Homework 3

William Wu

Question 1

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn import svm, datasets, neural_network
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score, train_test_split, GridSea
rchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, Ex
traTreesRegressor, GradientBoostingRegressor
from sklearn import linear_model
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall
_fscore_support, mean_absolute_error
from sklearn.metrics import precision_recall_curve, average_precision_score, r
oc_curve, auc, mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

Data Loading

In [3]:
```
df = pd.read_excel('HW3.xlsx', skiprows=1, header = None)
df.head()
features = df.iloc[:,0:22].values
target = df.iloc[:,24].values
x_train, x_test, y_train, y_test = train_test_split(features, target, test_siz
e= 0.2)
features
```

Out[3]:
```
array([[   1,    1,    0, ..., 3662,    1,    0],
       [   2,    1,    0, ..., 2900,    1,    1],
       [   3,    1,    0, ..., 3914,    0,    0],
       ...,
       [1998,    1,    0, ..., 3394,    0,    0],
       [1999,    1,    0, ...,  253,    0,    1],
       [2000,    1,    0, ..., 1844,    0,    0]], dtype=int64)
```

(a)

Linear Regression

In [18]:
```
param_grid ={'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':
[True, False]}

clf_lr = linear_model.LinearRegression()

grid = GridSearchCV(clf_lr, param_grid, cv = 10, scoring = 'neg_mean_absolute_
error')
grid.fit(x_train,y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test, grid.predict(x_test
)))
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-76.84419544679477
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Tru
e)
Mean Absolute Error:  78.9282734607993
Mean Squared Error:  17799.383455566836
```

k-NN

In [27]:
```
clf_knn = neighbors.KNeighborsRegressor()

k_range = list(range(1,20))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)

grid = GridSearchCV(clf_knn, param_grid, cv = 10, scoring = 'neg_mean_absolute
_error')
grid.fit(x_train,y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test, grid.predict(x_test
)))
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-94.94527793969209
{'n_neighbors': 19, 'weights': 'distance'}
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=19, p=2,
          weights='distance')
Mean Absolute Error:  93.93644867800187
Mean Squared Error:  26909.40331663546
```

Regression Tree

In [25]:
```
complexity_values = range(1,25)
max_nodes = [5, 15, None]
min_purity = [0, 0.5, .1]

param_grid = dict(max_depth = complexity_values,
                  max_leaf_nodes = max_nodes, min_impurity_decrease = min_puri
ty)

clf_dt = tree.DecisionTreeRegressor()

grid = GridSearchCV(clf_dt, param_grid, cv = 10, scoring = 'neg_mean_absolute_
error')
grid.fit(x_train,y_train)



print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test, grid.predict(x_test
)))
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-71.66875285786273
{'max_depth': 6, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1}
DecisionTreeRegressor(criterion='mse', max_depth=6, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.1,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
Mean Absolute Error:  71.5365658351666
Mean Squared Error:  16169.146597326464
```

SVM Regression

In [ ]:
```
Cs = [0.1, 1.0, 10] # SVM regularization parameter

Gammas = [0.01, 0.1, 1.0]
param_grid = {'C': [0.1, 1.0, 10], 'gamma' : [0.01, 0.1, 1.0], 'kernel':['line
ar', 'rbf']}
grid = GridSearchCV(svm.SVR(), param_grid, cv=5, scoring = 'neg_mean_absolute_
error')
grid.fit(x_train, y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test, grid.predict(x_test
)))
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

Neural Network

In [15]:
```
mlpr = neural_network.MLPRegressor()
param_list = {"hidden_layer_sizes": [1,50], "activation": ["identity", "logist
ic", "tanh", "relu"]}
grid = GridSearchCV(estimator=mlpr, param_grid=param_list, cv = 5, scoring =
'neg_mean_absolute_error')
grid.fit(x_train, y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test, grid.predict(x_test
)))
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-92.0630236947674
{'activation': 'identity', 'hidden_layer_sizes': 50}
MLPRegressor(activation='identity', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=50, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
Mean Absolute Error:  107.13040362760985
Mean Squared Error:  29162.450981650185
```

Ensemble Methods - Random Forest

In [20]:
```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10
)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
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}

rf = RandomForestRegressor()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_gr
id, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
rf_random.fit(x_train, y_train)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test, grid.predict(x_test
)))
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   48.6s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  7.2min finished

-76.84419544679477
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Tru
e)
Mean Absolute Error:  78.9282734607993
Mean Squared Error:  17799.383455566836
```

(b)

Data Transformation

In [10]:
```
df2 = pd.read_excel('HW3.xlsx', skiprows=1, header = None)
df2 = df2[df2[23]==1]
df2.head()
features = df.iloc[:,0:22].values
target = df.iloc[:,24].values
x_train2, x_test2, y_train2, y_test2 = train_test_split(features, target, test
_size= 0.2)
```

Linear Regression

In [21]:
```
param_grid ={'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':
[True, False]}

clf_lr = linear_model.LinearRegression()

grid = GridSearchCV(clf_lr, param_grid, cv = 10, scoring = 'neg_mean_absolute_
error')
grid.fit(x_train2,y_train2)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test2, grid.predict(x_test
2)))
print("Mean Squared Error: ",mean_squared_error(y_test2, grid.predict(x_test2
)))
```

```
-79.04038025213856
{'copy_X': True, 'fit_intercept': False, 'normalize': True}
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None,
         normalize=True)
Mean Absolute Error:  74.88739227110489
Mean Squared Error:  13981.918741906476
```

K-NN

In [22]:
```
clf_knn = neighbors.KNeighborsRegressor()

k_range = list(range(1,20))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)

grid = GridSearchCV(clf_knn, param_grid, cv = 10, scoring = 'neg_mean_absolute
_error')
grid.fit(x_train2,y_train2)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test2, grid.predict(x_test
2)))
print("Mean Squared Error: ",mean_squared_error(y_test2, grid.predict(x_test2
)))
```

```
-95.6114089211818
{'n_neighbors': 19, 'weights': 'distance'}
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=19, p=2,
          weights='distance')
Mean Absolute Error:  93.55243906949673
Mean Squared Error:  24551.642715824404
```

Regression Tree

In [23]:
```
complexity_values = range(1,25)
max_nodes = [5, 15, None]
min_purity = [0, 0.5, .1]

param_grid = dict(max_depth = complexity_values,
                  max_leaf_nodes = max_nodes, min_impurity_decrease = min_puri
ty)

clf_dt = tree.DecisionTreeRegressor()

grid = GridSearchCV(clf_dt, param_grid, cv = 10, scoring = 'neg_mean_absolute_
error')
grid.fit(x_train2,y_train2)



print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test2, grid.predict(x_test
2)))
print("Mean Squared Error: ",mean_squared_error(y_test2, grid.predict(x_test2
)))
```

```
-76.78214699475116
{'max_depth': 5, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.5}
DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.5,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
Mean Absolute Error:  76.77525493962408
Mean Squared Error:  20168.69343613505
```

SVM Regression

In [ ]:
```
Cs = [ 0.1, 1.0, 10] # SVM regularization parameter

Gammas = [0.01, 0.1, 1.0]
param_grid = {'C': [ 0.1, 1.0, 10], 'gamma' : [0.01, 0.1, 1.0], 'kernel':['lin
ear', 'rbf']}
grid = GridSearchCV(svm.SVR(), param_grid, cv=5, scoring = 'neg_mean_absolute_
error')
grid.fit(x_train2, y_train2)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test2, grid.predict(x_test
2)))
print("Mean Squared Error: ",mean_squared_error(y_test2, grid.predict(x_test2
)))
```

Neural Network

In [24]:
```
mlpr = neural_network.MLPRegressor()
param_list = {"hidden_layer_sizes": [1,50], "activation": ["identity", "logist
ic", "tanh", "relu"]} #, "solver": ["lbfgs", "sgd", "adam"], "alpha": [0.0000
5,0.0005]}

grid = GridSearchCV(estimator=mlpr, param_grid=param_list, cv = 5,scoring = 'n
eg_mean_absolute_error')
grid.fit(x_train2, y_train2)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Absolute Error: ",mean_absolute_error(y_test, grid.predict(x_test
)))
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-92.54430459667809
{'activation': 'identity', 'hidden_layer_sizes': 50}
MLPRegressor(activation='identity', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=50, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
Mean Absolute Error:  85.57882245920304
Mean Squared Error:  27831.225290051825
```

Ensemble Methods - Random Forest

```
In [25]: n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10
         )]
         max_features = ['auto', 'sqrt']
         max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
         max_depth.append(None)
         min_samples_split = [2, 5, 10]
         min_samples_leaf = [1, 2, 4]
         bootstrap = [True, False]
         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}

         rf = RandomForestRegressor()
         rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_gr
         id, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
         rf_random.fit(x_train2, y_train2)

         print (grid.best_score_)
         print (grid.best_params_)
         print (grid.best_estimator_)
         print("Mean Absolute Error: ",mean_absolute_error(y_test2, grid.predict(x_test
         2)))
         print("Mean Squared Error: ",mean_squared_error(y_test2, grid.predict(x_test2
         )))
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   41.6s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.6min finished

-92.54430459667809
{'activation': 'identity', 'hidden_layer_sizes': 50}
MLPRegressor(activation='identity', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=50, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
Mean Absolute Error:  73.66728669313828
Mean Squared Error:  20557.77195385305
```

(c)

As of 4:45PM, my SVM models have not finished running, despite doing so for the last 2.5 hours. In the interest of time, I'll be elminiating these models on the sheer basis of being computationally inefficient. I believe the code for SVM Regression runs properly in both scenarios, but I do not have the resources to generate results for it.

That being said, I managed to successfully generate 5 of the models.

In part A, the model that generated the best results was the Regression Tree, with a mean absolute error of 71. This was followed by both the Linear Regression and Random Forest models which for some reason generated identical results.

In part B, model with the best results was the Random Forest Model with a Mean Absolute Error of 73. However, the Linear Regression and Regression Tree models both generate a lower Mean Squared Error, which punishes larger errors. The choice for part B depends on what the end user values more.

In general, it appears that Part B results generated less Mean Absolute Error and Mean Squared Error. This may be because removing all the Non-Purchase variables also removed all the 0 values from the target variables. This removed any skew that these number may have been having on the results and allowed the model to fit better.

Question 2

Data Loading and Transformation

In [38]:
```
sb_df = pd.read_csv("spambase.data", header = None)
sb_df.head()
```

Out[38]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.00	0.000	0.0	0.778	0.000	0.
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	0.132	0.0	0.372	0.180	0.
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.01	0.143	0.0	0.276	0.184	0.
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.137	0.0	0.137	0.000	0.
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.135	0.0	0.135	0.000	0.

5 rows × 58 columns

In [39]:
```
features = sb_df.iloc[:,0:56].values
target = sb_df.iloc[:,57].values
x_train, x_test, y_train, y_test = train_test_split(features, target, test_siz
e= 0.2, stratify = target)
```

Decision Tree

In [40]:
```
complexity_values = range(1,25)
weight_options = ["gini", "entropy"]
max_nodes = [5, 15, None]
min_purity = [0, 0.5, .1]

param_grid = dict(max_depth = complexity_values, criterion = weight_options,
                  max_leaf_nodes = max_nodes, min_impurity_decrease = min_puri
ty)

clf_dt = tree.DecisionTreeClassifier()

grid = GridSearchCV(clf_dt, param_grid, cv = 10, scoring = 'accuracy')
grid.fit(x_train,y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test, grid.predict(x_test)))
```

```
0.9192934782608696
{'criterion': 'gini', 'max_depth': 11, 'max_leaf_nodes': None, 'min_impurity_
decrease': 0}
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=11,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
Prediction Accuracy:  0.9239956568946797
```

In [41]:
```
clf_dt = grid.fit(x_train,y_train)
y_predict = clf_dt.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[41]:
```
array([[526,  32],
       [ 35, 328]], dtype=int64)
```

In [42]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[42]:
```
(array([0.93761141, 0.91111111]),
 array([0.94265233, 0.90358127]),
 array([0.94012511, 0.90733057]),
 array([558, 363], dtype=int64))
```

Logistic Regression

In [43]:
```
param_grid ={'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

clf_lr = linear_model.LogisticRegression()

grid = GridSearchCV(clf_lr, param_grid, cv = 10, scoring = 'accuracy')
grid.fit(x_train,y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test, grid.predict(x_test)))
```

```
0.925
{'C': 1}
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
Prediction Accuracy:  0.9305103148751357
```

In [44]:
```
clf_lr = grid.fit(x_train,y_train)
y_predict = clf_lr.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[44]:
```
array([[537,  21],
       [ 43, 320]], dtype=int64)
```

In [45]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[45]:
```
(array([0.92586207, 0.93841642]),
 array([0.96236559, 0.8815427 ]),
 array([0.94376098, 0.90909091]),
 array([558, 363], dtype=int64))
```

K- Nearest Neighbor

In [46]:
```
clf_knn = neighbors.KNeighborsClassifier()

k_range = list(range(1,20))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)

grid = GridSearchCV(clf_knn, param_grid, cv = 10, scoring = 'accuracy')
grid.fit(x_train,y_train)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test, grid.predict(x_test)))
```

```
0.9160326086956522
{'n_neighbors': 6, 'weights': 'distance'}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=6, p=2,
           weights='distance')
Prediction Accuracy:  0.9098805646036916
```

In [47]:
```
clf_knn = grid.fit(x_train,y_train)
y_predict = clf_knn.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[47]:
```
array([[525,  33],
       [ 50, 313]], dtype=int64)
```

In [48]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[48]:
```
(array([0.91304348, 0.90462428]),
 array([0.94086022, 0.86225895]),
 array([0.92674316, 0.88293371]),
 array([558, 363], dtype=int64))
```

SVM

In [54]:
```
Cs = [0.001, 0.01, 0.1, 1.0, 10] # SVM regularization parameter

Gammas = [0.001, 0.01, 0.1, 1.0]
param_grid = {'C': [0.001, 0.01, 0.1, 1.0, 10], 'gamma' : [0.001, 0.01, 0.1,
1.0], 'kernel':['linear', 'rbf']}
grid = GridSearchCV(svm.SVC(), param_grid, cv=10, scoring = 'accuracy')
grid.fit(x_train, y_train)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test, grid.predict(x_test)))
```

```
0.9385869565217392
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Prediction Accuracy:  0.9283387622149837
```

In [55]:
```
clf_svm_linear = grid.fit(x_train,y_train)
y_predict = clf_svm_linear.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[55]:
```
array([[528,  30],
       [ 36, 327]], dtype=int64)
```

In [56]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[56]:
```
(array([0.93617021, 0.91596639]),
 array([0.94623656, 0.90082645]),
 array([0.94117647, 0.90833333]),
 array([558, 363], dtype=int64))
```

Model Selections

No Misclassification Costs

Out of the four classfication models I generated (Decision Tree, Logistic Regression, K-NN, and SVM), the one that produced the highest accuracy is SVM. The overall accuracy however, is not necessarily the most useful metric when comparin the models. The accuracy of SVM also isn't much higher than the other three. Precision and Recall are also important metrics to look at, and these metrics measure the following.

Precision: Out of how many emails we identified as spam, how many were truly spam.

Recall: Our of all the emails that are truly spam, how many of them did we correctly identify.

In this case, I'll be using recall as the more important metric. Emails that get flagged as spam can be further reviewed by a user checking the spam inbox, while failed identification can lead to annoyances at best and security issues or scams at worst. The Logisitic Regression model in this case has the highest recall at 96% without trading off much of the accuracy from SVM. Both accuracies are in the 94% range, and the precision also is not that much lower. Without misclassification costs, I would choose Logistic Regression.

With Misclassification Costs

With Misclassification costs, the game changes a little bit. False positives are the emails that the model identifies as spam, but are not, while false negatives are emails in which the model says are not spam but really are. The more important one depends on what standpoint you're looking from. False positives mean that a user might miss some important emails that get flagged into your spam box and false negatives might pose a security risk with the severity depending on the context.

In this case, I'll treat false negatives as the more important case. A rational user would occassionally check their spam box for any emails that got unfortunately flagged if they were expecting something important, while they would likely trust the spam filter to catch spam emails and could be caught off guard by malicious emails even if they're being careful. With my experience in information security, I see the latter case being more common and costly than the first case, and I prefer erring on the side of caution.

We can apply the cost ratio of 10:1 to each of the confusion matricies generated by the models. From there we can calculate the misclassification costs for each of the models.

Decision Tree: 382

Logistic Regression: 451

KNN: 533

SVM: 390

In this case, the model that generated the lowest cost was The decision tree, with SVM coming in a close second. The logisitic regression model which I chose for the evaluation without the cost produced a much higher misclassification cost. In choosing between the Decision Tree and SVM models however, the SVM model shows a greater accuracy and similar precision, recall, and fscore measure while not trading off as much cost. With this information, I would give the edge to SVM, even if Decision Tree costs slightly lower.