Visual Analytics on Credit Cards Defaults

Maria Ludovica Costagliola, Emanuele De Santis

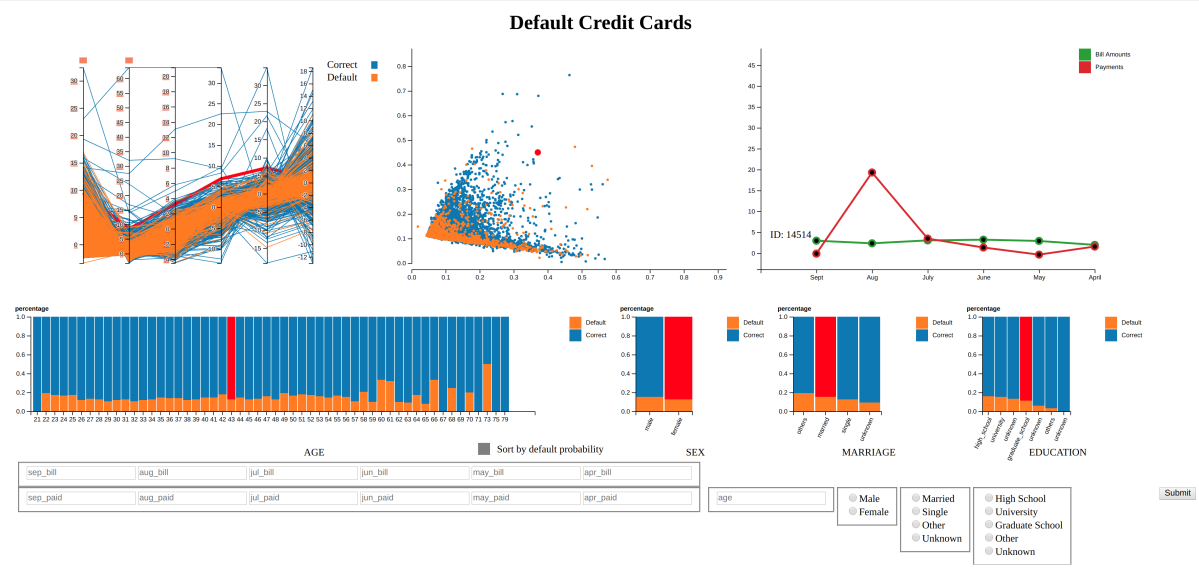

Fig. 1. Sceenshot of the entire interface

Abstract— The project was developed during the Visual Analytics course. It concerns the visualization of credit cards owners data in order to make the bank director knowing the customers that are supposed to not be able to pay the credit card bill in the next month. All data are represented using simple and well-known views that immediately highlights similarities among customers and give to the user an overview on all customers.
The system uses also a machine learning algorithm to classify new customers (manually added using a form) and represents them updating the views.

1 Introduction

After the paper presentation done during the lectures, we decided to focus our attention on a dataset related to bank transactions. Most of the bank transactions datasets are not public available (or they contains few useful information to protect users' privacy), but we were able to find a dataset related to this field.

We were thinking about the need for a bank director to always know how customers with a credit card issued by his financial institution behave. Particularly, we pay attention to the last payments and to the corresponding bank account balances of those customers.

From these data and from some other personal information of the customer (for example age, marriage status, ...), it is possible to identify the ones that probably will not be able to pay the credit card bill in the next month.

The prediction is done by a machine learning algorithm, but the result is useless if it is not combined with an efficient visualization of the whole data. In fact, with this visualization a bank director is able to better understand the result of the machine learning algorithm, considering also the similarity between the result and some preexisting patterns or clusters.

Moreover the bank director may be able to detect default customers even without using any machine learning algorithm, just seeing the information given by the visualization.

2 Dataset

The dataset used in this project is taken from UCI database [2] , it contains about 30000 tuples, each with 24 attributes. Some tuples lack of some values and so they were all removed in order to have a completely useful dataset. A few attributes for each remaining tuple were also removed because they were of no interest for us. We get in this way a more manageable dataset thanks to a lower number of tuples (15337) and of attributes (19). This computation was done by the python function `preprocessing.action()`, that creates a new csv file named `dataset.csv`.

About the used attributes, we can identify four of them that are categorical: age, sex, marriage status, education. These represent personal information about the owner of a particular credit card and they are used for statistical considerations. We have six numerical attributes named *Amount of bill statement*, one for each month from September 2005 to April 2005 and the corresponding six numerical, named *Amount of previous payment*.

The last attribute of each row is the *target*, that is the prediction about the ability to pay on October 2005.

3 Visualization

The visualization that we have developed is constituted by 7 views, each one that shows different aspects of the dataset. All the view are coordinated each other, so clicking on an element of a certain view

will highlight some elements in the other views and/or will show other information. Moreover when a new customer is added, all the views change according to the newly added tuple.

The interface was tought to fit in a FullHD (1920x1080) screen, even if it works also if we resize the browser window (the minimun resolution supported is 1024x768).

3.1 Scatterplot

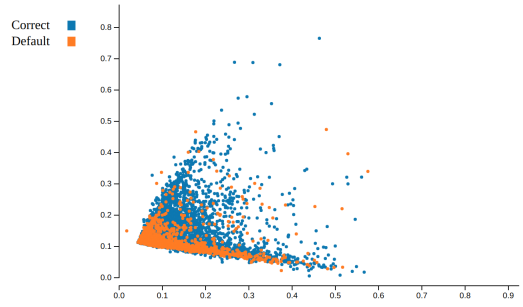

Fig. 2. Scatterplot view of the tool

The scatterplot is the main view in the visualization. Each point represents a customer and the color encodes the classification of the customer (default or correct). The color scaling used is the `d3.scaleOrdinal(d3.schemeCategory10)`.

Due to the high dimensionality of the dataset, it was mandatory to apply a dimensionality reduction algorithm. We decided to apply PCA because it is fast and it doesn't require a deep knowledge of the problem, and we then took the first 2 components. This work is done by the python function `pca.action()` using `pandas` to read and parse the csv original dataset and `sklearn` to compute PCA values. The result is saved in `pca.csv`. A 2D scatterplot is the best way to graphically represent the relation between points and to spot possible clusters. In this case it helps the bank director to quickly see that most of the bad customers are grouped togheter in a particular area (in figure 2 in the left-bottom part of the plot).

Each point has associated two event handlers. The first one allows to make visible a tooltip showing the identifier of the customer just by mouseovering over it; this handler also makes the point bigger to have it more visible. The second handler is associated to click events: by clicking on a point, it will highlight the point itself, will trigger the highlighting of the other related elements and will show up the corresponding lines in the linechart.

Associated to the scatterplot is present a legend explaining the color encoding and clicking on an element of the legend, it is possible to see on the foreground all the points with that classification. Once an element is clicked, a reset button appears in order to restore the original view.

3.2 Parallel Coordinates

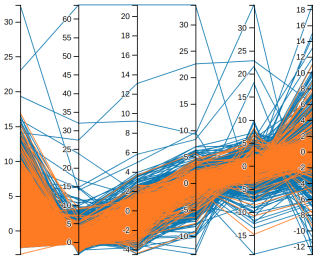

Fig. 3. Parallel Coordinates view of the tool

Parallel Coordinate view is an efficient way to represent data with more than 2 dimensions in a 2D space. Moreover it allows to grasp relations between tuples (for example crossing lines may correspond to a linear relation on some axis).

We have six axis, that correspond to the first six principal components given by the PCA algorithm applied on the original dataset (`dataset.csv`). Each line referrs to one single point that is the graphic representation of a customer. Also in this case we use the same color encoding as before to distinguish between default and correct customers.

There is the possiblilty to click on a line to select it and see the corresponding point in the scatterplot highlighted togheter to all the related information (having the same behaviour we had by clicking on a point in the scatterplot)

Moreover it is possible to exchange the order of the axis, dragging the axis that we want to move. Unfortunately, since the dataset contains a lot of tuples, the movement is a little bit glitchy.

3.3 Linechart

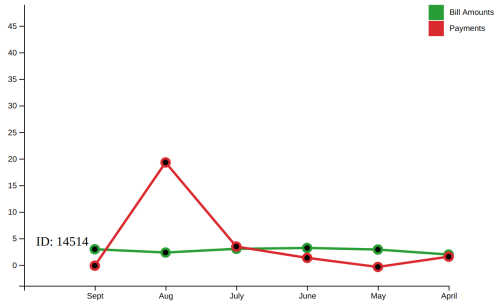

Fig. 4. Linechart view of the tool

Each customer has associated some numerical values that refer to a specific month. We decided to use a line chart to represent these attributes since they are temporal related data. We can distinguish two lines: the green one represents the amount of money the customer had in his bank account in the specified period, while the red one represents the payments that he did with his credit card.

The values that are represented are often not of the same order of magnitude, so we had to compute a normalized version of the dataset (saved in `standardized_data.csv`) using `Standardization.action()` function, that applies a `numpy` standard scaled to all these amounts.

On the plot is also written the ID of the displayed customer and, since the values are normalized we thought it could be useful to retrieve the original amounts in some way. In fact on each line there are 6 circles that can be mouseovered in orderd to see in a tooltip the original value of that point, while on the y-axis are represented the normalized values.

As soon as the page is loaded this view is empty and will be filled with the information described and with a legend of the color used once a point in the scatterplot or a line in the parallel coordinates is selected.

3.4 Stacked Barcharts

Each categorical attribute of the dataset is represented through a stacked barchart. This kind of visualization allows us to see at the same time the frequency distribution of both correct and default customers. There is one column for each different value of the categorical attribute composed by two sub-columns. Each of those represents the percentage of default (on the bottom) and of correct (on the top) instances in the dataset. These data are taken from `*_frequency.csv`.

We added some interactions with this chart. First of all, the element of the legend can be clicked in order to see only the frequency of the chosen classification; after this interaction, it spawns a reset button to restore the original view.

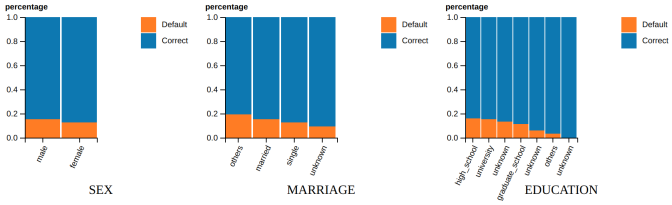

Fig. 5. Stacked Barcharts representing frequency distributions of the categorical attributes sex, marriage and education

A tooltip shows up mouseovering over a single bar; it contains the number of instances, taken from `*_instances.csv`, within its classification and its value for the attribute in exam. We decided to use a tooltip since the height of some bars can be very small, so a number inside them could not be crearly visible or it could overflow to the bars close to it.

Finally, each bar can be clicked: it will be highlighted together with all points in the scatterplot and lines in the parallel coordinates that correspond to customers that have the same value for the attribute and the corresponding classification of the clicked bar.

Once the user clicks on a point in the scatterplot or on a line in the parallel coordinates, the bars, that correspond to the value of the respective attribute for the selected customer, are highlighted in each stacked barchart.

The columns of all the stacked barcharts (except for the Age one) are sorted according to the percentage of default accounts, to improve visibility of catgories of people that are more at risk to be default customers; for the same reason we choose to put the default percentage at the bottom of the stack.

3.4.1 Stacked Barchart: Age

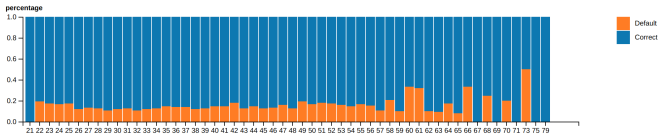

Fig. 6. Stacked Barchart representing frequency distribution of ages

This stacked barchart corresponding to the age behaves in the same way of the others, but it contains something more. Ages have a natural intrinsic order and so, by default, the columns are sorted in that way, but we added the possibility to change the order and have them sorted according to the default percentage.

To make this happen there is a button `Sort by default percentage` to click, the columns will change their order and the button will now permit to go back to the original sorting by age.

3.5 New Data Form

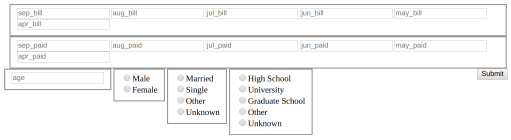

Fig. 7. Form used to add new instances to the dataset

We thought about the possibility to add a new instance to the dataset. The bank director could be interested to know if a customer that is not in the dataset with certain attributes values will be in default or correct.

Through this form, he will insert all the values associated to this new instance. Submitting them, the new tuple will be classified using a machine learning algorithm. The browser page will be reloaded and the new point highlighted in the scatterplot.

4 Analytics

As said before, it is important to let an user insert new instances, but these insertions have to be reflected on the entire interface, that has to be updated according to the new data. Moreover each new instance need to be classified to be correctly represented in the visualization and to update the frequency distributions.

Since Javascript is not so fast handling large amount of data and it is not suitable doing intense computations, we needed to move the analytics part into a python program.

To do so, we set up a small web server using the python library `flask` (that runs locally on our machines) that can communicate with the visualization.

The visualization can be accessed after the server is started at the address `http://127.0.0.1:5000` (or `http://localhost:5000`). It is now possible to pass every field value of the form using AJAX call. AJAX sends an HTTP POST message to a fixed route of our web server, that contains a JSON payload that includes all fields.

As soon as the web server is started, it starts training an SVM learner with all the pre-classified instances present in the original dataset (`dataset.csv`). SVM (Support Vector Machine) [1] is a machine learning algorithm that tries to find an hyperplain that maximizes the margin between itself and the closests points in the high dimensional space (these points are called *support vectors*). In this case SVM works on a kernelized space with an *rbf* kernel (Radial Basis Function), since the dataset is not linearly separable. We allow a certain soft margin where points can be misclassified around the classification hyperplane setting the *C* parameter to 1.

If an user wants to know the accuracy of the classifier, he can run the server putting the last parameter to 1, that makes the python program compute 3-fold cross validation on the trained algorithm. Using this dataset we reached an accuracy of 86%.

Once the web server receives the HTTP POST message from the Javascript code, it creates a new tuple to be added to `dataset.csv`. The new line is appended, and then the function calls all our python modules, that will re-compute all additional csv files, required by the interface.

5 Conclusion

References

[1] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[2] I. C. Yeh. UCI machine learning repository, 2016.