Automation of Biological Research: 02-450/02-750

Carnegie Mellon University

Homework 2

Version: 1.0; updated 2/3/2019

Due: February 25 by 11:59pm

Hand-in via Canvas:

- Your completed code
- A *single* pdf file that contains:
  - your name
  - your Andrew ID
  - your responses to the questions.

Overview

This homework has 2 questions that will require you to implement active learning algorithms covered in lecture. Both questions will require programming, so **START EARLY AND USE YOUR TIME WISELY**. The following versions of python packages used in this assignment should work (earlier versions may also work, but if you encounter problems, try updating each to these versions):

- Python 3.6.3 (python 3 necessary for modal)
- Scikit-learn 0.20.0
- Numpy 1.14.3
- Pandas 1.14.3 (only used in code provided to format the data)
- modAL
  - Note: modAL should be straight-forward to install under MacOS and Windows. We have included some instructions for installing under Linux in the enclosed text file 'modAL_installation_Ubuntu'.

Question 1 (50 points)

This first question will serve as an introduction to working with modAL in order to implement some of the basic active learning concepts we have covered thus far in lecture. This includes

pool-based sampling, as well as query by committee. Conveniently, modAL includes the ActiveLearner and Committee classes, which helps make both of these steps simpler. You will be required to instantiate the appropriate objects using these classes, utilize modAL supported query strategies, as well as implement one of your own. In doing so, you will perform some of your first active learning experiments, and report the results as asked below.

You are provided with an imaging data set "Data.csv"[1]. This data comprises 500 samples each with 26 features. Each sample is labeled with one of ten possible subcellular locations. You are also given the template file "hw2_q1.py". Notice the functions *load_setA()* and *split_data()* are provided (i.e., you don't need to write or edit those). Those functions load and partition the data for you. The remaining functions are left for you to complete (look for lines labeled ###TODO:). In general, **DO NOT CHANGE THE FUNCTION SIGNATURES, AND MAKE SURE THEY RETURN THE VALUES ASKED FOR WHEN CALLED**. Feel free to define any helper functions as you see fit. You may import and use any modules in scikit-learn and NumPy to help with your implementations (modAL is built specifically to support and mimic scikit-learn functionality)

1. *Statistical and visual differentiation of high throughput subcellular imaging*, N. Hamilton, J. Wang, M.C. Kerr and R.D. Teasdale, BMC Bioinformatics 2009, 10:94.

Part 1.1 (5 points)

Complete the functions *active_learner()* and *random_query()*. The *active_learner()* function should return an ActiveLearner object from modAL, and *random_query()* should provide a method for randomly selecting a sample to query from the oracle, and must be consistent with the query methods expected by modAL when called.

Hint: Check the modAL documentation for how to call an ActiveLearner. Within the documentation, check the Extending modAL section for a tutorial on implementing a custom query strategy.

Part 1.2 (20 points)

Since we now can properly call an ActiveLearner object, let's run some experiments that use different sampling methods. modAL natively supports uncertainty sampling, margin sampling, and entropy sampling. Within the function *pool_based()*, for each of these sampling methods as well as random sampling implemented previously, instantiate an ActiveLearner using a Random Forest classifier as the underlying estimator. Do not change the random seed. Perform pool based sampling for 50 calls to the oracle with each ActiveLearner. After each call to the oracle, measure the accuracy of each learner on predictions across the test sets (X_test and y_test).

Report the accuracy of each learner after the final call to the oracle. Additionally, make a plot with query number as the x-axis, and accuracy as the y-axis for each learner (one single plot with four curves, one for each sampling method. Some code is provided that you may use if you want). As stated in the function signature, make sure *pool_based()* returns the final accuracies asked for above.

Hint: Make sure to keep distinct unlabeled pools for each sampling method. The copy.deepcopy() method may be useful. Use learner.query(X_pool,y_pool) to query the oracle, and learner.teach(X,y) show the learner true labels. Just like with scikit-learn methods, you can use learner.score(X,y) to obtain prediction accuracies on data X with true labels y.

Part 1.3 (25 points)

Finally, we will now use modAL to implement query by committee. We will form two committees, one with the natively supported vote_entropy_sampling query method, and another with the random sampling method implemented previously. Within modAL, forming a committee is as easy as creating a list of ActiveLearners, and passing this along with the query method to *Committee()*. The Committee object can then be used exactly as the ActiveLearner objects we used in the previous parts (eg. Use Committee.query(X,y) or Committee.teach(X,y), etc.). Finish the committee() function by instantiating the two committees described above, each with three members. Again use Random Forest classifiers as the underlying estimators, and set the random seeds to 0, 1, and 2. As in the previous section, perform 50 queries to the oracle, and plot number of queries against prediction accuracy across the test sets (so one plot with two curves, one for each committee). To emphasize that there will be disagreement within a committee, record confusion matrices for each of the three active learners within the vote_entropy_sampling committee after the first call to the oracle. Also identify which query number results in the largest difference in test accuracy between the two committees, and what each of these accuracies are. Make sure that *committee()* returns these values.

Hint: As in part 1.2, you may find copy.deepcopy() helpful for ensuring each committee has its own list of ActiveLearners and pools of unlabeled data. modAL intentionally designed the ActiveLearner and Committee classes to be manipulated in the same ways, so you should in principle be able to easily adapt your code from the previous section to work here.

Question 2, implement CAL (50 points)

In this question, you will be implementing CAL algorithm on your own. The pseudocode for CAL can be found in the paper "Two faces of active learning" http://cseweb.ucsd.edu/~dasgupta/papers/twoface.pdf (see the top of page 6). A code template cal.py is also provided with some starter code to help you with the implementation and the use of modAL. You will be also comparing CAL query strategy with random query strategy on the same data set. Additionally instructions can be found in cal.py. Please do not change the function signatures, and make sure they return the values asked for when called. The estimator to be used for this part is the support vector machine (SVM) classifier, and you can simply take advantage of scikit-learn package for convenience.

The data used in this question are synthetic two-dimensional data that are linearly separable by a Support Vector Machine (SVM) classifier. The source code for generating the data is included in the code template just for your reference. You can find the data for this part in the /data folder, named as data_cal.npy and labels_cal.npy, which can be easily loaded into your workspace by np.load() method.

Part 2.1 (15 points): Implement CAL algorithm as the query strategy. Complete the function CAL() in cal.py., and you can find more instructions in detail in the comment under this function.

Part 2.2 (15 points): Complete the function CALLearner() in cal.py, which creates an ActiveLearner object and performs active learning with query strategy defined by CAL algorithm. Please read the comment under this function. For this part, you would need to properly maintained the labeled pool and unlabeled pool of data. Make sure fit the estimator on the current labeled pool of data before calculating the accuracy. You may find the source code in the link https://github.com/modAL-python/modAL/blob/master/modAL/models/base.py useful because there you can find a few methods, such as .query(), ._add_training_data(), ._fit_to_known(), etc. that can help you finish the task and can have a better understanding of the fields in the ActiveLearner object in modAL.

Part 2.3 (10 points): Complete the function RandomQuery() and RandomLearner() in cal.py, which perform active learning with a random selection query strategy. You can find more instructions in detail in the comment under these two functions.

Part 2.4 (10 points): Make the accuracy plots for CAL and the random active learner.