COPERTINA QUI

Sommario

**Nessuna voce di sommario trovata.**

# Introduction

## Goals

The aim of this paper is to explain the choices and the strategies we adopted on the project and development of AirBnb Price Estimator, whose aim is to help owners to decide the most correct price for their B ‘n‘ B.

In order to accomplish it, we started from web-scraped data, we performed all the preprocessing needed for having a suitable dataset and then we built several classifiers, using different strategies, in order to determine the one that predicts best the class attribute. All these classifiers have been tested using more than one method and the analysis of the results guided us in the choice of the best classifier.

Since the class attribute is numeric, we had two possible choices:

* Discretize the attribute, choosing the most appropriate algorithm
* Keep it numeric, using regression algorithms for the classification purposes

The first approach is surely easier, but it would not be as helpful as the second one for our application purposes: suggesting a precise value to a owner will give him/her a more accurate advice rather than a range.

The regression model that generalizes best the class feature has then been chosen as the “heart” of AirBnB Price Estimator: the application asks users to input the required fields that correspond to the attributes needed by the classificatory. On these fields bases, it simply outputs to the user the suggested price for night.

## Initial Dataset

The starting dataset is composed by web-scraped data collected in a csv file. The scraping has been performed the 11th December 2020. The web source is the airbnb.us domain and concerns all the registered B ‘n’ Bs on the Metropolitan Area of New York City (NY). The scraped data has been made available by third parties.

The initial dataset is composed by 36923 instances and 74 columns. In order not to confuse the reader with useless information, the attribute list is not here reported, however on the following chapter regarding preprocessing we justify in detail the actions performed on the data. For now, the most important things to know is that:

* The dataset is composed by various mixed features regarding the estate, the host and the geographical position
* The source file has been published without respecting exactly the csv format, thus some frameworks and csv handlers are not able to parse it
* The initial data is very dirty: there are missing values, redundant attributes, lists of strings embedded in a single column, pointless features and so on. For all of these problems, a suitable solution has been provided and it is fully reported on the next chapter.

# Preprocessing

## What can be preprocessed now?

Since we have to deal with classification problems, before preprocessing data we must be sure not to apply supervised filters on the whole dataset. If we need supervised filters, we must split the dataset in training set and test set before applying them. However in our case there is no need for a supervised filters but the attribute selection, so we performed all the unsupervised preprocessing operations at the beginning, postponing the attribute selection after the training/test split.

## Data cleaning and reduction

Since the Weka framework was not able to parse correctly the source file, for the following operations we used Microsoft Excel and Apple Pages

### Removing irrelevant attributes

The first preprocessing operation is the attribute reduction, indeed we decided to remove all the features that are not domain-specific, nor useful for classification purposes. Among them we deleted:

* IDs like *Bnb ID, scrape\_id, host\_id*
* URLs like *listing\_url, picture\_url, host\_url, host\_thumbnail\_url, host\_picture\_url*
* Information on the scraping like *last\_scraped\_date, calendar\_last\_scraped*
* Useless information on the rent for classification purposes like *name, description, first\_review\_date*
* Useless information about the host like *host\_name, host\_location*(it is not the place in which the rent is, but where the host lives), *host\_about, host\_has\_profile\_pic*

### Removing redundant attributes

The next step was the reduction of attributes explicitly redundant. Not for all of these we did compute the  2 test because of the explicit correlation between the features

* *host\_neighbourhood* and *neighbourhood* w.r.t. *neighbourhood\_cleansed* are explicit redundancies; *neighbourhood\_group\_cleansed* has been demonstrated to be very highly correlated with *neighbourhood\_cleansed* (P{independence}<0.05)
* *host\_total\_listings\_count* completely equal to *host\_listing\_count*
* *host verification* w.r.t *host\_identity\_verified*
* *minimum\_minimum\_nights, minimum\_maximum\_nights, maximum\_minimum\_night,* *maximum\_maximum­\_nights, minimum\_nights\_avg* and *maximum\_nights\_avg* are redundancies of the attributes *minimum\_nights* and *maximum\_nights*
* *review\_scores\_accuracy, review\_scores\_cleanliness, review\_scores\_checkin, review\_scores\_communications, review\_scores\_location, review\_scores\_value* are rounded values that have been more precisely combined in another already-existing attribute that is *review\_scores\_rating*
* *calculated\_host\_listings\_count(+category)* are redundant attributes of *listings\_count* valid only for their respective category

### Other removed features

Eventually, the other columns that have been discarded are:

* Attributes that strongly depend from the instant in which the scraping has been performed like *has\_availability\_now, availability\_30, availability\_60, availability\_90, number\_of\_reviews\_ltm, number\_of\_reviews\_l30*
* Empty attributes like *license*
* Others, like *latitude* and *longitude* that are not more useful than the easier data on the neighborhood

## Dealing with missing fields

Once the previous steps were performed, we achieved a Weka-convertible CSV file. Thus, the following steps have been implemented using the Java Weka API.

The original dataset contains several missing values sparse in more than one attribute. The amount of missing values is enough relevant to discourage the instance deletion, although they are pretty easy to handle, indeed:

* The majority of them are on numeric attributes characterized by low variance, like the *response\_rate* (σ=26.17 on a 0-100 interval) or the *review\_score\_rating* (σ=9.52 on a 0-100 interval). In this cases, replacing the missing value with the mean does not introduces big error rates.
* Some missing values can be easily inferred with the simple analysis by the domain expert: for example the attribute *bedrooms* contains missing values, but only when the corresponding *beds* value is 1: it’s reasonable that the missing value for *bedrooms* is 1 as well.
* Supervised approaches for missing values could have guaranteed more precise results, but since we are exploiting a regression problem and we didn’t split training and test sets yet, using a supervised filter here was an error.

## Data integration and transformation

* + 1. Attribute formats transformation *(+screenshot)*
    2. Dealing with lists *(+codice)*

## Preprocessing implementation

As already mentioned before, the preliminary operations needed to make the csv readable by Weka have been performed using spreadsheet software. The resulting cleansed file still needs some modifications, i.e. :

* As reported in par. 2.3, missing values need to be managed
* As widely discussed in par. 2.4, the *amenities* must be converted in a Hot Vector
* On the *price* attribute, we get rid of the $ sign and we convert it into numeric
* The *bathroom* attribute needs its format to be changed

After the loading of the CSV file, in order to speed up the operations we parallelized them using 4 different threads: the dataset is vertically split and every thread works only on its related partition, using Java Weka API or working directly on the text according to what was the fastest approach in every single scenario; every execution flow, before ending, writes results in a CSV file. The main thread spawns a thread for each of the tasks listed before, and then waits the termination of all of them before merging the results in a single file.

In addition to the benefits of parallelism, this approach promotes the separation of concerns of the tasks: the main method defines the preprocessing pipeline, but the actual operations on data are performed by separated and independent components. All the classes implemented at this point are collected in the *com.unipi.dmaml.airbnbpriceestimator.preprocessing* package.

# Classification

After the preprocessing phase, the dataset is ready to be used to learn regression models that will be used in the final application. In the following chapter all the chosen strategies are discussed and the results are compared.

## Strategies

### Train and test splitting

All the classifiers have been evaluated with the same strategy: 10-fold cross validation. This means that a model is learned from the folds of the training set (90%) and they are evaluated against the corresponding test fold (10%); then the operation is repeated 10 times, and at each iteration a different fold acts as test set.

Despite the fact that Weka provides the Evaluation.crossValidateModel() method, we wanted to save the results of every fold and to have access to the selected attributes in case of algorithm with feature selection.

For these reasons, instead of using a meta classifier, we manually split training set and test set through the Weka provided functions Instances.trainCV(numFolds, currentFold) and Instances.testCV(numFolds, currentFold) according to the WekaWiki tutorial on how to manually realize a k-fold cross validation; then the attribute selection algorithms (when present) have been built exclusively on the training set but applied both to training and to test set.

Finally, we saved the results of each fold, but since we need only a model of the classifier while this procedure generates 10 different models, we picked the one with the lowest *root mean squared error*, aware that this does not necessary means that we chose the best one.

### Chosen classifiers

According to our application domain, the classifiers needed are actually regression algorithms capable of handling our non-nominal class. The selected algorithms have been tested both on the full dimensionality of the dataset (138 features) and on the reduced subspaces identified by the supervised attribute selection methods. Once again, we underline the fact that those features selection algorithms have been modeled exclusively on the bases of the training set. They are *CfsSubsetEval+BestFirst* and *CfsSubsetEval+GreedyStepwise*.

We couldn’t exploit the *InfoGain* evaluator since it is not capable of working with numerical classes; we discarded PCA since it works transforming dimensions, thus it would be difficult to understand if we could have afforded to ask only for a limited number of parameters to input from the user in the final application (e.g. if *CfsSubsetEval+BestFirst* selects only 3 attributes, we can create an application that requires only 3 parameters as input. With PCA we don’t know which original features has been chosen in order to generate the new space, so we are forced to ask the user to input alle the 138 parameters); eventually, we discard the use of a wrapped classifier in order not to deteriorate much the performance of the application.

To summarize, the tested classifiers are:

* *Linear Regression, Linear Regression with attribute selection*
* *Random Forest, Random Forest with attribute selection*
* *5-NN, 5-NN “ “ “*
* *M5Rules* with and without “ “

## Building classification models

On the following pictures the implementation of the classifiers is reported.

The main method is not here shown, it simply loads data and triggers the algorithm. A multithreaded approach has been tested but not implemented since the huge amount of principal memory consumption due to the model building made the system fail more than once

### Immagine che contiene testo, screenshot, computer Descrizione generata automaticamenteData Loading

DatasetFromCsvLoader.java is the class in charge of loading the dataset stored as a CSV file. It simply uses the Weka CSVLoader but the options have been saved in a configuration file, so that to maximize the separation of concerns between the Weka internal representation of the data and our code.

### Immagine che contiene testo Descrizione generata automaticamenteClassifier definition

Every algorithm is implemented in a separate class, which contains different methods based on the way we want to test it (with or w/o attribute selection). In the image the classifier definition is reported: after a randomization of the dataset, we call numFolds time the executeCV() that performs a single round of the cross-validation.

### Immagine che contiene testo Descrizione generata automaticamenteClassifier Implementation

In this method we perform a round of the 10-fold cross validation, in the way suggested by the WekaWiki tutorial. At each iteration, the trainCV and testCV return non-overlapping training sets and test sets; if an attribute selection method has been defined, it is build on top of the training set and applied to both of them and the list of the chosen attributes is stored. In the end, the model built on the fold n.8 is stored in a .model file.

## Performance evaluation and effects of attribute selection

Once the classifiers have been built, as anticipated in the par. 3.1 we tested them using a 10-fold cross validation.

In the following table we report the **average** values respectively of *correlation coefficient(CC)*, *mean absolute error(mae), root mean squared error(rmse), relative absolute error(rae)* and *root relative squared error(rrse)* for each tested classifier (note that the chosen model, i.e. the one built in the 8th fold, shows better measures than the average for every tested classifier)*.*

The green-colored values are the best ones in absolute.

|  |  |  |  |
| --- | --- | --- | --- |
| ATTRIBUTE SELECTION  REGRESSION ALGORITHM | None | CfsSubsetEval  +  BestFirst | CfsSubsetEval  +  GreedyStepwise |
| Linear Regression | Out of memory error | *=0,6793*  *=40,0323*  *=57,8399*  *=66,9399*  *=73,3966* | *0,6794*  *40,0308*  *57,8339*  *66,9375*  *73,3891* |
| Random Forest | *0.7495*  *36,0881*  *52,8727*  *60,3423*  *67,0864* | *0.6988*  *37,9716*  *56,3332*  *63,4916*  *71,7302* | *0,7000*  *37,9451*  *56,5060*  *63,4470*  *71,6955* |
| 5-NN | *0,5413*  *46,4003*  *67,0074*  *77,2522*  *84,3622* | *0,6772*  *39,0709*  *58,0759*  *65,3630*  *73,6845* | *0,6772*  *39,0709*  *58,0759*  *65,3630*  *73,6845* |
| M5Rules | *0,6589*  *567,0975*  *31.491,7374*  *942,2971*  *39.372,3750* | *0,6942*  *38,5873*  *56,7530*  *64,5234*  *72,0102* | *0,6942*  *38,5872*  *56,7498*  *64,5211*  *72,0100* |

As shown in the table, Random Forest without attribute selection is the algorithm that performs best w.r.t. all the parameters. Note that the attribute selection improved the outcomes of certain algorithms, anyway deteriorating the ones of Random Forest.

Now we will discuss the pros and cons of each of them, justifying the final choice for our application:

1. All the classifiers without attribute selection are, unfortunately, not very suitable for the usage of the application: they would require the user to input 288 parameters, which is a huge amount of work to do for him/her. On the contrary, all the attribute selected classifier need less than 30 features.
2. Linear Regression shows good but not awesome results, it also requires few memory for the model loading and it is very fast at prediction time. However it showed very long times and memory consumptions at training time.
3. Random Forest is the one with the best results in absolute, both with and without attribute selection. It is quite fast both to train and to use for prediction, anyway it needs a lot of time and memory for the model to be saved and loaded. Since the considerations at the point 0), we discarded the pure Random Forest and we chose Random Forest with CfsSubsetEval+GreedyStepwise as the default regression model for our application.
4. K-NN is definitely inadequate, although it requires very little time and memory for the model to be built (and then also to be loaded), the computational effort is concentrated at prediction time (thus increasing the application response time), and it also shows quite poor results.
5. M5Rules has very good performances and it needs very little time and memory to save and load the model. The prediction is not so fast, the time required to build the model is terribly long (up to 3 or 4 times more than Random Forest). Anyway, since the majority of the effort is at training time, it is suitable for our application, thus M5Rules with CfsSubsetEval+GreedyStepwise has been chosen as backup regression model.

### Evaluation of significance of the classifiers’ results

Once evaluated the performance of the algorithms, the main question is if we can consider those differences statistically significant or not. In order to make such a computation, we used the paired Student’s t-test on the *RMSE* considering all the outcomes of all the folds (of every algorithm we chose only the *CfsSubsetEval+ GreedyStepwise* version). The results are the following (sig\_rate=5%, deg=9):

* Random Forest wrt Linear Regression: t=5.5342; p=0.004 =>stat. sign.
* Random Forest against M5Rules: t=0.805; p=0.4413 => non stat. sign.
* M5Rules against Linear Regression: t=5.5104; p=0.032 => stat. sign.
* Linear Regression against KNN: t=0.6601; p=0.5257=> non stat. sign.

Random Forest and M5Rules are hence comparable, so they are equally valid for our application. On the contrary, the t-test shows that Linear Regression’s results are much more similar to KNN ones rather than to the outcomes of the other two algorithms, thus it has been discarded.

1. AirBnB Price Estimator
   1. Functional Requirements
   2. *(Forse altri capitoli se avremo cose interessanti da mostrare)*