Assignment 1 (7 pts)

Due February 7, 2018

Read the PythonSetup.pdf *document (posted on Quercus) before beginning this assignment.*

Q1) 2pts Implement the k-NN algorithm for regression with three different distance metrics (ℓ_2, ℓ_1, and ℓ_∞). Use 5-fold cross-validation[1] to estimate k, and the preferred distance metric using a root-mean-square error (RMSE) loss. Compute nearest neighbours using a brute-force approach.

Apply your algorithm to all regression datasets (use `n_train=1000, d=2` for `rosenbrock`). For each dataset, report the estimated value of k and the preferred distance metric, and report the cross-validation RMSE and test RMSE with these settings. Format these results in a table.

Plot the cross-validation prediction curves (merging the predictions from all splits) for the one-dimensional regression dataset `mauna_loa` at several values of k for the ℓ_2 distance metric. In separate figures, plot the prediction on the test set, as well as the cross-validation loss across k for this model. Discuss your results.

Q2) 1pts Implement the k-NN algorithm for classification with three different distance metrics (ℓ_2, ℓ_1, and ℓ_∞). Estimate k and the preferred distance metric by maximizing the accuracy (fraction of correct predictions) on the validation split. Compute nearest neighbours using a brute-force approach.

Apply your algorithm to all classification datasets. For each dataset, report the estimated value of k and the preferred distance metric, and report the validation accuracy and test accuracy with these settings. Format these results in a table.

Q3) 2pts Test the performance of your k-NN regression algorithm with the following modifications

(a) Write a brute-force k-NN approach that uses a double for-loop over testing points and training points, computing the distance between each one at a time.

(b) Write a brute-force k-NN approach, replacing the for-loop over training points with vectorized python code to compute the distance between a test point and all training points in one line as follows, `np.sqrt(np.sum(np.square(x_train-x_test), axis=1))`, where it is assumed that the shape of `x_test` is $(1, d)$. Note that this takes advantage of numpy broadcasting. You should still loop over testing points.

(c) Write a fully vectorized brute-force approach to compute the distance between many test points and all training points. In other words, write a brute-force k-NN regression algorithm that can make predictions on many test points simultaneously without the use of *any* loops. *Hint:* consider introducing a third

[1] Note that `data_utils.load_dataset` returns a training and validation set, however, to perform cross-validation, merge these two sets first, i.e. for the inputs: `x_train = np.vstack([x_valid, x_train])`

dimension to your data arrays using `numpy.expand_dims`, and be sure to take advantage of broadcasting.

(d) Write an implementation that uses a k-d tree data structure[2] to compute the nearest neighbours for multiple test points simultaneously. Ensure that there are no loops in this k-NN implementation.

Conduct your performance studies by making predictions on the test set of the `rosenbrock` regression dataset with `n_train=5000`. Report the run-time of all four approaches, over varying values of `d` in a single plot. Use the ℓ_2 distance metric and $k = 5$. Comment on how the vectorization of python code effects performance. Also, comment on the relative performance of the k-d tree algorithm verses the brute-force approach. Use the `time.time` function to measure elapsed wall-clock time.

Q4) 2pts Implement a linear regression algorithm that minimizes the least-squares loss function (using the SVD). Apply to all datasets (regression and classification). Use both the training and validation sets to predict on the test set, and format your results in a table (present test RMSE for regression, and test accuracy for classification). Compare the performance of this method to the k-NN algorithm.

Submission guidelines: Submit an **electronic copy** of your report (**maximum 6 pages** in at least 10pt font) in **pdf** format and **documented** python scripts. You should include a file named "README" outlining how the scripts should be run. Upload a single tar or zip file containing all files to Quercus. You are expected to verify the integrity of your tar/zip file before uploading. Do not include (or modify) the supplied *.npz data files or the data_utils.py module in your submission. The report must contain

- Objectives of the assignment
- A brief description of the structure of your code, and strategies employed
- Relevant figures, tables, and discussion

Do not use scikit-learn for this assignment, except where explicitly specified. Also, do not use the `scipy.spatial` module in this assignment. The intention is that you implement the simple algorithms required from scratch. Also, for reproducibility, always set a seed for any random number generator used in your code. For example, you can set the seed in numpy using `numpy.random.seed`

[2]We do not expect you to implement this data structure. Instead, you may use the scikit-learn implementation `sklearn.neighbors.KDTree` with the default parameters.