ADMYou

Project: Data Science
Rainfall Prediction in Australia

Description of the project :

The aim of this project is to predict whether it will rain the next day in Australia. To do this, we will use supervised learning and more precisely a binary classification. The main steps of the project are: analysis of the dataset, data preparation, model training, model testing and model improvement.

Analysis of the data set :

The analysis of our dataset is done using the methods on Dataframe objects of the Pandas module. The dataset contains 142193 examples and 24 columns. The last column corresponds to the output variable (target variable). The 23rd column, "RISK_MM", is used directly to generate the values of the target variable and is therefore not a characteristic to be kept. Finally, there are 22 characteristics in our dataset. Most of the characteristics are quantitative (float type), but there are still six qualitative variables ("Date", "RainToday"...). The output variable is also qualitative.

We also note that data are missing for a large majority of variables. With the describe() method, we notice that there are outliers for the variable MaxTemp: the maximum value is 99°C.

To get a more precise idea of the distribution of the data, I visualized some variables in the form of a histogram, namely the variables "MinTemp", "MaxTemp", "RainFall",

"Pressure3pm", "Humidity3pm". For most of the variables (except "RainFall"), the distribution is similar to a normal distribution. However, some variables have a higher variance (e.g. "MinTemp" and "Humidity3pm"). It is also easy to get an idea of the mean of each variable with these histograms. However, given the large amount of data, it is difficult to identify the few outliers (for "MaxTemp"), as these are overwritten by the scale effect.

Preparing the data :

This step involves preparing the data for learning the model. The aim is to have quantitative, complete and consistent data. Again, Dataframe methods will be used in this section.

First, we focus on missing values. Variables with too many missing values (more than 47,000 missing values, which corresponds to 1/3 of the data) are removed. For the others, I have replaced the missing values with the mean values of each variable.

The outliers for the variable "MaxTemp" are also replaced by the mean value of this variable.

The value of RainTomorrow, being deduced from RISK_MM, we have to delete the variable RISK_MM which will not be an input variable. On the other hand, the qualitative variables, and also the target variable

"Rain Tomorrow", are transformed into quantitative variables, using the methods of the LabelEncoder class from Scikit-learn.

Feature selection :

I proceeded to the selection of the variables using two transformers, isssus of sklearn.feature_selection: SelectKBest and VarianceThreshold. The first is based on dependency tests. It selects the K variables with the highest dependency test scores with the output variable. I used here the ANOVA test. Finally, this transformer returns the K variables most correlated with the output. I chose K = 14.

The second transformer allows to select the variables whose variance is higher than a fixed threshold. Indeed, the variables which vary more allow the model to learn more efficiently, because the model will then have seen a larger range of values during its learning. After visualising the variances of all the variables in the dataset, I chose a threshold which allowed me to eliminate three additional variables. Among the variables deleted by VarianceThreshold was "RainToday". This was one of the variables most correlated with the output. Despite its low variance, I still decided to add it to the dataset.

12 variables will therefore be used for learning.

Correlations :

I have already started to deal with correlations with SelectKBest and the ANOVA dependency test in the previous section. I am interested here in the classical (Pearson) linear correlation coefficient, defined between two random variables by :

$$r = Cov(X, Y)/ \sigma(X)\, \sigma(Y)$$

where σ(X) and σ(Y) represent standard deviations, and Cov the covariance. It is between -1 and 1.

A correlation coefficient is used to measure the degree of dependency between two variables. The higher the absolute value, the higher the variables are highly correlated. For the correlation coefficient linear, if abs(r) > 0.8, the variables are highly correlated.

I examined the correlations between the selected variables and the output variable. The most relevant input variables for learning are those most correlated with output. Good that the variables selected are those that are most correlated, they have little correlation with the output: the most correlated are "Humidity3pm" (r = 0.439) and "RainToday" (r = 0.324), which remains a moderate correlation. I also checked the correlations between the input variables and the variable "MaxTemp". We can then notice that this variable is strongly correlated with "Temp3pm" (r = 0.962) and "MinTemp" (r = 0,728).

To visualize the correlations, I also used the function scatter_matrix, by selecting the "Temp3pm" variables,"MaxTemp, RainToday and RainTomorrow. You can then see

that there is a linear correlation between "Temp3pm" and "MaxTemp": the point cloud forms a straight line through the origin.

On the other hand, for the other variables, a weak correlation is observed linear.

Train set and test set :

Using Scikit-learn's train_test_split function, I created from the dataset, a train set and a test set. The train set represents 80% of the dataset, while the test set constitutes 20%.

Model training :

The binary classification model that will be used is a model of logistic regression. This algorithm uses the principle of regression linear and applies it to a classification problem. It is based on the criterion of maximum likelihood.

If we note

$$p(y = 1|x) = \mu(x)$$

then, as it is binary,

$$p(y = 0|x) = 1 - \mu(x)$$

We have therefore

$$log(\frac{p(y = 1|x)}{p(y = 0|x)}) = log(\frac{\mu(x))}{1 - \mu(x))})$$ called logit function.

It is then assumed that this logit function is a linear function of x :

$$f(x) = \omega^T x + \omega_0$$

The decision rule is as follows: if μ(x) > 0.5 then y = 1, if μ(x) < 0.5 then y = 0.

Equivalently, we then have: if f(x) > 0 then y = 1; if f(x) < 0 then y = 0.

We can therefore separate the two classes by the hyperplane separator of equation f(x) = 0.

The aim of the algorithm is to maximize the likelihood

$$\prod_{1}^{n} p(y^j|x^j)$$

which means maximising log-likelihood

$$\sum_{1}^{n} log(p(y^j|x^j))$$

There is no analytical solution to this optimisation problem. However, an approximate solution can be found that minimises a certain cost function directly related to log-likelihood. This minimization can be done by means of a gradient descent.

The parameters that are calculated during the learning phase are the coefficients of the separating hyperplane, namely w and w0.

On Python, the logistic regression algorithm is imported with the LogisticRegression() class, coming from Scikit-Learn.

We then train our model with the training set. The score of the model obtained on this set is 0.84.

Evaluation of the model :

I am now testing my model which has been trained, on the test game. While making predictions for the first 20 examples of the test game, 2 errors were made.

To evaluate more precisely the performance of the model, I used several Scikit-learn metrics: accuracy_score (0.837), confusion_matrix, precision_score (0.715), recall_score (0.466) and f1_score (0.564).

The precision score measures the degree of accuracy of positive predictions. Here it is 0.715, so positive predictions (corresponding to rain the next day) are fairly accurate.

The recall_score, or recall, measures the ability to retrieve data from the positive class. It is worth 0.466 here: the number of negative predictions that should have been positive is still significant.

The f1 score combines accuracy and recall. Here it is 0.564: the accuracy and recall of the model are therefore significantly different.

The confusion matrix makes it possible to highlight whether the classification is correct or not. Each row corresponds to an actual class and each column to a predicted class. The values on the diagonal thus correspond to the number of correctly classified values, the other values of the matrix correspond to the number of misclassified values. The confusion matrix is here: (20801 1197 .

3441 3000)

There are therefore 23801 values correctly classified and 4638 misclassified values. One notices the model is particularly good at correctly classifying the values of the positive class ("Yes" for RainTomorrow). However, it is more than half wrong in classifying negative class values.

Improvements of the model :

To improve the model, I use a 5-folds stratified cross-validation. The train set is then randomly cut into several subsets (folds) each containing approximately the same proportion between classes as the complete dataset. At each pass, one fold is used for evaluation and the other four folds for learning. The Scikit-Learn StratifiedKFold class is used for this purpose. The function cross_validate returns several scores from the cross-

validation. The scores can then be averaged over all the passes. We obtain a better estimation of the model performance. The results obtained are as follows: test_accuracy 0.839, test_average_precision 0.668, test_recall: 0.471, test_f1: 0, 568. These values are quite close to the previous scores: the cross-validation did not improve the performance of the model, although there are more examples for learning.

Keeping more variables does not improve the performance of the model. To improve it, variables with higher correlation with the output and higher variance would be needed.

Moreover, logistic regression may not be the most suitable model for our (too complex?) problem. Another type of algorithm could possibly provide better results.