University of Pannonia

Faculty of Information Technology

Department of Computer Science and System Technology

MSc in Computer Science Engineering

INDEPENDENT LABORATORY WORK

A simulation study for comparison of different classification methods in terms of robustness for handling missing covariates

Péter Csaba Tóth

Supervisor: Ágnes Vathy-Fogarassy, PhD

2020

1. Introduction

From the beginning of the 21th century, Information Technology has been a rapidly developing field. However thanks to this, more and more data is being produced minute by minute. To be able to store this large amount of data, data centers have been built from ground up, and a new branch of Information Technology has emerged, which is called Data Science. It focuses on analyzing the generated data and on giving insights.

By analyzing the stored data we can find new answers for unsolved problems, which can be crucial for the given enterprise, or for humanity. Thanks to the establishment of Data Science, long-forgotten areas, such as Machine Learning, and Deep Learning have been relooked. According to well known researchers and popular businessmen in the industry, further development of these previously mentioned areas could mean the next step for Information Technology. Nowadays computers have huge computing performance. By using this, humanity can reach more and more breakthroughs every year.

Machine Learning is a subset of artificial intelligence. It gives the ability to train mathematical models, algorithms to find new, previously unrecognized patterns in the given data. Moreover, by using the trained model on new data, we can predict outcomes, and make decisions. To further improve the model predictive ability we need to feed more data in it, and use more computing power. This will lead to a better performing model, which will make better decisions, and precise predictions. Unfortunately, computing powers, namely GPU powers are fairly expensive nowadays, and only big companies have the privilege to acquire and store big data, and to use the best GPUs in the world. However, every year, tech companies release new, better performing GPUs with affordable price tags. Machine Learning is actively being used by several big companies. In the future we will probably see more improvements and breakthroughs in this field. Data Science heavily relies on Machine Learning, Data Mining, and Data Analytics.

Machine Learning algorithms are greatly influenced by the property domain in which they work. The lack of an attribute can cause significant bias in the results. Therefore in this work, my aim is to create a tool which helps me run custom simulations on newly created datasets of self-made random dataset generator. Through the results of simulations, I try to conclude how each chosen Machine Learning model behaves in a changed environment/dataset.

2. Basic machine learning definitions

Machine learning is known for its versatility. All of its subfield requires deep understanding to later work in them. In this chapter I will introduce basic definitions which are required to understand before the upcoming chapters.

2.1. Supervised Learning

It is one of the subfields of Machine Learning. In Supervised Learning, both the input and output datasets are available for the model. It tries to learn a target function f on examples, which are characterized by covariates x and their target value $f(x)$. However the model can only train on the training dataset, because in the testing dataset there are not any output values. Only input values (features) are used at the testing phase for predicting the unknown outcomes. There are cases when the model performs poorly in the testing phase. We can say that our model reached an optimal state if it correctly predicted all the outcomes. Later in this chapter I will introduce how we can evaluate our model when we try to solve a classification problem.

2.2. Classification

Classification is a subtype of supervised learning. Its task is to differentiate each class labels from each other by learning a target function on the training dataset. Class labels code always distinct subgroups, and they are recorded most of the time with string values. They have to be converted to numeric values before giving it to an algorithm for training. Each class label gets their own numeric value, for example “apple” = 0. This step is called “label encoding” or “target encoding”. In classification there are two types of classification problems based on the number of class labels which are the following:

- Binary classification: we talk about binary classification problems when there can be only 2 different class labels, for example Dog - Cat, or 0 - 1.
- Multi-class classification: we talk about multi-class classification tasks when there can be more than 2 different class labels in our dataset.

There are a large number of classification algorithms available for usage. However each of them have their own advantages and drawbacks. Of course we can use every one of them on every dataset after the mandatory preprocessing steps. However, there is a possibility that the randomly chosen classification algorithm will not perform well on the given dataset.

Later in this chapter I will introduce some rule of thumbs for choosing the correct algorithm for the given task. Until then here are some popular classification algorithms:

- Logistic Regression
- Random Forest
- Gradient Boosted Trees
- k-nearest neighbours
- Support Vector Machine

2.3. Logistic regression

Logistic regression is a kind of classification model that predicts whether something is True or False. It fits an S shaped logistic function, also called sigmoid function, on the data, that goes to 0 to 1, which can be seen on the following figure:

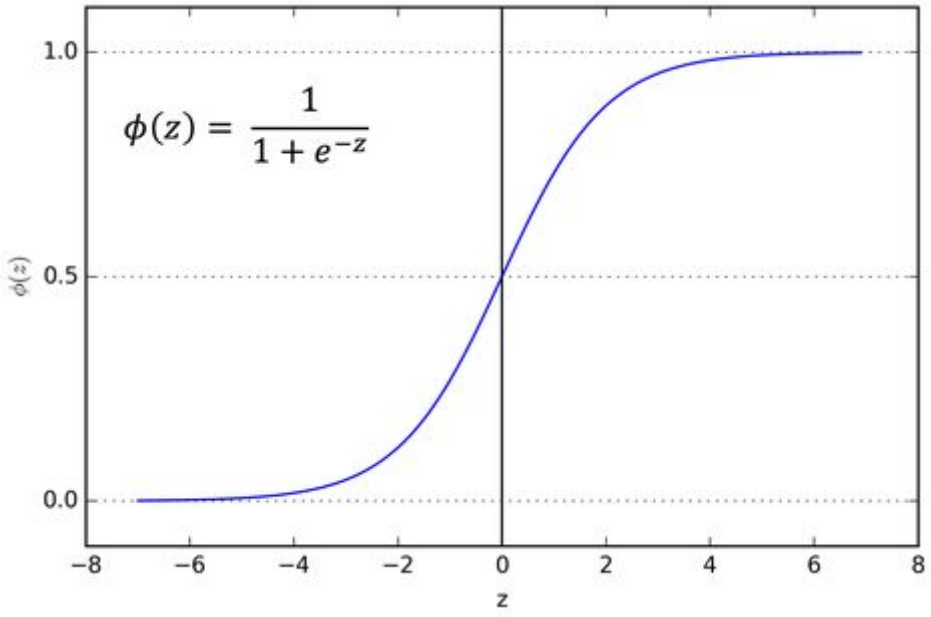

Figure 1.: Sigmoid function

The logistic regression's formal model is:

$$logit(p) = ln(\frac{p}{1-p})$$

By solving the previous formula for the probability p, can be represented by the estimated logistic regression equation:

$$\hat{p} = \frac{e^{(b_0 + b_1 \times x)}}{1 + e^{(b_0 + b_1 \times x)}}$$

where the $\hat{p}$ is the predicted outcome, b_0 is the coefficient or weight for the input (x), and b_1 is the bias. Every covariate has its corresponding coefficient, that the logistic regression model has to learn during the training phase. Most of the time, the learning is done by using Maximum-likelihood estimation.

2.5. Choosing the right algorithm

There are several great performing classification models. Almost every time, there is not a single machine learning model that can be used in every circumstance. Every dataset has its own nuisance. However there are some rules, which might be a good guidance for choosing the optimal model for the given problem:

1. Heterogeneity of the given data: model selection can be dependent on the structure of the input vector. Many algorithms, for example logistic regression, neural networks, and support vector machines (SVM) prefer the input vectors to be in scaled numerical form.
2. Linearity: many algorithms think that they can classify the outcomes by fitting a separated line on the data. These algorithms assume that the data trends follow a straight line. If this is the case, then we can say that the data is linear. Otherwise we say that the data is non-linear.
3. Dimensionality of features: When the number of features are high, machine learning models tend to perform worse, because they can not learn the input data as much as if there were less number of features. Although in some cases, Support Vector Machines might be a solution. However, most of the time we need to use dimensionality reduction, or feature selection techniques, before we give the data into the chosen model.

4. Usage of evaluation metrics: Albeit we can compare machine learning models based on how they perform on the training dataset by using accuracy metric. Unfortunately, this is not enough for deciding which algorithms are better. We need to use more metrics. In the following subchapter I will talk about evaluation metrics comprehensively.

These were only a few rules from many. In conclusion, before we fit the data on the training set, we need to observe and transform it.

2.6. Evaluation metrics

To interpret, train and to test the results of the model, we need to use evaluation metrics. These will help us understand how our model behaves. In machine learning there are many different metrics for classification, regression and also for clustering. At first glance people always want to use the most basic metric, the accuracy. However, it is known, the choice is not that simple. To understand how each metric works, we need to learn about its building blocks. In this work I will only show the most known classification metrics.

First of all, let me introduce the core parts of each classification metric. In supervised learning we can visualize how a classification model performs as a confusion matrix, also known as error matrix. It is a two by two matrix, which shows us how many true positive, true negative, false positive, false negative predictions we had during the model evaluation phase. The following figure will help us understand how a confusion matrix looks like:

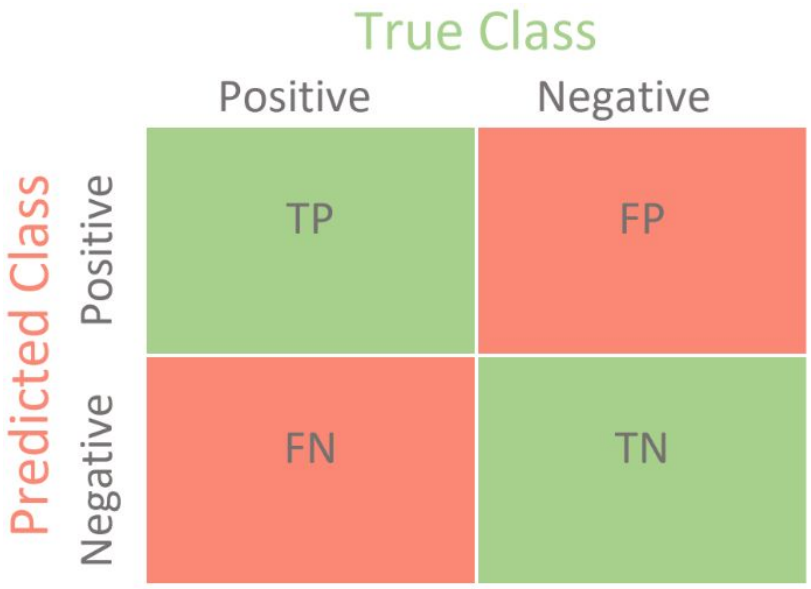

Figure 2.: Confusion matrix

As we can see in this figure there are four types of outcome:

- True Positive: When the classification model correctly predicts the positive output.
- True Negative: When the classification model correctly predicts the negative output.
- False Positive: When the classification model incorrectly predicts the positive output.
- False Negative: When the classification model incorrectly predicts the negative output.

These are the four main parts of each classification metric. These will affect how each model will perform on the evaluation dataset. Let us move to the introduction of the main metrics. Accuracy, it is the most known metric. It shows how many outcomes get correctly classified by the classification model. Accuracy can be calculated by the following formula [1]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

It is usually a good practice to use this metric to get a better understanding of the model story-telling ability. However, we must not rely on this on imbalanced datasets, because it will tell us half the story that our dataset has.

F1 score is more accurate than accuracy in most cases. It is used for measuring the test's accuracy. The greater the F1 score, the better the model's classifying ability is. Also it is often called the weighted average of precision and recall. F1 score formula is the following:

$$F1\ score = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

Precision shows how much percentage of the cases identified as positive, are correctly positive.

$$Precision = \frac{TP}{TP + FP}$$

Specificity is a ratio of how many cases were identified as true negative to actual negative cases.

$$Specificity = \frac{TN}{FP + TN}$$

Sensitivity (also known as recall) is a ratio of how many cases were identified as true positive to actual positive cases.

$$Sensitivity = \frac{TP}{FN + TP}$$

ROC (in other words Receiver Operating Characteristic) curve [2] is a curved line plot which plots the True Positive Ratio (alias sensitivity) against the True Negative Ratio (calculated as 1 - specificity) in different threshold values. It is often used when dealing with binary classification problems. It tells more stories about how the evaluation went, than the accuracy does. The greater the area under the ROC curve (AUC) the better the classifying ability of the model is. Following figure will be an example of ROC curve:

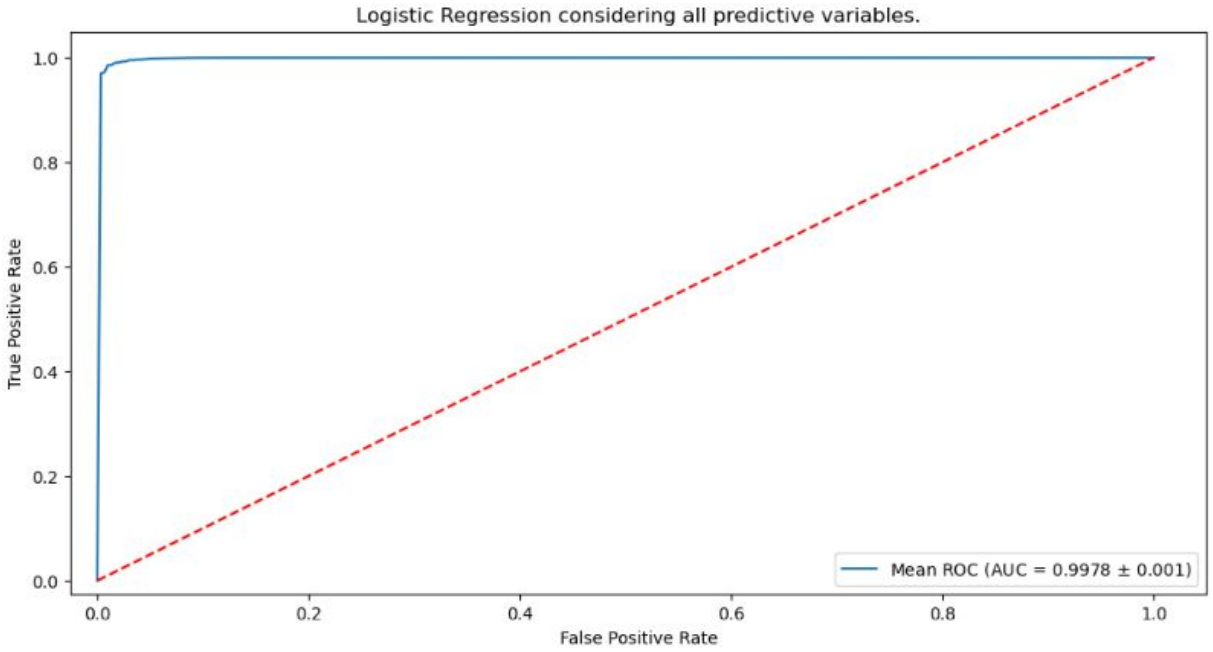

Figure 3.: ROC curve of one of the simulations.

3. Random dataset generation

At the beginning of the research the first problem was to find an ideal, benchmark dataset that can be used for simulation purposes. Benchmark dataset is a well-known dataset, which researchers use for testing their machine learning models performances. After they get the results of the model evaluation, they can easily compare them with previously established results of other researches on the benchmark dataset. Although, I have used one of the most used benchmark dataset, Census Income (Adult) dataset [3], to get a better understanding how each classification model, that I have chosen, behaves in a challenging dataset. Moreover I can not use the same dataset, for example in a 50 rounds simulation, because it would not give me a new, unforeseen result. This is why I created a random dataset generator.

3.1. Dataset structure

First of all the random dataset generator is built upon the programming language, C++. Also I have used Facebook's deep learning framework, Pytorch, for creating the columns and concatenating those into one dataset. The main idea of building this tool was to let the user create a custom dataset by only telling the tool the following ones, that can be seen in Figure 4.

```
1. RandomDatasetGenerator::ColumnDataType bern1{
2.   RandomDatasetGenerator::DistributionTypes::Bernoulli,
3.   {{"prob", 0.5}, {"weight", std::log(0.25)}}
4. };
```

Figure 4.: Creating one Bernoulli distributed column.

By using the previous code snippet, we can create one Bernoulli distributed column. However this is not the only type of column distribution that we can use. I have implemented many more distributions. In the following table (Figure 5.), we can see all the types of distributions that we can choose from during the dataset building process:

Name of distribution	*Parameters*
Binomial	number of trials, probability, weight
Bernoulli	probability, weight
Normal	mean, standard deviation, weight
Uniform Discrete	from, to, weight
Uniform Real	from, to, weight
Gamma	alpha, beta, weight

Figure 5.: Types of distribution can choose from

Every initialized column needs to be added into a columns storing list. We need to keep in mind that as many columns will be in the final dataset as many we have added to the previously mentioned list. Then later this list will be added to the main random dataset generator class. Besides initializing columns, we need to decide how many rows and how many datasets the dataset generator should create. Now we can move onto the core part of the generator.

3.2. Monte Carlo sampling

As I mentioned earlier, I have created this tool to help me create as many randomly sampled datasets as I would like to, which will be used later for simulation purposes. To achieve randomness during dataset generation, I needed a random sampler method. This is why I chose the Monte Carlo sampling method. The main idea of this method is to create values based on statistical analysis or repeated random sampling. Monte Carlo simulation can be seen as a set of random experiments in which the results of the experiments are not well known.

In C++, there are several ways to implement the Monte Carlo method. I chose the Mersenne Twister pseudo-random number generator [4]. In the following 6. Figure, we can see the code snippet which is used for creating a column by using a chosen distribution with the combination of Mersenne Twister generator.

```
std::mt19937 m_generator;
template<typename T>
torch::Tensor generateRandomValuesHelper(T &dist,
                                         const bool is_whole_number = true) {
  // Creating X type of distributed random numbers, and storing them in
distValues
  std::vector<double> distValues(m_rows);

  if (is_whole_number)
  {
     for (auto &elem : distValues) {
        elem = (int)(dist(m_generator));
     }
  }
  else
  {
     for (auto &elem : distValues) {
       elem = (dist(m_generator));
     }
   }
   // Converting the distValues vector into Tensor and returning it
   auto opts = torch::TensorOptions().dtype(torch::kFloat64);

   return torch::from_blob(distValues.data(), {static_cast<int>(m_rows), 1},
opts);
};
```

Figure 6.: Generation of random values for a chosen distribution by using Mersenne Twister sampler.

3.2. Binary target column creation

In this work, I have only researched binary classification's algorithms. Until this subchapter I have talked about how I custom columns for the dataset. However, if I would like to use the generated dataset for classification, I need to create an output or in other words, a target column. This can be achieved by using Logistic Regression's logit function. In Figure 7, we can see a portion of the function:

```
auto inverse_logit = [](double &p){
    return (std::exp(p) / (1 + std::exp(p)));
};

const double intercept = -1.5;

m_outcome_probabilities.reserve(m_rows);

// Get iterators for the m_features
const auto features_accessor = m_features.accessor<double, 2>();

// Calculating the row-by-row outcome's probability with inverse_logit
for(int i = 0; i < features_accessor.size(0); i++) {
   double probSum = 0.0;
   for(int j = 0; j < features_accessor.size(1); j++) {
      probSum = probSum + (features_accessor[i][j] * m_weights[j]);
   }
   auto logit = probSum + intercept;
   auto p = inverse_logit(logit);
   m_outcome_probabilities.push_back(p);
}
```

Figure 7.: Binary output probabilities by inverse logit function.

In the beginning of the code snippet we can see that I initialize an inverse_logit function which will be used in the main computing loop. Then I create a data accessor variable which is used for accessing the elements of the matrix. After this we have reached the main for loop which will be responsible to calculate each record's probability. After these steps I need to round the received probabilities to value 0 or value 1. I achieved this by using threshold value, as it can be seen in Figure 8.

```
// Calculating the binary outcomes
std::vector<double> binaryOutcome;
binaryOutcome.reserve(m_outcome_probabilities.size());

for(const auto &val: m_outcome_probabilities) {
  if (val < 0.5)
  {
    binaryOutcome.push_back(0.0);
  }
  else if (val >= 0.5)
  {
     binaryOutcome.push_back(1.0);
  }
}
```

Figure 8: Rounding probabilities to 0 or 1 by using a threshold.

4. Simulations

In the previous chapter I have talked about basic machine learning definitions which are crucial to know, and also about random dataset generation which let create custom datasets. In this chapter, I will talk about the main part of my work, which will glue together everything that I have talked about before.

4.1. Most known feature selection techniques for classification

In this subchapter, I will introduce widely used feature selection techniques. I think it is important to mention that there is no such thing as the best feature selection method. Every one of them has their own use cases. In instance, the main idea of backward elimination is to remove those features which are not going to play an important role in the prediction and evaluation phases. After we find them, we remove them from the dataset. By this, not only can we improve the model's predictive ability, but we can decrease the training and testing time, too. Furthermore we can avoid the curse of dimensionality.

Firstly, let me introduce the wrapper methods. The main idea is to train a classification or regression model on a given set of features, and after the training phase evaluate them in terms of a predefined accuracy measure or measures. Based on it we need to decide that we should add new features to the previous set of features or remove from it. These methods are very computationally hungry most of the time.

There are three main types of wrapper methods which should be familiar for most of the Data Scientists. They are the following:

- Backward elimination, is an iterative algorithm, which starts with a set that includes all the features. In every iteration it removes one feature that does not affect the outcome, which hopefully improves the model's predictive ability. This algorithm only stops when the predictive ability can not be improved furthermore.
- Recursive feature elimination (RFE) [5], is a greedy algorithm. In each iteration, it fits a machine learning model on the dataset. Following this, the algorithm will remove the weakest feature or features until the specified number of features is reached.

- Forward elimination is an iterative algorithm, which starts with an empty set. In every iteration it keeps adding a new feature to the set till those improve the predictive ability of the model.

On the other hand, there are filter methods, which heavily rely on the characteristics of the covariates. Most of the time these methods are only used before the model training begins. There are two types of filter methods: univariate and multivariate.

Univariate filter methods only work with one feature at a time. However, these algorithms only decide whether to take the feature or leave it based on a filter criteria. Most of the time it counts as a huge disadvantage because they can remove such a feature that can be a good predictive covariate. Among the univariate filter methods, the statistical & ranking filter methods are the most known, especially the following ones:

- Univariate ROC-AUC
- Chi-squared test
- Anova
- Mutual Information

Multivariate filter methods take into account all the features space during filtering. Due to their different approach, they tend to make better decisions. Correlation-based filter methods are the most-known ones amongst the multivariate filter methods, which are the following ones:

- Fast correlation based filter (FCBF) [6]
- Pearson correlation coefficient
- Spearman's rank correlation coefficient
- Kendall's rank correlation coefficient

In contrast to the wrapper methods, filter methods are computationally inexpensive. Also they are really useful for removing redundant, duplicated, or even unused features from the features' set.

4.2. Types of simulation

The aim of my work was to analyze how each covariate affects the model's performing ability on the test datasets. During my research I have learnt about several feature selection methods that I have talked about in the previous chapter. However they were not ideal for my needs. This is why I had to create custom methods that would tick all the checkboxes of the requirements.

First of all, I wanted to measure how much influence all the features had on the model performing ability. For this, I created a simulation which fits a machine learning model in on the whole generated random dataset. I named this simulation "Without column excluding method". Furthermore, I also wanted to see the influence of each variable separately. To achieve this, I needed to create another simulation. It is very similar to the previous one. However, in this one, I excluded one column at a time from the set of features. Then I fitted a model on the remaining dataset. At the end of the iteration I put back the excluded column. The simulation repeated the previous steps till it visited all the attributes of the dataset. I named this approach the "With column excluding method". These two simulations can be used on a large number of datasets. The whole "With column excluding method" can be seen in the following part:

1. For each dataset.
    1.1. In each iteration exclude one column from the given dataset
        1.1.1. Split the remaining data into features and target datasets.
        1.1.2. Split the features dataset into training and test datasets.
        1.1.3. Apply transformations (if there are any) on the given columns of the training and testing datasets.
        1.1.4. Run a machine learning model on the preprocessed training datasets and use a hyperparameter optimization method on them for finding the best possible parameters for the machine learning model.
        1.1.5. Test the trained model on the test datasets.
        1.1.6. Calculate evaluation metrics, for example, specificity, sensitivity, F1 score, accuracy, precision.
        1.1.7. Save the result dataset at the end of the iteration.
    1.2. End of the loop.

1.3. Save all the evaluation metrics corresponding to the dataset.

2. Repeat until there is no dataset left in the list.

4.3. Methodology of the research

During research I have run 5 scenarios. In each scenario I have run both the column excluding and the without column excluding simulations. For 4 scenarios, I have created roughly 200 datasets. Each dataset contains 1000 rows containing 10 feature columns. In the 5th scenario, I have used a popular benchmark dataset, which is called Census (Adult) Income dataset [7] to test my hypothesis. Furthermore, in every scenarios I have used the following machine learning algorithms:

- Logistic Regression with default parameters
- Random Forest with default parameters
- Random Forest by hyperparameter optimized for accuracy
- Random Forest by hyperparameter optimized for AUC value of the ROC analysis.

5. Results

5.1. Scenario 1

Scenario 1 was the starting point that led me to create 3 similar scenarios due to the number of resulting hypotheses. In this scenario the main idea was to create 50 Bernoulli distributed datasets, with the upcoming parameters, and to see how these parameters will affect the simulations' results.

- Column weights: Generation of weights starts from $log(0.25)$ until $log(2.5)$, by increment log(0.25).
- Probability of true: 0.5

After I run all the simulations, I got the following results, which were surprising to me. In the following figures, we can see how the Logistic Regression model performed during the column excluding simulation.

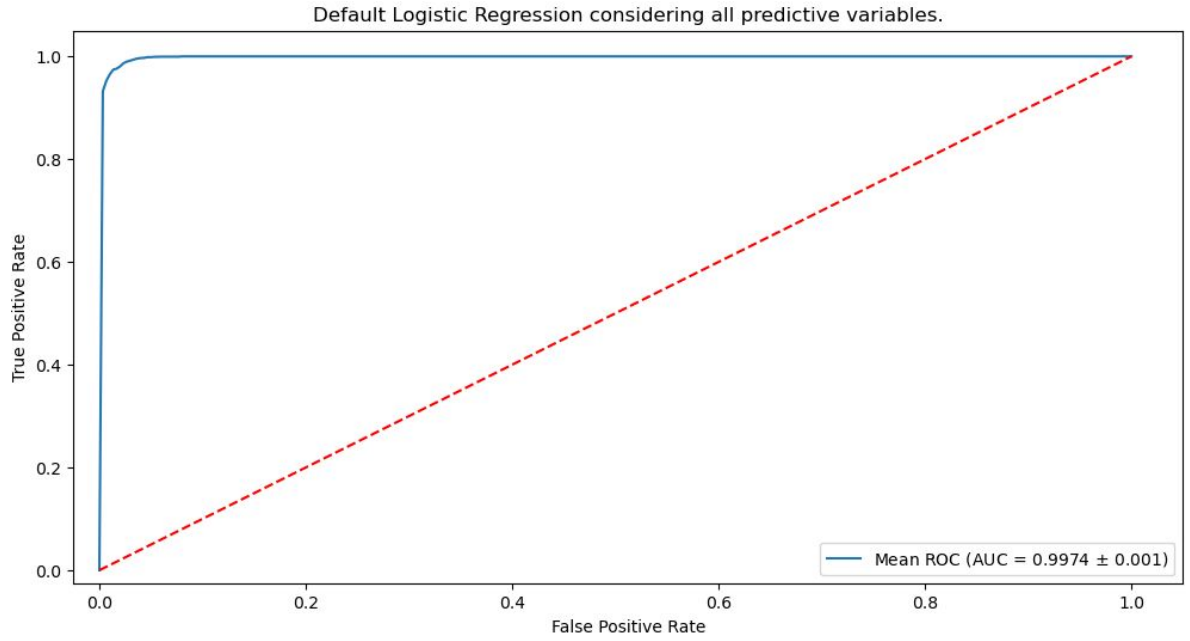

Figure 9.: Logistic Regression considering all predictive variables

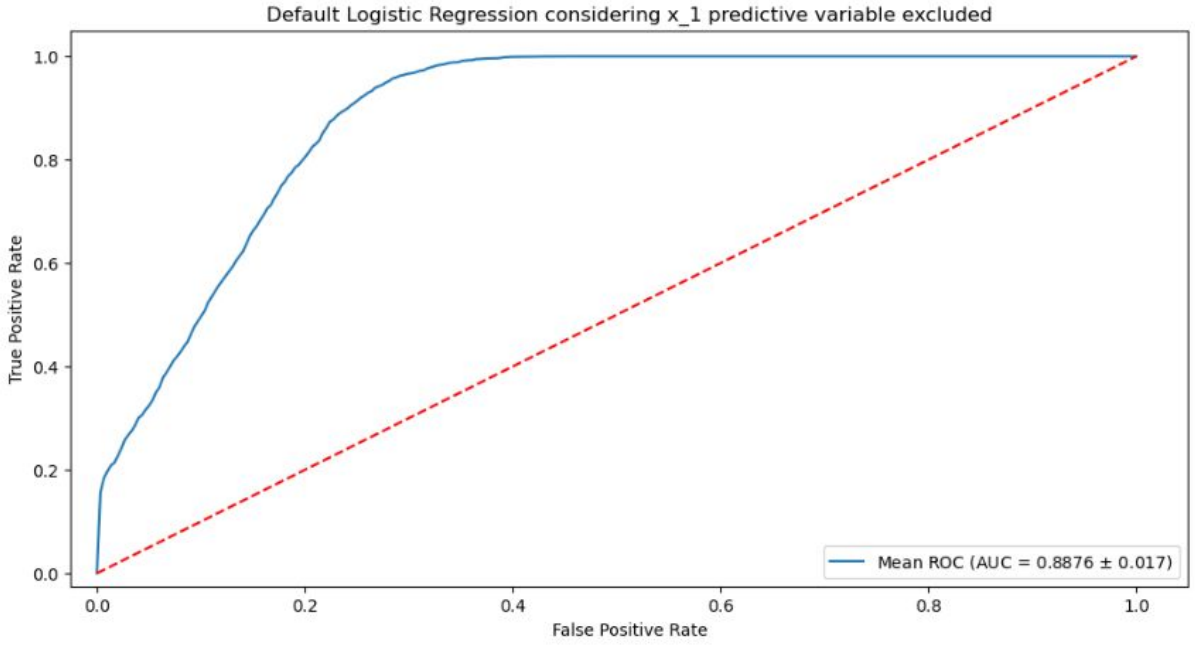

Figure 10.: Logistic Regression considering x_1 covariates excluded

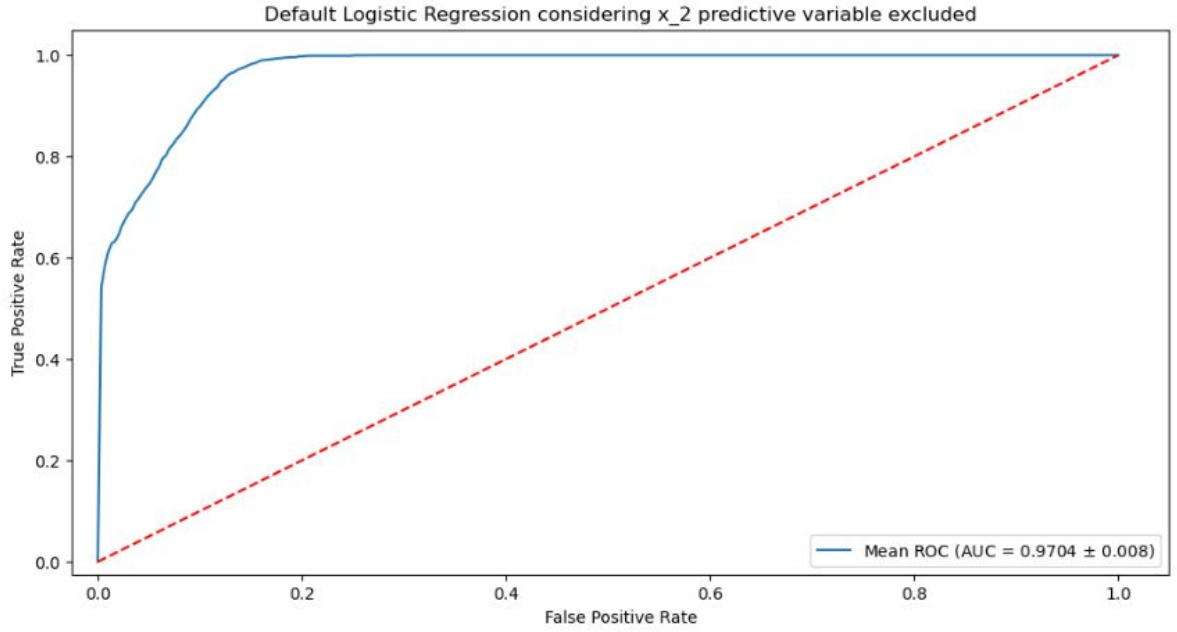

Figure 11.: Logistic Regression considering x_2 covariates excluded

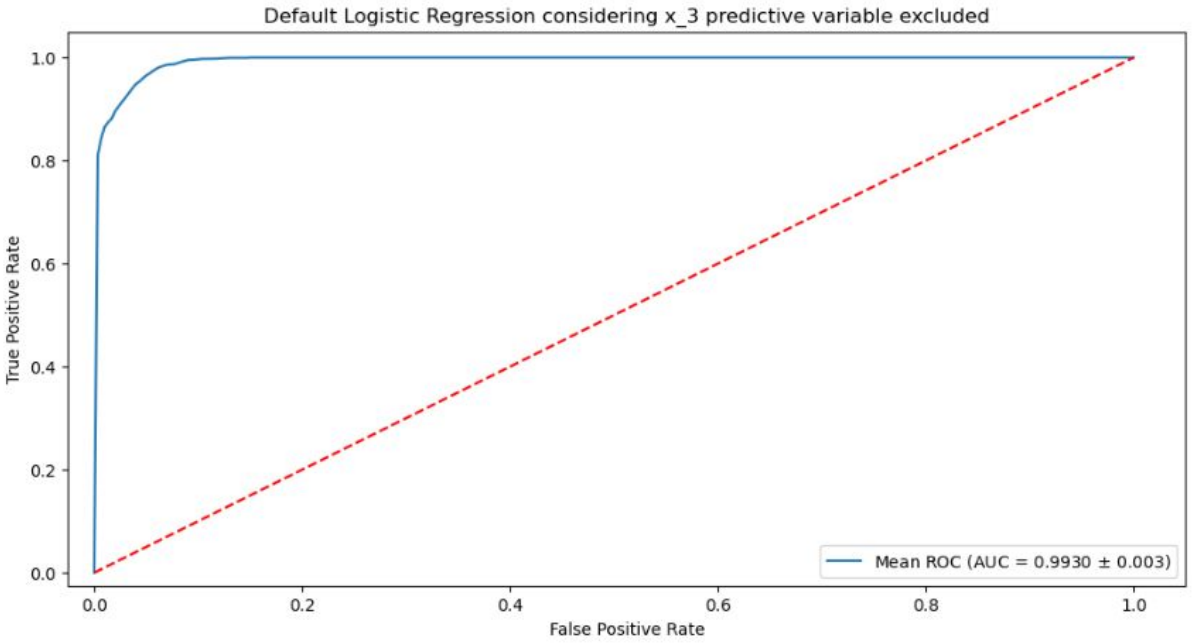

Figure 12.: Logistic Regression considering x_3 covariates excluded

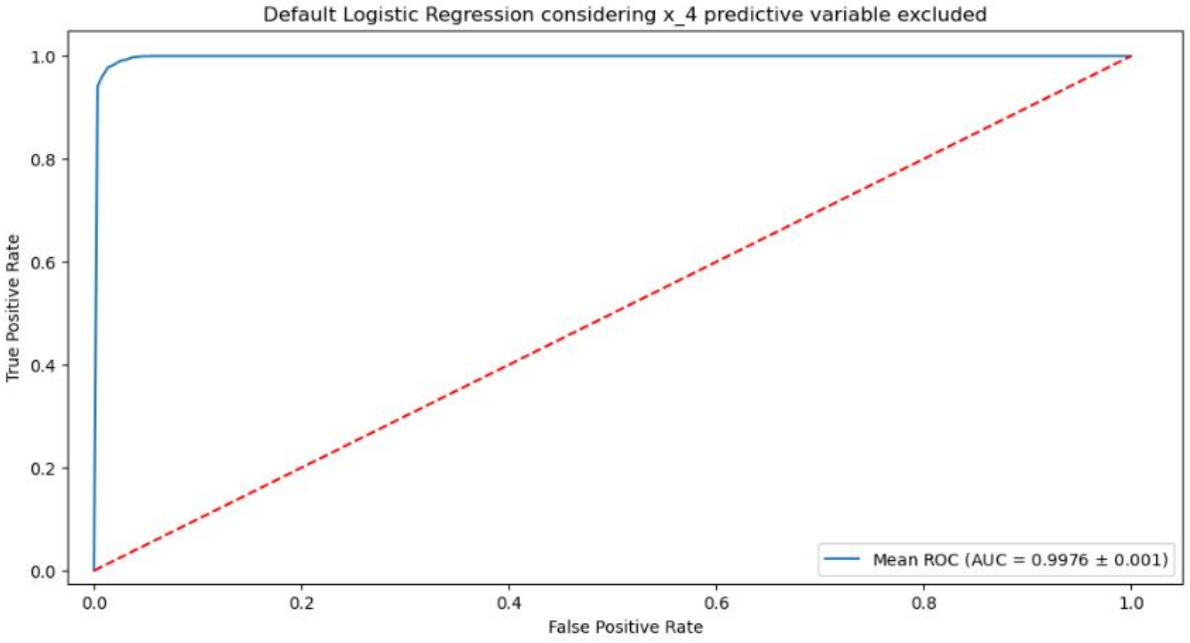

Figure 13.: Logistic Regression considering x_4 covariates excluded

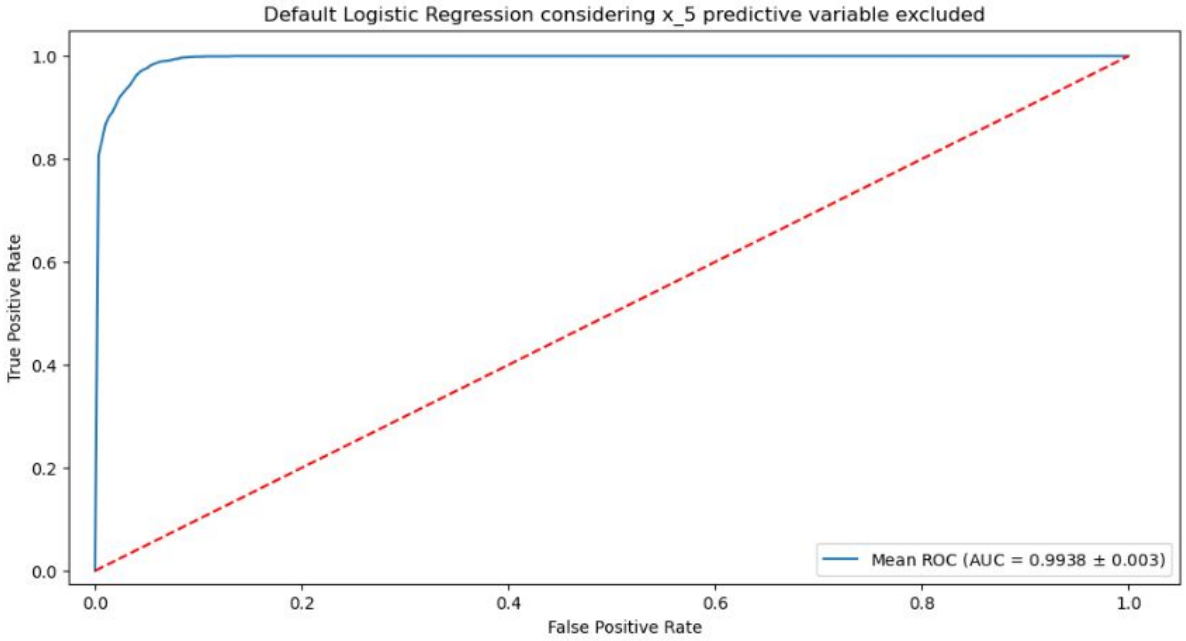

Figure 14.: Logistic Regression considering x_5 covariates excluded

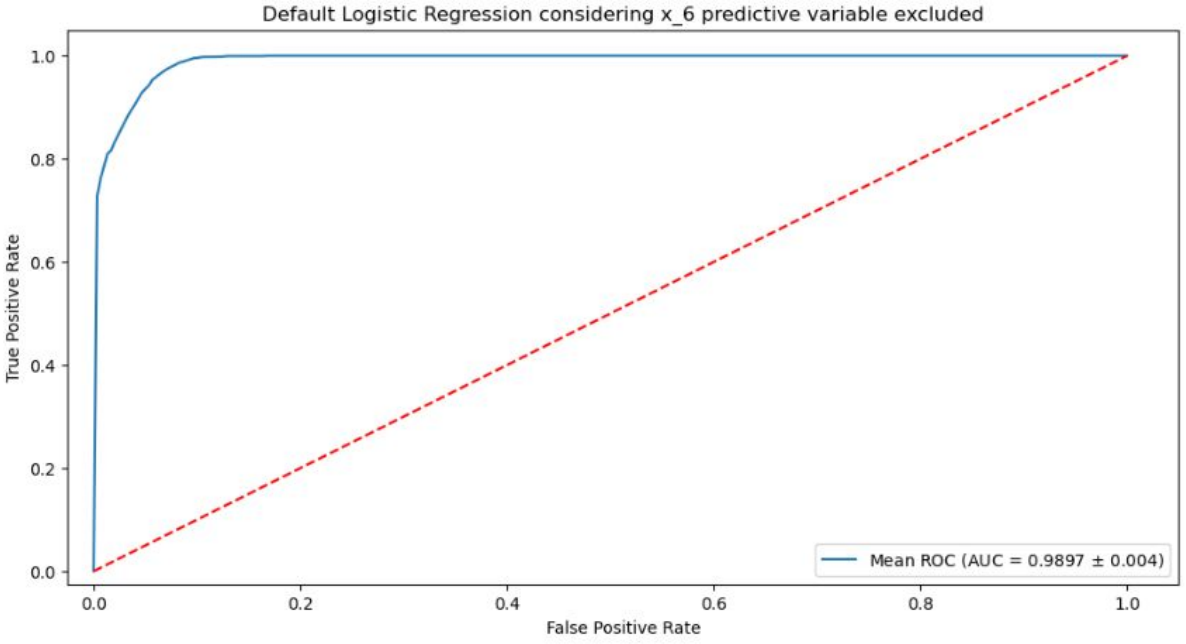

Figure 15.: Logistic Regression considering x_6 covariates excluded

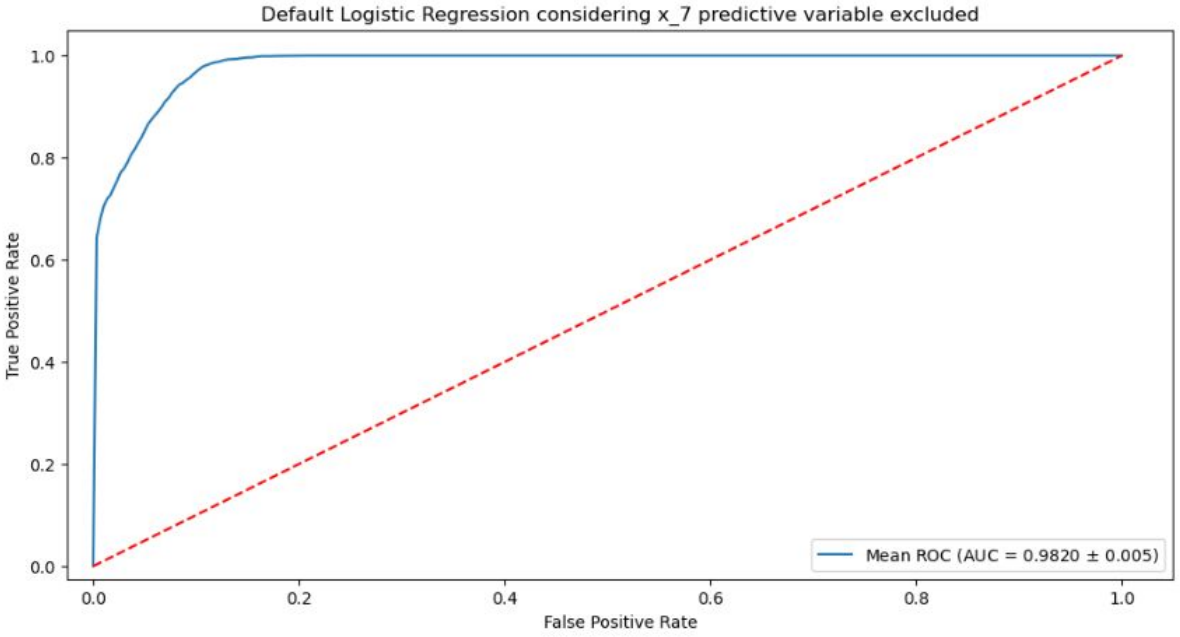

Figure 16.: Logistic Regression considering x_7 covariates excluded

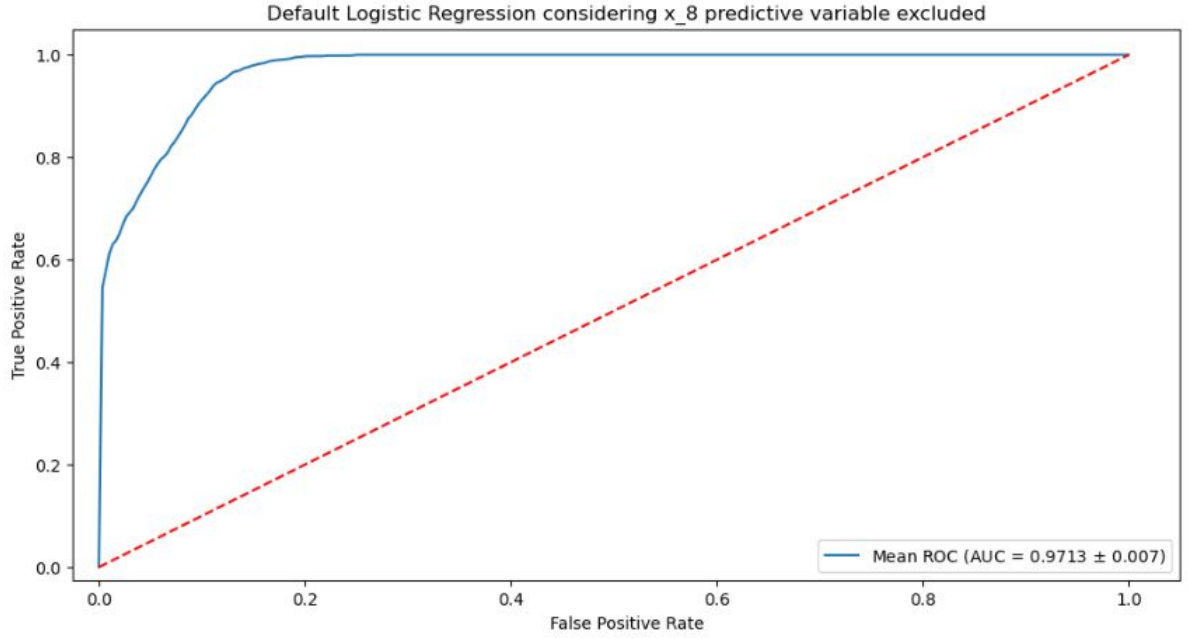

Figure 17.: Logistic Regression considering x_8 covariates excluded

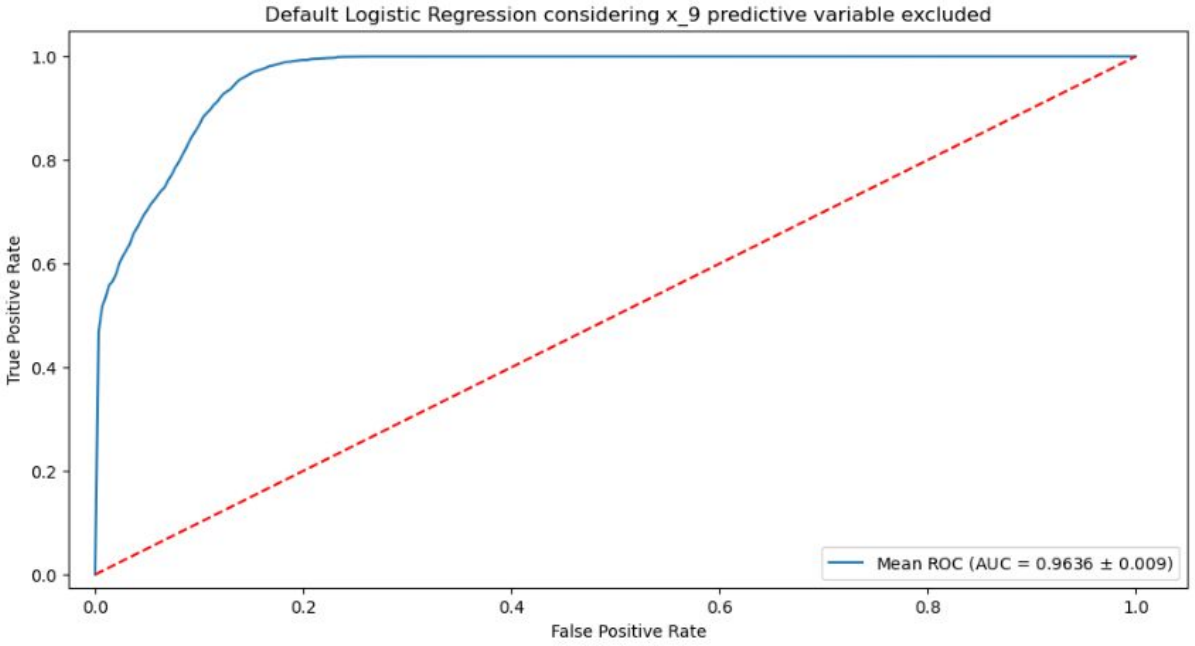

Figure 18.: Logistic Regression considering x_9 covariates excluded

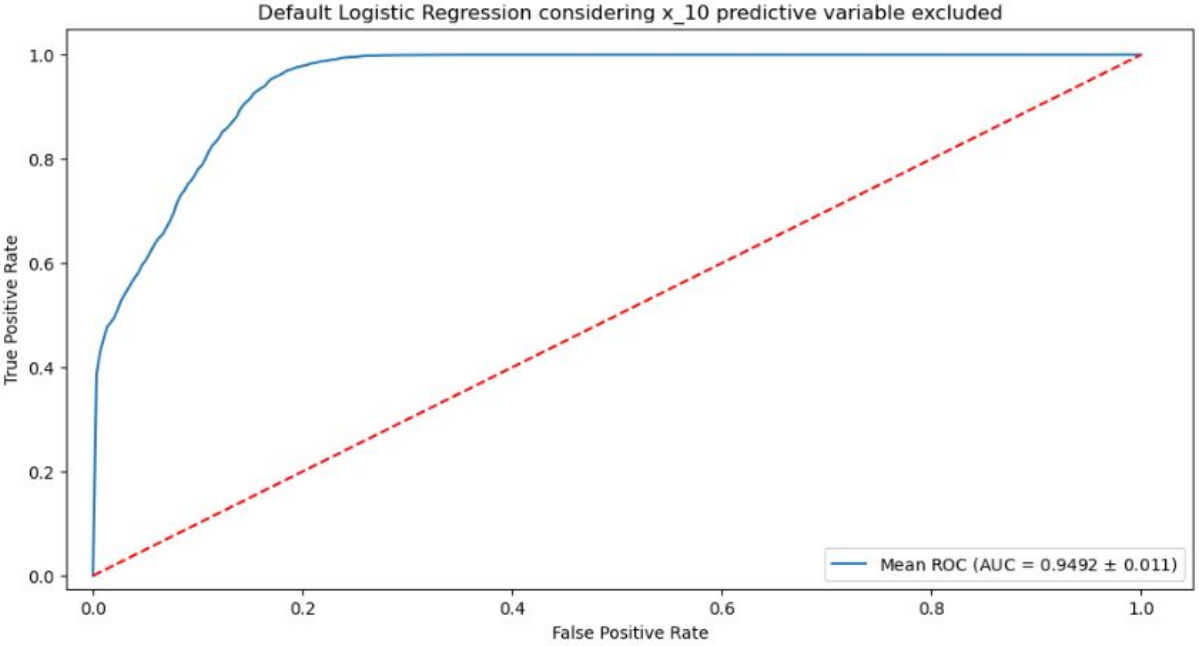

Figure 19.: Logistic Regression considering x_10 covariates excluded

From the previous figures we can conclude that by eliminating the first, second, and tenth columns we would achieve the worst results that this model could produce.

Figure 20 shows the minimum and maximum ROC curve of the logistic regression model built on all predictive variables.

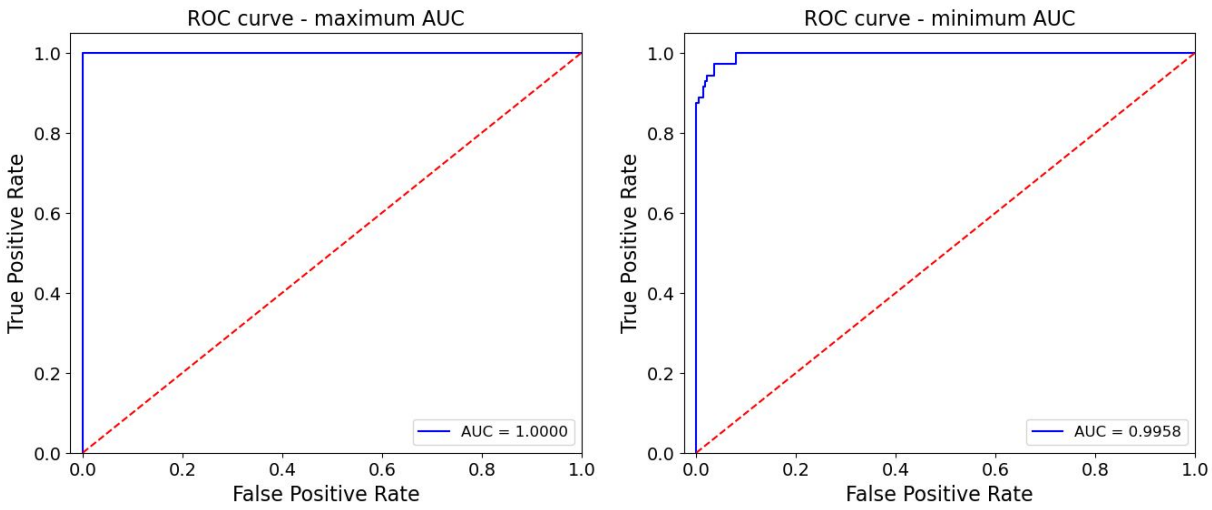

Figure 20.: ROC curves for default Logistic Regression without column excluding for dataset Scenario 1.

At this point I have only written about Logistic Regression, because it was the best performing model from all of the four models that I have tested. However, this kind of evaluation was performed for all other machine learning models listed in Section 4.3.

To see how each covariates perform separately, I created a diagram for all models, which shows the difference between the without column excluding and the column excluded results (Figure 21).

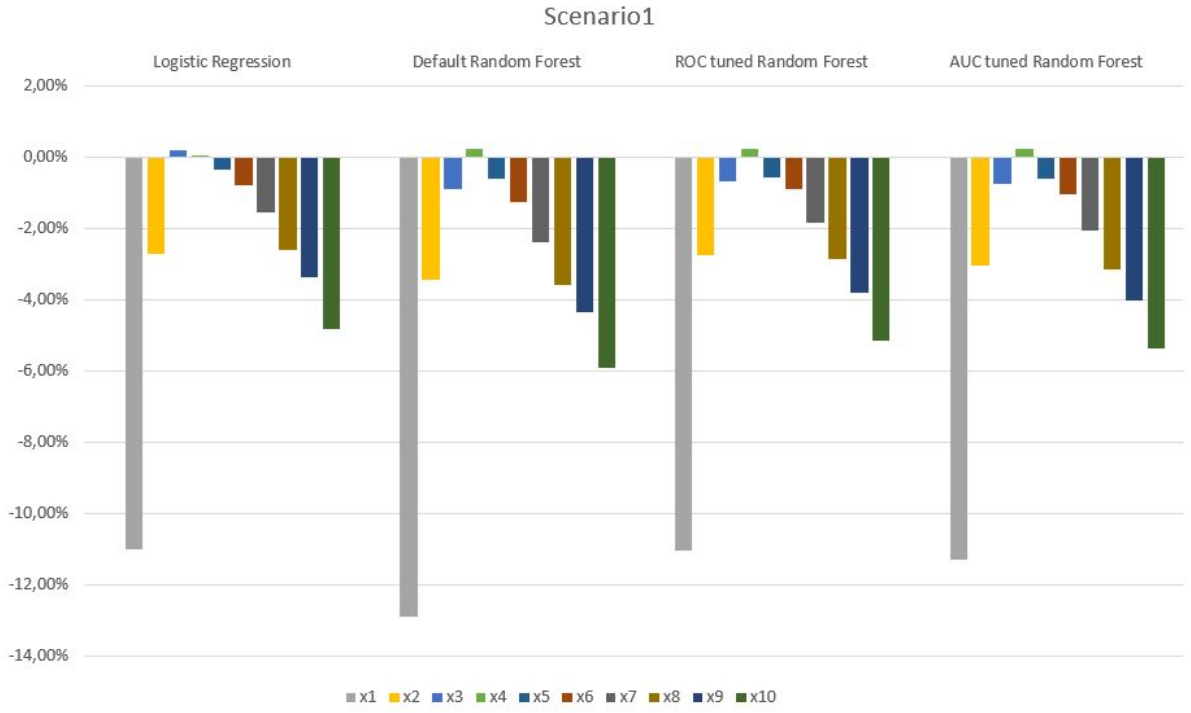

Figure 21.: Difference between accuracy of machine learning models run on the whole dataset and accuracy of the machine learning models operated on the truncated datasets (column excluded methods)

We can see that the Logistic Regression achieved two positive differences when I excluded the third and fourth covariates. However, all the Random Forest versions only achieved one positive difference by eliminating the fourth column. However, despite the fact that random forest produces a non-linear model, we can see that omitting individual variables meant a similar degree of accuracy loss as in the case of the logistic regression model.

On the other hand, before I would set the hypotheses, I wanted to see how the best and worst performing models' F1 score metric looked like, because I have said in the Evaluation metrics chapter, F1 score can be seen as a better accuracy metric.

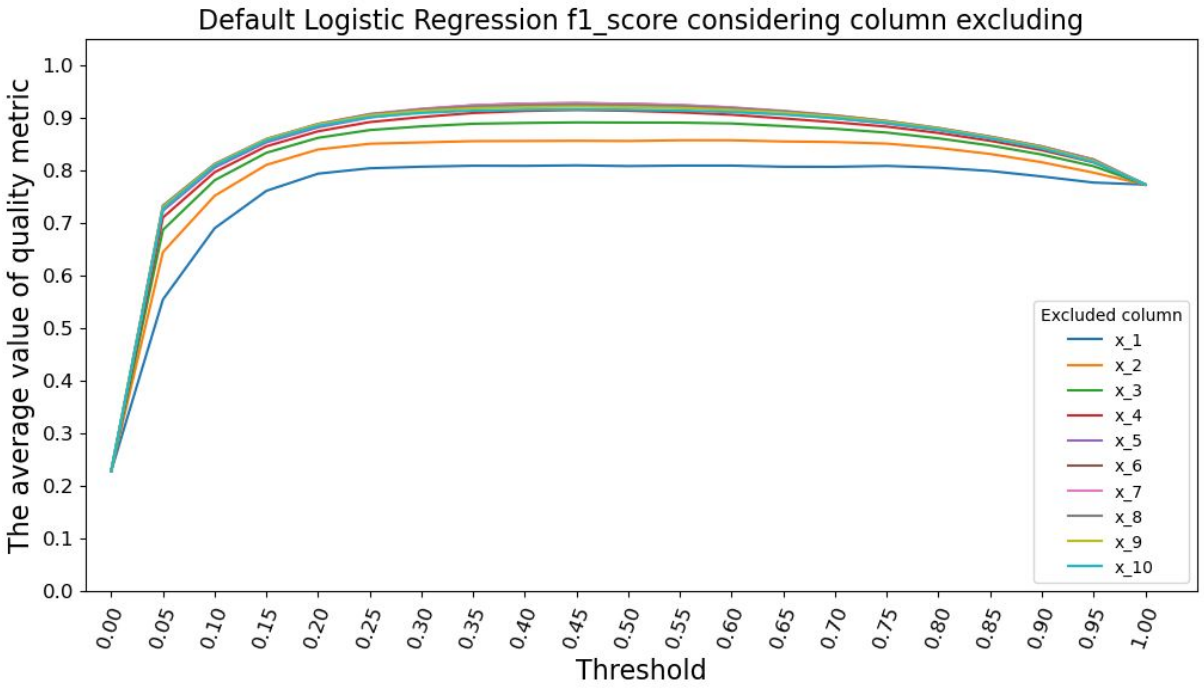

Figure 22.: Averages of F1 score of Logistic Regression considering column excluding

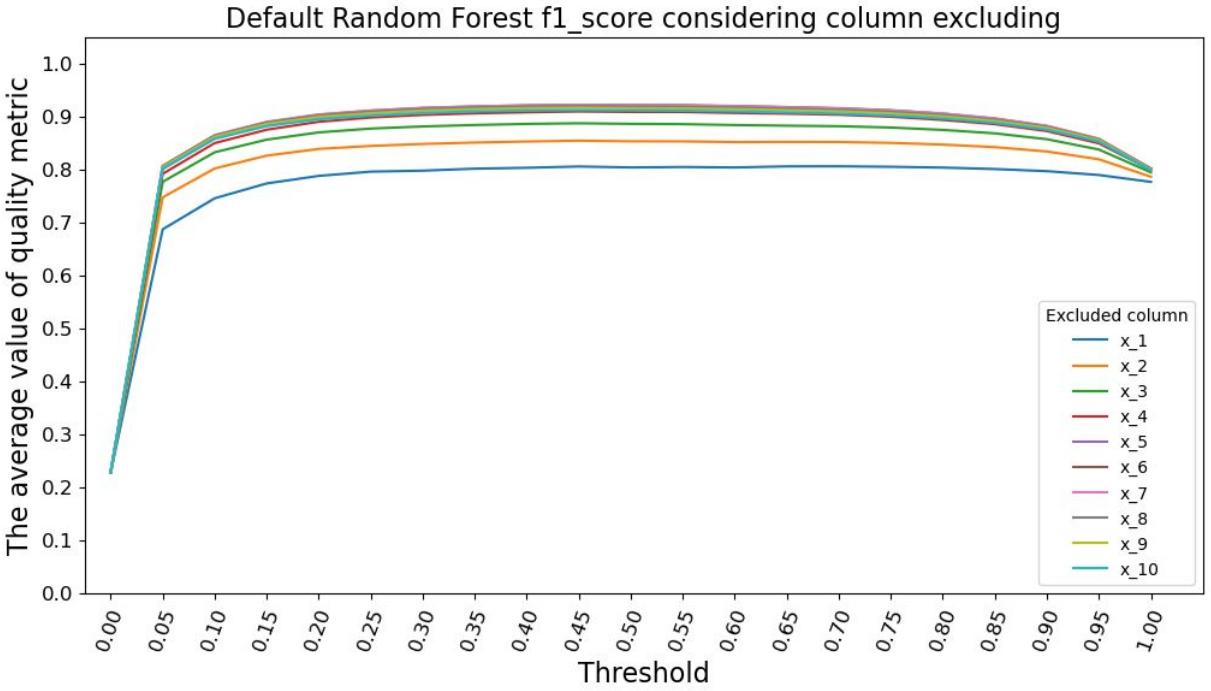

Figure 23.: Averages of F1 score of default Random Forest considering column excluding

At first glance, we can not see a huge difference between the two figures. However if we take a closer look we can see that while the default Random Forest flattens at 0.9 average F1 score, the Logistic Regression peaks above 0.9.

At the end of this scenario, I could conclude the following ones:

- Logistic regression could gain advantage over other machine learning models because I use its Logit function when I generated the binary outcome columns for each dataset.
- Generated data could be linearly separable. This is why datasets are more suitable for logistic regression.
- Despite the linear generation of the outcome variable, the default Random Forest method gave very similar results to the Logistic Regression model.
- Column weights might play a crucial part. I need to try different weights.

5.2. Scenario 2

The only change I made in this scenario are the column weights. Everything else is the same. In the mean ROC curves of Scenario 1 can be seen a pattern after column weight value 1. This why I started the generation of weights from $log(0.25)$ until $log(1)$ by increment 0.25. After $log(1)$ I continued the weight generation until $log(4)$ by increment 0.5. In Figure 11 we can see that the change of column weights have not made a huge impact in the results. This might be a sign that the usage of logit function for generating binary outcomes, plays a particularly big part in favor of the logistic regression model. However, before I would make a conclusion, I wanted to create another scenario.

Again, in this scenario, Logistic Regression achieved the best performance, which can be seen on Figure 24.

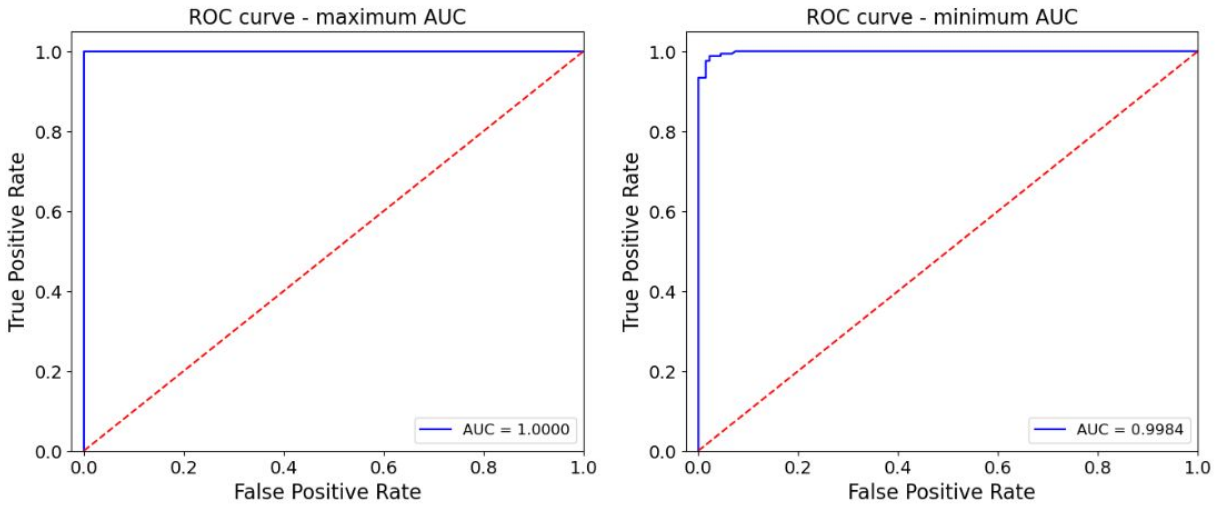

Figure 24.: Minimum and maximum Roc curves for Logistic Regression in Scenario 2

Due to the fact that I increased the column weights during dataset generation, each of the Random Forest versions got worse results from the without column excluding simulation than they were in the Scenario1. Moreover, as we can see in Figure 25., each model got better results at calculating the differences than in the previous scenario.

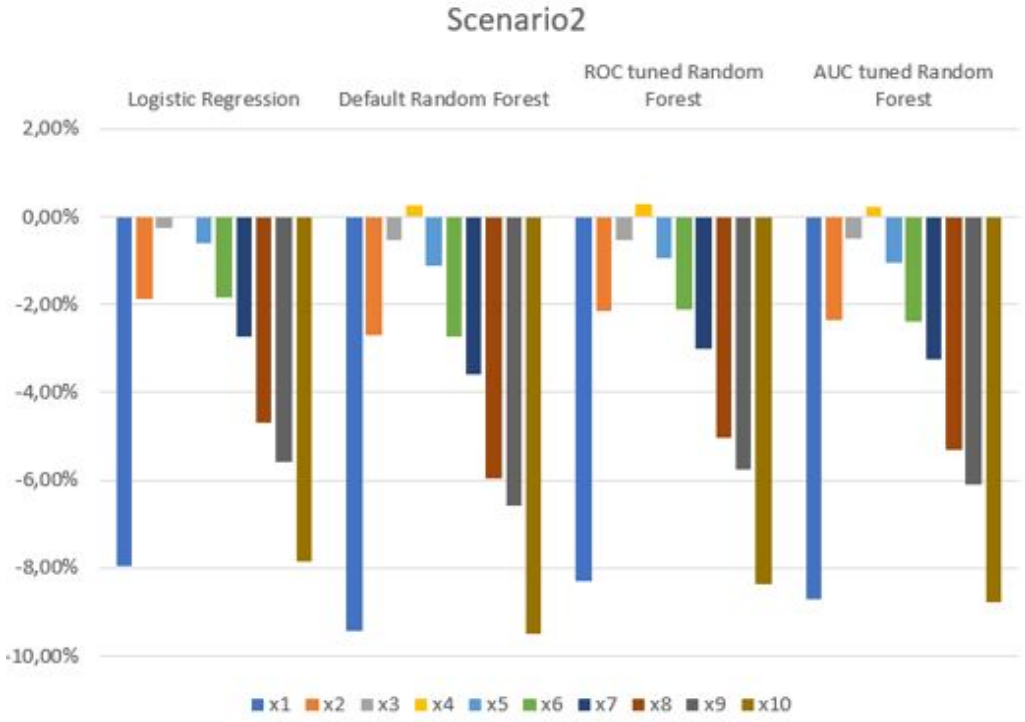

Figure 25.: Scenario 2's results of column excluding simulation

By seeing how much the calculated differences has been improved so far, I decided to create a new scenario with increased column weights. As during the research process, scenarios were processed parallel, the numbering of the scenarios is not continuous. Therefore, the new scenario was given the number 5.

5.3. Scenario 5

In this scenario I needed to create new sets of generated random datasets. I used the Bernoulli distribution, again, because I wanted to compare the upcoming results to the previous two scenarios' results. From this scenario I wanted to see what would happen if I increase the previous column weights further. This is why I started the generation of weights from $log(0.25)$ until $log(1)$ by increment 0.25. After $log(1)$ I continued the weight generation until $log(7)$ by increment 1.

Before I run all the simulations, I had high hopes that I would see huge drops in the models performances. However, the results were unexpected. Again, as we got used to it at this point, Logistic Regression was the best performing model (Figure 26).

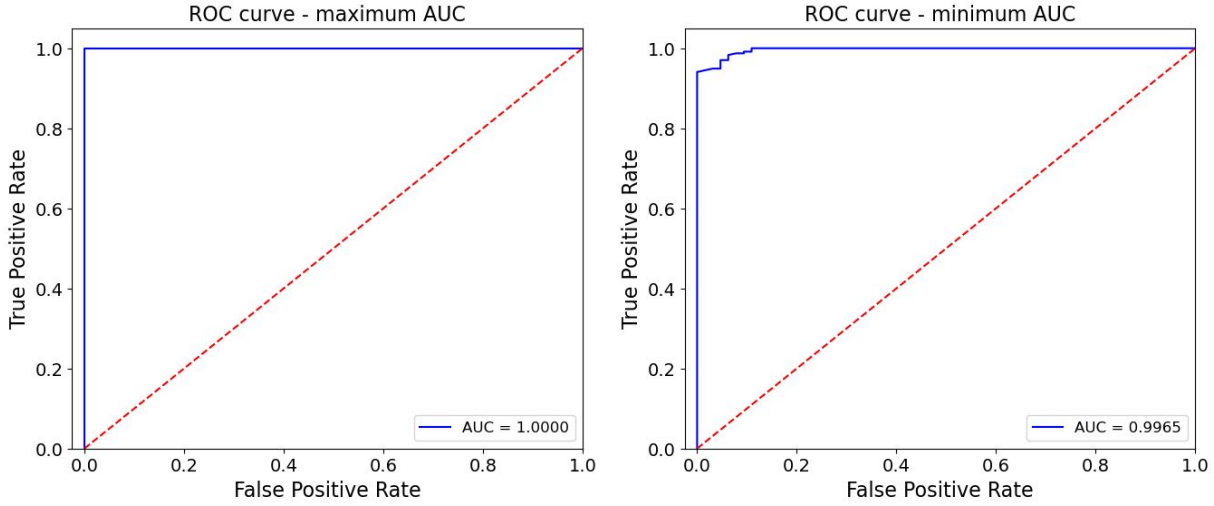

Figure 26.: Roc curves of default Logistic regression by using without column excluding in Scenario 5

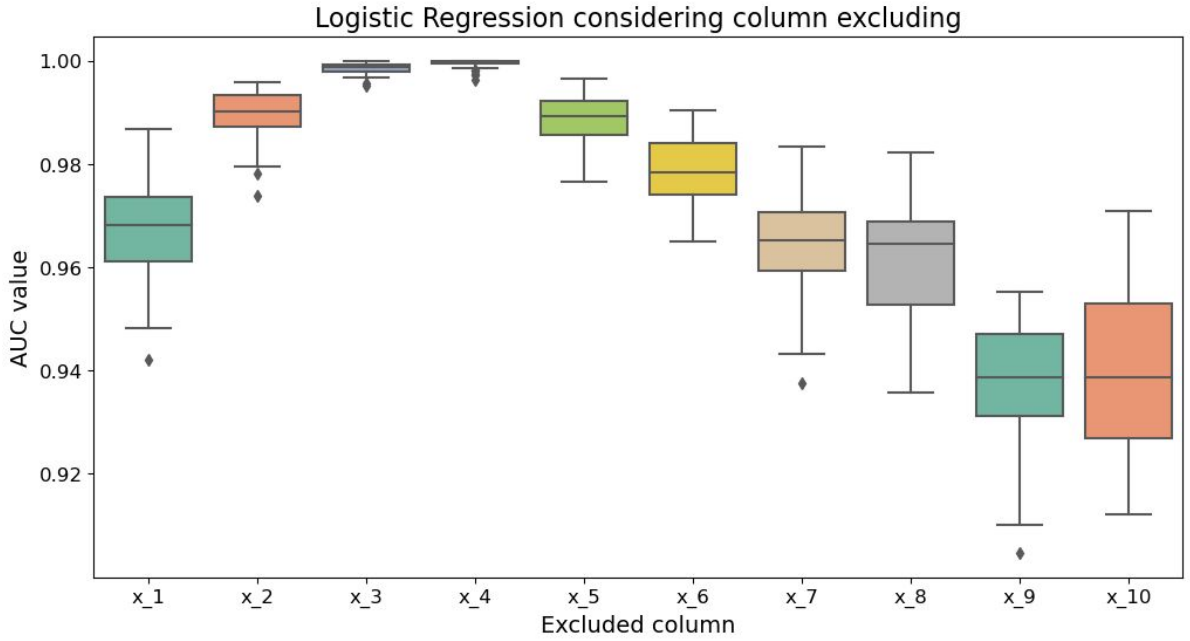

Figure 27.: Average AUC values of 50 simulations by using Logistic Regression

However to my surprise, every model further decreased the calculated differences (Figure 27).

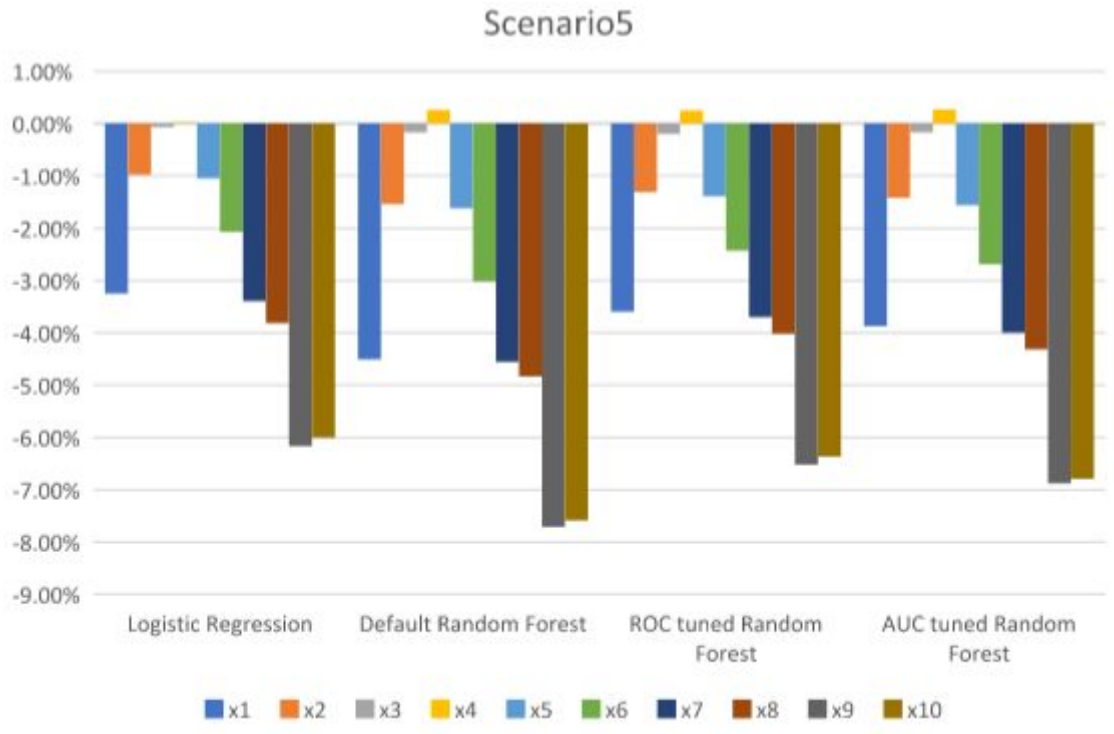

Figure 28.: Scenario 5's results of column excluding simulation

.

On the other hand, if we compare the best and the second best performing model's evaluation metrics that were created when I did not exclude any feature, we will see that the second best performing model outperforms the Logistic Regression in all cases except accuracy.

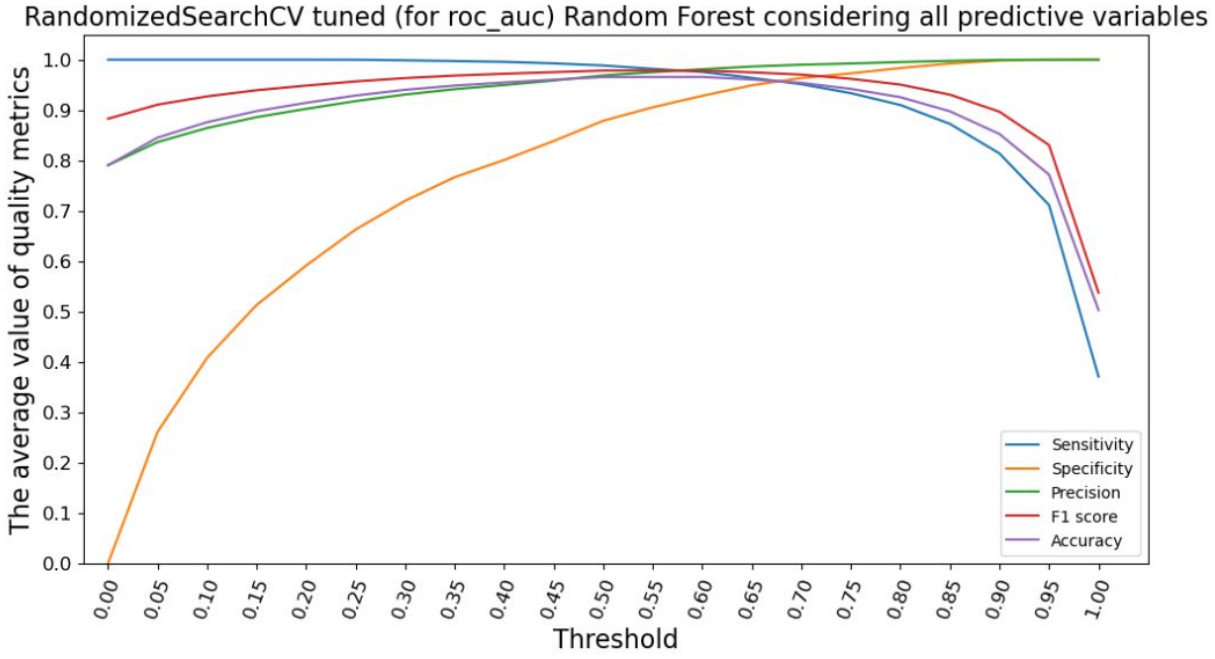

Figure 29.: Average evaluation metrics for ROC AUC tuned Random Forest using without column excluding simulation

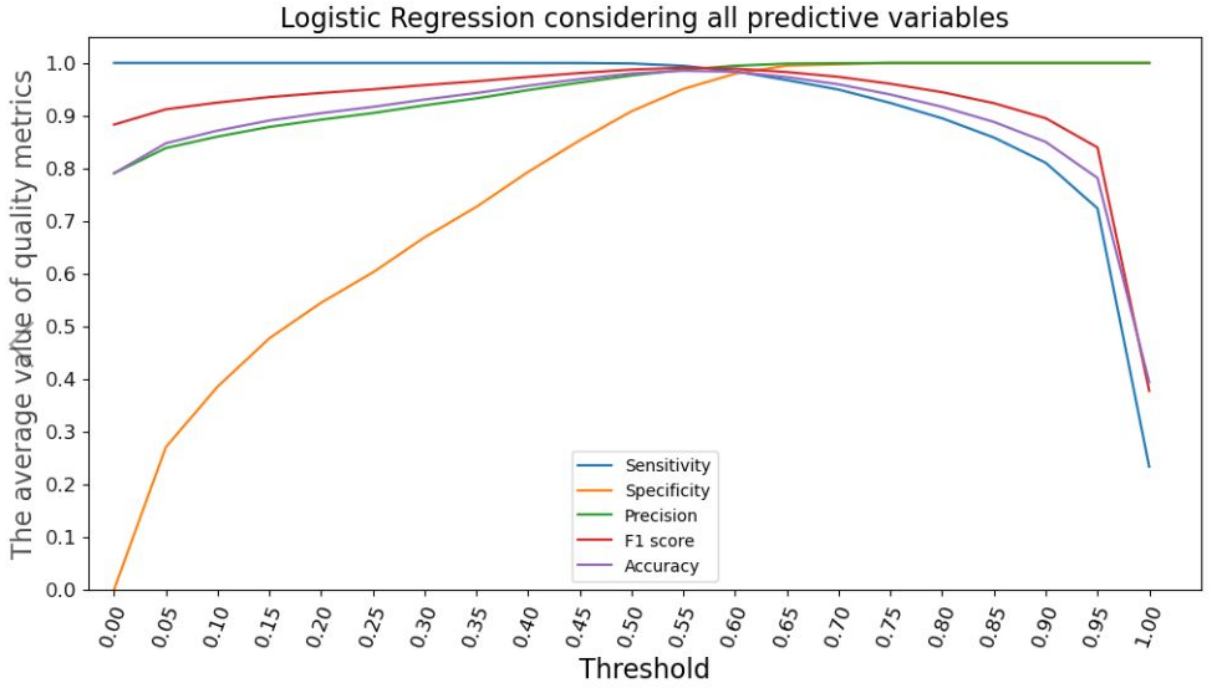

Figure 30.: Average evaluation metrics for default Logistic Regression using without column excluding simulation

In conclusion, in Scenarios 1, 2, and 5 we could see that the Logistic Regression won the competition every time. Probably, because the generated Bernoulli distributed random datasets are linearly separable. This is the territory, where the Logistic Regression can shine. After these three scenarios, I could tick most of the hypotheses that I had. However I still have three more, which are the following:

1. What would happen if I use Normal distribution instead of Bernoulli distribution? Could it be the turning point?
2. Does Logistic Regression come from only the linearly separability of the datasets? Or, does the usage of logit function when creating the binary output influence the performance of Logistic Regression?
3. What would happen if I use a benchmark dataset?

5.4. Scenario 3

In Scenario 1, 2, and 5 I used datasets, with Bernoulli distributed feature columns. In this scenario, the main idea was to create datasets which only contain Normal distributed

columns, and also a logit function created binary outcome column. I used the following parameters for the dataset generation:

- Column weights, remain the same as they were in Scenario 2
- Mean of normal distribution: 0.5
- Standard Deviation of normal distribution: 1

Unfortunately, if we take a look at the result of the column excluding simulation (Figure 31), we will see that it is almost the same as it was in Scenario 2. By using this information I can conclude that the Logistic Regression's major advantage comes from the usage of logit function for binary output column generation. Although these 4 scenarios solved almost all of my hypotheses, there remained one. Would the results be the same if I used a benchmark dataset?

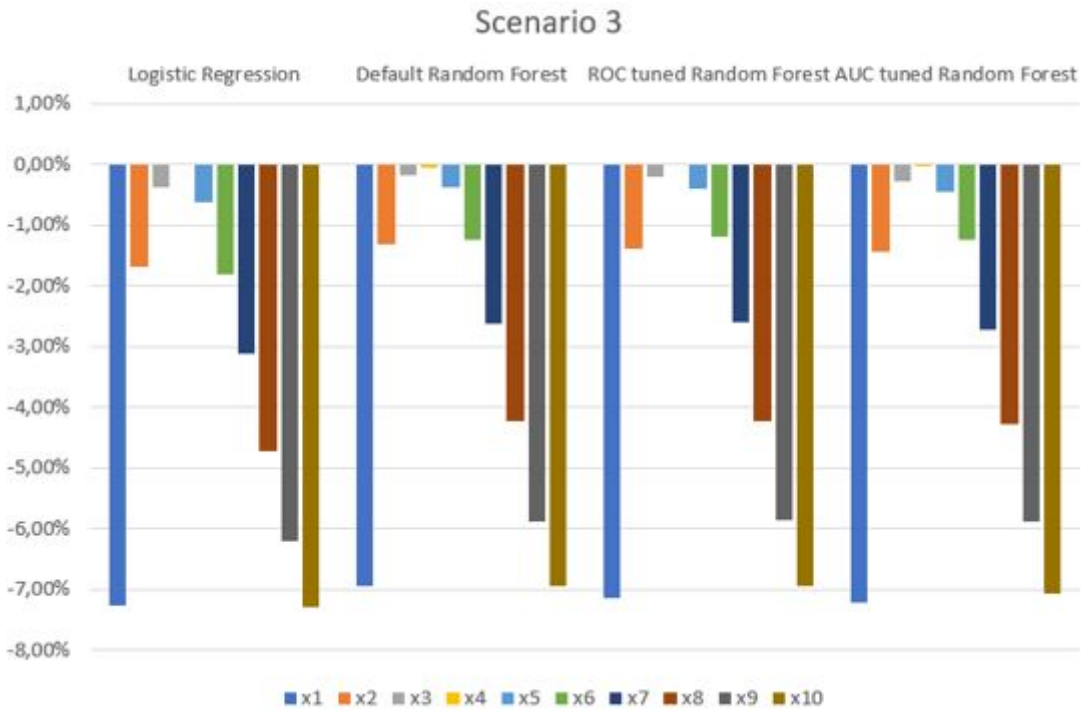

Figure 31.: Scenario 3's results of column excluding simulation

5.5. Scenario 4

In this scenario, I used a benchmark dataset, namely Census (Adult) Income. I wanted to compare the previous results which were based on random generated datasets to a dataset which outputs are not generated by logit function. Here I did not use column weights. The results of the simulations can be seen in Figure 32. From this figure I can conclude that the data is not linearly separable. Moreover we can see that Random Forest's versions outperformed the Logistic Regression model, by a lot. Not only in the column excluding, but in the without column excluding simulation, too. The results of without column excluding simulations can be seen in the following list:

- Logistic Regression with default parameters: 0.8557
- Random Forest with default parameters: 0.9077
- Random Forest tuned for roc_auc: 0.9181
- Random Forest tuned for accuracy: 0.9160

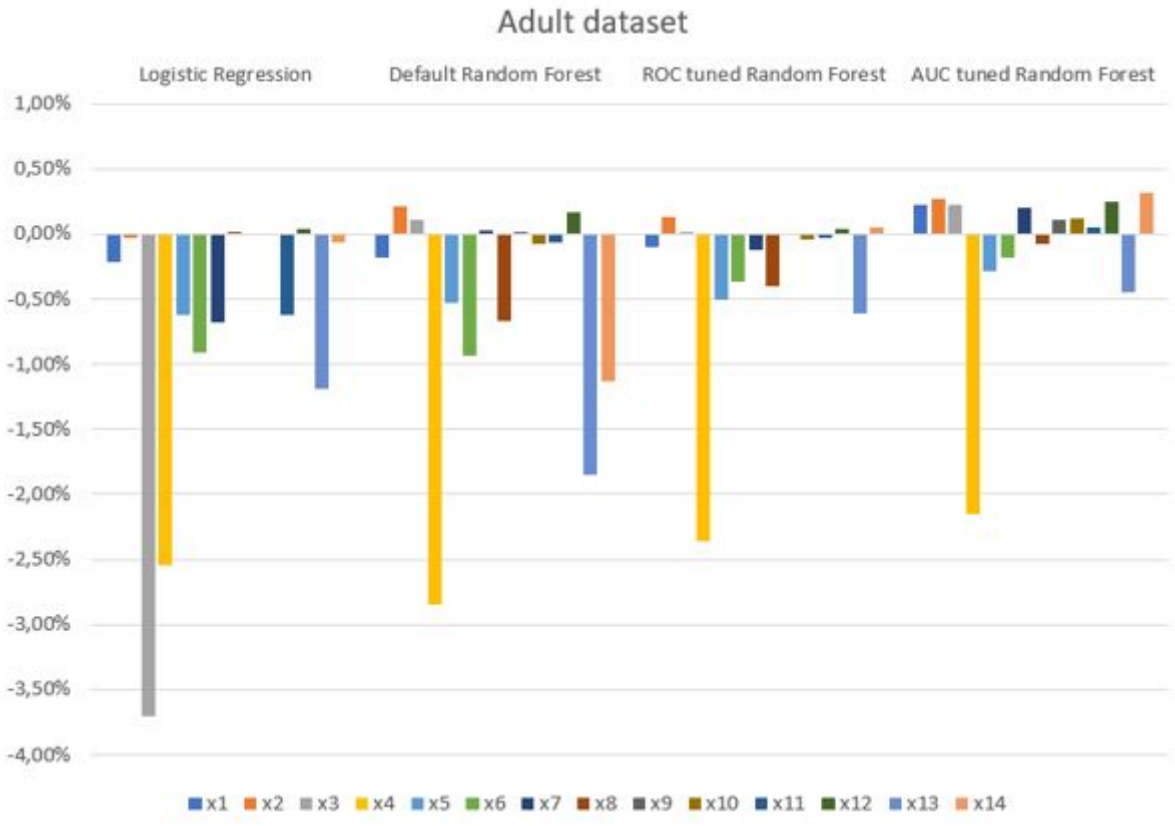

Figure 32.: Scenario 4's results of column excluding simulation

Results show that the Logistic Regression model and the Random Forest models were in a different degree sensitive to excluding the variables. This observation anticipates the need for further research.

Bibliography

[1] https://www.researchgate.net/publication/275224157_A_Review_on_Evaluation_Metrics_for_Data_Classification_Evaluations

[2] https://www.sciencedirect.com/science/article/pii/S1556086415306043

[3] https://archive.ics.uci.edu/ml/datasets/census+income

[4] https://www.researchgate.net/publication/267968705_Mersenne_Twister_-_A_Pseudo_Random_Number_Generator_and_its_Variants

[5] https://www.researchgate.net/publication/4321531_Enhanced_recursive_feature_elimination

[6] Fast correlation based filter (FCBF) page 3. https://www.researchgate.net/publication/221252792_Filter_Methods_for_Feature_Selection_-_A_Comparative_Study

[7] https://archive.ics.uci.edu/ml/datasets/adult

List of Figures