UNIVERSITY OF OXFORD

Advanced Machine Learning

February 14, 2014

Practical 3

This two-week practical will enable you to use scikit-learn to do Gaussian process (GP) regression and random forest (RF) classification. You won't need to know what a random forest classifier is. That is, you will treat the RF classifier as a black-box and carry out Bayesian optimization with GPs. The objective function will be the the cross-validation error of the RF classifier.

Hand in the answers to all questions before you leave on week two.

1 Gaussian process regression

In this exercise, you will learn to carry out GP regression using scikit-learn. You can find the documentation for the GP module here: `http://scikit-learn.org/stable/modules/gaussian_process.html`

1. Open the file `gp_demo.py`. You should read this file carefully and understand what each line does. Run the file. You should see a plot of $f(x) = x\sin(x)$, and a GP regression estimate of this function made using a few points marked in red.
2. The regression in the previous step gave a good fit. But let us now change the squared-exponential kernel width. Find the line that sets `theta0` in the code and change its value from `1e0` to `1e1`. What do you observe? What happens if you decrease the kernel width? Try different values until you have a good feel for how the kernel width affects the solution. You can also try different amounts of noise or vary the input training points.
3. Try a different type of kernel. To see what kernels are available, look at the documentation for the object *GaussianProcess* in scikit-learn.
4. There are some commented lines that set `thetaL` and `thetaU`. What happens when you uncomment these lines? Try different starting points for the maximum likelihood optimizer of the kernel hyper-parameter.

If you've done all the steps above and feel very comfortable with GP regression using scikit-learn, you can now embark on the development of a Bayesian optimization method.

2 Bayesian optimization

In this exercise you will implement Bayesian optimization using the expected improvement criterion to optimize a synthetic function.

1. Open the file `simple_bo.py` and read the comments. Try running the file. Make sure you understand what all the different things this plot is showing you are. EI and the GP predictions appear on the plot, but the code to compute them hasn't been written yet (that's what you're about to do) so don't be surprised that the predictions don't make sense right now.
2. Open the file `bo.py` and read the comments. This class implements Bayesian optimization, although there are a few pieces missing that you will need to fill in. Answer the following questions before moving on:
   (a) Where is the training data for the GP stored?
   (b) Where is the objective function stored? Where is it evaluated?
   (c) Where is the GP trained?
   (d) Describe the sequence of events, starting with instantiation of a BO instance, that must happen to cause the GP to be trained.
3. Find the `predict` function in `bo.py`. This function is supposed to compute predictions from the GP posterior, but right now it always predicts zero mean and constant variance. Implement this function. Make sure you return different things depending on the value of `predict_variance`. Remember you can run `simple_bo.py` to see your predictions in action.
4. Find the `expected_improvement` function in `bo.py`. This function is supposed to evaluate the expected improvement criterion but right now it just returns random numbers. Implement this function. Again, you can always run `simple_bo.py` to see your EI computation in action.

3 Random forests classifiers for MNIST

In this exercise you will apply the Bayesian optimization code you wrote to a hyperparameter tuning problem. Specifically, you will be training random forest classifiers on MNIST digit data.

You don't need to know what a random forest is to complete this exercise, but knowing a little bit about them will help you interpret the results.

You can read a brief introduction to random forests here: `http://scikit-learn.org/stable/modules/ensemble.html` and you can find the full documentation for scikit-learn's random forests here: `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestCla` `html` You can see from the documentation that this algorithm has several parameters.

You will be using random forests to classify MNIST, which is a very commonly used benchmark data set in machine learning. You can read about MNIST here `http://yann.lecun.com/exdb/mnist/`. You don't need to download MNIST from that site, we've provided a pre-processed version with the practical.

1. Run `mnist.py` to see some random digits from MNIST (run it again you'll get different ones). The task for this data set is to predict the image labels from the pixel values.
2. Use scikit-learn's random forests to classify MNIST. Evaluate your model using 3 fold cross validation (look in `black_box/objectives.py` for an example of how to do this). There is no code to modify for this task, you need to start from scratch.
   (a) What is the best cross validation accuracy you can get by selecting parameters by hand? (Don't set `n_estimators` higher than 10. In practice you might do this, but we want a fair comparison to Bayesian optimization in the next step).
   (b) What parameter settings give the best accuracy you found?
3. Use the starter code in `tune_random_forests.py` to tune random forests using Bayesian optimization.
   (a) What parameters does the code optimize?
   (b) Were you able to find better parameter settings by hand than using BO?
   (c) How far are your results from the state of the art? (See `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#4d4e495354`). Don't be surprised if the answer is "quite far," you're only working with a subset of MNIST.

Throughout this exercise you have been optimizing cross validation accuracy. Sometimes you care not only about the accuracy of the predictions but you also need to make predictions very fast. For example, the Goalkeeper CIWS needs to be able to predict the path of a missile fast enough to shoot it down. Speed is also important when you have a lot of predictions to make (like classifying every page on the Internet).

How would you use Bayesian optimization to find a classifier that is both fast and accurate?

Are there other than accuracy and prediction speed that you might want to optimize?