Machine Learning:

Karsten Standnes - STNKAR012

October 2017

1 Introduction

In the world today there are many methods that fall under the category of "Machine Learning", some being very similar and some very different. All of them share in common that they use data to produce a model that can be used on unseen data. When successful this is very appealing in today's society where we got lots of data on situations where there is likely to be a underlying pattern, another requirement for learning. There is broad agreement that Machine Learning is a good way to make prediction and classification models, and often the only computational feasible way to do so. This makes it a task to decide which method in machine learning to chose for a given problem. The answer is not always the same and several methods can be good, but for different reasons. In this task I will show several different machine learning algorithms and how they perform on classifying pictures of handwritten numbers into even and odd numbers.
COMMENT on data snooping

Contents

2 Theory

2.1 Tree based methods

2.1.1 Classification tree

Classification tree is maybe the method in Machine Learning which is easiest to interpret due to it's intuitive construction and clear visualization. The method uses a greedy approach using recursive binary splitting to structure a tree that can classify input based on it's variables. The goal of the classification tree is to classify a set of data as best as possible while have a low complexity to avoid overfitting. Below we see that one classification tree is not enough to make a great model for the problem, but combining many of them gives us a "Random Forest" which is discussed in subsection 2.1.2. Classification trees also gives a nice visual image of the classification.

2.1.2 Random Forest

Random Forest is as mentioned (subsection 2.1.1) constructed of many classification tree. After such a construction by using a set of training data, new data can be ran through the "forest". The new data is classified with the label the majority of trees labelled it. The trees in other words "vote" for the best classification for the data. As in real life, voting makes little difference if all the votes are the same. To avoid making a forest out of n ($n \in \mathbb{Z}_{>0}$) identical trees two aspects are changed when the constructing the classification trees. The first is that the data of size N used in training a tree is sampled from the whole set of N data **with** replacement. This will on average lead to about $\frac{2}{3}$ data to be used in creating each tree leaving on average $\frac{1}{3}$ Out-Of-Bag(OOB) which is used for validation. The OOB-data is critical in making insuring the Random Forest doesn't overfit. The other is that m randomly selected variables are used for each tree, where $m << M$ and M is the whole set of variables in the problem. This gives a rich variety of trees where with enough trees a predictive model can be created.

2.1.3 Bagging

Bagging is basically a Random Forest (see 2.1.2) where the number of variables considered for each split is the whole variable set for the problem. It shares the same behaviour as a Random Forest when sampling out the data and in classifying new data. The big difference is that the variables are not sampled in Bagging, this does that Bagging creates trees that are more correlated than a Random Forest making it more susceptible to dominating features. This will not directly lead to a bad classification model, but it does have the unwanted consequence of making semi-important features of the data to be underestimated in the contribution towards new predictions.

2.1.4 Boosting

Boosting takes a different direction to find make a good predictive tree model than the two previous tree based methods. Instead of growing n trees independently of each other, boosting "grow" trees sequentially moving towards a good model. Boosting is a slowly improving method and often use a small amount of splits for each tree. It also have a shrinkage parameter λ that controls the rate of change in the model. Boosting learns slower, which means more trees are required compared to the random forest or bagging method in order to achieve the same predictive power.

2.2 Deep Learning

2.2.1 Artificial Neural Network

Artificial Neural Networks got the name because it to some degree mimic the behaviour of neurons in the brain. It does so by connecting nodes (neurons) together with weights (synapses) connecting them. In the nodes that are not input nodes there is an activation function altering the values that are brought to the nodes through weights from other nodes. The activation is one of several factors that can be tuned when creating a neural network. The relationship between the weights can be seen the equation for calculating the nodes in the hidden layers

$$x_j = \theta(s_j^{(l)}) = \theta(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}) \tag{1}$$

, where l indicate the layer, i the input, j the output and θ the activation function. Other factors that can be altered is the size of the hidden layers and the amount of nodes in each layer of the network. This method learn through updating the weights of using stochastic gradient descent(SGD). This is applied through the backpropagation algorithm. This method takes run data forward through the network and the backpropagates back updating the weights based on the error in the output using SGD. The backpropagation will penalize weights that contributed a lot to the error more than those less relevant. Neural networks are exposed to overfitting, but there are several techniques that can be used to decrease the chance of overfitting. Some of these are validation, using dropout in input and/or hidden layers and regularization.

2.2.2 Convolutional Neural Network

Convolutional Neural Network(CNN) is in theory a great method to classify digits because of it's great reputation in performance for problems that can be represented as an "image". This because it CNN's works by recognizing patterns in problems that can be represented as by matrix or tensor(multidimensional matrix) in the computer. It does so by using convolutional, pooling and voting layers. Multiple of these can be stacked to create a very a precise model for recognising images.

2.3 Support Vector Machines

Support vector machines(SVM) is a powerful methods used either to classify data or in regression analysis. SVM tries as big a margin as possible between different categories. This is strived after to decrease the number of possible dichotomies. SVM's are powerful because of the ability to classify non-linearly separable data by transforming/mapping the data to higher dimensional space, using the inner product to classify the data. SVM's works well when using feature extraction like Principal Component Analysis(PCA) to reduce the amount of features to evaluate.

2.4 K-Nearest Neighbours

K-Nearest Neighbours(KNN) is a easy-to interpret and maybe one of the simplest methods in Machine Learning. The method does exactly as the name suggest; find the k nearest neighbours, where k is a number and labels each data point based on those neighbours. The method is attractive because of it's easy interpretability, fast execution time and often precise classification and regression power. There are many variants of the method where the distance measure and way to handle draws differ. KNN's with $k = 1$ have a special property that the hyperplane of the data points will be partitioned into an equal amount of partitions as there are data points. Each partition consist of all point in the plane closer a specific data point than any other. The amount of neighbours k can have a great impact on the accuracy of the classification. In some cases a low k will be preferable with very separated data, in other case it can lead to overfitting due to higher influence from noise and outliers. Because a even number of categories is classified in this problem the concerns of draws is present when using a even k.This makes a odd k more attractive in this problem.

3 Methods

As mentioned in the introduction different methods in Machine Learning will be used to try to build a model that manage to correctly classify if pictures of handwritten numbers are odd or even. This would be simple if the method should only classify previously seen data, the challenge arise when asked to make a model that is highly accurately in classifying new data. The goal is to make a a model that trains without overfitting on the in-sample data in order to well estimate the out-of-sample data. In the methods below this is done by using different techniques like cross-validation, validation-set and regularization. In all of the methods the data is classified into the numbers $0-9$. Another approach would be to classify the number's directly into either a odd or even category. The reason for using all ten numbers is mainly because this makes the extension of the use purpose a lot smoother, f.ex. to also classify if the number is a prime number or not.

3.1 Tree based methods

3.1.1 Classification tree

The regression tree is implemented with the R-package "tree" which has a function with the same name. In this method the feature being classified is set and which variables it should consider in the splits. When classifying digits many of the pixels will have a low variance nearing the borders, these will most likely not be used in any of the splits. To reduce unnecessary computation, columns with variance close to zero are removed. Other parameters that can be altered is to control when the tree should stop splitting. Here the minimum development is set as stopping metric. This is measured by calculating the reduction in Residual Sum of Squared - RSS given by:

$$RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{2}$$

, where i is a data point in the set of n data points. After a tree has been fitted it might be that the number of terminal nodes are too high and a lower amount of nodes can classify the data as well meanwhile reducing overfitting.s

3.1.2 Random Forest

To create a random forest to classify digits the package "randomForest" is used. The function with the same name as the library is used both for creating the random forest and the bagging. These parameters was used when running the random forest:

`ntree`	`mtry`
500	$\frac{1}{3} \times 784 \approx 261$

Table 1

, where `ntree` is the amount of trees and `mtry` the number of variables used in each tree. The fraction $\frac{1}{3}$ is comes from the default size of the subset of variables in the "randomForest" package. The total number of variables is 784, equal to the amount of pixels in each image.

3.1.3 Bagging

The implementation of bagging or bootstrap aggregation is identical to random forest as described above except for `mtry`. Here the subset of variables is the whole set of variables. This is implemented with the parameters:

`ntree`	`mtry`
500	784

Table 2

. For both bagging and random forest there is no need to specify a form of validation due to the OOB-estimate for the ut-of-sample error as explain in the theory (2.1.2).

3.1.4 Boosting

For the implementation of boosting, the library "gbm" is used. The function "gbm" is very similar in use compared to other tree-functions, but "gbm" has an advantage in being parallelized speeding up big runs. Boosting has the possibility to overfit and therefore the built-in validation metric is set to 10-fold cross-validation. When running boosting these parameter were set to:

`ntree`	`distribution`	`interaction.depth`	`shrinkage`	`cv.folds`	n.cores
10000	`"multinomial"`	2	0.01	10	4

Table 3

. Here the distribution is set to be multinomial in order to make the boosting method handle the 10 categories for the digits. Here it is possible to note if one directly classify digits as odd or even a binomial distribution would possible to use. The interaction depth is set to two in order to keep the trees small with a λ (shrinkage) of 0.01 slowly improving the performance. The last two parameters gives the number of cross-validation folds and cores to use.

3.2 Deep Learning

3.2.1 Artificial Neural Network

For the neural network a "h2o"-grid is run in order to find a good set up for the neural network. "h2o" is a 'R' package originally written in 'Java'. This is a powerful tool which has a deep learning function that suits very well for setting up a neural network. The function runs in parallel and can be used with the grid-function in "h2o". This makes it easier to build and compare several neural nets to find the best parameters for the network of a specific problem. To avoid overfitting 10-fold cross-validation is activated for all the models. The different parameters below were combined producing 40 neural networks:

`Hyper parameter`	`value(s)`
`hidden`	$(100, 100), (150, 150), (540, 320), (100, 100, 100), (540, 320, 100)$
`input_dropout_ratio`	0, 0.2
`epochs`	20
`activation`	`"Rectifier"`, `"RectifierWithDropout"`
`l1`	0, $1.4e-5$

Table 4

. The hidden layers are controlled by the "hidden"-parameter with the number of layers being the length of the vector and the size of each given by the number. "l1" regularization and drop-out in the input layers are given tried to deal with overfitting. To keep the neural network within feasible range 20 epochs were used for each model. The two rectifier activation functions for deep learning in "h2o" were in training the models.

3.2.2 Convolutional Neural Network

To implement a convolutional neural network the package "mxnet" was used. This is originally written in 'C++' and is available in several programming languages. The network is built by piecing together different layers and activation functions. A validation set were used when tweaking to find the best parameters. 15% of the training data was sampled out and used to estimate the out-of-sample error. Parameters for convolutional neural network with two convolutional layers:

Layer	parameters
convolution_1	kernel$= 5 \times 5$, num_filter $= 30$
activation_1	"tanh"
pooling_1	pool_type = "max", kernel $= 2 \times 2$
convolution_2	kernel$= 5 \times 5$, num_filter $= 50$
activation_2	"tanh"
pooling_2	pool_type = "max", kernel $= 2 \times 2$
flatten	
fully_1	num_hidden $= 500$
activation	"Rectifier Linear Unit - relu"
fully_2	num_hidden $= 40$

Table 5

. The full network is constructed of two convolution layers, two pooling layer, three activation function, one flatten layer and two fully connected layers. The network can be trained using either the CPU or a GPU, here the CPU is used due to lack of GPU.

3.3 Support vector machines/PCA

Before using support vector machines with the package "e1071" feature selection was performed on the data to reduce the number of features and thereby the complexity of the classification problem. To extract the features, principal component analysis was applied. PCA takes the original features and combines them making new features that tries to describe as much of the variance in the data as possible. After running PCA on the features the p most important PCA features is selected. PCA returns the most describing features in decreasing importance. Running the SVM with different amounts of PCA features yield different cross-validation error which is used to find the best number of principal components to use with SVM. The parameters used for support vector machines:

Parameter	value(s)
amount PCA	$p \in [1, 2, ..., 30]$
cost	$cost \in [0.5, 1, 1.5, ..., 10]$

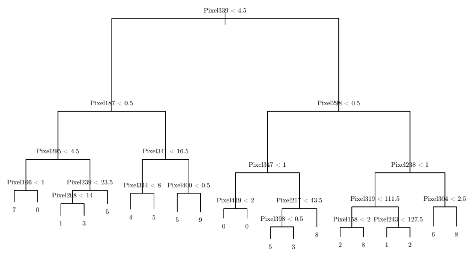
Table 6

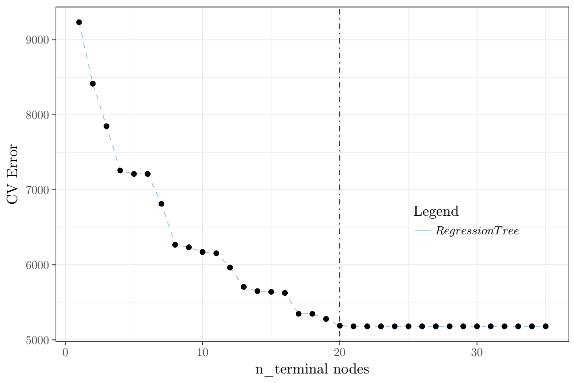
3.4 K-Nearest Neighbours

4 Results

4.1 Tree based methods

4.1.1 Classification tree





(a) E_{CV} for different amounts of leaf nodes.

(b) Regression tree after pruning

Figure 1: A figure with two subfigures

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	31	0	5	1	1	3	4	1	0	1	0.340	16/47
1	0	44	1	2	0	2	3	0	1	0	0.170	9/53
2	0	3	32	2	1	0	0	0	2	0	0.200	8/40
3	0	1	1	26	1	0	1	1	0	3	0.235	8/34
4	0	0	0	0	26	0	10	0	1	1	0.316	12/38
5	4	2	2	12	9	31	6	10	3	7	0.640	55/86
6	0	1	3	1	2	1	23	0	0	2	0.303	10/33
7	1	0	1	2	0	0	2	34	0	5	0.244	11/45
8	0	2	5	3	6	5	6	0	39	3	0.435	30/69
9	3	0	2	2	7	3	2	1	6	27	0.491	26/53
Total	8	9	20	25	27	14	34	13	13	22	0.371	185/498

4.1.2 Random Forest

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	46	0	0	1	0	0	0	0	0	0	0.021	1/47
1	0	42	0	0	0	1	0	1	0	0	0.045	2/44
2	0	2	48	0	0	0	0	1	0	0	0.059	3/51
3	0	0	0	41	0	1	0	0	3	0	0.089	4/45
4	0	1	0	0	52	1	1	4	0	1	0.133	8/60
5	0	0	0	4	0	42	1	0	1	0	0.125	6/48
6	0	0	0	0	1	0	47	0	2	0	0.060	3/50
7	0	1	0	0	0	0	0	48	0	2	0.059	3/51
8	1	2	0	1	0	0	0	0	42	1	0.106	5/47
9	0	0	0	1	2	1	0	0	1	48	0.094	5/53
Total	1	6	0	7	3	4	2	6	7	4	0.081	40/496

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	47	0	0	2	0	3	0	0	0	0	0.096	5/52
1	0	42	0	0	0	0	0	0	0	0	0.000	0/42
2	0	2	46	0	0	0	0	1	0	0	0.061	3/49
3	0	1	0	41	0	2	0	1	2	0	0.128	6/47
4	0	1	0	0	49	1	1	2	0	2	0.125	7/56
5	0	0	0	3	0	38	1	0	1	0	0.116	5/43
6	0	0	1	0	1	1	47	0	2	1	0.113	6/53
7	0	0	0	0	0	0	0	47	0	3	0.060	3/50
8	0	2	1	1	1	0	0	2	43	1	0.157	8/51
9	0	0	0	1	4	1	0	1	1	45	0.151	8/53
Total	0	6	2	7	6	8	2	7	6	7	0.103	51/496

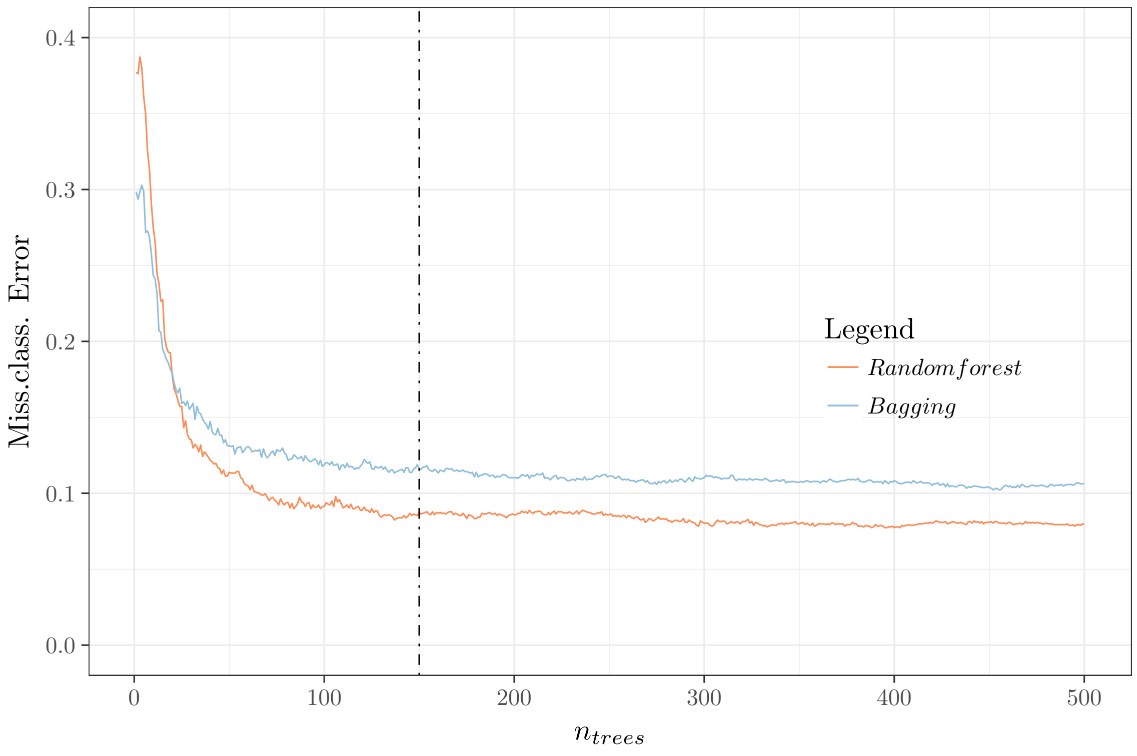


Figure 2

4.2 Neural Networks

4.2.1 Artificial Neural Network

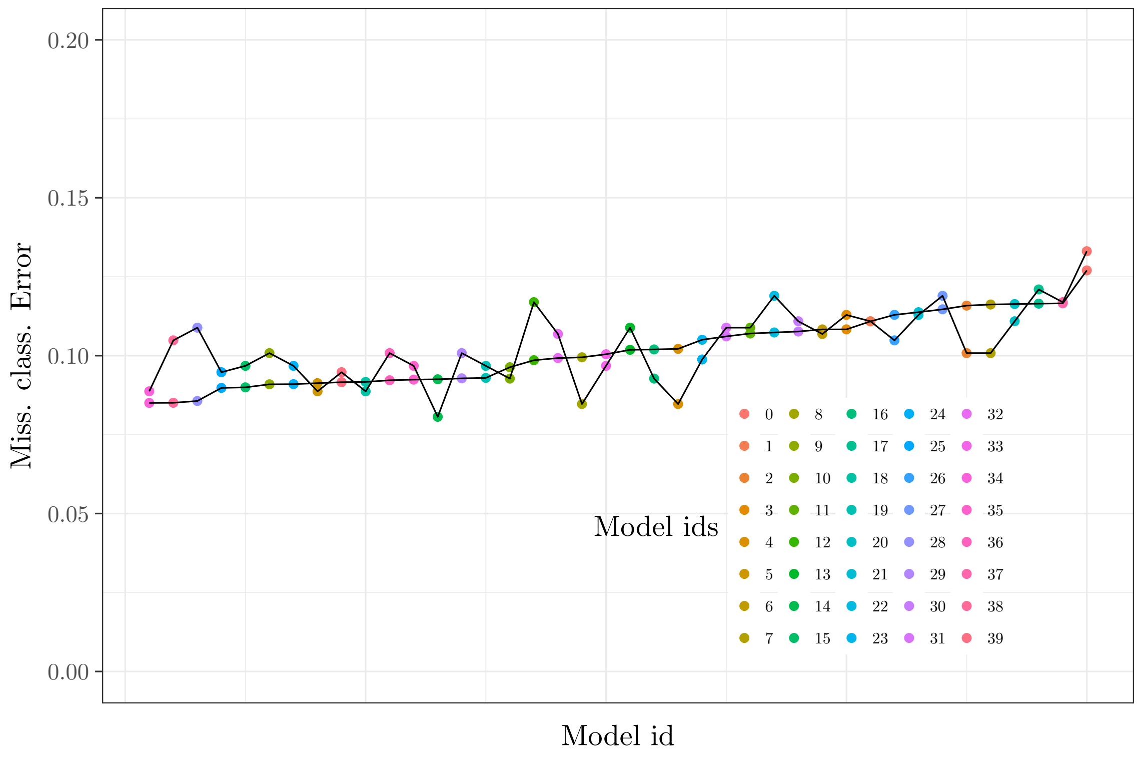


Figure 3

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	42	0	0	0	0	2	0	0	0	0	0.045	2/44
1	0	47	0	0	1	0	0	0	4	0	0.096	5/52
2	2	1	46	2	0	0	1	0	0	0	0.115	6/52
3	0	0	0	42	1	1	0	1	0	0	0.067	3/45
4	0	0	0	0	50	0	0	2	0	2	0.074	4/54
5	1	0	0	2	1	43	0	1	6	1	0.218	12/55
6	1	0	0	1	1	0	48	0	0	0	0.059	3/51
7	1	0	2	0	1	0	0	49	1	3	0.140	8/57
8	0	0	0	1	0	0	0	0	38	0	0.026	1/39
9	0	0	0	0	0	0	0	1	0	46	0.021	1/47
Total	5	1	2	4	5	3	1	5	11	6	0.091	45/496

4.2.2 Convolutional Neural Networks

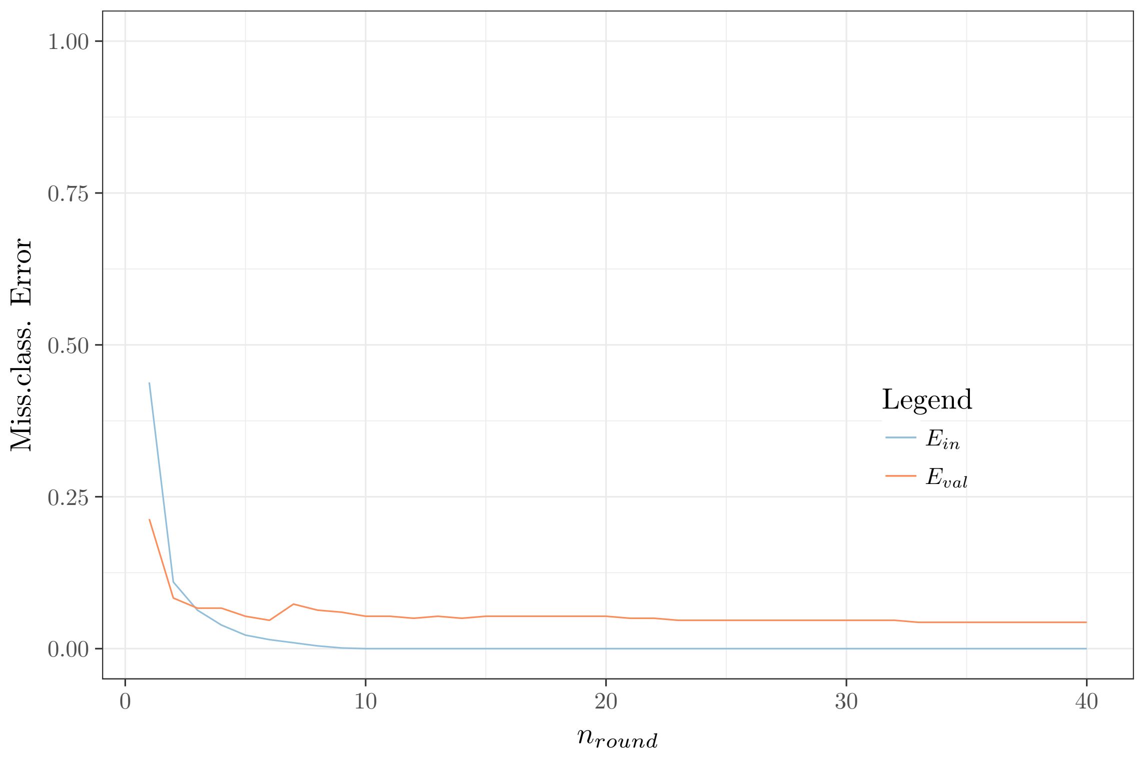
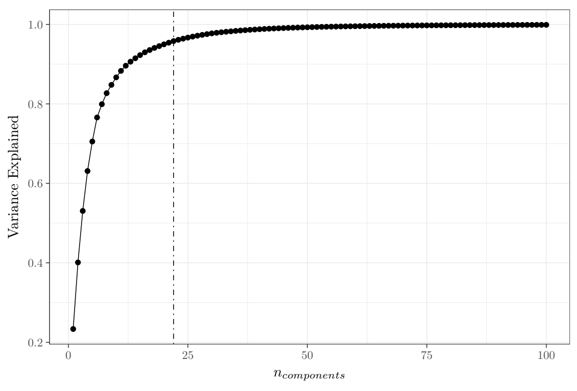


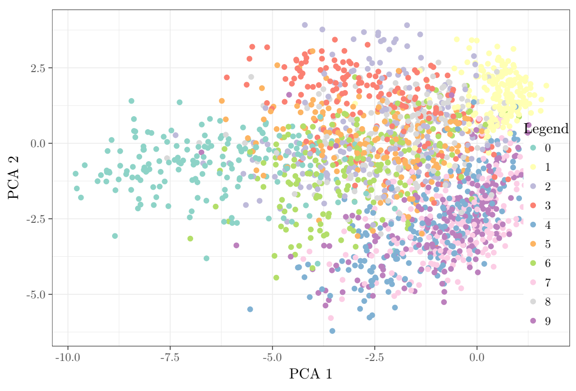
Figure 4

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	47	0	0	0	0	0	1	0	0	0	0.021	1/48
1	0	43	0	0	0	0	0	0	0	0	0.000	0/43
2	0	1	47	0	1	0	0	1	1	0	0.078	4/51
3	0	1	0	48	0	0	0	0	0	0	0.020	1/49
4	0	1	0	0	51	1	0	1	0	0	0.056	3/54
5	0	0	0	0	0	44	1	0	0	0	0.022	1/45
6	0	1	0	0	1	0	47	0	0	0	0.041	2/49
7	0	0	1	0	0	0	0	52	0	1	0.037	2/54
8	0	1	0	0	0	1	0	0	48	0	0.040	2/50
9	0	0	0	0	2	0	0	0	0	51	0.038	2/53
Total	0	5	1	0	4	2	2	2	1	1	0.036	18/496

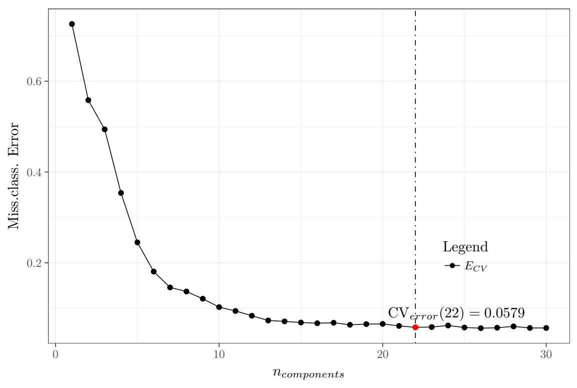
4.3 Support vector machines



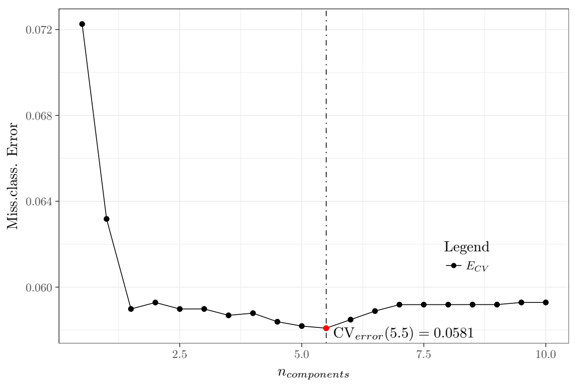
(a)



(b)



(a)



(b)

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	47	0	0	0	0	0	0	0	0	0	0.000	0/47
1	0	48	0	0	0	0	1	0	0	0	0.020	1/49
2	0	0	46	0	0	0	0	0	1	1	0.042	2/48
3	0	0	1	41	0	0	0	1	2	0	0.089	4/45
4	0	0	0	0	53	0	0	2	0	1	0.054	3/56
5	0	0	0	4	0	46	0	0	0	0	0.080	4/50
6	0	0	0	0	1	0	48	0	1	0	0.040	2/50
7	0	0	0	0	0	0	0	49	0	4	0.075	4/53
8	0	0	1	1	0	0	0	1	45	1	0.082	4/49
9	0	0	0	2	1	0	0	1	0	45	0.082	4/49
Total	0	0	2	7	2	0	1	5	4	7	0.056	28/496

4.4 K-Nearest Neighbors

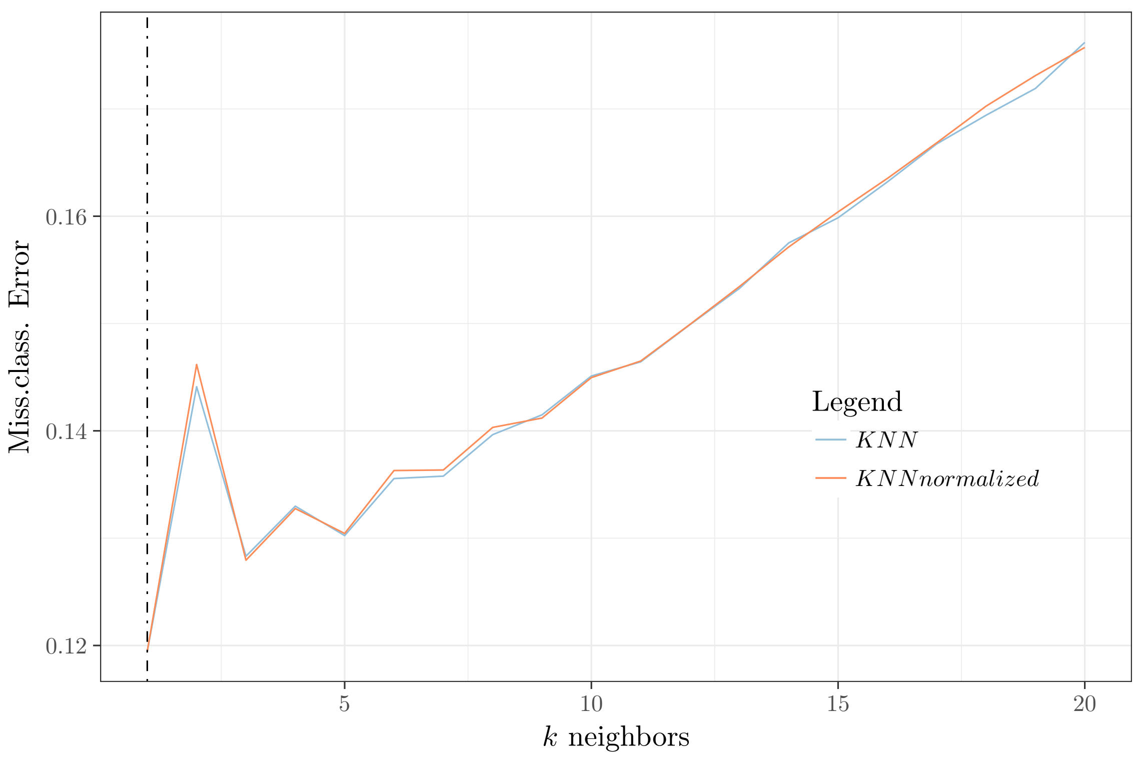


Figure 7

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	45	0	0	1	0	1	0	0	1	0	0.062	3/48
1	0	46	3	0	0	2	0	2	1	0	0.148	8/54
2	0	0	45	0	0	0	0	2	1	0	0.062	3/48
3	0	0	0	40	0	1	0	0	6	0	0.149	7/47
4	0	1	0	0	52	0	0	3	1	4	0.148	9/61
5	1	0	0	4	0	41	1	0	1	0	0.146	7/48
6	1	0	0	0	1	0	48	0	0	0	0.040	2/50
7	0	1	0	0	0	0	0	45	0	1	0.043	2/47
8	0	0	0	3	0	0	0	1	37	0	0.098	4/41
9	0	0	0	0	2	1	0	1	1	47	0.096	5/52
Total	2	2	3	8	3	5	1	9	12	5	0.101	50/496

5 Discussion

6 Appendices