**Some useful notes by Connie Sun**

**BASICS**

-restart kernel to change code (if you change code, make sure to run all cells again)

-ctrl+enter or run to execute cell

-hit a to create cell above, b for below

**IRIS DATASET**

Requirements for working with data in scikit-learn:

(response=target/label)

1. Features and response (target/label) are separate objects

2. Features and response should be numeric

iris.target is stored as 0,1,2 instead setosa, etc

3. Features and response should be NumPy arrays

4. Features and response should have specific shapes

feature shape: (first dimension = number of observations/instances, second dimension = number of features/attributes)

(150,4)

response shape: (single dimension matches number of observations)

(150,)

feature data stored in object X (matrix), response data stored in object y (vector)

**KNN CLASSIFICATION**

**-** calculates numerical distance between unknown iris and each of the 150 known irises, and selects the *k* (a number) known irises with the smallest distance to the unknown iris

- response value of *k* nearest neighbors are tallied, whichever is most popular is used as predicted response value

**GENERAL SCIKIT-LEARN MODELING PATTERN (machine learning process)**

1. import class you plan to use

2. instantiate (make an instance of) estimator (model)

all parameters not specified are set to defaults (to see default values, print estimator object)

3. fit the model with data (model training)

model learns relationship between feature X and response y **(don't need to assign it to a new object, because occurs in place)**

4. predict response for new observation

**predict method** expects NumPy array, NumPy will automatically convert lists into an array, returns NumPy array, and can also predict for multiple observations at once

**Evaluation Procedure: Train and Test**

1. train model on entire dataset (.predict(X))

2. test model on same dataset, how well it did (and see **training accuracy** bc same data)

**classification accuracy**: proportion of correct predictions (evaluation metric)

metrics.accuracy\_score(y,y\_pred)

KNN with k=1 is subject to overfitting, and won't do as well in out of samples despite high testing accuracy

**Evaluation Procedure: Train/test split**

1. split dataset into training set and testing set

2. train model on training set

3. test model on testing set, evaluate testing accuracy

**to decide which to train, which to test, optional test\_size parameter between .2 and .4, splits data randomly, unless you give the optional random state parameter a value**

**import matplotlib**

**%matplotlib inline** allows plots and graphs to be shown in notebook

\*after testing and ready to make predictions for out of samples, train on ALL available training data (including what was used as test data before)

**K-fold cross validation**

1. split dataset into k sections (folds)

Repeat following steps K times (but use a different fold as testing set each time):

2. assign one fold to be testing set; ALL other folds together = training set

3. get testing accuracy

4. average testing accuracy indicates out of sample accuracy

**Using K-fold for classification**: use stratified sampling

- in each of the K folds, there should be similar proportion corresponding to each label class (20% ham total observations, each K fold should have 20% ham)

- cross\_val\_score(model parameter (knn), feature matrix X, label vector y, cv=10 (# of folds), evaluation metric) **returns metrics as a numpy array**