This is called DeathToGridSearch because with this example you will never have to think about how to manage a large number of classifiers etc simultaneously. You will now be able to run and collect results in a very straightforward manner. #LongLongLiveGridSearch!

| # Homework 2 import numpy as np from sklearn.metrics import accuracy\_score # other metrics too pls! from sklearn.ensemble import RandomForestClassifier # more! from sklearn.model\_selection import KFold  # adapt this code below to run your analysis # 1. Write a function to take a list or dictionary of clfs and hypers(i.e. use logistic regression), each with 3 different sets of hyper parameters for each # 2. Expand to include larger number of classifiers and hyperparameter settings # 3. Find some simple data # 4. generate matplotlib plots that will assist in identifying the optimal clf and parampters settings # 5. Please set up your code to be run and save the results to the directory that its executed from # 6. Investigate grid search function  M = np.array([[1,2],[3,4],[4,5],[4,5],[4,5],[4,5],[4,5],[4,5]]) L = np.ones(M.shape[0]) n\_folds = 5  data = (M, L, n\_folds)  def run(a\_clf, data, clf\_hyper={}):  M, L, n\_folds = data # unpack data container  kf = KFold(n\_splits=n\_folds) # Establish the cross validation  ret = {} # classic explication of results   for ids, (train\_index, test\_index) in enumerate(kf.split(M, L)):  clf = a\_clf(\*\*clf\_hyper) # unpack parameters into clf is they exist  clf.fit(M[train\_index], L[train\_index])  pred = clf.predict(M[test\_index])  ret[ids]= {'clf': clf,  'train\_index': train\_index,  'test\_index': test\_index,  'accuracy': accuracy\_score(L[test\_index], pred)}  return ret  results = run(RandomForestClassifier, data, clf\_hyper={})  #LongLongLiveGridS#LongLon#LLongLiveGridSearch!gLiveGridSearch! |
| --- |