Data Science

Objectives

- Describe what supervised machine learning is
- Contrast the difference between **quantitative** and **categorical** varaibles
- Describe what classification is
- Demo supervised classification of Iris type data.

Supervised Learning ¶

- Take **known** input data
- Create a model of that data and **train** it based on **known** output data
- After that, predict outputs from previously unknown inputs.

Quantitative vs Qualititave variables

- Quantitative variables are continuous variable values. These are the petal and sepal lengths and widths in centimeters for this demo.
- Categorical variables seperate things into a controlled number of categories. These are the target Iris types in our data.
- Classification is used to take quantitative variables and place them into categories. For us, this figuring out what type of Iris an observation is.

Categorical variables on Wikipedia (https://en.wikipedia.org/wiki/Categorical_variable)

Modules to import

- NumPy: An exceptional numeric computation system. (we have already seen this)
- matplotlib: A system to make data visualizations.
- Pandas: An spreadsheet-like way of manipulating data in Python.
- sklearn: A machine learning system.

Exploration and Training

- When we first encounter a dataset it is useful to explore it to find out about its properties. So we will explore the data with matplotlib.
- Then we will train our classification model. In our case the is logistic regression.
- Finally we will test our model, and discuss ways it could be made better.

Preparing the Iris data

- Import Pandas and NumPy
- Get the data science logistic regression code

- Get the iris data set
- Prepare to do visualizations.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
%matplotlib inline
iris_data = load_iris()
```

Now explore what is there.

A long-winded description of the dataset.

In [2]:
```
print iris_data.DESCR
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris (http://archive.ics.uci.edu/m
l/datasets/Iris)

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field
 and
is referenced frequently to this day.  (See Duda & Hart, for example.)  T
he
data set contains 3 classes of 50 instances each, where each class refers
 to a
type of iris plant.  One class is linearly separable from the other 2; th
e
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions
 to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analy
sis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Expos
ed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transa
ctions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS
 II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

What are the types of irises in the data set?

In [3]:
```
print iris_data.target_names
```

```
['setosa' 'versicolor' 'virginica']
```

What are the features of the irises?

In [4]:
```
print iris_data.data
```

```
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]
 [ 5.4  3.7  1.5  0.2]
 [ 4.8  3.4  1.6  0.2]
 [ 4.8  3.   1.4  0.1]
 [ 4.3  3.   1.1  0.1]
 [ 5.8  4.   1.2  0.2]
 [ 5.7  4.4  1.5  0.4]
 [ 5.4  3.9  1.3  0.4]
 [ 5.1  3.5  1.4  0.3]
 [ 5.7  3.8  1.7  0.3]
```

What are the known types of Irises corresponding to each row of the data? First, the data set assigned integers of 0, 1, 2 and to the iris types.

```
In [5]: iris_data.target
```

```
Out[5]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1,
                1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
         2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
         2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

I hear that most humans prefer words though. So let's look at the translation of these numbers.

```
In [6]: print iris_data.target_names[iris_data.target]
```

```
['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica']
```

Pandas to the rescue

- Pandas will put all the data together in one place.
- The `label` is the type of Iris and what we aim to predict. Each label has an integer `target` that corresponds to the label name.

In [12]:
```
target_names = iris_data.target_names[iris_data.target]
iris_df = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
iris_df['target'] = iris_data.target
iris_df['label'] = target_names
# iris_df.head()
iris_df.tail()
```

Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	label
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

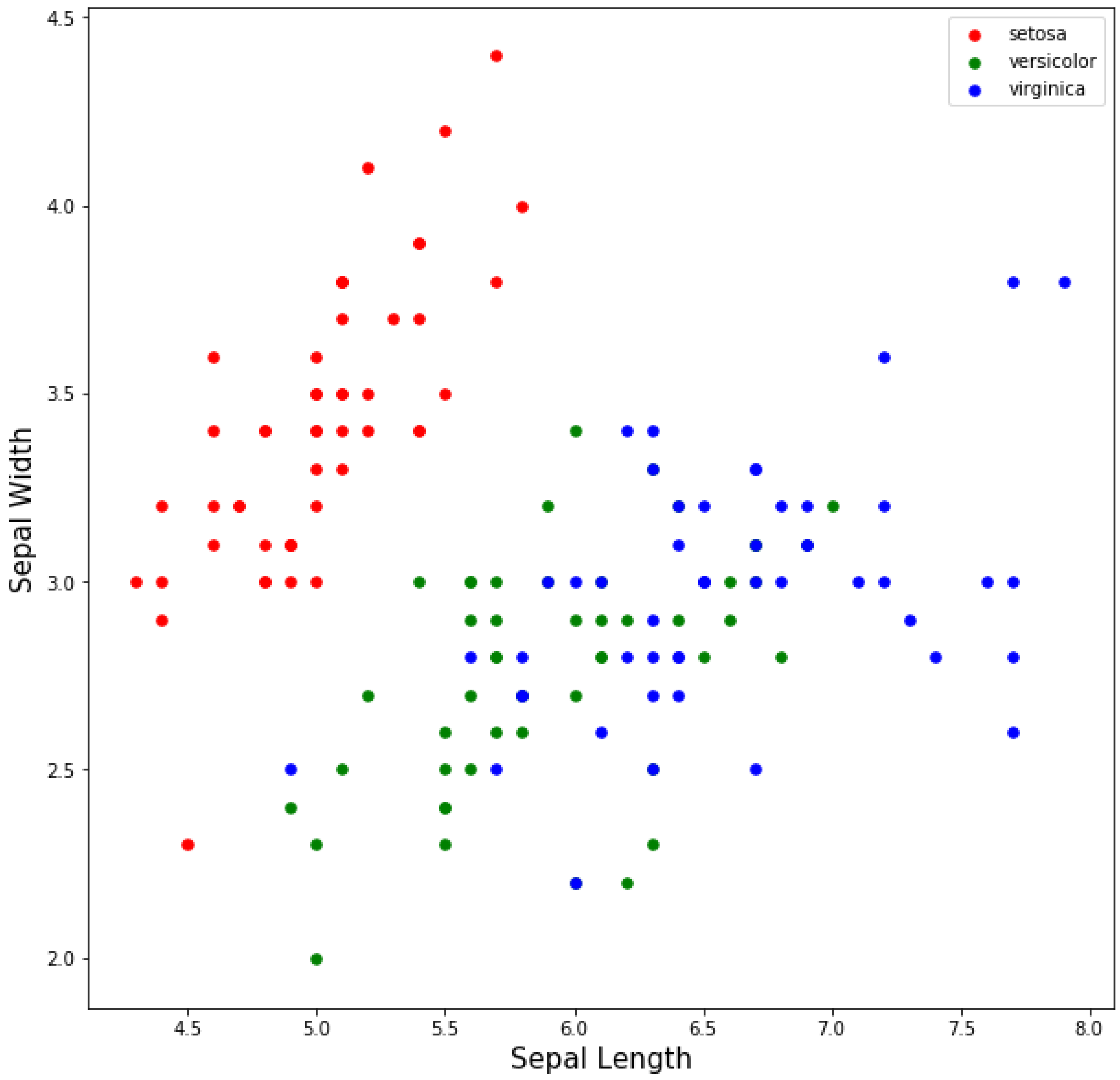
Now some pretty picture

Basically this is making a scatter plot for all the sepal measurement pairs for each iris type. First, a plot of

sepal length vs. sepal width.

```
In [8]: fig = plt.figure(figsize=(10, 10))
        ax = fig.add_subplot(1, 1, 1)
        for target, color in zip(iris_df.target.unique(), ['r', 'g', 'b']):
            sub_df = iris_df.query('target == @target')
            ax.scatter(x=sub_df['sepal length (cm)'].values, y=sub_df['sepal width (
                       color=color, label=iris_data.target_names[target], s=30)
        ax.legend(loc='best')
        ax.set_xlabel('Sepal Length', size=15)
        ax.set_ylabel('Sepal Width', size=15)
```

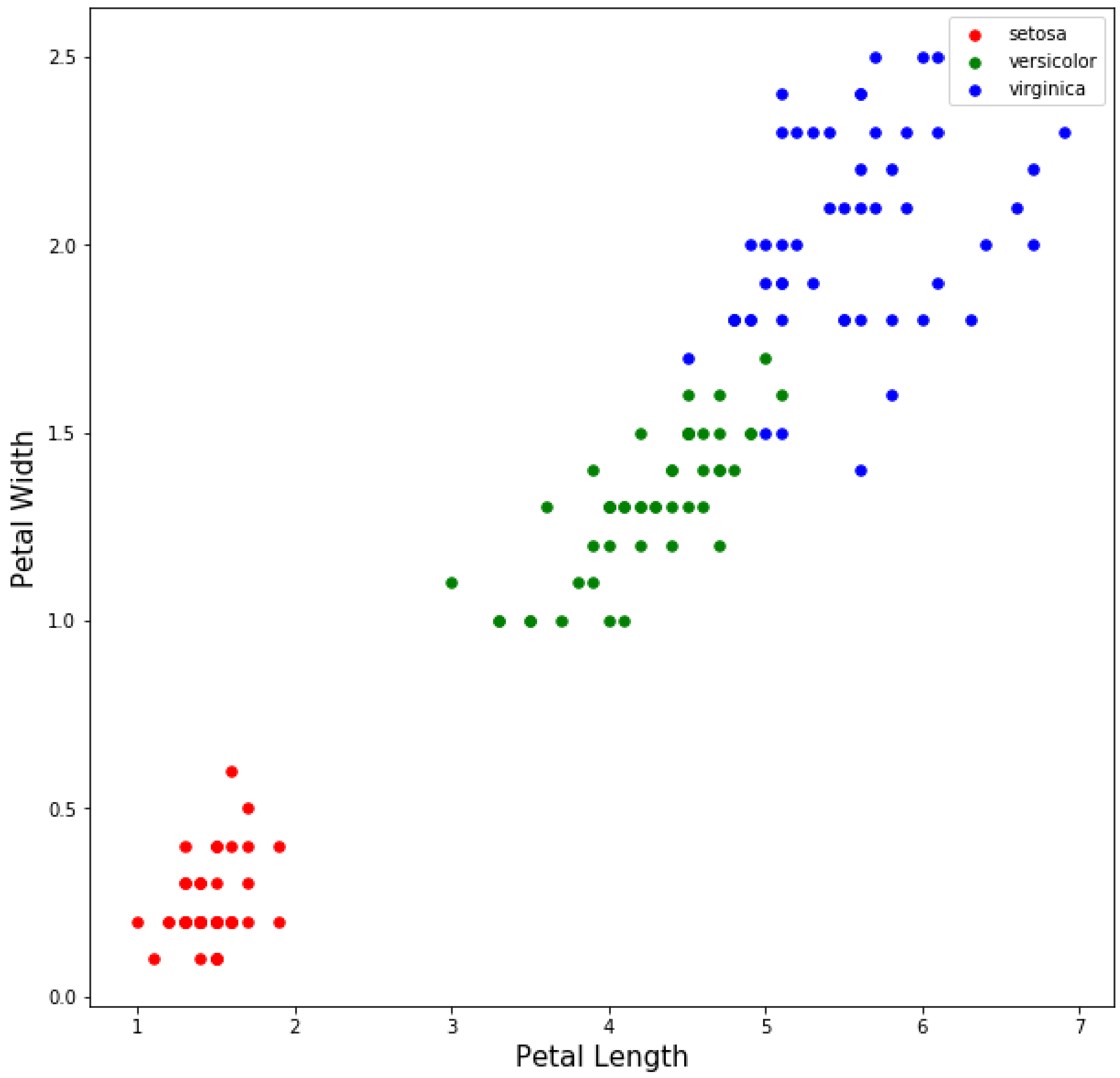
```
Out[8]: <matplotlib.text.Text at 0x11a131890>
```



Petal length vs. petal width

```
In [9]: fig = plt.figure(figsize=(10, 10))
        ax = fig.add_subplot(1, 1, 1)
        for target, color in zip(iris_df.target.unique(), ['r', 'g', 'b']):
            sub_df = iris_df.query('target == @target')
            ax.scatter(x=sub_df['petal length (cm)'].values, y=sub_df['petal width (
                       color=color, label=iris_data.target_names[target], s=30)
        ax.legend(loc='best')
        ax.set_xlabel('Petal Length', size=15)
        ax.set_ylabel('Petal Width', size=15)
```

```
Out[9]: <matplotlib.text.Text at 0x11a489390>
```



Now for some visual observations

Notice how Setosa is always in its own clumps?

Wait...but aren't we going to have the computer do this?

Yes, but we are doing something called supervised learning. Which means we start with data we already know the labels of (like the iris type here) to train the computer from that known source of truth.

Logistic Regression

- Fit a logisitc model that determines irises.
- Test the model on a single observation where it should predict Setosa as our iris type.
- Run the model on all known observations and score its accuracy.

In [10]:

```
X = iris_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', '
y = iris_df['label'].values
logistic_model = LogisticRegression()
logistic_model.fit(X, y)
```

Out[10]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
```

How well does it do?

In [13]:

```
print 'Model guess for a setosa: {}'.format(logistic_model.predict(np.array(
# logistic_model.predict(np.array([[7.2, 2.8, 6.6, 2], [6.2, 2.5, 3.6, 2],
print 'Score {}'.format(logistic_model.score(X, y))
```

```
Model guess for a setosa: setosa
Score 0.96
```

Wow that is a high number! But since we tested the data on our training data, we should get a high number. That isn't a good test though. You should always test on data other than the training set. But that is the jist of the basics.

What we did

- When we first encountered the dataset we explored it with matplotlib.
- Then we trained our classification model, which was logistic regression.
- Finally we will test our model, and discuss ways it could be made better.

BONUS!

Remember how I said it was a good idea to test the model on data other than training data? Here's how to do it.

Splitting test and train data

- Split the data into training and testing data
- Train the model on the training data
- Test model on test data by testing it on data it has not seen before. That is calld out-of-sample data.
- Run the score multiple times.
- Watch the scores change as different splits are made in the dataset.

In [15]:
```
from sklearn.model_selection import train_test_split
logistic_model = LogisticRegression()
trials = 10

for i in range(10):
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    logistic_model.fit(X_train, y_train)
    print 'Score {} of {} is {}'.format(i+1, trials, logistic_model.score(X_
```

```
Score 1 of 10 is 0.921052631579
Score 2 of 10 is 0.868421052632
Score 3 of 10 is 0.947368421053
Score 4 of 10 is 0.921052631579
Score 5 of 10 is 0.842105263158
Score 6 of 10 is 0.894736842105
Score 7 of 10 is 0.921052631579
Score 8 of 10 is 0.973684210526
Score 9 of 10 is 0.973684210526
Score 10 of 10 is 0.947368421053
```

We cut the training and test in multiple locations and that affects their score.

In [ ]: