Lee Kuczewski
ML Model Assignment 1
3/4/20

ML Model Assignment

Some tunings in the first round:

Random Forest Classifier

Tuning Max_Depth to 3:

```
{'Pos': 12971, 'Neg': 114685, 'TP': 3, 'TN': 114685, 'FP': 0, 'FN': 12968, 'Accuracy': 0.8984
144889390236, 'Precision': 1.0, 'Recall': 0.00023128517462030684, 'desc': 'rdf_train'}
```

Tuning Max_Depth to 500:

MODEL: Random Forest Classifier

In [17]:
```
from sklearn.ensemble import RandomForestClassifier
rdf = RandomForestClassifier(max_depth=500, random_state=0)
rdf.fit(X_train, y_train)

rdf_performance_train = BinaryClassificationPerformance(rdf.predict(X_train), y_train, 'rdf_tra
rdf_performance_train.compute_measures()
print(rdf_performance_train.performance_measures)
```

```
/Users/leekuczewski/Applications/anaconda3/anaconda3/lib/python3.7/site-packages/sklearn/ense
mble/forest.py:245: FutureWarning: The default value of n_estimators will change from 10 in v
ersion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12020, 'TN': 114673, 'FP': 12, 'FN': 951, 'Accuracy': 0.9
924562887760857, 'Precision': 0.9990026595744681, 'Recall': 0.9266825996453627, 'desc': 'rdf_
train'}
```

Tuning Max_Depth to 600:

MODEL: Random Forest Classifier

In [21]:
```
from sklearn.ensemble import RandomForestClassifier
rdf = RandomForestClassifier(max_depth=600, random_state=0)
rdf.fit(X_train, y_train)

rdf_performance_train = BinaryClassificationPerformance(rdf.predict(X_train), y_train, 'rdf_tra
rdf_performance_train.compute_measures()
print(rdf_performance_train.performance_measures)
```

```
/Users/leekuczewski/Applications/anaconda3/anaconda3/lib/python3.7/site-packages/sklearn/ense
mble/forest.py:245: FutureWarning: The default value of n_estimators will change from 10 in v
ersion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12082, 'TN': 114674, 'FP': 11, 'FN': 889, 'Accuracy': 0.9
929498025944726, 'Precision': 0.999090382866121, 'Recall': 0.9314624932541824, 'desc': 'rdf_t
rain'}
```

Tuning Max_Depth to 1000 and random state to 1:

MODEL: Random Forest Classifier

In [45]:
```
from sklearn.ensemble import RandomForestClassifier
rdf = RandomForestClassifier(max_depth=1000, random_state=1)
rdf.fit(X_train, y_train)

rdf_performance_train = BinaryClassificationPerformance(rdf.predict(X_trai
rdf_performance_train.compute_measures()
print(rdf_performance_train.performance_measures)
```

```
/Users/leekuczewski/Applications/anaconda3/anaconda3/lib/python3.7/site-p
ackages/sklearn/ensemble/forest.py:245: FutureWarning: The default value
of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12033, 'TN': 114674, 'FP': 11, 'FN':
938, 'Accuracy': 0.992565958513505, 'Precision': 0.9990866821653935, 'Rec
all': 0.9276848354020507, 'desc': 'rdf_train'}
```

Random Forest with n_estimators = 20:

MODEL: Random Forest Classifier

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
rdf = RandomForestClassifier(n_estimators = 20, bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

rdf.fit(X_train, y_train)

rdf_performance_train = BinaryClassificationPerformance(rdf.predict(X_train), y_train, 'rdf_train')
rdf_performance_train.compute_measures()
print(rdf_performance_train.performance_measures)


# # Creating a bar plot
# sns.barplot(x=, y=)

# # Add labels to your graph
# plt.xlabel('Feature Importance Score')
# plt.ylabel('Features')

# # Add title
# plt.title("Visualizing Important Features")

# # Plot
# plt.legend()
# plt.show()
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12671, 'TN': 114682, 'FP': 3, 'FN': 300, 'Accuracy': 0.9976264335401391, 'Precisi
on': 0.9997632949345115, 'Recall': 0.9768714825379693, 'desc': 'rdf_train'}
```

Random Forest with n_estimators = 20
False negatives 278 and FP 6. More than likely overfit

MODEL: Random Forest Classifier

In [117]:
```
from sklearn.ensemble import RandomForestClassifier
rdf = RandomForestClassifier(n_estimators = 20)

rdf.fit(X_train, y_train)

rdf_performance_train = BinaryClassificationPerformance(rdf.predict(X_trai
rdf_performance_train.compute_measures()
print(rdf_performance_train.performance_measures)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12693, 'TN': 114679, 'FP': 6, 'FN': 2
78, 'Accuracy': 0.9977752710409225, 'Precision': 0.9995275218521144, 'Rec
all': 0.9785675738185182, 'desc': 'rdf_train'}
```

Random Forest with n_estimators = 100 (overfit)
False Negative low, and False Positive Low, but overfit when run on the test data--Doesn't generalize well.

MODEL: Random Forest Classifier

```
In [36]: from sklearn.ensemble import RandomForestClassifier
         rdf = RandomForestClassifier(n_estimators=100)
         rdf.fit(X_train, y_train)

         rdf_performance_train = BinaryClassificationPerformance(rdf.predict(X_train), y_train, 'rdf_train')
         rdf_performance_train.compute_measures()
         print(rdf_performance_train.performance_measures)

         {'Pos': 12971, 'Neg': 114685, 'TP': 12962, 'TN': 114683, 'FP': 2, 'FN': 9, 'Accuracy': 0.9999138309205992, 'Precisio
         n': 0.9998457266275841, 'Recall': 0.9993061444761391, 'desc': 'rdf_train'}
```

Random Forest with max_depth = 1000, n_estimators = 10.

MODEL: Random Forest Classifier

```
In [18]: from sklearn.ensemble import RandomForestClassifier
         rdf = RandomForestClassifier(max_depth= 1000, n_estimators=10)
         rdf.fit(X_train, y_train)

         rdf_performance_train = BinaryClassificationPerformance(rdf.predict(X_train), y_train, 'rdf_train')
         rdf_performance_train.compute_measures()
         print(rdf_performance_train.performance_measures)

         {'Pos': 12971, 'Neg': 114685, 'TP': 11858, 'TN': 114629, 'FP': 56, 'FN': 1113, 'Accuracy': 0.9908425769254873, 'Preci
         sion': 0.9952996474735605, 'Recall': 0.9141932002158661, 'desc': 'rdf_train'}
```

RIDGE REGRESSION

Ridge Regression:
Normalize = True, Alpha = 0.01

MODEL: Ridge Regression Classifier

```
In [100]: from sklearn import linear_model
          rdg = linear_model.RidgeClassifier(normalize=True, alpha=0.01)
          rdg.fit(X_train, y_train)

          rdg_performance_train = BinaryClassificationPerformance(rdg.predict(X_trai
          rdg_performance_train.compute_measures()
          print(rdg_performance_train.performance_measures)

          {'Pos': 12971, 'Neg': 114685, 'TP': 12958, 'TN': 97728, 'FP': 16957, 'F
          N': 13, 'Accuracy': 0.8670646111424454, 'Precision': 0.43316062176165804,
          'Recall': 0.998997764243312, 'desc': 'rdg_train'}
```

Normalize = True, Alpha = 0.02

MODEL: Ridge Regression Classifier

```
In [102]: from sklearn import linear_model
          rdg = linear_model.RidgeClassifier(normalize=True, alpha=0.03)
          rdg.fit(X_train, y_train)

          rdg_performance_train = BinaryClassificationPerformance(rdg.predict(X_train
          rdg_performance_train.compute_measures()
          print(rdg_performance_train.performance_measures)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12956, 'TN': 97860, 'FP': 16825, 'F
N': 15, 'Accuracy': 0.8680829729899104, 'Precision': 0.43504247674691915,
'Recall': 0.9988435741268985, 'desc': 'rdg_train'}
```

Normalize = True, Alpha = 0.3

MODEL: Ridge Regression Classifier

```
In [99]: from sklearn import linear_model
rdg = linear_model.RidgeClassifier(normalize=True, alpha=0.3)
rdg.fit(X_train, y_train)

rdg_performance_train = BinaryClassificationPerformance(rdg.predict(X_train), y_train, 'rdg_train')
rdg_performance_train.compute_measures()
print(rdg_performance_train.performance_measures)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12951, 'TN': 99080, 'FP': 15605, 'FN': 20, 'Accuracy': 0.8776007394873723, 'Preci
sion': 0.45352990614932065, 'Recall': 0.9984580988358647, 'desc': 'rdg_train'}
```

Alpha = 100, brought the false positives down by 10K.

MODEL: Ridge Regression Classifier

```
In [102]: from sklearn import linear_model
rdg = linear_model.RidgeClassifier(normalize=True, alpha=100)
rdg.fit(X_train, y_train)

rdg_performance_train = BinaryClassificationPerformance(rdg.predict(X_train), y_train, 'rdg_train')
rdg_performance_train.compute_measures()
print(rdg_performance_train.performance_measures)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12034, 'TN': 108942, 'FP': 5743, 'FN': 937, 'Accuracy': 0.9476718681456414, 'Prec
ision': 0.6769421162175845, 'Recall': 0.9277619304602575, 'desc': 'rdg_train'}
```

Gradient Descent

Step 3 in Implementation of Gradient Descent:

```
else:
    # otherwise we're not improving, so try shrinking the step size
    iterations_with_no_improvement += 1
    alpha *= 0.9
```

Alpha: += from 0.9

In [14]:
```
# model
#random.seed(0)
# theta = [random.random(), random.random()]
theta = [3, 71]
alpha, beta = minimize_stochastic(squared_error,
  squared_error_gradient,
  x,
  y,
  theta,
  0.0001)
print("alpha", alpha)
print("beta", beta)
```

```
alpha 0.39526087355540895
beta 0.8013581136742408
```

As Alpha decreases to 0.1 → Alpha goes down to .3864 and and beta goes up to .803

In [32]:
```
# model
#random.seed(0)
# theta = [random.random(), random.random()]
theta = [3, 71]
alpha, beta = minimize_stochastic(squared_error,
  squared_error_gradient,
  x,
  y,
  theta,
  0.0001)
print("alpha", alpha)
print("beta", beta)
```

```
alpha 0.3864671894051644
beta 0.803820678064674
```

Alpha_0 = 0.01 and Alpha *= 0.001 → Alpha 0.3850 and beta to .8042 Iterations 1000

In [89]:
```
def minimize_stochastic(target_fn, gradient_fn, x, y, theta_0, alpha_0=0.01):

    data = list(zip(x, y))
    theta = theta_0                           # initial guess
    alpha = alpha_0                           # initial step size
    min_theta, min_value = None, float("inf")   # the minimum so far
    iterations_with_no_improvement = 0

    # if we ever go 100 iterations with no improvement, stop
    while iterations_with_no_improvement < 1000:
        value = sum( target_fn(x_i, y_i, theta) for x_i, y_i in data )

        if value < min_value:
            # if we've found a new minimum, remember it
            # and go back to the original step size
            min_theta, min_value = theta, value
            iterations_with_no_improvement = 0
            alpha = alpha_0
        else:
            # otherwise we're not improving, so try shrinking the step size
            iterations_with_no_improvement += 1
            alpha *= 0.001

        # and take a gradient step for each of the data points
        for x_i, y_i in in_random_order(data):
            gradient_i = gradient_fn(x_i, y_i, theta)
            theta = vector_subtract(theta, scalar_multiply(alpha, gradient_i))

    return min_theta
```

In [91]:
```
# model
#random.seed(0)
# theta = [random.random(), random.random()]
theta = [3, 71]
alpha, beta = minimize_stochastic(squared_error,
  squared_error_gradient,
  x,
  y,
  theta,
  0.0001)
print("alpha", alpha)
print("beta", beta)
```

```
alpha 0.3850216685784432
beta 0.8042361965102908
```

Naive Bayes

Alpha = 1.1

MODEL: Naive Bayes

In [23]:
```
from sklearn.naive_bayes import MultinomialNB
nbs = MultinomialNB(alpha = 1.1)
nbs.fit(X_train, y_train)

nbs_performance_train = BinaryClassificationPerformance(nbs.predict(X_train), y_train, 'nbs_train')
nbs_performance_train.compute_measures()
print(nbs_performance_train.performance_measures)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12700, 'TN': 103337, 'FP': 11348, 'FN': 271, 'Accuracy': 0.9089819514946419, 'Pre
cision': 0.5281104457751165, 'Recall': 0.9791072392259657, 'desc': 'nbs_train'}
```

Alpha = 1.8 FP goes down, FN goes up to 278 accuracy goes up, precision goes up, recall down.

MODEL: Naive Bayes

In [25]:
```
from sklearn.naive_bayes import MultinomialNB
nbs = MultinomialNB(alpha = 1.8)
nbs.fit(X_train, y_train)

nbs_performance_train = BinaryClassificationPerformance(nbs.predict(X_train), y_train, 'nbs_train')
nbs_performance_train.compute_measures()
print(nbs_performance_train.performance_measures)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12693, 'TN': 103459, 'FP': 11226, 'FN': 278, 'Accuracy': 0.9098828100520148, 'Pre
cision': 0.5306659977423805, 'Recall': 0.9785675738185182, 'desc': 'nbs_train'}
```

Perceptron

Added penalty and Alpha = 0.0001

MODEL: Perceptron

In [51]:
```
from sklearn import linear_model
prc = linear_model.SGDClassifier(loss='perceptron', penalty='l2', alpha= 0.0001)
prc.fit(X_train, y_train)

prc_performance_train = BinaryClassificationPerformance(prc.predict(X_train), y_train, 'prc_train')
prc_performance_train.compute_measures()
print(prc_performance_train.performance_measures)
```

```
{'Pos': 12971, 'Neg': 114685, 'TP': 12897, 'TN': 114588, 'FP': 97, 'FN': 74, 'Accuracy': 0.9986604624929498, 'Precisi
on': 0.9925350161613052, 'Recall': 0.9942949656926992, 'desc': 'prc_train'}
```