UMEÅ UNIVERSITY
Department of Mathematics
and Mathematical Statistics

November 30, 2022

Statistical learning with high dimensional data

Assignment 1

Supvervisor
Xijia Liu

Christoffer Sjölund: chsj0028@student.umu.se
Hynek: kydlicek.hynek@gmail.com
Sam Adhami: saad0029@student.umu.se

Contents

1 Task 1 - Proceptron Algorithm

1.1 Task description

In this task, we implement perceptron algorithm in R. Task 1.1: Write your own program to implement the algorithm in R Task 1.2: Use the R code 'Task1-DataGeneration.R' to generate the data. Apply your function of perceptron on the data and visualize the decision boundary in a plot. Tips: The perceptron model can be represented as Sign($w^T x$), where $x = (1, x_1, ..., x_p)$, $w = (w_0, w_1, ..., w_p)^T$

1.2 Method

The perceptron algorithm can be used for classification on datasets that can be separated by a line. The line is given by the weights $w = (w_0, w_1, w_2)$ where w_0 is the bias in the dataset, w_1 and w_2 are the coefficients orthogonal to the line. The observations are then checked to see that the each class is on different sides of the line. This is done on each observation using:

$$\hat{y} = Sign(\boldsymbol{w}_m^T \boldsymbol{x})_m \tag{1}$$

If one observation is on the wrong side, the weights are recalculated using:

$$\boldsymbol{w}^t = \boldsymbol{w}^{(t-1)} + y * \boldsymbol{x} \tag{2}$$

The algorithm iterates through all observations, if the weights are updated one time or more, then it will rerun until all observations are correctly classified.

1.3 Results

The resulting weights vector becomes w=(-4.46,-11.37,23.7)

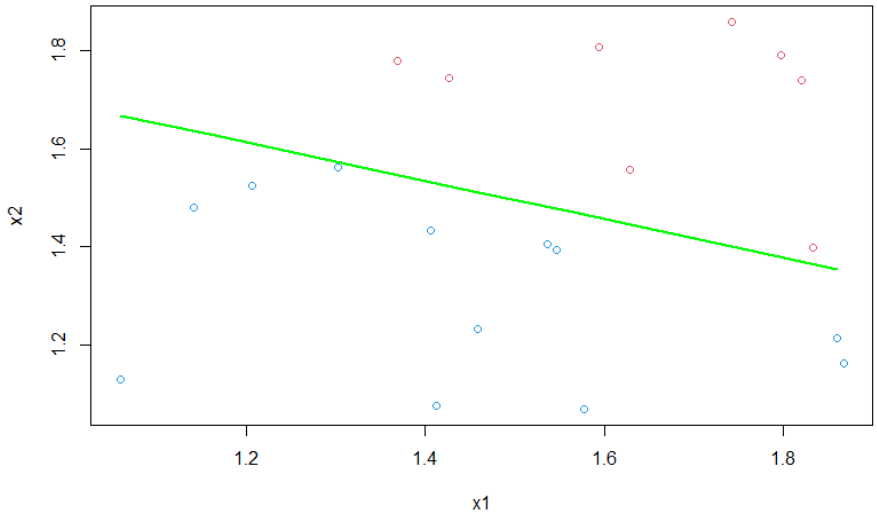


Figure 1: Caption

1.4 Discussion

As can be seen in figure 1, the algorithm correctly classifies each observation. The line generated after each update of the weights moved around a lot but as the final solution was found it stabilized. This could mean that there was a problem with updating the bias as it was running.

2 Task 2 - LASSO Regression

2.1 Task description

In this task, we solve the handwritten digit recognition problem with penalized multinomial regression. Data description: The training data contains 7291 images of handwritten digit. Each digit is presented by a 16 $\times$ 16 pixel image. The data matrix has 257 columns. The first column is the target variable, i.e. the digit presented by the image. The rest 256 columns are the feature variables (pixel values of the image) The data are stored in a R space, 'HWD.RData'. In addition, a R code 'HWD-vis.R', may help you visualize each image and understand the data.

2.2 Task 2.1:

Please set the random seed as 8312 to split the data into training set (80%) and testing set (20%). Train a LASSO penalized multinomial regression and tune the hyper-parameter by using the build-in function 'cv.glmnet'. Summarize your training results and estimate the accuracy and kappa statistic of your final model.

2.2.1 Task 2.2:

Visualize the feature selection results of the optimal LASSO penalized multinomial regression. Since it is a multinomial regression, you will get 10 sets of regression parameters. Each set of regression parameters is corresponding to one digit. It would be interesting to visualize each set as a 16 by 16 image. You may find some reasonable patterns. Some tips and suggestions:
• For multi-classes problem, we need to specify the family of response variable as 'multinomial' in function 'cv.glmnet'.
• 10 folds cross validation is recommended for tuning the shrinkage parameter.

2.3 Method

The data is split into the training and the testing set. The *cv.glmnet* function is used to perform the regression analysis with a 10-fold cross validation. The *predict* function was then used to classify the numbers in the training set. The accuracy is then calculated using:

$$\text{accuracy} = \frac{\text{Number of correctly classified observations}}{\text{Number of observations}} \tag{3}$$

The kappa statistic is then calculated and the numbers estimated from the model will be visualized.

2.4 Results

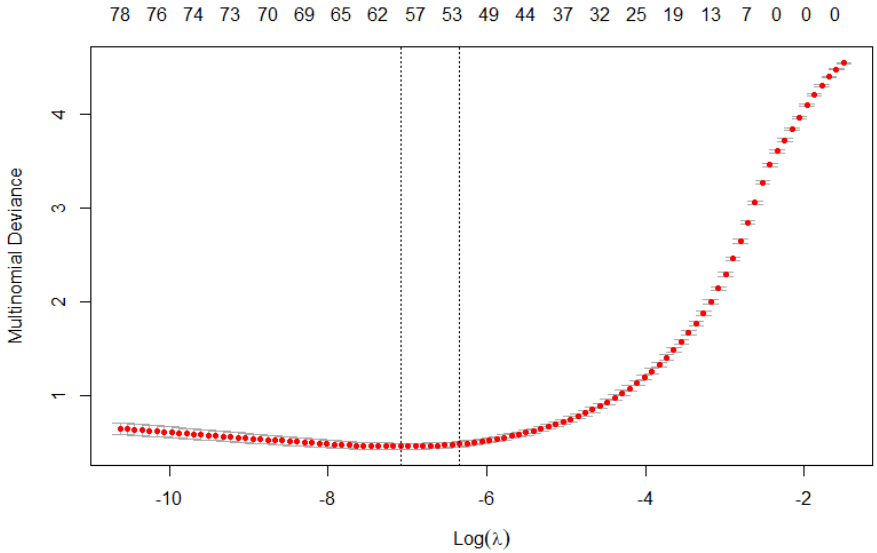


Figure 2: Log lambdas against the models deviance

The accuracy was 0.925 and the kappa static is 0.918.

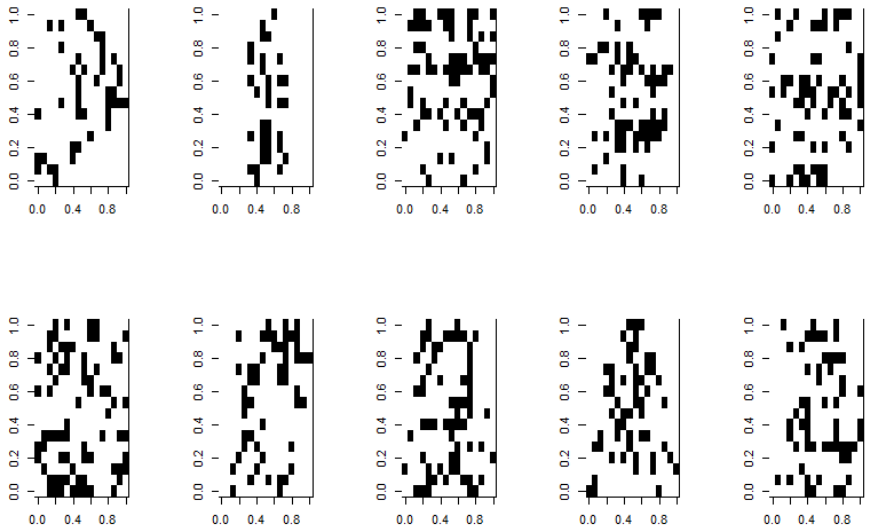


Figure 3: Estimated numbers from the model.

2.5 Discussion

From figure 2 we see the lambda is optimal when it is smaller yet close to 0. It has a high accuracy of 0.925 and kappa of 0.918. This combined with the small deviance shows that the model works fairly well. Figure 3 shows the estimated numbers for the smallest lambda. It shows some similarities to the numbers but it is far from being able clear what number is shown.

3 Task 3 - Kernel PCA

3.1 Task description

The purpose of this task is to implement PCA in R.

3.1.1 Task 3.1

Derive the following formula of calculating the kth kernel principal components for a new observation.

$$\lambda_k^{-1}\mathbf{u}_k^\top(\mathbf{I}-\mathbf{C})((\kappa(\mathbf{x}_1,\mathbf{x}_{new}),...,\kappa(\mathbf{x}_n,\mathbf{x}_{new}))^\top - n^{-1}\mathbf{K}\mathbf{1}_n) \tag{4}$$

where $\mathbf{K}$ is the Gram matrix of the training set, $(\lambda_k, \mathbf{u}_k)$ is the kth eigen pairs of $\mathbf{K}$, $\mathbf{x}_i$ is the ith observation, $i = 1, ..., n$ and $\mathbf{C} = n^{-1}\mathbf{1}_n\mathbf{1}_n^\top$.

3.1.2 Task 3.2

Use the R code 'Task3-KPCA' to generate the data. Write your own program, without using function from any package, to implement KPCA with kernel function $(\mathbf{x}_1^\top\mathbf{x}_2)^2$ over training set and visualize the 3rd kernel principal components in a figure. Calculate the 3rd kernel principal components for the testing set and add two points to the figure.

3.2 Method

The inputs of the algorithm should be the observations $x_i \in \mathbb{R}^p$ for $i = 1, ..., n$ and the kernel function κ.

1. Calculate the non-centralized Gram matrix $\mathbf{K}$ and centralized Gram matrix $\mathbf{K}^*$.
2. Eigen-decomposition on $\mathbf{K}^*$ to obtain all the eigen pairs $(\lambda_k, \mathbf{u}_k)$ where $k = 1, ..., n$.
3. For the observation within the sample, the kth PC is $\mathbf{u}_k$.
4. For a new observation, $\mathbf{x}_{new}$, kth PC is $\lambda_k^{-1}\mathbf{u}_k^\top(\mathbf{I}-\mathbf{C})((\kappa(\mathbf{x}_1,\mathbf{x}_{new}),...,\kappa(\mathbf{x}_n,\mathbf{x}_{new}))^\top - n^{-1}\mathbf{K}\mathbf{1}_n)$.

3.3 Results

3.3.1 Task 3.1

The normalized eigen vector is

$$e = \lambda^{-1}\Phi^{*\top}\mathbf{u} \longrightarrow e^\top = \lambda^{-1}\mathbf{u}^\top\Phi^* \tag{5}$$

The centralized augmented data matrix is

$$\Phi^* = (\Phi - n^{-1}\mathbf{1}_n\mathbf{1}_n^\top\Phi) = (\mathbf{I} - n^{-1}\mathbf{1}_n\mathbf{1}_n^\top)\Phi = (\mathbf{I} - \mathbf{C})\Phi \tag{6}$$

We can then use equation 6 in equation 5 and get the following

$$e^\top = \lambda^{-1}\mathbf{u}^\top(\mathbf{I} - \mathbf{C})\Phi \tag{7}$$

The augmented data matrix is

$$\Phi = (\phi(\mathbf{x}_1)^\top, ..., \phi(\mathbf{x}_n)^\top)^\top \tag{8}$$

We then have that

$$\Phi\phi(\mathbf{x}_{new}) = (\phi(\mathbf{x}_1)^\top, ..., \phi(\mathbf{x}_n)^\top)^\top\phi(\mathbf{x}_{new}) =$$

$$(\phi^\top(\mathbf{x}_1)\phi(\mathbf{x}_{new}), ..., (\phi^\top(\mathbf{x}_n)\phi(\mathbf{x}_{new}))^\top = (\kappa(\mathbf{x}_1, \mathbf{x}_{new}), ..., \kappa(\mathbf{x}_n, \mathbf{x}_{new}))^\top \tag{9}$$

The Gram matrix is defined as

$$\mathbf{K} = \Phi\Phi^\top \tag{10}$$

To calculate the PC for a new observation we have

$$e^\top(\phi(\mathbf{x}_{new}) - n^{-1}\Phi^\top\mathbf{1}_n) \tag{11}$$

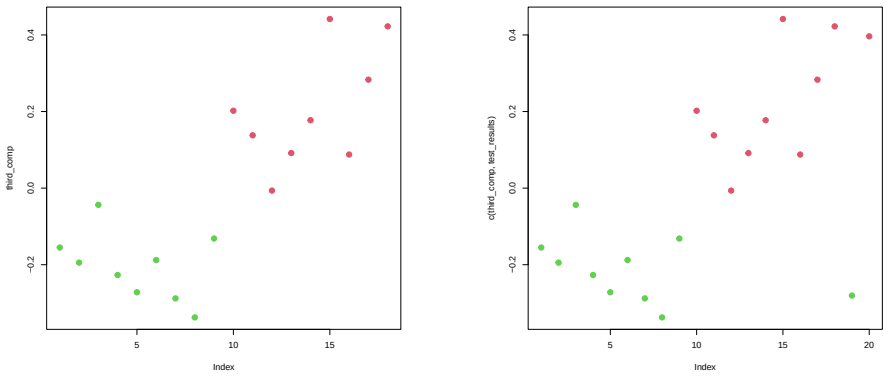
$n^{-1}\Phi\mathbf{1}_n$ is the sample mean vector in the augmented feature space.
Now that we have everything we need, we can use equation 7 and put it in equation 11

$$\lambda^{-1}\mathbf{u}^\top(\mathbf{I} - \mathbf{C})\Phi\phi(\mathbf{x}_{new}) - \lambda^{-1}\mathbf{u}^\top(\mathbf{I} - \mathbf{C})\Phi n^{-1}\Phi^\top\mathbf{1}_n \tag{12}$$

With a little bit of rearrangement and using equation 9 and equation 10 we get the final result

$$\lambda^{-1}\mathbf{u}^\top(\mathbf{I} - \mathbf{C})((\kappa(\mathbf{x}_1, \mathbf{x}_{new}), ..., \kappa(\mathbf{x}_n, \mathbf{x}_{new}))^\top - n^{-1}\mathbf{K}\mathbf{1}_n) \tag{13}$$

3.3.2 Task 3.2



(a) Without testing set.

(b) With testing set.

Figure 4: Plots of the 3rd kernel components with and without the testing set.

3.4 Discussion

Kernel PCA belongs to the memory based methods and takes a lot of computational cost. Even though it is cheaper in the training stage, we still have to pay for the computation later on in the algorithm. We can not say that KPCA is better than PCA or the other way around, it all depends on the problem. The good thing with KPCA is that we can capture nonlinear structures in the data, where PCA do not. KPCA requires choosing hyperparameters to the kernel function, optimal ones as well, whereas PCA do not. Though it is problem dependent.

4 Task 4 - Kernel Ridge Regression

4.1 Task description

The purpose of this task is to implement kernel (RBF kernel) ridge regression and train a predictive model on 'Boston data'.

4.1.1 Task 4.1

Implement kernel ridge regression (KRR) in R. Write a R function called 'predictKRR'. The following inputs should be included:

- `trainX` - A data matrix which contains feature variables for training KRR.
- `trainY` - A numeric vector that contains the target variable for training a KRR.
- `X` - A data matrix that contains the feature variables for prediction.
- `lambda` - A scalar to specify the shrinkage parameter in KRR.
- `sigma` - A scalar to specify the parameter in RBF kernel function.

The output of function 'predictKRR' should be the predicted value of the target variable given the input features variable 'X'.

4.1.2 Task 4.2

Train a regular regression model in the 'Boston data' as the baseline model. The last column, 'medv', is the target variable and the rest are feature variables. Estimate the performance of the regress model on the testing data. The performance of the model should be quantified by square root mean square errors.

4.1.3 Task 4.3

Train KRR with RBF kernel function on the 'Boston data'. Same as before, 'medv' is still the target variable. Train the KRR with the Gaussian kernel function to predict 'medv' by using other variables as inpts. The recommended hyper parameters for λ and σ are $(0.1, 0.01, 0.001)^\top$. A 10-fold cross-validation on the train data set should be runned. Root mean square error can be used as a metric of performance. And finally, an estimation of the performance of the final model should be done and compared with the result of Task 4.2.

4.2 Method

The Kernel Ridge Regression can be summarized in the following algorithm. The inputs of the algorithm should be the observations $(y_i, \mathbf{x}_i) \in \mathbb{R}^{p+1}$ for $i = 1, ..., n$ and the kernel function κ.

1. Centralize and normalize the feature matrix $\mathbf{X} = (\mathbf{x}_1^\top, ..., \mathbf{x}_n^\top)^\top$.
2. Calculate the Gram matrix $\mathbf{K}$.
3. Estimate the auxiliary vector $\mathbf{a}$.
4. Do prediction using $\mathbf{y}^\top(\mathbf{K} + \lambda\mathbf{I})^{-1}((\kappa(\mathbf{x}_1, \mathbf{x}_{new}), ..., \kappa(\mathbf{x}_n, \mathbf{x}_{new}))^\top$.

4.3 Results

Model	Test RMSE	Train RMSE	Best sigma	Best lambda
Linear model	5.256	4.527	-	-
KR Regression	3.572	1.694	0.316	0.01

4.4 Discussion

The kernel based methods belongs to the memory based methods. We need to pay extra computational cost because of the normalization and generalization of the matrices, though it is dependent on the size of the data. If we compare it to the regression model, the result from Task 4.2, we can see that the root mean squared error is twice as big as the result from Task 4.3 which is a kernel based regression model for the prediction. This may be because of that ridge regression does not include all the predictors in the final model or that it shrinks its coefficients towards zero, because we restrict the capacitance of the model this means that the model generalizes better. Or it could be that the kernel ridge regression uses kernel trick to go non-linear and therefore gives us a better result.

5 Task 5 - SVM

5.1 Task description

5.1.1 Task 5.1 Vanilladot kernel

Train a SVM with 'vanilladot' kernel function with a non zero slackness parameter C, i.e. linear soft margin classifier. We apply 10-fold cross validation to tune the hyper parame- ter C. The slackness parameter C can be proposed as (0.0001,0.0005,0.001,0.005,0.1,1). In the end, pick up the 'optimal' value for C, train the final model and estimate the performance by using the testing set.

5.1.2 Task 5.2 Rbfdot kernel

Train a kernel SVM with 'rbfdot' kernel function. Optimal you final model by tuning the hyper-parameters, i.e. slackness parameter C and kernel parameter sigma. Estimate the performance of your final model by using the testing set.

5.2 Method

5.2.1 MMC

To understand the SVM method, we first need to understand Maximum Margin Classifier(MMC). The MMC problem can be formulated as:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}||w||^2 \\ &s.t. y_i(w^T x_i + b) \geq 1, i = 1, 2, ..., N \end{aligned} \tag{14}$$

It's possible to prove that this is equivalent to:

$$\begin{aligned} &\max_{a_i} \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j x_i^T x_j \\ &s.t. \sum_{i=1}^{N} a_i y_i = 0 \text{ and } a_i \geq 0, i = 1, \ldots, N \end{aligned} \tag{15}$$

This is a quadratic programming problem, which can be solved. It's then possible to estimate w and b by:

$$\begin{aligned} w &= \sum_{i=1}^{N} a_i y_i x_i \\ b &= y_k - w^T x_k \end{aligned} \tag{16}$$

Yielding the following function for classification:

$$\begin{aligned} y_{new} &= Sign(w^T x_{new} + b) \\ y_{new} &= Sign(\sum_{i=1}^{N} a_i y_i x_i^T x_{new} + b) \end{aligned} \tag{17}$$

As we can see the kernelization can be applied rather easy due to structure. This means simply replacing $x_i^T x_j$ with $K(x_i, x_j)$, where K is a kernel function.

5.3 SVM

To get the SVM we need to add a slackness component to the kernelized MMC problem. This means to reformulate the MMC problem as:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i \\ &s.t. y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, 2, ..., N \end{aligned} \tag{18}$$

The slackness parameter C regulates how much we want to penalize for missclassification. The larger C is, the more we want to penalize for missclassification. The smaller C is, the more we want to allow missclassification.

5.3.1 Multiclass classification

As we can see the base SVM can only solve the task of binary classification. To extend it to multi-class we used the one-vs-all approach. This means that we train N binary classifiers, where N is the number of classes. Then we classify the new data point to the class with the highest score.

5.4 Hyper-paremeters

We used the proposed hyper-parameters (0.0001, 0.0005, 0.001, 0.005, 0.1, 1).

5.5 Proposed values from hyperparameters

We used $(0.0001, 0.0005, 0.001, 0.005, 0.1, 1)$ for C. For σ we let the ksvm library find the optimal value.

5.6 Results

Kernel	Test Accuracy	Train Accuracy	Best C	Best sigma
vanilla-dot	0.95	0.980	0.005	-
rbfdot	0.945	0.989	1	0.00273

5.7 Discussion

We got very good accuracy on both sets. This suggests that both models generalize fairly well on unseen data. Both models had similar accuracy suggesting that for this task the choice of kernel is not really important. That is surprising as the complex rbfdot projects to infinitely many dimensions while vanilla dot is simple projection to 3 dimensions.

6 Task 6 - Random Forest Algorithm

6.1 Task description

6.1.1 6.1 Dataset Review

Review Titanic dataset solution and summarize the key steps and results in your report.

6.1.2 6.2 Random Forest

Based on the processed data and extracted features, train your random forest model. Analysis the output of feature importance.

6.1.3 6.3 Misclassification error rate

Based on the optimal random forest model and the training dataset, calculate the average misclassification error rate of individual trees and the misclassification error rate of the random forest model.

6.2 Method

6.2.1 Data preparation

First the dataset is augment by adding Title, Surname, Name, Family Size and Family Variables. The Family Size is then discretized into 3 values based on size. From data it can be observed that singletons and large families have survivability penalty.

Then the Cabin column is inspected but due to sparseness it's not very helpful.

Next the embarked values are imputed for Passenger 62 and 830. It's observed that the median fare for first class departing from Charbourg(C) coincides with the passengers without embarkment. Thus the C value is imputed.

The Fare is then imputed to passenger 1044. The median fare for first class passengers departing from Southampton(S) is used.

The mice package is then used to impute missing data based on these features.

PassengerId, Name, Ticket, Cabin, Family, Surname, Survived

The age is then imputed based on the mice model.
Finally the Mother and Child variables are added.

6.2.2 Decision Trees

Decision trees is a supervised learning algorithm, which can be used for both classification and regression. The algorithm is based on the idea of splitting the data into smaller and smaller subsets based on a certain feature. The splitting

is done in a way, that the subsets are as pure as possible. This condition of splitting can be formulated as:

$$\begin{aligned} &\min_j \min_s \left(\sum_{x \in R_1(j,s)} (y_i - c_1)^2 + \sum_{x \in R_2(j,s)} (y_i - c_2)^2 \right) \\ &\text{where} \quad R_1(j,s) = \{x | \wedge x_i \leq s\} \\ &\text{and} \quad R_2(j,s) = \{x | \wedge x_i > s\} \\ &\text{and} \quad c_i = ave(y_i | x_i \in R_k(i,s)) \end{aligned} \tag{19}$$

The splitting is done recursively until the stopping criteria is met. The most common stopping criteria is either a maximum depth of the tree or a minimum number of samples in a leaf node. The prediction is then done by traversing the tree and predicting the class of the leaf node, which the sample ends up in.

6.2.3 Random Forest

Random forest is an Ensamble method. In it's core it creates n decision trees, where every tree can only use a subset of features and training data. The predicted value is then the average of the predicted values of all the trees(or most common prediction).

6.3 Results

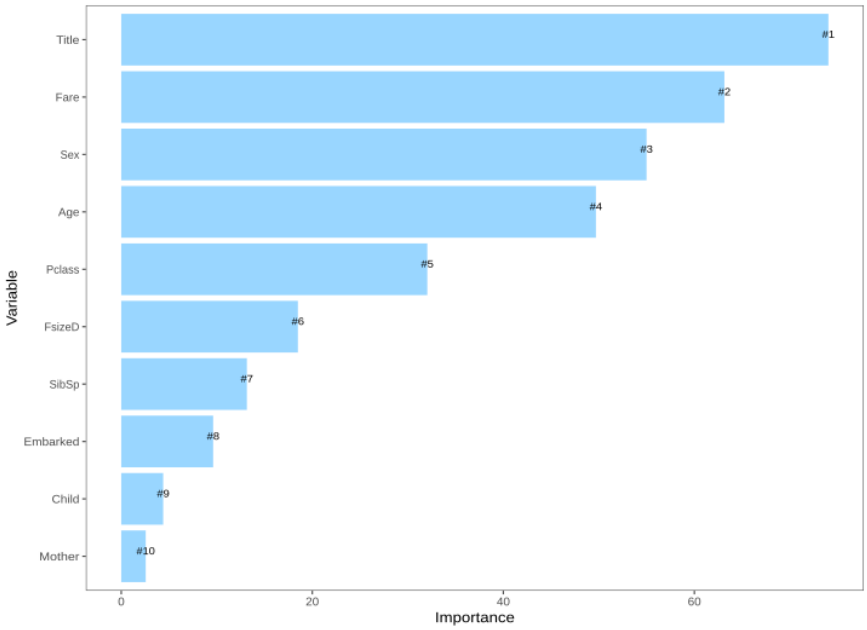


Figure 5: Feature Importance

6.3.1 Feature importance

The feature importance can be observed at Fig 5.

6.3.2 Error rate

Average misclassification rate of individual trees is 0.1357. The misclassification rate of the random forest model is 0.076

6.4 Discussion

6.4.1 Feature importance

We found the Title and Fare to be the most important features. While mother and child was the least important. Since there is a high corelation between Title and Sex, it's not surprising they are so not that far away in terms of importance. The higher importance of Title is probably due to Title being more informative than just a Sex.

6.4.2 Error rate

We found the ensamble model to have smaller err.rate than average err. rate of the individual trees. It should not be surprising because it follows the results of Theory of Uniform Aggregation which states that:

$$Avg(Err(f_j(x))) \geq Err(Avg(f_j(x)))$$