ICS5110 – Assignment

Albert Muscat, Benoît Pannetier, Luca Zammit

December 2019

Contents

1 Introduction

In this Assignment two Machine Learning Models are to be implemented from First principles. The performance of the implemented models are to be compared with models implemented by third party libraries (in our case sklearn). Moreover the difference in performance between the 2 models are also to be compared.

The chosen dataset for this Assignment is the Absenteeism at work dataset. The dataset consists of 740 observations and 21 features for a number of employees. Some of the features such as Seasons, Day of the Week, Social drinker and Social smoker contain categorical data. Other features such as Transportation Expense, Weight, Height contain continuous data. The target represents the number of hours the employee was absent from work.

For this assignment the two machine learning techniques that will be used are Logistic Regression and Random Forests.

1.1 Logistic Regressor

The Logistic Regression model will be used to predict output categories 0, 1 and 2, thus a multi-class implementation is required. In order to handle the prediction of one of three classes (as opposed to a binary classification), it will use One vs. All. The parameters of this model will be the option to include a bias in the weights, the maximum number of iterations to train the model and the learning rate of the gradient descent. Gradient descent will be used to find the best weight for each feature and compute the probability that an instance belongs to a given class, using the sigmoid function.

1.2 Random Forest

The Random Forest implemented for this assignment is a regressor that takes one parameter (n_estimators) as the number of decision trees to generate. A Decision Tree Regressor has been implemented that takes all observations and a random number of selected features and constructs a decision tree. Each node is split into two, referred to in the code as left and right.

2 Background

2.1 Mechanics of a Logistic Regressor

The fundamental idea for building a Logistic Regressor model is that of determining the weights of features to predict an output.

In order to obtain a probability (a number between 0 and 1), we use the sigmoid function. Where, w is each feature's weight, x represents the features, and $\hat{y}$ represents the predictions. Thus the equation for obtaining this probability is the sigmoid function as can be seen below:

$$\hat{y} = \frac{1}{1 + e^{-w^T x}} \tag{1}$$

Of course, this implies that the best weight associated with each feature needs to be found. To do so, gradient descent is used. At each iteration, we compute the cost with given weights (initialised as an array of zeros) and iterate to find the weights for which the cost is minimal.

The cost function is the following:

$$cost = -y \cdot log(\hat{y}) - ((1 - y) \cdot \log(1 - \hat{y})) \tag{2}$$

The cost for all the instances in one training iteration can be computed as the mean of each instance's cost.

The gradient descent will minimise the value of the cost. At each iteration of the gradient descent, we update the weights by subtracting the gradient, which is the derivative of the cost function with respect to the weights:

$$\frac{1}{m} X^T (\hat{y} - y) \tag{3}$$

This is repeated until the rate of change is as small as the threshold which is given as a hyper-parameter to the model, or until the maximum number of iterations is reached.

Once the model is trained (i.e. the best weights have been found), we compute the predictions by applying the sigmoid function to $\mathrm{w^T x}$.

2.2 Mechanics of a Random Forest Regressor

Random Forests are built by generating decision trees with the aim of obtaining a more accurate prediction. Random Forests can be used both for classification and regression problems. An important machine learning technique that is normally used in Random Forests is Bootstrapping. Bootstrapping is the process whereby a random samples from n observations are selected from the dataset. These random samples create what is called the Bootstrapped dataset. Bootstrapping minimises the risk of overfitting. From the bootstrapped dataset, the model selects a subset of the features and a decision tree is generated. The final prediction of the model is the prediction with the highest vote of all predictions of the decision trees.

2.3 Re-scaling and Normalization

2.3.1 Re-scaling

Re-scaling serves the purposed of changing the spread of the data, as well as the position of the data-points. This is useful for cases were the data is not all on the same scale, and thus this could effect and bias the results of the training and testing to be dependent on specific features.

To perform re-scaling on the data min-max scaling was used. Min-Max scaling performs scaling on a column by using the equation:

$$new\_value = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4}$$

This function transforms the values in a column into a value between 0 and 1, by subtracting the minimum value in the column from the value, and dividing the result by the range.

2.3.2 Normalization

Normalization serves the purpose of making the features have the properties of a normal distribution. Normalization changes the data to be centred around 0, and to have a standard deviation of 1. This is important for comparing measurements that have different scales.

To perform normalization, two methods were implemented: z-score normalization and mean normalization. Z-score normalization is implemented by using the following equation on every column.

$$new\_value = \frac{x - x_{mean}}{\sigma} \tag{5}$$

The mean normalization technique is implemented by using the following equation on each feature.

$$new\_value = \frac{x - x_{mean}}{x_{max} - x_{min}} \tag{6}$$

2.4 Cross Validation

Cross Validation is used to reduce the effect of the random nature of the test/train split, on the result of the model. Cross Validation is performed by creating a number of models (according to the number of folds of the cross validation), fitting each model on different train/test split training data, and testing the model by predicting the results over different train/test split testing data.

Each model is evaluated with a series of evaluation methods, in our case accuracy, precision, recall, and f1_score. The average of the evaluation results over all the folds, is calculated and is outputted.

Thus the cross validation function returns the average evaluation results over different train/test splits of the data.

2.5 Feature Selection and Dimensionality Reduction

Feature selection is the process that identifies and drops features that are related to each other. Related features do not add value to the training process, may reduce the accuracy of the model and may also cause overfitting. The output of the process is a subset of the initial features. Since the number of

features is reduced, the time to train the model is also reduced. The process of feature selection consists of the following steps:

- Compute a matrix with the relationships of each feature with respect of the other features. For n features, an n by n matrix is created.
- Select the minimum threshold of features to keep
- Drop one of the two correlated features that exceed the threshold
- Compute the Principle Components

Two methods used to identify correlated features are applying the Pearson's Correlation or calculating the Mutual Information. In this assignment the former is used. The formula to calculate the Pearson's Correlation is given in Equation (7),

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}} \tag{7}$$

where x and y are vectors representing the features to be tested. The result is a value between zero and one. A zero means that the two features are absolutely not related and one means that the features are identical to each other.

To compute the correlation matrix, a simple nested loop applied on x and y is implemented. A heatmap of the result can be seen in Fig 1.

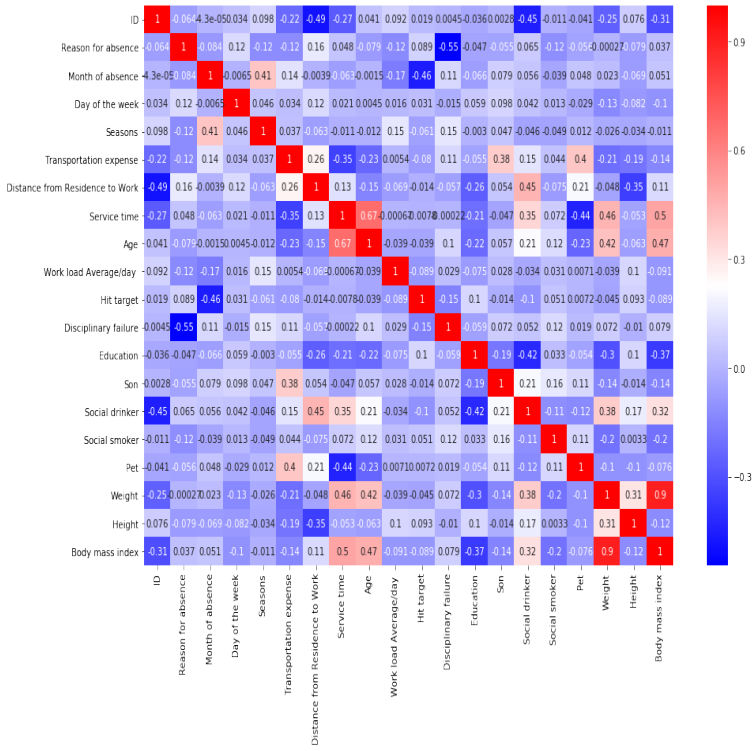

Figure 1: A heatmap of Pearson's Correlation Matrix

In Dimensionality Reduction, the data is mapped into a lower dimension without significant loss of information. Principle Component Analysis (PCA) is the technique used in this assignment to achieve dimensionality reduction. The process follows the steps below:

- Scaling & Normalization
- Compute the covariance matrix
- Calculate the eigenvalues and eigenvectors
- Select a threshold for the minimum variance to keep
- Selected a number of principle components that satisfy the threshold.

Feature normalization is an important early step in PCA. In this step, all the data is altered in a way so that all features lie in the same range. Mean Normalization is implemented as can be seen in equation (6) above. The range for Mean Normalization is between -1 and 1. Fig 2. shows an extract of the dataset after Mean Normalzation has been applied.

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day
0	-0.200502	0.242278	0.056306	-0.228716	-0.514865	0.250631	0.135509	0.015927	-0.111290	-0.184638
1	0.513784	-0.686293	0.056306	-0.228716	-0.514865	-0.382703	-0.353853	0.194498	0.437097	-0.184638
2	-0.429073	0.135135	0.056306	0.021284	-0.514865	-0.156777	0.454658	0.194498	0.050000	-0.184638
3	-0.314788	-0.436293	0.056306	0.271284	-0.514865	0.213594	-0.524066	0.051641	0.082258	-0.184638
4	-0.200502	0.135135	0.056306	0.271284	-0.514865	0.250631	0.135509	0.015927	-0.111290	-0.184638
...	...	...	...	...	...	...	...	...	...	...
735	-0.200502	-0.186293	0.056306	-0.228716	-0.514865	0.250631	0.135509	0.015927	-0.111290	-0.039812
736	-0.486216	-0.293436	0.056306	-0.228716	-0.514865	0.050631	-0.396406	0.051641	0.017742	-0.039812
737	-0.400502	-0.686293	-0.527027	-0.228716	-0.514865	-0.382703	-0.332576	0.015927	0.114516	-0.001568
738	-0.286216	-0.686293	-0.527027	0.021284	-0.181532	0.035816	0.114232	0.051641	0.082258	-0.001568
739	0.485212	-0.686293	-0.527027	0.521284	0.151802	-0.156777	0.326998	0.051641	0.533871	-0.001568

740 rows × 20 columns

Figure 2: Sample data after Mean Normalization

The covariance matrix is computed as an n x n matrix of n features. In this step, the dependency between the features is analyzed. Highly dependant features are represented with positive values and non-correlated features are given negative values. The formula used to compute the covariance matirx is shown in equation (8).

$$Cov(x) = \frac{1}{m} X X^T \tag{8}$$

To calculate the eigenvalues and eigenvectors of the covariance matix, the numpy method **np.linalg.svd** was used. The returned values of S & V represent the variance and eigen vectors respectively. Using a threshold of 0.9, K principle components are identified which preserve over 90% of the information.

Finally, the principle components are calculated from the eigen vectors. The model can then be applied on the principle components which should represent over 90% of the information from the original dataset in less dimensions (features).

2.6 Quantitive Measurements

To evaluate the performance of the Machine Learning models four evaluation methods were implemented.

- Accuracy
- Precision
- Recall
- F1 score

2.6.1 Accuracy

Accuracy is the measure of the amount of total correct predictions, over the amount of total predictions. The equation for accuracy is as can be seen in (9)

$$Accuracy = \frac{Good\ Predictions}{Total\ Predictions} \tag{9}$$

2.6.2 Precision

Precision is used to understand how accurate/precise a models is. It is the measure of True class predictions (True Positives) over the total predictions over a class (Total Predictive Positive). The equation for precision is as can be seen in (10). Thus to calculate precision one needs to calculate the correct predictions of a class, and divide it by this value added to the incorrect predictions over a class.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} = \frac{True\ Positives}{Total\ Predicted\ Positives} \tag{10}$$

For a multi-class implementation, the precision of each class needs to be calculated and then the mean of these precision results in the precision of the model.

2.6.3 Recall

Recall is the measure that needs to be used when there is a high cost associated with false class predictions. Recall calculates how many of the actual data points with a particular class (Total Actual Positives) the model captures by labelling them as pertaining to that class (True Positives). The equation for calculating the recall is as seen in (11).

$$Recall = \frac{True\ Positives}{True\ Positive + False\ Negative} = \frac{True\ Positives}{Total\ Actual\ Positives} \tag{11}$$

Similarly to precision, for a multi-class implementation, the recall for each class needs to be calculated and the mean of these recall results in the recall of the model.

2.6.4 F1 Score

F1 Score is a function of recall and precision. It is required when a balance between recall and precision is needed. It is preferred over accuracy because it is more robust to uneven class distributions. To calculate the f1 score equation 12 is used.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{12}$$

As done for Precision and Recall, for multi-class implementations, the f1 score needs to be calculated for every class and the mean result over the f1 scores, results in the model's f1 score.

3 Experiments

3.1 Processing the dataset

A python file for dataset preperation is created seperately to the python script running the Machine Learning techniques. This Data preperation code has the purpose of loading the data, altering and preparing it, and outputting the data in a CSV file to be used by the Machine Learning models. The dataset is initially loaded from the CSV file.

The target class is categorised into categories as follows: <0 results in category 0, >0 and <10 results into category 1, >10 results into category 2.

Feature selection is then performed with a correlation threshold of 60%. Principal Component Analysis is performed after Feature Selection with a data integrity threshold of 90%. From Feature Selection we result in a dataframe of 19 columns including the target, and after PCA we result with a dataframe of 13 columns including the target.

Normalization is then performed to make the features have the properties of a normal distribution. Scaling is also performed so as to make the spread of the datapoints between the range of 0 and 1. The target class is excluded from both normalization and scaling, so as to not result in a class where the category is a decimal value (i.e. 0.5). This is specifically for the sklearn classifiers which do not allow classes with decimal values.

The resulting dataframe is then outputted to a CSV file so as to be loaded by the python script which implements the Machine Learning models.

3.2 Experiments on Data Preperation

Experiments are carried out to understand what are the effects of certain data augmentations on the results of the ML models. The following are the list of the different experiments performed.

- Categorize c1<0>c2<10>c3, normalization and scaling
- Feature Selection with normalization and scaling
- PCA with normalization and scaling
- Normalization, Scaling, Feature Selection and PCA
- Categorize c1<0>c2<5>c3<10>c4

The first experiment is to perform categorization of the output with categories c1<0>c2<10>c3, followed by normalization and scaling. This results in the metrics in Fig. 3.

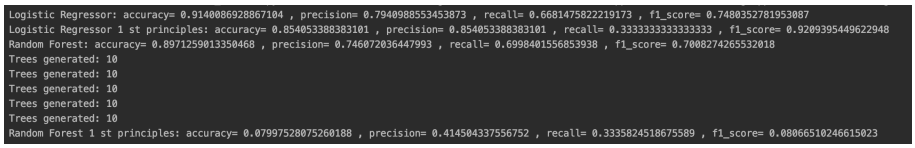

Figure 3: Categories c1<0>c2<10>c3, Normalization, Scaling Results

The second experiment is to perform categorization of the target with categories c1<0>c2<10>c3, followed by feature selection, normalization and scaling. This results in what can be seen in Fig. 4.

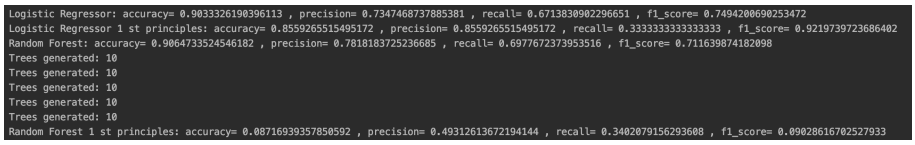

Figure 4: Feature Selection, Normalization, Scaling Results

The third experiment is similar to the second experiment above, but instead of feature selection implement PCA. This obtains the results in Fig. 5.

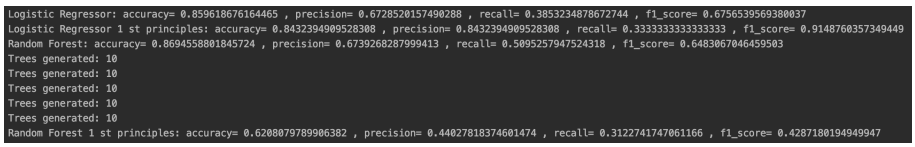

Figure 5: PCA, Normalization, Scaling Results

The fourth experiment implements categorization with categories c1<0>c2<10>c3, followed by feature selection, PCA, Normalization, and Scaling. The results are seen in Fig. 6

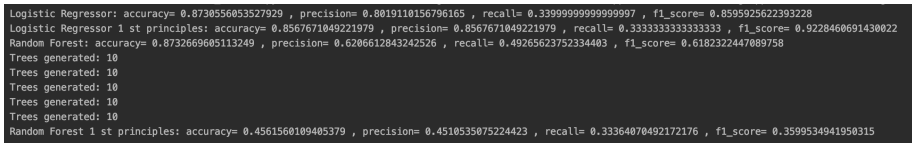

Figure 6: Feature Selection, PCA, Normalization, Scaling Results

The Last experiment is similar to the fourth experiment but implements categorization with categories c1<0>c2<5>c3<10>c4. The results are as seen in Fig. 7

The data preparation that obtains the best overall performance for all models would be experiment four, as can be seen in Figure 6. With the exception

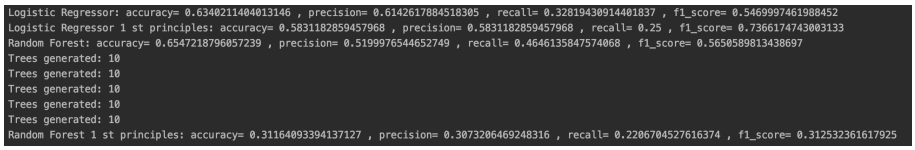

Figure 7: Categorization c1<0>c2<5>c3<10>c4 results

of the First Principle Random Forest, the accuracy obtained when not using any dimensionality reduction, or when simply using Feature Selection, is better than when using PCA. However When using PCA with Feature selection the reduced performance is not as much as when using only PCA. Moreover the best performance is obtain when categorizing into categories c1<0>c2<10>c3, rather than when categorizing into categories c1<0>c2<5>c3<10>c4.

3.3 Implementation - Logistic Regressor

Considering our Logistic Regression model uses One Vs. Rest to handle multi-class output, our model will be implemented using two classes. The first one will correspond to each individual classifier and the second class will predict the final output by comparing each classifier's prediction for each input instance.

For the individual classifiers, we define the class `LogisticRegressor`, which we initialise with:

- Learning rate (default value: 0.01)
- Maximum number of iterations (default value: 500)
- Boolean to add intercept (True by default)

This last parameter enables to add a bias w_0 to the weights. For example, with n being the number of features:

$$w^T x = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

The first parameter, the learning rate, can control the size of the steps in the gradient descent. The higher the learning rate, the bigger the step, which means it will be faster to train, but at the risk of missing the lowest cost value. The lower the learning rate, the slower the gradient descent.

This class also contains the implemented sigmoid function and the cost function, as explained earlier.

Naturally, it has a `fit()` function to train the model, like we previously mentioned. In this function, we also keep a history of the cost values through the gradient descent iterations, which can then be plotted. This was used when developing the individual classifier, but not for the One vs. All model, which we will detail later on. The class finally contains the `predict()` function, to make a prediction for a given input.

The second class, `LogisticRegressorOneVsAll`, will build a classifier for each output category and determine for each instance the classifier that gives the highest probability (and therefore the predicted output for the instance). The class is initialized with the same hyperparameters (which are then passed to each model). It simply contains a `fit()` and a `predict()` function. The fit (training) function builds a model for each possible output, given prepared data (i.e. training data where the "one" class is 1 and the "rest" classes are 0). Each model is trained and added to the list of classifiers, as an attribute of the function. The predicting function predicts an output for testing data with each classifier. Then, it compares the probability predicted by each model for one instance, and returns the predicted class for each instance (as opposed to a probability that the instance belongs to a given class).

3.4 Implementation - Random Forests Regressor

The code to implement a Random Forest Regressor is split into two classes. The first class provides a Decision Tree Regressor while the second class implements a Random Forest by creating a number of Decision Tree regressors. The residual sum of squares (RSS) method is used to select the best feature and the best value for splitting the nodes. The implemented calculation for RSS is:

$$RSS = \sum_{i=0}^{n} (\epsilon_i)^2 \tag{13}$$

A loop iterates through all values of the feature and the RSS is calculated. The minimum value of the calculated RSS values is taken as the best value to split that feature. This process continues for all features in the dataset. Once all RSS values are calculated for all the features, the feature that has the minimum calculated RSS value is chosen as the best feature to split the tree.

The decision tree nodes are stored in a python dictionary. A node is defined with key-value pairs for feature number, feature name, cutoff value, target value and the left and right nodes. Null values for the left or right nodes indicate no further splits or the end of the tree.

The Random Forest Regressor builds a number of Decision Tree Regressors. The number of trees is set in the parameter n_estimators. Bootstrapping is a method used to build Random Forests. In Bootstrapping, random observations are chosen from the full dataset. The bootstrapped dataset may (or may not) contain duplicate observations. Bagging is the process of selecting a subset of features so that a decision tree is built only from the selected features. In this assignment 4 features are selected to generate the Decision Trees. The predict method in the RandomForestRegressor class calls the predict method in the DecisionTreeRegressor class which in turn returns the value at the end of the tree. The value with the highest frequency, of all predicted values from all the trees is then taken as being the predicted value for the Random Forest.

3.5 Comparison of models

From the results in Section 3.2, it can be seen that the 2 models implemented through Sklearn are generally on par with regards to accuracy and precision. However the Random Forest performs better with regards to Recall, and thus the f1 score. With regards to the implementations from First Principle. The logistic regression implementation performs really well, and very closely to the Sklearn implementation. For the Random Forest implementation the best accuracy was when using PCA without feature selection, however there is a distinct desparity in performance between the one from First Principle and the one built by Sklearn.

4 Conclusions

In this assignment, various machine learning techniques were studied, implemented and tested using the Python language. The task was split between 3 students so coordination was very important to ensure that the implementation of the individual contribution works well to produce the final result.

For the chosen dataset, the target contains sparse values. Therefore, the preprocessing step of classifying the target values was very important to achieve better accuracy. In fact, the different experiments conducted using different splits for categorical data resulted in different accuracy given by the two models.

For the Random Forest model, one evaluation method that was left out of this assignment but could have improved the accuracy of the model is calculating the 'Out-of-bag' error. OOB data is the data left out during the bootstrapping process.