Predicting Salaries Index With Different Machine Learning Methods

Sara Amhan-2105687, Zeynep Balci-1903441, Elif Kilic-1903855

Introduction

In this report, we are going to train different machine learning methods and test their performance on our test data. Our data can be downloaded from the link:<https://data.tuik.gov.tr/Bulten/DownloadIstatistikselTablo?p=T5g/WOiY1BH4Jbmww3jtCo1q03
Our task is regression and the method we used in this report are: MLR, Random Forest, Decision Tree, k-NN, SVM regression and a stacking ensemble method from the library H20.

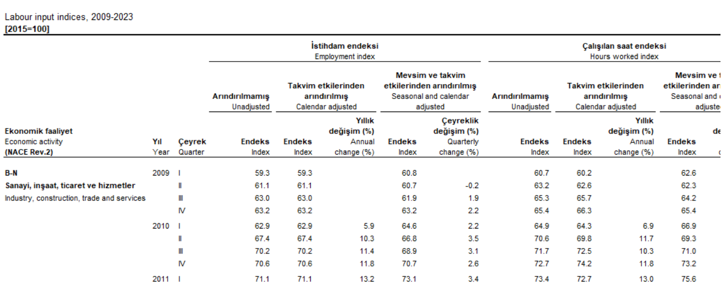
Research Question

Our research question is:

"What is the best machine learning method or algorithm that best predicts the salary index based on different factors?"

Data & data structure

Here's a look into the excel file…



Labour input indices, 2009-2023
[2015=100]

Ekonomik faaliyet / Economic activity (NACE Rev.2)	Yıl / Year	Çeyrek / Quarter	İstihdam endeksi / Employment index: Arındırılmamış / Unadjusted — Endeks / Index	Takvim etkilerinden arındırılmış / Calendar adjusted — Endeks / Index	Takvim etkilerinden arındırılmış / Calendar adjusted — Yıllık değişim (%) / Annual change (%)	Mevsim ve takvim etkilerinden arındırılmış / Seasonal and calendar adjusted — Endeks / Index	Mevsim ve takvim etkilerinden arındırılmış / Seasonal and calendar adjusted — Çeyreklik değişim (%) / Quarterly change (%)	Çalışılan saat endeksi / Hours worked index: Arındırılmamış / Unadjusted — Endeks / Index	Takvim etkilerinden arındırılmış / Calendar adjusted — Endeks / Index	Takvim etkilerinden arındırılmış / Calendar adjusted — Yıllık değişim (%) / Annual change (%)	Mevsim ve t… etkilerinden ar… / Seasonal and … adjuste… — Endeks / Index
B-N	2009	I	59.3	59.3		60.8		60.7	60.2		62.6
Sanayi, inşaat, ticaret ve hizmetler		II	61.1	61.1		60.7	-0.2	63.2	62.6		62.3
Industry, construction, trade and services		III	63.0	63.0		61.9	1.9	65.3	65.7		64.2
		IV	63.2	63.2		63.2	2.2	65.4	66.3		65.4
	2010	I	62.9	62.9	5.9	64.6	2.2	64.9	64.3	6.9	66.9
		II	67.4	67.4	10.3	66.8	3.5	70.6	69.8	11.7	69.3
		III	70.2	70.2	11.4	68.9	3.1	71.7	72.5	10.3	71.0
		IV	70.6	70.6	11.8	70.7	2.6	72.7	74.2	11.8	73.2
	2011	I	71.1	71.1	13.2	73.1	3.4	73.4	72.7	13.0	75.6
		II	76.3	76.3	13.2	75.6	3.3	79.5	78.5	12.4	77.8

Fig.1: Data Excel File

The excel file consists of multiple tables stacked on top of each
other each representing the features of an economic activity. The features columns are:

- Employment Index: Unadjusted, Calender adjusted, Seasonal and

  calender adjusted
- Hours worked Index: Unadjusted, Calender adjusted, Seasonal and

  calender adjusted
- Gross wages-salaries Index: Unadjusted, Calender adjusted,Seasonal and

  calender adjusted

Code:

```
library(readxl)
data1 <- read_excel("data/salary index data.xls",  range= "A127:Z726", col_names = FALSE)
data2 <- read_excel("data/salary index data.xls",  range= "A787:Z1266", col_names = FALSE)
full_data <- rbind(data1, data2)
dim(full_data)
```

```
## [1] 1080   26
```

```
print(full_data[1:3, 1:13])
```

```
## # A tibble: 3 x 13
##    ...1   ...2  ...3   ...4 ...5   ...6  ...7 ...8   ...9 ...10 ...11 ...12 ...13
```

```
##   <chr>   <chr> <chr> <dbl> <lgl> <dbl> <dbl> <lgl> <dbl> <dbl> <lgl> <dbl> <lgl>
## 1 ARM-A~ 2009  I      64.0 NA     64.0    NA NA     65.1 NA    NA     65.9 NA
## 2 IG-In~ <NA>  II     64.5 NA     64.5    NA NA     64.3 -1.19 NA     67.7 NA
## 3 <NA>   <NA>  III    66.5 NA     66.5    NA NA     65.8  2.32 NA     69.3 NA
```

Data Cleaning

Renaming and dropping unwanted columns

We are only using the unadjusted version of the columns…

```r
library(dplyr)
select_and_rename_v2 <- function(df) { # cleaning pipeline
  selected_df <- df[, c(1, 2, 3, 4, 12, 20)] # Select columns 1, 2, 3, 4, 12, and 20
  # Rename the columns with desired names
  names(selected_df) <- c("Economic activity", "Year", "Quarter", "Employment index", "Hours worked index", "salaries index
  # Reordering
  selected_df = selected_df[c("Year", "Quarter", "Employment index", "Hours worked index", "salaries index", "Economic acti
  return(selected_df) }
clean_full_data = select_and_rename_v2(full_data)
head(clean_full_data, 3)
```

```
## # A tibble: 3 x 6
##   Year  Quarter `Employment index` `Hours worked index` `salaries index`
##   <chr> <chr>                <dbl>                <dbl>            <dbl>
## 1 2009  I                     64.0                 65.9             35.6
## 2 <NA>  II                    64.5                 67.7             35.1
## 3 <NA>  III                   66.5                 69.3             37.5
## # i 1 more variable: `Economic activity` <chr>
```

Fixing wrong entries

Some comment where made by tuik and that caused some entries in the Year column to have letters so we are fixing them so that the column can be of type int…

```r
clean_full_data[53,]
```

```
## # A tibble: 1 x 6
##   Year    Quarter `Employment index` `Hours worked index` `salaries index`
##   <chr>   <chr>                <dbl>                <dbl>            <dbl>
## 1 2022(r) I                     123.                 115.             450.
## # i 1 more variable: `Economic activity` <chr>
```

```r
clean_full_data$Year <- gsub("\\(r\\)", "", clean_full_data$Year) # using regular expressions to fix wrong entries
clean_full_data$Year <- as.numeric(clean_full_data$Year)
clean_full_data[53,]
```

```
## # A tibble: 1 x 6
##    Year Quarter `Employment index` `Hours worked index` `salaries index`
##   <dbl> <chr>                <dbl>                <dbl>            <dbl>
## 1  2022 I                     123.                 115.             450.
## # i 1 more variable: `Economic activity` <chr>
```

Filling NA's

seeing what column have NA values..

```r
colSums(is.na(clean_full_data))
```

```
##               Year            Quarter   Employment index Hours worked index
##                810                  0                  0                  0
##     salaries index  Economic activity
##                  0               1021
```

Year and Quarter columns: the year column has empty entries that indicate the value of the last non empty and because we stacked all the tables. For those empty entries we will be using a function that replaces the NAs with the last non-NA value. We are making a function pipeline that replaces quarter with a number and merges both columns into one so it becomes our only time series column.

```r
library(dplyr)
library(zoo)
one_column_function <- function(df) {
  quarter_mapping <- c("I" = 1, "II" = 2, "III" = 3, "IV" = 4)
  df <- df |>
    mutate(
      Year = na.locf(df$Year, na.rm = FALSE),
      Year_Quarter = as.numeric(as.numeric(Year) + (quarter_mapping[Quarter] - 1) / 4
    )) |>
    select(-Year, -Quarter)
  return(df) }
clean_full_data = one_column_function(clean_full_data)
head(clean_full_data,3)
```

```
## # A tibble: 3 x 5
##   `Employment index` `Hours worked index` `salaries index` `Economic activity`
##                <dbl>                <dbl>            <dbl> <chr>
## 1               64.0                 65.9             35.6 ARM-Ara malı imalatı
## 2               64.5                 67.7             35.1 IG-Intermediate goods
## 3               66.5                 69.3             37.5 <NA>
## # i 1 more variable: Year_Quarter <dbl>
```

Economic Activity Column: has empty entries for the same reason above but in this case each table from the tables stacked has a different value for that we are making a window and iterating through each window to fill the NA's with the first non-NA value in that window (each table from the stacked tables has 60 rows thus the /60). After that we made the column of type factor...

```r
library(dplyr)
fill_pattern <- function(df) {
  n_rows <- nrow(df)
  num_windows <- ceiling(n_rows / 60)
  for (i in 1:num_windows) { # Iterate over each window
    start_index <- (i - 1) * 60 + 1 # Determine the start and end indices of the current window
    end_index <- min(i * 60, n_rows)
    df[start_index:end_index, "Economic activity"] <- df[start_index+1, "Economic activity"] # Fill the current window with
  }
  return(df)}
clean_full_data = fill_pattern(clean_full_data)
head(clean_full_data,3)
```

```
## # A tibble: 3 x 5
##   `Employment index` `Hours worked index` `salaries index` `Economic activity`
##                <dbl>                <dbl>            <dbl> <chr>
## 1               64.0                 65.9             35.6 IG-Intermediate goods
## 2               64.5                 67.7             35.1 IG-Intermediate goods
## 3               66.5                 69.3             37.5 IG-Intermediate goods
## # i 1 more variable: Year_Quarter <dbl>
```

```r
clean_full_data$`Economic activity` = factor(clean_full_data$`Economic activity`)
```

fixing names: because some names have some special Turkish characters and changing them would mean not running into unexpected errors when training

```r
library(dplyr)
library(forcats)
clean_full_data <- clean_full_data |> # changing name with forcats
  mutate(`Economic activity` = fct_recode(`Economic activity`, "Intermediate goods" = "IG-Intermediate goods",
          "Durable consumer goods" = "DCG-Durable consumer goods","Non-durable consumer goods" = "NDCG-Non-durable consumer
          "Energy" = "NRG-Energy","Capital goods" = "CG-Capital goods",
          "Mining and quarrying" = "Madencilik ve taş ocakçılığı","Manufacturing" = "İmalat",
          "Electricity, gas, steam and air conditioning supply" = "Elektrik, gaz, buhar ve iklimlendirme",
          "Water supply, sewerage, waste management and remediation activities" = "Su temini; kanalizasyon, atık yönetimi",
          "Construction" = "İnşaat","Wholesale and retail trade" = "Toptan ve perakende ticaret;",
          "Transportation and storage" = "Ulaştırma ve depolama","Accommodation and food service activities" = "Konaklama v
          "Information and communication" = "Bilgi ve iletişim","Financial and insurance activities" = "Finans ve sigorta f
          "Real estate activities" = "Gayrimenkul faaliyetleri",
          "Professional, scientific and technical activities" = "Mesleki, bilimsel ve teknik faaliyetler",
          "Administrative and support service activities" = "İdari ve destek hizmet faaliyetleri"))
head(levels(clean_full_data$`Economic activity`),3)
```

```
## [1] "Information and communication" "Capital goods"
## [3] "Durable consumer goods"
```

Final structure

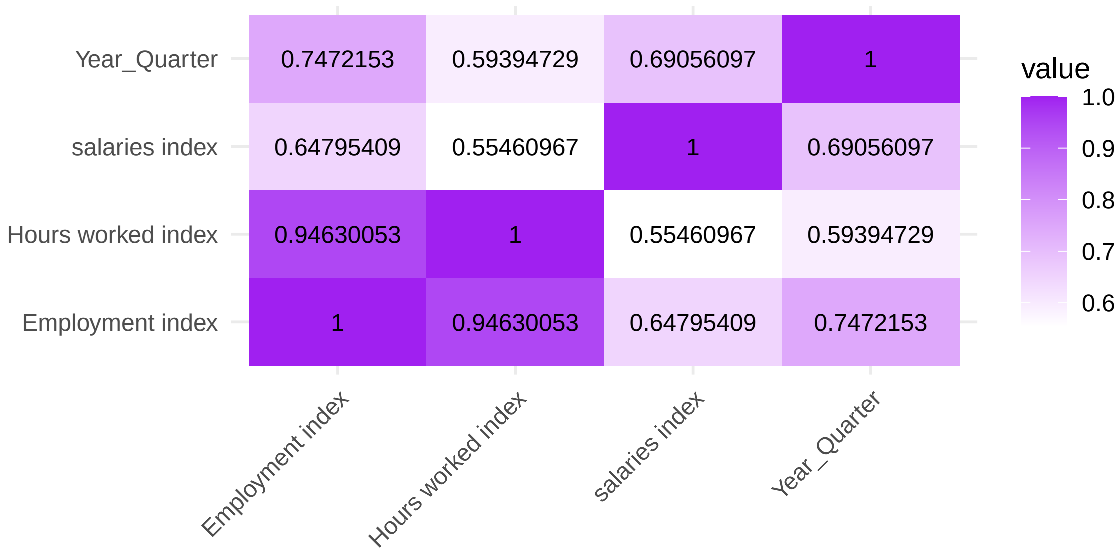
```r
str(clean_full_data)
dim(clean_full_data)
```

```
## tibble [1,080 x 5] (S3: tbl_df/tbl/data.frame)
##  $ Employment index  : num [1:1080] 64 64.5 66.5 67.3 67.7 ...
##  $ Hours worked index: num [1:1080] 65.9 67.7 69.3 71 70.7 ...
##  $ salaries index    : num [1:1080] 35.6 35.1 37.5 38.7 39.4 ...
##  $ Economic activity : Factor w/ 18 levels "Information and communication",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ Year_Quarter      : num [1:1080] 2009 2009 2010 2010 2010 ...
## [1] 1080    5
```

Visualizations

- correlation heatmap between all variables

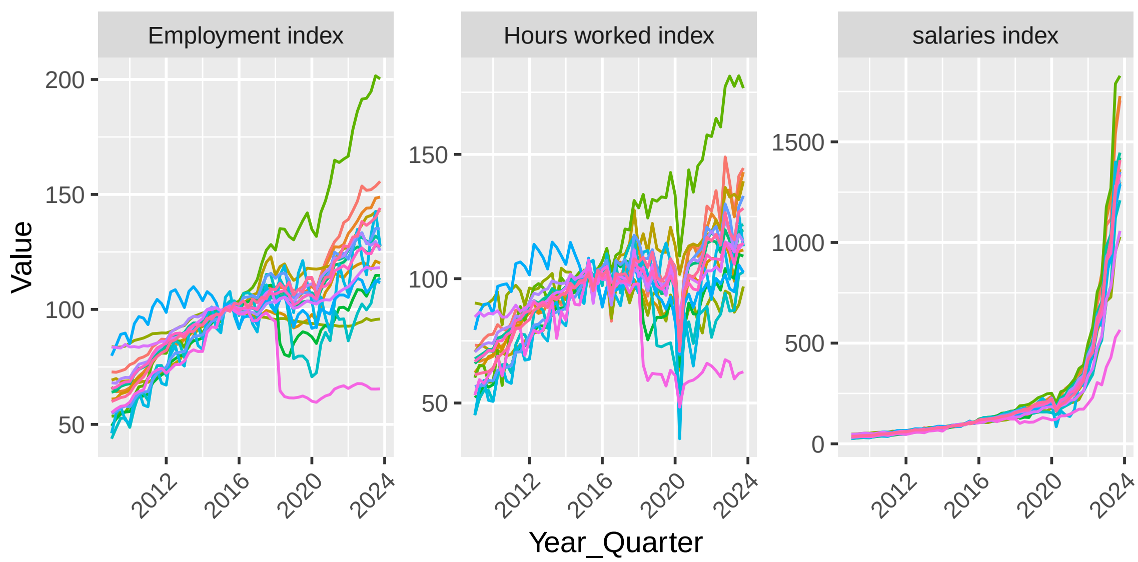
```r
library(ggplot2)
library(reshape2)
data <- cor(clean_full_data[sapply(clean_full_data, is.numeric)]) # Calculating correlation matrix
data1 <- melt(data) # Reshaping data
p <- ggplot(data1, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  geom_text(aes(label = round(value, 8)), color = "black", size = 3) +
  scale_fill_gradient(low = "white", high = "purple") + # Color gradient for heatmap
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), # Rotating x-axis labels for better readability
        axis.title.x = element_blank(), # Removing axes titles
        axis.title.y = element_blank())
print(p)
```



The correlation between the target and the features is not bad but the biggest maybe problem hear is the feature co-linearity, although that is a bad thing we are not going to drop any feature because we don't have a lot of features to begin with.

- Visualizing trend over time

```
library(tidyr)
library(ggplot2)
# Reshaping data into long format, excluding Economic_activity and Year_Quarter columns
df_long <- gather(clean_full_data, key = "Variable", value = "Value", -Year_Quarter, -`Economic activity`)
# Ploting line plots for each variable, using Economic_activity as hue
ggplot(df_long, aes(x = Year_Quarter, y = Value, color = `Economic activity`, group = `Economic activity`)) +
  geom_line() +
  facet_wrap(~ Variable, scales = "free_y", nrow = 1) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  guides(color = FALSE)  # Removing the legend
```



We can see the co-linearity between the previous two features but notice how the trends are a bit different which is information we obtain by keeping the feature and training on the whole data set.

Data Pre-processing

Feature Selection

- we are gonna use all the features despite two being highly correlated. We will later use other methods to ensure good accuracy. . .

```
data <- clean_full_data
```

Encoding

encoding is recommended so that the algorithm doesn't have to deal with characters, we are using one-hot encoding because the categorical column is not ordinal but nominal.

```
library(fastDummies)
data <- dummy_cols(data, select_columns = "Economic activity")
data <- subset(data, select = -c(`Economic activity`))
tail(colnames(data),3)
```

```
## [1] "Economic activity_Water supply, sewerage, waste management and remediation activities"
## [2] "Economic activity_Wholesale and retail trade"
## [3] "Economic activity_Transportation and storage"
```

Renaming "salaries index" to y for easier and shorter code

```
names(data)[names(data) == "salaries index"] <- "y"
head(data$y,3)
```

```
## [1] 35.60527 35.13005 37.46780
```

Train Test Split

We will be doing an 80% training, 10% validation, 10% testing ratio for most models thus the code below…

```
library(caret)
set.seed(45) # Setting the seed for reproducibility
train_indices <- createDataPartition(data$y, p = 0.9, list = FALSE) # Splitting the dataset into 90% training and 10% test
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]
cat("Training data dimensions:", dim(train_data), "\n")
cat("Test data dimensions:", dim(test_data), "\n")
```

```
## Training data dimensions: 972 22
## Test data dimensions: 108 22
```

Scaling

because scaling is recommended for most models we are applying it but for any other case we will first save the unscaled version…

```
unscaled_train_data = train_data # Saving unscaled version
unscaled_test_data = test_data
library(caret)
set.seed(45)
columns_to_normalize <- c("Employment index", "Hours worked index", "y", "Year_Quarter")
preprocess <- preProcess(train_data[, columns_to_normalize], method = c("center", "scale"))# Pre-processing pipeline to *n
norm_train_data <- predict(preprocess, newdata = train_data)# Applying the pipeline to the training data
norm_test_data <- predict(preprocess, newdata = test_data)# Applying the same pipeline to the test data
print(norm_train_data[1:3, 1:4])
```

```
## # A tibble: 3 x 4
##   `Employment index` `Hours worked index`      y Year_Quarter
##                <dbl>                <dbl>  <dbl>        <dbl>
## 1              -1.58                -1.58 -0.616        -1.70
## 2              -1.55                -1.47 -0.617        -1.64
## 3              -1.46                -1.39 -0.609        -1.59
```

Specifying X & Y

Assigning X and Y for upcoming testing…

```r
library(dplyr)
# Unscaled train & test, y & x
unscaled_train_x = select(unscaled_train_data, -y)
unscaled_train_y = as.numeric(unscaled_train_data[["y"]])
unscaled_test_x = select(unscaled_test_data, -y)
unscaled_test_y = as.numeric(unscaled_test_data[["y"]])
# Scaled train & test, y & x
train_x = select(norm_train_data, -y)
train_y = as.numeric(norm_train_data[["y"]])
test_x = select(norm_test_data, -y)
test_y = as.numeric(norm_test_data[["y"]])
```

Models

Testing score function

We are going to use the normalized version of RMSE as the normal RMSE does not take into consideration the error according to the range of our target variable and we would have to analyze the range to determine if the score is good or not. Normalized RMSE takes into consideration our Y range and outputs a decimal between 0-1 and the closer it is to 0 the better the model is. Here's how we implemented it…

```r
normalized_rmse = function(pred, real){ #Normalized Root Mean Squared Error
  error = real - pred
  sqrt(mean(error^2))/(max(real) - min(real)) }
```

MLR: Multiple Linear Regression

- **Training and testing…**

```r
set.seed(123)
mlr_model <- lm(y ~ ., data = norm_train_data) # Training...
mlr_predictions <- predict(mlr_model, newdata = test_x)  # Testing...
mlr_rmse <- normalized_rmse(mlr_predictions, test_y)
print(mlr_rmse)
```

```
## [1] 0.1529321
```

Random Forest

- **Training…**

```r
library(caret)
set.seed(123)
train_control <- trainControl(method = "cv", number = 5) # Performing cross validation
rf_model <- train( # Training the Random Forest model
  y ~ .,
  data = norm_train_data,
  method = "rf",
  trControl = train_control,
  ntree = 500)  # Number of trees in the forest
print(rf_model) # Printing metrics scores
```

```
## Random Forest
##
## 972 samples
##  21 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 778, 777, 777, 779, 777
## Resampling results across tuning parameters:
```

```
##
##   mtry  RMSE       Rsquared   MAE
##    2    0.4377012  0.9129451  0.24195517
##   11    0.1213881  0.9875847  0.04769325
##   21    0.1197615  0.9864714  0.04728059
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 21.
```

- **Testing...**

```r
set.seed(123)
rf_predictions <- predict(rf_model, newdata = norm_test_data)
rf_rmse = normalized_rmse(rf_predictions, test_y)
print(rf_rmse)
```

```
## [1] 0.06032038
```

Decision Tree

- **Training...**

```r
library(rpart)
set.seed(123)
tree_model <- rpart(
  formula = y ~ .,
  data = norm_train_data,
  method = "anova", # "anova" to specify regression
  control = rpart.control(cp = 0.01))  # Control parameters (complexity parameter)
```

- **Testing...**

```r
set.seed(123)
dt_predictions <- predict(tree_model, newdata = norm_test_data)
dt_rmse = normalized_rmse(dt_predictions, test_y)
print(dt_rmse)
```

```
## [1] 0.09560546
```

KNN: K-Nearest Neighbor

- **Training...**

```r
library(caret)
set.seed(123)
knn_model <- train( # Training the KNN regression model
  x = train_x,
  y = train_y,
  method = "knn",
  trControl = trainControl(method = "cv", number = 5), # Cross-validation settings
  tuneGrid = expand.grid(k = c(1, 3, 5)))  # Hyperparameter grid (k)
```

- **Testing...**

```r
set.seed(123)
knn_predictions <- predict(knn_model, newdata = test_x)
knn_rmse = normalized_rmse(knn_predictions, test_y)
print(knn_rmse)
```

```
## [1] 0.09896978
```

SVM Regression

- Now we will be trying the regression version of the support vector machines we will be implementing grid search for hyper-parameter tuning...
- **Training with hyper-parameter tuning...**

```
library(e1071)
set.seed(45)
#Tuning the SVM model
tuned_svm=tune(svm, y~., data=norm_train_data, ranges=list(epsilon=seq(0,1,0.1), cost=c(0.01, 0.1, 1, 10, 100),
                                                           kernal = c("linear", "polynomial", "radial basis", "sigmoid")))
print(tuned_svm) # performance measure = MSE
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  epsilon cost kernal
##      0.2  100 linear
##
## - best performance: 0.1069891
```

- **Training and validation accuracy**

```
best_tuned_svm = tuned_svm$best.model
svm_train_err = normalized_rmse(predict(best_tuned_svm, train_x), train_y)
print(svm_train_err)
```

```
## [1] 0.03864575
```

- **Testing...**

```
svm_predictions = predict(best_tuned_svm, test_x)
svm_rmse = normalized_rmse(svm_predictions, test_y)
print(svm_rmse)
```

```
## [1] 0.06845775
```

ALL IN ONE: Blender

- In this section we are going to be implementing the ensemble **stacking** method involving three base learners and a meta learner which learns from the predictions of the base learners...
- *LEARNER-1: Gradient Boosting Machine*

```
library(h2o)
h2o.init() # Initiating h20 environment
set.seed(42)
nfolds <- 10 # Number of CV folds (to generate level-one data for stacking)
gbm <- h2o.gbm(x = names(train_x), # Gradient Boosting Machine
               y = "y",
               training_frame=as.h2o(norm_train_data),
               nfolds = nfolds,
               keep_cross_validation_predictions = TRUE,
               seed = 42)
```

- *LEARNER-2: Generalized Linear Model*

```r
set.seed(42)
glm <- h2o.glm(x = names(train_x), # Generalized Linear Model
               y = "y",
               training_frame=as.h2o(norm_train_data),
               nfolds = nfolds,
               keep_cross_validation_predictions = TRUE,
               seed = 42)
```

- *LEARNER-3: Fully connected Neural Network*

```r
set.seed(42)
dl<- h2o.deeplearning(x = names(train_x), # Fully Connected Neural Network
                      y = "y",
                      training_frame=as.h2o(norm_train_data),
                      nfolds = nfolds,
                      keep_cross_validation_predictions = TRUE,
                      seed = 42)
```

- **META LEARNER: Random Forest**

```r
set.seed(42)
# Train a stacked Random forest ensemble using the previously trained models
ensemble <- h2o.stackedEnsemble(x = names(train_x),
                                y = "y",
                                training_frame=as.h2o(norm_train_data),
                                metalearner_algorithm="drf",
                                metalearner_nfolds = 30,
                                base_models = list(gbm, glm, dl),
                                metalearner_params = list(ntrees = 100, keep_cross_validation_predictions = TRUE),
                                seed = 42)
```

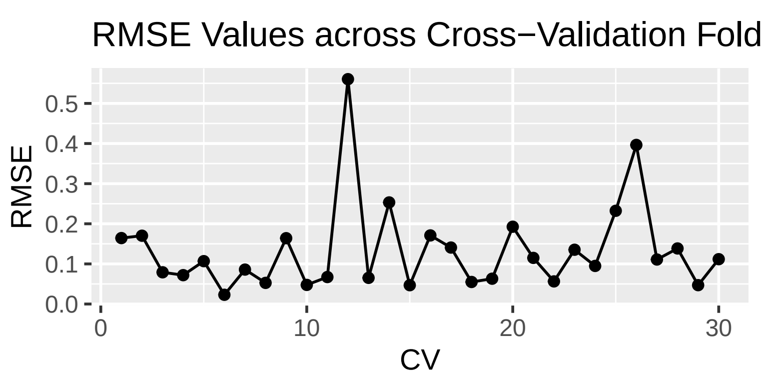
- **meta learner training metrics. . .**

```r
ensemble@model$training_metrics
```

```
## H2ORegressionMetrics: stackedensemble
## ** Reported on training data. **
##
## MSE:  0.007774602
## RMSE:  0.0881737
## MAE:  0.03754815
## RMSLE:  0.04059596
## Mean Residual Deviance :  0.007774602
```

- **meta learner validation rmse curve. . .**

```r
rmse_df = as.data.frame(t(as.data.frame(ensemble@model$cross_validation_metrics_summary)[6,3:32]))
ggplot(rmse_df, aes(x = index(rmse_df), y = rmse)) +
  geom_line() +
  geom_point() +
  labs(x = "CV", y = "RMSE", title = "RMSE Values across Cross-Validation Folds")
```



- **Testing base learners performance...**

```r
set.seed(45)
h20_test = as.h2o(norm_test_data) # data to H2O data
gbm_perf <- h2o.performance(gbm, newdata = h20_test)
glm_perf <- h2o.performance(glm, newdata = h20_test)
dl_perf <- h2o.performance(dl, newdata = h20_test)
print(gbm_perf)
print(glm_perf)
print(dl_perf)
```

```
##   |                                                                              |
## H2ORegressionMetrics: gbm
##
## MSE:  0.08580718
## RMSE:  0.2929286
## MAE:  0.09201284
## RMSLE:  0.1049959
## Mean Residual Deviance :  0.08580718
##
## H2ORegressionMetrics: glm
##
## MSE:  0.5224682
## RMSE:  0.7228196
## MAE:  0.5215098
## RMSLE:  NaN
## Mean Residual Deviance :  0.5224682
## R^2 :  0.52373
## Null Deviance :119.1995
## Null D.o.F. :107
## Residual Deviance :56.42657
## Residual D.o.F. :88
## AIC :278.3781
##
## H2ORegressionMetrics: deeplearning
##
## MSE:  0.08850366
## RMSE:  0.2974956
## MAE:  0.1749232
## RMSLE:  0.1888018
## Mean Residual Deviance :  0.08850366
```

- **meta learner performance on test data...**

```r
set.seed(45)
meta_perf <- h2o.performance(ensemble, newdata = h20_test)
print(meta_perf)
```

```
## H2ORegressionMetrics: stackedensemble
##
## MSE:  0.02367752
## RMSE:  0.153875
## MAE:  0.05941173
## RMSLE:  0.05632648
## Mean Residual Deviance :  0.02367752
```

Test Results Summary

- Now to sum up all the results of our project we are going to make a table with all the models' RMSE scores on the test data and compare them...

```r
ens_rmse = meta_perf@metrics$RMSE/(max(test_y) - min(test_y))
rmse_df = data.frame("Model" = c("Multiple Linear Regression", "Random Forest", "Decision Tree", "K-Nearest Neighbors", "Su
                "Normalized RMSE" = c(mlr_rmse, rf_rmse, dt_rmse, knn_rmse, svm_rmse, ens_rmse), check.names=FALSE)
rmse_df <- rmse_df[order(rmse_df$`Normalized RMSE`, decreasing = TRUE), ]
print(rmse_df)
```

```
##                                Model Normalized RMSE
## 1        Multiple Linear Regression      0.15293210
## 4               K-Nearest Neighbors      0.09896978
## 3                     Decision Tree      0.09560546
## 5 Support Vector Machine Regression      0.06845775
## 2                     Random Forest      0.06032038
## 6                Stacking (Blender)      0.03253521
```

Table 2: Best Performing Model

Model	*Normalized RMSE*
Stacking (Blender)	0.025 ± 0.005

Conclusion

As shown in the results summary, the answer to our research question is…

"The best out of all the tested machine learning algorithm is the stacking method called Blender more commonly"

which makes sense as it aligns with the initial assumption of the ensemble method which imply that combining more than one learner produces better predictions.