20596 Machine Learning
Final Data Challenge Report: High Water
by So Mi PARK_3250632

Summary

In the high water dataset, bagging of classification tree and random forest models performed equally best with 100% accuracy level with the train dataset and 0.12 evaluation score with the test dataset.

Introduction

The high water dataset comprises 37 input variables and it can be narrowed down to 13 ignoring the locational factor. The output variable y gas y=2 (Tide height of more than 90 cm) if high tide was observed exactly after 6 hours following the observation and y=1 otherwise. The objective of this prediction model is to predict with the given variables if high tide will occur 6 hours prior to the actual high tide so MOSE (Modulo Sperimentale Elettromeccanico) can be activated to prevent flooding in Venice.

Data Exploration

The input variables: average tide level, average wind direction, average wind speed, max wind speed, humidity, solar radiation, air temperature, water temperature, pressure, significant wave length, max wave height, and rain level. These are collected from 7 different locations. Below is a table that describes the locational data of the variables collected.

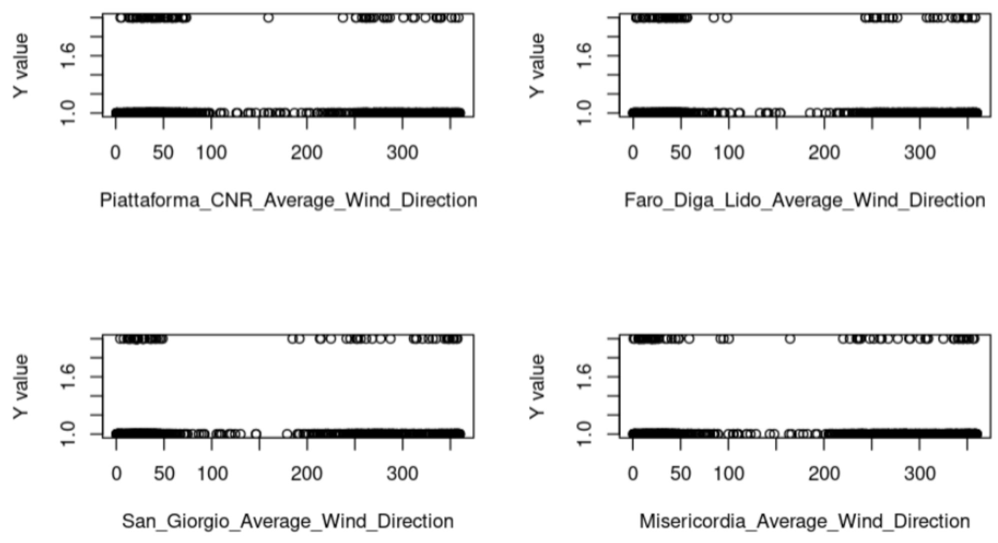
<Table 1 Locations which variables were collected>

Location	Average Tide Level	Average WInd Direction	Average Wind Speed	Max Wind Speed	Humidity	Solar Radiation	Air Temperature	Water Temperature	Pressure	Significant Wave Height	Max Wave Height	Rain Level
Piattaforma	o	o	o	o	o	o	o	o	o	o	o	o
Diga Sud Lido	o	o	o	o								
San Giorgio		o	o	o	o	o	o					
Punta Salute	o							o				
Misericordia	o	o	o	o						o	o	
Palazzo Cavalli					o	o	o		o			o
Burano	o											

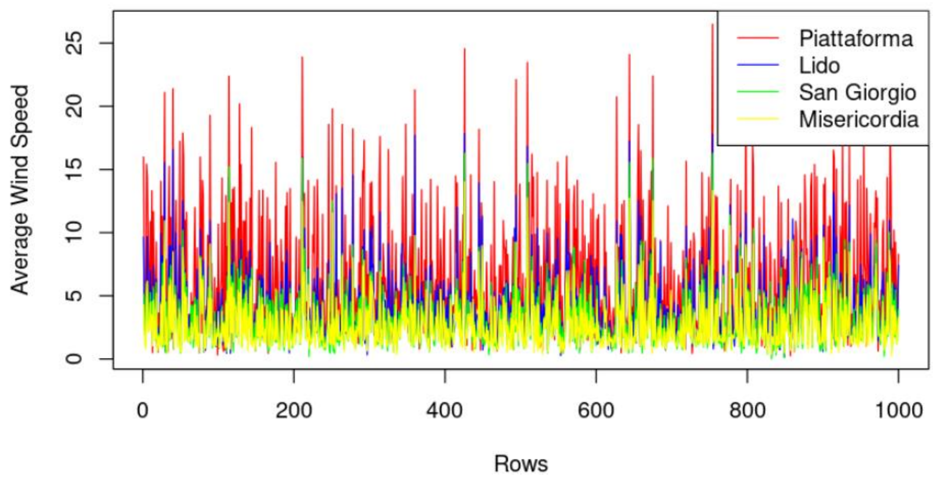
There are noticeable input variables when exploring the dataset. The average wind direction has a pattern for output value 2. Since a wind direction is a circular data, 0 and 360 degrees are equal. According to <Plot 1> below, it looks like when there is a high tide which y variable = 2, the average wind direction is between 1-100 degrees and 200-300 degrees. In <Plot 2>, the average wind speed shows a decreasing pattern as it approaches the land. For example, Piattaforma, the observation located in the furthest from Venice center has a pattern of highest speed and Misericordia the furthest observation center located in Venice has lowest wind speed. The order of average wind speed moves from Piattaforma, Lido, San Giorgio, and Misericordia.

In addition, there is high correlation between the same input variables collected from different locations showing multicollinearity. For example, the average tide level collected from Piattaforma, Diga Sud Lido, Punta Salute, Misericordia, Burano shows correlation of more than 75% and some nearly 99% as shown in <Table 1>.

<Plot 1 Train dataset: Average wind direction pattern per Y value>



<Plot 2 Train dataset: Average wind speed pattern with difference locations>



<Table 1 Correlation matrix of average tide levels from different locations>

	Piattaforma	DS_Lido	P_Salute	Misericordia	Burano
Piattaforma	1	0.9982674119	0.894606452	0.8620554245	0.7984839788
DS_Lido	0.9982674119	1	0.8925546204	0.8604597514	0.7964574253
P_Salute	0.894606452	0.8925546204	1	0.9939021986	0.9660452404
Misericordia	0.8620554245	0.8604597514	0.9939021986	1	0.9869420598
Burano	0.7984839788	0.7964574253	0.9660452404	0.9869420598	1

Missing data

There is missing data for 8 columns in the train dataset. 24 missing data for San_Giorgio_Average_Wind_Direction, 22 missing data for San_Giorgio_Average_Wind_Speed, San_Giorgio_Max_Wind_Speed, San_Giorgio_Humidity, Palazzo_Cavalli_Solar_Radiation, San_Giorgio_Solar_Radiation, San_Giorgio_Air_Temperature, and 456 missing data for Piattaforma_CNR_Solar_Radiation. If we remove the missing data, 612 out of 1000 rows will be deleted.

Therefore, missing data was filled-in by using Multivariate Imputation by Chained Equation (MICE). ‘Pmm’, ‘cart’, ’lasso.norm’ Imputation methods were tested and chose ‘cart’ and ‘lasso.norm’ methods considering the patterns discovered in the data exploration step.

<Code 1 Handling missing data with MICE imputation>

```r
mice_imputed <- data.frame(
  sg_avgwinddir_imputed_cart = complete(mice(train_x, method = "cart"))$San_Giorgio_Average_Wind_Direction,
  sg_avgwindspd_imputed_lasso = complete(mice(train_x, method = "lasso.norm"))$San_Giorgio_Average_Wind_Speed,
  sg_maxwindspd_imputed_lasso = complete(mice(train_x, method = "lasso.norm"))$San_Giorgio_Max_Wind_Speed,
  sg_humid_imputed_cart = complete(mice(train_x, method = "cart"))$San_Giorgio_Humidity,
  Piatta_solar_imputed_cart = complete(mice(train_x, method = "cart"))$Piattaforma_CNR_Solar_Radiation,
  sg_solar_SG_imputed_cart = complete(mice(train_x, method = "cart"))$San_Giorgio_Solar_Radiation,
  Palazzo_solar_imputed_cart = complete(mice(train_x, method = "cart"))$Palazzo_Cavalli_Solar_Radiation,
  sg_Airtemp_Palazzo_imputed_cart = complete(mice(train_x, method = "cart"))$San_Giorgio_Air_Temperature
)
```

Treating average wind direction data

As mentioned in the data exploration, average wind direction data is a circular data meaning 0 degree and 360 degrees are the same so they should not be treated as a regular numerical value. In order to avoid misleading the data prediction, wind direction data has been updated to radian data. By doing this, it normalizes the data. The formula used is radian = degrees * π / 180.

<Code 2 Handling directional data>

```r
train_x$Faro_Diga_Lido_Average_Wind_Direction <- train_x$Faro_Diga_Lido_Average_Wind_Direction / 180 * pi
train_x$Misericordia_Average_Wind_Direction <- train_x$Misericordia_Average_Wind_Direction / 180 * pi
train_x$San_Giorgio_Average_Wind_Direction <- train_x$San_Giorgio_Average_Wind_Direction / 180 * pi
train_x$Piattaforma_CNR_Average_Wind_Direction <- train_x$Piattaforma_CNR_Average_Wind_Direction / 180 * pi
```

Normalizing input variables

Rest of the columns that were untreated by the average wind direction radian formula is normalized through the Min-Max Scaling method so the values range from 0 to 1. It will scale the negative data in some input variables and also different scale sizes. Also, it will reduce the issue of outliers. Then normalized input variables and the wind direction data which has been normalized through radian was merged back together as an one dataframe.

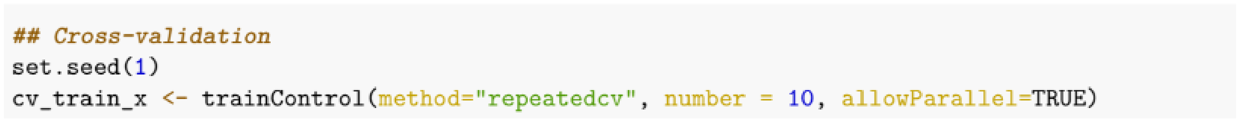
<Code 3 Normalization of input variables>

```r
# Create a new data frame with the selected columns
test_selected <- test[, -(8:11)]

# Normalize the data
processtest <- preProcess(as.data.frame(test_selected), method=c("range"))
test_norm_scale_selected <- predict(processtest, as.data.frame(test_selected))

# Merge the normalized columns back into the original data frame
test_norm <- cbind(test_norm_scale_selected, test[,8:11])
```

Cross-validation method

Since there are only 1000 rows for the train test dataset, a cross-validation method is used to get a more accurate estimate of the data on the new data. It divides the dataset into multiple folds and uses each fold for training and testing. In cross-validation, ‘repeatedcv’ method was used in which the process is repeated multiple times with different random divisions of the data into folds. Since LOOCV is too computing-power intensive, ‘repeatedcv’ method is used which is less intensive but repeats the folding process more than regular cross-validation.

<Code 4 Cross-validation with ‘repeatedcv’ method>



```
## Cross-validation
set.seed(1)
cv_train_x <- trainControl(method="repeatedcv", number = 10, allowParallel=TRUE)
```

Model Selection

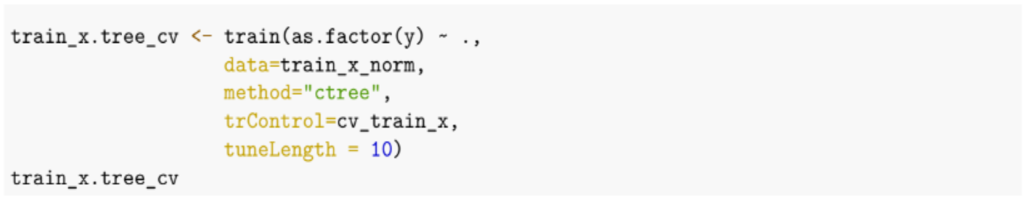
There are several methods and models to apply when the input variables are highly correlated among the location of the data collection showing multicollinearity. Principal Component Analysis, Ridge and Lasso regularization, and other methods can be considered. However, a tree-based model performs well in this case since it’s a classification problem with high correlation among columns. The advantage of tree-based models is that it can capture nonlinear relationships between the input features and the output variable.

Among tree-based models, 4 models are considered. They are a single classification tree, bagging of classification tree, random forest for classification tree, and gradient boosting.

1. Classification tree

It is a basic tree-based decision model used for predicting categorical or discrete target variables. In a classification tree model, the target variable is a categorical variable that is split into different classes or categories based on the values of the input features. The decision tree is built by recursively splitting the data into subsets based on the values of the input features. At each split, the feature that provides the most information gain about the target variable is selected as the split criterion. This model is expected to have the least performance compared to other tree models combined with boosting or bagging.

<Code 5 Training classification tree>



```
train_x.tree_cv <- train(as.factor(y) ~ .,
                         data=train_x_norm,
                         method="ctree",
                         trControl=cv_train_x,
                         tuneLength = 10)
train_x.tree_cv
```

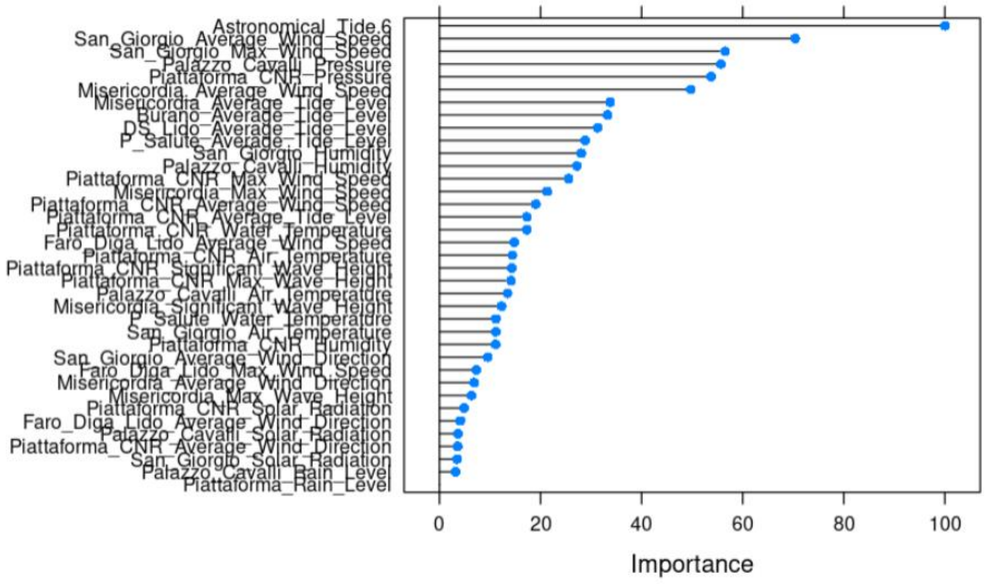
2. Bagging of classification tree

Bagging (Bootstrap Aggregation) trains multiple decision tree models on different subsets of the training data and combines their predictions to make a final prediction. It provides the most commonly occurring class among the predictions. With bagging, the most important input variables are Astronomical_Tide.6, San_Giorgio_Average_Wind_Speed, San_Giorgio_Max_Wind_Speed and the least important input variables are San_Giorgio_Solar Radiation, Palazzo_Cavalli_Rain_Level, and Piattaforma_Rain_Level.

<Code 6 Training bagging classification tree>

```
train_x.bagg_cv <- train(as.factor(y) ~ .,
                         data=train_x_norm,
                         method="treebag",
                         trControl=cv_train_x,
                         importance=TRUE)

train_x.bagg_cv
```

<Plot 3 Importance of features with bagging>



3. Random forest

Random forest solves the issue with bagged trees where they will be highly correlated since most of the trees will use the strong input in the top split. It randomly selects the input variables to be considered at each split to reduce the sampling correlation between the trees.

<Code 7 Training random forest>

```
train_x.rf_cv <- train(as.factor(y) ~ .,
                       data=train_x_norm,
                       method="rf",
                       trControl=cv_train_x,
                       importance=TRUE)
train_x.rf_cv
```

4. Gradient boosting

It trains a sequence of models, each of which corrects the errors of the previous model. It combines multiple trees and these trees are grown sequentially using weighted versions of the training data.

<Code 8 Training gradient boosting>

```
train_x.gbm <- train(as.factor(y) ~ .,
                     data=train_x_norm,
                     method="gbm",
                     verbose=F,
                     trControl=cv_train_x)
train_x.gbm
```

Model results with train data

With the train data, bagging of classification trees and random forest predicted the training set well with a 100% accuracy.
Model 1: Classification tree showed accuracy of 96.1%. Model 2: Bagging of classification tree showed accuracy of 100%. Model 3: Random forest showed accuracy of 100%. Model 4: Gradient boosting showed accuracy of 98.9%.

<Table 2 Confusion matrix of 4 tree models using train dataset>

Classification tree (96.1%)

```
          Reference
Prediction   0   1
         0 906  15
         1  24  55
```

Bagging of classification tree (100%)

```
          Reference
Prediction   0   1
         0 921   0
         1   0  79
```

Random forest (100%)

```
          Reference
Prediction   0   1
         0 921   0
         1   0  79
```

Gradient boosting (98.9%)

```
          Reference
Prediction   0   1
         0 921   0
         1  11  68
```

Model results with test data

In the test dataset, bagging of classification tree and random forest also performed well as in the train dataset.

<Table 3 Evaluation score of 4 tree models using test dataset>

Model	Classification tree	Bagging of classification tree	Random forest	Gradient boosting
Evaluation score	0.19	0.12	0.12	0.15