Machine Learning (aka Predictive Analytics)

Definitions

-Unknown function z=f(x,y) including noise

-Function is sampled at various data points: z1 = (x1,y1), z2 = (x2,y2)….

-(x,y) are called features (aka variables)

-z is called the response (aka target, label).

-Data points are called samples

-Machine Learning (supervised): Input data (x,y) + output data (z) -> computer -> program/model -> predict new data.

-Essentially an automated form of performing stats.

-Standard method: input data (x,y) + program/model -> computer -> output data (z)

Unsupervised Learning: Categorize/find trends in an arbitrary dataset (e.g. clustering in the x,y plane). It’s literally pattern matching. It doesn’t necessarily tell us what the pattern represents (e.g. if matching things that look like ‘0’, it won’t identify them as the number 0), it will just recognize that these things are alike.

Supervised Learning (most standard): predict after learning from a set of example datapoints: z=f(x,y)

Regression: Supervised learning with continuous, ordered response (e.g. 0-1000).

Classification: Supervised learning with categorical responses (e.g. yes/no, black/white).

A variety of algorithms!

Image classification is a huge task.

-Unclear what represents the best machine learning algorithm to a given problem.

-Data Science Competitions: Kaggle, training competitions.

E.g. Fitting:

How to fit well but not overfit using least squares? What metric do I use? Use Train-Test Split – if 20 data points total: take e.g. 13 of them and fit the data to those. Then, use that same model and see if it accurately predicts the remaining e.g. 7 data points. This avoids fitting the noise in the test data.

-The better version of train-test split is cross validation.

Algorithms:

**k-nearest neighbour**: See ipython notebook (x axis is e.g. petal length, y-axis = petal height). It essentially categorizes regions (e.g. blue, red, green), and tries to predict what a new data point will be and classify it based on the existing distribution.

-Problems (N=1):

a) Outliers can misclassify things (e.g. a green point making an island in a sea of red).

b) Small sample sizes if uncontested can create a region of classification even if maybe it doesn’t deserve that domain.

-With k=5 nearest neighbours, it makes classification much better. “Islands” disappear, and regions of uncertainty appear (where there isn’t sufficient sampling to create a decisive region).

-Usually more useful to give probabilities of outcomes instead of firm classification.

-What if petal length was measured in km and height in cm? Then, x^2 + y^2 for the nearest neighbours is going to be hugely skewed towards large numbers. Re-scaling variables is really important. Typically what is done is that each axis is normalized to a Gaussian – subtract each value by the mean and divide by the standard deviation (i.e. 0 mean, 68% of data fall in one standard deviation). This assumes equal skewedness in each direction, which is maybe not true. Priors about the data can become very useful!

-Many different algorithms to choose from in sci-kit, and the beauty of it is that they’re all in the same format, i.e. they all just take X, y as input. Each algorithm will have different “hyperparamters” (i.e. particular parameters for that algorithm, the k-nearest neighbours hyperparameter is the number of neighbours).

-Many algorithms to choose from, important to measure the performance in order to decide which algorithm is best! One way is to feed x values and have it predict the y values, and see what proportion of answers it gets right.

Linear regression:

You have data of the volume of a tree ‘y’

F(x) = a\*height + b\*girth + c\*girth^2

A linear regression minimizes the sum of squares between the volume ‘y’ and F(x). The best fit should be somewhere around F(x) = a\*girth^2\*height, but there could be additional terms that matter – e.g. the trunk gets thinner near the top (not a perfect cylinder), so introducing extra terms can help (e.g. b\*girth\*height^2).