Data Science Challenge

Edward Fry

4/13/2021

Phishing Attacks

Microsoft Data & AI

Joao Pedro Martins, Proctor

Abstract

XYZ Corporation is a credit card company. They are having difficulty controlling their fraud detection and recovery costs, especially through phishing attacks. It is important that XYZ understand first, who is likely to be attacked and why, and second, activities they can engage in to start decreasing the volume of attacks they receive.

They have collected data on their on their email phishing attacks that they have found so far and would like to use this dataset to predict which employees are being targeted and why. The dataset contains records for email users stretching back for one year.

The dataset has been obfuscated to prevent any leak of IP or identities from our analysis, and thus the column variables will be general in nature. The columns included were what the IT leads could put together is such short notice, but they are trying to get more data in the coming months. The business leaders of XYZ company would like to understand causality if at all possible, especially to understand which variables they should be looking at and if there are any more that would be worthwhile to try to get for future attempts.

The variable that XYZ would like you to try to build a model around is labeled “EmployeeTargetedOverPastYear”.

Auto ML

Please do NOT use any form of AutoML for your models

Explainability

Please prepare to explain local feature effects for every observation of the dataset.

Deployment

Please deploy your models and be ready to demonstrate a live online REST API call to those models.

Introduction

With the significant amount of data that the businesss needs to analyze, it is infeasible for humans to sift through it in order to determine how to best identify phishing attacks. Thus, this study aims to identify a model (or models) that can be used to accurately identify attack candidates as new data comes in. Phishing attacks expose XYZ to risk and potentially high reactive costs, including expenses for legal activity, brand reputation, and regulatory consequences. As such, it is important to minimize the volume of attacks. One promising approach is to predict which employees may be targeted. Additionally, if reasons can be pinpointed as to why certain employees are targeted, then more holistic remedies can be implemented proactively.

In this study, data is first normalized, cleaned-up, and imputed. Then, an exhaustive search of possible parameter combinations for each of 11 candidate models is undertaken. Keeping the prediction goal in mind, a scoring function is chosen to minimize the cost by maximizing the true values chosen by the model. Specifically, 4 classification states are possible: True and false positives and negatives. True positives and true negatives represent accurate identification, so these are desirable. False positives are less desirable; however, the consequences of a false positive are that an employee perhaps takes unnecessary security precations, which is not very impactful. The consquences of a false negative, however, would mean that a likely target ends up not taking additional precautions, plus systemic remedies might also be missed. Therefore, it would be useful to weight false negatives more highly than false positives.

Once results are computed, then, those results are compared to identify the lowest cost and other metrics of interest. Finally, promising models are discussed and recommmended to the business along with specific steps to address the most common reasons for being targeted.

Methods

Data cleanup

As with any data set, it's important to make sure that the data is clean, normalized, and as complete as possible. The incoming dataset was 14k rows in length. The data was well-behaved for the most part with no extraneous characters, obvious errors, or crazy outliers. Most of the columns were categorial and numerical. Two columns, city & email, were strings and accordingly converted to lowercase for normalization.

A short function to display any missing values revealed that there were, in fact, a few missing values throughout the dataset for every variable. Very few values were missing relative to the total size of the dataset that only performing a visual inspection of the data would have probably led one to conclude that there were no missing values, so having the computer perform the check is important. Four columns have missing values.

Once missing values are identified, an appropriate imputation strategy for each variable is determined. Some variables, especially the many categoricals, are best imputed by looking at the mode. Some need the median to protect against outliers, and the remainder can use the mean. To determine the distribution, each variable's histogram was plotted and examined. The few continuous variables followed a slightly right-skewed normal distribution.

As a final check, once all imputation is complete, the program again checks each variable for missing values. This time, nothing is displayed, so the data is ready for modeling.

Phishing Leading Indicators of "Causality"

Truly determing causality requires a randomized experiment. Since this dataset is observational, the best we can do is look at correlation. A heat map of all feature correlations to the phishing target reveals that the red team evaluation is a leading indicator of someone being a phishing target. Additional variables that showed some correlation - though nowhere near the extent of the red team eval - were usage metric 5, fraud training, and gender. Future data collection might consider focusing on these variables and perhaps adding variables that may be related to these that may not have been collected this time, which could yield a more accurate association.

Grid Search

Because it is so important to get the model right in order to correctly identify the maximum number of potential phishing targets, a variety of potential classifers were chosen for examination. For each model, various combinations of parameters were used to fit models on a set of training data. A hold-out test set was then used to compute the relative performance of each set of parameters by model. The collection of tested parameters is called a "grid" since the parameters could be listed in a tabular or grid-like format. Because each combintation is fit in turn, the technique is a "grid search". The grid search results in a thorough study of possible models and parameters with minimal preconceptions about which ones might ultimately be the best. While a random grid search, in which only a portion of possible parameters in the grid are chosen at random for evaluation, was an option, it seemed that an exhaustive grid search that tried every single combination would be better in this situation since the business stands to lose so much money on false results.

Part of the grid search process is cross-validation to guard against overfitting. It works by holding out a 1/n potion of the training data, then using the heldout data for validation. Then, the next portion is held out until all n portions have been held out and used for validation. Originally, a 10-fold cross validation scheme was attempted; however, for the number of models and parameter grids under examination, 10-fold cross validation simply took too long to compute. Thus, this was reduced to 5-fold cross validation. This is still an adequate amount according to the literature and equates to 20% of the data being held out for validation on each pass.

Scoring

All models need a way to determine the relative effectiveness of the model, and the models under grid search are no exception. In fact, choosing a scoring model was very important to this particular problem. Because we want to optimize to minmimize losses and specifically to minimize false positives and false negatives, this is accounted for in the scoring model. Minimizing the number of false positives could also be considered maximizing the precision, which denotes the relative number of true positives. Similarly, minimizing the number of false negatives is equivalent to maximizing recall. There is a commonly used scoring metric called F1, which is the harmonic mean of precision and recall. That seems promising all by itself, but the next step is to assign appropriate weighting. For this exercise, let's assume that false negatives are 5 times worse than false positives due to the much more serious repercussions of a false negative. Fortunately, there is a weighted variant of the F1 score known as f-beta, where beta is the relative weight to use. Thus, for the main scorer, f-beta is maximized to meet XYZ's requirements.

Models

Eleven models were examined for this problem, each one with its own set of parameters. Tree-based classifiers included decisions trees and random forests, and they were fit with gini and entropy criterion, 200 estimators (which is twice the new scikit default of 100), bootstrap and oob both on and off, various leaf configurations, and square root and log maximum features.

A ridge classifer examined 6 solvers for a range of alpha between 0 and 1. Logistic regression tried 5 solvers with each of l1, l2, elastic net, or no penalty, an exponential set of C values, and an l1 ratio between 0 and 1. A passive aggressive classifer used a similar range of C values. Support vector machines can provide accurate predictions; however, they are computationally expensive. For 160,000 rows of data, a linear SVM is used in its place with an exponential range of C, hinge and squared hinge losses, and l1 and l2 penalties. AdaBoost simply varied the learning rate between .01 and .5. Stochastic gradient descent followed a similar scheme to logistic regression.

Finally, three more exotic models were fit. LightGBM was fit on each of 4 boosting methods with depths between 2 and 10 and learning rates between .05 and .1. XGBoost ranged the same depths and learning rate while varying the number of estimators between 60 and 220. And a multilayer perceptron (neural network) combined 4 activations, 3 common solvers, and 3 learning rates with 5 exponentially increasing alphas.

Computational Considerations

As noted above, the cross validation rate was reduced to 5 from 10 in order to speed up calculations. In addition, each model examined in the grid search needed to be temporarily pulled into its own small Python code file that contained the logic for that one model along with some logic to save the results in a pickle file. This let the project leverage parallel computation. Later, the saved results of each grid search were read back into the main application and further examined for the final solution to the problem.

Results

Top Models

Several models scored perfectly in the grid search, including Ada Boost, Decision Tree, Logistic Regression, Passive Aggressive, Random Forest, Ridge Regression and Support Vector Machine . Cost was calculated using the formula false positive + 5 * false negative. For completeness, the exact counts of true and false positives and negatives is available.

The other models were lower performers by all of the same measures, and the multilayer perceptron (neural network), surprisingly, ended up being the worst performer of all. This is perhaps because the simple nature of the correlations lend themselves well to simple linear models. The full list of optimal parameters is in Appendix A.

Relative performance

In order to better visualize the relative performance of each model against its peers, the four key metrics of f-beta, accuracy, precision, and recall were plotted together. This might be convenient if, for example, it was desired to look at another metric, like precision, and see at a glance if that might be a better solver than f-beta. It also gives a good indication of how the models stratify in their performance. For the most part, models tended to stay in their performance "lane" and not vary too much. For instance, a moderate f-beta model would likely also have a moderate accuracy, precision, and recall. There were exceptions to this, of course, but they were not too much outside expectations. For instance, AdaBoost had a moderately high precision even though its other scores were all at the bottom of the group. However, the top models dominated all of their categories, and there is clear white space between those and the others below them.

Conclusion

This model study sought to identify that model that would produce accurate identification of potential phishing targets. False positives are undesirable, but false negatives are much more so and weighted accordingly. After cleaning and normalizing the data, an exhaustive grid search was executed on 11 candidate models. The best model was the Ada Boost. The neural network was poor and is not recommended.

Thoughts for additional research might include using a tool like Tensorflow or PyTorch to try a variety of neural networks to see if that model can be improved with a different configuration. In addition, an ensemble method combining the top models from this study might provide a small improvement over using them separately.

Appendix A: Best Model Parameters

Optimal parameters

Ada Boost

```
C : 0.001
class_weight : balanced
fit_intercept : False
loss : squared_hinge
penalty : l2
```

Decision Tree

```
bootstrap : False
criterion : gini
max_depth : None
max_features : sqrt
max_leaf_nodes : None
min_impurity_split : 0.1
n_estimators : 50
oob_score : False
```

Extreme Gradient Boosting (XGB)

```
alpha : 1e-06
fit_intercept : False
l1_ratio : 0.0
penalty : elasticnet
```

Gradient Boosted Decision Trees (LGBM)

```
alpha : 1e-06
fit_intercept : False
l1_ratio : 0.0
penalty : elasticnet
```

Logistic Regression

```
bootstrap : False
criterion : gini
max_depth : None
max_features : sqrt
max_leaf_nodes : None
min_impurity_split : 0.1
n_estimators : 50
oob_score : False
```

Multilayer Perceptron

```
activation : tanh
alpha : 1
early_stopping : True
learning_rate : adaptive
solver : adam
```

Passive Aggressive

```
bootstrap : False
criterion : gini
max_depth : None
max_features : sqrt
max_leaf_nodes : None
min_impurity_split : 0.1
n_estimators : 50
oob_score : False
```

Random Forest

```
bootstrap : False
criterion : gini
max_depth : None
max_features : sqrt
max_leaf_nodes : None
min_impurity_split : 0.1
n_estimators : 50
oob_score : False
```

Ridge Regression

```
bootstrap : False
criterion : gini
max_depth : None
max_features : sqrt
max_leaf_nodes : None
min_impurity_split : 0.1
n_estimators : 50
oob_score : False
```

Stochastic Gradient Descent

```
alpha : 1e-06
fit_intercept : False
l1_ratio : 0.0
penalty : elasticnet
```

Support Vector Machine

```
C : 0.001
class_weight : balanced
fit_intercept : False
loss : squared_hinge
penalty : l2
```

In [1]:

```
import pandas as pd, numpy as np, matplotlib.pyplot as plt, os, sklearn, re, p
ickle, json, joblib, glob
import plotly.graph_objects as go
from utils import load_data
import seaborn as sb
from operator import attrgetter
from collections import Counter

from sklearn import metrics
from sklearn.metrics import fbeta_score, make_scorer, precision_score, recall_
score, confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.dummy import DummyClassifier
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn import linear_model as lm
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, Gradi
entBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from lightgbm.sklearn import LGBMClassifier
from xgboost import XGBClassifier

import azureml.core
from azureml.core.webservice import AciWebservice, Webservice
from azureml.core.model import InferenceConfig, Model
from azureml.core.environment import Environment
from azureml.core import Workspace, Dataset

%matplotlib inline

# Load data
wd = r"C:\Code\EdwardFryDataScienceChallenge"
dataSourceFilename = r"EdwardFry_Microsoft_issueDataset.csv"
os.chdir(wd)
df = pd.read_csv(dataSourceFilename)
df.head()
```

Out[1]:

	Activity on Company Forums	City	Email Domain	EmployeeTargetedOverPastYear	Gender (code)	Manager Rating of Likelihood to Leave Company	N
0	12.0	Kachkanar	si.edu	0.0	1	1.0	
1	28.0	Kista	earthlink.net	1.0	1	1.0	
2	16.0	Belleville	sciencedaily.com	1.0	1	1.0	
3	30.0	Chwałowice	tripadvisor.com	0.0	1	1.0	
4	6.0	Kachkanar	redcross.org	1.0	1	1.0	

5 rows × 22 columns

In [2]:
```
# Normalize and examine column names; refer to cols by name in case future dat
asets are in a different order
col_names = list(df.columns)
print(col_names)
```

```
['Activity on Company Forums', 'City', 'Email Domain', 'EmployeeTargetedOverP
astYear', 'Gender (code)', 'Manager Rating of Likelihood to Leave Company',
'NS100Training Completed', 'Social Media Activity (Scaled)', 'Survey, Employe
e Satisfaction (Scaled)', 'behaviorPattern3', 'behaviorPattern5', 'datumThrea
tMonitoringScore', 'fraudTraining Completed', 'peerUsageMetric2', 'peerUsageM
etric3', 'peerUsageMetric4', 'peerUsageMetric5', 'phishingTraining Complete
d', 'redTeamEval', 'usageMetric1', 'usageMetric4', 'usageMetric5']
```

In [3]:
```
# Init each index in case a future column is missing
col_forums = -1
col_city = -1
col_email = -1
col_employee_target = -1
col_gender = -1
col_leave = -1
col_ns100 = -1
col_social = -1
col_sat = -1
col_behavior3 = -1
col_behavior5 = -1
col_threat = -1
col_fraud = -1
col_peer2 = -1
col_peer3 = -1
col_peer4 = -1
col_peer5 = -1
col_phish = -1
col_redteam = -1
col_usage1 = -1
col_usage4 = -1
col_usage5 = -1
```

In [4]:
```
# Name each index so column can be referred to by name regardless of input ord
ering
# Use matching to identify columns in case ordering or name strings are confou
nded
for name in col_names:
    name_lower = name.lower()
    if re.search(r"forums", name_lower):
        col_forums = col_names.index(name)
    elif re.search(r"city", name_lower):
        col_city = col_names.index(name)
    elif re.search(r"email", name_lower):
        col_email = col_names.index(name)
    elif re.search(r"employee.*target", name_lower):
        col_employee_target = col_names.index(name)
    elif re.search(r"(gender|sex)", name_lower):
        col_gender = col_names.index(name)
    elif re.search(r"like.*leave", name_lower):
        col_leave = col_names.index(name)
    elif re.search(r"ns100", name_lower):
        col_ns100 = col_names.index(name)
    elif re.search(r"social.*media", name_lower):
        col_social = col_names.index(name)
    elif re.search(r"satisfaction", name_lower):
        col_sat = col_names.index(name)
    elif re.search(r"behavior.*3", name_lower):
        col_behavior3 = col_names.index(name)
    elif re.search(r"behavior.*5", name_lower):
        col_behavior5 = col_names.index(name)
    elif re.search(r"threat.*mon", name_lower):
        col_threat = col_names.index(name)
    elif re.search(r"fraud.*training", name_lower):
        col_fraud = col_names.index(name)
    elif re.search(r"peer.*2", name_lower):
        col_peer2 = col_names.index(name)
    elif re.search(r"peer.*3", name_lower):
        col_peer3 = col_names.index(name)
    elif re.search(r"peer.*4", name_lower):
        col_peer4 = col_names.index(name)
    elif re.search(r"peer.*5", name_lower):
        col_peer5 = col_names.index(name)
    elif re.search(r"ish.*training", name_lower):
        col_phish = col_names.index(name)
    elif re.search(r"red.*team", name_lower):
        col_redteam = col_names.index(name)
    elif re.search(r"^usage.*1", name_lower):
        col_usage1 = col_names.index(name)
    elif re.search(r"^usage.*4", name_lower):
        col_usage4 = col_names.index(name)
    elif re.search(r"^usage.*5", name_lower):
        col_usage5 = col_names.index(name)

all_col_indices = [col_forums, col_city, col_email, col_employee_target,
                   col_gender, col_leave, col_ns100, col_social, col_sat,
                   col_behavior3, col_behavior5, col_threat, col_fraud,
                   col_peer2, col_peer3, col_peer4, col_peer5, col_phish,
                   col_redteam, col_usage1, col_usage4, col_usage5]
```

```
print(all_col_indices)  # If all goes as planned, the output from the original training dataset should be a list of increasing integers
```

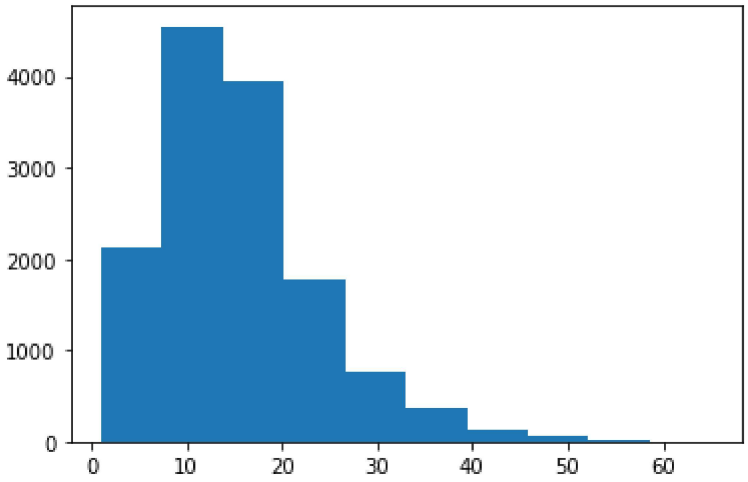
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
1]
```

In [5]:
```
# Initialize containers
est_with_means = [col_redteam]
est_with_medians = [col_forums]  # Determined after looking at histogram plots for outliers (below)
est_with_modes = []  # Most categoricals will get this
est_excludes = [col_employee_target]
target = col_employee_target
categoricals = [col_city, col_email, col_employee_target,
                col_gender, col_leave, col_ns100, col_social, col_sat,
                col_behavior3, col_behavior5, col_threat, col_fraud,
                col_peer2, col_peer3, col_peer4, col_peer5, col_phish,
                col_usage1, col_usage4, col_usage5]

# Normalize categoricals
for i in categoricals:
    try:
        df[i] = df[i].str.lower()
    except:  # Ignore non-strings
        pass
```

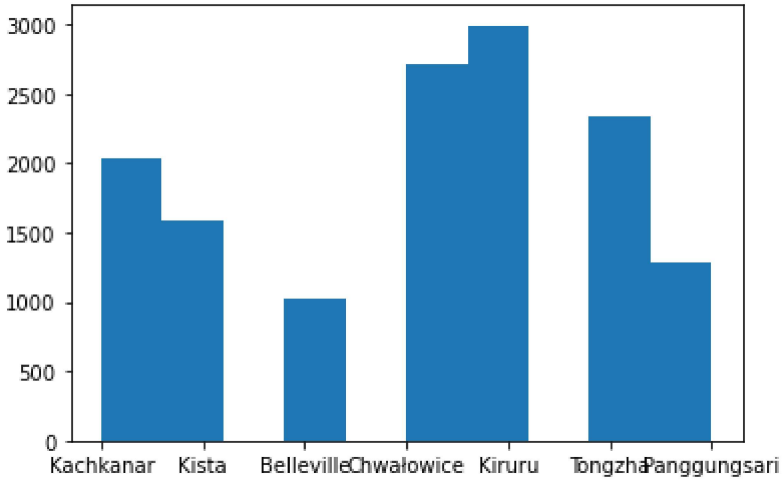
In [6]:
```
# Functions
def show_missing(df):
    print("Number of missing values by column:")
    for i in df.columns:
        if df.loc[df[i].isna(), i].shape[0] > 0:
            print("\t", i, df.loc[df[i].isna(), i].shape)
```

In [7]:
```
# Examine data
print("Shape: ", df.shape)

# Study data topography to determine best way to fill in missing values and pl
ace in appropriate container
show_missing(df)
print("\nMost commmonly used values and distribution by column:")
for i in all_col_indices:
    if df.iloc[:, [i]].isna().shape[0] > 0:
        print("\t", i, Counter(df.iloc[:, [i]]).most_common(3))

        # Looking for outliers, if normal, can you mean, otherwise use median
        plt.hist(df.iloc[:, [i]])
        plt.show()

        # Sort columns by the type of missing value estimation best for it to
 use
        if i in categoricals:
            est_with_modes.append(i)
        else:   # Anything not using median or mode will use mean
            if i not in categoricals + est_with_means + est_with_medians + est
_with_modes + est_excludes:
                est_with_means.append(i)

print("\nCategorical variables: ", categoricals)
print("\nMean estimation variables: ", est_with_means)
print("\nMedian estimation variables: ", est_with_medians)
print("\nMode estimation variables: ", est_with_modes)
```

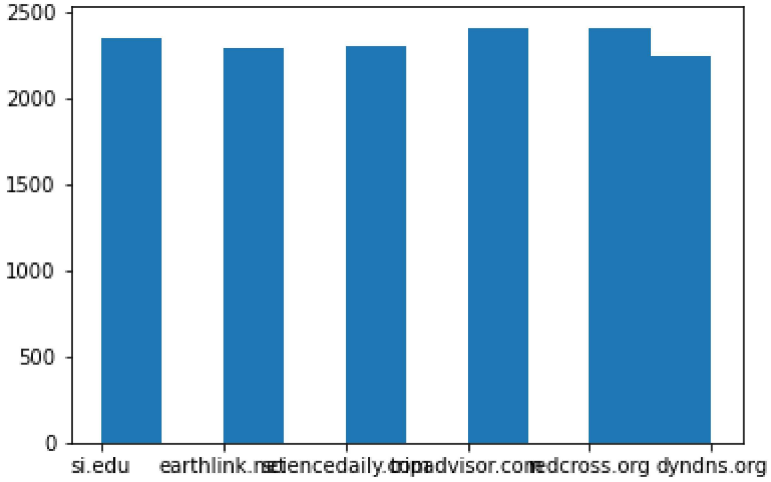
```
Shape:  (14000, 22)
Number of missing values by column:
         Activity on Company Forums (282,)
         Manager Rating of Likelihood to Leave Company (290,)
         usageMetric1 (273,)
         usageMetric5 (1278,)

Most commmonly used values and distribution by column:
         0 [('Activity on Company Forums', 1)]
```



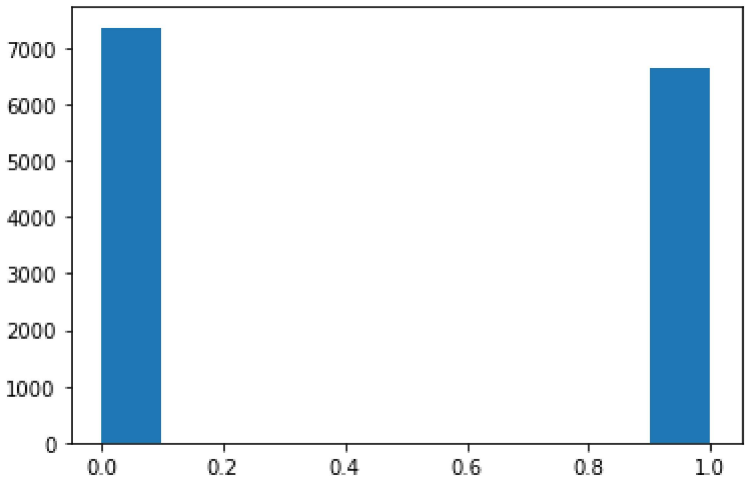

```
         1 [('City', 1)]
```



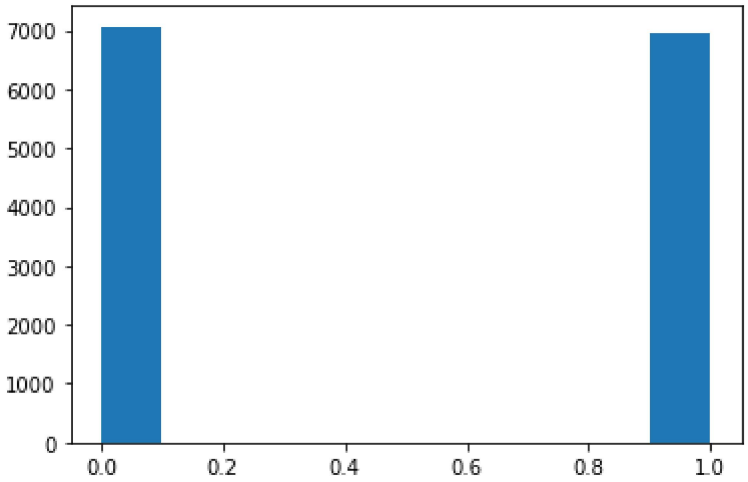

```
         2 [('Email Domain', 1)]
```



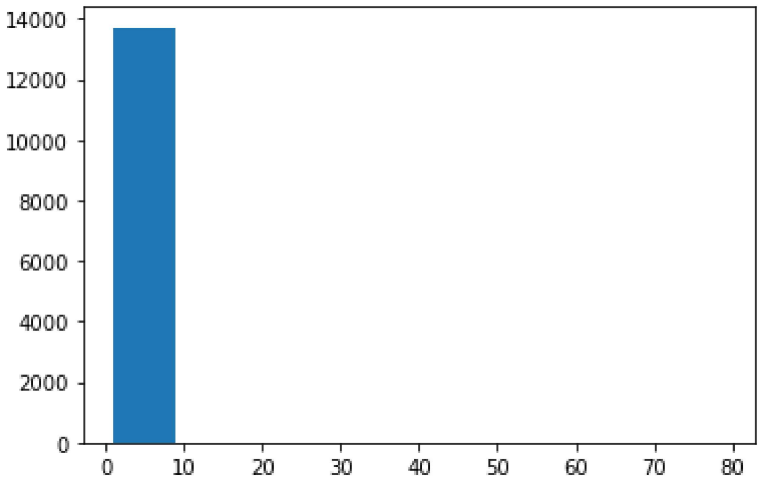
3 [('EmployeeTargetedOverPastYear', 1)]



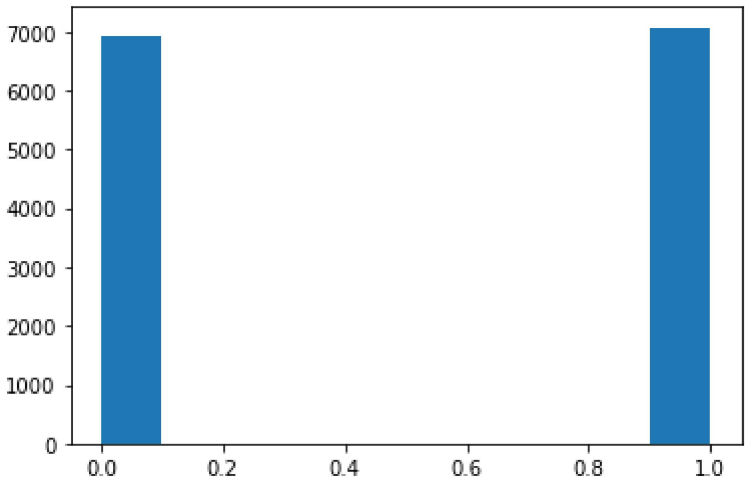
4 [('Gender (code)', 1)]



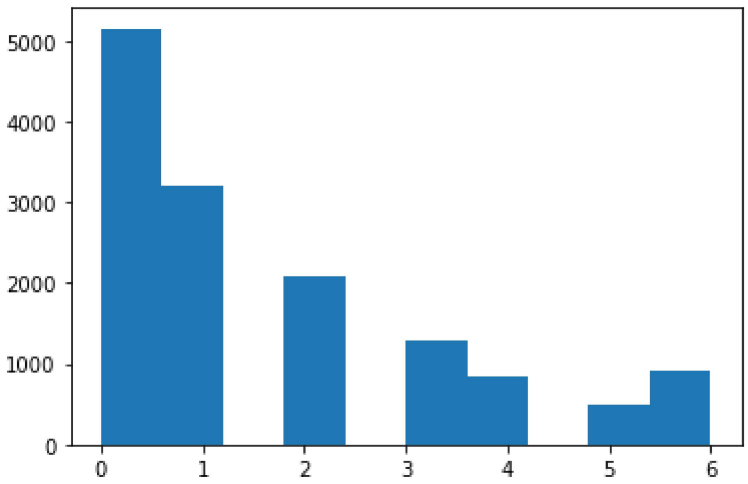
5 [('Manager Rating of Likelihood to Leave Company', 1)]



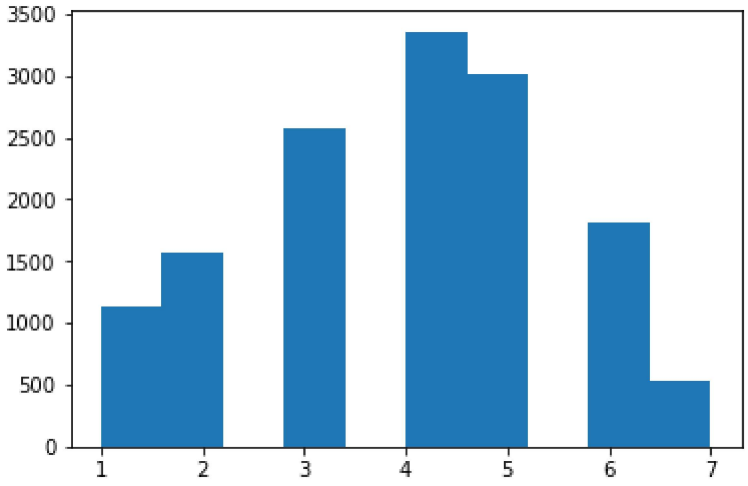
6 [('NS100Training Completed', 1)]



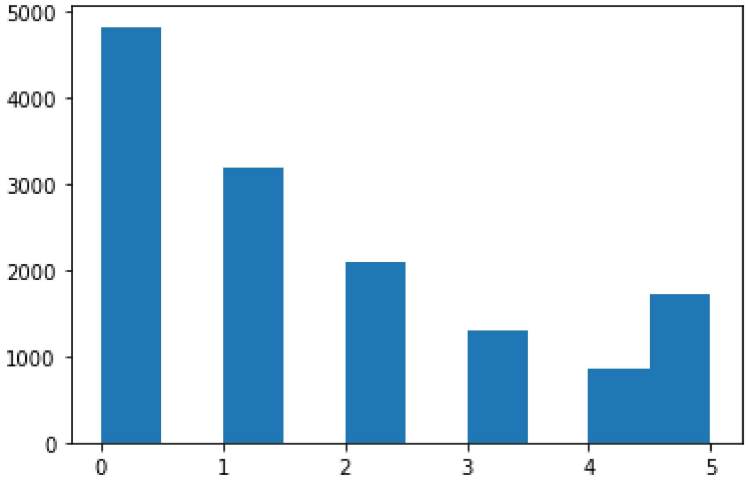
7 [('Social Media Activity (Scaled)', 1)]



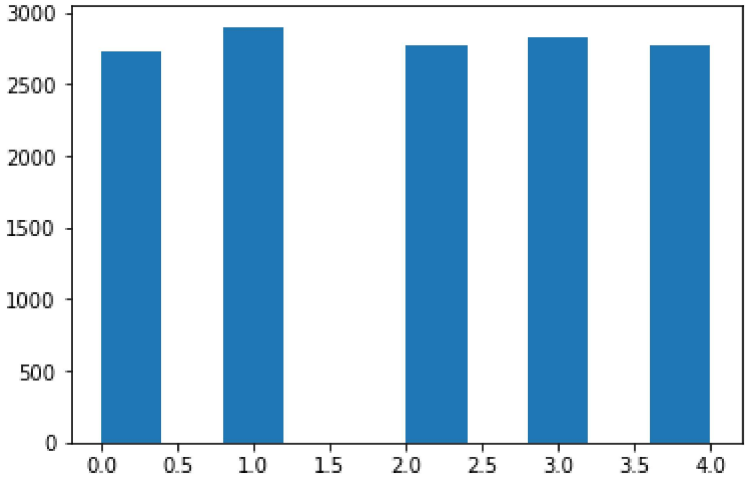
8 [('Survey, Employee Satisfaction (Scaled)', 1)]



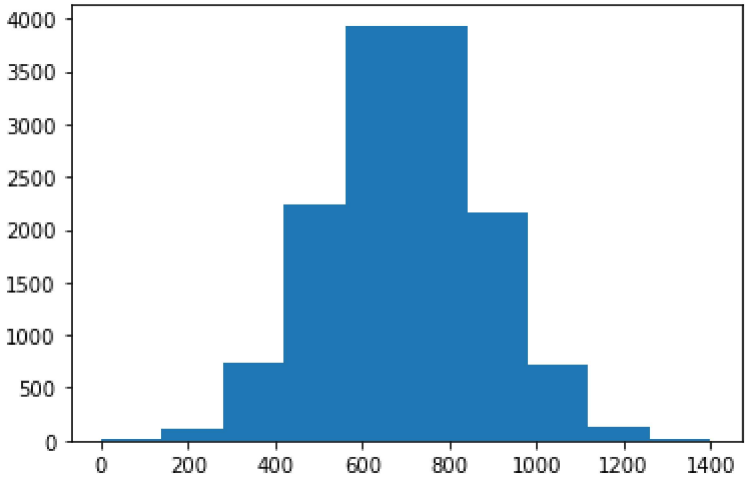
9 [('behaviorPattern3', 1)]



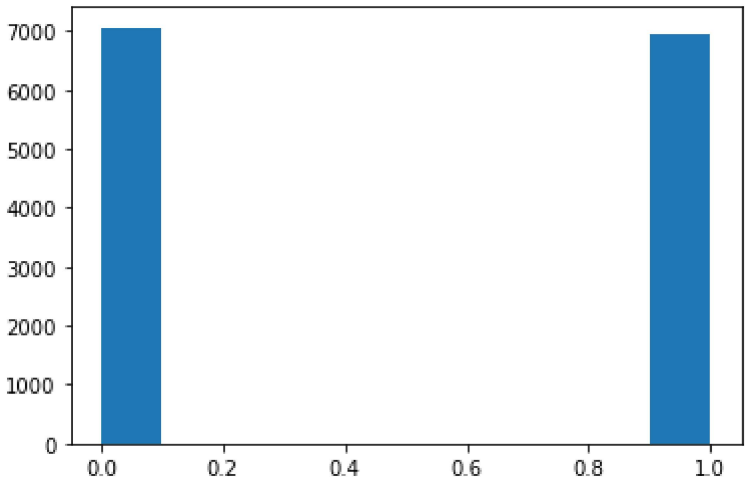
10 [('behaviorPattern5', 1)]



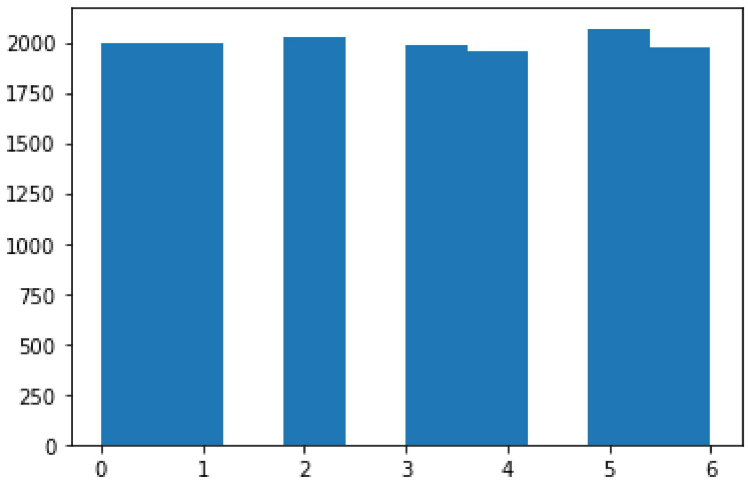
11 [('datumThreatMonitoringScore', 1)]



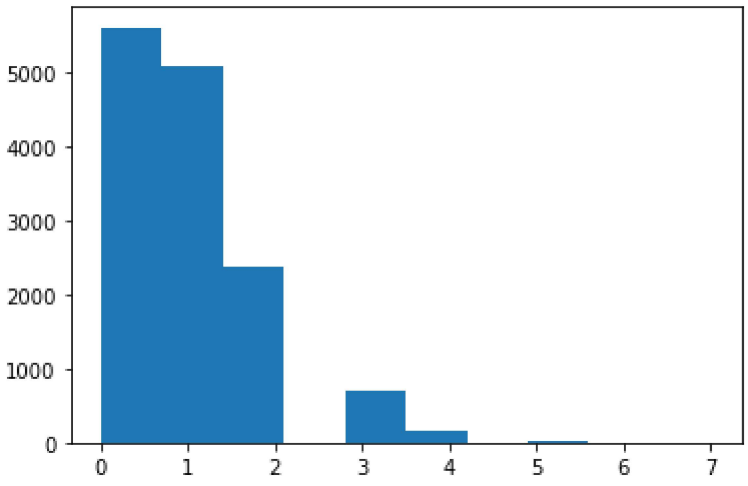
12 [('fraudTraining Completed', 1)]



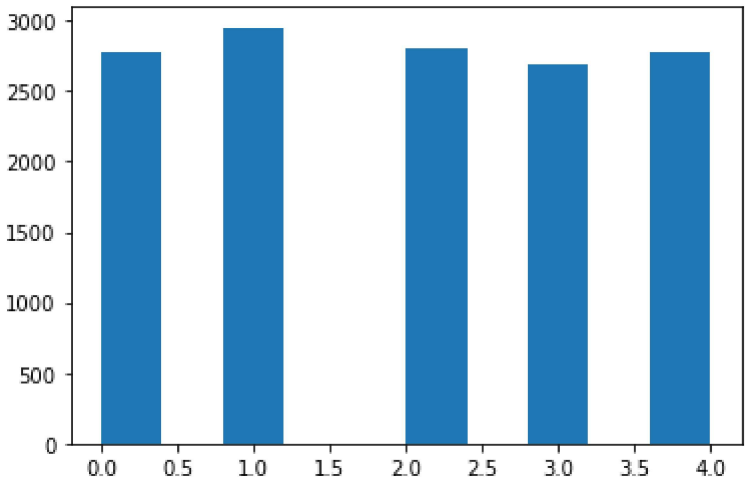
13 [('peerUsageMetric2', 1)]



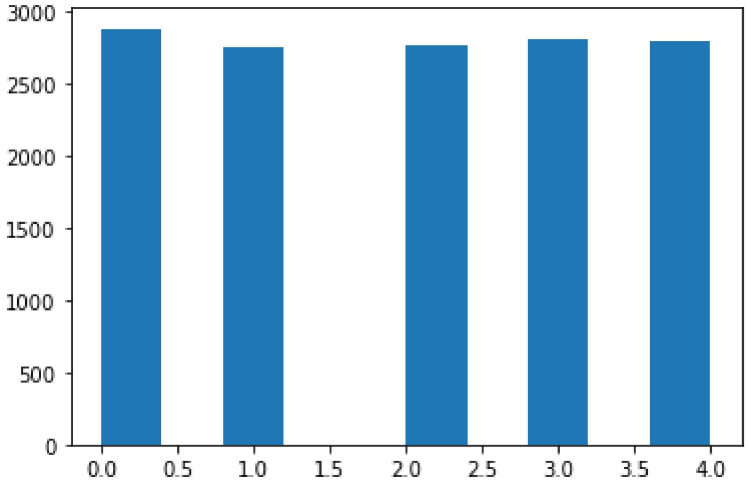
14 [('peerUsageMetric3', 1)]



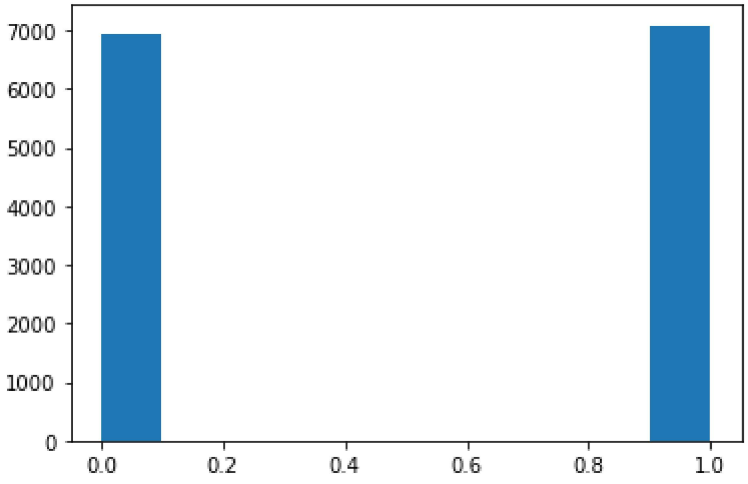
15 [('peerUsageMetric4', 1)]



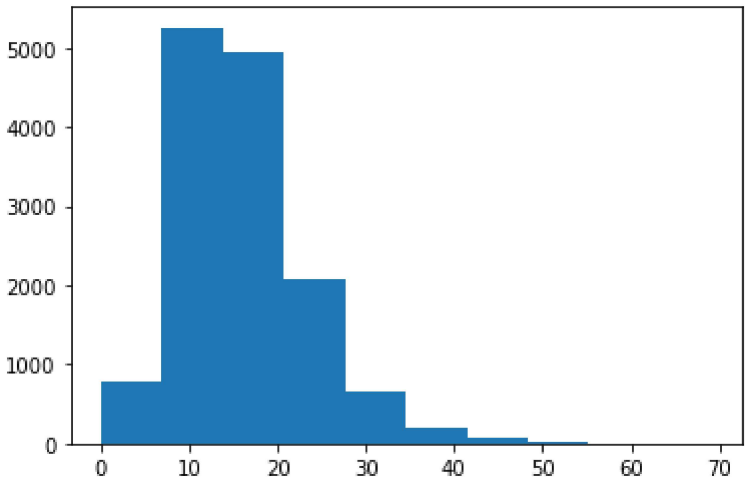
16 [('peerUsageMetric5', 1)]



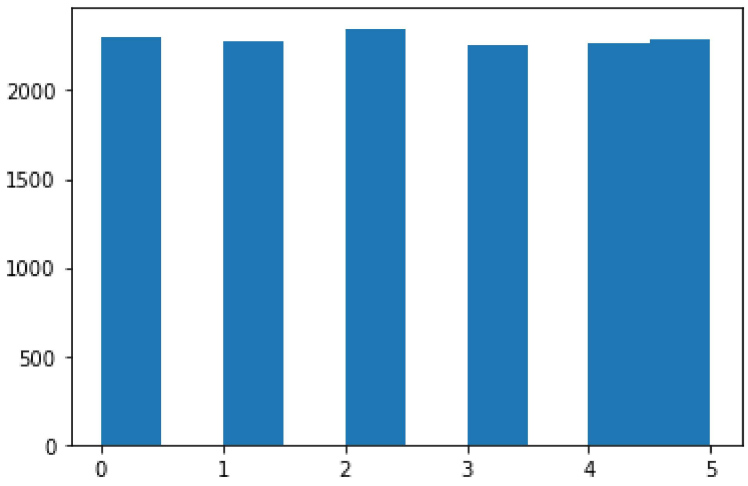
17 [('phishingTraining Completed', 1)]



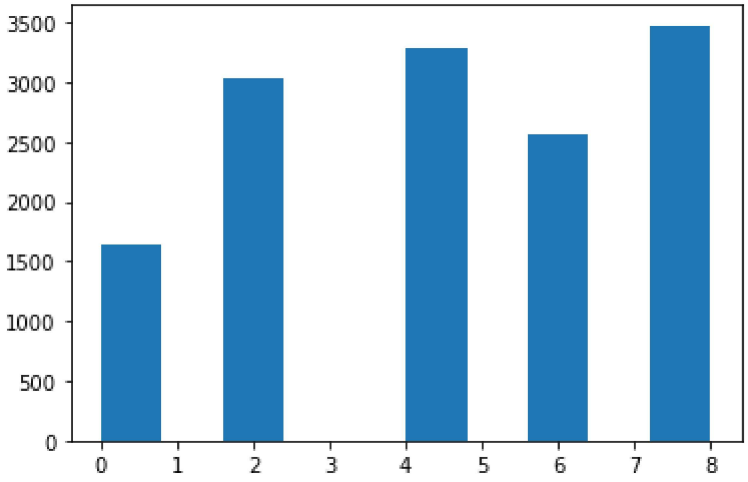

```
18 [('redTeamEval', 1)]
```



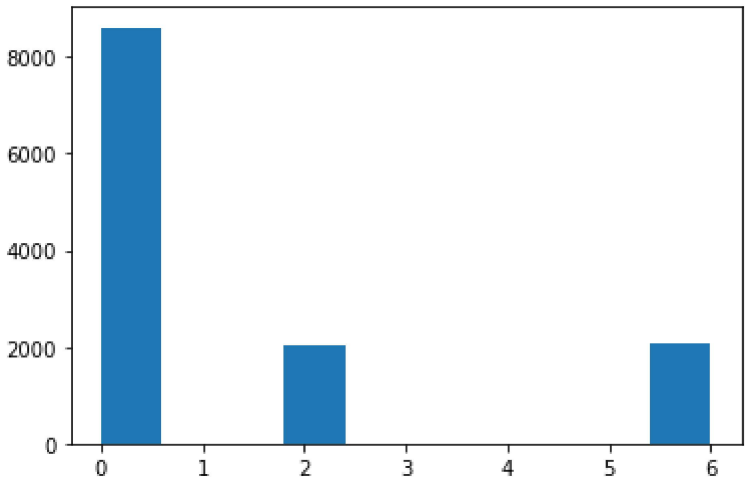

```
19 [('usageMetric1', 1)]
```




```
20 [('usageMetric4', 1)]
```



21 [('usageMetric5', 1)]

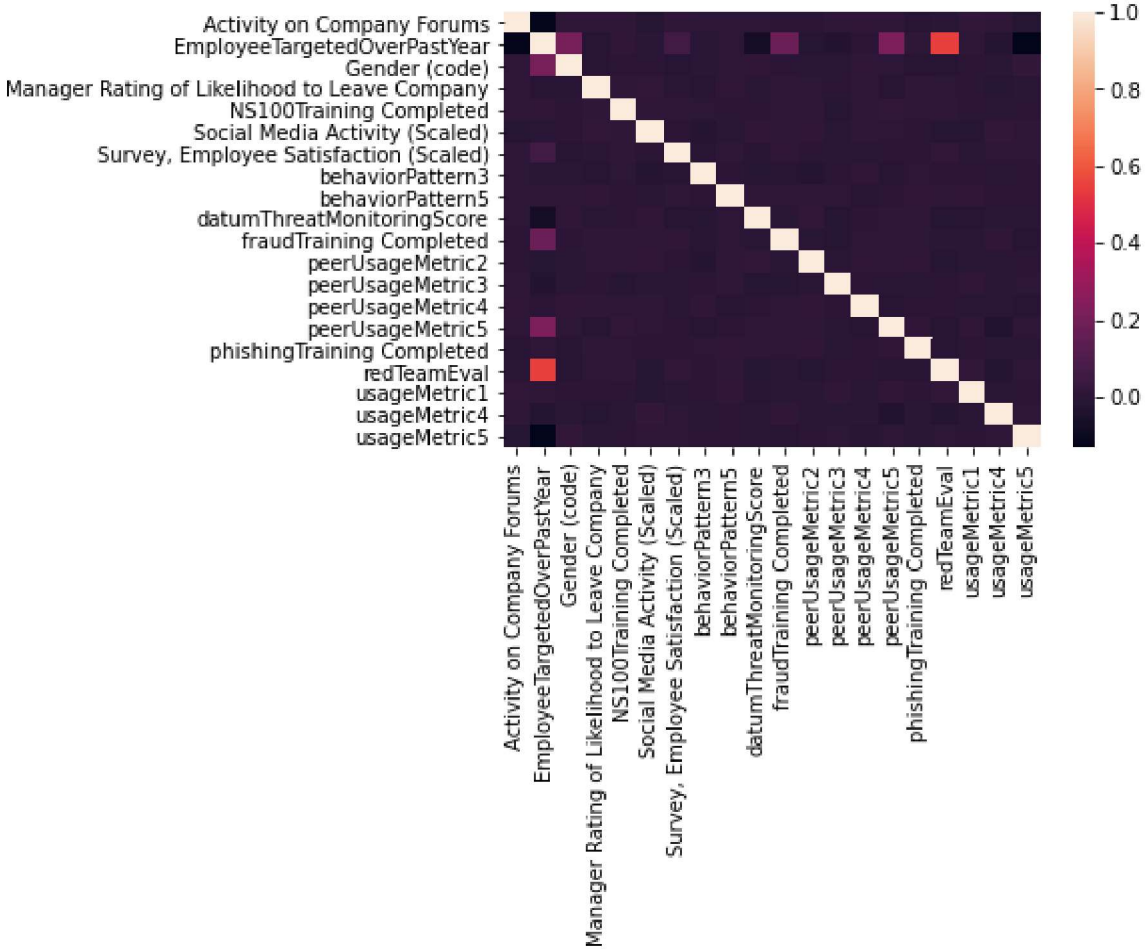



```
Categorical variables:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1
6, 17, 19, 20, 21]

Mean estimation variables:  [18]

Median estimation variables:  [0]

Mode estimation variables:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1
5, 16, 17, 19, 20, 21]
```

In [8]:
```
# Impute missing values
for i in est_with_medians:
    col_name = df.columns[i]
    df[col_name] = df[col_name].fillna(df[col_name].median())

for i in est_with_means:
    col_name = df.columns[i]
    df[col_name] = df[col_name].fillna(df[col_name].mean())

for i in est_with_modes:
    col_name = df.columns[i]
    df[col_name] = df[col_name].fillna(df[col_name].mode()[0])

show_missing(df)  # Should display nothing
```

```
Number of missing values by column:
```

In [9]:
```
# Plotting correlation heatmap
xyz_corr = sb.heatmap(df.corr())

# Displaying heatmap
plt.show()
```



In [10]:
```
# Pre-train prep

# Identify correlated features
corr_features = [col_redteam, col_peer5, col_fraud, col_sat, col_gender]

# Split target from features and drop unused columns
drop_columnns = []
df = df.drop(drop_columnns, axis=1)

X = df.iloc[:, df.columns != target]  # All except target column
y = df.iloc[:, target]  # Target column

# One hot encode the categoricals
for column in categoricals:
    column_title = str(df.columns[column])
    ohe = pd.get_dummies(df[column_title], prefix=column_title)
    X = X.drop([column_title], axis=1)
    X = pd.concat([X, ohe], axis=1)  # Merge the continuous and one-hot frames
together

# Standardize the data
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, rand
om_state=14)

print(r"Working dataframe...")
print(X[:3])
print(y.head())
```

```
Working dataframe...
[[-0.39944777 -1.63092843 -0.28136172 ...  0.64487022 -0.41216968
  -0.41761001]
 [ 1.56107502  1.55670217 -0.28136172 ...  0.64487022 -0.41216968
  -0.41761001]
 [ 0.09068293 -0.79937262  3.55414373 ...  0.64487022 -0.41216968
  -0.41761001]]
0    0.0
1    1.0
2    1.0
3    0.0
4    1.0
Name: EmployeeTargetedOverPastYear, dtype: float64
```

In [17]:
```
# The models and their parameter permutations to be examined
models = [
    {
        "model" : [RandomForestClassifier()],
        "name" : "Random Forest",
        "params" :
        {
            "n_estimators"        : [50],
            "max_features"        : ["sqrt", "log2"],
            "bootstrap"           : [False, True],
            "criterion"           : ['gini', 'entropy'],
            "oob_score"           : [True, False],
            "max_depth"           : [None, 6, 10],
            "max_leaf_nodes"      : [None, 5, 10],
            "min_impurity_split"  : [0.1, 0.3]
        }
    },
    {
        "model" : [lm.RidgeClassifier()],
        "name" : "Ridge Regression",
        "params" :
        {
            "alpha"               : [0.1, 0.5, 1.0],
            "fit_intercept"       : [False],  # Data should already be normal
ized
            "solver"              : ['svd', 'cholesky', 'lsqr', 'sparse_cg',
'sag', 'saga']
        }
    },
    {
        "model" : [lm.LogisticRegression()],
        "name" : "Logistic Regression",
        "params" :
        {
            "penalty"             : ['l1', 'l2', 'elasticnet', 'none'],
            "C"                   : [0.001, 0.1, 1, 10, 100, 10e5],
            "fit_intercept"       : [False],  # Data should already be normal
ized
            "solver"              : ['newton-cg', 'lbfgs', 'liblinear', 'sag'
, 'saga'],
            "l1_ratio"            : [0.0, 0.2, 0.6, 1.0]
       }
    },
    {
        "model" : [lm.PassiveAggressiveClassifier()],
        "name" : "Passive Aggressive",
        "params" :
        {
            "fit_intercept"       : [False],  # Data should already be normal
ized
            "C"                   : [0.001, 0.1, 1, 10, 100, 10e5],
            "early_stopping"      : [True]
        }
    },
    {
        "model" : [DecisionTreeClassifier()],
```

```
        "name" : "Decision Tree",
        "params" :
        {
            "criterion"            : ['gini','entropy'],
            "splitter"             : ['best','random'],
            "class_weight"         : ['balanced', None],
            "max_depth"            : [None, 6, 8, 10],
            "max_features"         : ["sqrt", "log2"]
        }
    },
    {
        "model" : [LinearSVC()],  # Regular SVC is not practical for this much
data
        "name" : "Support Vector Machine",
        "params" :
        {
            "C"                    : [0.001, 0.1, 1, 10, 100, 10e5],
            "class_weight"         : ['balanced'],
            "penalty"              : ['l1', 'l2'],
            "loss"                 : ['hinge', 'squared_hinge'],
            "fit_intercept"        : [False] # Data should already be normali
zed
        }
    },
    {
        "model" : [AdaBoostClassifier()],
        "name" : "Ada Boost",
        "params" :
        {
            "n_estimators"         : [50],
            "learning_rate"        : [0.01, 0.1, 0.5]
        }
    },
    {
        "model" : [lm.SGDClassifier()],
        "name" : "Stochastic Gradient Descent",
        "params" :
        {
            "alpha"                : [0.00001, 0.000001],
            "penalty"              : ['l1', 'l2', 'elasticnet'],
            "l1_ratio"             : [0.0, 0.2, 0.6, 1.0],
            "fit_intercept"        : [False]  # Data should already be normal
ized
        }
    },
    {
        "model" : [LGBMClassifier()],
        "name" : "Gradient Boosted Decision Trees (LGBM)",
        "params" :
        {
            "boosting"             : ['gbdt', 'rf', 'dart', 'goss'],
            "max_depth"            : range (2, 10, 1),
            "learning_rate"        : [0.1, 0.01, 0.05]
        }
    },
    {
        "model" : [XGBClassifier()],
```

```
        "name" : "Extreme Gradient Boosting (XGB)",
        "params" :
        {
            "max_depth"              : range (2, 10, 1),
            "n_estimators"           : range(60, 220, 40),
            "learning_rate"          : [0.1, 0.01, 0.05]
        }
    },
    {
        "model" : [MLPClassifier()],
        "name" : "Multilayer Perceptron",
        "params" :
        {
            "alpha"                  : [0.001, 0.1, 1, 10, 100],
            "activation"             : ['identity', 'logistic', 'tanh', 'relu'
],
            "solver"                 : ['lbfgs', 'sgd', 'adam'],
            "learning_rate"          : ['constant', 'invscaling', 'adaptive'],
            "early_stopping"         : [True]
        }
    }]
```

In [18]:

```
# Grid search for optimal model and parameters


# Minimize false positive = maximize precision
# Minimize false negative = maximize recall
# F-beta is weighted harmonic mean of preicision and recall, so weight ratio r
ecall:precision is 5:1
custom_scorer = make_scorer(fbeta_score, beta=5)

results=[]

for model in models:

    # Classifier
    clf = model['model'][0]
    print("Grid search on", model['name'])

    # Grid search many potential models and parameters to find optimal fit and
save the boss some $$$
    grid = GridSearchCV(clf, param_grid=model['params'], scoring=custom_scorer
, cv=5, n_jobs=-1, verbose=4)
    grid.fit(X_train, y_train.values.ravel())

    # Get metrics of interest for report
    y_pred = grid.predict(X_test)
    pre = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    CM = confusion_matrix(y_test, y_pred)

    # Results for later comparison
    results.append(
        {
            'name'       : model['name'],
            'grid'       : grid,
            'classifier' : grid.best_estimator_,
            'best score' : grid.best_score_,
            'best params': grid.best_params_,
            'precision'  : pre,
            'recall'     : rec,
            'TN'         : CM[0][0],
            'FN'         : CM[1][0],
            'TP'         : CM[1][1],
            'FP'         : CM[0][1],
            'cv'         : grid.cv
        })

    # Save model
    modelFilename = model['name'] + ".pkl"
    os.chdir(wd)

    modelData = results

    modelDataFile = open(modelFilename, 'wb')
    pickle.dump(modelData, modelDataFile)
    modelDataFile.close()
```

```
# Sort result by best score
results = sorted(results, key=lambda x: x['best score'],reverse=True)
```

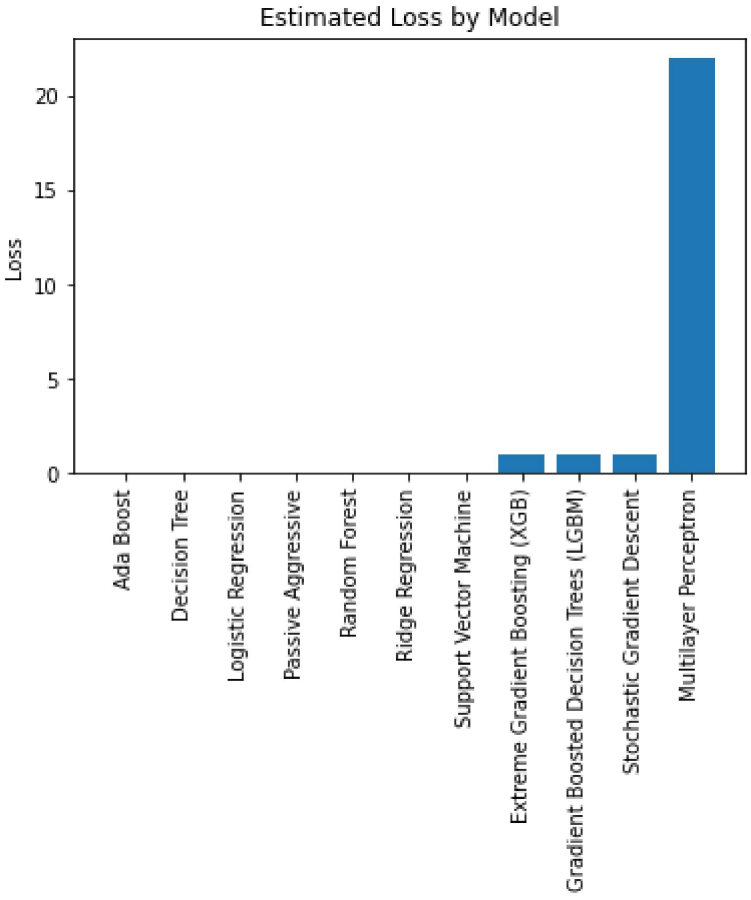
```
Grid search on Multilayer Perceptron
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   37.4s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.2min remaining:   2
1.3s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.3min finished
```

In [31]:
```
p =  r"C:\Code\EdwardFryDataScienceChallenge"
os.chdir(p)

pick = glob.glob(r"*.pkl")

results = []
for result in pick:
    try:
        file = open(result, 'rb')
        r = pickle.load(file)
        file.close()
        r[0]['name'] = result[:-4]
        results.append(r[0])
    except:
        pass

[print(r['name']) for r in results]
```

```
Ada Boost
Decision Tree
Extreme Gradient Boosting (XGB)
Gradient Boosted Decision Trees (LGBM)
Logistic Regression
Multilayer Perceptron
Passive Aggressive
Random Forest
Ridge Regression
Stochastic Gradient Descent
Support Vector Machine
```

Out[31]:
```
[None, None, None, None, None, None, None, None, None, None, None]
```

In [32]:
```
# Create a bar chart to illustrate the most important metric according to the
 business owner - cost

cost = []
xlabels = []
for result in results:
    FP = result['FP']
    FN = result['FN']
    TP = result['TP']
    TN = result['TN']
    score = result['best score']
    loss = (FP + 5*FN)
    cost.append((result['name'], loss))

# Plot the results neatly
cost.sort(key = lambda x: x[1])
xlabels = [x[0] for x in cost]
costvals = [x[1] for x in cost]

N = len(cost)
ind = np.arange(N)    # X locations for the groups

p = plt.bar(ind, costvals)

plt.ylabel('Loss')
plt.title('Estimated Loss by Model')
plt.xticks(ind, xlabels, rotation = 90)

plt.show()
```


Estimated Loss by Model
Loss
0
5
10
15
20
Ada Boost
Decision Tree
Logistic Regression
Passive Aggressive
Random Forest
Ridge Regression
Support Vector Machine
Extreme Gradient Boosting (XGB)
Gradient Boosted Decision Trees (LGBM)
Stochastic Gradient Descent
Multilayer Perceptron

In [33]:
```
# Print out the final top parameters for each model

title = "Optimal parameters"
print(title)
print('='*len(title))

for r in results:
    print(r['name'])

    for i, (k, v) in enumerate(r['best params'].items()):
        print('\t', k, ':', v)
```

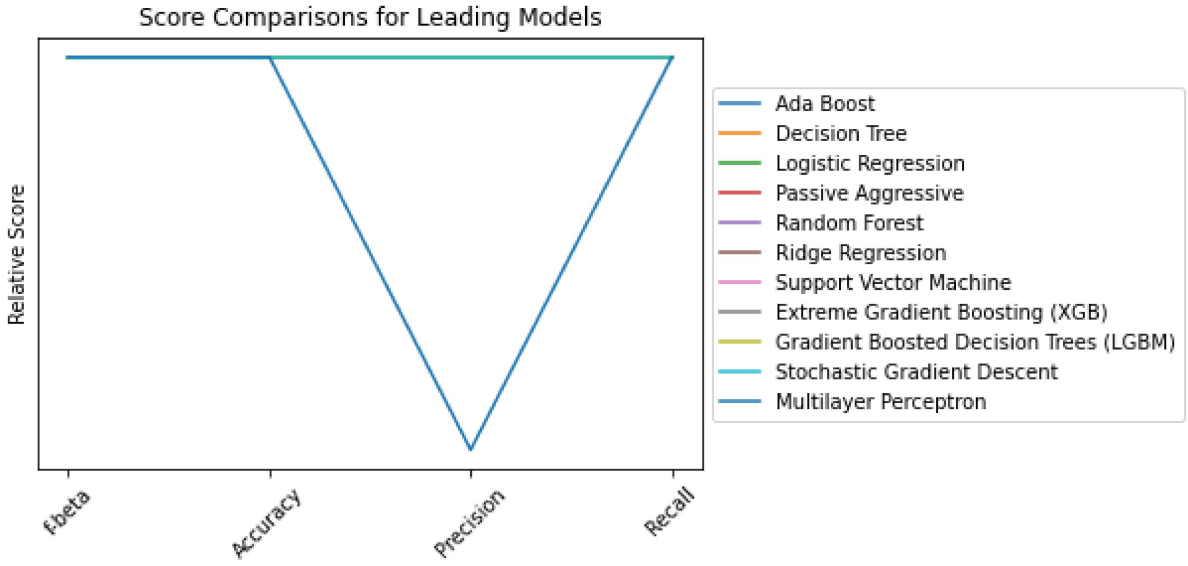
```
Optimal parameters
==================
Ada Boost
         C : 0.001
         class_weight : balanced
         fit_intercept : False
         loss : squared_hinge
         penalty : l2
Decision Tree
         bootstrap : False
         criterion : gini
         max_depth : None
         max_features : sqrt
         max_leaf_nodes : None
         min_impurity_split : 0.1
         n_estimators : 50
         oob_score : False
Extreme Gradient Boosting (XGB)
         alpha : 1e-06
         fit_intercept : False
         l1_ratio : 0.0
         penalty : elasticnet
Gradient Boosted Decision Trees (LGBM)
         alpha : 1e-06
         fit_intercept : False
         l1_ratio : 0.0
         penalty : elasticnet
Logistic Regression
         bootstrap : False
         criterion : gini
         max_depth : None
         max_features : sqrt
         max_leaf_nodes : None
         min_impurity_split : 0.1
         n_estimators : 50
         oob_score : False
Multilayer Perceptron
         activation : tanh
         alpha : 1
         early_stopping : True
         learning_rate : adaptive
         solver : adam
Passive Aggressive
         bootstrap : False
         criterion : gini
         max_depth : None
         max_features : sqrt
         max_leaf_nodes : None
         min_impurity_split : 0.1
         n_estimators : 50
         oob_score : False
Random Forest
         bootstrap : False
         criterion : gini
         max_depth : None
         max_features : sqrt
         max_leaf_nodes : None
```

```
         min_impurity_split : 0.1
         n_estimators : 50
         oob_score : False
Ridge Regression
         bootstrap : False
         criterion : gini
         max_depth : None
         max_features : sqrt
         max_leaf_nodes : None
         min_impurity_split : 0.1
         n_estimators : 50
         oob_score : False
Stochastic Gradient Descent
         alpha : 1e-06
         fit_intercept : False
         l1_ratio : 0.0
         penalty : elasticnet
Support Vector Machine
         C : 0.001
         class_weight : balanced
         fit_intercept : False
         loss : squared_hinge
         penalty : l2
```

In [35]:
```
# Display a neatly formatted table with metrics of interest, sorted from best
 to worst, for easy comparison

tableHeaders = [['Method'],
                ['f-beta'],
                ['Accuracy'],
                ['Precision'],
                ['Recall'],
                ['True Positive'],
                ['True Negative'],
                ['False Positive'],
                ['False Negative'],
                ['Cost($)']]
tableValues = [[r['name'] for r in results],
               [round(r['best score'], 2) for r in results],
               [round((r['TP'] + r['TN'])/(r['TP'] + r['TN'] + r['FP'] + r['F
N']), 2) for r in results],
               [round(r['precision'], 2) for r in results],
               [round(r['recall'], 2) for r in results],
               [r['TP'] for r in results],
               [r['TN'] for r in results],
               [r['FP'] for r in results],
               [r['FN'] for r in results],
               [int(r['FP'] + 5*r['FN']) for r in results]]

# Need to sort by cost because that's what's important to the business owner
tv = np.transpose(tableValues)  # Transpose to rows for sort
tv = sorted(tv, key = lambda x: x[-1])   # Sort by last column
tableValues = np.transpose(tv)  # Transpose back to columns for table display

fig = go.Figure(data=[go.Table(
  header = dict(
    values = tableHeaders,
    line_color='darkslategray',
    fill_color='royalblue',
    align=['left','center'],
    font=dict(color='white', size=12),
    height=40
  ),
  cells=dict(
    values=tableValues,
    line_color='darkslategray',
    fill=dict(color=['paleturquoise', 'white']),
    align=['left', 'center'],
    font_size=12,
    height=30)
    )
])

fig.update_layout(title = {
        'text': "Model Performance by Loss",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
```

```
fig.show()

# Save the table for inclusion in the report
os.chdir(wd)
fig.write_html(r"ModelPerfTable.html")
```

In [36]:
```
# Plot the relative scores of each model against each other for comparison

x = np.arange(4)
xlabels = [['f-beta'],
           ['Accuracy'],
           ['Precision'],
           ['Recall']]

fig=plt.figure()
ax=fig.add_subplot(111)

tv = np.transpose(tableValues)

for v in tv:
    ax.plot(x, [v[1], v[2], v[3], v[4]], label = v[0])

plt.xticks(x, [l[0] for l in xlabels], rotation = 45)
plt.ylabel('Relative Score')
axes = plt.gca()
ax.yaxis.set_major_locator(plt.NullLocator())
plt.gca().invert_yaxis()
plt.title('Score Comparisons for Leading Models')
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
plt.show()
```



In [38]:
```
# Train and save the winning model
model = {
        "model" : [AdaBoostClassifier()],
        "name" : "Ada Boost",
        "params" :
        {
            "n_estimators"          : [50],
            "learning_rate"         : [0.01]
        }
    }
clf = model['model'][0]
print("Training on", model['name'])

# Grid search (for consistency) winning model
grid = GridSearchCV(clf, param_grid=model['params'], scoring=custom_scorer, cv
=5, n_jobs=-1, verbose=4)
grid.fit(X_train, y_train.values.ravel())

# Get metrics of interest
y_pred = grid.predict(X_test)
pre = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
CM = confusion_matrix(y_test, y_pred)

# Save model
modelFilename = r"xyz_phish_model.pkl"
os.chdir(wd)
joblib.dump(grid.best_estimator_, modelFilename, compress = 1)
```

```
Training on Ada Boost
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    4.6s remaining:
6.9s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    4.7s finished
```

Out[38]:
```
['xyz_phish_model.pkl']
```

In [ ]:
```
%%writefile score.py

def init():
    global model
    # AZUREML_MODEL_DIR is an environment variable created during deployment.
    # It is the path to the model folder (./azureml-models/$MODEL_NAME/$VERSIO
N)
    # For multiple models, it points to the folder containing all deployed mod
els (./azureml-models)
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'sklearn_mnist_m
odel.pkl')
    model = joblib.load(model_path)

def run(raw_data):
    data = np.array(json.loads(raw_data)['data'])
    # make prediction
    y_hat = model.predict(data)
    # you can return any data type as long as it is JSON-serializable
    return y_hat.tolist()
```

In [ ]:
```
aciconfig = AciWebservice.deploy_configuration(cpu_cores=8,
                                               memory_gb=8,
                                               tags={"data": "XYZ",  "method"
: "sklearn"},
                                               description='Predict XYZ phishi
ng with sklearn')
```

In [ ]:
```
%%time

ws = Workspace.from_config()
model = Model(ws, 'sklearn_xyz_phishing')


myenv = Environment.get(workspace=ws, name="tutorial-env", version="1")
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)

service = Model.deploy(workspace=ws,
                       name='sklearn-xyz_phishing-svc3',
                       models=[model],
                       inference_config=inference_config,
                       deployment_config=aciconfig)

service.wait_for_deployment(show_output=True)
print(service.scoring_uri)
```

In [ ]:
```
data_folder = os.path.join(os.getcwd(), 'data')
os.makedirs(data_folder, exist_ok=True)

mnist_file_dataset = MNIST.get_file_dataset()
mnist_file_dataset.download(data_folder, overwrite=True)
```

In [ ]:
```
data_folder = os.path.join(os.getcwd(), 'data')
# note we also shrink the intensity values (X) from 0-255 to 0-1. This helps t
he neural network converge faster
X_test = load_data(glob.glob(os.path.join(data_folder,"**/t10k-images-idx3-uby
te.gz"), recursive=True)[0], False) / 255.0
y_test = load_data(glob.glob(os.path.join(data_folder,"**/t10k-labels-idx1-uby
te.gz"), recursive=True)[0], True).reshape(-1)
```

In [ ]:
```
test = json.dumps({"data": X_test.tolist()})
test = bytes(test, encoding='utf8')
y_hat = service.run(input_data=test)
```

In [ ]:
```
conf_mx = confusion_matrix(y_test, y_hat)
print(conf_mx)
print('Overall accuracy:', np.average(y_hat == y_test))
```

In [ ]:
```
# find 30 random samples from test set
n = 30
sample_indices = np.random.permutation(X_test.shape[0])[0:n]

test_samples = json.dumps({"data": X_test[sample_indices].tolist()})
test_samples = bytes(test_samples, encoding='utf8')

# predict using the deployed model
result = service.run(input_data=test_samples)

# compare actual value vs. the predicted values:
i = 0
plt.figure(figsize = (20, 1))

for s in sample_indices:
    plt.subplot(1, n, i + 1)
    plt.axhline('')
    plt.axvline('')

    # use different color for misclassified sample
    font_color = 'red' if y_test[s] != result[i] else 'black'
    clr_map = plt.cm.gray if y_test[s] != result[i] else plt.cm.Greys

    plt.text(x=10, y=-10, s=result[i], fontsize=18, color=font_color)
    plt.imshow(X_test[s].reshape(28, 28), cmap=clr_map)

    i = i + 1
plt.show()
```