
# ITC6230
# Advanced Machine Learning

## Final Project

Implementation of a Neural Network with Adam optimizer from scratch

Athens, 31/3/2022

Contents

Table of Figures

## Abstract

The task is the implementation of the Adam optimizer for training feed-forward neural networks using the Back-Propagation (BP) method by automatically computing the gradient of the loss function. The implemented binary classification - regression model includes several activation functions, regularization methods for avoiding overfitting (L2 and Dropout), optimization algorithms (Gradient Descent and Adam optimizer), as well as grid search that serves hyperparameter tuning purposes. The full model pipeline is tested on several datasets.

## Problem Description

The task is the prediction of whether a person is considering purchasing an electric or a hybrid vehicle given a dataset containing socio-demographic attributes including age, gender, education, work status, economic attributes (income level) attributes and preference attributes. The problem could be treated as either a binary classification or a regression problem. Multi-classification problems are currently not supported: although the computation of the categorical cross entropy as a loss function is available, the backpropagation is pending, and so the implemented algorithm supports binary classification and regression problems.

## Data Description

The dataset provided by the instructor consists of a training set (OPTION2_train_data.csv), which in this case will be used as train and test set (the data is split based on a percentage defined by the user) and validation data (validation_data_no_labels.csv) for vehicle preferences. One could provide an additional dataset containing the labels for the validation data and obtain a data frame with the predicted and the true values along with their absolute difference (only in the case of regression problem) and a sign whether the prediction was True or False. In addition, for the binary problem one can obtain the confusion matrix of True Positive, False Positive, True Negative and False Negative. The other option is just the computation of predicted labels for a given validation set. The training dataset contains information about almost 15 thousand data rows and consists of 36 columns corresponding to the users' characteristics.

## Data pre-processing

To ensure that most of the information is extracted from the dataset and could be fed to the model, as some data may not sufficient or maybe not in the proper form, pre-processing is essential. It was decided that features containing more than 80% of missing values will be dropped, as if they do not, it is highly likely that they will affect the model's performance.

Checking whether a dataset is balanced or imbalanced is always beneficial: in the case of imbalanced classification, the predictive modeling task is primarily challenging, because of the severely skewed class distribution, and can lead to poor performance. By finding the percentage of each label, the user can test the previous statement and decide as to whether he/she will proceed to under-sampling; by selecting a different proportion of labels resulting in a more balanced dataset. The selection of under-sampling should depend on the size of the dataset.

It is essential to investigate the dataset regarding the correlation of its features, as well as the outliers, to achieve data reduction and as a result to obtain the results promptly. The user can activate the high correlated features flag along with the drop highly correlated features. By

doing so, the correlation matrix of all features and a score indicating the highly correlated features are produced. In case, the user decides that by dropping the highly correlated features, a better model's performance is achieved, he/she needs to set the threshold indicating whether a feature is considered highly correlated or not.
Please note that due to time restrictions, the pre-processing related to the under-sampling and correlated features is not built from scratch, but rather it requires the imblearn and sklearn libraries. Another option offered to the user is the replacement of outliers through both elimination and replacement with median values.

The algorithm was designed to offer the possibility of the Gaussian noise addition by allowing the user to activate the noise flag along with the parameters mu and sigma. This step is optional and is applied to the data before the normalization step (see Regularization techniques section).

The last step of pre-processing was the normalization of the data. Generally, it is considered a good practice to normalize the features to zero mean and unit variance, as this selection contributes into transforming all the features to similar scale and similar deviation.

Before proceeding to the model training, the data is split to training and test data after being shuffled (by adding the random state argument). The user can define the splitting percentage himself/herself.

## Neural Network Model

This section refers to the neural network architecture by making a brief analysis of how a typical neural network works. The key points of the algorithm's implementation are further explained in the following subsections.

Traditional neural networks are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. The neural network is fully connected meaning that each node (neuron) is connected to another and has an associated weight and bias.

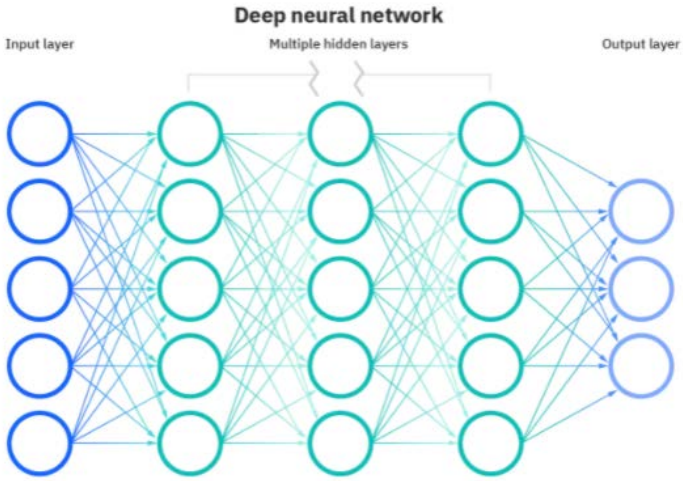


*Figure 1: The neural network architecture*

All inputs are multiplied by their respective weights (see Initialization of bias and weights section) and summed. The result is passed through an activation function (see Activation Functions section), which determines the output. The process describing the information

flowing from one layer to the next defines the neural network as a feedforward network. As the training is happening, to evaluate the accuracy of the model, the selection of a loss function is necessary. The goal is to minimize the loss function to ensure correctness of fit for any given observation. As the model adjusts its weights and bias (see Updating the weights section), it uses the loss function and reinforcement learning to reach the point of convergence. The process in which the algorithm adjusts its weights is through gradient descent/Adam optimizer, allowing the model to determine the direction to minimize the loss function. The current model is trained through backpropagation with batches. The learning algorithm will work through the entire training dataset many times (number of epochs) and at each iteration, a decrease in the loss function is expected [1]. The evaluation of the model's performance is measured by metrics, such as MAE, RMSE and $R^2$ for regression and by accuracy for binary classification problems (see Evaluation of the model section).

## Hyperparameters selection

The algorithm is designed to allow the user to effectively interfere to the neural network's architecture. In this section, the hyperparameters concerning the model, regularization techniques and Gradient Descent and Adam optimizer are discussed.

Both epochs and batch size are hyperparameters of the gradient descent. The first corresponds to the number of passes through the training data and the second to the number of training samples to work through before the model's internal parameters are updated [2].

The penalty term $\lambda_2 \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function J.

$$\tilde{J}(\theta;\, X, y) \,=\, J(\theta;\, X, y) \,+\, \lambda_2 \Omega(\theta)$$

Setting $\lambda_2$ to 0 results in no regularization. Larger values of $\lambda_2$ correspond to more regularization. Reasonable values of $\lambda_2$ range between 0 and 0.1 [3].

The dropout threshold, set by the user, is the probability of keeping a neuron active during drop-out. For example, in the case where the dropout threshold is set to 0.5, every hidden neuron is set to 0 with a probability of 0.5, meaning that there is a 50% change that the output of a given neuron will be forced to 0.

Learning rate is a hyperparameter of the updating weights phase. In the gradient descent algorithm, it refers as the parameter to which the gradient of the weight and the bias is being multiplied with (see Gradient Descent section).

The Adam optimizer hyperparameters that can be set by the user are $b_1$, which is the exponential decay rate for the first moment estimates, $b_2$, which is the exponential decay rate for the second moment estimates and $\varepsilon$, which is a very small number to prevent any division by zero in the implementation (see Annex). In the Adam optimizer, learning rate is the parameter to which a correction of first and second raw moment estimates is being multiplied with (see Adam optimizer section).

As the learning rate is considered one of the hyperparameters that is the most difficult to set having a significant impact on model performance, one should be cautious. Larger values result in faster initial learning before the rate is updated, whereas smaller values slow learning right down during training.

### Grid search

Grid search is a tuning technique that attempts to compute the optimum values of hyperparameters. This exhaustive search was decided to perform on the batch size and the number of epochs.

## Initialization of biases and weights

One of the most important steps in a neural network algorithm is the initialization of weight and bias values. It is typical that the biases for each unit are set to heuristically chosen constants (at most weight initialization schemes the biases are set to zero), and only the weights are initialized randomly. Though there are a few situations where some biases are set to non-zero values. In the case where ReLU is used as the activation function for hidden layers, the user has the option to activate the initialization ReLU flag, which will set the biases to 0.1 to avoid too much saturation at initialization.

The standard approach for initialization of the weights of the layers that use the Sigmoid or Hyperbolic Tangent activation function is called Xavier (also known as Glorot) initialization.

$$weight = U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \qquad \text{(Xavier weight initialization)}$$

$$weight = U\left[-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}\right] \qquad \text{(Xavier Uniform weight initialization)}$$

where n: the number of nodes in the previous layer and m: the number of nodes in the current layer.

A modified version of the Xavier approach was developed specifically for layers that use ReLU activation, the He weight initialization.

$$weight = U\left[-\sqrt{\frac{2}{n}}, \sqrt{\frac{2}{n}}\right] \qquad \text{(He weight initialization)}$$

The weights in the model can also be initialized to values drawn randomly from a Gaussian or uniform distribution. The choice of Gaussian or uniform distribution does not seem to matter very much but has not been exhaustively studied. However, the scale of the initial distribution seems to have a significant effect on the optimization procedure's outcome as well as on the ability of the network to generalize [4].

## Activation functions

Activation functions in neural network define how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.

### Sigmoid

The sigmoid function squashes a real-valued number to the range between 0 and 1. In practice, the sigmoid non-linearity is rarely ever used due to its two major drawbacks: the fact that it saturates and kills gradients and that its sigmoid outputs are not zero-centered. The sigmoid activation function is applied to the output layer in the case of binary classification problems.

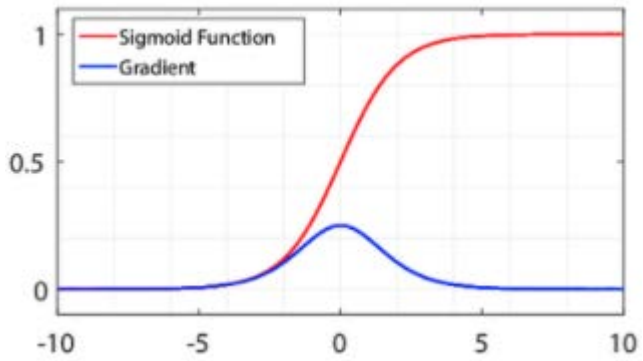


*Figure 2: Sigmoid activation (blue) and derivative (red) for efficient gradient computation*

## ReLU

The Rectified Linear Unit (ReLU) can greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions, and as a result is considered a rather popular activation function. ReLU has a significant drawback though, as its units can be fragile during training, and this can result in saturation.

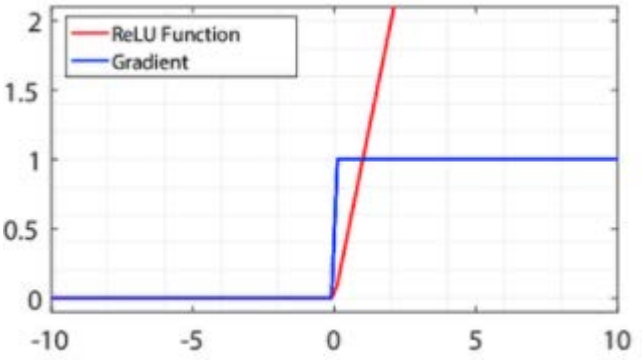


*Figure 3: ReLU activation (red) and derivative (blue) for efficient gradient computation*

## Leaky ReLU

Leaky ReLU is proposed as an attempt to correct ReLU's saturation. For $x < 0$, leaky ReLUs have a positive (but almost insignificant) slope (usually of 0.01). Some researchers suggest that the 'dying' problem of the ReLU function is solve with this alternative function, but alas the results are not always consistent.

## Hyperbolic tangent

The tanh non-linearity squashes a real-valued number to the range [-1, 1]. Its activations saturate, but unlike the sigmoid neuron its output is zero-centered, and thus it is preferred over the sigmoid [5].

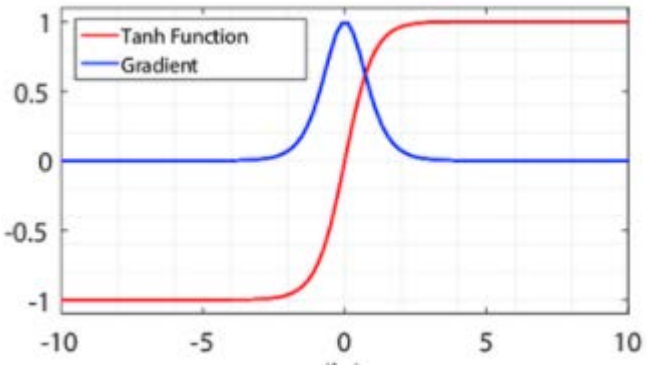


*Figure 4: Hyperbolic tangent activation (blue) and derivative (red) for efficient gradient computation*

### Softmax

The softmax function provides a "softened" version of the arg max. This function has multiple output values in which it saturates when the differences between input values become extreme. When the softmax saturates, many cost functions based on the softmax also saturate, unless they are able to invert the saturating activating function. The softmax activation function is applied to the output layer in the case of multiclass classification problems.

### Linear

The linear activation function (also known as identity and no activation) does not change the weighted sum of the input in any way, as it returns the value directly. The linear activation function is applied to the output layer in the case of regression problems.

## Loss functions

There are different types of loss functions based on the problem statement, which the user is trying to optimize. The loss functions, that are at the disposal of the user, for the regression problems are the Mean Squared Error (MSE), Mean Absolute Error (MAE), and Relative Absolute Error (RAE), and for the binary classification problems, the binary cross entropy.

## Regularization techniques

One of the main challenges in machine learning is how assure a satisfying model's performance on unseen data. The literature is full of strategies (known as regularization techniques) that are designed to specifically for reducing the test error, possibly at the expense of increased training error.

### Gaussian Noise

Gaussian Noise is one of the regularization techniques that reduce overfitting and can be used on both the input and the hidden layers. The implementation of the algorithm allows the noise to be applied only to the input layers.

### L2 Regularization

Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty Ω(θ) to the objective function J. L2 regularization (known also as ridge regression or Tikhonov regularization) strategy drives the weights closer to the origin by adding a regularization term $\Omega(\theta) = \frac{1}{2}\|w\|^2$ to the objective function and thereby preventing overfitting [6].

### Dropout

Dropout is another popular technique that is used to prevent a model from overfitting. The way dropout works is by randomly setting the outgoing edges of the neurons existing in the hidden layers to 0 after each update of the training phase. This results to a more sensitive loss function to the neighboring neurons [7].

## Updating the weights

During the learning phase, the network learns by adjusting the weights to be able to predict the correct class label of input samples. Weight and bias updates will be reflective of the magnitude of error propagated backward after a forward pass has been completed. In this section, two popular methods: Gradient Descent and Adam optimization algorithm are presented.

### Gradient Descent

Gradient descent is an optimization algorithm that maintains a single learning rate for all weight updates and the learning rate does not change during training.

$$w' = w - \alpha\, dw$$

$$b' = b - \alpha\, db$$

where $\alpha$ the learning rate and $dw$, $db$ the gradients of weights and bias respectively.

### Adam optimizer

Adam (Adaptive Movement Estimation) is an optimization algorithm; the natural successor of AdaGrad and RMSProp that can be used instead of the classical gradient descent procedure to update network weights iterative based in training data. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients, as follows:

A moment vector ($m$) and exponentially weighted infinity norm ($n$) are maintained for each parameter referred to as m and v respectively. They are initialized to 0 at the start of the search.

- $m = 0$
- $v = 0$

The algorithm is executed iteratively over time t starting at $t = 1$, and each iteration involves calculating a new set of parameter values x. The gradients (of the weights and biases) are calculated for the current time step.

$dW(t)$: $gradient(dW)$ and $db(t)$: $gradient(db)$

The first moment is updated using the gradient and a hyperparameter beta1.

- $m_W(t) = b_1 * m(t-1) + (1 - b_1) * dW(t)$
- $m_b(t) = b_1 * m(t-1) + (1 - b_1) * db(t)$

The second moment is updated using the squared gradient and a hyperparameter beta2.

- $v_W(t) = b_2 * v(t-1) + (1 - b_2) * dW(t)^2$
- $v_b(t) = b_2 * v(t-1) + (1 - b_2) * db(t)^2$

The first and second moments are biased because they are initialized with zero values.

The first and second moments are bias-corrected, starring with the first moment:

- $\hat{m}_W(t) = m_W(t) / (1 - {b_1}^t)$
- $\hat{m}_b(t) = m_b(t) / (1 - {b_1}^t)$

- $\hat{v}_W(t) = v_W(t) / (1 - {b_2}^t)$
- $\hat{v}_b(t) = v_b(t) / (1 - {b_2}^t)$

Note, ${b_1}^t$ and ${b_1}^t$ refer to the $b_1$and $b_2$ hyperparameters that are decayed on a schedule over the iterations of the algorithm.

The updated parameters are calculated, as follows:

- $w' = w - \frac{\alpha * \hat{m}_W(t)}{\sqrt{\hat{v}_W(t)} + \varepsilon}$

- $b' = b - \frac{\alpha * \hat{m}_b(t)}{\sqrt{\hat{v}_b(t)} + \varepsilon}$

where the $\alpha$ (learning rate) is the step size hyperparameter, $\varepsilon$ is a small value that ensures the non-deviation by zero error [8].

# Derivation of N-Layer Neural Network

In this section, the equations for the feed forward propagation and backpropagation are being presented. For the following calculations $y_{hat}$ (the predicted output) will be denoted as $a$. Also, some more notations need to be defined:

$\frac{\partial J_k}{\partial z}$ the error in loss function with respect to $z$

$\frac{\partial J_k}{\partial w}$ the error in loss function with respect to $w$

$\frac{\partial J_k}{\partial b}$ the error in loss function with respect to $b$

## Equations of the Feed Forward propagation

For any layer $l$, the following equations hold true:

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g(Z^{[l]})$$

where $A^{[l-1]}$ the inputs from the previous layer, $W^{[l]}$ the weights, $b^{[l]}$ the biases, $Z^{[l]}$ the product sum plus bias, $g()$ the activation function applied $Z^{[l]}$ and $A^{[l]}$ the predicted output for the layer $l$.

The same formulas are applied for the Dropout regularization with the only difference that in the input and hidden layers some neurons are randomly shut down on each layer on every iteration. This is achieved by initializing a matrix of the shape of the predicted output that is populated with random samples from a uniform distribution over [0, 1). The portion of zeros and ones is decided based on the dropout threshold (a parameter set by the user). Some neurons are shut down and the ones remaining (the active ones) are scaled.

## Equations of the Backpropagation

For the output unit $k$, where the linear function is the activation function for regression problems [9]:

$$\frac{\partial J_k}{\partial w_{j,k}} = \frac{\partial J_k}{\partial a_k}\frac{\partial a_k}{\partial w_{j,k}}$$

Depending on the loss function:

- $\frac{\partial J_k}{\partial a_k} = 2\frac{a_k - y_k}{m}$ (MSE)
- $\frac{\partial J_k}{\partial a_k} = \frac{a_k - y_k}{\sqrt{m}|a_k - y_k|}$ (RMSE)
- $\frac{\partial J_k}{\partial a_k} = \frac{a_k - y_k}{m|a_k - y_k|}$ (MAE)
- $\frac{\partial J_k}{\partial a_k} = \frac{a_k - y_k}{m|a_k - y_k||y_k - y_{mean}|}$ (RAE)

$$\frac{\partial a_k}{\partial w_{j,k}} = \frac{\partial g(in_k)}{\partial w_{j,k}} = g'(in_k)\frac{\partial in_k}{\partial w_{j,k}} = g'(in_k)\frac{\partial}{\partial w_{j,k}}\left(\sum_j w_{j,k}a_j + b_k\right) = g'(in_k)a_j$$

$$\frac{\partial J_k}{\partial z_k} = \frac{\partial J_k}{\partial a_k}\frac{\partial a_k}{\partial z_k}$$

$$\frac{\partial a_k}{\partial z_k} = \frac{\partial g(in_k)}{\partial z_k} = g'(in_k)\frac{\partial in_k}{\partial z_k} = g'(in_k)\frac{\partial}{\partial z_k}\left(\sum_j w_{j,k}a_j + b_k\right) = g'(in_k)$$

$$\frac{\partial J_k}{\partial b_k} = \frac{\partial J_k}{\partial z_k}\frac{\partial z_k}{\partial b_k}$$

$$\frac{\partial z_k}{\partial b_k} = \frac{\partial}{\partial b_k}\left(\sum_j w_{j,k}a_j + b_k\right) = 1$$

For the hidden unit $j$:

$$\frac{\partial J_k}{\partial w_{i,j}} = \frac{\partial J_k}{\partial a_k}\frac{\partial a_k}{\partial w_{i,j}}$$

$$\frac{\partial a_k}{\partial w_{i,j}} = \frac{\partial g(in_k)}{\partial w_{i,j}} = g'(in_k)\frac{\partial in_k}{\partial w_{i,j}} = g'(in_k)\frac{\partial}{\partial w_{i,j}}\left(\sum_j w_{j,k}a_j\right) = g'(in_k)w_{j,k}\frac{\partial a_j}{\partial w_{i,j}}$$

$$= g'(in_k)w_{j,k}g'(in_j)\frac{\partial in_j}{\partial w_{i,j}} = g'(in_k)w_{j,k}g'(in_j)\frac{\partial}{\partial w_{i,j}}\left(\sum_i w_{i,j}a_i\right)$$

$$= g'(in_k) w_{j,k} g'(in_j) a_i$$

$$\frac{\partial J_k}{\partial z_j} = \frac{\partial J_k}{\partial a_k} \frac{\partial a_k}{\partial z_j}$$

$$\frac{\partial a_k}{\partial z_j} = \frac{\partial g(in_k)}{\partial z_j} = g'(in_k) \frac{\partial in_k}{\partial z_j} = g'(in_k) \frac{\partial}{\partial z_j} \left( \sum_j w_{j,k} a_j + b_j \right) = g'(in_k)$$

$$\frac{\partial J_k}{\partial b_j} = \frac{\partial J_k}{\partial z_j} \frac{\partial z_j}{\partial b_j}$$

$$\frac{\partial z_j}{\partial b_j} = \frac{\partial}{\partial b_j} \left( \sum_i w_{i,j} a_i + b_j \right) = 1$$

# Evaluation of the model

In this section, the model's performance is assessed by different metrics and the results of the loss function per epoch, as well as the evaluation metric's value per epoch for both the training and the test set are presented. The subsection of Comparisons refers to the results obtained by changing various parameters, such as the regularizations (L2 and dropout), different hyperparameters (epochs, batch size and learning rate).

## Evaluation Metrics

Depending on the problem, different evaluation metrics are used to measure the performance of the machine learning model. For regression problems, the user has the option to use either the Relative Mean Squared Error (RMSE), Mean Absolute Error (MAE) or $R^2$, whereas for binary classification problems, accuracy is the only option.

## Comparisons

In the case where the problem is treated as binary, the drop in the loss values is clear.

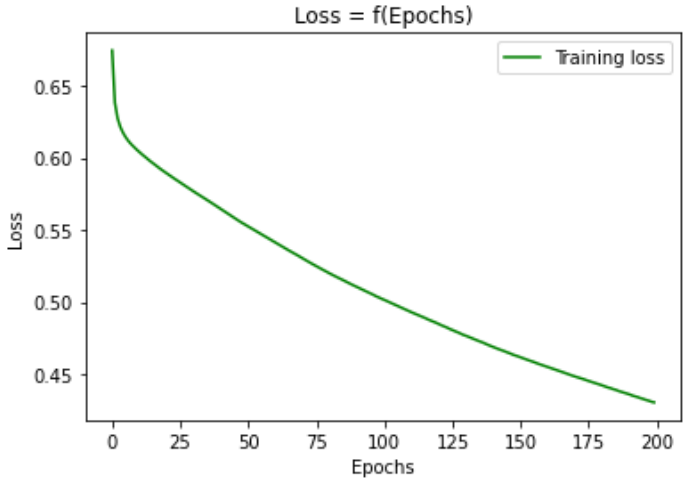


*Figure 5: (Binary classification problem) Loss vs. epochs for the training data: Neural Network Architecture parameters = {'layers': {1: 11, 2: 90, 3: 90, 4: 1}, 'activation': {1: 'relu', 2: 'relu', 3: 'relu', 4: 'sigmoid'}}*

Regarding the accuracy graph, a sudden increase in the accuracy of both training and test set is visible in the first few epochs. After the increase, for the case of the training data, the curve

seems to be smoother with the highest accuracy values reaching above 0.8. For the case of test data, accuracy tends to drop as the training of the model continuous. Since the gap of the two curves follows an increasing trend, we could argue that overfitting is taking place.

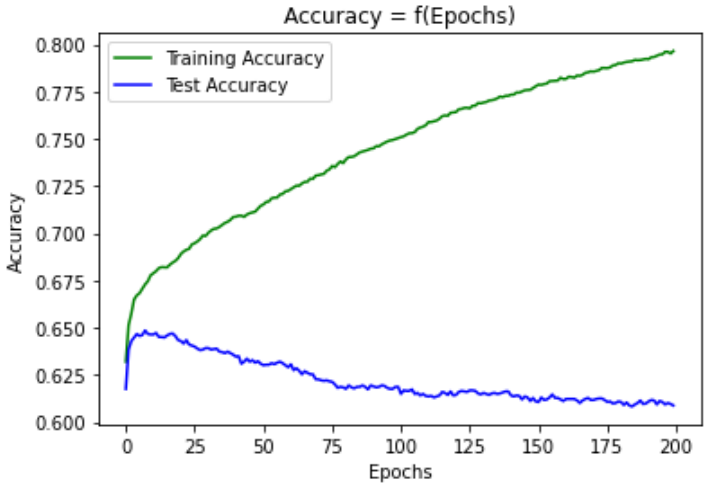


*Figure 6: (Binary classification problem) Accuracy vs. epochs for the training and test data: Neural Network Architecture parameters = {'layers': {1: 11, 2: 90, 3: 90, 4: 1}, 'activation': {1: 'relu', 2: 'relu', 3: 'relu', 4: 'sigmoid'}}*

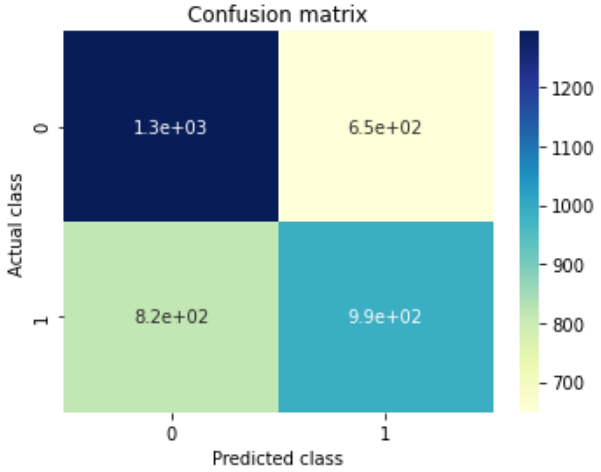


*Figure 7: (Binary classification problem) Confusion Matrix for the test data: Neural Network Architecture parameters = {'layers': {1: 11, 2: 90, 3: 90, 4: 1}, 'activation': {1: 'relu', 2: 'relu', 3: 'relu', 4: 'sigmoid'}}*

In the case where the problem is treated as regression, without any regularization, the loss curve has a similar behavior. Similarly to the previous evaluation metrics graph (accuracy vs. epochs for the binary problem), the model shows overfitting. The only difference between the graphs is the curve's trend; in the case of the RMSE, it is decreasing, which is the desideratum as lower values indicate better fit. As overfitting is present, we will proceed with the introduction of L2 regularization and dropout, hoping that the results will show a better performance on unseen data.

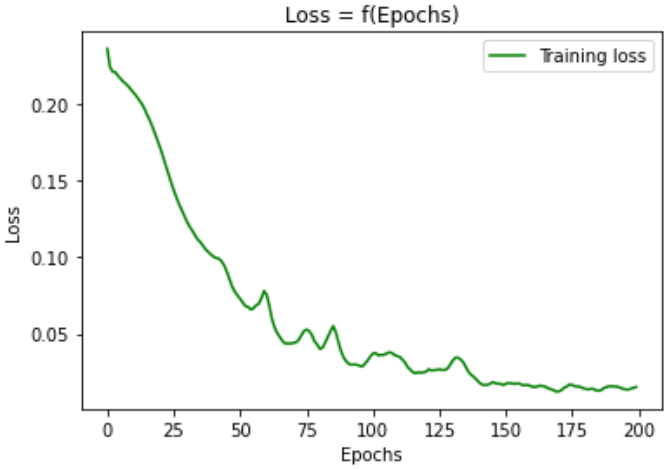


*Figure 8: (Regression problem without any regularization) Loss vs. epochs for the training data: Neural Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

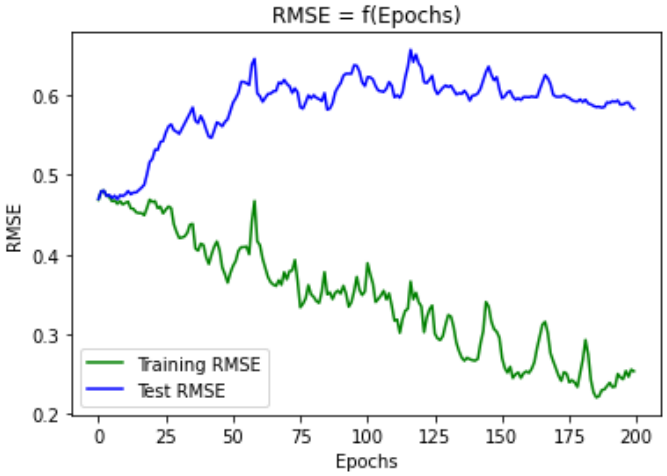


*Figure 9: (Regression problem without any regularization) RMSE vs. epochs for the training and test data: Neural Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

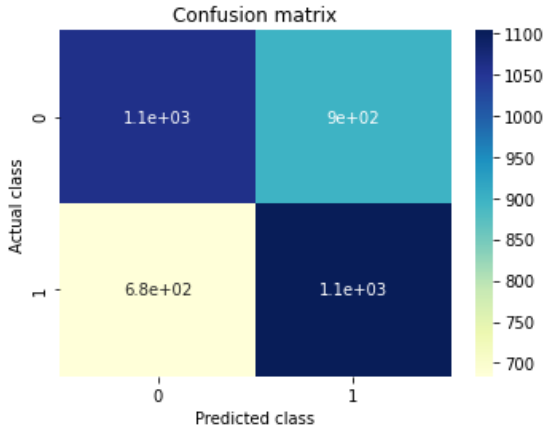


*Figure 10: (Regression problem) Confusion Matrix for the test data: Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

The addition of L2 regularization (in the binary classification) and dropout (in both the binary classification and the regression problem) seems to reduce the overfitting effect, as both curves referring to the training and the test data have a similar behavior; in fact, they decrease and at some point, they reach a plateau. Note that due to the algorithm's restriction the two optimization techniques cannot be applied simultaneously.

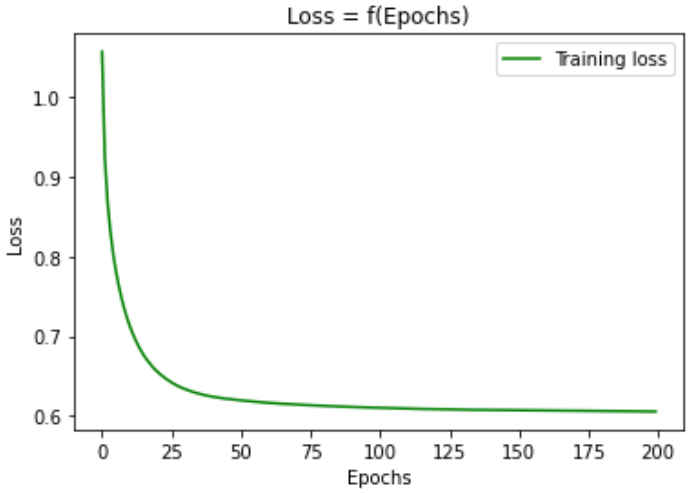


*Figure 11: (Binary classification problem with L2 regularization $\lambda_2 = 0.1$) Loss vs. epochs for the training data: Neural Network Architecture parameters = {'layers': {1: 11, 2: 90, 3: 90, 4: 1}, 'activation': {1: 'relu', 2: 'relu', 3: 'relu', 4: 'sigmoid'}}*

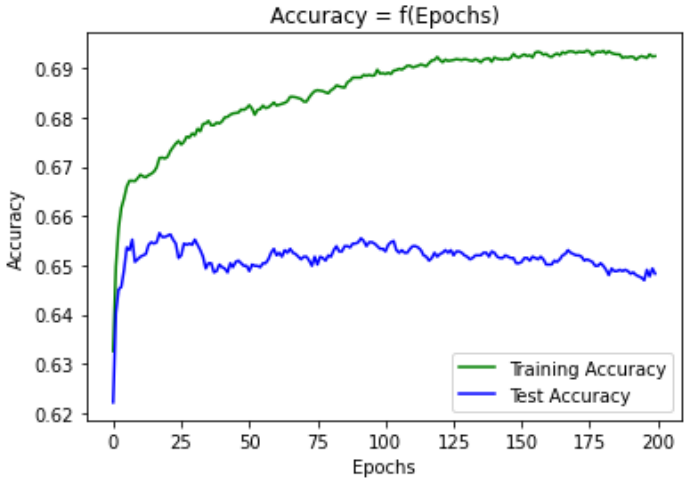


*Figure 12: (Binary classification problem with L2 regularization $\lambda_2 = 0.1$) Accuracy vs. epochs for the training and test data: Neural Network Architecture parameters = {'layers': {1: 11, 2: 90, 3: 90, 4: 1}, 'activation': {1: 'relu', 2: 'relu', 3: 'relu', 4: 'sigmoid'}}*

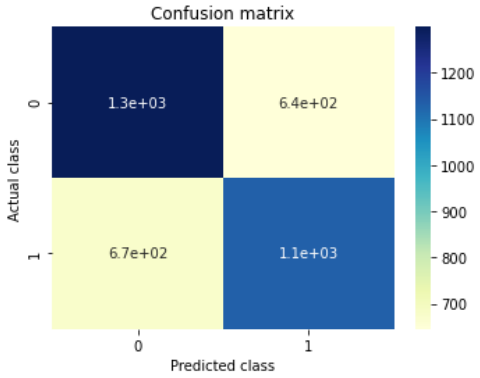


*Figure 13: (Binary classification problem with L2 regularization $\lambda_2 = 0.1$) Confusion Matrix for the test data: Neural Network Architecture parameters = {'layers': {1: 11, 2: 90, 3: 90, 4: 1}, 'activation': {1: 'relu', 2: 'relu', 3: 'relu', 4: 'sigmoid'}}*

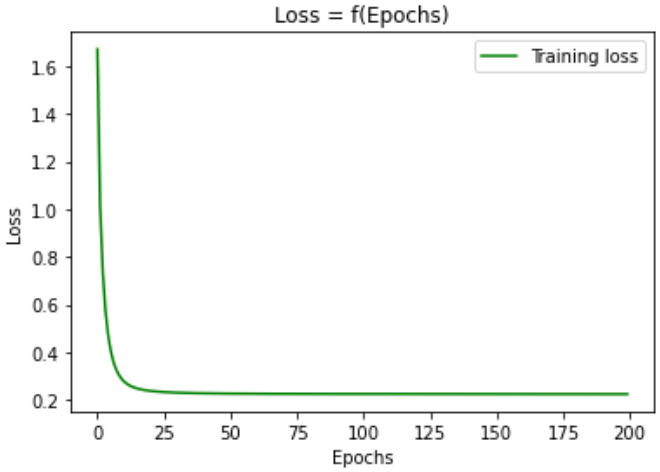


*Figure 14: (Regression problem with L2 regularization $\lambda_2 = 0.1$) Loss vs. epochs for the training data: Neural Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

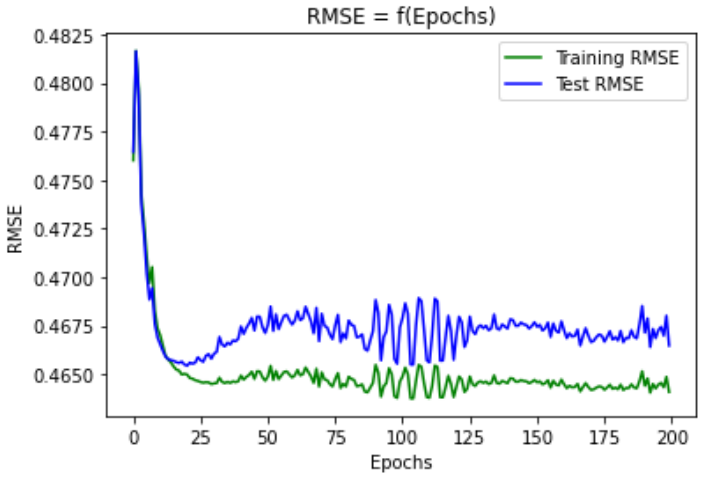


*Figure 15: (Regression problem with L2 regularization of $\lambda_2 = 0.1$) RMSE vs. epochs for the training and test data: Neural Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

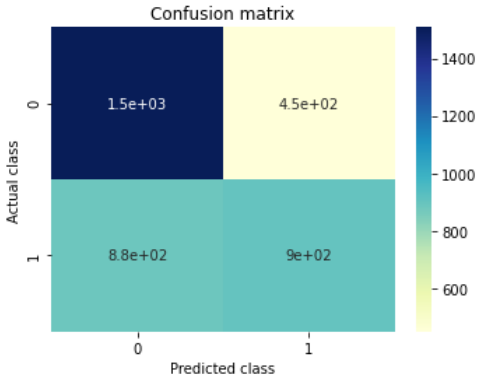


*Figure 16: (Regression problem with L2 regularization of $\lambda_2 = 0.1$) Confusion Matrix for the test data: Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

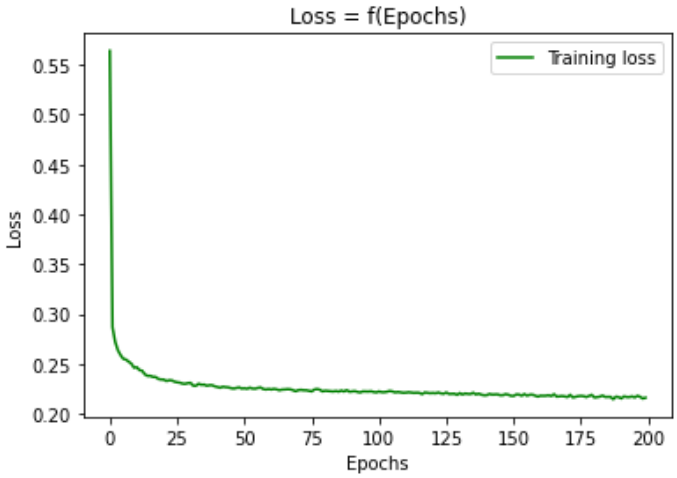


*Figure 17: (Regression problem with dropout of 0.5) Loss vs. epochs for the training data: Neural Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

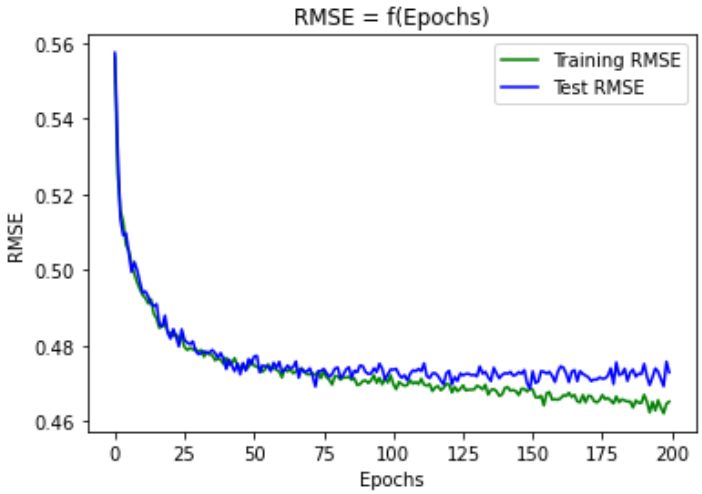


*Figure 18: (Regression problem with dropout of 0.5) RMSE vs. epochs for the training and test data: Neural Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

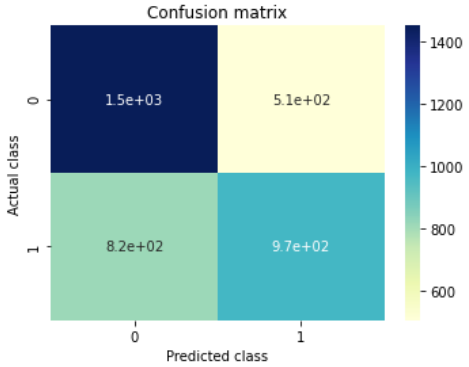


*Figure 19: (Regression problem with dropout of 0.5) Confusion Matrix for the test data: Network Architecture parameters = {'layers': {1: 425, 2: 268, 3: 100, 4: 58, 5: 1}, 'activation': {1: 'linear', 2: 'relu', 3: 'relu', 4: 'relu', 5: 'linear'}}*

A synopsis of different neural network models for various parameters along with their performance is available in the "Evaluation results synopsis" Section of Annex.

## Future Implementation

As a suggestion for future work, a further preprocessing on data could be done to each dataset individually, as well as the creation of additional features deriving from the existing ones could be included. In terms of the neural network algorithm, the supported types of problems could be expanded with the addition of multiclassification. L2 regularization and dropout techniques could also be supported for the classification problems. More basic algorithms, such as Momentum and algorithms with adaptive learning rates, such as AdaGrad and RMSProp along with more regularization techniques, such as L1 and early stopping could be included. The grid search could also be expanded with the addition of further hyperparameters, such as the learning rate, $\lambda_2$ parameter of L2 regularization, $b_1$, $b_2$ and $\varepsilon$ parameters of the Adam optimizer. These suggested additions could provide interesting comparisons between different regularization techniques and optimizers and a fruitful discussion regarding the selection of the most proper algorithm for each dataset. Furthermore, these additions could result to a generic and robust to overfitting model with more capabilities.

# References

[1] Education, I. C. (2021, August 3). Neural Networks. IBM. https://www.ibm.com/cloud/learn/neural-networks

[2] Brownlee, J. (2019, October 25). Difference Between a Batch and an Epoch in a Neural Network. Machine Learning Mastery. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[3] Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling (1st ed. 2013, Corr. 2nd printing 2018 ed.). Springer.

[4] Brownlee, J. (2021, February 7). Weight Initialization for Deep Learning Neural Networks. Machine Learning Mastery. https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/

[5] CS231n Convolutional Neural Networks for Visual Recognition. (2021). CS231n: Convolutional Neural Networks for Visual Recognition. https://cs231n.github.io/neural-networks-1/

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning (Adaptive Computation and Machine Learning series) (Illustrated ed.). The MIT Press.

[7] Nitish Srivastava, N. S., Geoffrey Hinton, G. H., Alex Krizhevsky, A. K., Ilya Sutskever, I. S., & Ruslan Salakhutdinov, R. S. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929–1958.

[8] Kingma, D. P. & Ba, J. (2014). Adam: A Method for Stochastic Optimization (cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015)

[9] Bishop, C. M. (2022). Pattern Recognition and Machine Learning (text only) 2nd(Second) edition BY C.M. Bishop (2nd printing edition). Springer.

[10] Brownlee, J. (2021a, January 12). Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. Machine Learning Mastery. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

# Annex

## Data Graphical Representations

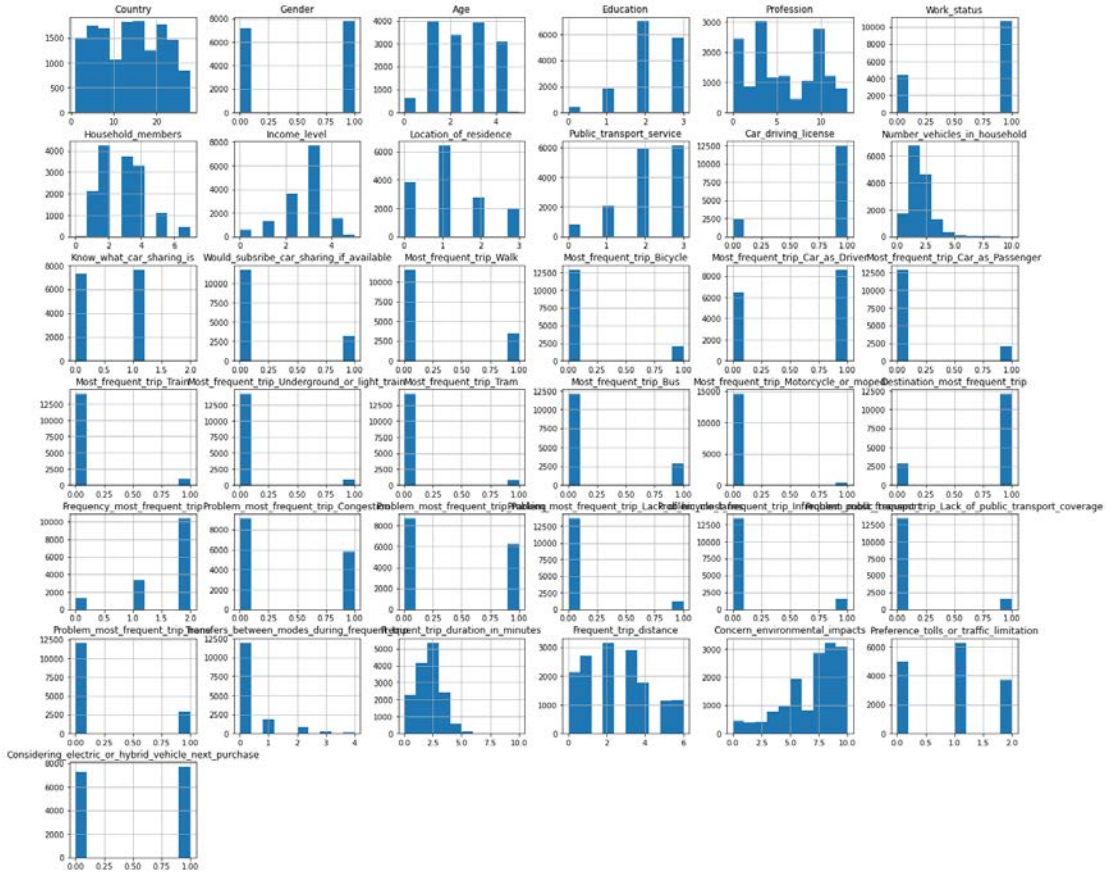


*Figure 20: Histogram of attributes*

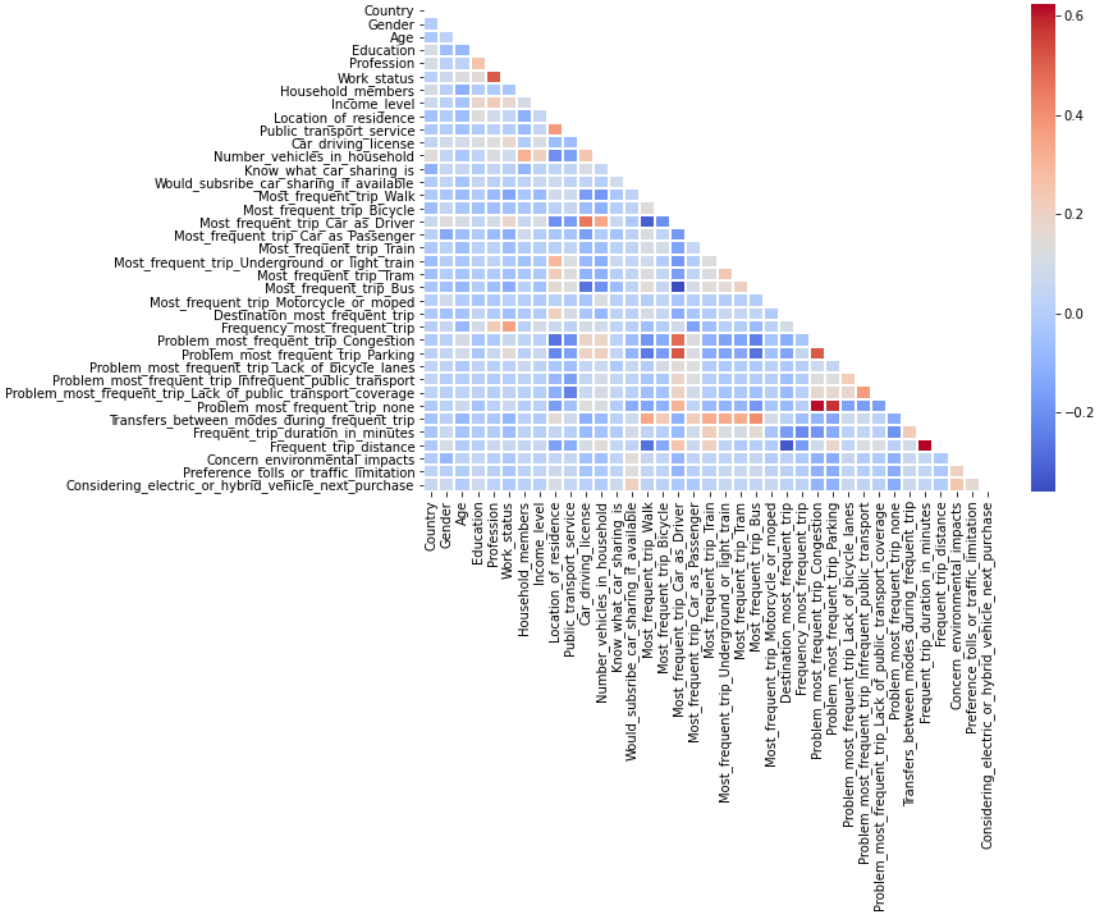


*Figure 21: Correlation matrix of the features*

## Adam Optimizer hyperparameters

The Adam paper suggests that good default settings for the tested machine learning problems are considered to be $learning\ rate = 0.001, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$.

The popular deep learning libraries generally use the default parameters:

- TensorFlow: $learning\ rate = 0.0001, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$
- Keras: $learning\ rate = 0.001, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$
- Blocks: $learning\ rate = 0.002, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$
- Lasagne: $learning\ rate = 0.001, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$
- Caffe: $learning\ rate = 0.001, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$
- MxNet: $learning\ rate = 0.001, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$.
- Torch: $learning\ rate = 0.001, b_1 = 0.9, b_2 = 0.999, \varepsilon = 10^{-8}$.

## Evaluation results synopsis

| Model | Parameters | Evaluation Metrics | | |
|---|---|---|---|---|
| | | Accuracy | RMSE | MAE |
| Binary Classification | n epochs 200<br>batch size 40<br>no layers 4 | 0.600 | | |
| Binary Classification with L2 regularization | n epochs 200<br>batch size 40<br>no layers 4<br>$\lambda_2 = 0.1$ | 0.648 | | |
| Regression | n epochs 200<br>batch size 30<br>no layers 5 | | 0.594 | 0.478 |
| Regression with L2 regularization | n epochs 200<br>batch size 30<br>no layers 5<br>$\lambda_2 = 0.1$ | | 0.466 | 0.430 |
| Regression with dropout | n epochs 200<br>batch size 30<br>no layers 5<br>$dropout\ thres = 0.5$ | | 0.473 | 0.436 |