Homework 5 _ Andrea Tricarico *UIN 667299954*

GitHub rep:https://github.com/Trik17/MachineLearningProject (Look at the ReadMe to know how to run the .py)

I started my work from a CSV file taken from Kaggle, it contains information about the answers given from 1010 student (rows of the database) to 150 questions (features, columns). Among these questions there is one in particular “I am empathetic person.: Strongly disagree 1-2-3-4-5 Strongly agree” that is what I want to predict using the answers given to the other questions, I’m only interested in predicting if a student is very empathetic (value of 4-5) or not very empathetic (1-3). The database contains missing values, so the first thing that I’ve done is to deal with them. I’ve eliminated the rows where the feature “empathy” was missing because those data are not useful for neither training or testing. For the other missing values, I’ve imputed them using the mean value of each feature for the integer ones (because most of the algorithms uses them as floats) and the mode for the categorical values. Then, in order to deal with the categorical features that can’t be used from most of the algorithms I’ve transformed them into numerical ones. I used one-hot-encoding for the categorical features “Genger”, “Left_right_handed”, “Only_child”, “Village_town”, “House_block_of_flats” that represents actual categorical characteristics of a student. For the others, I’ve transformed them in numerical features manually assigning at each value a number, because they actually represent scales of value, like the Smoking attribute where the scale start with “never smoked” and end to “actual smoker”, so using one-hot-encoding is not correct because it will lead to lose this “scale” of values.

My baseline is a model that predict always the most frequent class of the training set and it reaches an accuracy of 66%. I’ve built various models then and I’ve evaluated them using accuracy and for the best one also the F1 score. I split the dataset in 2 parts, training set (80%) and test set (20%). I have then used the training set to tune also the hyperparameters using cross-validation during the choice of the models. I haven’t divided the original dataset in three parts because it was very small and dividing it in three parts lead to having a too small training (bad models) or testing set (not accurate results in evaluation of the models).

I put in my project (you can find all of them in the python notebook) 7 models and then another version of two of them that uses a feature selection before the training. The best models are model 3 and model 4 with the feature selection. They are the best models as shown by the results, they have both high accuracy and F1-score, so they have both reasonably high precision (few false positive) and recall (few false negative). Both the models are ensemble methods, and this is very important for a small dataset like the one of this project.

Model 3 is a Random Forest with the hyperparameter tuned using a cross validation on the training set (best way to use the dataset at the best) using the function GridSearchCV of Sklearn. Random forest is a learning ensemble consisting of a bagging of unpruned decision tree learners with randomized selection of features at each split. This model is resistant to noise and to overfitting. With this model I have reach a testing accuracy of 76% and an F1 score (on the testing set) of 0.842.

Model 4 with feature selection is a XGBoost model where the hyperparameters have been chosen in the same way as the model 3. XGBoost is an efficient and scalable implementation of gradient boosting applied to classification and regression trees, it approximates tree learning using quantile sketch and it is a regularized model that control overfitting. The feature selection that I made for the training (and testing) data for this model has been done training a RandomForestClassifier and taking from it the list of the most important features, sorting them in order and selecting the most important ones. I plotted the cumulative importance of those features and using the plot I have decided to take the first 120 of them. Model 4 reaches and accuracy of 73.4% and a F1 score of 0.82, and with the feature selection the accuracy improved to 74%.

Looking at the importance of the feature and at some particular cases it is possible to understand the reasons why the models predict correctly or not. For example the model 3 make a wrong prediction on the second line of the dataset (index 1), this because between the most important feature that the model uses there are 'Friends versus money' and 'Compassion to animals' where this example has, for both, an high value (4 over 5, it is intuitive also for a person that these values indicate a great empathy) so this is an example of an outlier for these features and our model probably has not recognize other feature to eliminate this type of outlier. Indeed, the next example (index 2) has similar values for those features and it is correctly classified as a person with high empathy. A model that probably would be better for this type of classification is a model based on Deep Neural Networks, but unfortunately with a small database it is difficult to train in a good way such a model (as shown in the Jupiter notebook).

I’ve used various libraries: Pandas for dataframe management, NumPy for numerical computations, Seaborn for visualization, Sklearn for lots of the models that I’ve used and for functions like GridSearchCV or the F1score function, XGBoost for the creation of model 4 and TensorFlow to build a Neural Network (present only in the notebook). During my work I used also tools like Anaconda and PyCharm.