Homework 2

William Wu

Question 1

With the recent Trump rally and its corresponding protest that took place in Minneapolis last week, I started thinking about the effect it would have on the city and the local businesses. Whenever a political candidate comes to town, there will always be people who come in from outside the city who come in to see them as well as people who come in to protest. There will also be people who avoid the city, like my father who chose not to go into work that day to avoid the massive influx of people.

a)
There are a couple of business decisions that might be affected. The level of inventory a store might have to maintain, the revenues of the businesses surounding the venue, tax dollars spent on security by the city, and many other factors could be shaped by the event. The key decision to be made is how to prepare for the changes in traffic.

b)
We would use this model during election season scheduled around when a political candidate decides to hold a rally in a city.

c)
I mentioned this before, but a political rally undoubtedly brings in a different crowd of people. Some come in from out of town, others specifically avoid the town. Whatever the case, the key demographic of people shifts. There will very likely be an effect on the local businesses. Hotels might see an increased capacity, stationary stores might see more business for printing signage, the city itself might close down streets which might hurt businesses surrounding the venue. There might be other reasons and effects, but the key point is that things change.

d)
Politicians have held rallies whenever the election cycle comes about. City and state governments will likely have access to a large amount of data about streets around the venue area. The local businesses themselves will have sales and revenue data in both days surrounding the rally and days that are nowhere near the rally for control.

e)
By forecasting the effects of a political rally, the local businesses can be better prepared to handle them. The can stock more or less inventory in specific days, offer discounted rates to get people in the store, or just close down entirely for the day.

f)
The more pressing quantity to predict is revenue, followed by foot traffic. Additionally, demographics will likely change, which might effect certain businesses who don't see or see more of a certain type of people.

g)
This is a probability estimation problem, assuming the revenues gained and lost are put into buckets. We can use the information surrounding the rally to estimate the probability that a business would gain or lose money from the political candidate coming into town.

h)
Potential features would be revenue, foot traffic, candidate political affiliation, size of venue, days before and/or after rally.

i)
Training data would include the features listed above for days surrounding the rally as well as dates completely unrelated to the rally.

Question 2

A)
At probability cutoff model of 0.3 generates 2 errors out of the 10 records (the records at 0.67 and 0.46). This gives an overall accuracy of 80%.

At cutoff of 0.8, there are 3 errors (the records at at 0.75, 0.61, and 0.42). This is an overall accuracy of 70%).

B)
For 100% precision, all fraud predictions need to be correct. A cutoff at 0.68 achieves this purpose since going any lower would predict a non-fraud case to be fraud.

In this model, there are 2 errors at 0.61 and 0.42 for an 80% overall accuracy.

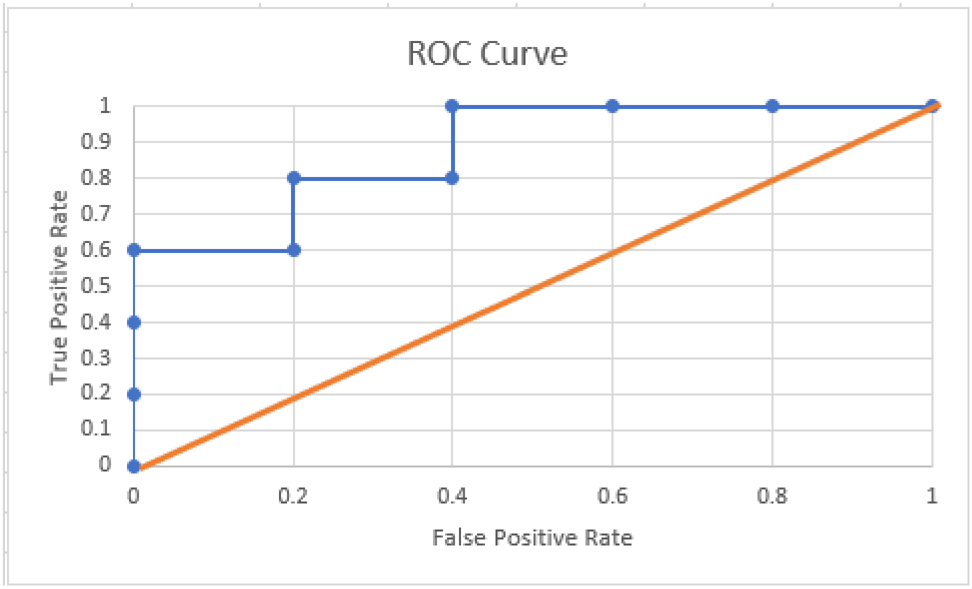
C)

For 100% recall, all we need to capture all true cases of fraud. 0.41 is probably the best cutoff to achieve this. Any higher and we miss the 0.42 fraud case and going lower might decrease the accuracy of our model more than necessary.

The accuracy of this model is also 80% (2 errors at 0.46 and 0.67).

D)

The ROC plot is created by gradually decreasing the cutoff and calculating the TP and FP rates at each step.



Where the blue line is the fraud ROC line and the red line is the random line.

E)

This is kind of a similar approach as the previous problem. Find the number of false negatives and false positives and match them to the cost structures. Assuming each value in the cost structure is a dollar. Starting from 0.0 and moving to 1 at each cutoff threshold, the costs at each cutoff for each cost structure are as follows:

Cutoffs (Inclusive):
0, 0.04, 0.9, 0.25, 0.42, 0.46, 0.61, 0.67, 0.75, 0.91, 0.95, 1

False Negatives (Count):
0, 0, 0, 0, 1, 1, 2, 2, 3, 4, 5

False Positives (Count):
5, 4, 3, 2, 2, 1, 1, 0, 0, 0, 0

Structure 1:
5, 4, 3, 2, 3, 2, 3, 2, 3, 4, 5
The lowest costs are at cutoffs 0.25, 0.46, and 0.67 with a cost of $2.

Structure 2:
10, 8, 6, 4, 5, 3, 4, 2, 3, 4, 5 The lowest cost here is the cutoff at 0.67 with a cost of $2.

Structure 3:
5, 4, 3, 2, 4, 3, 5, 4, 6, 8, 10
The lowest cost here is the cost at 0.24 with a cost of $2.

Question 3

MAE (Mean Absolute Error)

```
In [7]: ((710-670)+(680-660)+(600-550)+(800-740)+(700-600))/5
```

```
Out[7]: 54.0
```

MAPE (Mean Absolute Percentage Error)

```
In [13]: round(((710-670)/670+(680-660)/680+(600-550)/550+(800-740)/740+(700-600)/700)/
5,5)
```

```
Out[13]: 0.08079
```

Root Mean Squared Error

```
In [23]: (((710-670)**2+(660-680)**2+(600-550)**2+(800-740)**2+(600-700)**2)/5)**(1/2)
```

```
Out[23]: 60.166435825965294
```

Average Error

```
In [15]: ((710-670)+(660-680)+(600-550)+(800-740)+(600-700))/5

Out[15]: 6.0
```

Question 4

A)
Overfitting happens when a model is too specific to the training data from which it was built. An overfitted model tends to fail when applied to a more general use case or data that was not in the training model.

B)
The key difference between the two is that in SVM Classification, more nodes outside the margin is better where as in SVM Regression, more nodes inside is better. Classification is trying to find the margin that separates the points the best while Regression tries to minimize it.

Question 5

The Code for the decision tree, k-nn, and logistic regression comes from my previous homework submission.

```
In [59]: 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 import svm, datasets
         from sklearn import preprocessing
         from sklearn.model_selection import cross_val_score, train_test_split, GridSea
         rchCV
         from sklearn import linear_model
         from sklearn.naive_bayes import GaussianNB
         from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall
         _fscore_support
         from sklearn.metrics import precision_recall_curve, average_precision_score, r
         oc_curve, auc
         import warnings
         warnings.filterwarnings('ignore')
```

```
In [44]: bc_df = pd.read_csv("wdbc.data", header = None)
         features = bc_df.iloc[:,2:33].values
         target = bc_df.iloc[:,1].values
         x_train, x_test, y_train, y_test = train_test_split(features, target, test_siz
         e= 0.2, stratify = target)
```

Decision Tree

```
In [3]: 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)))
```

```
0.9406593406593406
{'criterion': 'entropy', 'max_depth': 14, 'max_leaf_nodes': None, 'min_impuri
ty_decrease': 0}
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=14,
            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:  0.9385964912280702
```

```
In [4]: clf_dt = grid.fit(x_train,y_train)
        y_predict = clf_dt.predict(x_test)
        confusion_matrix(y_test,y_predict)
```

```
Out[4]: array([[68,  4],
               [ 2, 40]], dtype=int64)
```

In [5]:
```
precision_recall_fscore_support(y_test, y_predict)
```
Out[5]:
```
(array([0.97142857, 0.90909091]),
 array([0.94444444, 0.95238095]),
 array([0.95774648, 0.93023256]),
 array([72, 42], dtype=int64))
```

Logisitic Regression

In [48]:
```
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)))
```
```
0.9582417582417583
{'C': 100}
LogisticRegression(C=100, 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:  0.9736842105263158
```

In [49]:
```
clf_lr = grid.fit(x_train,y_train)
y_predict = clf_lr.predict(x_test)
confusion_matrix(y_test,y_predict)
```
Out[49]:
```
array([[71,  1],
       [ 2, 40]], dtype=int64)
```

In [50]:
```
precision_recall_fscore_support(y_test, y_predict)
```
Out[50]:
```
(array([0.97260274, 0.97560976]),
 array([0.98611111, 0.95238095]),
 array([0.97931034, 0.96385542]),
 array([72, 42], dtype=int64))
```

K-Nearest Neighbor

In [45]:
```
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)))
```

```
0.9318681318681319
{'n_neighbors': 5, 'weights': 'distance'}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='distance')
Prediction Accuracy:  0.9298245614035088
```

In [46]:
```
clf_knn = grid.fit(x_train,y_train)
y_predict = clf_knn.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[46]:
```
array([[71,  1],
       [ 7, 35]], dtype=int64)
```

In [47]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[47]:
```
(array([0.91025641, 0.97222222]),
 array([0.98611111, 0.83333333]),
 array([0.94666667, 0.8974359 ]),
 array([72, 42], dtype=int64))
```

SVM

In [29]:
```
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)))
```

```
0.9516483516483516
{'C': 10, 'gamma': 0.001, 'kernel': 'linear'}
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Prediction Accuracy:  0.956140350877193
```

In [30]:
```
clf_svm_linear = grid.fit(x_train,y_train)
y_predict = clf_svm_linear.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[30]:
```
array([[69,  3],
       [ 2, 40]], dtype=int64)
```

In [31]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[31]:
```
(array([0.97183099, 0.93023256]),
 array([0.95833333, 0.95238095]),
 array([0.96503497, 0.94117647]),
 array([72, 42], dtype=int64))
```

Much like in the previous instance of this problem in homework 1, recall is still the most important metric. We want to successfully detect cases of breast cancer and allowing it to go undetected could be a potentially fatal error. Precision error only means that a patient pays a little extra money to err on the side of caution while accuracy is more of a measurement of the model being viable as a whole.

Even with the SVM model in comparison, the best model is still the logisitic regression model since it has the highest recall while not trading off much precision.

Question 6

In [136]:
```
car_df = pd.read_csv("car.data", header = None)
car_df = car_df.apply(preprocessing.LabelEncoder().fit_transform)
features = car_df.iloc[:,0:6]
target = car_df.iloc[:,6]
car_df.head()
```

Out[136]:

	0	1	2	3	4	5	6
0	3	3	0	0	2	1	2
1	3	3	0	0	2	2	2
2	3	3	0	0	2	0	2
3	3	3	0	0	1	1	2
4	3	3	0	0	1	2	2

In [137]:
```
features = car_df.iloc[:,0:6]
target = car_df.iloc[:,6]
x_train, x_test, y_train, y_test = train_test_split(features, target, test_siz
e= 0.2, stratify = target)
```

In [115]:
```
car_df2 = car_df.copy()
car_df2 = car_df2.apply(preprocessing.LabelEncoder().fit_transform)
features2 = car_df2.iloc[:,0:6]
target2 = car_df2.iloc[:,6]
enc = preprocessing.OneHotEncoder()
features2 = enc.fit(features2).transform(features2).toarray()
print(enc.categories_)
target2.unique()
```

```
[array([0., 1., 2., 3.]), array([0., 1., 2., 3.]), array([0., 1., 2., 3.]), a
rray([0., 1., 2.]), array([0., 1., 2.]), array([0., 1., 2.])]
```

Out[115]:
```
array([2, 0, 3, 1], dtype=int64)
```

In [139]:
```
x_train2, x_test2, y_train2, y_test2 = train_test_split(features2, target2, te
st_size= 0.2, stratify = target2)
```

Decision Tree (Regular)

In [140]:
```
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)))
```

```
0.984081041968162
{'criterion': 'entropy', 'max_depth': 24, 'max_leaf_nodes': None, 'min_impuri
ty_decrease': 0}
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=24,
            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:  0.9855491329479769
```

In [141]:
```
clf_dt = grid.fit(x_train,y_train)
y_predict = clf_dt.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[141]:
```
array([[ 73,   2,   2,   0],
       [  0,  13,   1,   0],
       [  0,   0, 242,   0],
       [  1,   0,   0,  12]], dtype=int64)
```

In [142]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[142]:
```
(array([0.98648649, 0.86666667, 0.9877551 , 1.        ]),
 array([0.94805195, 0.92857143, 1.        , 0.92307692]),
 array([0.96688742, 0.89655172, 0.99383984, 0.96      ]),
 array([ 77,  14, 242,  13], dtype=int64))
```

Decision Tree (One Hot Encoded)

In [119]:
```
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_train2,y_train2)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test2, grid.predict(x_test2)))
```

```
0.975397973950796
{'criterion': 'entropy', 'max_depth': 14, 'max_leaf_nodes': None, 'min_impuri
ty_decrease': 0}
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=14,
            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:  0.9797687861271677
```

In [120]:
```
clf_dt = grid.fit(x_train2,y_train2)
y_predict2 = clf_dt.predict(x_test2)
confusion_matrix(y_test2,y_predict2)
```

Out[120]:
```
array([[ 75,   1,   1,   0],
       [  0,  14,   0,   0],
       [  2,   0, 240,   0],
       [  0,   3,   0,  10]], dtype=int64)
```

In [121]:
```
precision_recall_fscore_support(y_test2, y_predict2)
```

Out[121]:
```
(array([0.97402597, 0.77777778, 0.99585062, 1.        ]),
 array([0.97402597, 1.        , 0.99173554, 0.76923077]),
 array([0.97402597, 0.875     , 0.99378882, 0.86956522]),
 array([ 77,  14, 242,  13], dtype=int64))
```

K-Nearest Neighbor (Regular)

In [143]:
```
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)))
```

```
0.9276410998552822
{'n_neighbors': 7, 'weights': 'distance'}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=7, p=2,
           weights='distance')
Prediction Accuracy:  0.9104046242774566
```

In [144]:
```
clf_knn = grid.fit(x_train,y_train)
y_predict = clf_knn.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[144]:
```
array([[ 57,   0,  18,   2],
       [  8,   6,   0,   0],
       [  1,   0, 241,   0],
       [  1,   1,   0,  11]], dtype=int64)
```

In [145]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[145]:
```
(array([0.85074627, 0.85714286, 0.93050193, 0.84615385]),
 array([0.74025974, 0.42857143, 0.99586777, 0.84615385]),
 array([0.79166667, 0.57142857, 0.96207585, 0.84615385]),
 array([ 77,  14, 242,  13], dtype=int64))
```

K-Nearest Neighbor (One_Hot Encoded)

In [122]:
```
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_train2,y_train2)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test2, grid.predict(x_test2)))
```

```
0.9276410998552822
{'n_neighbors': 9, 'weights': 'uniform'}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=9, p=2,
           weights='uniform')
Prediction Accuracy:  0.9421965317919075
```

In [124]:
```
clf_knn = grid.fit(x_train2,y_train2)
y_predict2 = clf_knn.predict(x_test2)
confusion_matrix(y_test2,y_predict2)
```

Out[124]:
```
array([[ 69,   1,   7,   0],
       [  7,   7,   0,   0],
       [  0,   0, 242,   0],
       [  4,   1,   0,   8]], dtype=int64)
```

In [125]:
```
precision_recall_fscore_support(y_test2, y_predict2)
```

Out[125]:
```
(array([0.8625    , 0.77777778, 0.97188755, 1.        ]),
 array([0.8961039 , 0.5       , 1.        , 0.61538462]),
 array([0.87898089, 0.60869565, 0.98574338, 0.76190476]),
 array([ 77,  14, 242,  13], dtype=int64))
```

Logistic Regression (Regular)

In [150]:
```
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)))
```

```
0.7076700434153401
{'C': 0.01}
LogisticRegression(C=0.01, 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:  0.6994219653179191
```

In [151]:
```
clf_knn = grid.fit(x_train,y_train)
y_predict = clf_knn.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[151]:
```
array([[  6,   0,  71,   0],
       [  0,   0,  14,   0],
       [  6,   0, 236,   0],
       [  4,   0,   9,   0]], dtype=int64)
```

In [152]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[152]:
```
(array([0.375     , 0.        , 0.71515152, 0.        ]),
 array([0.07792208, 0.        , 0.97520661, 0.        ]),
 array([0.12903226, 0.        , 0.82517483, 0.        ]),
 array([ 77,  14, 242,  13], dtype=int64))
```

Logistic Regression (One_Hot Encoded)

In [163]:
```
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_train2,y_train2)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test2, grid.predict(x_test2)))
```

```
0.894356005788712
{'C': 100}
LogisticRegression(C=100, 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:  0.8815028901734104
```

In [164]:
```
clf_knn = grid.fit(x_train2,y_train2)
y_predict2 = clf_knn.predict(x_test2)
confusion_matrix(y_test2,y_predict2)
```

Out[164]:
```
array([[ 57,   3,  17,   0],
       [ 12,   2,   0,   0],
       [  8,   0, 234,   0],
       [  1,   0,   0,  12]], dtype=int64)
```

In [165]:
```
precision_recall_fscore_support(y_test2, y_predict2)
```

Out[165]:
```
(array([0.73076923, 0.4       , 0.93227092, 1.        ]),
 array([0.74025974, 0.14285714, 0.96694215, 0.92307692]),
 array([0.73548387, 0.21052632, 0.94929006, 0.96      ]),
 array([ 77,  14, 242,  13], dtype=int64))
```

Naive Bayes (Regular)

In [168]:
```
gnb = GaussianNB()
clf_nb = gnb.fit(x_train,y_train)
y_predict = clf_nb.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[168]:
```
array([[  5,   0,  31,  41],
       [  3,   0,   4,   7],
       [  5,   0, 195,  42],
       [  0,   0,   0,  13]], dtype=int64)
```

In [169]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[169]:
```
(array([0.38461538, 0.        , 0.84782609, 0.12621359]),
 array([0.06493506, 0.        , 0.80578512, 1.        ]),
 array([0.11111111, 0.        , 0.82627119, 0.22413793]),
 array([ 77,  14, 242,  13], dtype=int64))
```

Naive Bayes (One_Hot Encoded)

In [173]:
```
gnb = GaussianNB()
clf_nb = gnb.fit(x_train2,y_train2)
y_predict2 = clf_nb.predict(x_test2)
confusion_matrix(y_test2,y_predict2)
```

Out[173]:
```
array([[ 57,  18,   0,   2],
       [  0,  12,   0,   2],
       [ 46,   2, 194,   0],
       [  0,   0,   0,  13]], dtype=int64)
```

In [174]:
```
precision_recall_fscore_support(y_test2, y_predict2)
```

Out[174]:
```
(array([0.55339806, 0.375     , 1.        , 0.76470588]),
 array([0.74025974, 0.85714286, 0.80165289, 1.        ]),
 array([0.63333333, 0.52173913, 0.88990826, 0.86666667]),
 array([ 77,  14, 242,  13], dtype=int64))
```

SVM (Regular)

In [147]:
```
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)))
```

```
0.9790159189580319
{'C': 10, 'gamma': 1.0, 'kernel': 'rbf'}
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1.0, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Prediction Accuracy:  0.9797687861271677
```

In [148]:
```
clf_svm_linear = grid.fit(x_train,y_train)
y_predict = clf_svm_linear.predict(x_test)
confusion_matrix(y_test,y_predict)
```

Out[148]:
```
array([[ 72,   0,   4,   1],
       [  0,  14,   0,   0],
       [  1,   0, 241,   0],
       [  1,   0,   0,  12]], dtype=int64)
```

In [149]:
```
precision_recall_fscore_support(y_test, y_predict)
```

Out[149]:
```
(array([0.97297297, 1.        , 0.98367347, 0.92307692]),
 array([0.93506494, 1.        , 0.99586777, 0.92307692]),
 array([0.95364238, 1.        , 0.98973306, 0.92307692]),
 array([ 77,  14, 242,  13], dtype=int64))
```

SVM (One_Hot Encoded)

In [129]:
```
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_train2, y_train2)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",accuracy_score(y_test2, grid.predict(x_test2)))
```

```
0.9942112879884226
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Prediction Accuracy:  0.9942196531791907
```

In [130]:
```
clf_svm_linear = grid.fit(x_train2,y_train2)
y_predict2 = clf_svm_linear.predict(x_test2)
confusion_matrix(y_test2,y_predict2)
```

Out[130]:
```
array([[ 76,   1,   0,   0],
       [  0,  14,   0,   0],
       [  1,   0, 241,   0],
       [  0,   0,   0,  13]], dtype=int64)
```

In [131]:
```
precision_recall_fscore_support(y_test2, y_predict2)
```

Out[131]:
```
(array([0.98701299, 0.93333333, 1.        , 1.        ]),
 array([0.98701299, 1.        , 0.99586777, 1.        ]),
 array([0.98701299, 0.96551724, 0.99792961, 1.        ]),
 array([ 77,  14, 242,  13], dtype=int64))
```

Model Selection

For this analysis, SVM with One Hot Encoding (the Categorical version) gave the best accuracy. The confusion matrix parameters all perform above the other models. The advantage of categorical is that we don't know the difference between the variables, ie: unacceptable vs good isn't a fixed distance of 2. We don't know what the distance between the two of them is. On the other had, a theoretical predicted value of say 2.3 has little to no meaning. Leaving it as an ordinal variable would be a little more interpetable.