The Methylation Machine Learning Challenge: pair-up methylation profiles by subject from two timepoints

Peter Hebden

25 January 2019

1 Introduction

This machine learning challenge is to match pairs of vectors of equal length that contain floating point numbers. Each vector represents a methylation profile. There are profiles from 74 subjects; two profiles per subject, one from each visit to the clinic. Full methylation profiles are 870K sites long, and they would be easy to match. However, can an algorithm match profiles with just 217 sites?

In the training set, the first number in each row of data identifies the subject. In the test set, the 1st number in each row is just a row number.

Given a randomly shuffled set of profiles from the second visit, match each row of the visit 1 profiles with each row of the unseen visit 2 profiles and provide a probability that the match is correct.

Evaluation is by log-loss. In theory, to get the maximum score, make correct predictions of a match with a probability p=1, and correct predictions of a mismatch with a p=0, e.g. log(1) + log(1 - 0) = 0. In practice, due to the way the system computes the scores, aim for probabilities that are between 1 and 0, e.g. giving p=0 for a correct match results in log(0) and a large penalty. But for convenience and clarity, I will use p=1 and p=0 in this report.

Matching two vectors of numbers based on a distance metric sounds easy. Perhaps just calculate the squared Euclidean distances between the first visit 1 profile and all visit 2 profiles and predict that the two closest profiles are from the same subject, assign that match p=1 and all others p=0; Then do the same for the second visit 1 profile, and so on. But that did not get a good score. So, what worked?

Well, the data set is unusual. There are only a few positive examples to learn from, and many negative examples. It seems impossible to build a classifier that separates matches from mismatches with a classic machine learning algorithm. There are many features, but perhaps some are noise or redundant. I tried PCA to reduce the dimensions and extract the most important features, and then applied a distance metric. Greater distance from a visit 1 profile implies a lower probability. No luck.

For this data set, the Pearson correlation coefficient provided a good start. In theory, use pairwise correlations as the measure of distance, and a tight probability distribution that assigns close to p=1 for the pair with the highest correlation, and close to p=0 for the other pairs. Do this for each visit 1 profile. That worked okay. After adding some details, this algorithm iterated over the data set, slowly tightening the probability distribution and knocking out profile features one at a time if that knock out improved the score. The best algorithm used a form of gradient descent that slowly tightened two parameters of the probability distribution. Final score: 0.00186.

```
for each epoch
  for each feature
    knock out feature temporarily
    cc = corrcoef(v1, v2); {do pairwise correlations}
    distances = 1-cc;   {distances for each pair}
    sd = std(distances);
    z_scores = distances/sd;
    mu = minimum(z_scores);
    probabilities = M *(1-normcdf(z, mu, sigma)) {calc probs}
    score probabilties
    if knockout improves score → knock out feature permanently
  end

  if odd epoch → decrease sigma
  if even epoch → increase M

  update probablities
  update scores
end
```

Figure 1: Pseudo code for profile matching. Keep trying slowly, then give up.