CIS192 Python Programming
Supervised Learning and Data Visualization

Harry Smith
University of Pennsylvania

October 26, 2016
Outline

1. ML: Concepts
2. ML: Classification
3. ML: Regression
4. ML: General Tips
5. Data Vis: Getting Started
6. Data Vis: 3D
Unsupervised Learning has no knowledge of the labels, and generally seeks to cluster related points.
- Eg. K-Means

Supervised Learning has training data with labels attached. We want to extrapolate from that data to new data.
- We’ve seen the K-Nearest Neighbor and Decision Tree approaches.
- We’ll see more in this class.
Bias is error that emerges from incorrect assumptions in the learning model.

Variance is error that emerges from oversensitivity to small fluctuations in the training data.

The more important the weight of a single datapoint, the higher the variance.

In K-Nearest Neighbors, a higher k means more bias.

In Decision Trees, a greater tree height means more variance.

Overfitting occurs when your model is more attuned to the noise in your dataset than the actual underlying pattern.
Classification vs. Regression

- **Classification** assigns each data point to one of \( n \) distinct groups.
- **Regression** assigns each data point a real number.
  - Eg. a probability in \([0, 1]\) or an estimated height in \([0, 8]\) feet.
K Nearest Neighbors

- `neighbors.KNeighborsClassifier`
  - Parameter: `n_neighbors` - specifies the number of neighbors $k$.

What it sounds like - looks for the $k$ points most similar to a given point in the data and returns the most common label of those points.

- High variance when $k = 1$
- High bias when $k$ is proportional to the size of the dataset.
- Slow: Compares points to every point in the training data.
Decision Trees

- `tree.DecisionTreeClassifier`
  - Parameter: `max_depth` - specifies the maximum tree depth (we can alternatively specify `max_leaf_nodes`).
- Each node splits the data according to a specific feature.
- Greater tree height -> more variance.
- Smaller tree height -> more bias.
Naive Bayes

- eg. `naive_bayes.GaussianNB`, `naive_bayes.MultinomialNB`
- A powerful and efficient algorithm that assumes *independence* between features.
- User specifies the assumed underlying distribution - Gaussian, Bernoulli etc.
- Classifies points using Maximum Likelihood Estimation (MLE) of $P(x, y)$ via $P(x|y)$ and $P(y)$.
Logistic Regression

(Note: Generally used for classification, not regression, despite its name.)

- eg. `linear_model.LogisticRegression`
  - Parameter: `solver` - Specifies the mathematical method used to estimate MLE

- Tries to directly calculate $P(y \mid x)$

- Uses an iterative technique like Gradient Descent to estimate MLE.

- Finds a linear boundary between the two points being classified.

- Primarily for binary decision problems but can also be used in stages for nary classification.
Support Vector Machines

- eg. `svm.SVC`, `svm.LinearSVC`
  - Parameter: `kernel` - a function that specifies the form of the separation

- Carves up the space using *support vectors* - lines of maximum thickness that divide the space into two.

- Primarily for binary decision problems but can also be used in stages for nary classification.
Outline

1. ML: Concepts
2. ML: Classification
3. ML: Regression
4. ML: General Tips
5. Data Vis: Getting Started
6. Data Vis: 3D
Linear Regression

- eg. `linear_model.LinearRegression`
- Assumes the output is a linear function of the input.
  - $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ for some $\theta$s.
  - Note that we can handle polynomials simply by adding $x_i^2$, $x_i^3$ etc. to the prediction data.

- We penalize functions by their euclidean ($L_2$) distance from the line to the point.
Tips and Tricks

- Visualize the data before running an algorithm on it - what kind of approach is appropriate to the problem.
- Partition your data (randomly!) into 3 sets of data:
  - Training (80%): The core training data.
  - Cross Validation (10%): Data left out, test the model with different parameters on it.
  - Test Data (10%): Also withheld, used to determine the ultimate accuracy of the model.

You’ll generally find that your model has either too much bias or too much variance - try to find a sweet spot.

Avoid overfitting - this is generally the point where accuracy on your training set is substantially better than on your cross-validation set.

Don’t become too attached to one algorithm - no amount of tweaking will make the wrong algorithm into the right one.
Outline

1. ML: Concepts
2. ML: Classification
3. ML: Regression
4. ML: General Tips
5. Data Vis: Getting Started
6. Data Vis: 3D
Data Comes in Many Forms

- **CSV**
  - Use native `csv` library from Python
  - Simple, robust
  - Stands for Comma Separated Values
  - Can also read Tab-Delimited Files

- **Excel Spreadsheets**
  - Install with: `pip install xlrd`
  - Plays nicely with the Excel models of Books, Sheets, and Cells

- **Fixed Width Data Files**
  - Use native `struct` library from Python
  - Similar to CSVs but lacking a specific data separator.
  - Implemented in C rather than Python (Cython): very fast!

- **JSON**
  - Use native `requests` library from Python
  - Get data straight from the web.
Simple types of plots to plot

1. `plot()` is a marked scatter plot with the individual data points unenumerated by default.
2. `bar()` is a bar plot.
3. `hist()` is a histogram bar plot.
4. `hbar()` is a *horizontal* bar plot.
5. `boxplot()` is a box and whisker plot.
6. `scatter()` is a scatter plot with line markings turned off by default.
Matplotlib and Formatting the Figures

Methods of changing the appearance of a plot

1. `subplot(int x)` allows you to choose a section of a figure that you want to plot on. For example, `subplot(311)` means that you have a 3-row 1-column plot and you will plot in the 1st (top) section.
2. `title()` gives the graph a title.
3. `xlim()`, `ylim()` allow for the setting of the ranges of the axes.
4. `xticks()`, `yticks()` allow for the placement of tick marks and labels on the graph’s axes.
5. `legend()` generates a legend for your graph. You can specify names for the plotted figures in plotting order or use labels passed in at the time of plotting.
6. `annotate()` allows for the highlighting of a specific value or region.
This goes much deeper than the above.
Visit matplotlib.org to check out all optional parameters for each of the above functions.
- color and colormaps
- thickness
- background coloring
- location on plot
- formatting modes
Pre-processing and Useful Tricks

- **Removing outliers**
  - If you know what behavior your data should follow, you can remove outliers to make the picture better.

- **Smoothing**
  - Sometimes in data presentation, it’s better to show the big idea rather than all the minute details.
  - Can use median filters (`matplotlib.signal.medfilt()`) or averaging boxes (`convolve()`).
Be Honest!

Don’t misrepresent your data! Use the previous tricks to clarify rather than obfuscate.
Outline

1. ML: Concepts
2. ML: Classification
3. ML: Regression
4. ML: General Tips
5. Data Vis: Getting Started
6. Data Vis: 3D
3D Plotting

- **Use** `mpl_toolkits.mplot3d`, **which features the following classes:**
  1. `axes3d` is a 3D plotting library that works very similar to typical `matplotlib` 2D plotting.
  2. `axis3d` is an outdated 3D plotting library that apparently suffers from being buggy and poorly designed. **Avoid!**
  3. `art3d` is a 3D art class which is used to build components of `axes3d`, but has some interesting features of its own right.
  4. `proj3d` is the background class for these others.

- When plotting in 3D, you must always be careful to specify your dimensions.
Functions to produce 3D plots

1. `Axes3D.plot()` gives a marked scatter
2. `Axes3D.scatter()` gives an unmarked scatter plot
3. `Axes3D.plot_wireframe()` plots a transparent mesh of a surface.
4. `Axes3D.plot_surface()` plots a solid surface
5. `Axes3D.plot_trisurf()` plots a solid surface made from a Triangulation object
6. `Axes3D.contour()` plots a 3D contour
7. Others, like quivers, 2D plots, bar plots, polygon plots.