For this project we were given the 'Young People Survey' dataset from Miroslav Sabo. It contains the answers to a survey from Slovakian people between 15 and 30. The Dataset is composed by 1010 examples (rows) and 150 features (columns). This dataset with a limited number of examples with respect to a correspondent high number of features corresponds to the main difficulty for the task of predicting a column of the dataset, in particular the column Empathy.
More in depth, the task was referring to predict whether a person is 'very empathetic' (classes 4 and 5) against 'not very empathetic' (classes 1,2,3), so I thought it was meaningful to reduce the problem to a binary classification problem, treating the column 'Empathy' a binary value, mapping 4 and 5 to value '1' and 1,2,3 to value '0'.
Before starting to code, I tried to reason on the problem and I came up with the idea that to have good performances I needed to reduce the number of features or at least to use an algorithm that automatically performs feature selection, due to the fact the number of examples was so small. So everything I did in my experiments had this aim.
At the beginning I thought that, since a lot of features are in 1-5 range, I could apply the same mapping to all these features, but then I realized that in doing this I would have lost the information given by the ordinality of attributes.
Considering this, I decided to convert just all the features that has string values into an appropriate mapping to keep the order, for example for 'Smoking' I did the following conversion 'Never Smoking' -> 1, 'tried smoking'-> 2, 'former smoker'-> 3, 'current smoker'-> 4. Instead, for attributes that had just two possible string values I mapped them into binary variables 0/1.
Then, First of all, I had a look to the data and I decided to eliminate a couple of rows in which I found outliers values for Height and Weight cause they could be noisy, done that, I Imputed the missing values by replacing them with the median of the correspondent column, to fit their categorical nature and not to affect too much the data distribution. Also considering the size of the dataset I decided to split the dataset just into training and test sets using hold-out technique (80% training, 20%testing), setting the random seed to '0', so to compare every approach with respect the same train and test sets, as far as the hyperparameter tuning are concerned, I decided to perform 10-fold Cross validation on the training set, in order to avoid to waste a portion of the limited 1010 examples I had for the dev set. I kept this set up for evaluating the performances of all approaches I tried out.
I worked exclusively with python notebooks and in particular I have largely drawn from Sklearn libraries for my experiments, as well as Numpy, Pandas and Matplotlib and Seaborn for visualization.
I decided to use the accuracy to evaluate my tests and first of all I checked the accuracy of the dummy classifier that always predicts the most frequent class and it was around 66%, I took this as baseline classifier.
After this step I tried to do a little bit of feature exploration through Pearson coefficient and I discovered that the features were generally uncorrelated among themselves.
Given this, I thought to perform Principal Component Analysis, to try to reduce the number of features by finding the directions along which the most part of information was, I built up a new dataset, and I evaluate a Random forest applied to this dataset, obtaining around 70% of accuracy.
Even if I tried other algorithms, such as KNN (I thought it was worth it, due to the limited size of the dataset) whose performances were very prone to the different dataset split, In general random forest seemed to be more robust and stable, so I decided to go with this algorithm, also because it performs feature selection by itself. However, I tried different approaches with Random Forests. Firstly, I trained a Random Forest classifier on the standard preprocessed dataset, then I picked the best features out of this classifier and I trained another Random Forest classifier using just these best attributes but I resulted in overfitting, indeed my accuracy on the training set was around 99%, but performances on test were good, around 72%. Secondly, I performed cross-validation on the training set to tune the hyperparameters and I trained a Random Forest classifier on the standard preprocessed training set, and I tested it on the test set generated with the hold-out method. At first, using the hyperparameters given by the cross validation, I got a good accuracy, around 73% but I was overfitting, so I had decided to change the algorithm, I went with Ridge classification because it performs L2-regularization. I thought it would have been meaningful to reduce the magnitude of the attributes considering that in our dataset on one hand we have features like Height, Weight and Age that can assume a wide set of values, on the other hand a lot of variables with values in the range 1-5. Firstly I performed cross-validation on the training set to tune the hyperparameter alpha, then I trained a ridge classifier on the train set and I got a quite high accuracy on the test set, around 72%, similar to random forest but with performances on the training set around 78%, so overfitting less. At the very end, always aiming to reduce the number of features, I tried to apply the 'selectKBest' algorithm of scikit-learn on the dataset before performing the Ridge classification, I used both chi-squared and f-classification measures and it turned out that the latter worked better than all the other approaches.
In fact, I got an accuracy 77% on the test set and 75% on the training set, it seems I was underfitting a little bit, probably, due to the split, in fact by trying it on a different split I got 69% on the test set and 75% on the training set. This is the solution I have chosen, I think it is the most appropriate because it gives me the highest accuracy on the test set but mostly it represents the one that generalizes better, as it is the one in which we have the lowest training accuracy with respect to test one.
REFERENCES:
For this project I had a look to a few Kaggle discussions but I couldn't find something really useful, I also discussed with my classmates but mostly I drawn from the experience I got from this and the Data Mining course, during which I also did this project: https://github.com/MatteoMarziali/DMTM-project-BIP.
PROJECT REPOSITORY: https://github.com/MatteoMarziali/CS412---Machine-Learning-project-.