Data Mining: Learning From Large Data Sets

Fall Semester 2015

{usamuel, adavidso, kurmannn}@student.ethz.ch

November 5, 2015

Large Scale Image Classification

Problem formulation. The goal of this project was to construct an SVM that classifies images into one of two classes, these classes being images of people versus images of nature. The images were provided in the form of precomputed feature vectors, each containing 400 features.

Approach and Results. We constructed a classifier by running SGD in each mapper. Each mapper reads values (y_t, x_t), for $1 \leq t \leq T$, and then computes w_t using the SGD algorithm, as presented in class. The final result of the mapper is the average of all w_t. The reducer collects and averages over these hyperplanes, and outputs the final classifier w.

We tried several approaches to compute a good classifier for the image classification problem.

At first, we used OCP to compute a linear classifier. The features were used "as is" with the addition of a constant 1 to account for an intercept. We chose $\eta_t = 1/\sqrt{t}$ and $\lambda = 1$. This approach yielded a score of about 60 percent.

To improve on this, we tried to process the feature vectors in random order, as is done in SGD. We adapted the mapper to first store the entire input in a matrix. This matrix was then processed uniformly at random. To make the most out of the available computation time, we kept on reading from the matrix in random order for around 4.5 minutes. In addition, we chose $\lambda = 1/N$, with N being the number of feature vectors passed to the mapper, and $\eta = 1/(\lambda t)$. This approach yielded a score of approximately 70 percent.

We were not able to improve on this and therefore assumed that the feature vectors were not (sufficiently) linearly separable. We experimented with several non-linear feature transformations, and were able to boost our score to 81 percent with the following transformation, which uses built-in NumPy functions:

```
x <- [numpy.sqrt(x), numpy.diff(x), numpy.gradient(x)].
```

This might seem like an odd transformation, but the motivation for doing this was that the original feature vectors were sparse, and so we wanted to remove some of this sparsity, which is why we use `diff` and `gradient`.

Finally, we were able to beat the hard baseline by using random Fourier features on top of the above transformed features. Sampling 3,500 of these resulted in a score of 84 percent. At this point in time, we noticed that the first entry in every original feature vector in the training set was zero, so we dropped this feature.

Workload distribution. Nico and Alexander wrote the initial structure for the mapper and reducer. Samuel implemented the basic OCP algorithm. Alexander extended this to include SGD and he also came up with the non-linear transformations. Samuel implemented random Fourier features (with errors), and Alexander corrected the errors, which then resulted in our final mapper. Everybody contributed to the report.