0.0.1 Import Libraries:

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from sklearn import metrics
```

0.0.2 Read input data:

```
In [2]: dataset = pd.read_csv("wisconsin.csv", delimiter=",")
```

Using `dataset`, the wisconsin.csv data read by `pandas`, declare variables X as the feature matrix (data of `dataset`) and y as the response vector (target).

Remove the column containing the target name since it doesn't contain numeric values.

Also remove the column that contains the row number. `axis=1` means we are removing columns instead of rows.

Function takes in a pandas array and column numbers and returns a numpy array without the stated columns

```
In [3]: def removeColumns(pandasArray, *column):
            return pandasArray.drop(pandasArray.columns[[column]], axis=1).values
```

Create new matrix of features `X` by removing columns 1 and 2:

```
In [4]: X = removeColumns(dataset,0,1)
```

Now we create the response vector `y`, then use `LabelEncoder` to encode it as 0 and 1:

```
In [5]: # Select all rows of column 1 from my_data
        y = dataset.iloc[:, 1].values

        # Encoding the Dependent Variable to set malignant to 1 and benign to 0
        from sklearn.preprocessing import LabelEncoder
        labelencoder_y = LabelEncoder()
        y = labelencoder_y.fit_transform(y)
```

0.0.3 Recursive Feature Elimination

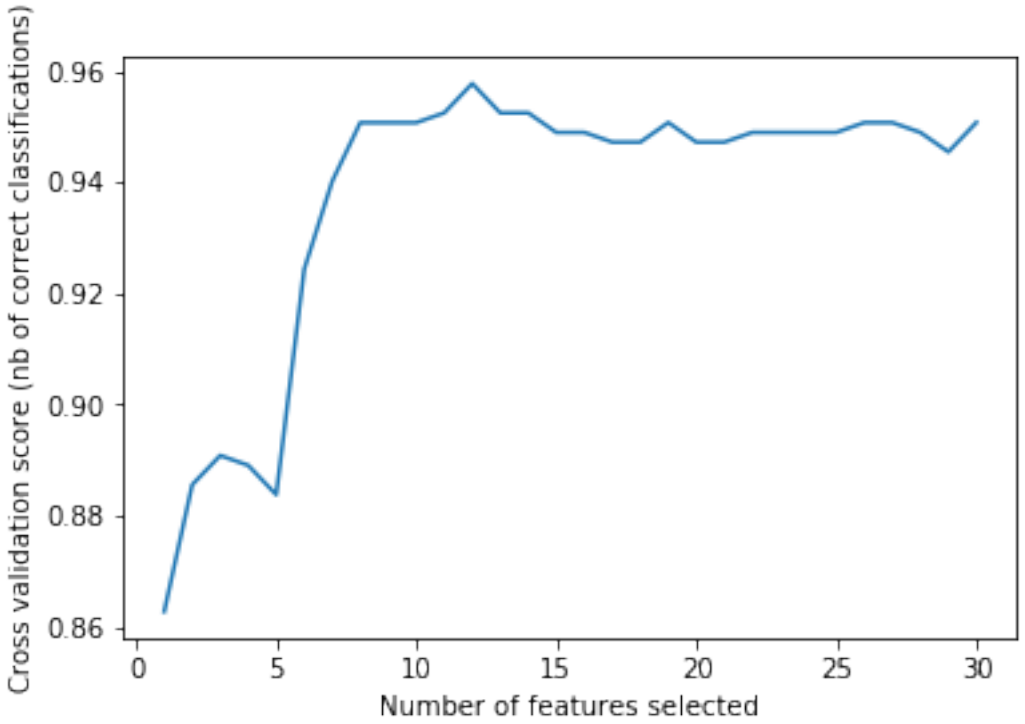
We need to see if we can eliminate some of the many features in the dataset, as some may not add to our predictive ability and only serve to create noise in the data.

We will use the Support Vector Classifier (SVC) as our estimator, which will be used to perform Recursive Factor Elimination (RFE) on our dataset:

```
In [6]: from sklearn.model_selection import StratifiedKFold
        from sklearn.feature_selection import RFECV
        from sklearn.svm import SVC

        # Create the RFE object and compute a cross-validated score
        svc = SVC(kernel="linear")
```

```
        # The "accuracy" scoring is proportional to the number of correct classifications
        rfecv = RFECV(estimator=svc, step=1, cv=StratifiedKFold(3),
                      scoring='accuracy')
        rfecv.fit(X, y)

        print("Optimal number of features : %d" % rfecv.n_features_)
        print(rfecv.ranking_)
        print(rfecv.grid_scores_)

        # Plot number of features vs. cross-validation scores
        plt.figure(1)
        plt.xlabel("Number of features selected")
        plt.ylabel("Cross validation score (nb of correct classifications)")
        plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
        plt.show()
```

```
Optimal number of features : 12
[ 1 10  5 19  2  1  1  1  1 16 14  1  3 11 12  8  7  9 17 15  1  4 13 18  1
  1  1  1  1  6]
[ 0.86271234  0.88552864  0.89080108  0.88902813  0.8837464   0.92432934
  0.94016523  0.95070083  0.95070083  0.95070083  0.95245521  0.95774622
  0.95245521  0.95245521  0.94894644  0.94894644  0.94719205  0.94719205
  0.95071939  0.94719205  0.94719205  0.94894644  0.94894644  0.94894644
  0.94894644  0.95071011  0.95071011  0.94894644  0.94543767  0.95072867]
```



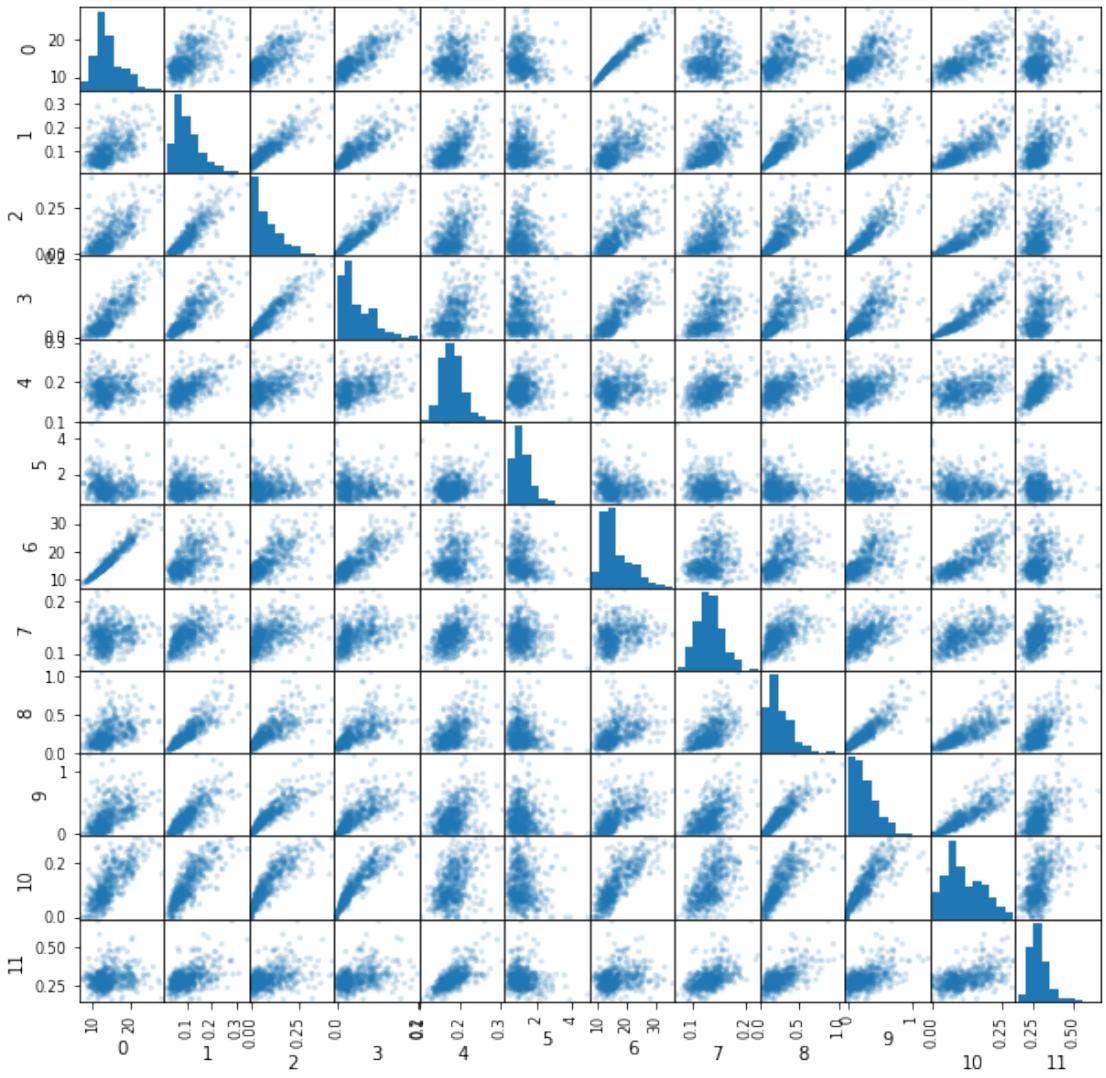
```
In [7]: # Apply the transformation to reduce X down to 12 features
        print(X.shape)
        X = rfecv.transform(X)
        print(X.shape)
```

```
(568, 30)
(568, 12)
```

0.1 Create Pairwise Plots of Reduced Feature Set

We will use the scattermatrix command from pandas to make pairwise plots of the features we selected previously using RFE.

```
In [8]: X_frame = pd.DataFrame(X)
        axes = pd.plotting.scatter_matrix(X_frame, alpha=0.2)
        fig = plt.gcf()
        fig.set_size_inches(10, 10)
```



0.2 Split the Data into Training and Test Sets

Perform train/test split we have to split the X and y into two different sets: The training and testing set.

```
In [9]: from sklearn.cross_validation import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4)
```

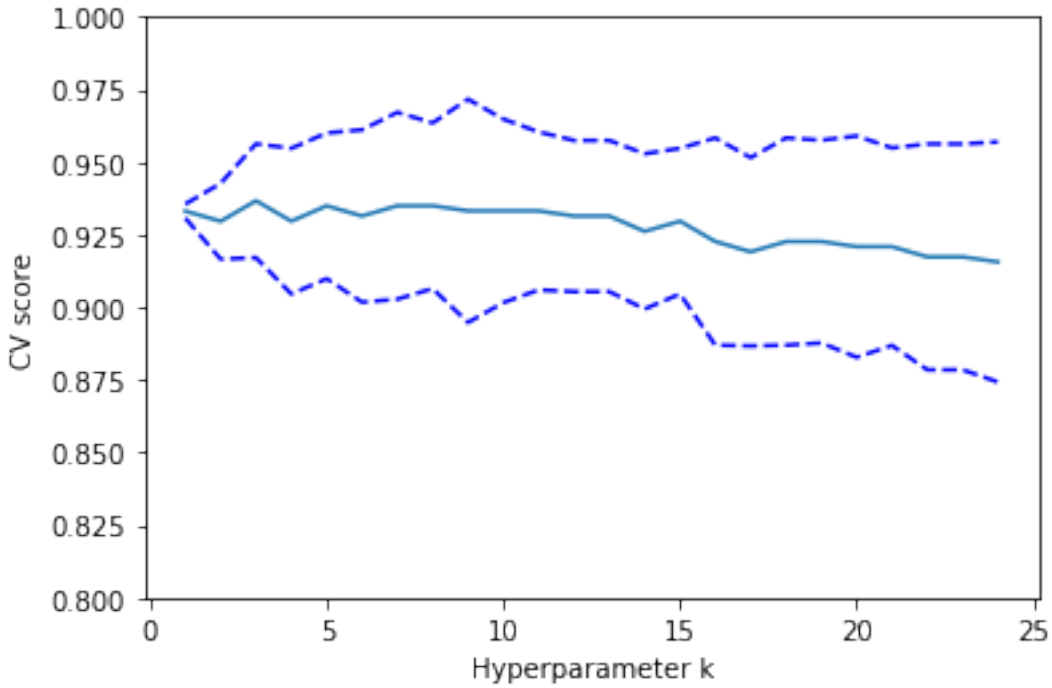
```
C:\Users\Ravenscliffe\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWar
  "This module will be removed in 0.20.", DeprecationWarning)
```

0.3 K-Nearest Neighbors

0.3.1 Select value of k based on cross-validation score

Here we will select the optimal value for k by fitting several models with different values for k and comparing their cross_val_score values:

In [10]:
```
# Create base k-NN classifier
from sklearn import neighbors
neigh = neighbors.KNeighborsClassifier()

# Create array of values of k from 1 to 20
k_s = np.arange(1,25)

# Iterate over different values of k, building models and storing the cross_val_score
from sklearn.model_selection import cross_val_score
scores = list()
scores_std = list()
for k in k_s:
    neigh.n_neighbors = k
    this_scores = cross_val_score(neigh, X, y, n_jobs=1)
    scores.append(np.mean(this_scores))
    scores_std.append(np.std(this_scores))

# Plot the cross-validation scores to select optimal value for k
plt.figure(2)
plt.plot(k_s, scores)
plt.plot(k_s, np.array(scores) + np.array(scores_std), 'b--')
plt.plot(k_s, np.array(scores) - np.array(scores_std), 'b--')
locs, labels = plt.yticks()
plt.ylabel('CV score')
plt.xlabel('Hyperparameter k')
plt.ylim(0.8, 1.0)
plt.show()
```



0.3.2 Fitting the model

As we can see from above, the cross-validation score seems to be maximized when k = 9. We will now create a KNeighborsClassifier where the number of neighbors = 3, then fit the data of each using the fit function with the X and y variables.

```
In [11]: from sklearn import neighbors
         neigh = neighbors.KNeighborsClassifier(n_neighbors=3)
         neigh.fit(X_train,y_train)

Out[11]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                    weights='uniform')
```

Create list for predicted targets:

```
In [12]: neigh_pred = list(neigh.predict(X_test))
```

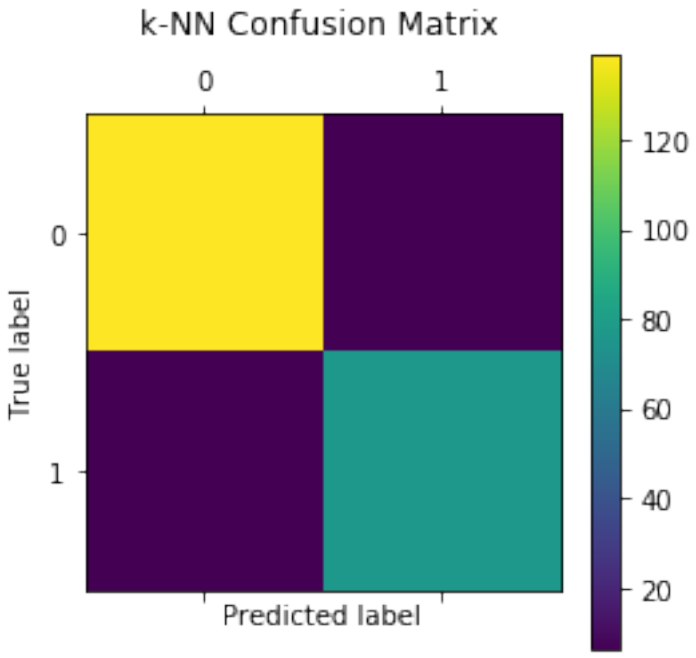
0.3.3 Confusion Matrix for k-NN

We can plot a confusion matrix to compute the accuracy of our k-NN classifier. This will allow us to visualize true positives and negatives, as well as false positives and negatives.

```
In [13]: from sklearn.metrics import confusion_matrix
         print (confusion_matrix(neigh_pred, y_test),)
```

```
        plt.matshow(confusion_matrix(neigh_pred, y_test),)
        plt.title('k-NN Confusion Matrix \n')
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
        print("Accuracy: ", metrics.accuracy_score(neigh_pred, y_test))
```

```
[[139   6]
 [  6  77]]
```




```
Accuracy:  0.947368421053
```

0.4 Naive Bayes

Now we'll fit a Naive Bayes classifier on the same dataset. We'll need to use StandardScaler to scale our data for this classifier to work properly.

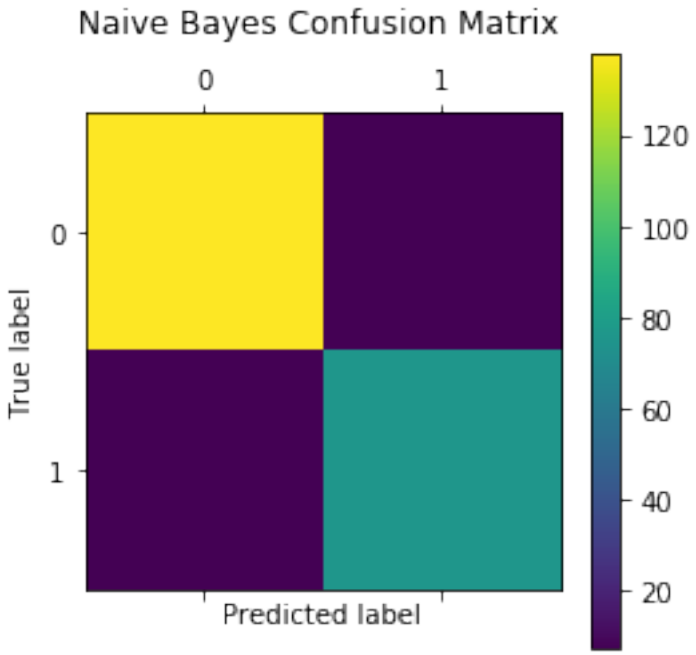
```
In [14]: # Feature Scaling
         from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_train_sc = sc.fit_transform(X_train)
         X_test_sc = sc.transform(X_test)
```

```
# Fit the Naive Bayes model to the data
from sklearn.naive_bayes import GaussianNB
naive = GaussianNB()
naive.fit(X_train_sc, y_train)

# Make predictions based on the test set
naive_pred = naive.predict(X_test_sc)
```

0.4.1 Confusion Matrix for Naive Bayes

```
In [15]: print (confusion_matrix(naive_pred, y_test),)
         plt.matshow(confusion_matrix(naive_pred, y_test),)
         plt.title('Naive Bayes Confusion Matrix \n')
         plt.colorbar()
         plt.ylabel('True label')
         plt.xlabel('Predicted label')
         plt.show()
         print("Accuracy: ", metrics.accuracy_score(naive_pred, y_test))
```

```
[[138   7]
 [  7  76]]
```

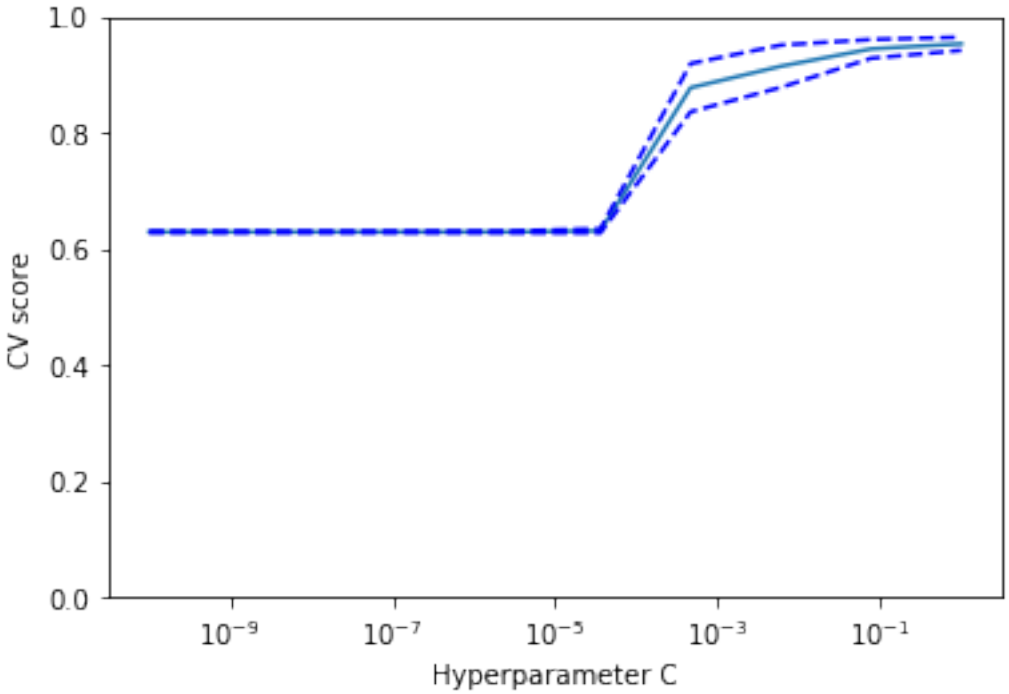



```
Accuracy:  0.938596491228
```

0.5 Support Vector Machine

0.5.1 Select value for C hyperparameter using cross-validation

In [16]:
```
# Create array of values of C on the log scale, from 10^-10 all the way to 10^0 (C = 1)
C_s = np.logspace(-10, 0, 10)

# Iterate over different values of C, building models and storing the cross_val_score
from sklearn.model_selection import cross_val_score
scores = list()
scores_std = list()
for C_param in C_s:
    svmc = SVC(C=C_param, kernel='linear')
    this_scores = cross_val_score(svmc, X, y, n_jobs=1)
    scores.append(np.mean(this_scores))
    scores_std.append(np.std(this_scores))

# Plot the cross-validation scores to select optimal value for C
plt.figure(3)
plt.semilogx(C_s, scores)
plt.semilogx(C_s, np.array(scores) + np.array(scores_std), 'b--')
plt.semilogx(C_s, np.array(scores) - np.array(scores_std), 'b--')
locs, labels = plt.yticks()
plt.ylabel('CV score')
plt.xlabel('Hyperparameter C')
plt.ylim(0, 1.0)
plt.show()
```



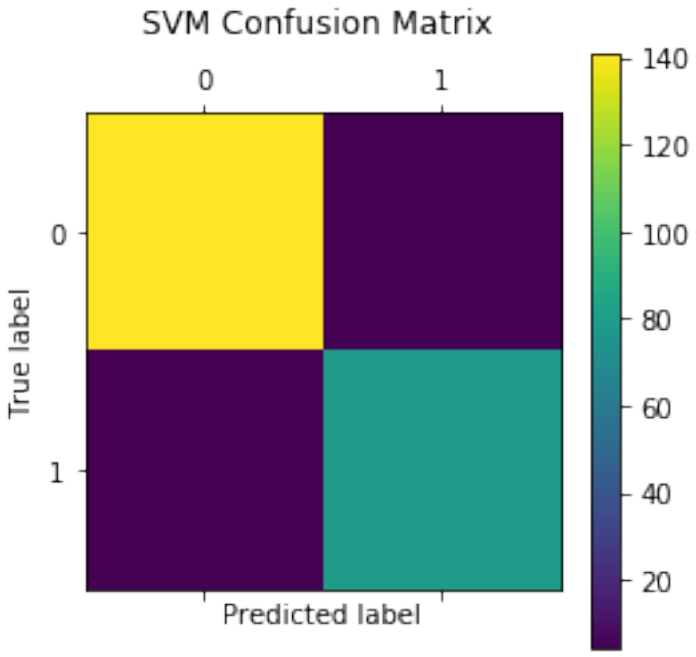
As can be seen from the plot, it appears that for this dataset the optimal parameter for C is at the default of C = 1. We will now fit a Support Vector Machine classifier to the data using a Gaussian kernel.

```
In [17]: # Fit the SVM model to the data
         svmc = SVC(C=1, kernel='linear', probability = True)
         svmc.fit(X_train_sc, y_train)

         # Make predictions based on the test set
         svm_pred = svmc.predict(X_test_sc)
```

0.5.2 Confusion Matrix for SVM

```
In [18]: print(confusion_matrix(svm_pred, y_test),)
         plt.matshow(confusion_matrix(svm_pred, y_test),)
         plt.title('SVM Confusion Matrix \n')
         plt.colorbar()
         plt.ylabel('True label')
         plt.xlabel('Predicted label')
         plt.show()
         print("Accuracy: ", metrics.accuracy_score(svm_pred, y_test))
```

```
[[141   4]
 [  4  79]]
```

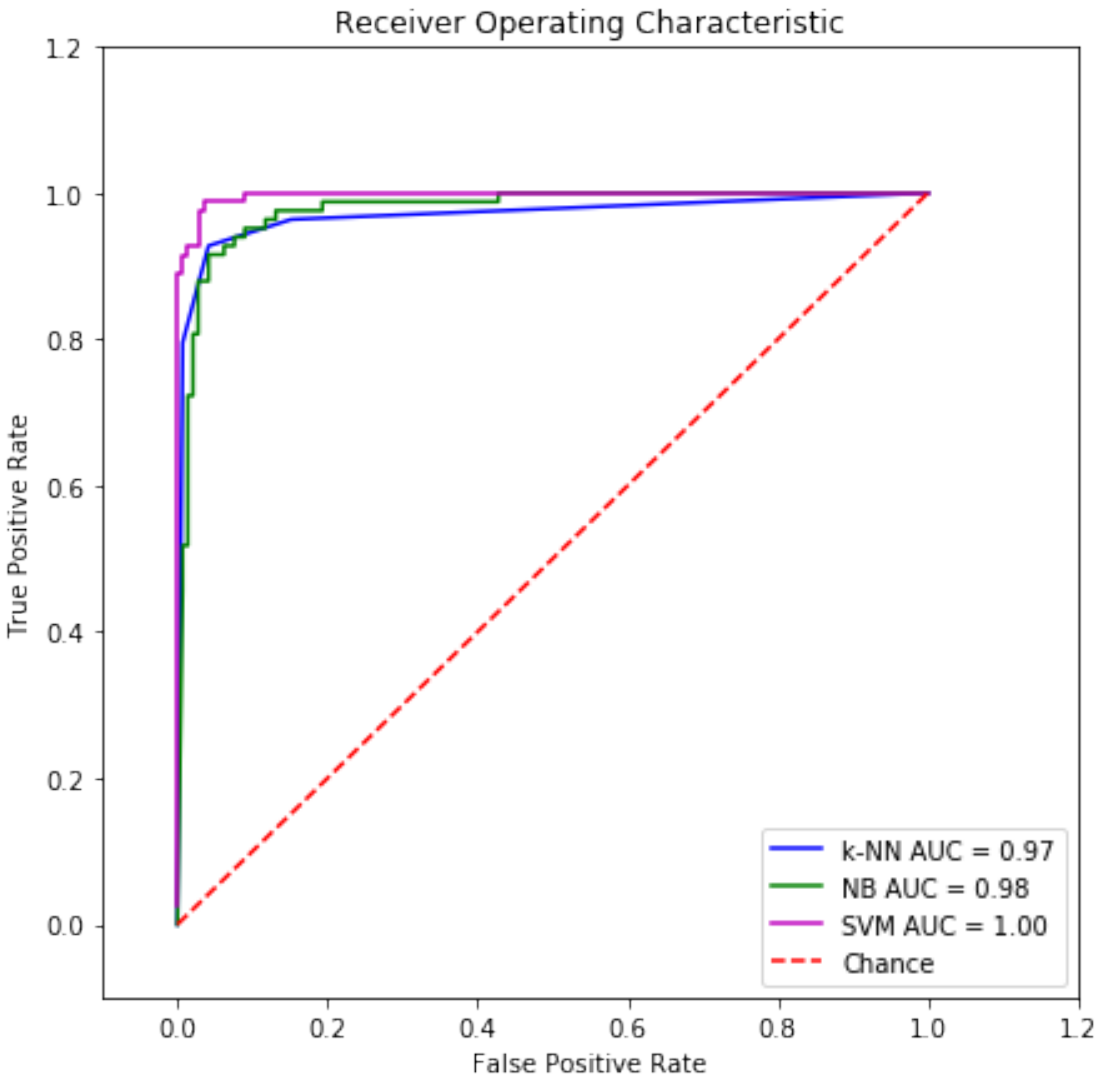


```
Accuracy:  0.964912280702
```

0.6 Classifier Comparison

We can use a ROC curve to compare our classifiers:

```
In [19]: from sklearn.metrics import roc_curve, auc
         # Get scores for k-NN
         roc_pred_knn = neigh.predict_proba(X_test)[:,1]
         fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, roc_pred_knn)
         roc_auc_knn = auc(fpr_knn, tpr_knn)

         # Get scores for Naive Bayes
         roc_pred_nb = naive.predict_proba(X_test_sc)[:,1]
         fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, roc_pred_nb)
         roc_auc_nb = auc(fpr_nb, tpr_nb)

         # Get scores for SVM
         roc_pred_svm = svmc.predict_proba(X_test_sc)[:,1]
         fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test, roc_pred_svm)
         roc_auc_svm = auc(fpr_svm, tpr_svm)

         plt.figure(4, figsize=(7,7))
         plt.title('Receiver Operating Characteristic')
         plt.plot(fpr_knn, tpr_knn, 'b', label='k-NN AUC = %0.2f'% roc_auc_knn)
         plt.plot(fpr_nb, tpr_nb, 'g', label='NB AUC = %0.2f'% roc_auc_nb)
         plt.plot(fpr_svm, tpr_svm, 'm', label='SVM AUC = %0.2f'% roc_auc_svm)
         plt.plot([0,1],[0,1],'r--', label='Chance')
         plt.legend(loc='lower right')
         plt.xlim([-0.1,1.2])
         plt.ylim([-0.1,1.2])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```



0.7 Dimensionality Reduction

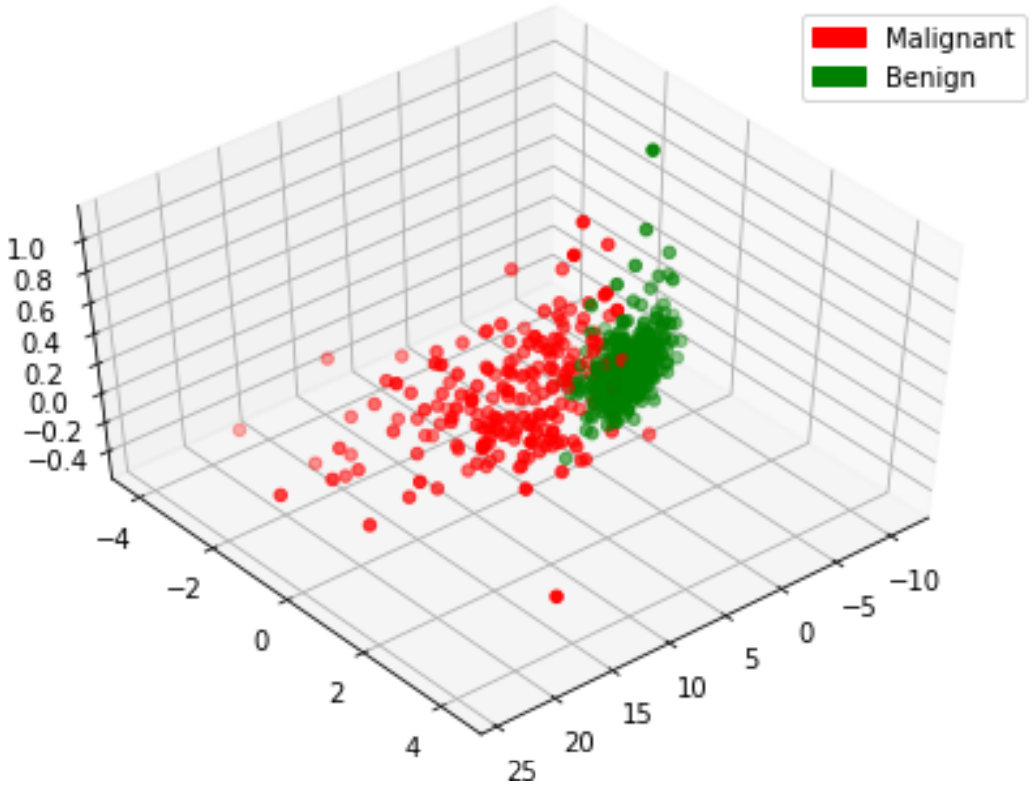
Reduce X(processed_data) and y(target) into 3 dimensions and call it X_new.

```
In [20]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2
         X_new = SelectKBest(chi2, k=3).fit_transform(X, y)
         print ("Shape before dimensionality reduction  ", X.shape)
         print ("Shape after dimensionality reduction  ", X_new.shape)
```

```
Shape before dimensionality reduction   (568, 12)
Shape after dimensionality reduction   (568, 3)
```

Plot reduced array as a 3D scatterplot.

```
In [21]: # Create label colour dictionary for 0s and 1s
         label_colours = {0 : 'green', 1 : 'red'}
         colour_vector = [label_colours[response] for response in y]

         from sklearn import decomposition
         from mpl_toolkits.mplot3d import Axes3D
         pca = decomposition.PCA(n_components=3)
         pca.fit(X_new)
         PCA_X = pca.transform(X_new)
         fig = plt.figure(5)
         ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=50, azim=50)
         ax.scatter(PCA_X[:, 0], PCA_X[:, 1],PCA_X[:,2],c=colour_vector, cmap=plt.cm.spring, dep

         # Add a legend
         import matplotlib.patches as mp
         red_patch = mp.Patch(color = 'red', label = 'Malignant')
         green_patch = mp.Patch(color = 'green', label = 'Benign')
         handles = [red_patch, green_patch]
         plt.legend(handles=handles)

Out[21]: <matplotlib.legend.Legend at 0x10c29668>
```



Reduce X(processed_data) and y(target) into 2 dimensions and call it X_new2.

```
In [22]: X_new2 = SelectKBest(chi2, k=2).fit_transform(X, y)
         print ("Shape before dimensionality reduction  ", X.shape)
         print ("Shape after dimensionality reduction  ", X_new2.shape)
```

```
Shape before dimensionality reduction   (568, 12)
Shape after dimensionality reduction   (568, 2)
```

Plot reduced array as a 2D scatterplot.

```
In [23]: pca = decomposition.PCA(n_components=2)

         pca.fit(X_new2)
         PCA_X = pca.transform(X_new2)

         plt.scatter(PCA_X[:, 0], PCA_X[:, 1],cmap=plt.cm.spring,c=colour_vector,alpha=0.4)

         # Add a legend
         plt.legend(handles=handles)
         plt.show()
```