Aprendizagem Automática 23/24

Second Home Assignment

Grupo 33

Authors: Guilherme Cepeda – 62931 horas: 31

Guilherme Rosário – 62543 horas: 0

Marco Viana – 62550 horas: 10

1 - Introduction

The data provided consists of a .csv file with 78095 entries, each containing data regarding a different type of hand posture. Each column has information about the coordinates of the marked fingers of a glove in a hand motion capture environment.

The assignment consists of providing the best possible classification models regarding the target variable "Class", using methods and models covered in class.

2 - Exploratory Data Analysis (EDA)

Our focus in EDA is to understand the data and its structure. We look at the data types, missing values, and other characteristics of the data. Discover and visualize the data to gain insights.

We started our Exploratory Data Analysis by checking if there was any duplicated data in our dataset. We then replaced all the **"?"** values with **"NaN"** values to improve our data analysis, without it our EDA would be less meaningful because we wouldn't be able to analyze the non-numeric columns accurately and observe the full extent of our data, consequence of the columns having more than one type of data in them (column type object). So now having missing values with **"NaN"** instead of **"?"**, allowed us to convert all columns with type object to type float. This is a data processing task done in EDA because it has a large impact on data analysis.

After plotting the histograms for each column, we can conclude that since the 5 values of the target variable ***Class*** have similar distribution the target dataset is relatively balanced and so there is no imbalanced data which would need other approaches and techniques. We also can verify the column user goes from 0 to 14 but skips the 3, so there are only 14 users, this feature is numerical and should be categorical.

In our analysis, we observed the columns X11, Y11 and Z11 had a very dispersed range of values as well as a very low total count of data in each column with only 32 values non-null out of 78095.

The next step was checking/identifying outliers and anomalies in the data, with that in mind we created a boxplot for each column, then we found outliers in most of the columns present in the dataset (except for columns Class and User).

Lastly, we verified the correlations between the features of the dataset and the target variable ***Class,*** and based on the correlation function we could understand how much each attribute correlates with the target variable, the 2 more positive correlated attributes are the *Y9* and *Y10* and the most negative correlated is Z7.

3- Data Processing

In data processing we prepare the data for modeling by cleaning it up and transforming it into a format that can be used by machine learning algorithms.

After the **Exploratory Data Analysis,** we are confident in our conclusions about the data analysis and applied them here in the data processing.

We first started by eliminating the first row of the dataset (only zeros) as stated in the assignment statement.

Then we eliminated the last 3 columns of the dataset, X11, Y11 and Z11 as they are 99,9% null values and don’t have impact on the dataset.

The next step was transforming the numeric column ***User*** into a categorical one, we achieved this using the “pd.get_dummies” function from the pandas library that divided the 14 users into 14 different categorical columns named “user_0”, “user_1”, “user_2”, …,”user_14”. After this step we had to remove the initial column ***User*** since it was no longer necessary.

4- Modeling/Selecting and Training Models

We created models that can be used to make predictions or classify new data.

The models were trained in simple cross-validation.

Here we divided the data set into a train set (75% of the full dataset) and a test set (25% of the full dataset.

5- Evaluation

With the objective of providing the best possible classification model out of the following models:

- Linear Models
- Tree Based Models
- Naïve Bayes
- K-Nearest Neighbors

We tested a combination of 3 scalers, 3 imputers and 15 different models with different hyperparameters as seen below, which gave us a total of 135 models tested.

Scalers:

- PowerTransformer()
- MinMaxScaler()
- StandardScaler()

Imputers:

- SimpleImputer(missing_values=np.nan, strategy='mean')
- SimpleImputer(missing_values=np.nan, strategy='median')
- KNNImputer(n_neighbors=3)

Models:

- LogisticRegression(C = 0.01)
- LogisticRegression(C = 1)
- DecisionTreeClassifier(max_depth = 10)
- DecisionTreeClassifier(max_depth = 20)
- DecisionTreeClassifier(min_samples_leaf = 5)
- DecisionTreeClassifier(min_samples_leaf = 10)
- DecisionTreeClassifier(criterion = 'gini')
- DecisionTreeClassifier(criterion = 'entropy')
- GaussianNB()
- KNeighborsClassifier(n_neighbors = 3, algorithm = 'ball_tree')
- KNeighborsClassifier(n_neighbors = 3, algorithm = 'kd_tree')
- KNeighborsClassifier(n_neighbors = 5, algorithm = 'ball_tree')
- KNeighborsClassifier(n_neighbors = 5, algorithm = 'kd_tree')
- KNeighborsClassifier(n_neighbors = 5, algorithm = 'ball_tree',weights = 'distance')
- KNeighborsClassifier(n_neighbors = 5, algorithm = 'kd_tree',weights = 'distance')

We tried 2 approaches for the data imputation, the Univariate imputation with the **SimpleImputer** with 2 different strategies *mean* and *median*, and one Multivariate **KNN** imputation which is slower with all the missing values present in this dataset.

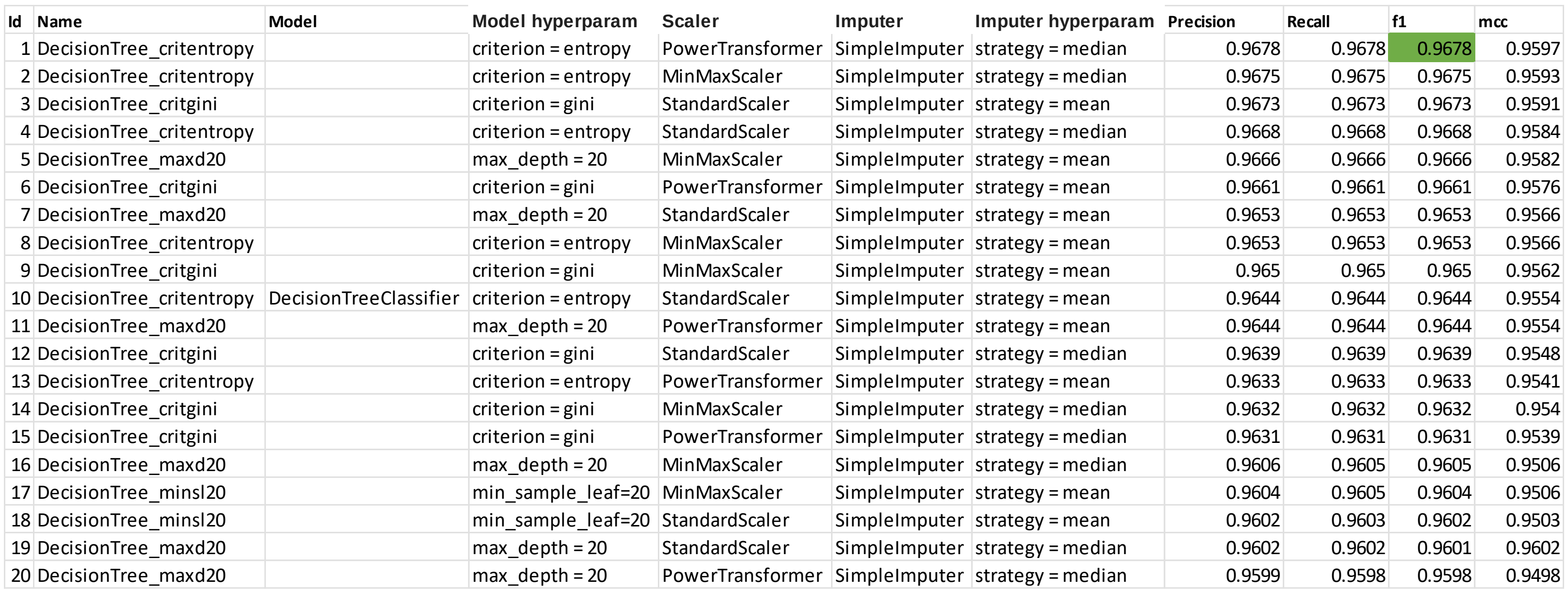
We chose the *C* = 0.01 and 1 in the **LogisticRegression** Classification model because the dataset has a good number of features and low *C* values have a stronger regularization and helps prevent the overfitting of the model.

In the **Tree Based Models** we tested a diverse set of hyperparameters using 2 values for the *max_depth* and *min_sample_leaf* hyperparameters and 2 other simpler hyperparameters *criterion = gini and criterion = entropy.*

Only one Naïve Bayes Model, that assumes that each variable is normally distributed and therefore can be modeled as a Gaussian.

We know from a baseline that the **KNN** model is intrinsically slower than most supervised learning models and that sometimes it can become impractical for a large dataset and the one on this assignment is a relatively large dataset, so bearing that in mind we used the **Ball tree** and the **K-Dimensional tree algorithms** as a strategy used mostly in large datasets.

The evaluation metric used to evaluate the classification models performance was the **F1-Score.** Below we have a table with the top 20 models.



Id	Name	Model	Model hyperparam	Scaler	Imputer	Imputer hyperparam	Precision	Recall	f1	mcc
1	DecisionTree_critentropy		criterion = entropy	PowerTransformer	SimpleImputer	strategy = median	0.9678	0.9678	0.9678	0.9597
2	DecisionTree_critentropy		criterion = entropy	MinMaxScaler	SimpleImputer	strategy = median	0.9675	0.9675	0.9675	0.9593
3	DecisionTree_critgini		criterion = gini	StandardScaler	SimpleImputer	strategy = mean	0.9673	0.9673	0.9673	0.9591
4	DecisionTree_critentropy		criterion = entropy	StandardScaler	SimpleImputer	strategy = median	0.9668	0.9668	0.9668	0.9584
5	DecisionTree_maxd20		max_depth = 20	MinMaxScaler	SimpleImputer	strategy = mean	0.9666	0.9666	0.9666	0.9582
6	DecisionTree_critgini		criterion = gini	PowerTransformer	SimpleImputer	strategy = mean	0.9661	0.9661	0.9661	0.9576
7	DecisionTree_maxd20		max_depth = 20	StandardScaler	SimpleImputer	strategy = mean	0.9653	0.9653	0.9653	0.9566
8	DecisionTree_critentropy		criterion = entropy	MinMaxScaler	SimpleImputer	strategy = mean	0.9653	0.9653	0.9653	0.9566
9	DecisionTree_critgini		criterion = gini	MinMaxScaler	SimpleImputer	strategy = mean	0.965	0.965	0.965	0.9562
10	DecisionTree_critentropy	DecisionTreeClassifier	criterion = entropy	StandardScaler	SimpleImputer	strategy = mean	0.9644	0.9644	0.9644	0.9554
11	DecisionTree_maxd20		max_depth = 20	PowerTransformer	SimpleImputer	strategy = mean	0.9644	0.9644	0.9644	0.9554
12	DecisionTree_critgini		criterion = gini	StandardScaler	SimpleImputer	strategy = median	0.9639	0.9639	0.9639	0.9548
13	DecisionTree_critentropy		criterion = entropy	PowerTransformer	SimpleImputer	strategy = mean	0.9633	0.9633	0.9633	0.9541
14	DecisionTree_critgini		criterion = gini	MinMaxScaler	SimpleImputer	strategy = median	0.9632	0.9632	0.9632	0.954
15	DecisionTree_critgini		criterion = gini	PowerTransformer	SimpleImputer	strategy = median	0.9631	0.9631	0.9631	0.9539
16	DecisionTree_maxd20		max_depth = 20	MinMaxScaler	SimpleImputer	strategy = median	0.9606	0.9605	0.9605	0.9506
17	DecisionTree_minsl20		min_sample_leaf=20	MinMaxScaler	SimpleImputer	strategy = mean	0.9604	0.9605	0.9604	0.9506
18	DecisionTree_minsl20		min_sample_leaf=20	StandardScaler	SimpleImputer	strategy = mean	0.9602	0.9603	0.9602	0.9503
19	DecisionTree_maxd20		max_depth = 20	StandardScaler	SimpleImputer	strategy = median	0.9602	0.9602	0.9601	0.9602
20	DecisionTree_maxd20		max_depth = 20	PowerTransformer	SimpleImputer	strategy = median	0.9599	0.9598	0.9598	0.9498

Table 1 – Top 20 classification models performance

The best classification Model obtained was the **DecisionTreeClassifier(criterion = entropy), Scaler PowerTransformer() and Imputer SimpleImputer(strategy = median).**

The simpler model got the best results, it's a Tree Based Model the **DecisionTreeClassifier** with the hyperparameter *criterion = entropy.* This makes sense in being the best classification model combined with the **Scaler PowerTransformer()** and a **Imputer SimpleImputer(*strategy = median*)**, as we found in the EDA the dataset has outliers and the PowerTransformer() Scaler handles outliers by transforming data into a normal distribution via a power transformation, it compresses them making the data less dispersed.

After analyzing the results, we noticed as expected that the **KNN imputer** has generally worse results because it essentially uses the closest instances with common feature to infer the missing values (NaN values), in this dataset we had several missing values so the use of this imputer compromised the speedy use of the various combinations of models and its performance.

The **KNeighborsClassifier()** model, is generally pretty slow compared to the other models because it searches for neighbors in the full dataset, still the use of the **Ball Tree and K-Dimensional tree algorithms** in this dataset produced decent results always with a **F1-Score** greater than 0.92.

The **LogisticRegression()** model, the best combination is **Scaler StandardScaler()** and **Imputer SimpleImputer(strategy = mean)**. The **F1-Score** of most combinations is always greater than 0.75 which turns out to be a low score compared to the top 20 classification models **F1-Score** as seen in table 1.

The Naïve Bayes model **GaussianNB(),** had extremely poor results in all of its combinations with different scalers and imputers because Gaussian Naïve Bayes algorithm is mainly used for features that have continuous values and for features that are normally distributed, with the lowest results being the combination of the **StandardScaler()** and the **KNNImputer().**