COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Week 11
Machine Learning - Basic Regression and Classification

Applied Data Science

Columbia University - Columbia Engineering

Course Agenda

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

- Week 1: Python Basics: How to Translate Procedures into Codes
- Week 2: Intermediate Python — Data structures for Your Analysis
- Week 3: Relational Databases — Where Big Data is Typically Stored
- Week 4: SQL — Ubiquitous Database Format/Language
- Week 5: Statistical Distributions — The Shape of Data
- Week 6: Sampling — When You Can't or Won't Have ALL the Data
- Week 7:Hypothesis Testing — Answering Questions about Your Data
- Week 8: Data Analysis and Visualization — Using Python's NumPy for Analysis
- Week 9: Data analysis and visualization — Using Python's Pandas for Data Wrangling
- Week 10: Text Mining — Automatic Understanding of Text
- **Week 11: Machine learning — Basic Regression and Classification**
- Week 12: Machine learning — Decision Trees and Clustering

Machine learning using Regression

Read the data

Generate a few summary statistics

Data set 1: Rocks vs. Mines

- Independent variables: sonar soundings at different frequencies
- Dependent variable (target): Rock or Mine

Out[1]:

	0	1	2	3	4	5	6
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570	0.121747
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900

In [1]:
```
import pandas as pd
from pandas import DataFrame
url="https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-benc
df = pd.read_csv(url,header=None)
df.describe()
```

Examine the distribution of the data in column 4

- Quartile 1: from .0067 to .03805
- Quartile 2: from .03805 to .0625
- Quartile 3: from .0625 to .100275
- Quartile 4: from .100275 to .401

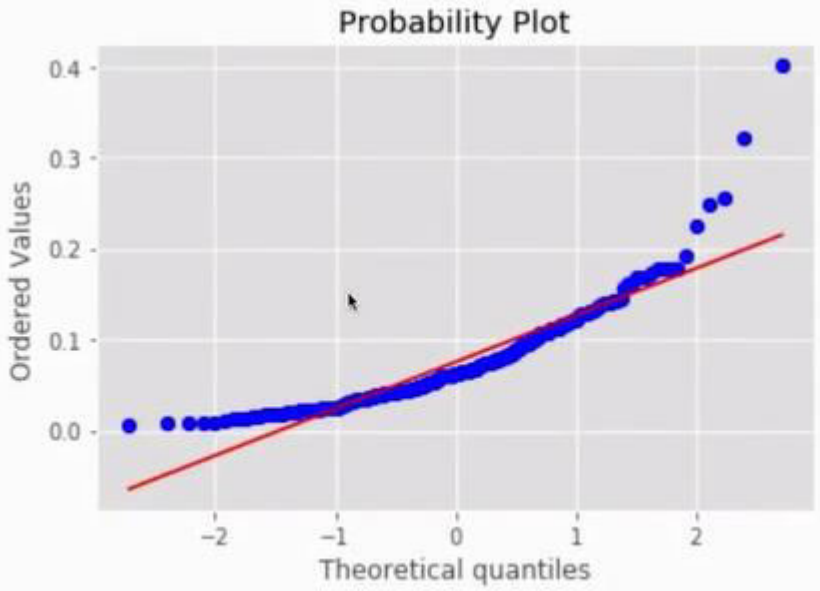
Quartile 4 is much larger than the other quartiles. This raises the possibility of outliers

A Quantile - Quantile (qq) plot can help identify outliers

- y-axis contains values
- x-axis is the cumulative normal density function plotted as a straight line (-3 to +3)
- y-axis is the values ordered from lowest to highest
- the closer the curve is to the line, the more it reflects a normal distribution

```
In [3]: import numpy as np
        import pylab
        import scipy.stats as stats
        import matplotlib
        import matplotlib.pyplot as plt
        matplotlib.style.use('ggplot')
        %matplotlib inline

        stats.probplot(df[4], dist="norm", plot=pylab)
        pylab.show()
```



Examine dependent variable

```
In [4]: df[60].unique()
```

```
Out[4]: array(['R', 'M'], dtype=object)
```

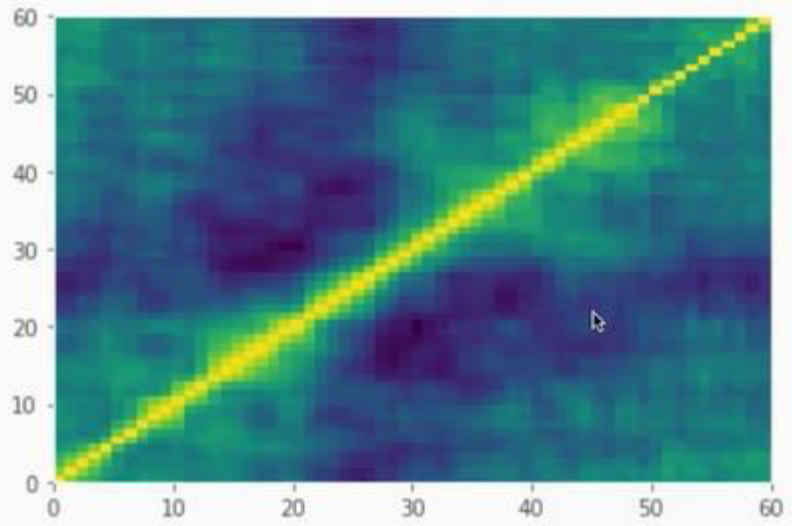
Examine correlations

```
In [5]: df.corr()
```

Out[5]:

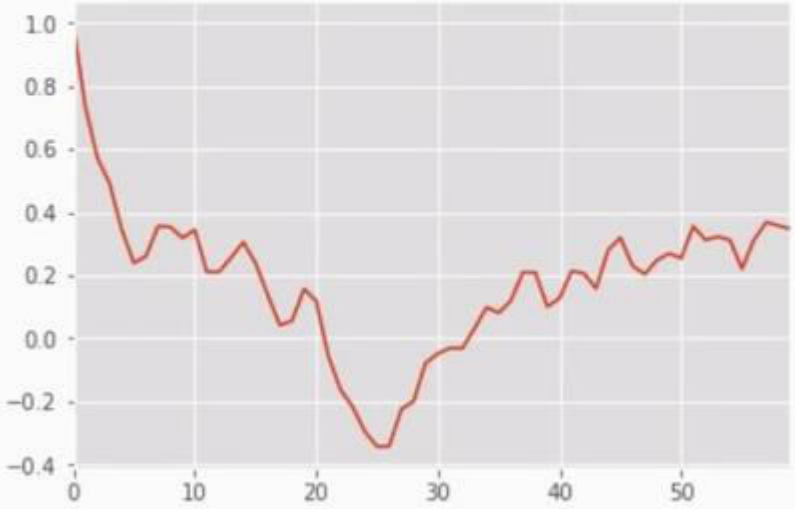
	0	1	2	3	4	5	6
0	1.000000	0.735896	0.571537	0.491438	0.344797	0.238921	0.260815
1	0.735896	1.000000	0.779916	0.606684	0.419669	0.332329	0.279040
2	0.571537	0.779916	1.000000	0.781786	0.546141	0.346275	0.190434
3	0.491438	0.606684	0.781786	1.000000	0.726943	0.352805	0.246440
4	0.344797	0.419669	0.546141	0.726943	1.000000	0.597053	0.335422
5	0.238921	0.332329	0.346275	0.352805	0.597053	1.000000	0.702889

```
In [6]: import matplotlib.pyplot as plot
        plot.pcolor(df.corr())
        plot.show()
```




```
In [7]: df.corr()[0].plot()

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1169575f8>
```



Highly correlated items = not good!

Low correlated items = good ¶

Correlations with target (dv) = good (high predictive power)

Data Set: Wine Data

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

- Independent variables: Wine composition (alcohol content, sulphites, acidity, etc.)
- Dependent variable (target): Taste score (average of a panel of 3 wine tasters)

```
In [8]: url = "http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/wine
        import pandas as pd
        from pandas import DataFrame
        w_df = pd.read_csv(url,header=0,sep=';')
        w_df.describe()
```

Out[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000

```
In [9]: w_df['volatile acidity']
```

Out[9]:

0	0.700
1	0.880
2	0.760
3	0.280
4	0.700
5	0.660
6	0.600
7	0.650
8	0.580
9	0.500
10	0.580
11	0.500
12	0.615
13	0.610
14	0.620
15	0.620
16	0.280
17	0.560
18	0.590
19	0.320
20	0.220
21	0.390

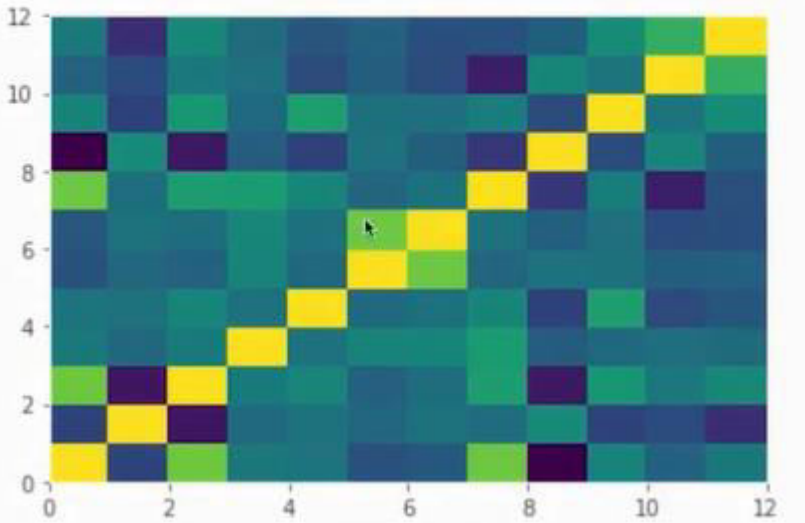
Data Set: Wine Data


```
In [10]: w_df.corr()
```

Out[10]:

	fixed acidity	volatile acidity	citric acid	residual sugar
fixed acidity	1.000000	-0.256131	0.671703	0.114777
volatile acidity	-0.256131	1.000000	-0.552496	0.001918

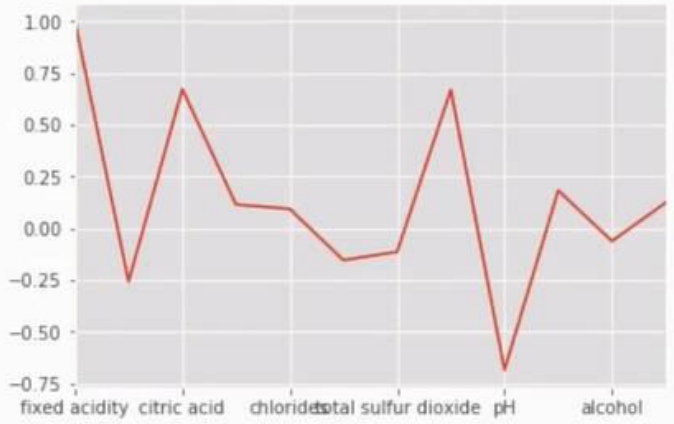
```
In [11]: import matplotlib.pyplot as plot
plot.pcolor(w_df.corr())
plot.show()
```



Examining the Correlation of One Variable with Other

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

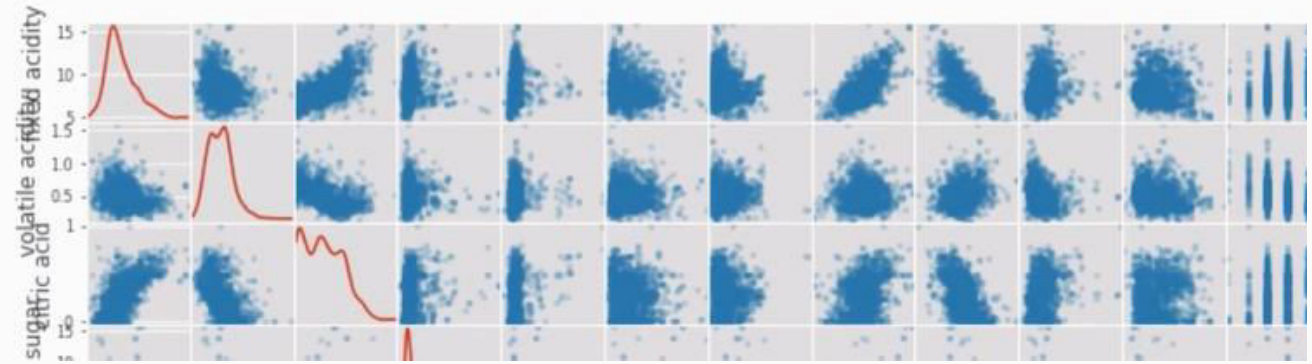
```
In [12]: w_df.corr()['fixed acidity'].plot()
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x116a9c390>
```



Pandas scatter matrix function helps visualize the relationship between features

Use with care though, because it is processor intensive

```
In [13]: from pandas.tools.plotting import scatter_matrix
         p=scatter_matrix(w_df, alpha=0.2, figsize=(12, 12), diagonal='kde')
```

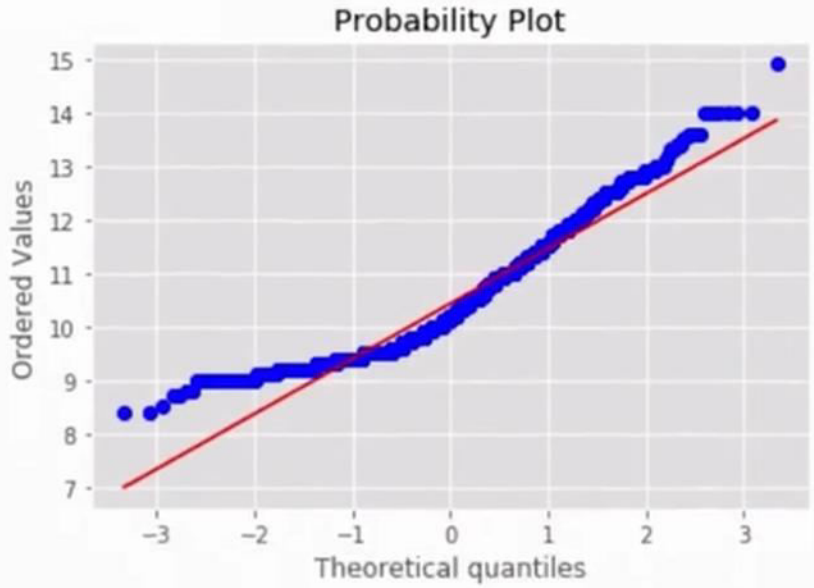


Examining the Correlation of One Variable with Other

And we can examine quintile plots as we did with the rocks and mines data

In [14]:
```
import numpy as np
import pylab
import scipy.stats as stats
%matplotlib inline

stats.probplot(w_df['alcohol'], dist="norm", plot=pylab)
pylab.show()
```



Training a Classifier on Rock Vs Mine

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [ ]:
```
import numpy
import random
from sklearn import datasets, linear_model
from sklearn.metrics import roc_curve, auc
import pylab as pl
```

In [ ]:
```
import pandas as pd
from pandas import DataFrame
url="https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectio
df = pd.read_csv(url,header=None)
df.describe()
```

Out[16]:

	0	1	2	3	4	5
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300

Training a Classifier on Rock Vs Mine

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Convert labels R and M to 0 and 1

In [17]:
```
df[60]=np.where(df[60]=='R',0,1)
```

Divide the dataset into training and test samples

Separate out the x and y variable frames for the train and test samples

In [18]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size = 0.3)
x_train = train.iloc[0:,0:60]
y_train = train[60]
x_test = test.iloc[0:,0:60]
y_test = test[60]
y_train
```

Out[18]:
```
6      0
193    1
179    1
139    1
94     0
24     0
153    1
```

Build the model and fit the training data

In [ ]:
```
model = linear_model.LinearRegression()
model.fit(x_train,y_train)
```

Interpreting Categorical Prediction Results

Precision

Recall

True Positive Rate

False Positive Rate

Precision recall curve

ROC curve

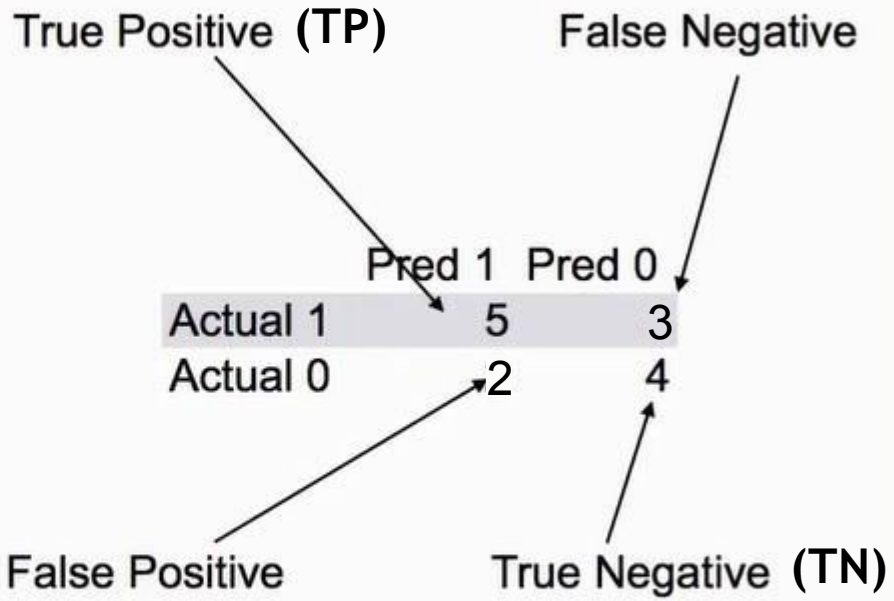
F-Score

Area under PR curve

Area under ROC curve

Classification Metrics: Confusion Matrix

	Actual	Predict
TP	1	1
FN	1	0
TP	1	1
TP	1	1
TP	1	1
FN	1	0
TP	1	1
TN	0	0
TN	0	0
FP	0	1
TN	0	0
TN	0	0
FP	0	1
FP	0	1



Classification Metrics

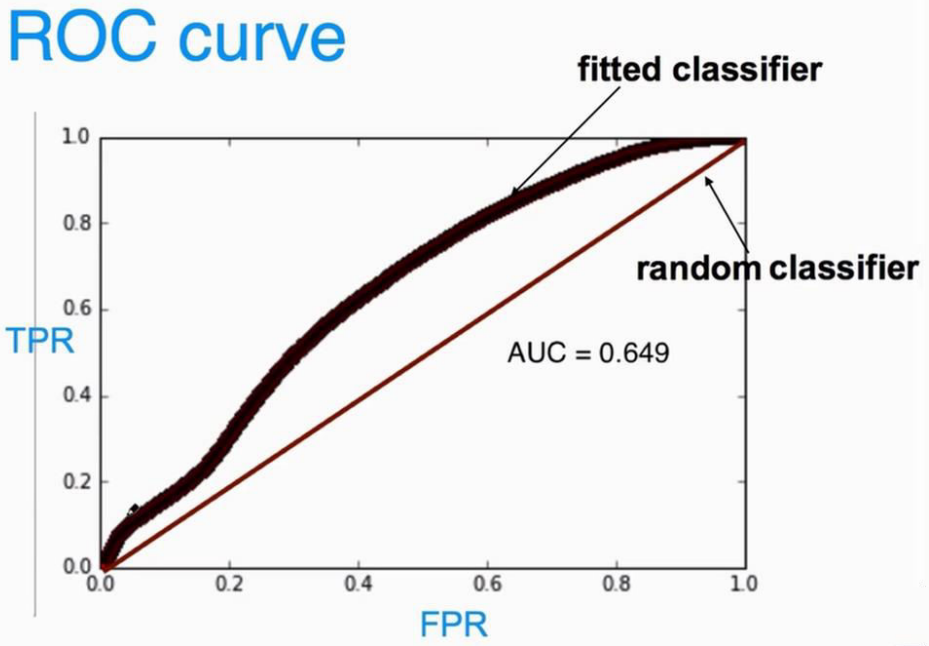
1. Precision: what proportion of the cases that the model said were 1 were actually 1
   - precision = TP/(TP+FP)
   - 5/(5+2) = 71.4%
2. Recall: what proportion of the cases that were actually 1 were identified as 1 by the model
   - recall = TP/(TP + FN)
   - 5/(5+3) = 62.5%
3. F-Score: measures accuracy by balancing precision and recall
   - fscore = 2 * (P * R)/(P + R)
   - 67%

1. True Positive Rate (TPR): what proportion of the cases that were actually 1 were identified as 1(tpr = recall)
   - tpr = TP/(TP+FN)
   - 5/(5+3) = 62.5%
2. False Positive Rate (FPR): what proportion of the cases that the model said were 1 were actually zero
   - fpr = FP/(TN + FP)
   - 2/(4+2) =33.3%

Fixing and Determining the Threshold

Fixing the Threshold

1. ROC Curve: plots the True Positive Rate against the False Positive Rate as the threshold varies from 0 to 1
2. Precision-Recall Curve: Plots precision against recall as the threshold varies from 0 to 1



Determining the Threshold

1. AUROC: The area under the ROC curve. AUROC is used to determine if the classifier is doing better than a random classifier. It can also help pick a threshold.
2. AUPRC: The area under the PRC curve.

Generate Predictions in Sample Error

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

```
In [22]: training_predictions = model.predict(x_train)
         print(np.mean((training_predictions - y_train) ** 2))

         0.08541463252093508
```

```
In [23]: print('Train R-Square:',model.score(x_train,y_train))
         print('Test R-Square:',model.score(x_test,y_test))

         Train R-Square: 0.657934733571
         Test R-Square: 0.0425249928917
```

```
In [24]: training_predictions

Out[24]: array([-0.10188075,  0.38338
                0.06581291,  0.51937
                0.65898004, -0.36202
                0.8790505 ,  0.22359
                0.44177435,  0.66517
```

```
In [25]: y_train

Out[25]: 6      0
         193    1
         179    1
         139    1
         94     0
         24     0
```

These are horrible!

But do we really care?

- Focus on the problem
- Do we need to recognize both rocks as well as mines correctly?
- How do we interpret the predicted y-values

Generate Predictions in Sample Error


```
In [ ]: print(max(training_predictions),min(training_predictions),np.mean(training_predictions))
```

We want to predict categories: Rocks or Mines

But we're actually getting a continuous value

Not the same things. So R-Square probably doesn't mean a whole lot

We need to convert the continuous values into categorical 1s and 0s. We can do this by fixing a threshold value between 0 and 1
Values greater than the threshold are 1 (Mines). Values less than or equal to the threshold are 0 (Rocks)

Confusion Matrix

- Reports the proportion of

1. **true positive**: predicts mine and is a mine
2. **false positive**: predicts mine and is not a mine
3. **true negative**: predicts not mine and is not a mine
4. **false negative**:Predicts not mine but turns out to be a mine (BOOM!)

In [26]:
```
def confusion_matrix(predicted, actual, threshold):
    if len(predicted) != len(actual): return -1
    tp = 0.0
    fp = 0.0
    tn = 0.0
    fn = 0.0
    for i in range(len(actual)):
        if actual[i] > 0.5: #labels that are 1.0  (positive examples)
            if predicted[i] > threshold:
                tp += 1.0 #correctly predicted positive
            else:
                fn += 1.0 #incorrectly predicted negative
        else:                #labels that are 0.0 (negative examples)
            if predicted[i] < threshold:
                tn += 1.0 #correctly predicted negative
            else:
                fp += 1.0 #incorrectly predicted positive
    rtn = [tp, fn, fp, tn]

    return rtn
```

In [28]:
```
testing_predictions = model.predict(x_test)
```

In [29]:
```
testing_predictions = model.predict(x_test)
confusion_matrix(testing_predictions,np.array(y_test),0.5)
```

Out[29]:
```
[27.0, 9.0, 5.0, 22.0]
```

Misclassification rate = (fp + fn)/number of cases

In [30]:
```
cm = confusion_matrix(testing_predictions,np.array(y_test),0.5)
misclassification_rate = (cm[1] + cm[2])/len(y_test)
misclassification_rate
```

Out[30]:
```
0.2222222222222222
```

Precision and Recall

In [31]:
```
[tp, fn, fp, tn] = confusion_matrix(testing_predictions,np.array(y_test),0.5)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f_score = 2 * (precision * recall)/(precision + recall)
print(precision,recall,f_score)
```

```
0.84375 0.75 0.7941176470588235
```

Confusion Matrix

Confusion matrix (and hence precision, recall etc.) depend on the Selected threshold

As the threshold changes, we will need to tradeoff precision and recall

In [32]:
```
[tp, fn, fp, tn] = confusion_matrix(testing_predictions,np.array(y_test),0.9)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f_score = 2 * (precision * recall)/(precision + recall)
print(precision,recall,f_score)
```

```
0.8823529411764706 0.4166666666666667 0.5660377358490566
```

ROC: Receiver Order Characteristic

- An ROC curve shows the performance of a binary classifier as the threshold varies.
- Computes two series:

1. False positive rate (FPR) Fall out/false alarm = False Positives/(True Negatives + False Positives)
    - Or, what proportion of rocks are identified as mines
2. True Positive rate (TPR) Sensitivity/recall = True Positives/(True Positives + False Negatives)
    - Or, what proportion of actual mines are identified as mines

- **true positive**: predicts mine and is a mine
- **false positive**: predicts mine and is not a mine
- **true negative**: predicts not mine and is not a mine
- **false negative**:Predicts not mine but turns out to be a mine (BOOM!)

ROC: Receiver Order Characteristic

Let's first plot the predictions against actuals

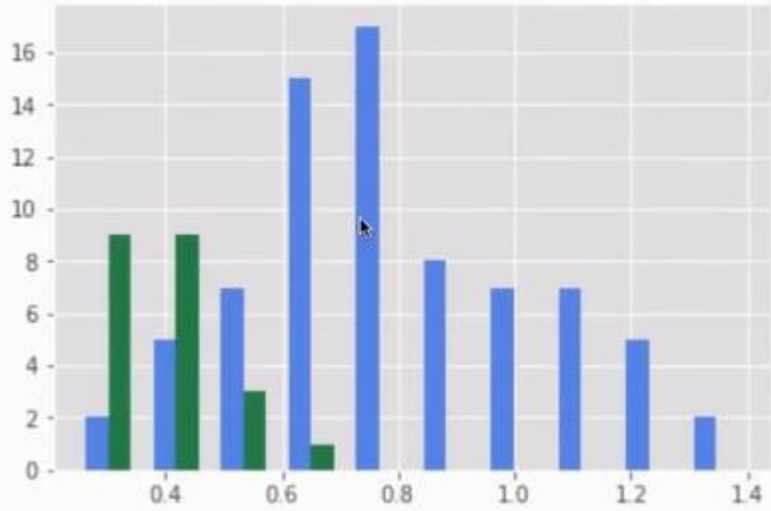
The goal is to see if our classifier has discriminated at all

In [ ]:
```
positives = list()
negatives = list()
actual = np.array(y_train)
for i in range(len(y_train)):

    if actual[i]:
        positives.append(training_predictions[i])
    else:
        negatives.append(training_predictions[i])
```

In [34]:
```
df_p = pd.DataFrame(positives)
df_n = pd.DataFrame(negatives)
fig, ax = plt.subplots()
a_heights, a_bins = np.histogram(df_p)
b_heights, b_bins = np.histogram(df_n, bins=a_bins)
width = (a_bins[1] - a_bins[0])/3
ax.bar(a_bins[:-1], a_heights, width=width, facecolor='cornflowerblue')
ax.bar(b_bins[:-1]+width, b_heights, width=width, facecolor='seagreen')
```

Out[34]: <Container object of 10 artists>



Drawing ROC

sklearn has a function roc_curve that does this for us

In [36]:
```
from sklearn.metrics import roc_curve, auc
```

In-sample ROC Curve

In [38]:
```
(fpr, tpr, thresholds) = roc_curve(y_train,training_predictions)
area = auc(fpr,tpr)
area
```

Out[38]:
```
0.98323809523809524
```

In [ ]:
```
(fpr, tpr, thresholds) = roc_curve(y_train,training_predictions)
area = auc(fpr,tpr)
pl.clf() #Clear the current figure
pl.plot(fpr,tpr,label="In-Sample ROC Curve with area = %1.2f"%area)

pl.plot([0, 1], [0, 1], 'k') #This plots the random (equal probability line)
pl.xlim([0.0, 1.0])
pl.ylim([0.0, 1.0])
pl.xlabel('False Positive Rate')
pl.ylabel('True Positive Rate')
pl.title('In sample ROC rocks versus mines')
pl.legend(loc="lower right")
pl.show()
```

In-sample ROC Curve

In [39]:
```
(fpr, tpr, thresholds) = roc_curve(y_train,training_predictions)
area = auc(fpr,tpr)
fpr,tpr,thresholds
```

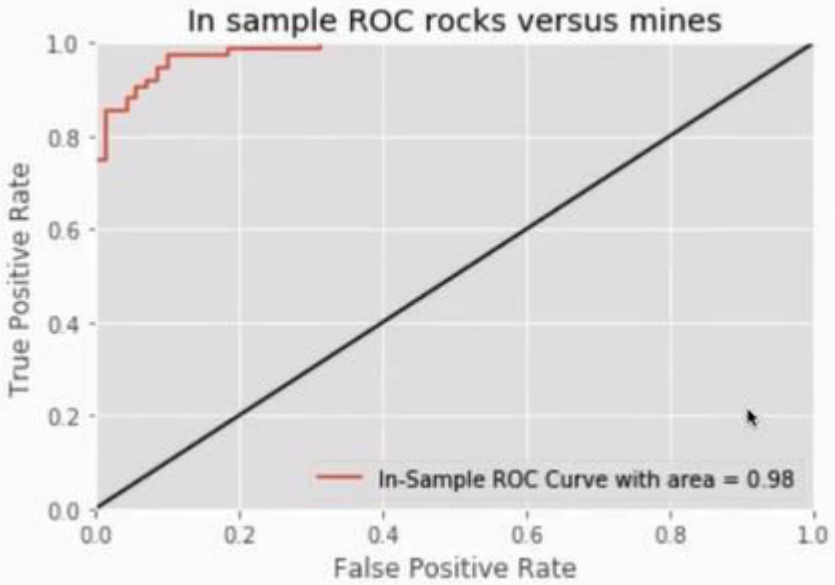
Out[39]:
```
(array([ 0.        ,  0.        ,  0.01428571,  0.01428571,  0.04285714,
         0.04285714,  0.05714286,  0.05714286,  0.07142857,  0.07142857,
         0.08571429,  0.08571429,  0.1       ,  0.1       ,  0.18571429,
         0.18571429,  0.31428571,  0.31428571,  1.        ]),
 array([ 0.01333333,  0.74666667,  0.74666667,  0.85333333,  0.85333333,
         0.88      ,  0.88      ,  0.90666667,  0.90666667,  0.92      ,
         0.92      ,  0.94666667,  0.94666667,  0.97333333,  0.97333333,
         0.98666667,  0.98666667,  1.        ,  1.        ]),
 array([ 1.44222918,  0.68043704,  0.67032538,  0.60883173,  0.57137311,
         0.56550712,  0.52840754,  0.51937525,  0.51039308,  0.5088578 ,
         0.48382492,  0.47656187,  0.47511402,  0.47086282,  0.40703878,
         0.38338798,  0.28354984,  0.28275932, -0.47183391]))
```

Drawing ROC


```
In [40]: (fpr, tpr, thresholds) = roc_curve(y_train,training_predictions)
         area = auc(fpr,tpr)
         pl.clf() #Clear the current figure
         pl.plot(fpr,tpr,label="In-Sample ROC Curve with area = %1.2f"%area)

         pl.plot([0, 1], [0, 1], 'k') #This plots the random (equal probability line)
         pl.xlim([0.0, 1.0])
         pl.ylim([0.0, 1.0])
         pl.xlabel('False Positive Rate')
         pl.ylabel('True Positive Rate')
         pl.title('In sample ROC rocks versus mines')
         pl.legend(loc="lower right")
         pl.show()
```

```
In [ ]: (fpr, tpr, thresholds)
```



Drawing ROC

So, what threshold should we actually use?

ROC curves and AUC give you a sense for how good your classifier is and how sensitive it is to changes in threshold

Too sensitive is not good

Example: Let's say

- Everything classified as a rock needs to be checked with a hand scanner at 200/scan
- Everything classified as a mine needs to be defused at 1000 if it is a real mine or 300 if it turns out to be a rock

In [ ]:
```
cm = confusion_matrix(testing_predictions,np.array(y_test),.1)
cost1 = 1000*cm[0] + 300 * cm[2] + 200 * cm[1] + 200 * cm[3]
cm = confusion_matrix(testing_predictions,np.array(y_test),.9)
cost2 = 1000*cm[0] + 300 * cm[2] + 200 * cm[1] + 200 * cm[3]

print(cost1,cost2)
```

```
41100.0 24800.0
```

Example: Let's say

- Everything classified as a rock will be assumed a rock and if wrong, will cost 5000 in injuries
- Everything classified as a mine will be left as is (no one will walk on it!)

In [ ]:
```
cm = confusion_matrix(testing_predictions,np.array(y_test),.1)
cost1 = 0*cm[0] + 0 * cm[2] + 5000 * cm[1] + 0 * cm[3]
cm = confusion_matrix(testing_predictions,np.array(y_test),.9)
cost2 = 0*cm[0] + 0 * cm[2] + 5000 * cm[1] + 0 * cm[3]
print(cost1,cost2)
```

```
10000.0 105000.0
```

www.emeritus.org