Stabilization Matrix Method

Sigmar Stefànson and Francesco Favero

DTU

29 Oct. 2010

Code Change

We used brute-force method to evaluate the optimal lambda. To be able to run the smm algorithm as efficiently as possible we made some minor changes. Some of these changes are probably made by the compiler optimization.

Another change we made was the ability to evaluate model efficiency on training and testing data for each cycle side by side. The test data file is then an extra parameter to the program.

Code Change

We used brute-force method to evaluate the optimal lambda. To be able to run the smm algorithm as efficiently as possible we made some minor changes. Some of these changes are probably made by the compiler optimization.

Another change we made was the ability to evaluate model efficiency on training and testing data for each cycle side by side. The test data file is then an extra parameter to the program.

Our update_weight implementation, optimization mainly involving join of parameters and removal of memory allocation.

```
inline void update_weight(float *w, float *inp, float d_o, int n)
{
    int j;
    float p_etad_o = p_eta * d_o;
    for ( j=0; j<n; j++ )
    {
        w[j] -= p_etad_o * inp[j] + p_eta2lambda * w[j];
    }
}
```

Running the algorithms for all the 35 alleles with different λ values we improved the overlall running time by parallelisng the processes. In short, a script performed the algorithms with all the needed λ at the same time on different groups of alleles

Our update_weight implementation, optimization mainly involving join of parameters and removal of memory allocation.

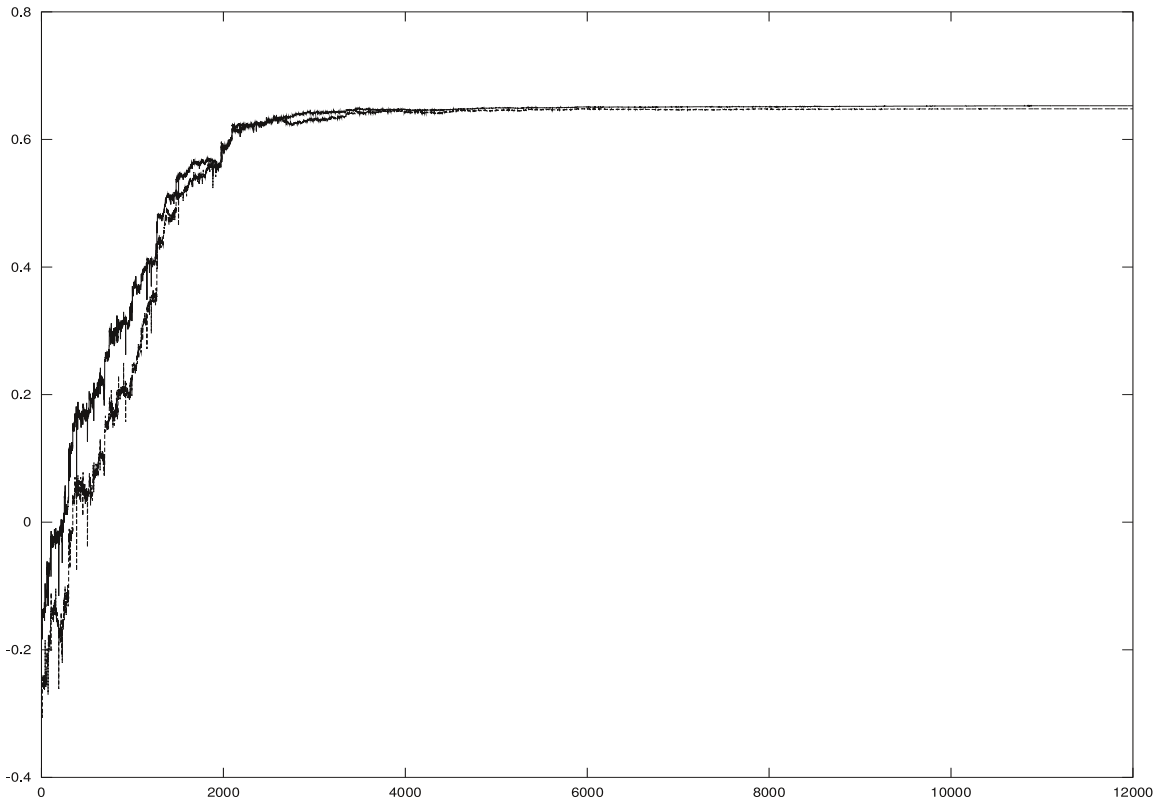
```
inline void update_weight(float *w, float *inp, float d_o, int n)
{
    int j;
    float p_etad_o = p_eta * d_o;
    for ( j=0; j<n; j++ )
    {
        w[j] -= p_etad_o * inp[j] + p_eta2lambda * w[j];
    }
}
```

Running the algorithms for all the 35 alleles with different λ values we improved the overlall running time by parallelisng the processes. In short, a script performed the algorithms with all the needed λ at the same time on different groups of alleles

Training vs testing Pearson correlation coefficient

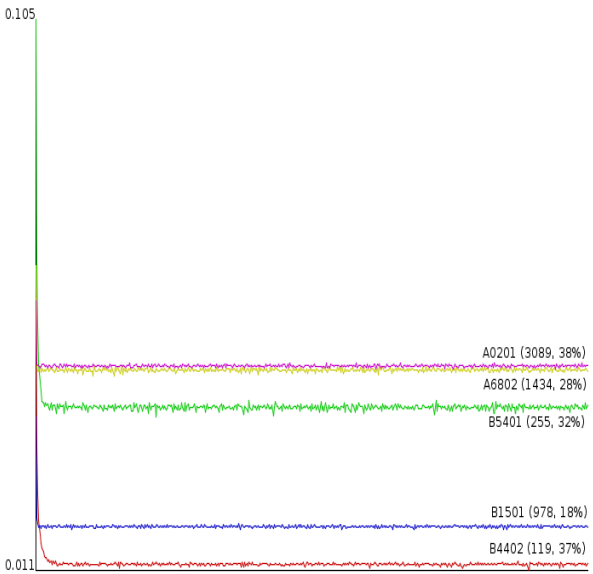
We added the ability to watch testing PCC and MSE for each step in the training (in addition to the training values). The purpose of this was to be able to investigate over-fitting.

Training vs testing PCC

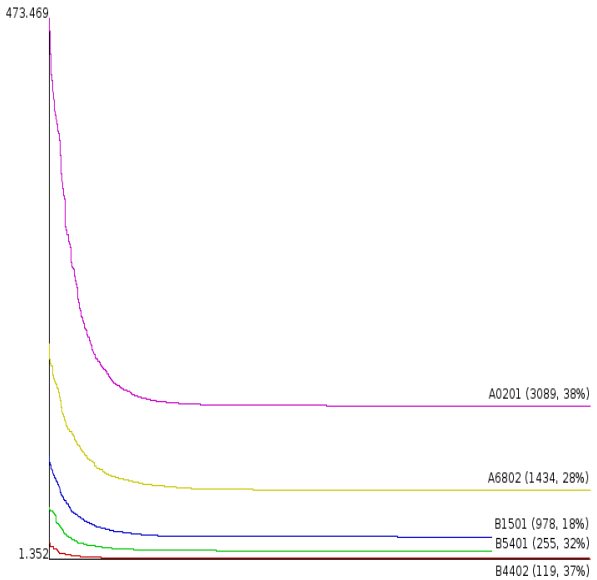


Difference in error values for Monto Carlo and Gradient Descent implementations for 12000 and 500 iterations respectively

Gradient decent

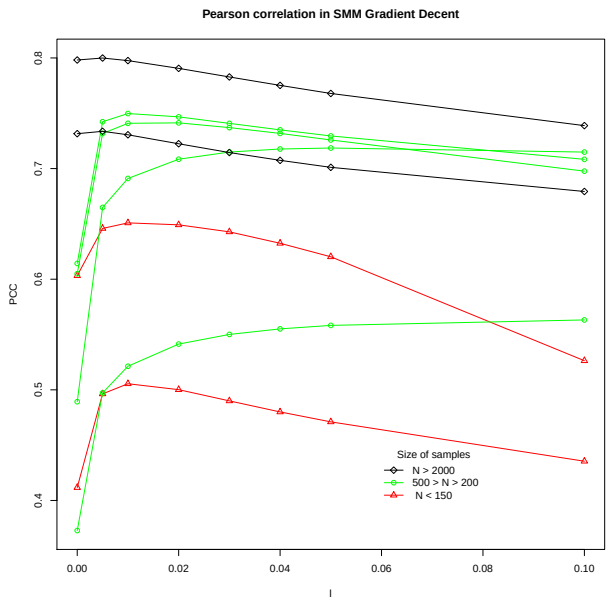


Monte Carlo

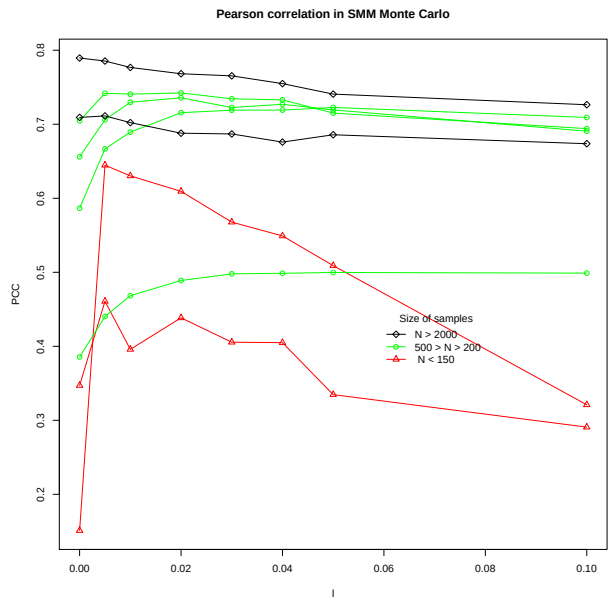


Size depending Pearson correlation coefficent

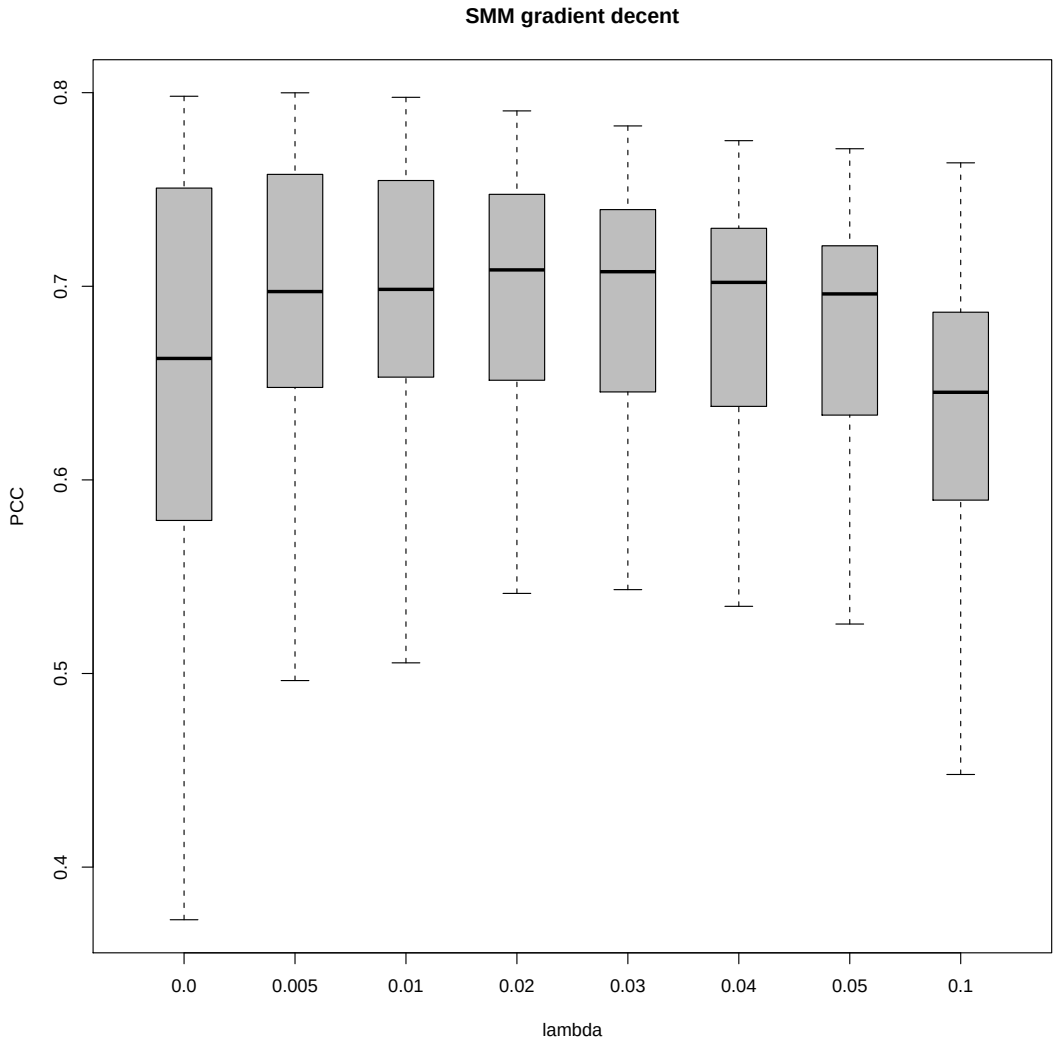
Gradient Decent



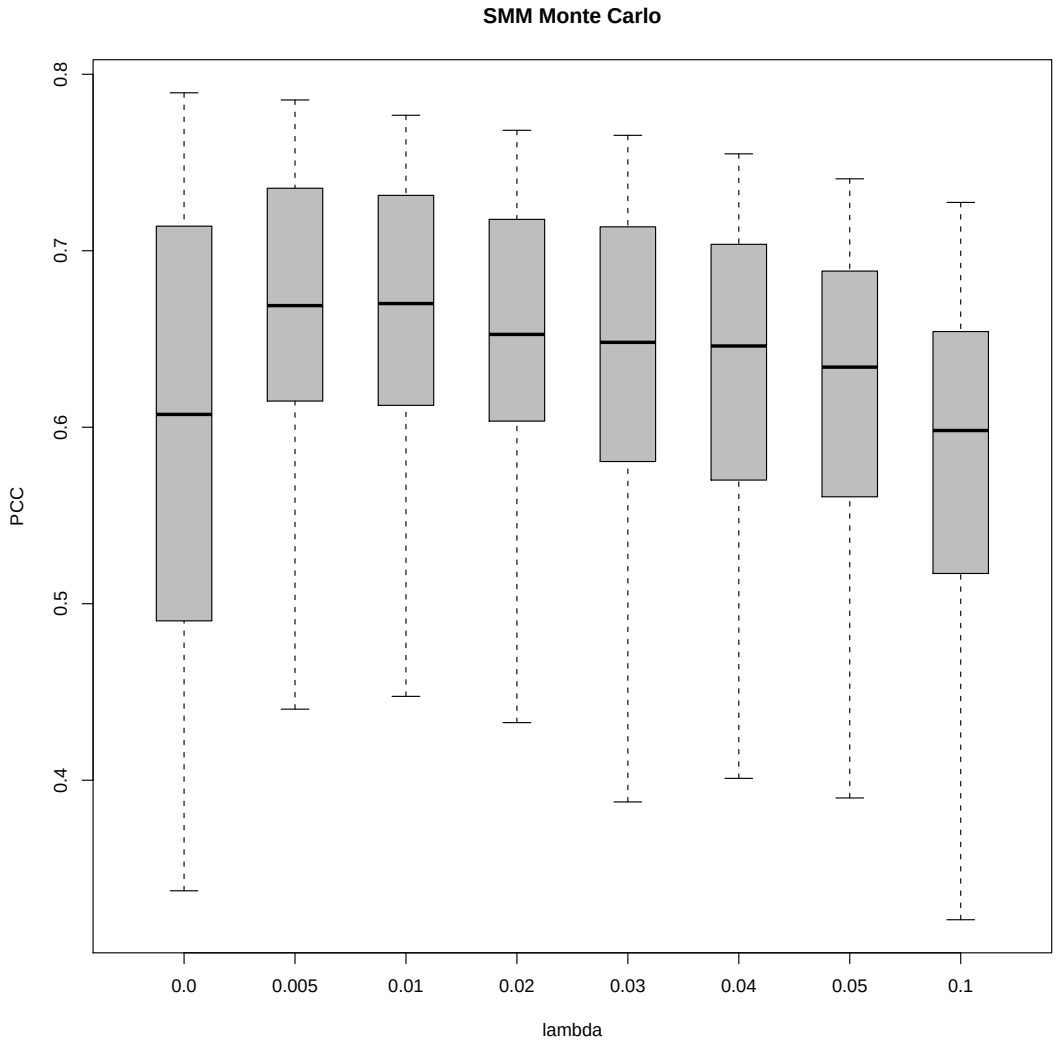
Monte Carlo



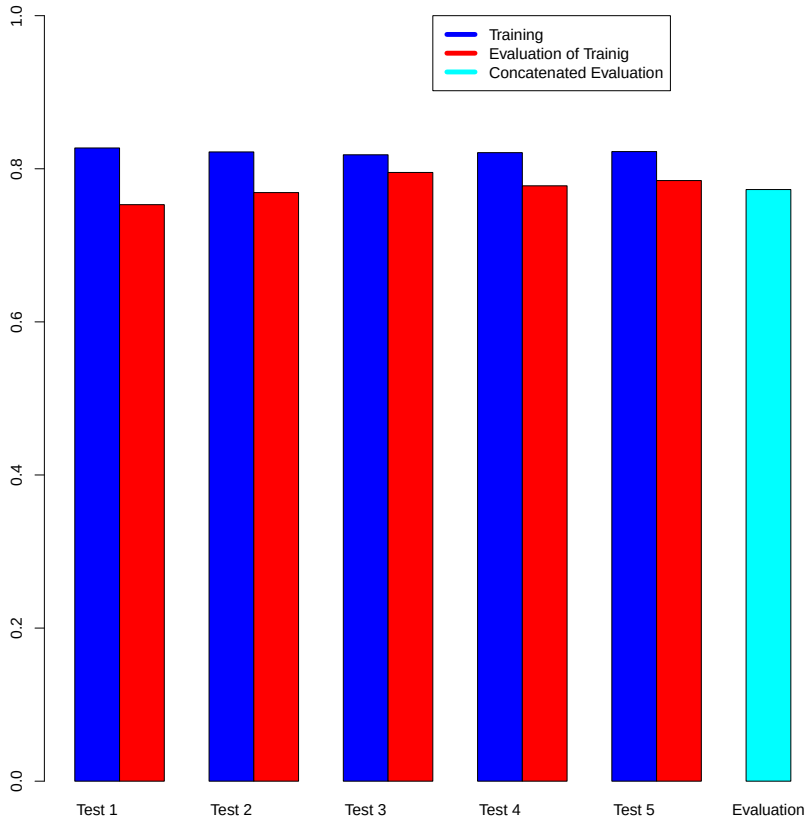
Choice of best λ value for gradient decent



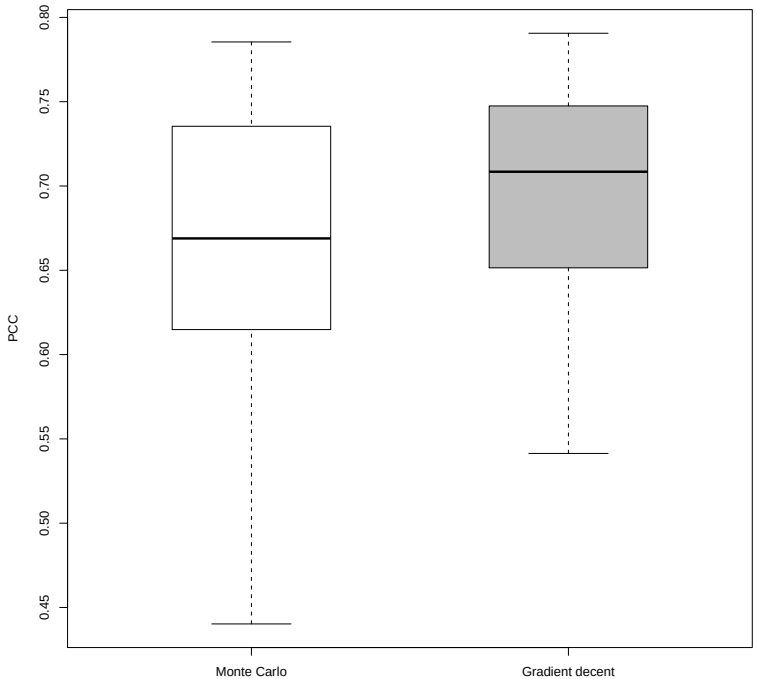
Choice of best λ value for Monte Carlo



Evaluation



This figure shows the difference between the prediction quality on the training vs testing data for each step in the K-fold cross-validation. The bar to the right shows the prediction quality on the concatinated data.



Conlclusion

The Gradient Descent was both faster and created more efficient models than the Monte Carlo one.

Not only we have better correlation with gradient decent, but also the results are included in a smaller range. The algorithms at optimal λ is less depending on the size or other differencies in the data.