P	uint8
325	v113_Q	uint8
326	v113_R	uint8
327	v113_S	uint8
328	v113_T	uint8
329	v113_U	uint8
330	v113_V	uint8
331	v113_W	uint8
332	v113_X	uint8
333	v113_Y	uint8
334	v113_Z	uint8
335	v125_A	uint8
336	v125_AA	uint8

337	v125_AB	uint8
338	v125_AC	uint8
339	v125_AD	uint8
340	v125_AE	uint8
341	v125_AF	uint8
342	v125_AG	uint8
343	v125_AH	uint8
344	v125_AI	uint8
345	v125_AJ	uint8
346	v125_AK	uint8
347	v125_AL	uint8
348	v125_AM	uint8
349	v125_AN	uint8
350	v125_AO	uint8
351	v125_AP	uint8
352	v125_AQ	uint8
353	v125_AR	uint8
354	v125_AS	uint8
355	v125_AT	uint8
356	v125_AU	uint8
357	v125_AV	uint8
358	v125_AW	uint8
359	v125_AX	uint8
360	v125_AY	uint8
361	v125_AZ	uint8
362	v125_B	uint8
363	v125_BA	uint8
364	v125_BB	uint8
365	v125_BC	uint8
366	v125_BD	uint8
367	v125_BE	uint8
368	v125_BF	uint8
369	v125_BG	uint8
370	v125_BH	uint8
371	v125_BI	uint8
372	v125_BJ	uint8
373	v125_BK	uint8
374	v125_BL	uint8
375	v125_BM	uint8
376	v125_BN	uint8
377	v125_BO	uint8
378	v125_BP	uint8
379	v125_BQ	uint8
380	v125_BR	uint8
381	v125_BS	uint8
382	v125_BT	uint8
383	v125_BU	uint8
384	v125_BV	uint8
385	v125_BW	uint8
386	v125_BX	uint8
387	v125_BY	uint8
388	v125_BZ	uint8
389	v125_C	uint8
390	v125_CA	uint8
391	v125_CB	uint8
392	v125_CC	uint8
393	v125_CD	uint8

```
 394 v125_CE   uint8
 395 v125_CF   uint8
 396 v125_CG   uint8
 397 v125_CH   uint8
 398 v125_CI   uint8
 399 v125_CJ   uint8
 400 v125_CK   uint8
 401 v125_CL   uint8
 402 v125_D    uint8
 403 v125_E    uint8
 404 v125_F    uint8
 405 v125_G    uint8
 406 v125_H    uint8
 407 v125_I    uint8
 408 v125_J    uint8
 409 v125_K    uint8
 410 v125_L    uint8
 411 v125_M    uint8
 412 v125_N    uint8
 413 v125_O    uint8
 414 v125_P    uint8
 415 v125_Q    uint8
 416 v125_R    uint8
 417 v125_S    uint8
 418 v125_T    uint8
 419 v125_U    uint8
 420 v125_V    uint8
 421 v125_W    uint8
 422 v125_X    uint8
 423 v125_Y    uint8
 424 v125_Z    uint8
dtypes: uint8(425)
memory usage: 46.3 MB
```

In [55]:

```
# drop one-hot encoded columns from dataframe
train = train.drop(drop_cols, axis=1)
```

In [56]:

```
# merge one-hot encoded columns to dataframe
frames = [train, object_one_hot_df]
train = pd.concat(frames,axis=1)
# Output train dataframe
train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114321 entries, 0 to 114320
Data columns (total 538 columns):
 #   Column    Dtype
---  ------    -----
 0   ID        int64
 1   v1        float64
 2   v2        float64
 3   v4        float64
 4   v5        float64
 5   v6        float64
 6   v7        float64
 7   v8        float64
 8   v9        float64
 9   v10       float64
 10  v11       float64
 11  v12       float64
 12  v13       float64
 13  v14       float64
 14  v15       float64
 15  v16       float64
 16  v17       float64
 17  v18       float64
 18  v19       float64
 19  v20       float64
 20  v21       float64
 21  v23       float64
 22  v25       float64
 23  v26       float64
 24  v27       float64
 25  v28       float64
 26  v29       float64
 27  v32       float64
 28  v33       float64
 29  v34       float64
 30  v35       float64
 31  v36       float64
 32  v37       float64
 33  v38       int64
 34  v39       float64
 35  v40       float64
 36  v41       float64
 37  v42       float64
 38  v43       float64
 39  v44       float64
 40  v45       float64
 41  v46       float64
 42  v48       float64
 43  v49       float64
 44  v50       float64
 45  v51       float64
 46  v53       float64
 47  v54       float64
 48  v55       float64
 49  v57       float64
 50  v58       float64
 51  v59       float64
```

52	v60	float64
53	v61	float64
54	v62	int64
55	v63	float64
56	v64	float64
57	v65	float64
58	v67	float64
59	v68	float64
60	v69	float64
61	v70	float64
62	v72	int64
63	v73	float64
64	v76	float64
65	v77	float64
66	v78	float64
67	v80	float64
68	v81	float64
69	v82	float64
70	v83	float64
71	v84	float64
72	v85	float64
73	v86	float64
74	v87	float64
75	v88	float64
76	v89	float64
77	v90	float64
78	v92	float64
79	v93	float64
80	v94	float64
81	v95	float64
82	v96	float64
83	v97	float64
84	v98	float64
85	v99	float64
86	v100	float64
87	v101	float64
88	v102	float64
89	v103	float64
90	v104	float64
91	v105	float64
92	v106	float64
93	v108	float64
94	v109	float64
95	v111	float64
96	v114	float64
97	v115	float64
98	v116	float64
99	v117	float64
100	v118	float64
101	v119	float64
102	v120	float64
103	v121	float64
104	v122	float64
105	v123	float64
106	v124	float64
107	v126	float64
108	v127	float64

```
109 v128      float64
110 v129      int64
111 v130      float64
112 v131      float64
113 v3_A      uint8
114 v3_B      uint8
115 v3_C      uint8
116 v22_AAPP  uint8
117 v22_ABF   uint8
118 v22_ABOF  uint8
119 v22_ACHJ  uint8
120 v22_ACWE  uint8
121 v22_ACXD  uint8
122 v22_ADDF  uint8
123 v22_ADGN  uint8
124 v22_ADMI  uint8
125 v22_ADMP  uint8
126 v22_AFOZ  uint8
127 v22_AFYU  uint8
128 v22_AGDF  uint8
129 v22_AGON  uint8
130 v22_AGZT  uint8
131 v22_AHE   uint8
132 v22_AJQ   uint8
133 v22_AMR   uint8
134 v22_AWT   uint8
135 v22_AXH   uint8
136 v22_BLE   uint8
137 v22_DJU   uint8
138 v22_EJC   uint8
139 v22_GBS   uint8
140 v22_GEB   uint8
141 v22_GEJ   uint8
142 v22_HDD   uint8
143 v22_HUU   uint8
144 v22_HZE   uint8
145 v22_JGY   uint8
146 v22_KLZ   uint8
147 v22_LIP   uint8
148 v22_MNZ   uint8
149 v22_MQE   uint8
150 v22_NGS   uint8
151 v22_NRT   uint8
152 v22_NWG   uint8
153 v22_NXE   uint8
154 v22_OFD   uint8
155 v22_PBC   uint8
156 v22_PFR   uint8
157 v22_PSE   uint8
158 v22_PTJ   uint8
159 v22_PTO   uint8
160 v22_PWR   uint8
161 v22_QKI   uint8
162 v22_QKP   uint8
163 v22_QVR   uint8
164 v22_RIC   uint8
165 v22_ROZ   uint8
```

```
166 v22_TG    uint8
167 v22_TVR   uint8
168 v22_UAG   uint8
169 v22_VVI   uint8
170 v22_VZF   uint8
171 v22_WFT   uint8
172 v22_WNI   uint8
173 v22_WRI   uint8
174 v22_YEP   uint8
175 v22_YGJ   uint8
176 v22_YOD   uint8
177 v24_A     uint8
178 v24_B     uint8
179 v24_C     uint8
180 v24_D     uint8
181 v24_E     uint8
182 v30_A     uint8
183 v30_B     uint8
184 v30_C     uint8
185 v30_D     uint8
186 v30_E     uint8
187 v30_F     uint8
188 v30_G     uint8
189 v31_A     uint8
190 v31_B     uint8
191 v31_C     uint8
192 v47_A     uint8
193 v47_B     uint8
194 v47_C     uint8
195 v47_D     uint8
196 v47_E     uint8
197 v47_F     uint8
198 v47_G     uint8
199 v47_H     uint8
200 v47_I     uint8
201 v47_J     uint8
202 v52_A     uint8
203 v52_B     uint8
204 v52_C     uint8
205 v52_D     uint8
206 v52_E     uint8
207 v52_F     uint8
208 v52_G     uint8
209 v52_H     uint8
210 v52_I     uint8
211 v52_J     uint8
212 v52_K     uint8
213 v52_L     uint8
214 v56_A     uint8
215 v56_AA    uint8
216 v56_AB    uint8
217 v56_AC    uint8
218 v56_AE    uint8
219 v56_AF    uint8
220 v56_AG    uint8
221 v56_AH    uint8
222 v56_AI    uint8
```

223	v56_AJ	uint8
224	v56_AK	uint8
225	v56_AL	uint8
226	v56_AM	uint8
227	v56_AN	uint8
228	v56_AO	uint8
229	v56_AP	uint8
230	v56_AR	uint8
231	v56_AS	uint8
232	v56_AT	uint8
233	v56_AU	uint8
234	v56_AV	uint8
235	v56_AW	uint8
236	v56_AX	uint8
237	v56_AY	uint8
238	v56_AZ	uint8
239	v56_B	uint8
240	v56_BA	uint8
241	v56_BC	uint8
242	v56_BD	uint8
243	v56_BE	uint8
244	v56_BF	uint8
245	v56_BG	uint8
246	v56_BH	uint8
247	v56_BI	uint8
248	v56_BJ	uint8
249	v56_BK	uint8
250	v56_BL	uint8
251	v56_BM	uint8
252	v56_BN	uint8
253	v56_BO	uint8
254	v56_BP	uint8
255	v56_BQ	uint8
256	v56_BR	uint8
257	v56_BS	uint8
258	v56_BT	uint8
259	v56_BU	uint8
260	v56_BV	uint8
261	v56_BW	uint8
262	v56_BX	uint8
263	v56_BY	uint8
264	v56_BZ	uint8
265	v56_C	uint8
266	v56_CA	uint8
267	v56_CB	uint8
268	v56_CC	uint8
269	v56_CD	uint8
270	v56_CE	uint8
271	v56_CF	uint8
272	v56_CG	uint8
273	v56_CH	uint8
274	v56_CI	uint8
275	v56_CJ	uint8
276	v56_CK	uint8
277	v56_CL	uint8
278	v56_CM	uint8
279	v56_CN	uint8

280	v56_CO	uint8
281	v56_CP	uint8
282	v56_CQ	uint8
283	v56_CS	uint8
284	v56_CT	uint8
285	v56_CV	uint8
286	v56_CW	uint8
287	v56_CX	uint8
288	v56_CY	uint8
289	v56_CZ	uint8
290	v56_D	uint8
291	v56_DA	uint8
292	v56_DB	uint8
293	v56_DC	uint8
294	v56_DD	uint8
295	v56_DE	uint8
296	v56_DF	uint8
297	v56_DG	uint8
298	v56_DH	uint8
299	v56_DI	uint8
300	v56_DJ	uint8
301	v56_DK	uint8
302	v56_DL	uint8
303	v56_DM	uint8
304	v56_DN	uint8
305	v56_DO	uint8
306	v56_DP	uint8
307	v56_DQ	uint8
308	v56_DR	uint8
309	v56_DS	uint8
310	v56_DT	uint8
311	v56_DU	uint8
312	v56_DV	uint8
313	v56_DW	uint8
314	v56_DX	uint8
315	v56_DY	uint8
316	v56_DZ	uint8
317	v56_E	uint8
318	v56_F	uint8
319	v56_G	uint8
320	v56_H	uint8
321	v56_I	uint8
322	v56_L	uint8
323	v56_M	uint8
324	v56_N	uint8
325	v56_O	uint8
326	v56_P	uint8
327	v56_Q	uint8
328	v56_R	uint8
329	v56_T	uint8
330	v56_U	uint8
331	v56_V	uint8
332	v56_W	uint8
333	v56_X	uint8
334	v56_Y	uint8
335	v56_Z	uint8
336	v66_A	uint8

337	v66_B	uint8
338	v66_C	uint8
339	v71_A	uint8
340	v71_B	uint8
341	v71_C	uint8
342	v71_D	uint8
343	v71_F	uint8
344	v71_G	uint8
345	v71_I	uint8
346	v71_K	uint8
347	v71_L	uint8
348	v74_A	uint8
349	v74_B	uint8
350	v74_C	uint8
351	v75_A	uint8
352	v75_B	uint8
353	v75_C	uint8
354	v75_D	uint8
355	v79_A	uint8
356	v79_B	uint8
357	v79_C	uint8
358	v79_D	uint8
359	v79_E	uint8
360	v79_F	uint8
361	v79_G	uint8
362	v79_H	uint8
363	v79_I	uint8
364	v79_J	uint8
365	v79_K	uint8
366	v79_L	uint8
367	v79_M	uint8
368	v79_N	uint8
369	v79_O	uint8
370	v79_P	uint8
371	v79_Q	uint8
372	v79_R	uint8
373	v91_A	uint8
374	v91_B	uint8
375	v91_C	uint8
376	v91_D	uint8
377	v91_E	uint8
378	v91_F	uint8
379	v91_G	uint8
380	v107_A	uint8
381	v107_B	uint8
382	v107_C	uint8
383	v107_D	uint8
384	v107_E	uint8
385	v107_F	uint8
386	v107_G	uint8
387	v110_A	uint8
388	v110_B	uint8
389	v110_C	uint8
390	v112_A	uint8
391	v112_B	uint8
392	v112_C	uint8
393	v112_D	uint8

```
394 v112_E    uint8
395 v112_F    uint8
396 v112_G    uint8
397 v112_H    uint8
398 v112_I    uint8
399 v112_J    uint8
400 v112_K    uint8
401 v112_L    uint8
402 v112_M    uint8
403 v112_N    uint8
404 v112_O    uint8
405 v112_P    uint8
406 v112_Q    uint8
407 v112_R    uint8
408 v112_S    uint8
409 v112_T    uint8
410 v112_U    uint8
411 v112_V    uint8
412 v113_A    uint8
413 v113_AA   uint8
414 v113_AB   uint8
415 v113_AC   uint8
416 v113_AD   uint8
417 v113_AE   uint8
418 v113_AF   uint8
419 v113_AG   uint8
420 v113_AH   uint8
421 v113_AI   uint8
422 v113_AJ   uint8
423 v113_AK   uint8
424 v113_B    uint8
425 v113_C    uint8
426 v113_D    uint8
427 v113_E    uint8
428 v113_F    uint8
429 v113_G    uint8
430 v113_H    uint8
431 v113_I    uint8
432 v113_J    uint8
433 v113_L    uint8
434 v113_M    uint8
435 v113_N    uint8
436 v113_O    uint8
437 v113_P    uint8
438 v113_Q    uint8
439 v113_R    uint8
440 v113_S    uint8
441 v113_T    uint8
442 v113_U    uint8
443 v113_V    uint8
444 v113_W    uint8
445 v113_X    uint8
446 v113_Y    uint8
447 v113_Z    uint8
448 v125_A    uint8
449 v125_AA   uint8
450 v125_AB   uint8
```

451	v125_AC	uint8
452	v125_AD	uint8
453	v125_AE	uint8
454	v125_AF	uint8
455	v125_AG	uint8
456	v125_AH	uint8
457	v125_AI	uint8
458	v125_AJ	uint8
459	v125_AK	uint8
460	v125_AL	uint8
461	v125_AM	uint8
462	v125_AN	uint8
463	v125_AO	uint8
464	v125_AP	uint8
465	v125_AQ	uint8
466	v125_AR	uint8
467	v125_AS	uint8
468	v125_AT	uint8
469	v125_AU	uint8
470	v125_AV	uint8
471	v125_AW	uint8
472	v125_AX	uint8
473	v125_AY	uint8
474	v125_AZ	uint8
475	v125_B	uint8
476	v125_BA	uint8
477	v125_BB	uint8
478	v125_BC	uint8
479	v125_BD	uint8
480	v125_BE	uint8
481	v125_BF	uint8
482	v125_BG	uint8
483	v125_BH	uint8
484	v125_BI	uint8
485	v125_BJ	uint8
486	v125_BK	uint8
487	v125_BL	uint8
488	v125_BM	uint8
489	v125_BN	uint8
490	v125_BO	uint8
491	v125_BP	uint8
492	v125_BQ	uint8
493	v125_BR	uint8
494	v125_BS	uint8
495	v125_BT	uint8
496	v125_BU	uint8
497	v125_BV	uint8
498	v125_BW	uint8
499	v125_BX	uint8
500	v125_BY	uint8
501	v125_BZ	uint8
502	v125_C	uint8
503	v125_CA	uint8
504	v125_CB	uint8
505	v125_CC	uint8
506	v125_CD	uint8
507	v125_CE	uint8

```
 508 v125_CF   uint8
 509 v125_CG   uint8
 510 v125_CH   uint8
 511 v125_CI   uint8
 512 v125_CJ   uint8
 513 v125_CK   uint8
 514 v125_CL   uint8
 515 v125_D    uint8
 516 v125_E    uint8
 517 v125_F    uint8
 518 v125_G    uint8
 519 v125_H    uint8
 520 v125_I    uint8
 521 v125_J    uint8
 522 v125_K    uint8
 523 v125_L    uint8
 524 v125_M    uint8
 525 v125_N    uint8
 526 v125_O    uint8
 527 v125_P    uint8
 528 v125_Q    uint8
 529 v125_R    uint8
 530 v125_S    uint8
 531 v125_T    uint8
 532 v125_U    uint8
 533 v125_V    uint8
 534 v125_W    uint8
 535 v125_X    uint8
 536 v125_Y    uint8
 537 v125_Z    uint8
dtypes: float64(108), int64(5), uint8(425)
memory usage: 144.9 MB
```

In [19]:
```
# Save training set to pickle
pickle.dump(train, open("Pickle/train.pkl", "wb"))
```

In [20]:
```
# Load train pickle for consistent shared data across models
train = pickle.load( open("Pickle/train.pkl", "rb" ) )
```

In [22]:
```
# create test/train split of data and target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train, target, test_
size=0.33, random_state=42)
```

In [24]:
```
#### Save split train and test sets for consistant model testing and tim
e savings.
pickle.dump(X_train, open("Pickle/X_train.pkl", "wb"))
pickle.dump(y_train, open("Pickle/y_train.pkl", "wb"))
pickle.dump(X_test, open("Pickle/X_test.pkl", "wb"))
pickle.dump(y_test, open("Pickle/y_test.pkl", "wb"))
```

In [111]:
```
#### Load split train and test sets for consistant model testing and tim
e savings.
X_train = pickle.load( open("Pickle/X_train.pkl", "rb"))
y_train = pickle.load( open("Pickle/y_train.pkl", "rb"))
X_test = pickle.load( open("Pickle/X_test.pkl", "rb"))
y_test = pickle.load( open("Pickle/y_test.pkl", "rb"))
```

In [ ]:
```
##############################################################
# End of Data Preparation

# Begin Models
#################################################################
```

In [ ]:
```
##############################################################
# XGBoost Model
#################################################################
```

In [ ]:
```
# BEGIN JR XGB RANDOMIZED SEARCH
```

In [ ]:
```
# combine data into DMatrix for XGBoost
xgtrain = xgb.DMatrix(X_train.values, y_train.values)
xgtest = xgb.DMatrix(X_test.values, y_test.values)
```

In [ ]:
```
clf = xgb.XGBClassifier()
```

In [ ]:
```
param_grid = {
        'silent': [False],
        'max_depth': [6, 10, 15, 20],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3],
        'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bylevel': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
        'gamma': [0, 0.25, 0.5, 1.0],
        'reg_lambda': [0.1, 1.0, 5.0, 10.0, 50.0, 100.0],
        'n_estimators': [100]}
```

In [ ]:
```
xgb_rs_clf = RandomizedSearchCV(clf, param_grid, n_iter=5,
                            n_jobs=-1, verbose=2, cv=5,
                            scoring='neg_log_loss', refit=True, random_s
tate=123)
```

In [ ]:
```
import time
print("Randomized search..")
search_time_start = time.time()
xgb_rs_clf.fit(X_train, y_train)
xgb_rs_elapsed = time.time() - search_time_start
print("Randomized search time:", xgb_rs_elapsed)
```

In [ ]:
```
pickle.dump(xgb_rs_elapsed, open("xgb_rs_elapsed.pkl", "wb"))
```

In [ ]:
```
xgb_rs_elapsed = pickle.load( open("xgb_rs_elapsed.pkl", "rb" ) )
```

In [ ]:
```
pickle.dump(xgb_rs_clf, open("xgb_rs_clf.pkl", "wb"))
```

In [ ]:
```
xgb_rs_clf = pickle.load( open("xgb_rs_clf.pkl", "rb" ) )
```

In [ ]:
```
print('The best combination of parameters, based on an evaluation of log
-loss is:')
xgb_rs_clf.best_params_
```

In [ ]:
```
best_params = xgb_rs_clf.best_params_
```

In [ ]:
```
print(f'The log-loss value for the best model is {round(xgb_rs_clf.best_
score_ * -1,4)}.')
```

In [ ]:
```
results = pd.DataFrame(xgb_rs_clf.cv_results_)
```

In [ ]:
```
pickle.dump(results, open("results.pkl", "wb"))
```

In [ ]:
```
results = pickle.load( open("results.pkl", "rb" ) )
```

In [ ]:
```
results.iloc[xgb_rs_clf.best_index_]
```

In [ ]:
```
xgb_test_preds = xgb_rs_clf.predict(X_test)
```

In [ ]:
```
pickle.dump(xgb_test_preds, open("xgb_test_preds.pkl", "wb"))
```

In [ ]:
```
xgb_test_preds = pickle.load( open("xgb_test_preds.pkl", "rb" ) )
```

In [ ]:
```
print(accuracy_score(y_test,np.rint(xgb_test_preds)))
```

In [ ]:
```
results_summary = pd.DataFrame([['XGBoost', 'RandomSearchCV',xgb_rs_clf.
best_score_ * -1,accuracy_score(y_test,np.rint(xgb_test_preds)),xgb_rs_e
lapsed ]],columns=['Model', 'Tuning', 'log loss', 'accuracy', 'time'])
```

In [ ]:
```
#####   END JR XGB RANDOMIZED SEARCH
```

In [ ]:
```
##### BEGIN JR XGB ORIGINAL EXAMPLE
```

In [ ]:
```
print('Fit the model...')
# XGBoost params:
xgboost_params = {
   "objective": "binary:logistic",
   "booster": "gbtree",
   "eval_metric": "logloss",
   "eta": 0.01,
   "subsample": 0.5,
   "colsample_bytree": 0.5,
   "max_depth": 3
}
boost_round = 50

xgb_clf_start = time.time()
xgb_clf = xgb.train(xgboost_params,xgtrain,num_boost_round=boost_round,v
erbose_eval=True,maximize=False)
xgb_clf_elapsed = time.time() - xgb_clf_start
```

In [ ]:
```
pickle.dump(xgb_clf_elapsed, open("xgb_clf_elapsed.pkl", "wb"))
```

In [ ]:
```
xgb_clf_elapsed = pickle.load( open("xgb_clf_elapsed.pkl", "rb" ) )
```

In [ ]:
```
#Make predict
print('Predict...')

xgb_test_preds_orig = clf.predict(xgtest, ntree_limit=clf.best_iteration
)

# Save results
```

In [ ]:
```
pickle.dump(xgb_clf, open("xgb_clf.pkl", "wb"))
```

In [ ]:
```
xgb_clf = pickle.load( open("xgb_clf.pkl", "rb" ) )
```

In [ ]:
```
print(log_loss(y_test,xgb_test_preds_orig))
print(accuracy_score(y_test,np.rint(xgb_test_preds_orig)))
```

In [ ]:
```
results_summary = results_summary.append(pd.DataFrame([['XGBoost','Base'
,log_loss(y_test,xgb_test_preds_orig),accuracy_score(y_test,np.rint(xgb_
test_preds_orig)), xgb_clf_elapsed ]],columns=['Model', 'Tuning', 'log l
oss', 'accuracy', 'time']))
```

In [ ]:
```
results_summary = results_summary.reset_index()
```

In [ ]:
```
results_summary
```

In [ ]:
```
pickle.dump(results_summary, open("jr_results_summary.pkl", "wb"))
```

In [ ]:
```
######  END XGB XGB ORIGINAL EXAMPLE
```

In [ ]:
```python
##### BEGIN XGB VISUALIZATIONS
```

In [ ]:
```python
import matplotlib.pyplot as plt
from xgboost import plot_tree
```

In [ ]:
```python
fig, ax = plt.subplots(figsize=(30, 30))
plot_tree(xgb_rs_clf.best_estimator_, num_trees=1, ax=ax)
plt.show()
```

In [ ]:
```python
from yellowbrick.classifier import ClassificationReport, ClassPrediction
Error
```

In [ ]:
```python
report = ClassificationReport(xgb_rs_clf, size=(1080, 720), classes=[0,1
])

report.score(X_test, y_test)
c = report.poof()
```

In [ ]:
```python
error = ClassPredictionError(xgb_rs_clf, size=(1080, 720), classes=[0,1
])

error.score(X_test, y_test)
e = error.poof()
```

In [ ]:
```python
#### END XGB VISUALIZATIONS
```

In [ ]:
```python
##############################################################
# SVM model
#################################################################
```

In [146]:
```python
#### Load split train and test sets for consistant model testing and tim
e savings.
X_train = pickle.load( open("Pickle/X_train.pkl", "rb"))
y_train = pickle.load( open("Pickle/y_train.pkl", "rb"))
X_test = pickle.load( open("Pickle/X_test.pkl", "rb"))
y_test = pickle.load( open("Pickle/y_test.pkl", "rb"))
```

In [142]:

```
#####################
###   Original SVM Code from Dr. Slater with Added Timing
#########################################################


start = time.time()
#####################################
# WARNING THIS TAKES AN HOUR TO RUN #
# Using LinearSVC for faster returns#
#####################################

svm = LinearSVC(verbose=True, random_state=42)
svm.fit(X_train, y_train)

end = time.time()
LinearSVC_time=round((end-start),2)
LinearSVC_time
```

```
[LibLinear]

2021-03-01 00:26:57,492 [35816] WARNING  py.warnings:110: [JupyterRequi
re] C:\Anaconda\lib\site-packages\sklearn\svm\_base.py:977: Convergence
Warning: Liblinear failed to converge, increase the number of iteration
s.
  "the number of iterations.", ConvergenceWarning)
```

Out[142]:

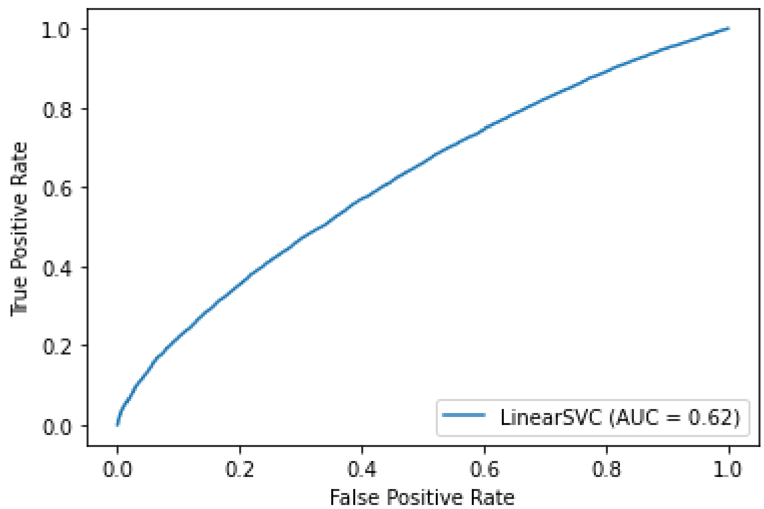
```
55.44
```

In [144]:

```
# Predict using the model and Measure Accuracy

X_pred=svm.predict(X_test)

#from sklearn.metrics import accuracy_score
svm_base_accuracy = accuracy_score(y_test,X_pred)
print(svm_base_accuracy)
```

```
0.5046122037851879
```

In [145]:
```
# Plot the SCM curve
svm_disp = plot_roc_curve(svm, X_test, y_test)
```



In [ ]:
```
########################
#### Begin MM CODE
#################################
```

In [ ]:
```
# FULL DATASET TUNED SVM MODEL
# Ignore to save time - Picked Models are available

# Gridsearch to determine the value of C
param_grid = {'C': [0.001,0.01,0.1],
              'loss': ['hinge', 'squared_hinge'],
              'penalty' : ['l2'],
              'dual' : [True,False],
              'tol': [0.00001,0.01], #0.0001 is the Default
              'max_iter': [100],
             }

SVC_Linear = LinearSVC(random_state=42)
CV_svc = GridSearchCV(estimator = SVC_Linear, param_grid=param_grid, cv=
5, n_jobs =-1,verbose=1)
Start=time.time()
CV_svc_mod = CV_svc.fit(X_train, y_train)
Stop=time.time()
Time1=Stop-Start
Time1
pkl_filename = "Pickle/CV_SVM_Linear.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(CV_svc_mod, file)
```

In [ ]:
```
Time1
```

In [112]:
```
# Load CV_svc_mod
#with open("C://Users/18322/OneDrive - Southern Methodist University/Des
ktop/QOW/week7/case study 8/case_study_81_2/CV_SVM_Linear.pkl", 'rb') as
file:
#     CV_svc = pickle.load(file)

CV_svc_mod = pickle.load( open("Pickle/CV_SVM_Linear.pkl", "rb" ) )
```

In [113]:
```
svc_gridsearch = pd.DataFrame(CV_svc_mod.cv_results_)
svc_columns = [
    "param_C",
    "param_loss",
    "param_dual",
    "param_tol",
    "param_max_iter",
    "mean_fit_time",
    "mean_test_score",
    "rank_test_score"

]
```

In [114]:
```
svc_gridsearch[svc_columns].sort_values(by="rank_test_score").head(10)
```

Out[114]:

	param_C	param_loss	param_dual	param_tol	param_max_iter	mean_fit_time	mean_test_s
6	0.001	squared_hinge	False	1e-05	100	872.557506	0.77
2	0.001	squared_hinge	True	1e-05	100	94.016914	0.77
3	0.001	squared_hinge	True	0.01	100	77.806056	0.77
7	0.001	squared_hinge	False	0.01	100	119.676386	0.77
23	0.1	squared_hinge	False	0.01	100	145.308221	0.77
14	0.01	squared_hinge	False	1e-05	100	1386.344374	0.77
15	0.01	squared_hinge	False	0.01	100	269.671581	0.77
22	0.1	squared_hinge	False	1e-05	100	1003.456923	0.77
10	0.01	squared_hinge	True	1e-05	100	207.509046	0.77
11	0.01	squared_hinge	True	0.01	100	214.311796	0.77

In [ ]:
```
print('Best Accuracy:', CV_svc_mod.best_score_)
```

In [115]:

```
# Create Dataframe of 1000 Rows
Xtrain_1000 = pd.DataFrame.sample(X_train, n=1000, random_state=123)
ytrain_1000 = pd.DataFrame.sample(y_train,n=1000, random_state=123)
Xtest_1000 = pd.DataFrame.sample(X_test,n=1000, random_state=123)
ytest_1000 = pd.DataFrame.sample(y_test,n=1000, random_state=123)
print('Training Features 1000:', Xtrain_1000.shape)
print('Training Labels 1000:', ytrain_1000.shape)
print('Testing Features 1000:', Xtest_1000.shape)
print('Testing Labels 1000:', ytest_1000.shape)
```

```
Training Features 1000: (1000, 538)
Training Labels 1000: (1000,)
Testing Features 1000: (1000, 538)
Testing Labels 1000: (1000,)
```

In [137]:

```
#1000 Rows
# Gridsearch to determine the value of C
param_grid = {'C': [0.001,0.01,0.1],
              'loss': ['hinge', 'squared_hinge'],
              'penalty' : ['l2'],
              'dual' : [True,False],
              'tol': [0.00001,0.01], #0.0001 is the Default
              'max_iter': [100],
             }

SVC_Linear = LinearSVC(random_state=42)
CV_svc = GridSearchCV(estimator = SVC_Linear, param_grid=param_grid, cv=
5, n_jobs =-1,verbose=1)
Start=time.time()
CV_svc_mod_1000 = CV_svc.fit(Xtrain_1000, ytrain_1000)
Stop=time.time()
Time2=Stop-Start
Time2
pkl_filename = "Pickle/CV_SVM_Linear_1000.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(CV_svc_mod_1000, file)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:    1.3s finished
```

In [ ]:

```
Time2
```

In [138]:

```
#1000
#with open("C://Users/18322/OneDrive - Southern Methodist University/Des
ktop/QOW/week7/case study 8/case_study_81_2/CV_SVM_Linear_1000.pkl", 'r
b') as file:
#    CV_svc = pickle.load(file)

CV_svc_mod_1000 = pickle.load( open("Pickle/CV_SVM_Linear_1000.pkl", "r
b" ) )
```

In [139]:
```
svc_gridsearch = pd.DataFrame(CV_svc_mod_1000.cv_results_)
svc_columns = [
    "param_C",
    "param_loss",
    "param_dual",
    "param_tol",
    "param_max_iter",
    "mean_fit_time",
    "mean_test_score",
    "rank_test_score"
]
```

In [140]:
```
#for 1000 samples
svc_gridsearch[svc_columns].sort_values(by="rank_test_score").head(10)
```

Out[140]:

	param_C	param_loss	param_dual	param_tol	param_max_iter	mean_fit_time	mean_test_s
23	0.1	squared_hinge	False	0.01	100	0.044880	C
15	0.01	squared_hinge	False	0.01	100	0.087365	C
7	0.001	squared_hinge	False	0.01	100	0.058641	C
6	0.001	squared_hinge	False	1e-05	100	0.117089	C
14	0.01	squared_hinge	False	1e-05	100	0.164079	C
22	0.1	squared_hinge	False	1e-05	100	0.122875	C
10	0.01	squared_hinge	True	1e-05	100	0.158082	C
11	0.01	squared_hinge	True	0.01	100	0.165455	C
1	0.001	hinge	True	0.01	100	0.156181	C
0	0.001	hinge	True	1e-05	100	0.143215	C

In [ ]:
```
print('Best Accuracy:', CV_svc_mod_1000.best_score_)
```

In [121]:
```
#2000
# Create Dataframe of 2000 Rows
Xtrain_2000 = pd.DataFrame.sample(X_train, n=2000, random_state=123)
ytrain_2000 = pd.DataFrame.sample(y_train,n=2000, random_state=123)
Xtest_2000 = pd.DataFrame.sample(X_test,n=2000, random_state=123)
ytest_2000 = pd.DataFrame.sample(y_test,n=2000, random_state=123)
print('Training Features 2000:', Xtrain_2000.shape)
print('Training Labels 2000:', ytrain_2000.shape)
print('Testing Features 2000:', Xtest_2000.shape)
print('Testing Labels 2000:', ytest_2000.shape)
```

```
Training Features 2000: (2000, 538)
Training Labels 2000: (2000,)
Testing Features 2000: (2000, 538)
Testing Labels 2000: (2000,)
```

In [122]:

```
#2000
# Gridsearch to determine the value of C
param_grid = {'C': [0.001,0.01,0.1],
              'loss': ['hinge', 'squared_hinge'],
              'penalty' : ['l2'],
              'dual' : [True,False],
              'tol': [0.00001,0.01], #0.0001 is the Default
              'max_iter': [100],
             }

SVC_Linear = LinearSVC(random_state=42)
CV_svc = GridSearchCV(estimator = SVC_Linear, param_grid=param_grid, cv=
5, n_jobs =-1,verbose=1)
Start=time.time()
CV_svc_mod_2000 = CV_svc.fit(Xtrain_2000, ytrain_2000)
Stop=time.time()
Time3=Stop-Start
Time3
pkl_filename = "Pickle/CV_SVM_Linear_2000.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(CV_svc_mod_2000, file)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    8.5s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:   10.6s finished
2021-03-01 00:17:45,154 [35816] WARNING  py.warnings:110: [JupyterRequi
re] C:\Anaconda\lib\site-packages\sklearn\svm\_base.py:977: Convergence
Warning: Liblinear failed to converge, increase the number of iteration
s.
  "the number of iterations.", ConvergenceWarning)
```

In [ ]:

```
Time3
```

In [123]:

```
#2000
#with open("C://Users/18322/OneDrive - Southern Methodist University/Des
ktop/QOW/week7/case study 8/case_study_81_2/CV_SVM_Linear_2000.pkl", 'r
b') as file:
#    CV_svc = pickle.load(file)

CV_svc_mod_2000 = pickle.load( open("Pickle/CV_SVM_Linear_2000.pkl", "r
b" ) )
```

In [124]:
```
svc_gridsearch = pd.DataFrame(CV_svc_mod_2000.cv_results_)
svc_columns = [
    "param_C",
    "param_loss",
    "param_dual",
    "param_tol",
    "param_max_iter",
    "mean_fit_time",
    "mean_test_score",
    "rank_test_score"
]
```

In [126]:
```
svc_gridsearch[svc_columns].sort_values(by="rank_test_score").head(10)
```

Out[126]:

	param_C	param_loss	param_dual	param_tol	param_max_iter	mean_fit_time	mean_test_s
6	0.001	squared_hinge	False	1e-05	100	0.417046	0.7
23	0.1	squared_hinge	False	0.01	100	0.132361	0.7
15	0.01	squared_hinge	False	0.01	100	0.134945	0.7
7	0.001	squared_hinge	False	0.01	100	0.139359	0.7
14	0.01	squared_hinge	False	1e-05	100	0.567534	0.7
22	0.1	squared_hinge	False	1e-05	100	0.492509	0.7
19	0.1	squared_hinge	True	0.01	100	0.442345	0.7
18	0.1	squared_hinge	True	1e-05	100	0.464916	0.7
17	0.1	hinge	True	0.01	100	0.449724	0.7
16	0.1	hinge	True	1e-05	100	0.507707	0.7

In [ ]:
```
print('Best Accuracy:', CV_svc_mod_2000.best_score_)
```

In [127]:
```
#5000
# Create Dataframe of 5000 Rows
Xtrain_5000 = pd.DataFrame.sample(X_train, n=5000, random_state=123)
ytrain_5000 = pd.DataFrame.sample(y_train,n=5000, random_state=123)
Xtest_5000 = pd.DataFrame.sample(X_test,n=5000, random_state=123)
ytest_5000 = pd.DataFrame.sample(y_test,n=5000, random_state=123)
print('Training Features 5000:', Xtrain_5000.shape)
print('Training Labels 5000:', ytrain_5000.shape)
print('Testing Features 5000:', Xtest_5000.shape)
print('Testing Labels 5000:', ytest_5000.shape)
```

```
Training Features 5000: (5000, 538)
Training Labels 5000: (5000,)
Testing Features 5000: (5000, 538)
Testing Labels 5000: (5000,)
```

In [ ]:
```
# Gridsearch to determine the value of C
param_grid = {'C': [0.001,0.01,0.1],
              'loss': ['hinge', 'squared_hinge'],
              'penalty' : ['l2'],
              'dual' : [True,False],
              'tol': [0.00001,0.01], #0.0001 is the Default
              'max_iter': [100],
             }

SVC_Linear = LinearSVC(random_state=42)
CV_svc = GridSearchCV(estimator = SVC_Linear, param_grid=param_grid, cv=
5, n_jobs =-1,verbose=1)
Start=time.time()
CV_svc_mod_5000 = CV_svc.fit(Xtrain_5000, ytrain_5000)
Stop=time.time()
Time4=Stop-Start
Time4
pkl_filename = "Pickle/CV_SVM_Linear_5000.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(CV_svc_mod_5000, file)
```

In [ ]:
```
Time4
```

In [128]:
```
#with open("C://Users/18322/OneDrive - Southern Methodist University/Des
ktop/QOW/week7/case study 8/case_study_81_2/CV_SVM_Linear_5000.pkl", 'r
b') as file:
#    CV_svc = pickle.load(file)

CV_svc_mod_5000 = pickle.load( open("Pickle/CV_SVM_Linear_5000.pkl", "r
b" ) )
```

In [129]:
```
svc_gridsearch = pd.DataFrame(CV_svc_mod_5000.cv_results_)
svc_columns = [
    "param_C",
    "param_loss",
    "param_dual",
    "param_tol",
    "param_max_iter",
    "mean_fit_time",
    "mean_test_score",
    "rank_test_score"
]
```

In [130]:
```python
svc_gridsearch[svc_columns].sort_values(by="rank_test_score").head(10)
```

Out[130]:

	param_C	param_loss	param_dual	param_tol	param_max_iter	mean_fit_time	mean_test_s
6	0.001	squared_hinge	False	1e-05	100	1.029846	0.
22	0.1	squared_hinge	False	1e-05	100	0.939270	0.
23	0.1	squared_hinge	False	0.01	100	0.230186	0.
15	0.01	squared_hinge	False	0.01	100	0.244742	0.
7	0.001	squared_hinge	False	0.01	100	0.227392	0.
14	0.01	squared_hinge	False	1e-05	100	1.177088	0.
0	0.001	hinge	True	1e-05	100	1.197936	0.
1	0.001	hinge	True	0.01	100	1.296276	0.
8	0.01	hinge	True	1e-05	100	1.226919	0.
9	0.01	hinge	True	0.01	100	1.168087	0.

In [ ]:
```python
print('Best Accuracy:', CV_svc_mod_5000.best_score_)
```

In [131]:
```python
#10000
# Create Dataframe of 10000 Rows
Xtrain_10000 = pd.DataFrame.sample(X_train, n=10000, random_state=123)
ytrain_10000 = pd.DataFrame.sample(y_train,n=10000, random_state=123)
Xtest_10000 = pd.DataFrame.sample(X_test,n=10000, random_state=123)
ytest_10000 = pd.DataFrame.sample(y_test,n=10000, random_state=123)
print('Training Features 10000:', Xtrain_10000.shape)
print('Training Labels 10000:', ytrain_10000.shape)
print('Testing Features 10000:', Xtest_10000.shape)
print('Testing Labels 10000:', ytest_10000.shape)
```

```
Training Features 10000: (10000, 538)
Training Labels 10000: (10000,)
Testing Features 10000: (10000, 538)
Testing Labels 10000: (10000,)
```

In [133]:
```
# Gridsearch to determine the value of C
param_grid = {'C': [0.001,0.01,0.1],
              'loss': ['hinge', 'squared_hinge'],
              'penalty' : ['l2'],
              'dual' : [True,False],
              'tol': [0.00001,0.01], #0.0001 is the Default
              'max_iter': [100],
             }

SVC_Linear = LinearSVC(random_state=42)
CV_svc = GridSearchCV(estimator = SVC_Linear, param_grid=param_grid, cv=
5, n_jobs =-1,verbose=1)
Start=time.time()
CV_svc_mod_10000 = CV_svc.fit(Xtrain_10000, ytrain_10000)
Stop=time.time()
Time5=Stop-Start
Time5
pkl_filename = "Pickle/CV_SVM_Linear_10000.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(CV_svc_mod_10000, file)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.5s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:   14.0s finished
```

In [ ]:
```
Time5
```

In [134]:
```
#with open("C://Users/18322/OneDrive - Southern Methodist University/Des
ktop/QOW/week7/case study 8/case_study_81_2/CV_SVM_Linear_10000.pkl", 'r
b') as file:
#    CV_svc = pickle.load(file)

CV_svc_mod_10000 = pickle.load( open("Pickle/CV_SVM_Linear_10000.pkl",
"rb" ) )
```

In [135]:
```
svc_gridsearch = pd.DataFrame(CV_svc_mod_10000.cv_results_)
svc_columns = [
    "param_C",
    "param_loss",
    "param_dual",
    "param_tol",
    "param_max_iter",
    "mean_fit_time",
    "mean_test_score",
    "rank_test_score"
]
```

In [136]:
```
svc_gridsearch[svc_columns].sort_values(by="rank_test_score").head(10)
```

Out[136]:

	param_C	param_loss	param_dual	param_tol	param_max_iter	mean_fit_time	mean_test_s
14	0.01	squared_hinge	False	1e-05	100	1.248989	0.`
22	0.1	squared_hinge	False	1e-05	100	1.117655	0.`
6	0.001	squared_hinge	False	1e-05	100	1.184583	0.`
23	0.1	squared_hinge	False	0.01	100	0.249406	0.`
15	0.01	squared_hinge	False	0.01	100	0.241507	0.`
7	0.001	squared_hinge	False	0.01	100	0.262091	0.`
19	0.1	squared_hinge	True	0.01	100	1.260491	0.`
18	0.1	squared_hinge	True	1e-05	100	1.278927	0.`
10	0.01	squared_hinge	True	1e-05	100	1.309391	0.`
11	0.01	squared_hinge	True	0.01	100	1.259128	0.`

In [ ]:
```
print('Best Accuracy:', CV_svc_mod_10000.best_score_)
```

In [5]:
```
##############################################################
# Random Forest Model
#################################################################
```

In [6]:

```
#######################
# RF Base Model
########################

#from sklearn.ensemble import RandomForestClassifier

# Full Dataset used for Base Model.
# Default max_levels is None, so the tree is very large

rf_base = RandomForestClassifier(n_estimators=10, random_state=123 )

start = time.time()
rf_base.fit(X_train, y_train)

end = time.time()
rf_base_time=round((end-start),2)
rf_base_time
```

```
-----------------------------------------------------------------------
----
NameError                              Traceback (most recent call l
ast)
<ipython-input-6-cda592a95acd> in <module>
      8 # Default max_levels is None, so the tree is very large
      9
---> 10 rf_base = RandomForestClassifier(n_estimators=10, random_state=
123 )
     11
     12 start = time.time()

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

In [63]:

```
rf_base_preds = rf_base.predict_proba(X_test)
```

In [64]:

```
rf_base_log_loss = log_loss(y_test,rf_base_preds[:,1]) # each column is
 class probability,
print(rf_base_log_loss)
rf_base_accuracy = accuracy_score(y_test,np.rint(rf_base_preds[:,1]))
print(rf_base_accuracy)
```

```
0.9337292313310098
0.7515241478025765
```

In [65]:

```
##############################################################
# Begin Hypertuning for Random Forest
#################################################################
```

In [66]:
```
# Citation:  Thank you to Will Koehrsen and Towards Data Science. RF Hyp
ertuning code is borrowed from his example.
# https://towardsdatascience.com/hyperparameter-tuning-the-random-forest
-in-python-using-scikit-learn-28d2aa77dd74

# Note - different from tutorial, we are using rf=RandomForestClassifier
(), not RandomForestRegressor()
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters Used by Base Model:\n')
pprint(rf_base.get_params())
```

```
Parameters Used by Base Model:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 10,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 123,
 'verbose': 0,
 'warm_start': False}
```

In [67]:
```
# Define Random Grid Parameters and use RandomizedSearchCV to choose
# different combinations of parameters for different sizes of datasets

#from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [100,500,1000,1500]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [10, 100, 200,300,400,500,600,700,800,900,1000]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print('Random Grid Parameters:\n')
pprint(random_grid)
```

```
Random Grid Parameters:

{'bootstrap': [True, False],
 'max_depth': [10, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [100, 500, 1000, 1500]}
```

In [68]:
```
# Adapt variable names for this example
train_features = X_train
train_labels = y_train
test_features = X_test
test_labels = y_test
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (76595, 538)
Training Labels Shape: (76595,)
Testing Features Shape: (37726, 538)
Testing Labels Shape: (37726,)
```

In [69]:
```
# Create Smaller Dataframe of 1000 Rows
train_features_1000 = pd.DataFrame.sample(X_train, n=1000, random_state=
123)
train_labels_1000 = pd.DataFrame.sample(y_train,n=1000, random_state=123
)
test_features_1000 = pd.DataFrame.sample(X_test,n=1000, random_state=123
)
test_labels_1000 = pd.DataFrame.sample(y_test,n=1000, random_state=123)
print('Training Features 1000:', train_features_1000.shape)
print('Training Labels 1000:', train_labels_1000.shape)
print('Testing Features 1000:', test_features_1000.shape)
print('Testing Labels 1000:', test_labels_1000.shape)
```

```
Training Features 1000: (1000, 538)
Training Labels 1000: (1000,)
Testing Features 1000: (1000, 538)
Testing Labels 1000: (1000,)
```

In [70]:
```
# Create Medium DataFrame of 5000 Rows
train_features_5000 = pd.DataFrame.sample(X_train, n=5000, random_state=
123)
train_labels_5000 = pd.DataFrame.sample(y_train,n=5000, random_state=123
)
test_features_5000 = pd.DataFrame.sample(X_test,n=5000, random_state=123
)
test_labels_5000 = pd.DataFrame.sample(y_test,n=5000, random_state=123)
print('Training Features 5000:', train_features_5000.shape)
print('Training Labels 5000:', train_labels_5000.shape)
print('Testing Features 5000:', test_features_5000.shape)
print('Testing Labels 5000:', test_labels_5000.shape)
```

```
Training Features 5000: (5000, 538)
Training Labels 5000: (5000,)
Testing Features 5000: (5000, 538)
Testing Labels 5000: (5000,)
```

In [71]:
```
# Original Code Commented - Using Pickled Models

# Use the random grid to search for best hyperparameters
# Define estimator - same parameters as base model

# rf = RandomForestClassifier(n_estimators=50, random_state=123 )
```

In [72]:
```
# Create the FULL model based on the random_grid parameters
# rf_random = RandomizedSearchCV(estimator = rf, param_distributions = r
andom_grid, n_iter = 5, cv = 5, verbose=2, random_state=123, n_jobs = -
1)
```

In [73]:
```
# Fit the SMALL random search model with 1000 rows
# rf_random_1000 = rf_random.fit(train_features_1000, train_labels_1000)
```

In [74]:
```
# Fit the MEDIUM model with 5000 rows
# rf_random_5000 = rf_random.fit(train_features_5000, train_labels_5000)
```

In [75]:
```
# Fit a model on all data using the same random_grid parameters
# rf_random_all = rf_random.fit(train_features, train_labels)
```

In [76]:
```
#pickle.dump(rf_random_all,open("rf_random_all.pkl","wb"))
#pickle.dump(rf_random_1000,open("rf_random_1000.pkl","wb"))
#pickle.dump(rf_random_5000,open("rf_random_5000.pkl","wb"))
#pickle.dump(rf_base,open("rf_base.pkl","wb"))
#pickle.dump(rf_base_time,open("rf_base_time.pkl","wb"))
```

In [77]:
```
# Load Pickled Models to avoid re-running models

rf_random_all = pickle.load( open("Pickle/rf_random_all.pkl", "rb"))
rf_random_1000 = pickle.load( open("Pickle/rf_random_1000.pkl", "rb"))
rf_random_5000 = pickle.load( open("Pickle/rf_random_5000.pkl", "rb"))
rf_base = pickle.load( open("Pickle/rf_base.pkl", "rb"))

# Load other stored variables
rf_base_time = pickle.load( open("Pickle/rf_base_time.pkl", "rb"))
```

In [78]:
```
#
# Compare Random Grid Parameters among different datasets - 1000, 5000,
 All
########################################################################
####
```

In [79]:
```
rf_random_1000.best_params_
```

Out[79]:
```
{'n_estimators': 1000,
 'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': False}
```

In [80]:
```
rf_random_5000.best_params_
```

Out[80]:
```
{'n_estimators': 1000,
 'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': 1000,
 'bootstrap': False}
```

In [81]:
```
rf_random_all.best_params_
```

Out[81]:
```
{'n_estimators': 1000,
 'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': 1000,
 'bootstrap': False}
```

In [82]:
```
#
# Determine Log-Loss and Accuracy for All RF Models
##############################################
```

In [83]:
```
# Generate Predictions for Base model
rf_base_preds = rf_base.predict_proba(X_test)
```

In [84]:
```
# Log-Loss and Accuracy for Base Model
# import numpy as np
# from sklearn.metrics import log_loss, accuracy_score
rf_base_log_loss = log_loss(y_test,rf_base_preds[:,1]) # each column is
 class probability,
print(rf_base_log_loss)
rf_base_accuracy = accuracy_score(y_test,np.rint(rf_base_preds[:,1]))
print(rf_base_accuracy)
```

```
0.24818377998398874
0.9273975507607486
```

In [86]:
```
# Generate Predictions for 1000 element tuned model
rf_1000_preds = rf_random_1000.predict_proba(test_features_1000)
```

In [87]:
```
# Log-Loss and Accuracy for 1000 Row Model
rf_1000_log_loss = log_loss(test_labels_1000,rf_1000_preds[:,1]) # each
 column is class probability,
print(rf_1000_log_loss)
rf_1000_accuracy = accuracy_score(test_labels_1000,np.rint(rf_1000_preds
[:,1]))
print(rf_1000_accuracy)
```

```
0.48383002447131424
0.786
```

In [88]:
```
# Generate Predictions for 5000 element tuned model
rf_5000_preds = rf_random_5000.predict_proba(test_features_5000)
```

In [89]:
```
# Log-Loss and Accuracy for 5000 Row Model
rf_5000_log_loss = log_loss(test_labels_5000,rf_5000_preds[:,1]) # each
 column is class probability,
print(rf_5000_log_loss)
rf_5000_accuracy = accuracy_score(test_labels_5000,np.rint(rf_5000_preds
[:,1]))
print(rf_5000_accuracy)
```

```
0.2587950210933267
0.9172
```

In [90]:
```
# Generate Predictions for Full Model
rf_all_preds = rf_random_all.predict_proba(test_features)
```

In [91]:
```
# Log-Loss and Accuracy for Full Model
rf_all_log_loss = log_loss(test_labels,rf_all_preds[:,1]) # each column
 is class probability,
print(rf_all_log_loss)
rf_all_accuracy = accuracy_score(test_labels,np.rint(rf_all_preds[:,1]))
print(rf_all_accuracy)
```

```
0.26495811383785384
0.9139585431797699
```

In [92]:
```
#
#  Feature Importance of Base RF Model
######################################
```

In [93]:

```
rf_base.feature_importances_
```

```
Out[93]: array([2.19576698e-02, 4.61408532e-03, 4.54027314e-03, 4.70482752e-03,
                5.04434562e-03, 4.89174431e-03, 4.29335739e-03, 4.71446799e-03,
                4.33285167e-03, 2.53118395e-02, 4.18768936e-03, 2.74401645e-02,
                4.24549574e-03, 2.24728625e-02, 4.51155587e-03, 4.70138546e-03,
                4.07981463e-03, 4.66154902e-03, 4.32649362e-03, 4.55287778e-03,
                2.36252049e-02, 2.97495910e-03, 4.83270092e-03, 4.10554682e-03,
                4.39553108e-03, 4.58738124e-03, 3.93261673e-03, 4.28856168e-03,
                4.11493552e-03, 2.35251656e-02, 4.48150958e-03, 4.97092891e-03,
                4.69097167e-03, 8.44852224e-04, 4.56074273e-03, 2.38307012e-02,
                4.06558251e-03, 4.10675339e-03, 4.14507300e-03, 4.40307200e-03,
                4.35607051e-03, 4.62930976e-03, 4.05495603e-03, 4.23632962e-03,
                6.04834561e-02, 4.31834504e-03, 4.42748648e-03, 4.86906825e-03,
                4.45527694e-03, 4.49465475e-03, 4.47505121e-03, 4.40967058e-03,
                4.30586393e-03, 4.24420884e-03, 5.77902399e-03, 4.65919257e-03,
                4.29646248e-03, 4.23596110e-03, 4.13769516e-03, 4.62747596e-03,
                4.79847581e-03, 4.96326744e-03, 4.85468987e-03, 4.34816359e-03,
                4.08267411e-03, 4.10669970e-03, 4.48577477e-03, 4.27239846e-03,
                4.90572625e-03, 4.94663101e-03, 4.45542365e-03, 4.20347419e-03,
                4.88923151e-03, 4.37896839e-03, 4.82268390e-03, 4.60666937e-03,
                4.63525565e-03, 4.42760693e-03, 4.04569835e-03, 4.08931141e-03,
                4.35296175e-03, 4.23361427e-03, 4.06210305e-03, 4.26940698e-03,
                5.03295089e-03, 4.90955336e-03, 4.48772712e-03, 4.54793638e-03,
                4.66487371e-03, 4.52246681e-03, 4.29532532e-03, 4.51421155e-03,
                3.91441636e-03, 4.94528532e-03, 4.76524496e-03, 4.36378715e-03,
                2.49368096e-02, 4.72404952e-03, 4.22808435e-03, 4.68432405e-03,
                4.31596025e-03, 4.50018141e-03, 4.82522551e-03, 4.14196338e-03,
                4.34990601e-03, 4.29001663e-03, 4.71185014e-03, 4.56621538e-03,
                4.47170104e-03, 4.57825000e-03, 5.53908348e-03, 4.12908563e-03,
                4.41852369e-03, 2.54004394e-05, 2.74933728e-05, 2.12694991e-05,
                1.24306579e-04, 7.06309615e-05, 1.83493530e-04, 1.95827069e-04,
                1.66573377e-04, 1.89412937e-04, 2.01096373e-04, 1.09923060e-04,
                2.23111695e-04, 1.08342626e-04, 1.36453881e-04, 1.40313608e-04,
                9.23674689e-04, 1.80819595e-04, 7.88653360e-05, 1.74448423e-04,
                1.81885577e-04, 2.70236367e-04, 1.66474916e-04, 1.42727207e-04,
                1.55435636e-04, 8.89766900e-05, 1.53067639e-04, 2.73087641e-04,
                1.03984958e-04, 8.11105103e-05, 1.15089862e-04, 1.28179643e-04,
                2.71396471e-04, 8.82736849e-05, 1.25203360e-04, 1.25073893e-04,
                3.63750965e-04, 1.48598787e-04, 1.26851273e-04, 1.95659326e-04,
                1.32535811e-04, 1.07272428e-04, 1.04785043e-04, 7.72072166e-05,
                1.77091445e-04, 2.36757909e-04, 1.70837183e-04, 2.90568738e-04,
                3.12715120e-04, 3.43341381e-04, 1.02091925e-04, 1.43551548e-04,
                1.19334686e-04, 1.88758136e-04, 1.32031075e-04, 1.70739977e-04,
                1.34041510e-04, 2.47020596e-04, 2.36518270e-04, 8.68803090e-05,
                1.76367906e-04, 1.00568646e-04, 1.90118218e-04, 7.75785426e-04,
                1.75561688e-04, 1.22049161e-03, 1.54836293e-03, 2.61477475e-03,
                3.61264114e-03, 3.43330327e-03, 5.43763469e-04, 4.22599984e-05,
                2.04852161e-03, 1.07197816e-03, 5.68018268e-04, 5.29238189e-04,
                1.38754061e-03, 4.73326287e-03, 1.27595597e-03, 5.74684827e-04,
                3.04144390e-05, 1.45258311e-05, 2.68055121e-03, 2.33255330e-04,
                1.07274183e-03, 6.86160049e-04, 7.46941091e-04, 0.00000000e+00,
                2.68969182e-03, 7.07513138e-04, 2.27735842e-03, 2.24227069e-03,
                2.46207353e-03, 2.32227765e-03, 2.33086267e-03, 2.52010851e-03,
                2.37228516e-03, 2.29398076e-03, 2.48736925e-03, 2.66476152e-03,
                2.13190093e-03, 2.28080932e-03, 1.20596312e-05, 3.13181928e-05,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.53716157e-04,
                4.62617392e-04, 3.22426617e-05, 1.32723410e-04, 0.00000000e+00,
                3.96113319e-05, 7.24101742e-05, 4.80447222e-07, 1.59100020e-04,
```

```
1.00228622e-04, 0.00000000e+00, 7.62196353e-05, 9.13298174e-04,
2.29698122e-06, 1.56694305e-05, 0.00000000e+00, 9.42177642e-04,
0.00000000e+00, 2.51536688e-05, 9.65568867e-05, 7.24120813e-07,
1.75796957e-04, 0.00000000e+00, 1.02147554e-05, 0.00000000e+00,
2.31585403e-06, 5.78321660e-05, 2.23130924e-05, 6.82549862e-06,
8.58384446e-04, 5.74466965e-04, 5.14431604e-04, 1.02733948e-04,
1.31125866e-05, 0.00000000e+00, 0.00000000e+00, 5.13112954e-05,
1.21673683e-06, 1.20252649e-05, 0.00000000e+00, 2.50264158e-05,
4.63700364e-04, 2.60120604e-03, 3.11364563e-04, 4.93507569e-05,
1.23851821e-03, 3.46982077e-05, 4.79993237e-05, 6.57883778e-06,
9.26484896e-05, 0.00000000e+00, 0.00000000e+00, 3.31148179e-05,
0.00000000e+00, 5.61997005e-05, 1.10851826e-04, 0.00000000e+00,
0.00000000e+00, 1.79237738e-05, 1.86935927e-04, 1.19757564e-03,
1.49372707e-05, 1.91049485e-04, 0.00000000e+00, 1.19433666e-04,
8.53069442e-07, 0.00000000e+00, 9.14139534e-06, 7.14993594e-07,
1.79725404e-03, 0.00000000e+00, 0.00000000e+00, 1.87453695e-04,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 7.23094797e-06,
6.76028780e-04, 1.86917371e-05, 2.88857284e-04, 1.35746406e-03,
5.10741293e-04, 3.77701227e-05, 4.21926473e-05, 1.93095000e-05,
5.43978123e-05, 1.39638366e-03, 1.13187976e-03, 1.69381039e-05,
5.39827564e-04, 4.40109539e-04, 0.00000000e+00, 4.42090457e-05,
2.72504858e-05, 1.72614436e-05, 8.07719691e-04, 5.93405813e-04,
3.00543378e-05, 0.00000000e+00, 1.72129229e-06, 4.38762351e-04,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
7.50338005e-04, 0.00000000e+00, 2.74232348e-03, 0.00000000e+00,
2.55836996e-04, 0.00000000e+00, 3.72568845e-04, 2.21190838e-04,
1.43084138e-07, 0.00000000e+00, 2.03106847e-04, 4.14633004e-05,
4.47640158e-03, 5.46458158e-03, 6.95946871e-03, 0.00000000e+00,
2.93317877e-03, 1.73990829e-03, 0.00000000e+00, 2.99929110e-03,
2.56776743e-06, 1.70231687e-05, 0.00000000e+00, 0.00000000e+00,
7.52471660e-06, 2.43865672e-04, 2.66432655e-04, 1.72003022e-05,
2.76946338e-03, 3.87356054e-05, 2.86155345e-03, 8.84101574e-05,
4.00973227e-03, 2.34966438e-03, 9.46756254e-04, 1.32674100e-03,
7.47842613e-05, 1.50761212e-05, 5.60172094e-04, 8.50274306e-04,
1.14709526e-04, 6.07251902e-04, 0.00000000e+00, 8.09251666e-04,
1.83656204e-05, 3.58782012e-04, 2.03227006e-04, 5.64559573e-04,
1.28860827e-05, 3.10084072e-03, 2.84709424e-03, 2.96224209e-03,
9.68176636e-05, 7.26218018e-04, 2.19342257e-03, 3.15095697e-03,
2.19159580e-03, 2.84088295e-03, 2.88437032e-03, 2.96774490e-03,
3.09536333e-03, 7.05958777e-04, 1.15147677e-04, 4.08359096e-03,
3.22983596e-03, 2.58329173e-04, 1.88723942e-03, 9.34313633e-04,
7.57055648e-04, 1.76222247e-03, 1.49479948e-03, 3.08634085e-03,
7.13091307e-04, 1.44654322e-03, 2.20296468e-03, 1.39830127e-03,
1.12689287e-03, 1.23370596e-03, 5.72953441e-04, 1.88697845e-03,
1.16985829e-03, 1.46906566e-03, 8.77790809e-04, 1.27767507e-03,
4.11196560e-04, 1.16643982e-03, 1.33464864e-03, 7.74161609e-04,
2.88282305e-04, 1.56963778e-08, 6.58285595e-04, 1.66247279e-03,
2.01795038e-04, 5.57773878e-04, 1.19065500e-03, 8.59052095e-04,
5.14626666e-04, 3.26117624e-04, 5.58825139e-04, 2.21572557e-07,
5.84951695e-04, 3.85437459e-04, 1.76457117e-04, 2.17623585e-04,
9.21530546e-05, 3.21602650e-03, 1.22140220e-05, 9.60876638e-04,
1.60124049e-04, 3.09974776e-04, 9.75534992e-04, 2.01688740e-04,
8.31186668e-05, 9.71383991e-04, 3.57265403e-04, 1.25548067e-04,
5.10551899e-04, 4.94192270e-04, 3.61042372e-04, 6.57704490e-04,
7.08350161e-04, 6.41547994e-04, 7.26995550e-04, 2.72957610e-04,
8.33271017e-04, 3.85393631e-04, 9.61226200e-05, 1.01715035e-03,
3.61970603e-04, 4.12161562e-04, 7.12664832e-04, 4.26350366e-04,
```

```
       3.09721409e-04, 6.00479217e-04, 8.51278834e-05, 1.36145431e-03,
       5.99698843e-04, 3.72760455e-04, 6.86691465e-04, 3.75835536e-04,
       1.36797109e-03, 2.47316479e-04, 1.02583890e-03, 3.23943725e-04,
       4.87512832e-04, 5.91838392e-04, 3.65676160e-04, 3.69384385e-04,
       4.17627861e-05, 5.08768857e-04, 7.78862369e-04, 8.68579165e-04,
       2.50864184e-04, 7.93733299e-05, 2.64925241e-04, 9.41540318e-04,
       4.17095219e-04, 4.02885951e-04, 1.60405536e-04, 4.96035495e-04,
       4.28745449e-04, 1.42545366e-03, 7.53628543e-04, 6.26934009e-04,
       1.77717844e-03, 4.61356253e-04, 5.11173338e-04, 4.01239311e-04,
       3.67931304e-04, 4.21301233e-04, 2.35173402e-04, 2.49097441e-04,
       5.74008478e-04, 4.82927529e-04, 1.04483759e-03, 7.18448703e-04,
       1.37272816e-03, 1.82682680e-06, 3.31779683e-04, 6.97077519e-04,
       4.24798918e-04, 4.37114327e-04, 8.57595020e-04, 4.95756146e-04,
       5.86053205e-04, 1.21320890e-03, 4.93604290e-04, 2.52189488e-04,
       7.74686488e-04, 3.62113573e-04, 2.91734814e-04, 5.98962065e-04,
       1.01921155e-03, 2.43672730e-04, 9.76607112e-04, 1.24287848e-03,
       2.86509193e-04, 5.14574235e-04, 1.17435983e-03, 1.18877379e-03,
       4.67374030e-04, 6.28369628e-04, 2.37936407e-04, 7.97930511e-04,
       6.62439286e-04, 5.69390890e-04, 4.97797317e-04, 4.72505073e-04,
       5.31497414e-04, 1.00473644e-03, 3.97094551e-04, 4.76147072e-04,
       5.54053070e-04, 5.95136452e-04])
```

In [94]:

```python
# Top 10 Features for Base Model
importances = rf_base.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_base.estimators_
],
             axis=0)
indices = np.argsort(importances)[::-1]
```

In [95]:

```python
#import numpy as np
#import matplotlib.pyplot as plt

# Print the feature ranking
# print("Feature ranking:")

#for f in range(X_train.shape[1]):
#    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indic
es[f]]))

print("Top Ten Features:")

for f in range(0, 10):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indice
s[f]]))
```

```
Top Ten Features:
1. feature 44 (0.060483)
2. feature 11 (0.027440)
3. feature 9 (0.025312)
4. feature 96 (0.024937)
5. feature 35 (0.023831)
6. feature 20 (0.023625)
7. feature 29 (0.023525)
8. feature 13 (0.022473)
9. feature 0 (0.021958)
10. feature 338 (0.006959)
```

In [96]:
```
#######################
#  RESULTS
#######################
```

In [97]:
```
# Build Summary Table for All Model Types
data = [['XGBoost','Base',xgb_base_log_loss, xgb_base_accuracy, xgb_base
_time],
        ['Random Forest Complete Dataset','Base',rf_base_log_loss, rf_ba
se_accuracy, rf_base_time],
        ['Random Forest Complete Dataset','Tuned',rf_all_log_loss, rf_al
l_accuracy, 90],
        ['Random Forest 1000 Entries', 'Tuned', rf_1000_log_loss, rf_100
0_accuracy, 2],
        [' Random Forest 5000 Entries', 'Tuned', rf_5000_log_loss, rf_50
00_accuracy, 7]]
```

In [98]:
```
#from tabulate import tabulate
print (tabulate(data, headers=["Model", "Tuning","Log-Loss", "Accuracy",
"Wall Time"]))
```

```
Model                           Tuning      Log-Loss    Accuracy    Wal
l Time
------------------------------  --------  ----------  ----------  -----
------
XGBoost                         Base        0.585187    0.766395
7.89
Random Forest Complete Dataset  Base        0.248184    0.927398
41.22
Random Forest Complete Dataset  Tuned       0.264958    0.913959
90
Random Forest 1000 Entries      Tuned       0.48383     0.786
2
Random Forest 5000 Entries      Tuned       0.258795    0.9172
7
```

In [ ]:
```
##
#  Visualizations
###########################################
```

In [ ]:
```
# Single Tree Visualization
# All Credit to https://www.kaggle.com/willkoehrsen/a-complete-introduct
ion-and-walkthrough#Visualize-Single-Decision-Tree

#model = RandomForestClassifier(max_depth = 3, n_estimators=10)
#model.fit(train_selected, train_labels)
model = rf_base
estimator_limited = model.estimators_[1]
estimator_limited
```

In [ ]:
```
train_selected = X_train
```

In [ ]:
```
#from sklearn.tree import export_graphviz

export_graphviz(estimator_limited, out_file='Images/1_tree.dot', feature
_names = train_selected.columns,
                rounded = True, proportion = False, precision = 2, fille
d = True)
```

In [ ]:
```
# Convert .dot file to .png - commented out because run time is long.
# Using .png in write-up
#import os
#os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'
#os.system('dot -Tpng tree_limited.dot -o tree_limited.png')
```

In [99]:
```
### Create Small Tree with 3 Levels to be able to visualize
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf_base.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 50,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 123,
 'verbose': 0,
 'warm_start': False}
```

In [ ]:
```
# Limit depth of tree to 3 levels to be able to see details.
rf_small = RandomForestClassifier(n_estimators=10, max_depth = 3)
rf_small.fit(train_features, train_labels)
# Extract the small tree
tree_small = rf_small.estimators_[5]

# Save the tree as a png image - Commented out to save time since alread
y done
# export_graphviz(tree_small, out_file = 'Images\\small_tree.dot', featu
re_names = train_selected.columns, rounded = True, precision = 1)
# (graph, ) = pydot.graph_from_dot_file('small_tree.dot')
# graph.write_png('small_tree.png');
```

In [ ]:

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```python
################# Old Code Below Here #################
```

In [37]:

```python
#from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=50)

start = time.time()
rf.fit(X_train, y_train)

end = time.time()
rf_time=round((end-start),2)
rf_time
```

Out[37]:

```
31.57
```

In [40]:

```python
preds = rf.predict_proba(X_test)
```

In [41]:

```python
rf_base_log_loss = log_loss(y_test,preds[:,1]) # each column is class pr
obability,
print(rf_base_log_loss)
rf_base_accuracy = accuracy_score(y_test,np.rint(preds[:,1]))
print(rf_base_accuracy)
```

```
0.4925212965112307
0.7761755818268569
```

In [43]:

```python
data = [['XGBoost',xgb_base_log_loss, xgb_base_accuracy, xgb_time],
         ['LinearSVC', "N/A", svm_base_accuracy, LinearSVC_time],
         ['Random Forest',rf_base_log_loss, rf_base_accuracy, rf_time]]
```

In [44]:

```python
print(data)
```

```
[['XGBoost', 0.5851586336031108, 0.7655198006679743, 10.97], ['LinearSV
C', 'N/A', 0.528547951015215, 61.67], ['Random Forest', 0.4925212965112
307, 0.7761755818268569, 31.57]]
```

In [45]:

```python
rowlen=len(data)
print(rowlen)
```

```
3
```

In [46]:
```
import numpy as np
rownums=np.arange(0,rowlen,1)
rownums=rownums+1

headers=['Model', 'Log-Loss', 'Accuracy', 'Wall Time']
print(headers)
print(rownums)
```

```
['Model', 'Log-Loss', 'Accuracy', 'Wall Time']
[1 2 3]
```

In [47]:
```
print(pd.DataFrame(data, rownums,headers))
```

```
           Model  Log-Loss  Accuracy  Wall Time
1        XGBoost  0.585159  0.765520      10.97
2      LinearSVC       N/A  0.528548      61.67
3  Random Forest  0.492521  0.776176      31.57
```

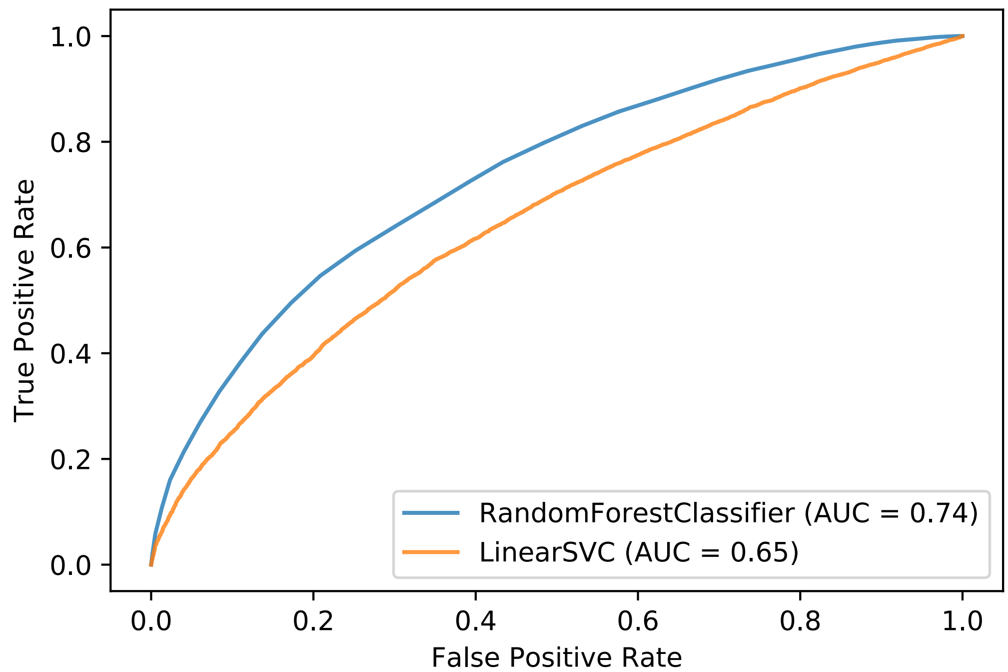
In [48]:
```
# Output table of the models results

print (tabulate(data, headers=["Model", "Log-Loss", "Accuracy", "Wall Ti
me"]))
```

```
Model          Log-Loss              Accuracy    Wall Time
-------------  ------------------  ----------  -----------
XGBoost        0.5851586336031108    0.76552         10.97
LinearSVC      N/A                   0.528548        61.67
Random Forest  0.4925212965112307    0.776176        31.57
```

In [ ]:
```
######################################
#. Extra code for loading and plotting
######################################
```

In [42]:
```python
# Plot the Random Forest model vs SVM
ax = plt.gca()
rfc_disp = plot_roc_curve(rf, X_test, y_test, ax=ax, alpha=0.8)
svm_disp.plot(ax=ax, alpha=0.8)
```

Out[42]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1a23145f50>



In [31]:
```python
svc = LinearSVC(random_state=123)
svc.fit(X_train, y_train)
svc_disp = plot_roc_curve(svc, X_test, y_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
<ipython-input-31-863beec980c9> in <module>
----> 1 svc = SVCLinear(random_state=42)
      2 svc.fit(X_train, y_train)
      3 svc_disp = plot_roc_curve(svc, X_test, y_test)

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

In [ ]:
```python
# Ramdom forest classifier
rfc = RandomForestClassifier(random_state=42)
rfc.fit(X_train, y_train)

ax = plt.gca()
rfc_disp = plot_roc_curve(rfc, X_test, y_test, ax=ax, alpha=0.8)
svc_disp.plot(ax=ax, alpha=0.8)
```

In [ ]:
```
X, y = datasets.make_classification(random_state=0)
clf = svm.SVC(random_state=0)
clf.fit(X_train, y_train)
SVC(random_state=0)
metrics.det_curve(clf, X_test, y_test)
plt.show()
```

In [ ]:
```
import scikitplot as skplt
rf = RandomForestClassifier()
lr = LogisticRegression()
nb = GaussianNB()
svm = LinearSVC()
rf_probas = rf.fit(X_train, y_train).predict_proba(X_test)
lr_probas = lr.fit(X_train, y_train).predict_proba(X_test)
nb_probas = nb.fit(X_train, y_train).predict_proba(X_test)
svm_scores = svm.fit(X_train, y_train).decision_function(X_test)
probas_list = [rf_probas, lr_probas, nb_probas, svm_scores]
clf_names = ['Random Forest', 'Logistic Regression',
              'Gaussian Naive Bayes', 'Support Vector Machine']
skplt.metrics.plot_calibration_curve(y_test,
                                      probas_list,
                                     clf_names)
plt.show()
```

In [ ]:
```
from sklearn import metrics
metrics.det_curve(clf, X_test, y_test)
plt.show()
```

In [ ]: