DeathToGridSearch.py

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!
```