Analyzing the Reality Mining Data using supervised classifiers

Nerea Losada **Maitane Martinez**

Abstract

In this document we report our proposal for the application of supervised classification methods to the Reality Mining dataset. The task that we have chosen is classifying the data depending on the subject's affiliation. The classifiers that we have selected for the classification task are Decision Tree, K-Nearest Neighbor and Random Forest. We have implemented the classification process using mainly *scikit-learn* and *pandas* libraries. In addition, we have read the dataset, prepared data for classification and learned the classifiers using the train data and computing the score in the test data. From our results the best score has been produced by the K-Nearest Neighbors classifier.

Contents

1 Description of the problem

The task we have to solve is to apply classifying algorithms based on the recorded information and later analyze whether the classification corresponds to the different affiliations of the subjects.

The goal of this experiment is to explore the capabilities of the smart phones that enabled social scientists to investigate human interactions beyond the traditional survey based methodology or the traditional simulation based methodology, which is introduced in [1]. For that aim, we have some features describing different attributes, so after choosing our task, we selected the most relevant ones to solve our problem, which is to predict the class of affiliation among seven different types.

The database has the following characteristics:

- 33 attributes.
- 7 classes: 'mlgrad', '1styeargrad', 'mlfrosh', 'mlstaff', 'mlurop', 'mlurop'. 'professor', 'sloan'.
- 94 instances.
- The data is not balanced. The number of instances in each class are: 36, 15, 6, 6, 3, 1, 27.

2 Description of our approach

We organized the implementation of the project according to the tasks:

1. Reading the dataset
2. Preprocessing of the dataset.
3. Define and learn the classifiers using the training data.
4. Design the validation method to evaluate the score of the proposed classification approaches.

2.1 Reading the data

The data is in a *.mat* file that contains two main structures: *S* and *network*. We have use the *S* structure to obtain the features we have selected.

First of all, some instances have some empty arrays in the 'GROUP' feature, which contains the subject's research group, those we substitute by the *NaN* value. This value is a special value defined in *numpy* library [2]. Furthermore, the structure of this feature for the rest of the instances is an array of arrays, in which there is just one value. So to get those values, we take the first (and only) array of the feature's array and we take the value it contains.

In the case of 'REGULAR', 'PREDICTABLE', 'TRAVEL' and 'TEXT' features we preprocess them in the same way as previously explained. The 'REGULAR' feature has information about the subject's working schedule, either if he/she reported having a regular one or not. The 'PREDICTABLE' feature tell us if the subject reported having a predictable schedule. The 'TRAVEL' feature contains if the subject reported often travelling. Finally, the 'TEXT' feature gets the value of how often the subject reported sending text messages.

Apart from that, the 'LOCS' feature, which contains the unique set of towers seen by the subject, has an array structure. To deal with it, if the array is empty, we assign *numpy.NaN* value, and if it has a list of arrays, we take the most frequent value, this is, the ID of the tower that the subject has seen most. This way we can know more or less where the subject usually is. For it we have created a function ($mostfrequent()$): first, we use the function *numpy.concatenate* [3] to put together the sequence of arrays in just one array. After doing so, we convert the value to *int* to remain only the *areaID* value. At last, we use the *collections* library to find the most repeated value.

In regard to the 'APPS' feature, it is the one that contains the total number of times each app was used, and we get the most used application for each subject. For that aim we follow a similar process that for the previous feature: we obtain the list of arrays, concatenate them and count the occurrences, getting the one that most appears.

The D_i features are the most complex ones. We obtain the inferred locations at each hour of the day from the given data. Therefore, we get 24 new features that tell us where the subjects are at every hour of the day. To fill the information of those features, we take the most frequent location of each subject at each hour, since we have information of many days. To do so we create a function that first takes away the 0 and the *numpy.NaN* values, by using *numpy.logical_not* function [4]. If the list is empty after removing those values, it returns *numpy.NaN* value; otherwise, it returns the most repeated value using [5] library.

The 'SMS' feature has the number of text messages sent and received. The structure is composed by an array of arrays, so to get the value, we take the one in the first position of both arrays. The same happens with the feature 'VOICE', the one that has the number of voice calls made and received.

Finally, to obtain the 'CLASS' values, we have to check if the array is empty. If it is, we assign *numpy.NaN* value, if not, we get the first position of both arrays, as we have done with the previous feature.

2.2 Preprocessing

We have used the *pandas* library to represent the dataset as a dataframe. We have converted the values of the affiliation classes, that were represented as strings, to ordinal values between 0 and 7, as we have seven classes.

For preprocessing the dataset we have chosen different techniques. On the one hand, we have observed that the data is not always represented in the same way, that is, for the same value we can have two or more different strings. On the other hand, we considered that in some cases is more relevant to group the data.

In 'CLASS', there is a mistake in the data: it is a single value call 'grad' and another 'sloan_2'. These values should be 'mlgrad' and 'sloan', so we have grouped them in order to resolve the problem.

For the 'GROUP' feature we have make groups. The first is compose by 'wakeborders', 'water skiers', 'waterskiiers', 'Waterskiers', 'Wakeboarders', 'Windsurfers', 'Kiteboarders and 'Surfers'. We can observe that they are similar sports or they are written in different ways, but with the same meaning. Another group is formed by 'Caribean Snorklers' and 'Snorkelers', because of the similarity of their names. The remaining groups are those that have the same word but misspelled, such as 'chris', 'chrisc' and 'chriss', or 'john' and 'johm'. With the single values we make different groups.

In 'REGULAR' and 'PREDICTABLE' features we have three answers: 'very', 'somewhat' and 'no at all', so we convert each of these to a number between 1 and 3. In some cases, the value of 'very' is written as 'Very' instead of 'very', so we deal with that problem by keeping it in mind while preprocessing the data.

In the 'TRAVEL' feature we have four answers: 'Very often - more than a week/month', 'Often - week/month', 'Sometimes - several days/month' and 'Rarely - several days/term', so once again we convert each of these to a number between 1 and 4.

It is similar with the 'TEXTS' feature. We also have different values to represent the same, like 'often' and 'Often - once/day'. In addition, we have a new value that we convert to the number 5: 'never'.

In the 'APPS' feature, after taking the most used application from each subject, we have just two most used apps: Phone and ScreenSaver. So we replace the values for 1 and 2.

Regarding the 'SMS' and 'VOICE' features, since the number of sms sent or voice calls of each subject doesn't give much information itself, we have grouped them by ranges.

When we get together all the frames, we observe that there are rows without or with few data, so we delete them since they don't give information about the class.

The data has been split into two different sets: train and test, for validation. We use the same train set to learn all the classifiers, and the same test set for evaluating their score.

2.2.1 Missing data

Knowing that the dataset is based on real data, there is a lot of missing data. These missing data are due to two main errors: data corruption and powered-off devices. On average we have logs accounting for approximately $\%85.3$ of the time that the phones have been deployed.

To solve that problem, we have replaced NaN values. Depending on the type of information that the feature gives, we have replaced it by two different methods of the *imputer* function [6] : the median and the most frequent.

On the one hand, in the case of the 'GROUP' feature we use the most frequent value considering that big groups would have more people. We use the same criteria in 'LOCS' and 'APPS' features. In the first one because the most seen tower would be the one that most people will see, and in the second one, due to the fact that most of the people use the same applications.

On the other hand, for the 'REGULAR', 'PREDICTABLE', 'TRAVEL' and 'TEXTS' features we use the median, since the value in the middle would be the most balanced one.

2.3 Classifiers

In accordance with our task, we have chosen three classification algorithms that are relevant for the type of classification problem, this way we can compare the scores got by those classifiers.

We have chosen Decision Tree, K-Nearest Neighbor and Random Forest.

2.3.1 Decision Tree

[7] The Decision Tree classifier selects the best attribute using Attribute Selection Measures(ASM) to split the records. Then it makes that attribute a decision node and breaks the dataset into smaller subsets. After that, it starts tree building by repeating this process recursively for each child until one of the conditions matches: 1. All the tuples belong to the same attribute value. 2. There are no more remaining attributes. 3. There are no more instances.

2.3.2 K-Nearest Neighbor

[8] In term of the K-Nearest Neighbor classifier, it is a simple supervised classification algorithm we can use to assign a class to new data point. KNN does not make any assumptions on the data distribution, hence it is non-parametric. It keeps all the training data to make future predictions by computing the similarity between an input sample and each training instance. This way, it computes the distance between the new data point with every training example, by using the distance measures such as Euclidean distance, Hamming distance or Manhattan distance. Then the model picks K entries in the database which are closest to the new data point and it does the majority vote: the most common class among those K entries will be the class of the new data point. In our case, we have used the default value, this is, k=5, so the classifier uses 5 neighbors to do the majority vote. We have tried other k values, bigger and smaller ones, but we get the best result with k=5.

2.3.3 Random Forest

[9] Random Forest is a classifier that evolves from decision trees. It actually consists of many decision trees. To classify a new instance, each decision tree provides a classification for input data; random forest collects the classifications and chooses the most voted prediction as the result. The input of each tree is sampled data from the original data set. In addition, a subset of features is randomly selected from the optional features to grow the tree at each node. Essentially, random forest enables a large number of weak or weakly-correlated classifiers to form a strong classifier. We use it to compare it mainly with the Decision Tree.

2.4 Validation

To validate our results we have tried to compute the cross-validation, but it is not possible to use it with this model. Since the least populated class in the data has only 1 member, which is too few,

col_0	1	2	3	7
1	15	2	0	0
2	5	1	0	0
3	0	0	2	0
4	0	2	0	1
5	1	1	0	0
7	1	6	6	4

Table 1: Confusion matrix produced by the Decision Tree classifier

col_0	1	2	3	7
1	12	1	0	4
2	4	0	0	2
3	0	0	2	0
4	0	0	0	3
5	0	0	0	2
7	2	4	0	11

Table 2: Confusion matrix produced by the K-Nearest Neighbors

the minimum number of members in any class cannot be more than that, and k-fold cross-validation requires at least one train/test split by setting nsplits=2 or more.

Consequently, we have validated our model by computing the *fscore* metric [10]. So we split the data between train and test and compute it for each classifier. We show the results in the section 4.

As an additional validation step we compute the confusion matrices for the three classifiers (tables 1, 2 and 3).

As we can see in the confusion matrices, the predictions are quite bad. Since we have few instances and 7 classes, there is few data for the model to learn. Furthermore, the data is unbalanced, so it is actually difficult for the model to make a good classification.

In all tables we can observe that instances of classes 4, 5 and 6 are not well predicted, due to the fact there are too few instances of those classes.

3 Implementation

All the project steps were implemented in Python. We used *pandas* for reading and preprocessing the dataset, and *scikit-learn* for the classification tasks. We illustrate how the implementation works in the Python notebook *Reality_Mining.ipynb*.

col_0	1	2	3	4	7
1	16	0	0	0	1
2	5	0	0	1	0
3	0	0	2	0	0
4	0	1	0	0	2
5	1	1	0	0	0
7	4	7	0	0	6

Table 3: Confusion matrix produced by the Random Forest

4 Results

The accuracies produced by the Decision Tree, K-Nearest Neighbor and Random Forest classifiers were, respectively, 0.4680, 0.5319 and 0.4042. Therefore, the best classifier was K-Nearest Neighbor.

The results of the computation of the confusion matrices for the three classifiers are respectively shown in tables 1, 2 and 3, as we mentioned before.

Apart from what we have explained in the second section, we can observe that the first class is quite well classified, but the others are not. Most of those classification errors are due to data errors, such as missing data.

The best classifier has been K-Nearest Neighbor, nevertheless the score is just a little bit higher than 0.5, so the value is actually quite low.

5 Conclusions

In our project we have applied the Random Forest, K-Nearest Neighbor and Decision Tree classifiers to the *Reality mining dataset*, introduced in [11]. We have computed the score of these classifiers and observed that K-Nearest Neighbor produces the highest score. We have also observed, from the analysis of the confusion matrices, that all classifiers make a good classification for the first class, which is the most populated, but not for the other ones. This happens since the quantity of data is really important in classification algorithms, and we just have a few instances. Therefore, we can conclude that if we had more instances, this is, more people participating, we would have more data to make classifiers learn, and we would have best results. In addition, having that much missing data makes more difficult to get a good score, but these are limitations of a real-life based data collection.

References

[1] MIT reality commons [http://realitycommons.media.mit.edu/realitymining.html]. *MIT Human Dynamics Lab*.

[2] NaN numpy value [https://docs.scipy.org/doc/numpy-1.13.0/user/misc.html].

[3] concatenate numpy function [https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html].

[4] logical_not function [https://docs.scipy.org/doc/numpy/reference/generated/numpy.logical_not.html].

[5] collections library [https://docs.python.org/2/library/collections.html].

[6] imputer function [http://lijiancheng0614.github.io/scikit-learn/modules/generated/sklearn.preprocessing.imputer.html].

[7] Decision Tree classifier [https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb].

[8] K-Nearest Neighbor classifier [https://en.wikipedia.org/wiki/k-nearest_neighbors_algorithm].

[9] Random Forest classifier [https://www.sciencedirect.com/topics/engineering/random-forest].

[10] f1_score function [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html].

[11] Alex Pentland Nathan Eagle and David Lazer. Inferring social network structure using mobile phone data. Vol 106 (36) pp. 15274-15278, Proceedings of the National Academy of Sciences (PNAS), 2009.