**Report on the group project for the course of Machine Learning for Big Data**

**Insurance Expenses Prediction**

*Mohamad Abou Hamdan*

*Aleksandr Mikhailovitch Molodets*

# Introduction

Today, the insurance industry is massive. It protects hundreds of millions against different types of risk. Such include health insurance, law insurance, the risk of dying early, or even the risk of living longer than expected. Insurance premiums make up a significant part of GDP in many first world countries.

With the use of Machine-Learning, Health insurance companies can utilize data in a more efficient way. This aspect can help profitability in these companies through providing them with resources that aid in a more informative decision-making process.

In this application of Machine-Learning we attempt to inspect different features such as age, social, physical, and locational attributes across a database against their existing medical expense. This would be used for predicting future medical expenses of individuals and, in turn, better insight into what to charge pertaining to the premiums.

Given the present dataset on the behavioral and personal characteristics of the customers, we set out to build a Machine-Learning model best suited to estimate future medical expenditures.

A white rectangular box with black text

Description automatically generatedThe data-set included 1338 observations of 8 variables, separated into dependent (Expenses) and independent variables ( the rest) seen below:

1338 observations of 8 variables

**Factor variables: sex, smoker, region**

Creating dummy variables

7IV’s and 1 DV

No N/A values

A white rectangular box with black text and orange letters

Description automatically generated

# Logic of the code

## Preliminary Actions:

We introduced “Caret” Library into R and installed the packages.

install.packages("Caret")

install.packages("NeuralNetTools")

library(caret)

library(NeuralNetTools)

## Data Inspection & Preparation

Before going into the process of dealing with the given database, a sincere data inspection was to be made. This inspection helps to identify and, in turn, address the factors pertaining to the quality of the data, inaccuracies, outliers, and missing data. This would be essential prior to any form of analytics.

### To check for dimensions:

Using dim, head and str functions

> dim(mydata\_ori)

[1] 1338 7

> head(mydata\_ori)

age sex bmi children smoker region expenses

1 19 female 27.9 0 yes southwest 16884.92

2 18 male 33.8 1 no southeast 1725.55

3 28 male 33.0 3 no southeast 4449.46

4 33 male 22.7 0 no northwest 21984.47

5 32 male 28.9 0 no northwest 3866.86

6 31 female 25.7 0 no southeast 3756.62

> str(mydata\_ori)

'data.frame': 1338 obs. of 7 variables:

$ age : int 19 18 28 33 32 31 46 37 37 60 ...

$ sex : chr "female" "male" "male" "male" ...

$ bmi : num 27.9 33.8 33 22.7 28.9 25.7 33.4 27.7 29.8 25.8 ...

$ children: int 0 1 3 0 0 0 1 3 2 0 ...

$ smoker : chr "yes" "no" "no" "no" ...

$ region : chr "southwest" "southeast" "southeast" "northwest" ...

$ expenses: num 16885 1726 4449 21984 3867 ...

>

### To check for missing values

> colSums(is.na(mydata\_ori))

age sex bmi children smoker region expenses

0 0 0 0 0 0 0

There are no missing values

### To print the different possible variables of sex, smoker and region

> table(mydata\_ori$sex)

female male

662 676

> table(mydata\_ori$smoker)

no yes

1064 274

> table(mydata\_ori$region)

northeast northwest southeast southwest

324 325 364 325

### Checking for outliers of non-factor variables

To check for outliers of non-factor (non-dummy) variables, we drew up box plots for such variables as BMI (Body Mass Index), age and children.

#### For IV BMI:

The metric that was used can be seen as follows according to recognized values:

* Underweight : below 18.5
* Healthy Weight: 18.5 – 24.9
* Overweight: 25.0 – 29.9
* Obesity: 30.0 and above

>boxplot(mydata\_ori$bmi,

horizontal = FALSE,

lwd = 1,

col = "darkblue",

xlab = "",

ylab = "",

main = "bmi",

notch = FALSE,

border = "grey",

outpch = 22,

outbg = "orange",

whiskcol = "blue",

whisklty = 2,

lty = 1)

A diagram of a graph

Description automatically generated

> mean(mydata\_ori$bmi)

[1] 30.66547

> median(mydata\_ori$bmi)

[1] 30.4

> max(mydata\_ori$bmi)

[1] 53.1

Accordingly, the results show an obese mean (30.0 and above). However, this should not be taken as a worrying sign of low quality or inaccuracy in our data as according to the [National Library of Medicine](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10213181/#:~:text=Between%20the%20early%201970s%20and,30.0%20(logged%20%3D%203.4).) the average BMI in the United States is 30.0 which is classified as obese.

#### For IV Age:

No significant outliers found

>boxplot(mydata\_ori$age,

horizontal = FALSE,

lwd = 1,

col = "darkblue",

xlab = "",

ylab = "",

main = "Age",

notch = FALSE,

border = "grey",

outpch = 22,

outbg = "orange",

whiskcol = "blue",

whisklty = 2,

lty = 1)

A blue square with white text

Description automatically generated

> mean(mydata\_ori$age)

[1] 39.20703

> median(mydata\_ori$age)

[1] 39

> max(mydata\_ori$age)

[1] 64

> min(mydata\_ori$age)

[1] 18

#### For IV “children” (how many children)

No significant outliers found

>boxplot(mydata\_ori$children,

horizontal = FALSE,

lwd = 1,

col = "darkblue",

xlab = "",

ylab = "",

main = "Children",

notch = FALSE,

border = "grey",

outpch = 22,

outbg = "orange",

whiskcol = "blue",

whisklty = 2,

lty = 1)

A graph with blue squares and lines

Description automatically generated

> mean(mydata\_ori$children)

[1] 1.094918

> median(mydata\_ori$children)

[1] 1

> max(mydata\_ori$children)

[1] 5

> min(mydata\_ori$children)

[1] 0

#### For expenses

Outliers in DV are expected and due to be explained

>boxplot(mydata\_ori$expenses,

horizontal = FALSE,

lwd = 1,

col = "darkblue",

xlab = "",

ylab = "",

main = "expenses",

notch = FALSE,

border = "grey",

outpch = 22,

outbg = "orange",

whiskcol = "blue",

whisklty = 2,

lty = 1)

A graph with blue and yellow lines

Description automatically generated

> mean(mydata\_ori$expenses)

[1] 13270.42

> median(mydata\_ori$expenses)

[1] 9382.03

> max(mydata\_ori$expenses)

[1] 63770.43

> min(mydata\_ori$expenses)

[1] 1121.87

In summary, the database being void of outliers suggests that the data is consistent in a way where it doesn’t diverge away from the overall pattern. Since outliers can disrupt statistical analyses or our machine learning model, this amounted to more reliable data. In addition, one can assume that the data is homogenous.

### Data prepairation (dummy variables and scaling)

Creating dummy variables, adding fullRank method to remove excessive dummies

> library(caret)

> mydata <- mydata\_ori

> dmy <- dummyVars(" ~ .", data = mydata, fullRank=T)

> mydata\_v1 <- data.frame(predict(dmy, newdata = mydata))

Scaling of the data was mandatory because the inconsistency in scale amongst the independent variables. To render the features’ contributions in an equal manner a scaling of the dataset to prevent larger magnitudes and other variables to undermine the learning process in relation to other variables was made.

> mydata\_v2<-preProcess(mydata\_v1,method="range")

> mydata\_v3<-predict(mydata\_v2,mydata\_v1)

### Correlation Check to account for multicollinearity:

Correlation between the independent variables creates challenges, particularly when trying to isolate the intrinsic effects of a particular variable. In addition to that, this would help guide our decision-making in preprocessing the data. It would have been convenient to merge the variables with high correlation. In this case we considered visualizing these correlation to be sufficient for our preprocessing of the data.

The top-5 correlations between independent variables were:

* BMI/Age (0.107457087) \*figure
* BMI/Southeast (0.269927945) as seen in the figure (could be explainable by regional cuisine; more research needed for such conclusions)
* all the regions with each other (0.345395934)

Firstly, the correlation matrix is being presented:

>corr.matrix <- cor(mydata\_v3)

>View(corr.matrix)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | age | sexmale | bmi | children | smokeryes | regionnorthwest | regionsoutheast | regionsouthwest | expenses |
| age | 1 | -0.0209 | 0.1093 | 0.0424 | -0.025 | -0.0004 | -0.0116 | 0.01 | 0.2990 |
| sexmale | -0.0209 | 1 | 0.0464 | 0.0171 | 0.0761 | -0.0111 | 0.0171 | -0.0042 | 0.0573 |
| bmi | 0.1093 | 0.0464 | 1 | 0.0126 | 0.0039 | -0.1359 | 0.2701 | -0.0064 | 0.1986 |
| children | 0.0425 | 0.0172 | 0.0126 | 1 | 0.0076 | 0.0248 | -0.0231 | 0.022 | 0.068 |
| smokeryes | -0.025 | 0.0762 | 0.004 | 0.0076 | 1 | -0.0369 | 0.0685 | -0.0369 | 0.7873 |
| regionnorthwest | -0.0004 | -0.0111 | -0.1359 | 0.0248 | -0.0369 | 1 | -0.3463 | -0.3208 | -0.0399 |
| regionsoutheast | -0.0116 | 0.0171 | 0.2701 | -0.0230 | 0.0684 | -0.3462 | 1 | -0.3463 | 0.074 |
| regionsouthwest | 0.0100 | -0.0041 | -0.0063 | 0.0219 | -0.0369 | -0.3208 | -0.3462 | 1 | -0.0432 |
| expenses | 0.299 | 0.0573 | 0.1986 | 0.068 | 0.7873 | -0.0399 | 0.074 | -0.0432 | 1 |

Better visualizing the correlation between age and bmi using density-based ggplot with correlation line:

>ggplot(mydata\_v1, aes(x=age, y=bmi) ) + geom\_hex(bins = 20) + scale\_fill\_continuous(type = "viridis") + >theme\_bw() + geom\_smooth(method = "lm", se = FALSE,col="orange")

A graph of a number of people

Description automatically generated with medium confidence

Visualizing binary variable correlations using boxplot function:

ggplot(mydata\_v3) + geom\_bar(aes(regionsoutheast, bmi, fill = regionsoutheast), position = "dodge", stat = "summary", fun = "mean")

A graph of a bar chart

Description automatically generated with medium confidence

### Partitioning the data and calculating default prediction and RSS:

Training our model and, in turn, testing it on another would allow us to see how it asseses new unseen data. Thus, we split our data into an 80%-20% Train/Test

>dim(mydata\_v3)

>head(mydata\_v3)

>str(mydata\_v3)

>set.seed(123)

>index <- sample(1:nrow(mydata\_v3), nrow(mydata\_v3)\*0.8, replace=F)

>index

>mydata\_v3.train <- mydata\_v3[index,]

>mydata\_v3.test <- mydata\_v3[-index,]

>summary(mydata\_v3.train)

>summary(mydata\_v3.test)

To calculate the default prediction for the dependent variable:

> default.pred <- mean(mydata\_v3.train$expenses)

> default.pred

[1] 0.1987984

RSS (The residual sum of squares) for training and training set

> default.train.rss <- sum((mydata\_v3.train$expenses-default.pred)^2)

> default.train.rss

[1] 41.37523

> default.test.rss <- sum((mydata\_v3.test$expenses-default.pred)^2)

> default.test.rss

[1] 8.613895

## Cross-validation and repeats

We set out to configure the settings of the training control of the model using the Caret “trainControl” function to set up cross validation for model training. In this case we performed repeated cross validation, “repeatedcv”. The data is divided into 10 parts, with 10 folds for each iteration. I.e. the dataset is trained and tested 10 times. 2 repetition because when choosing the tune grid for neural networks the RMSE and R-squared started to diverge between tune grid. In addition, this was used to achieve robustness in our model’s performance.

>TControl <- trainControl(method="repeatedcv", number=10,repeats = 2)

We used repeatedcv parameter with 2 repetitions because when choosing the tunegrid for nnet (neural network with 1 hidden layer) the RMSE and Rsquared started to diverge between tunegrid sizes

## Models

We set out to perform different models as they are based on different approaches. Exploring these differences would give us insight into the possible disadvantages and capture patterns, thus increasing the overall predictive performance. We set explore how these models have different capabilities in detecting non/linearity in relationships.

### OLS Model

Even though the OLS is universally understandable method, the assumption of linearity between the independent and dependent variable would create biased estimates.

>set.seed(123)

>ols <- train(expenses ~ ., data=mydata\_v3.train, method="lm", trControl=TControl, metric="Rsquared")

>ols

>summary(ols)

>ols$finalModel #Inspecting final model

>prediction.train <- predict(ols, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>ols.r2 <- 1.0-train.rss/default.train.rss

>ols.r2

>prediction.test <- predict(ols, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>ols.pseudoR2 <- 1.0-test.rss/default.test.rss

>ols.pseudoR2

>report <- rbind(report, data.frame(Model="Linear Regression", R2.Train=ols.r2, R2.Test=ols.pseudoR2))

In addition, OLS allows to see the coefficients values (without the interaction terms) to define the most influential IV’s: see at the table below: OLS allows to achieve pseudo-R2 of 0.7367 on the test set

|  |  |
| --- | --- |
|  | **Values of coefficients** |
| (Intercept) | -0.049353 (0.010757) \*\*\* |
| age | 0.178726 (0.010023) \*\*\* |
| sexmale | -0.00346 (0.006042) |
| bmi | 0.210286 (0.019317) \*\*\* |
| children | 0.044712 (0.012472) \*\*\* |
| smokeryes | 0.385820 (0.007342) \*\*\* |
| regionnorthwest | -0.00889 (0.008613) |
| regionsoutheast | -0.013671 (0.008647) |
| regionsouthwest | -0.015437 (0.008651) |

Pseudo R2 for test set:

0.7367

### Ridge regression

To address multicollinearity, the Ridge regression utilizes a regularization term that would lessen the effects of multicollinearity. In this case we decided against using the cycle for alpha adjustment (no L1 regularization), because the results seem to be diverging with every increase. The lambda is the regularization term added to the linear regression function. The sequence starts from 0.0 and increases towards 0.05 with 0.001 increments.

The table below shows the tuning process for the hyperparameters, in order to find the best set-up. The resulting Lambda is equal to 0.01.

|  |  |  |  |
| --- | --- | --- | --- |
| **Lambda** | **RMSE** | **Rsquared** | **MAE** |
| 0 | 0.09908424 | 0.749312 | 0.07016327 |
| **0.01** | **0.09908424** | **0.749312** | **0.07016327** |
| 0.02 | 0.09954816 | 0.7492457 | 0.07081842 |
| 0.03 | 0.10084844 | 0.7490722 | 0.07248005 |
| 0.04 | 0.10242469 | 0.7488739 | 0.07421301 |
| 0.05 | 0.10417994 | 0.7486586 | 0.07595175 |

>ridgeGrid <- expand.grid(alpha=0.00, lambda=seq(from=0.0, to=0.05, by=0.001))

>set.seed(123)

>ridge <- train(expenses ~ ., data=mydata\_v3.train, method="glmnet", tuneGrid=ridgeGrid, trControl=TControl, metric="Rsquared")

>ridge

>summary(ridge)

>plot(ridge,xvar = "lambda") #here we are plotting the hyperparameter tuning for the better representation of the process

A graph with a line

Description automatically generated

>ridge$bestTune$alpha

>prediction.train <- predict(ridge, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>ridge.r2 <- 1.0-train.rss/default.train.rss

>ridge.r2

>prediction.test <- predict(ridge, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>ridge.pseudoR2 <- 1.0-test.rss/default.test.rss

>ridge.pseudoR2

>report <- rbind(report, data.frame(Model="Ridge Regression", R2.Train=ridge.r2, R2.Test=ridge.pseudoR2))

Ridge regression allows to achieve pseudo-R2 of 0.7377 on the test set

### K-Nearest Neighbors

In K-nearest neighbors, we benefit from its adaptability to non-linear relationships in the data.

> knnGrid <- expand.grid(k=seq(from=2, to=15, by=1))

> set.seed(123)

> knnmodel <- train(expenses ~ ., data=mydata\_v3.train, method="knn", tuneGrid=knnGrid, trControl=TControl, metric="Rsquared")

> knnmodel

k-Nearest Neighbors

1070 samples

8 predictor

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 2 times)

Summary of sample sizes: 962, 962, 962, 964, 964, 965, ...

Resampling results across tuning parameters:

k RMSE Rsquared MAE

2 0.10010777 0.7436762 0.05687515

3 0.09625821 0.7607429 0.05757486

4 0.09431223 0.7692670 0.05780796

5 0.09535921 0.7641311 0.05986982

6 0.09640674 0.7594578 0.06111390

7 0.09744520 0.7539209 0.06134600

8 0.09863479 0.7477050 0.06195016

9 0.09861239 0.7478860 0.06190854

10 0.09862229 0.7481347 0.06160344

11 0.09900916 0.7463305 0.06175682

12 0.09975262 0.7424666 0.06230479

13 0.10055603 0.7384751 0.06271655

14 0.10137284 0.7338123 0.06317930

15 0.10161326 0.7328284 0.06344426

Rsquared was used to select the optimal model using the largest value.

The final value used for the model was k = 4.

> summary(knnmodel)

Length Class Mode

learn 2 -none- list

k 1 -none- numeric

theDots 0 -none- list

xNames 8 -none- character

problemType 1 -none- character

tuneValue 1 data.frame list

obsLevels 1 -none- logical

param 0 -none- list

> plot(knnmodel)

A graph with blue dots and numbers

Description automatically generated

> knnmodel$finalModel

4-nearest neighbor regression model

>prediction.train <- predict(knnmodel, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>knn.r2 <- 1.0-train.rss/default.train.rss

>knn.r2

>prediction.test <- predict(knnmodel, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>knn.pseudoR2 <- 1.0-test.rss/default.test.rss

>knn.pseudoR2

>report <- rbind(report, data.frame(Model="k-NN", R2.Train=knn.r2, R2.Test=knn.pseudoR2))

Resulting 4-neighbours model gave the resulting Pseudo-R2 of 0.7482 on the test set

### Support vector machines/Kernel

To make sure that relationship between variables is indeed non-linear, it is rational to compare Support Vector Machines with radial and linear Kernel. We expect the radial Kernel to give way better results, thus supporting the non-linear relationships between variables.

#### Linear Kernel:

>svlGrid <- expand.grid(C=seq(from=0.01, to=1, by=0.1))

>eps <- 0.1

>set.seed(123)

>svr.linear <- train(expenses ~ ., data=mydata\_v3.train, method="svmLinear", tuneGrid=svlGrid, trControl=TControl, metric="Rsquared", epsilon = eps)

>svr.linear

Support Vector Machines with Linear Kernel

1070 samples

8 predictor

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 2 times)

Summary of sample sizes: 962, 962, 962, 964, 964, 965, ...

Resampling results across tuning parameters:

C RMSE Rsquared MAE

0.01 0.1116129 0.7335193 0.06243475

0.11 0.1075764 0.7340327 0.05682491

0.21 0.1077608 0.7340529 0.05679029

0.31 0.1078024 0.7340082 0.05679094

0.41 0.1078469 0.7340042 0.05677943

0.51 0.1079183 0.7339861 0.05675915

0.61 0.1079417 0.7339642 0.05676524

0.71 0.1079495 0.7339565 0.05676327

0.81 0.1079537 0.7339688 0.05676364

0.91 0.1079507 0.7339609 0.05676019

Rsquared was used to select the optimal model using the largest value.

The final value used for the model was C = 0.21.

>plot(svr.linear,xvar = "C")

A graph with a line going up

Description automatically generated

>svlres<-cbind(svr.linear$results$C, svr.linear$results$Rsquared,svr.linear$results$MAE)

>prediction.train <- predict(svr.linear, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>svrlin.r2 <- 1.0-train.rss/default.train.rss

>svrlin.r2

>prediction.test <- predict(svr.linear, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>svrlin.pseudoR2 <- 1.0-test.rss/default.test.rss

>svrlin.pseudoR2

>report <- rbind(report, data.frame(Model="Linear SVR", R2.Train=svrlin.r2, R2.Test=svrlin.pseudoR2))

Support vector machine Linear Kernel with C = 0.21. gave the result of Pseudo R2 on test set of 0.6685.

#### Radial Kernel:

>svrGrid <- expand.grid(sigma = c(.01, .015, 0.2), C = c(1, 10, 50, 100))

>eps <- 0.1

>set.seed(123)

>svr.rbf <- train(expenses ~ ., data=mydata\_v3.train, method="svmRadial", tuneGrid=svrGrid, trControl=TControl, metric="Rsquared", epsilon = eps)

>svr.rbf: #resulting hyperparameters are sigma = 0.01 and C = 100

|  |  |
| --- | --- |
| **sigma** | **C** |
| 0.01 | 1 |
| 0.015 | 1 |
| 0.2 | 1 |
| 0.01 | 10 |
| 0.015 | 10 |
| 0.2 | 10 |
| 0.01 | 50 |
| 0.015 | 50 |
| 0.2 | 50 |
| 0.01 | 100 |
| 0.015 | 100 |
| 0.2 | 100 |

>plot(svr.rbf)

A graph with lines and numbers

Description automatically generated

>prediction.train <- predict(svr.rbf, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>svrrbf.r2 <- 1.0-train.rss/default.train.rss

>svrrbf.r2

>prediction.test <- predict(svr.rbf, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>svrrbf.pseudoR2 <- 1.0-test.rss/default.test.rss

>svrrbf.pseudoR2

>report <- rbind(report, data.frame(Model="Radial SVR", R2.Train=svrrbf.r2, R2.Test=svrrbf.pseudoR2))

Support vector machine Radial Kernel with sigma = 0.01 and C = 100 gave the result of Pseudo R2 on test set of 0.8307, which drastically improved results, suggesting non-linear relationships between variables

### Neural Network (1 hidden layer).

Observing non-linear relationships between variables, we decided that the next algorithm to evaluate will be Neural Network type 1 – with one hidden layer, called by the function nnet in train. It has 2 adjustable hyperparameters: size and decay, where size signifies the size of the hidden layer.

>nnGrid <- expand.grid(size=1:8, decay=seq(from=0.1, to=0.5, by=0.1))

>set.seed(123)

>nnmodel <- train(expenses ~ ., data=mydata\_v3.train, method="nnet", tuneGrid=nnGrid, trControl=TControl, trace=FALSE, metric="Rsquared")

>nnmodel

>library(NeuralNetTools)

>plotnet(nnmodel)

>nnmodel$results

>plot(nnmodel)

A diagram of a graph

Description automatically generated with medium confidence

>nnmodel$finalModel

#Using (8)-2-(1) model with decay=0.1

>prediction.train <- predict(nnmodel, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>nn.r2 <- 1.0-train.rss/default.train.rss

>nn.r2

>prediction.test <- predict(nnmodel, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>nn.pseudoR2 <- 1.0-test.rss/default.test.rss

>nn.pseudoR2

>report <- rbind(report, data.frame(Model="Neural Network 1", R2.Train=nn.r2, R2.Test=nn.pseudoR2))

This model gives Pseudo R2 for test set of 0.7694. It might be rational to adjust the decay: the new grid for decay will be between 0.1 and 0.2 (using 0.05 as a margin added to the previous tuneGrid), because of the observable high performance of decay 0.1:

>nnGrid\_1 <- expand.grid(size=1:8, decay=seq(from=0.05, to=0.25, by=0.01))

>set.seed(123)

>nnmodel\_1 <- train(expenses ~ ., data=mydata\_v3.train, method="nnet", tuneGrid=nnGrid\_1, trControl=TControl, trace=FALSE, metric="Rsquared")

>nnmodel\_1

#Using (8)-5-(1) model with decay=0.05

>plotnet(nnmodel\_1)

A graph of a number of numbers and a number of objects

Description automatically generated with medium confidence

>nnmodel\_1$finalModel

>prediction.train <- predict(nnmodel\_1, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>nn.r2 <- 1.0-train.rss/default.train.rss

>nn.r2

>prediction.test <- predict(nnmodel\_1, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>nn.pseudoR2 <- 1.0-test.rss/default.test.rss

>nn.pseudoR2

>report <- rbind(report, data.frame(Model="Neural Network DecAdj", R2.Train=nn.r2, R2.Test=nn.pseudoR2))

This adjusted model gives Pseudo R2 for test set of 0.8162, which shows significant improvement.

### Neural Network (multiple hidden layers)

To further expand upon multifauceted interactions between variables, usage of multiple-layer model might be rational using “neuralnet” method, which has only hyperparameters related to respective sizes of layers (layer\_1, layer\_2, layer\_3).

>nn2Grid <- expand.grid(layer1=c(1,3,5,7,8), layer2=c(0,1,3,5,7,8), layer3=c(0,1,3,5,7,8))

>set.seed(123)

>nnmodel2 <- train(expenses ~ ., data=mydata\_v3.train, method="neuralnet", tuneGrid=nn2Grid, >trControl=TControl, metric="Rsquared")

>nnmodel2

>plot(nnmodel2$finalModel)

A diagram of a network

Description automatically generated

>nnmodel2$bestTune

>nnmodel2$finalModel

#Using (8)-3-3-7-(1) model

>prediction.train <- predict(nnmodel2, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>nn.r2 <- 1.0-train.rss/default.train.rss

>nn.r2

>prediction.test <- predict(nnmodel2, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>nn.pseudoR2 <- 1.0-test.rss/default.test.rss

>nn.pseudoR2

>report <- rbind(report, data.frame(Model="Neural Network 2", R2.Train=nn.r2, R2.Test=nn.pseudoR2))

This model of structure (8)-3-3-7-(1) gave Pseudo R2 for test set of 0.8531, which improved the results of the 1-layered model discussed previously.

Nevertheless, this type of model has several downsides:

1. It is taking much more time to train
2. The neuralnet package does not allow for additional hyperparameters or increased size of the neural network
3. R has limited number of functions to inspect the model and hyperparameter tuning.

It is suggested that for the future research on the matter, Python will be used with more expandable libraries for neural networks.

### Random Forest (no optimisation)

The last group of algorithms to try are Random-forest related algorithms. In the first iteration the hyperparameter maxnodes is not tuned, thus the train function performs only automatic mtry tuning:

>set.seed(123)

>rfmodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", trControl=TControl, >metric="Rsquared")

>rfmodel

|  |  |  |  |
| --- | --- | --- | --- |
| **mtry** | **RMSE** | **Rsquared** | **MAE** |
| 2 | 0.084449 | 0.842333 | 0.057702 |
| 5 | 0.074729 | 0.852679 | 0.041999 |
| 8 | 0.076909 | 0.844876 | 0.043334 |

>summary(rfmodel)

>plot(rfmodel)

A graph with a line and numbers

Description automatically generated

>prediction.train <- predict(rfmodel, newdata = mydata\_v3.train)

>train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

>rf.r2 <- 1.0-train.rss/default.train.rss

>rf.r2

>prediction.test <- predict(rfmodel, newdata = mydata\_v3.test)

>test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

>rf.pseudoR2 <- 1.0-test.rss/default.test.rss

>rf.pseudoR2

>report <- rbind(report, data.frame(Model="Random Forest", R2.Train=rf.r2, R2.Test=rf.pseudoR2))

This model of with mtry of 5 gave Pseudo R2 for test set of 0.8481, however also showing R2 for training set:

of 0.9573, suggesting that the model is overfit. It signifies the need for tuning maxnodes hyperparameter.

### Random Forest (reduced step maxnodes optimization)

We decided to perform algorythmical hyperparameter tuning in order to reduce the size of the TuneGrid.

The idea of optimization of complex hyperparameter H is to find an extremum of the corresponding accuracy metric (R2 on training set).

Thus, we research consecutively several tuneGrids with reducing steps between the grid elements: 50->25->12->6->1 was used. In order to find an extremum on the specific scale (i.g.50), we need to find such H that hyperparameters lower and higher that H on the TuneGrid will give the lower metric (simple local extremum).

After the H is found, two closest measures of H are now the limitations of the new grid, and the step is also gradually reduced.

The code looks as follows:

#ZOOMING IN:Choosing initial maxnode step 50

for (maxnodes in c(10,60,110,160,210)) {

set.seed(123)

rfmodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", metric="Rsquared",

trControl=TControl, maxnodes=maxnodes)

prediction.train <- predict(rfmodel, newdata = mydata\_v3.train)

train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

rf.r2 <- 1.0-train.rss/default.train.rss

prediction.test <- predict(rfmodel, newdata = mydata\_v3.test)

test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

rf.pseudoR2 <- 1.0-test.rss/default.test.rss

cat("maxnodes = ", maxnodes, "\n")

cat(rf.r2, rf.pseudoR2, "\n\n")

}

#110 - max R2 for test: The needed maxnode parameter is located somewhere between 60 and 160

#ZOOMING IN:Step 25

for (maxnodes in c(60,85,110,135,160)) {

set.seed(123)

rfmodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", metric="Rsquared",

trControl=TControl, maxnodes=maxnodes)

prediction.train <- predict(rfmodel, newdata = mydata\_v3.train)

train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

rf.r2 <- 1.0-train.rss/default.train.rss

prediction.test <- predict(rfmodel, newdata = mydata\_v3.test)

test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

rf.pseudoR2 <- 1.0-test.rss/default.test.rss

cat("maxnodes = ", maxnodes, "\n")

cat(rf.r2, rf.pseudoR2, "\n\n")

}

#Needed result is between 85 and 135

#ZOOMING IN:Step is somewhere around 12

for (maxnodes in c(85,97,110,122,135)) {

set.seed(123)

rfmodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", metric="Rsquared",

trControl=TControl, maxnodes=maxnodes)

prediction.train <- predict(rfmodel, newdata = mydata\_v3.train)

train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

rf.r2 <- 1.0-train.rss/default.train.rss

prediction.test <- predict(rfmodel, newdata = mydata\_v3.test)

test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

rf.pseudoR2 <- 1.0-test.rss/default.test.rss

cat("maxnodes = ", maxnodes, "\n")

cat(rf.r2, rf.pseudoR2, "\n\n")

}

#Needed result is between 97 and 122

#Step is somewhere around 12

#ZOOMING IN: Step is 6

for (maxnodes in c(97,103,110,116,122)) {

set.seed(123)

rfmodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", metric="Rsquared",

trControl=TControl, maxnodes=maxnodes)

prediction.train <- predict(rfmodel, newdata = mydata\_v3.train)

train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

rf.r2 <- 1.0-train.rss/default.train.rss

prediction.test <- predict(rfmodel, newdata = mydata\_v3.test)

test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

rf.pseudoR2 <- 1.0-test.rss/default.test.rss

cat("maxnodes = ", maxnodes, "\n")

cat(rf.r2, rf.pseudoR2, "\n\n")

}

#Needed result is between 110 and 122

#ZOOMING IN: Step is 3

for (maxnodes in c(110,113,116,119,122)) {

set.seed(123)

rfmodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", metric="Rsquared",

trControl=TControl, maxnodes=maxnodes)

prediction.train <- predict(rfmodel, newdata = mydata\_v3.train)

train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

rf.r2 <- 1.0-train.rss/default.train.rss

prediction.test <- predict(rfmodel, newdata = mydata\_v3.test)

test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

rf.pseudoR2 <- 1.0-test.rss/default.test.rss

cat("maxnodes = ", maxnodes, "\n")

cat(rf.r2, rf.pseudoR2, "\n\n")

}

#Needed result is between 113 and 119

#ZOOMING IN: Step is 3

for (maxnodes in c(113,114,115,116,117,118,119)) {

set.seed(123)

rfmodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", metric="Rsquared",

trControl=TControl, maxnodes=maxnodes)

prediction.train <- predict(rfmodel, newdata = mydata\_v3.train)

train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

rf.r2 <- 1.0-train.rss/default.train.rss

prediction.test <- predict(rfmodel, newdata = mydata\_v3.test)

test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

rf.pseudoR2 <- 1.0-test.rss/default.test.rss

cat("maxnodes = ", maxnodes, "\n")

cat(rf.r2, rf.pseudoR2, "\n\n")

}

The most effective maxnodes parameter for the test set is achieved at =116.

However, adjusting the maxnodes parameter for test set is faulty, for it reduces external validity of the model.

Therefore, the operations must be performed with the same algorithm, but concerning training set (rf.r2), while keeping in mind overfitting dangers. It is not reflected in the code, but is expected to bring upon better performance.

Finally, the code using maxnodes parameter of 116:

set.seed(123)

rfomodel <- train(expenses ~ ., data=mydata\_v3.train, method="rf", trControl=TControl,

metric="Rsquared", maxnodes=116)

rfomodel

summary(rfomodel)

rfomodel$finalModel

prediction.train <- predict(rfomodel, newdata = mydata\_v3.train)

train.rss <- sum((mydata\_v3.train$expenses-prediction.train)^2)

rf.r2 <- 1.0-train.rss/default.train.rss

rf.r2

prediction.test <- predict(rfomodel, newdata = mydata\_v3.test)

test.rss <- sum((mydata\_v3.test$expenses-prediction.test)^2)

rf.pseudoR2 <- 1.0-test.rss/default.test.rss

rf.pseudoR2

report <- rbind(report, data.frame(Model="Random Forest opt.", R2.Train=rf.r2, R2.Test=rf.pseudoR2))

Which returns pseudo R2 for test set of 0.8547, which is fit to the test data, thus lacking external validity

## Report building

After all the models are evaluated and put into report the following code returns the necessary files and visualisations of model performance. We also included result output model for the futher work with results in tools such as Excel and Tableau.

>report

>results <- resamples(list(OLS=ols, Ridge=ridge, SVM.L=svr.linear, SVM.R=svr.rbf,

NeuNet=nnmodel,NeuNetAdj=nnmodel\_1, NeuNet2=nnmodel2, RFor=rfmodel, RFor.o=rfomodel, KNN=knnmodel))

>summary(results)

>dotplot(results)

A graph with numbers and letters

Description automatically generated with medium confidence

>write.csv(report, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\report.csv", row.names=FALSE)

>write.csv(corr.matrix, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\correlation.csv", row.names=FALSE)

>write.csv(svlGrid, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\svlgrid.csv", row.names=FALSE)

>write.csv(svrGrid, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\svrgrid.csv", row.names=FALSE)

>write.csv(svr.linear$results, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\svlres.csv", row.names=FALSE)

>write.csv(nnmodel$results, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\nnmodres.csv", row.names=FALSE)

>write.csv(nnGrid, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\nngrid.csv", row.names=FALSE)

>write.csv(rfmodel$results, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\rfres.csv", row.names=FALSE)

>write.csv(knnmodel$results, "D:\\FSFM studies\\ML & Business Experiments\\Groupwork\_ML\_Workplace\\knnres.csv", row.names=FALSE)

Finalized version of the report shows that the best algorithm to use for the following problem is **neural network with several hidden layers in the (8)-3-3-8-(1) structure.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **R2.Train** | **R2.Test** | **Hyperparameters** |
| Random Forest optimized | 0.927427 | 0.854702 | maxnodes = 116 (lacks ext.validity) |
| Neural Network Multi-Layer | 0.867340 | 0.853113 | (8)-3-3-8-(1) Neuralnet |
| Random Forest | 0.957386 | 0.848082 | mtry=5 |
| Radial SVR | 0.852334 | 0.830719 | sigma = 0.01, Cost = 100 |
| Neural Network Single-Layer Adjusted Decay | 0.835498 | 0.816199 | (8)-5-(1) Nnet decay = 0.05 |
| Neural Network Single-Layer | 0.793841 | 0.769397 | (8)-2-(1) Nnet decay = 0.1 |
| k-NN | 0.866813 | 0.748233 | k=4 |
| Linear Regression | 0.753285 | 0.736734 | - |
| Ridge Regression | 0.750788 | 0.736660 | Alpha=0, Lambda = 0.01 |
| Linear SVR | 0.699909 | 0.668502 | c = 0.21 |

# Conclusion of the report

The model for predicting insurance expenses was successfully selected and tuned.

In our work, we focused not only on the modelling itself, but on the numerous visualisations.

Throughout the work it became apparent that the Machine Learning methods allow to achieve outstanding results in business situations when applied thoughtfully. Furthermore, the visual support and well-defined explanations of the actions, as we learned during the presentation, might lead to the fruitful collaborations with contributors.

We would like to voice special thanks to Prof. Dr. Peter Roßbach for the course, the guidance and assistance as well as to independent contributions of other MIM students during the presentation day, specifically Jan-Oliver Tiedemann, for emphasizing the corrections to be made.