**Random forests**

In addition to a straight forward random forest with 2000 trees (guessing Yang is writing about this), we used the tuneRF function find the optimal m\_try. It also gives us the variable importance based on OOB. We then used rfcv which cross-validates to find the optimal number of predictors, and used only these predictors in the random forests function. It turned out that the optimal model contains almost all the original predictors and as we know from lecture random forest is not really affected by a few noisy predictors. The results confirm this, so the method is no better or worse than straight forward random forests.

**Discriminant analysis**

We have seen that the linear classifier logistic regression performs really well, so it could be interesting to see how LDA performs. If, by some reason, the data is somewhat Gaussian, LDA might be a good method. With all the predictors the result was correctness of around 40%. One could consider to find the threshold from CV, but we just used the out-of-box threshold.

We know LDA use the covariance matrix of our predictors, so one could assume it is pretty sensitive to noise. Therefore we sorted the variables using the variable importance in random forests. We cross-validated to find the optimal number of predictors in LDA. The result was pretty unstable, but between 30 and 200 predictors. The methods performance was a little worse than random forest, so a huge improvement from LDA, but not anywhere near our best methods. We could have used other forms of variable selection, but we had the information from RF available, and the resulting CV-error was a convex function in nr of predictors, so it is in some sense optimal. We considered the chances for other variable selection methods improving the LDA beyond SVM's was too small to spend more time on.

As we got a fairly good performance out of LDA with variable selection, it could be interesting to test QDA and RDA in the same framework. However, we demand fewer predictors than then elements in the smallest class. As we used CV to find the optimal amount of predictors, the training sets had to few elements in each class, so the results was not usable. We concluded that our training set was to small for this method (and RDA).

We also made a brief attempt to kernelize LDA (Kernel Fisher Discriminant). After some numerical issues and poor results, we decided out time might be better used on other algorithms.

**Local Logistic**

As logistic regression perform so well, we thought a local version might improve our results. The package locfit has the ability to do local logistic regression, but only for 2 classes. We were not able to make it work with more than 3 predictors, and the results we got was horrible. The example sets for the method was really simple (2 predictors, 100 points, 2 classes), so we assumed the package might not be fully developed yet (or we were not able to use it right). So we gave up.

**Naive Bayes**

Under the assumption that our dataset was a little small, we decided to try the Naive Bayes classifier. We used naiveBayes from e1071, which assumes Gaussian distribution of metric predictors. It gave us and CV correctness rate of 24% , so a few persent worse than random forests. We have not really discussed the how to solve for the functions in this algorithm. All we know is that although it has the form of a GAM it is not solved the same way. Without this information we could not really modify the algorithm.

**Neural Nets**

At the end we tried to fit neural nets. We used the caret package to tuned it but without good results. We should maybe have tried to tune on our own, but we did not.

**Super Class**

One final idea was to cross-validate all our functions and record their performed on the individual classes. You can see an example for penalized logistic regression under. We did a 10-fold CV to estimate the errors and their SD. Ideally we should have repeated this (to get better estimates of SD) or increased nr of folds (to get better estimates), but as some method are really computationally extensive, we chose to do everything like this.

The “should” column is the nr of missed classifications over size of the class, (pred[class k] != y[class k] / size(class k)) or nr of points that should have been classified as k, but was not, divided by size to the class k.

The “shouldn't” column is nr of points wrongly classified in class k, over number of points classified as k.

We hoped that some of our algorithms would be better at classifying certain classes, so we could combine our method to include the results from the best classifiers at every point. However, it seems like a method is either better at everything of worse.

$tot

mean sd

0.16750000 0.04972145

$indiv

should sd shouldn't sd

1 0.22379870 0.07063279 0.22730221 0.11614462

2 0.01547619 0.03274711 0.08682040 0.08900461

3 0.15968615 0.16623279 0.15852442 0.12274507

4 0.18758603 0.11062367 0.18198635 0.11383232

5 0.14728355 0.10696839 0.18045649 0.13659406

6 0.32284632 0.20119952 0.25922990 0.14586505

7 0.10276611 0.10193176 0.08619048 0.06247751

8 0.20033272 0.12452617 0.10250722 0.10810794