Classification and Clustering of Yelp Open Dataset

Group Name: MontyPython

Contents

1 Introduction

The following report will show a classification and clustering study using *Yelp Open Dataset*. In this study, in addition to the application of different Machine Learning algorithms, *Natural Language Processing* was also used for text processing. Finally, a web platform will be shown where it will be possible to view the result of applying two clustering algorithms within a map and check whether one's review is considered positive or negative.

1.1 Tasks Description

Each of the three required tasks was done using the *Yelp Open Dataset*, which contains a set of data regarding business activities, users, and reviews issued. In the next sections, we will see:

- How it was possible to recognize whether a written review is considered positive or negative;
- How users are grouped according to their preferences or behaviors within the platform;
- How venues are grouped according to similarity criteria.

1.2 Description of the proposed solution

The solution proposed in the first task differs from those proposed for the other two. In fact, in the first one we will have a combination of the use of *Natural Language Processing* with Supervised Learning techniques. In contrast, in the second and third tasks, Unsupervised Learning algorithms

have been used, thanks to which it is possible to visualize how the elements considered are grouped according to the provided features.

1.3 Literature Review

After several searches for solutions in the existing literature, several papers related to the prediction of reviews were found. However, none of them are similar to our task. Among the main ones we have:

- The first solution found [1] in the literature is very different from the task we have to perform, as it is proposed to provide the number of stars of each review. Consequently, it's not a binary configuration problem like ours, although it starts from the same premise i.e. that of analyzing the text of the review.

- The second solution found [5] uses the reviews dataset to say whether a review is useful or not. So it is a binary classification problem, but very different from ours because it does not only analyze the text of the review.

- In [4], consumer reviews can be used to predict future business success. in the paper aimed to predict if the restaurant will still open till 2017, using yelp dataset from 2016.

- In the paper [2], researchers predict the the sentiment of restaurant reviews based on a subset of the Yelp Open Dataset. They utilize the meta features and text available in the dataset and evaluate several machine learning and state-of-the-art deep learning approaches for the prediction task.

- Finally, in [3] the researchers are looking to the issues associated with setting-up of a new restaurant business. To strategize a new restaurant business, they propose a restaurant business framework that comprises the 3 most important tasks, namely, high-frequency attributes, most crowded day and location of the restaurant.

1.4 Collaboration

The first task was done together, both to acclimate ourselves to the *Python* language, which we had not yet used, and to get a good understanding of the practical use of Machine Learning algorithms. Instead, we implemented the second and third tasks individually (Giulio the second and Silvano the third). Finally, as for the web platform, it was carried out together.
The *"Code With Me"* function within the *PyCharm IDE* (Section 3.2.1) was used to implement the project. Both files with the *.py* extension, where the most frequently used functions were written (to avoid duplication of code), and files with the *.ipynb* (*Jupyter Notebook*) extension were created, which allow writing, in addition to the code, in *Markdown*, for better use and reading of the code.

1.5 Presentation of the results obtained

For each task, different types of algorithms and, for each, different types of parameters were compared to find the best solution.

- Within the supervised learning (reviews task), 7 different algorithms (section 2.2.3) with different metrics were compared. The best was

found to be SVM (Section 3.4);

- As for the tasks related to unsupervised learning in users_task, K-Means and DBSCAN algorithms were used. In both tasks, no direct comparisons were made on the data of the two algorithms, but for each algorithm the best parameters were found and, by comparing 3 different metrics (*silhouette_score*, *calinski_harabasz_score*, *davies_bouldin_score*), the best number of clusters was found (Section 3.3.2);

- In business_task, in addition to what was explained above, the clustering of the two algorithms was also compared visually on a map (Section 3.5.3).

2 Proposed Method

For each of the three tasks, comparisons of various algorithms were performed before reaching the conclusion. This was useful to understand which algorithm returned better output and, as for the Web platform, to have a visual comparison between two clustering algorithms applied to a map.

2.1 Searching for the solution

The following three sections will specifically explain our approach to the three problems mentioned in section 1.1, starting with the choice of datasets used and ending with the use of machine learning algorithms.

2.2 Searching for the solution - Reviews Task

A single dataset, *review.json*, from those available in Yelp Open Dataset was used for the first task. The fields that belong to this dataset are as follows:

- ***review_id***: string, 22 character unique review id;
- ***user_id***: string, 22 character unique user id, maps to the user in user.json;
- ***business_id***: string, 22 character business id, maps to business in business.json;
- ***stars***: integer, star rating;
- ***date***: string, date formatted YYYY-MM-DD;
- ***text***: string, the review itself;
- ***useful***: integer, number of useful votes received;
- ***funny***: integer, number of funny votes received;
- ***cool***: integer, number of cool votes received.

Many rows in the dataset were deleted to allow faster testing. Within *Python*, it is possible to import a *CSV* file as a dataframe element, which gives the ability to manage the dataset as if it were a table. Next, from the *stars* field in the dataset, the *review_rating* column was created in the data frame, which will represent the label column. In the latter column, the value 1 or 0 is added based on the number of stars:

- 1 for rows that have 4 or 5 stars;
- 0 otherwise.

In Figure 1 you can see the number of rows for each of the *review_rating* values.

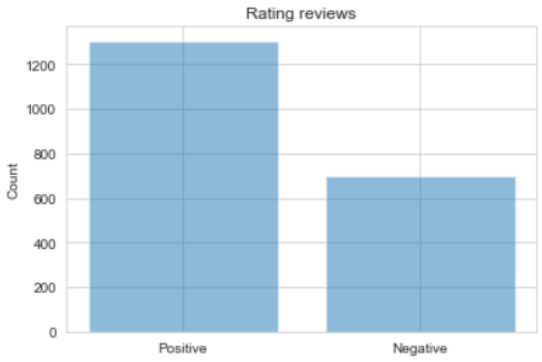


Figure 1: Number of positive and negative reviews

2.2.1 Text preprocessing

Actually, we have only one feature: the review text. To use words as features we need to preprocess text. We created a Python file *text_processing.py* dedicated to this aspect. The main steps are:

- **Normalization**: remove English language contractions;

```
def expand_english_contractions(row):
    expanded_words = []
    for word in row['text'].split():
        expanded_words.append(contractions.fix(word))
    return ' '.join(expanded_words)
```

```
6
```

Listing 1: Normalization function

- **Tokenization**: the text is cleaned of separators and special characters and then broken down into words;

```
def tokenize(row):
    return word_tokenize(row['expanded_text'])

```

Listing 2: Tokenization function

- **Remove punctuation**: remove punctuation from the sentence;

```
def remove_punct(row):
    return [word for word in row['tokenized_text'] if
   word.isalpha()]

```

Listing 3: Remove punctuation function

- **Stemming**: reducing derived words to their base form;

```
def stem(row):
    return [PorterStemmer().stem(word) for word in row['
   no_punct_text']]

```

Listing 4: Stemming function

- **Lemmatization**: reduce the inflected form of a word to its canonical form.

```
def lemmatize(row):
    return [WordNetLemmatizer().lemmatize(word) for word
   in row['stemmed_text']]

```

Listing 5: Lemmatization function

2.2.2 Split dataframe and Feature Extraction

Next, we will split our dataframe into *training_set* (80% of original dataframe) and *test_set* (20% of original dataframe).
In addition, we use the class *TfidfVectorizer()*, from sklearn, to convert a collection of raw documents into a *TF-IDF* feature matrix, where *TF-IDF* is a numerical statistic intended to reflect the importance of a word for a document in a collection or corpus.

```
vectorizer = TfidfVectorizer()
X_train_vectorizer = vectorizer.fit_transform(X_train)
X_test_vectorizer = vectorizer.transform(X_test)
```

Listing 6: Usage of TfidfVectorizer()

2.2.3 Algorithm Selection

To check whether a review is positive or negative, seven Machine Learning algorithms were compared and the one with better results was exported. The algorithms that were compared, and will be explained in section 2.2.4, are:

1. **AdaBoostClassifier**
2. **GaussianNB**

3. **DecisionTreeClassifier**
4. **LogisticRegression**
5. **KNeighborsClassifier**
6. **Support Vector Machine**
7. **RandomForestClassifier**

To avoid code duplication, we created four functions that will be used for all algorithms:

- ***get_best_parameters(alg, params)***: using *GridSearchCV* we can get the best parameters for each algorithm;

```
def get_best_parameters(alg, params):
    gs_algorithm = GridSearchCV(alg, params, verbose=3,
   scoring='f1', n_jobs=-1)
    gs_algorithm.fit(X_train_df, y_train)
    print(gs_algorithm.best_params_)

```

Listing 7: *get_best_parameters* function

- ***fit_and_predict(algorithm)***: to get the algorithm score, we need to pass our *training_set* to use the fit method and then we'll pass the feature columns of *test_set* to predict their values. Finally, we use the function *algorithm.score(X_test, y_test)* to get finally accuracy of the algorithm;

```
def fit_and_predict(algorithm):
    algorithm.fit(X_train_df, y_train)
```

```
    y_pred_algorithm = algorithm.predict(X_test_df)
    print(f"Score on training set: {algorithm.score(
X_train_df, y_train)}")
    print(f"Score on test set: {algorithm.score(X_test_df
, y_test)}")
    return y_pred_algorithm, test_set_score

```

Listing 8: *fit_and_predict* function

- ***confusion_matrix_plot(alg_name, y_pred)***: evaluates classification accuracy by computing the Confusion Matrix which is one of the methods to graphically show how values were predicted by the machine learning algorithm, that is whether there is "confusion" in the classification of different classes. Then we visually represent a confusion matrix using a heatmap plot;

```
def confusion_matrix_plot(alg_name, y_pred):
    confusion_matrix_algorithm = confusion_matrix(y_test,
    y_pred)
    df_confusion_matrix = pd.DataFrame(
confusion_matrix_algorithm, columns=['Predicted 0', '
Predicted 1'], index=['True 0', 'True 1'])
    sns.heatmap(df_confusion_matrix, annot=True, fmt='d')
    plt.title(alg_name + " confusion matrix")
    plt.figure(figsize=(24, 14), dpi=100)
    plt.close()

```

Listing 9: *confusion_matrix_plot* function

- ***classification_report_plot(algorithm_name, y_pred)***: shows the main classification metrics (Precision, Recall and F1-score; section 3.3.1) and displays them with a graph.

```
1 def classification_report_plot(alg_name, y_pred):
2     class_report_alg = classification_report(y_test,
  y_pred, output_dict=True)
3     df_alg_report = pd.DataFrame(class_report_alg).
  transpose()
4     df_alg_report.iloc[:3, :3].plot(kind='bar', title=
  alg_name + 'classification report', rot=0)
5
```

Listing 10: *classification_report_plot* function

2.2.4 Explanation of algorithms used

1. AdaBoostClassifier (Adaptive Boost)

AdaBoostClassifier belongs to the family of Boosting algorithms and it is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

2. GaussianNB

A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It's specifically used when the features have continuous values. These classifiers assume that the value of a particular feature is independent of the value of any other feature.

3. DecisionTreeClassifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualized;
- Requires little data preparation;
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree;
- Able to handle multi-output problems;
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model;

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting;
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble;
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems;

4. LogisticRegression

Logistic Regression is a supervised learning technique that is used to evaluate the relationship between dependent and independent variables by probabilities using a logistic function. These probabilities will be transformed into binary values between 0 and 1 to proceed then with the prediction. This algorithm is used the most in binary classification problems such as predicting whether an email is spam or not, whether a tumor is malignant, or, like in our situation, telling if a review is positive or negative.

5. KNeighborsClassifier

Neighbors-based classification is a type of instance-based learning or non generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

6. Support Vector Machine

Support Vector Machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are:

- Effective in high-dimensional spaces;
- Still effective in cases where the number of dimensions is greater than the number of samples;

- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient;
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and the regularization term is crucial;
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

7. RandomForestClassifier

This algorithm is a supervised learning technique where the "forest" terms point to a set of decision trees, obtained by aggregation using the "bagging" algorithm, that should be trained. Random Forest builds multiple decision trees and merges them to get a more accurate and stable prediction.

2.2.5 Export model

Finally, we exported *SVC* model using Pickle to use it as prediction algorithm.

```
1 pickle.dump(svc, open('svc_model.pkl', 'wb'))
```

```
pickle.dump(vectorizer, open('vectorizer.pkl', 'wb'))
```

Listing 11: Export model

2.3 Searching for the solution - Users Task

As well as for the first task, a single dataset was used from those available in Yelp Open Dataset. The dataset considered for the second task is the one named *user.json*, which contains several fields within it:

- ***user_id***: string, 22 character unique user id, maps to the user in *user.json*;
- ***name***: string, the user's first name;
- ***review_count***: integer, the number of reviews they've written;
- ***yelping_since***: string, when the user joined Yelp, formatted like YYYY-MM-DD;
- ***friends***: array of strings, an array of the user's friend as *user_ids*;
- ***useful***: integer, number of useful votes sent by the user;
- ***funny***: integer, number of funny votes sent by the user;
- ***cool***: integer, number of cool votes sent by the user;
- ***fans***: integer, number of fans the user has;
- ***elite***: array of integers, the years the user was elite;
- ***average_stars***: float, average rating of all reviews;

- ***compliment_hot***: integer, number of hot compliments received by the user;
- ***compliment_more***: integer, number of more compliments received by the user;
- ***compliment_profile***: integer, number of profile compliments received by the user;
- ***compliment_cute***: integer, number of cute compliments received by the user;
- ***compliment_list***: integer, number of list compliments received by the user;
- ***compliment_note***: integer, number of note compliments received by the user;
- ***compliment_plain***: integer, number of plain compliments received by the user;
- ***compliment_cool***: integer, number of cool compliments received by the user;
- ***compliment_funny***: integer, number of funny compliments received by the user;
- ***compliment_writer***: integer, number of writer compliments received by the user;

- ***compliment_photos***: integer, number of photos compliments received by the user.

This was also imported as a dataframe and, as in the first task, rows were deleted to accomplish tasks faster.

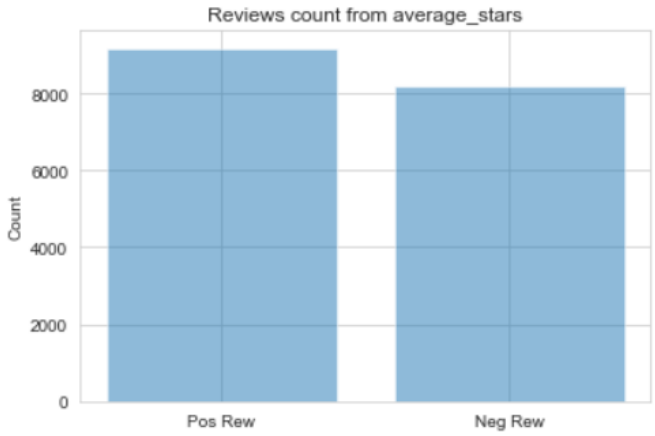


Figure 2: Reviews count from *average_stars* field

2.3.1 Algorithm utils

We created a separate Python file dedicated to clustering algorithms *algorithm.py*. This was necessary to avoid code duplication in other tasks, in this case, the business task. The three main functions used are:

- ***score_plot_and_get_best***: used to get the best parameters and get clustered dataframe;
- ***add_new_column***: used to add assigned cluster to each row of original dataframe;
- ***plot_clusters***: used to plot every possible combinations of all features.

2.3.2 Subdivision of the task

The original task was divided into two subtasks:

- Grouping users according to their satisfactions (preferences);
- Grouping of users based on the behavior on the platform.

In the first, the dataframe contains the following columns:

- review_count;
- useful;
- funny;
- cool;
- average_stars.

While for the second the fields considered are:

- compliment_list;
- compliment_cool;
- compliment_writer;
- compliment_funny;
- compliment_hot;
- compliment_cute;
- compliment_more;
- compliment_note;
- compliment_photos;
- compliment_plain;
- compliment_profile.

2.3.3 K-Means

The first clustering algorithm used for the tasks is K-Means . The K-Means algorithm clusters data by trying to separate samples into n groups of equal variance, minimizing a criterion known as **inertia**. This algorithm requires the number of clusters to be specified. It scales well to a large number of samples and has been used across a large range of application areas in many fields.

n_clusters	silhouette_score	calinski_harabasz_score	davies_bouldin_score
2	0.861961	9176.354559	0.740320
3	0.858419	11504.096866	0.484570
4	0.852039	13062.960067	0.367715

Best n_clusters: 4

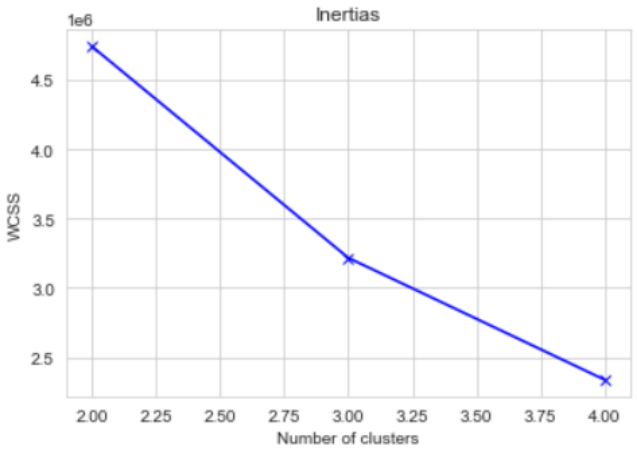


Figure 3: K-Means metrics and best number of clusters for behavior

2.3.4 DBSCAN

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to K-Means which assumes that clusters are convex shaped. There are two parameters to the algorithm, ***min_samples*** and ***eps***, which define the density of the algorithm. Higher *min_samples* or lower *eps* indicate higher density necessary to form a cluster.

min_samples parameter

According to the best parameter evaluations, there is no automatic way to

n_clusters	silhouette_score	calinski_harabasz_score	davies_bouldin_score
2	0.993658	7368.354884	0.004790
3	0.885431	8765.786849	0.787136
4	0.915587	10367.084776	0.601031

Best n_clusters: 2

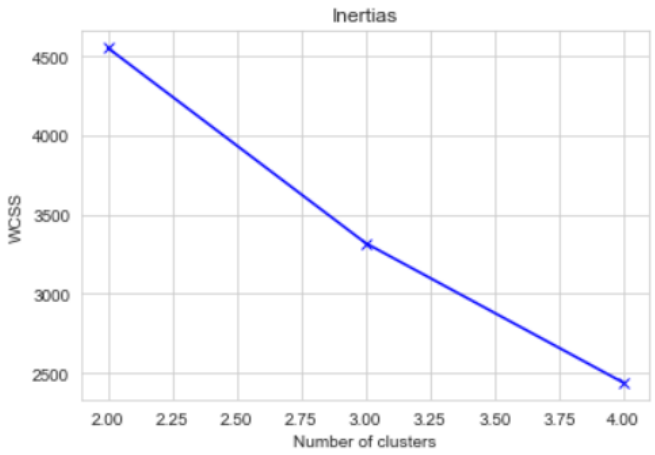


Figure 4: K-Means metrics and best number of clusters for satisfaction

determine the *min_samples* value for DBSCAN. This parameter should be set using domain knowledge and familiarity with the data set. The main rules are:

- The larger the data set, the larger the value of *min_samples* should be;
- Generally, *min_samples* should be greater than or equal to the dimensionality of the data set:
  - for 2-dimensional data should be used default value of

$$min\_samples = 4$$

– for more than 2 dimensions, choose:

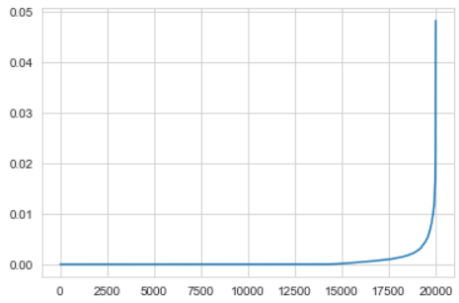
$$min\_samples \ = \ 2 \ * \ dim$$

So, based on our dataset we set *min_samples* to 10 because we have almost 5 features.

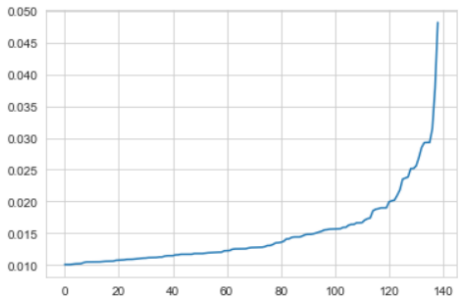
eps **parameter**

The *eps* parameter, however, can be automatically determined by calculating the average distance between each point and its k nearest neighbors, where k is equal to the *min_samples* value you selected (10 in our case). The average k-distances are then plotted in ascending order on a k-distance graph. You'll find the optimal value for *eps* at the point of maximum curvature (where the graph has the greatest slope). To do this we created 3 functions:

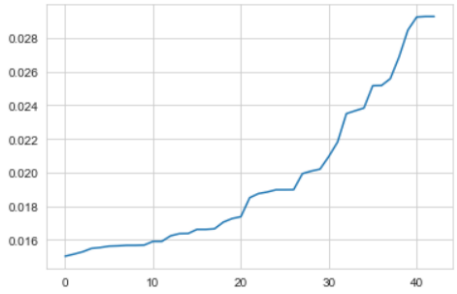
- ***get_distances_and_plot***: used to calculate *NearestNeighbors* and plot them in ascending order;
- ***filter_distances_and_plot***: remove noisy data;
- ***remove_duplicates***: remove duplicated data.



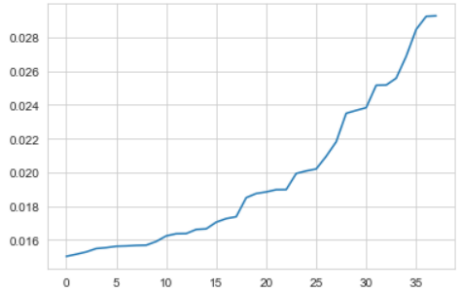
(a) *get_distances_and_plot*



(b) *filter_distances_and_plot*



(c) *filter_distances_and_plot*



(d) *remove_duplicates*

Figure 5: Calculate best *eps*

2.4 Searching for the solution - Business Task

The last task is to automatically group rooms based on similarity criteria given a certain location. The *business.json*, *checkin.json* and *tip.json* datasets from Yelp Open Dataset were used to solve it. The business one contains the following fields:

- ***business_id***: string, 22 character unique string business id;
- ***name***: string, the business's name;
- ***address***: string, the full address of the business;

- ***city***: string, the city;
- ***state***: string, 2 character state code, if applicable;
- ***postal***: string, the postal code;
- ***latitude***: float, latitude;
- ***longitude***: float, longitude;
- ***stars***: float, star rating, rounded to half-stars;
- ***review_count***: integer, number of reviews;
- ***is_open***: integer, 0 or 1 for closed or open, respectively;
- ***attributes***: object, business attributes to values. note: some attribute values might be objects;
- ***categories***: an array of strings of business categories;
- ***hours***: an object of key day to value hours, hours are using a 24hr clock.

Attributes is a complex object (a nested *JSON* in this case), because of this we have got all nested elements and brought them to the main level.
While *checkin* contains:

- ***business_id***: string, 22 character business id, maps to business in *business.json*;
- ***date***: string which is a comma-separated list of timestamps for each checkin, each with format YYYY-MM-DD HH:MM:SS.

Finally, *tip.json* contains:

- ***text***: string, text of the tip;
- ***date***: string, when the tip was written, formatted like YYYY-MM-DD;
- ***compliment_count***: integer, how many compliments it has;
- ***business_id***: string, 22 character business id, maps to business in *business.json*;
- ***user_id***: string, 22 character unique user id, maps to the user in *user.json*.

Before using Machine Learning algorithms, a lengthy preprocessing phase of the dataset was necessary. This was done within the *processing_business_dataset.ipynb* file and each part will be explained in the following sections.

2.4.1 Expand features

We find some items that could be themselves complex objects, they are:

- ***hours***
- ***attributes***
- ***BusinessParking***
- ***GoodForMeal***
- ***Ambience***
- ***Music***
- ***BestNights***
- ***DietaryRestrictions***

For each, we get all items inside and brought to the main level. In this way, we'll be able to use it as a feature.

```
def expand(row, dict):
    for key, values in dict.items():
        row[key] = values
    return row
def expand_row(row, col_name):
    try:
        return expand(row, ast.literal_eval(row[col_name]))
    except:
        return row
```

Listing 12: *expand* function

2.4.2 Drop non-essential features

We identified some features that were not necessary for the purpose (e.g. *NoiseLevel*, *RestaurantsAttire*) and some others not well formatted (e.g. *WiFi*, *Weekdays*). All of these have been dropped from the dataframe.

2.4.3 Merge datasets

To add the *compliment_count* and *date* features to the final dataset, we used the *pd.merge* function to merge the three datasets, mentioned in section 2.4, via the *business_id* column. In doing so, we obtained the final dataset even though it is not yet usable because it contains raw data. In the next sections, you can see how the dataset was modified to make it usable by the various algorithms.

2.4.4 Dataset processing

We processed the dataset, to make it compatible with K-Means and DBSCAN algorithms, in this way:

- Each *Nan* and *None* value has been converted to 0.0;
- Each *True* value has been converted to 1.0;
- Each *False* value has been converted to 0.0;
- Each integer value (e.g. 1) has been converted to correspondent float value (e.g. 1.0).

After this step, we exported the dataset to use it in the *business_task.ipynb* file where the two Machine Learning algorithms will be used.

2.4.5 Filter dataset by city

As required by the outline, the grouping of premises must be done in a certain location. To do this, the function *filter_by_city(city, df)* was created, which returns a dataframe containing the rows of the city passed as a parameter. For testing purposes, we entered the city of *"Tucson"*.

2.4.6 Drop non-number features

K-Means algorithm wants only numerical features, so we dropped string features as:

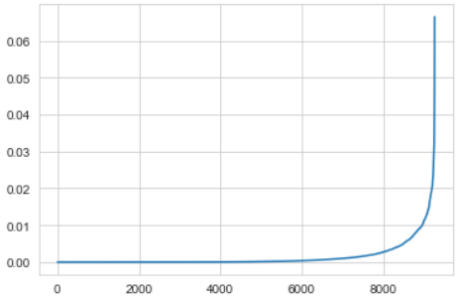
- ***business_id***
- ***address***
- ***categories***

- ***city***
- ***latitude***
- ***longitude***
- ***name***
- ***state***
- ***postal_code***

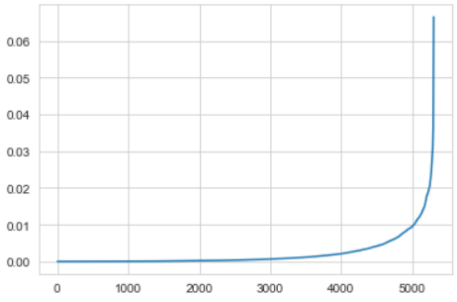
These features will be retrieved after clustering.

2.4.7 Clustering algorithm

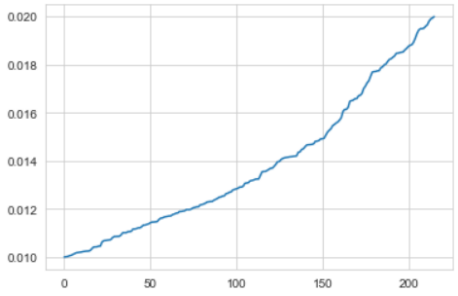
As with the second task, the K-Means algorithm, explained in section 2.3.3, and the DBSCAN algorithm, explained in section 2.3.4, were applied to solve the task.



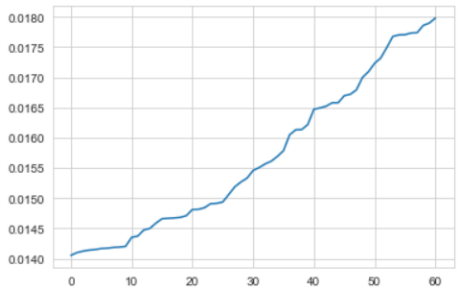
(a) *get_distances_and_plot*



(b) *remove_duplicates*



(c) *filter_distances_and_plot*



(d) *filter_distances_and_plot*

Figure 6: Calculate best *eps*

2.5 Justification of choice

This section will describe how some of the choices present within the project were made.

2.5.1 Text preprocessing and Supervised Learning

For the text preprocessing part of the first task, we chose to use Python's *NLTK* library for two reasons:

- It has many pre-trained models and corpora that help us analyze things very easily;
- It is the best-known and most complete NLP library, with many extensions.

Instead, in terms of Machine Learning algorithms, we decided to compare some from different "families", such as:

- **Bayesian algorithms**: which is a family of probabilistic classifiers;
- **DecisionTree algorithms**: which is a simple way to visualize models in tree form;
- **Support Vector Machine algorithms**: which is another group of algorithms for classification and regression. These are good algorithms because it gives fairly accurate results with minimal computing power.

2.5.2 Unsupervised Learning

As for the second and third tasks, two algorithms of unsupervised learning were used:

- **K-Means**, chosen because is the most commonly used unsupervised learning algorithm. It is said to be the simplest because it does not have many computations.
- **DBSCAN**, better than other cluster algorithms because it does not require a pre-set number of clusters. We used it with the eps parameter to calculate the best one.

3 Experimental Results

3.1 Instructions for demonstration

As can be seen in Figure 7, the project has been divided into 4 main parts:

- ***tasks***: where the three main tasks of the project have been placed;
- ***utils***: where there are files necessary for the operation of the tasks that are reused to avoid duplication of code;
- ***models***: where the models necessary for the operation of the web application have been exported;
- ***datasets***: where the datasets used and the processed ones are exported.

As explained in the section 3.1.3, in addition, an HTTP server and web project were also developed for demonstration of the review prediction feature and visualization of places clustering.

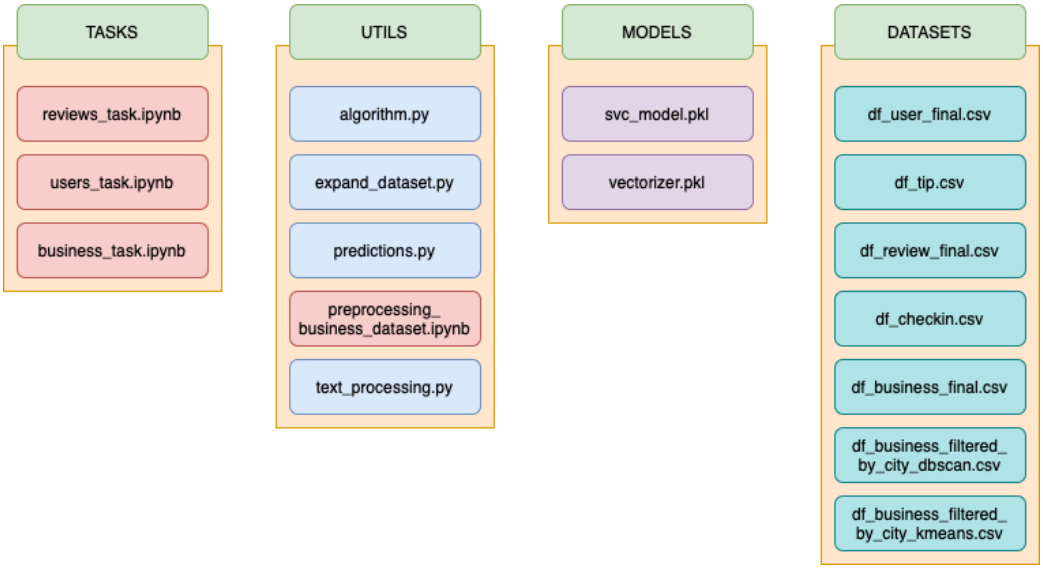


Figure 7: Project Structure

3.1.1 Review Task

For the review task, it is necessary to run the related *.ipynb* file, paying attention to the presence of the *df_reviews_final.csv* dataset and the *text_processing.py* file in the *utils* folder. At this point the *vectorizer.pkl* and *svc_model.pkl* models will be exported, which will then be used by the server to provide via HTTP the review prediction.

To check how it works you need to start the web project and go to the */predict_review.html* page where you will be able to enter the review and receive the prediction.

3.1.2 Users Task

For the users task, it is necessary to run the related *.ipynb* file making sure that the *df_users_final.csv* dataset and the *algorithm.py* file are present in the

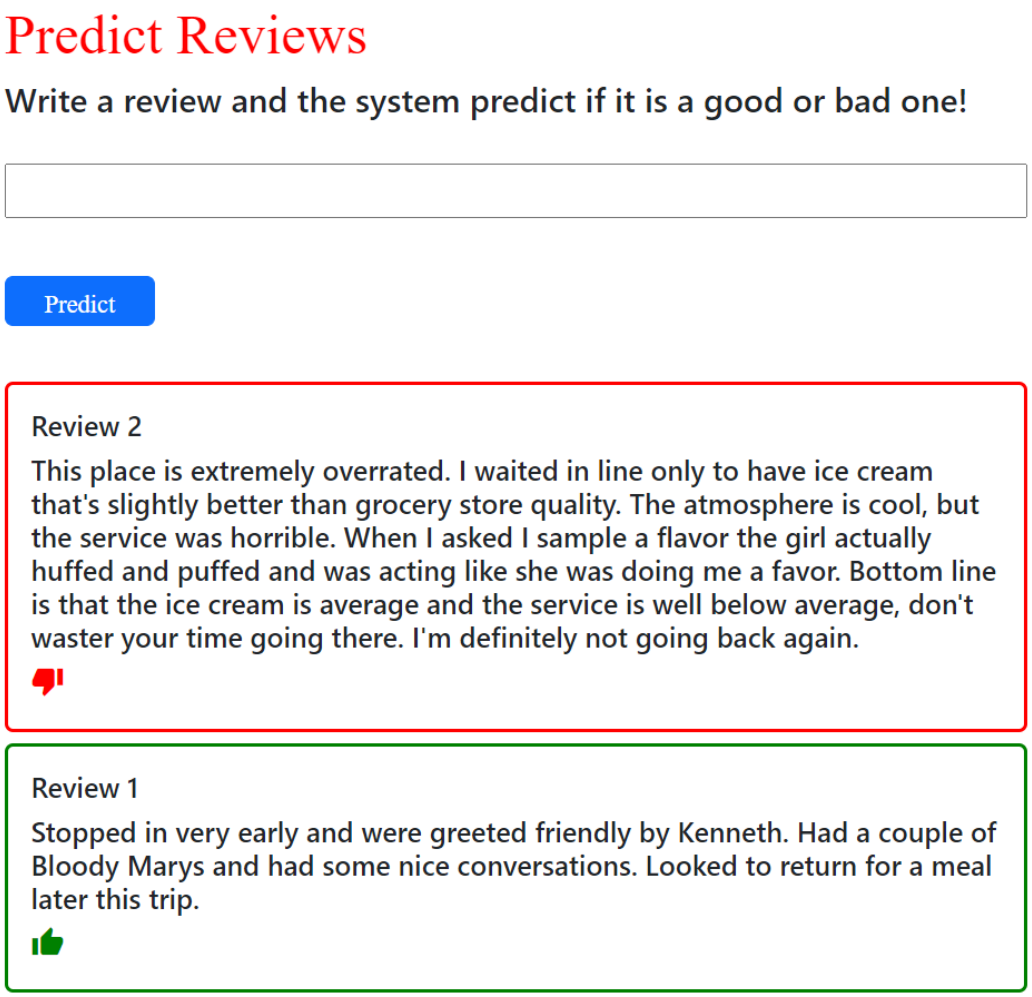


Figure 8: HTML page for review predictions

utils folder. Immediately after the execution of each clustering algorithm, you can view the graphs for each possible combination of features.

3.1.3 Business Task

Before you can run the business task, you need to run the file related to preprocessing the *df_business.csv* dataset, which will export the final dataset

df_business_final.csv. At this point, you can run the *.ipynb* file corresponding to the business task which will export two datasets:

- *df_business_filtered_by_city_dbscan.csv*
- *df_business_filtered_by_city_kmeans.csv*

which will be needed to visualize the two algorithms on a map and compare them. Now you can start the server and open the web project on the main page. Here two instances of the same map with the two different clustering algorithms will be displayed.

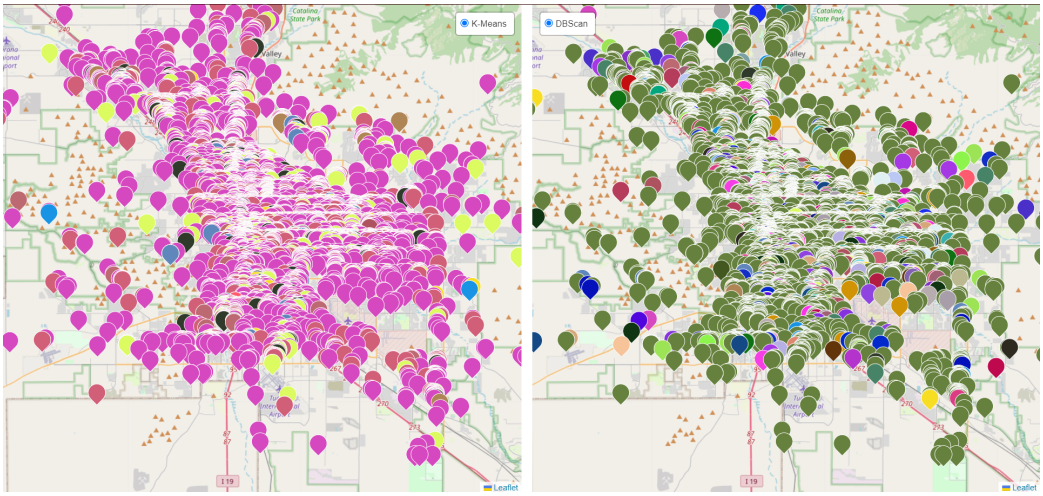


Figure 9: HTML page for K-Means vs DBSCAN clustering

3.2 List of technologies used for experiments

3.2.1 IDEs

PyCharm Initially, we tried to develop the project on *Google Colab* to be able to have direct access to the code at the same time. After some testing,

however, we realized that working with a native IDE like *PyCharm* and using ***Git*** for versioning was more efficient. We, therefore, used *PyCharm 2022.1.3 (Professional Edition) (Build #PY-221.5921.27)* licensed from the University of Bologna. The *"Code With Me"* feature provided by the IDE was also used, particularly at some junctures remotely, for better synchronization.

WebStorm To create the web platform to visualize clustering and predict reviews, we used *WebStorm IDE 2022.1.3 (Build #WS-221.5921.27).*

3.2.2 Language and external libraries

As a language, we used *Python* because it provides several libraries suitable for Machine Learning. In particular, the version used is the latest version available, namely *3.10.5*. The following were used as external libraries:

- ***Pandas*** *1.4.3* - most common, open source data analysis and manipulation tool;z
- ***NumPy*** *1.23.1* - used for mathematical support for some processes;
- ***Matplotlib*** *3.5.2* - library used to create plots and interactive visualizations for our data. Used to create all project plots;
- ***Scikit-learn*** *1.1.1* - simple and efficient tools for predictive data analysis. We used it for all Machine Learning tasks;
- ***Seaborn*** *0.11.2* - data visualization library based on Matplotlib. We used it to create the confusion matrix;

- ***Pickle*** *0.7.5* - used for serializing a Python object structure. In our case, we used it to export the algorithm model and vectorizer (Section 2.2.5)
- ***Natural Language Toolkit (NLTK)*** *3.7* - used to work with human language data. We used it to process review text (Section 2.2.1);
- ***Flask*** *2.1.3* - used to build HTTP server.

3.3 Description of the method for measuring performance

3.3.1 Supervised Learning metrics

Regarding the review task using Supervised Learning algorithms, the most common metrics were measured. As can be seen in Section 3.5, where the results obtained were presented, the classification report was generated for each algorithm, which has the measurements of these metrics:

- **Precision**: the ratio of relevant instances among the retrieved instance

$$precision = \frac{true\_positives}{true\_positives + false\_positives}$$

- **Recall**: the ratio of relevant instances that were retrieved

$$recall = \frac{true\_positives}{true\_positives + false\_negatives}$$

- **F1-Score**: the weighted average of precision and recall

$$f1\_score = 2 \text{ x } \frac{precision \text{ x } recall}{precision + recall}$$

At the end of the run, all the accuracy of the various algorithms were then compared by taking the one with the best performance.

3.3.2 Unsupervised Learning

For users task and business task, which use Unsupervised Learning algorithms, three metrics were used to be able to identify the best number of clusters in the K-Means and DBSCAN algorithms:

- **Silhouette Coefficient**

  The *Silhouette Coefficient* is defined for each sample and is composed of two scores:

  - The mean distance between a sample and all other points in the same class. This score measure the closeness of points in the same cluster.
  - The mean distance between a sample and all other points in the next nearest cluster. This score measures the distance of points of different clusters.

  A higher *Silhouette Coefficient* score relates to a model with better-defined clusters. The score is bounded between *-1* for incorrect clustering and *+1* for highly dense clustering. Scores around zero indicate overlapping clusters. The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

- **Calinski-Harabasz**

  The Calinski-Harabasz index is the ratio of the sum of between-clusters

dispersion and of inter-cluster dispersion for all clusters, the higher the score, the better the performance.

- **Davies-Bouldin Index**

  This index signifies the average similarity between clusters, where the similarity is a measure that compares the distance between clusters with the size of the clusters themselves. A lower Davies-Bouldin index relates to a model with better separation between the clusters.

3.4 Best configuration results

As explained in section 2.2.3, in supervised learning task (reviews task) several algorithms were compared. Among them, the one with the best performance is SVM. The final accuracy of the SVM algorithm is 85%.
To achieve this, different values were compared for each parameter to be able to identify the best ones:

```
svc_params = {'C': [0.001, 0.01, 0.1, 1.0],
              'kernel': ['linear', 'rbf', 'sigmoid', 'poly'],
              'decision_function_shape': ['ovo', 'ovr']}
get_best_parameters(SVC(), svc_params)
```

Listing 13: SVC parameters

The best parameters were found to be:

- *C=1.0*;
- *kernel='linear'*;
- *decision_function_shape='ovo'*.

After calculating accuracy, the confusion matrix was created, as shown in Figure 10, and the classification report, containing key metrics for comparing the algorithms, was created (Figure 11).

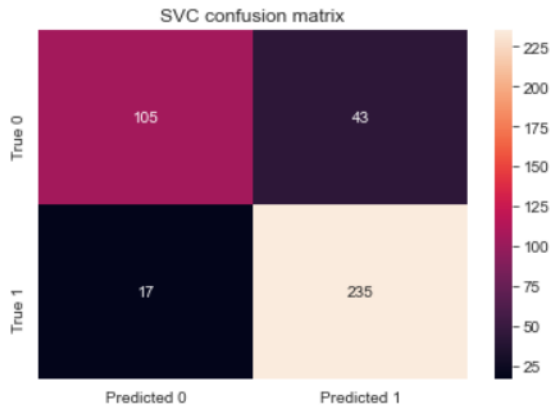


Figure 10: Confusion Matrix SVM

As can be seen, it is difficult for the model to predict a negative review if the review is positive. In fact, the largest values are on the diagonal, and this means that most of the time the model correctly classifies reviews.

3.5 Ablation study: comparison of different configurations

In this section, all the algorithms used and the different configurations of each will be compared for each task.

3.5.1 Reviews task

All algorithms have been explained in the section 2.2.4, here only the results will be indicated.

	precision	recall	f1-score	support
0	0.860656	0.709459	0.777778	148.00
1	0.845324	0.932540	0.886792	252.00
accuracy	0.850000	0.850000	0.850000	0.85
macro avg	0.852990	0.821000	0.832285	400.00
weighted avg	0.850997	0.850000	0.846457	400.00

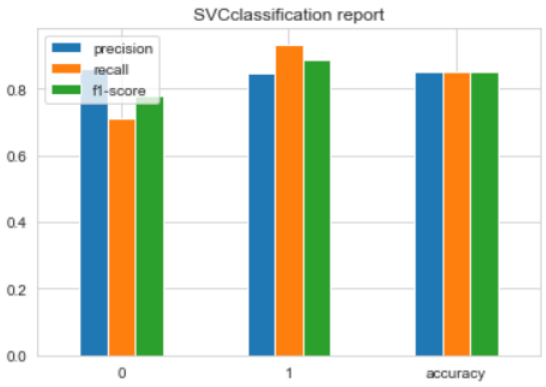


Figure 11: Classification Report SVM

The graphs for accuracy, confusion matrix and classification report of each algorithm are shown below.

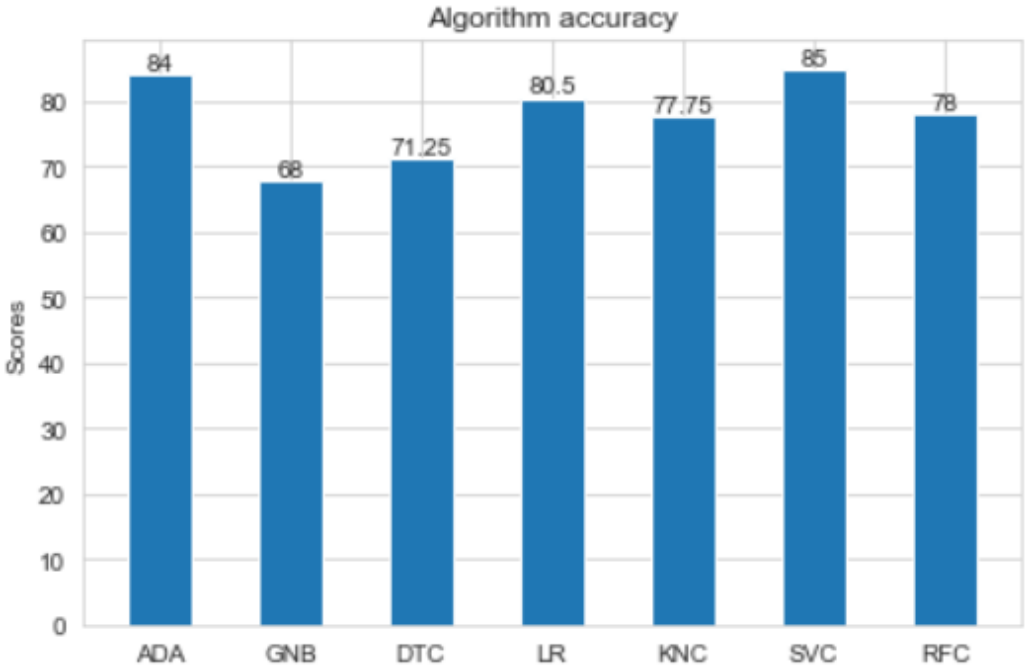


Figure 12: Accuracy of all algorithms

AdaBoostClassifier

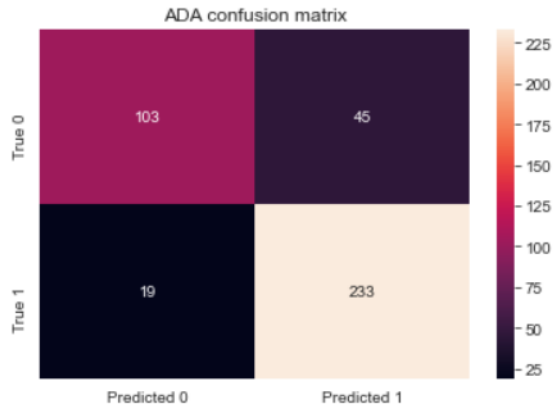


Figure 13: Confusion Matrix AdaBoostClassifier

	precision	recall	f1-score	support
0	0.844262	0.695946	0.762963	148.00
1	0.838129	0.924603	0.879245	252.00
accuracy	0.840000	0.840000	0.840000	0.84
macro avg	0.841196	0.810275	0.821104	400.00
weighted avg	0.840399	0.840000	0.836221	400.00

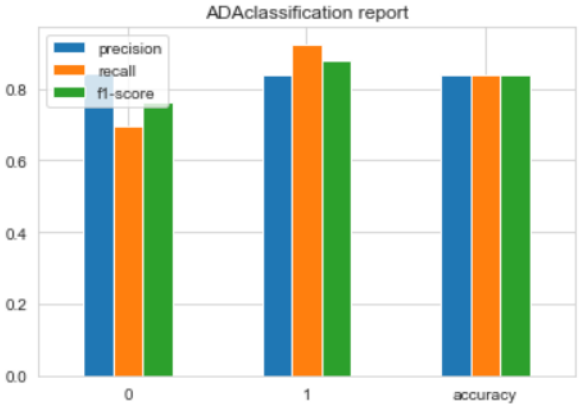


Figure 14: Classification Report AdaBoostClassifier

GaussianNB

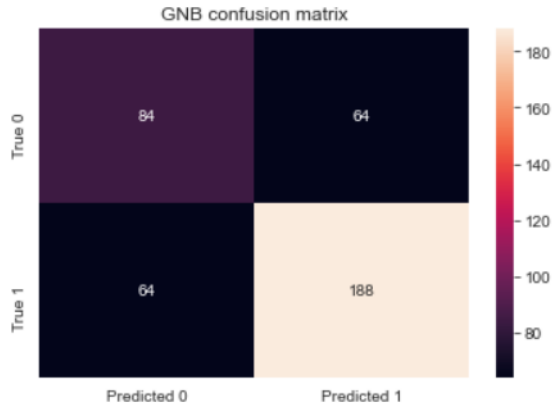


Figure 15: Confusion Matrix GaussianNB

	precision	recall	f1-score	support
0	0.567568	0.567568	0.567568	148.00
1	0.746032	0.746032	0.746032	252.00
accuracy	0.680000	0.680000	0.680000	0.68
macro avg	0.656800	0.656800	0.656800	400.00
weighted avg	0.680000	0.680000	0.680000	400.00

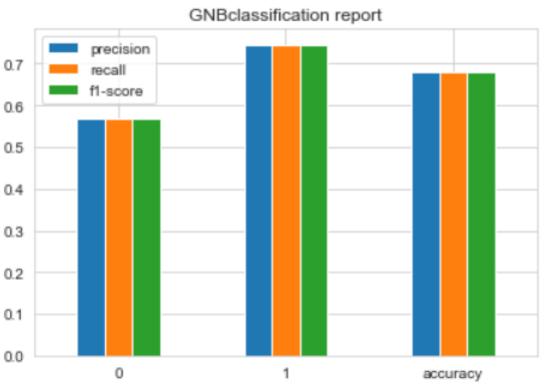


Figure 16: Classification Report GaussianNB

DecisionTreeClassifier

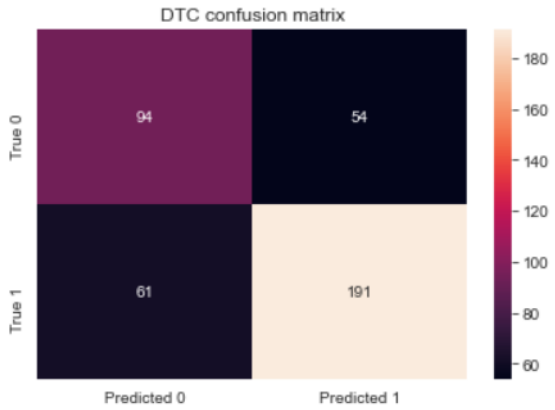


Figure 17: Confusion Matrix DecisionTreeClassifier

	precision	recall	f1-score	support
0	0.606452	0.635135	0.620462	148.0000
1	0.779592	0.757937	0.768612	252.0000
accuracy	0.712500	0.712500	0.712500	0.7125
macro avg	0.693022	0.696536	0.694537	400.0000
weighted avg	0.715530	0.712500	0.713796	400.0000

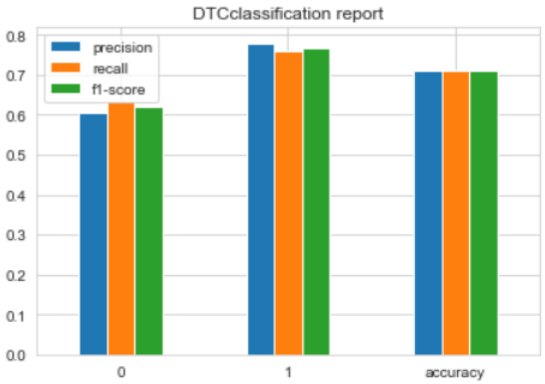


Figure 18: Classification Report DecisionTreeClassifier

LogisticRegression

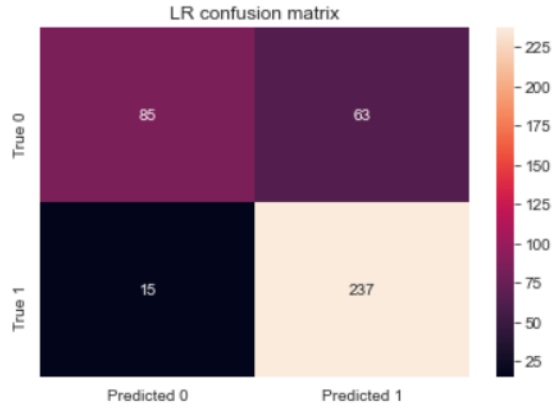


Figure 19: Confusion Matrix LogisticRegression

	precision	recall	f1-score	support
0	0.8500	0.574324	0.685484	148.000
1	0.7900	0.940476	0.858696	252.000
accuracy	0.8050	0.805000	0.805000	0.805
macro avg	0.8200	0.757400	0.772090	400.000
weighted avg	0.8122	0.805000	0.794607	400.000

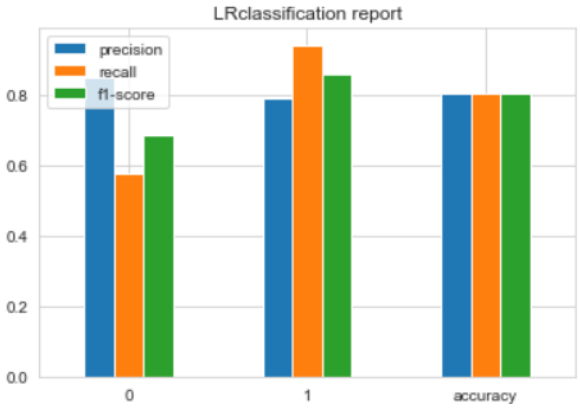


Figure 20: Classification Report LogisticRegression

KNeighborsClassifier

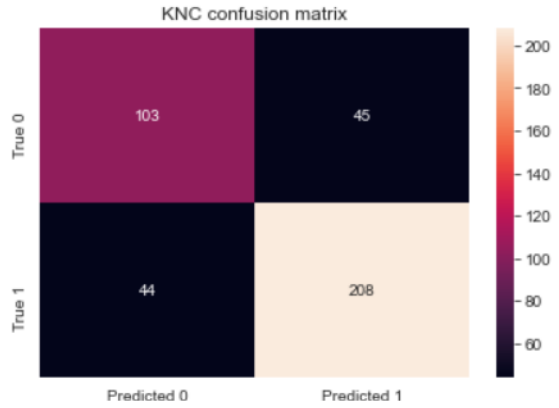


Figure 21: Confusion Matrix KNeighborsClassifier

	precision	recall	f1-score	support
0	0.700680	0.695946	0.698305	148.0000
1	0.822134	0.825397	0.823762	252.0000
accuracy	0.777500	0.777500	0.777500	0.7775
macro avg	0.761407	0.760671	0.761034	400.0000
weighted avg	0.777196	0.777500	0.777343	400.0000

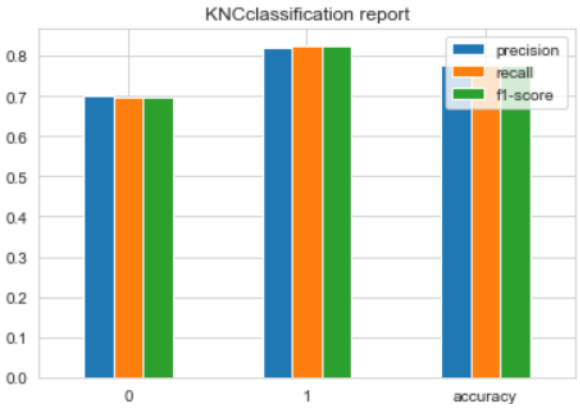


Figure 22: Classification Report KNeighborsClassifier

RandomForestClassifier

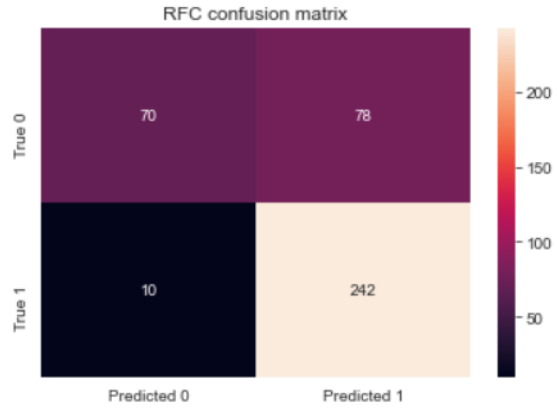


Figure 23: Confusion Matrix RandomForestClassifier

	precision	recall	f1-score	support
0	0.875000	0.472973	0.614035	148.00
1	0.756250	0.960317	0.846154	252.00
accuracy	0.780000	0.780000	0.780000	0.78
macro avg	0.815625	0.716645	0.730094	400.00
weighted avg	0.800187	0.780000	0.760270	400.00

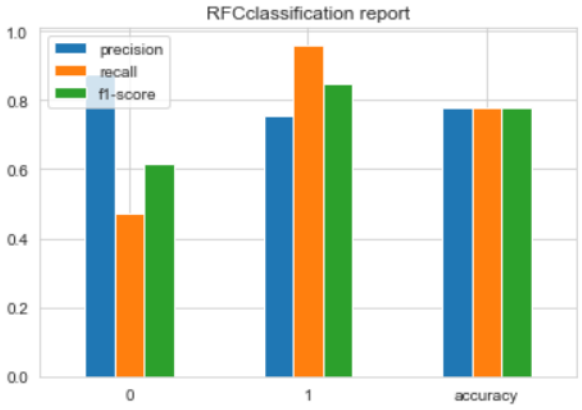


Figure 24: Classification Report RandomForestClassifier

As you can see from the figures above, almost all of the confusion matrix algorithms have the largest values on the diagonal, which means that most of the time the reviews are predicted correctly. Only for the RandomForest algorithm does this not happen, in fact, this one is more likely to predict a positive review when it is actually negative. This can also be verified by the report of this algorithm, where the F1-score for *label* 0 is about 0.6 while for *label* 1 it is higher than 0.8.

Support Vector Machine

This turned out to be the best algorithm, in the previous section there are all the details (section 3.4).

3.5.2 Users task

Two algorithms, K-Means and DBSCAN, were used in this task. In the area of features related to user behavior, we take as an example two graphs related to the comparison of the features *review_count*, *average_stars*, *useful* and *funny*. In particular, for the K-Means algorithm 4 was found as the best number of clusters, while for DBSCAN 162. Comparing graphically K-Means (Figure 25) and DBSCAN (Figure 26), it can be seen that K-Means does a better job by being able to separate users mainly into 2 clusters.

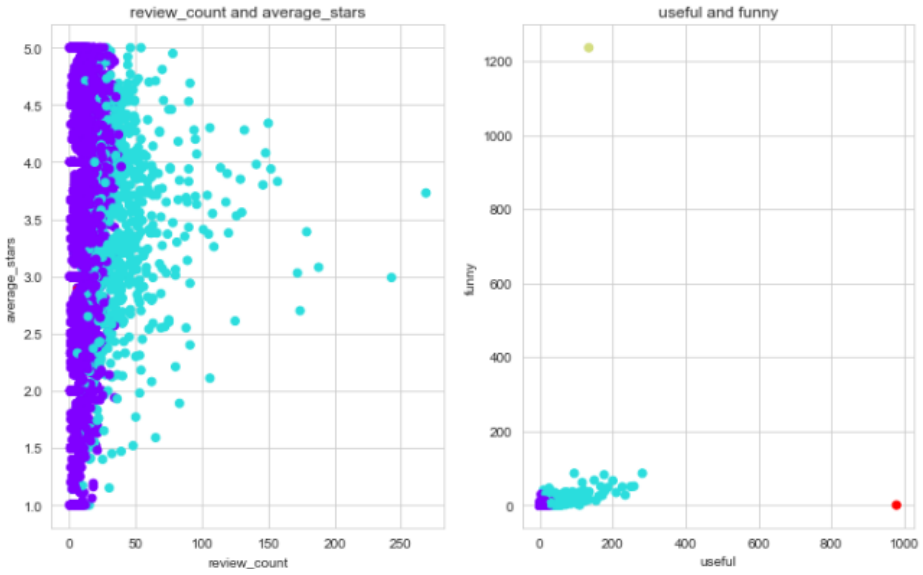


Figure 25: K-Means clusterization

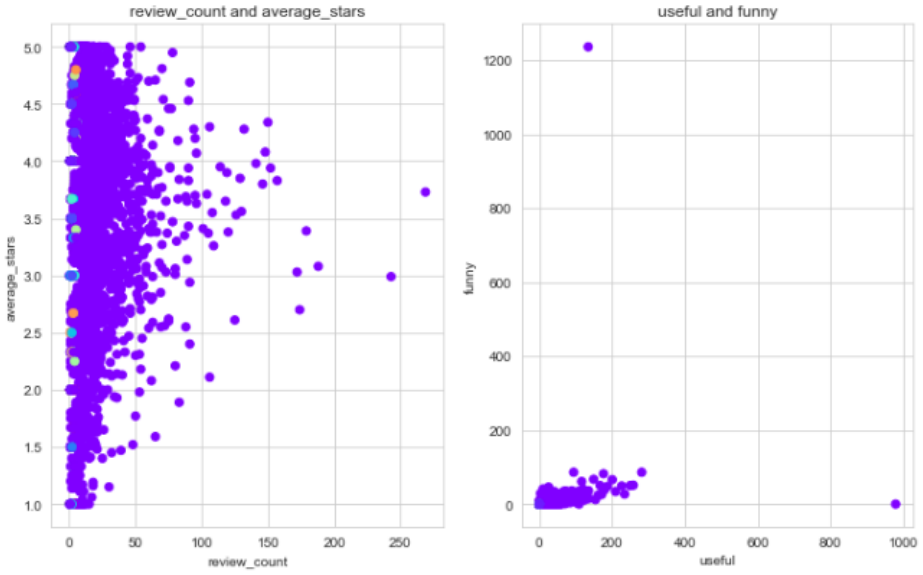


Figure 26: DBSCAN clusterization

3.5.3 Business task

K-Means and DBSCAN were also used for this task. The city of *Tucson* was used for testing, and concerning K-Means 17 clusters were identified as the best number of clusters (Figure 27 shows the distribution in the various clusters), and for DBSCAN 71 (Figure 28 shows the distribution in the various clusters). Having location data available, we directly compared the geographic distribution of the various clusters by placing two maps with the same colored markers side by side to reflect cluster membership (Figure 9).

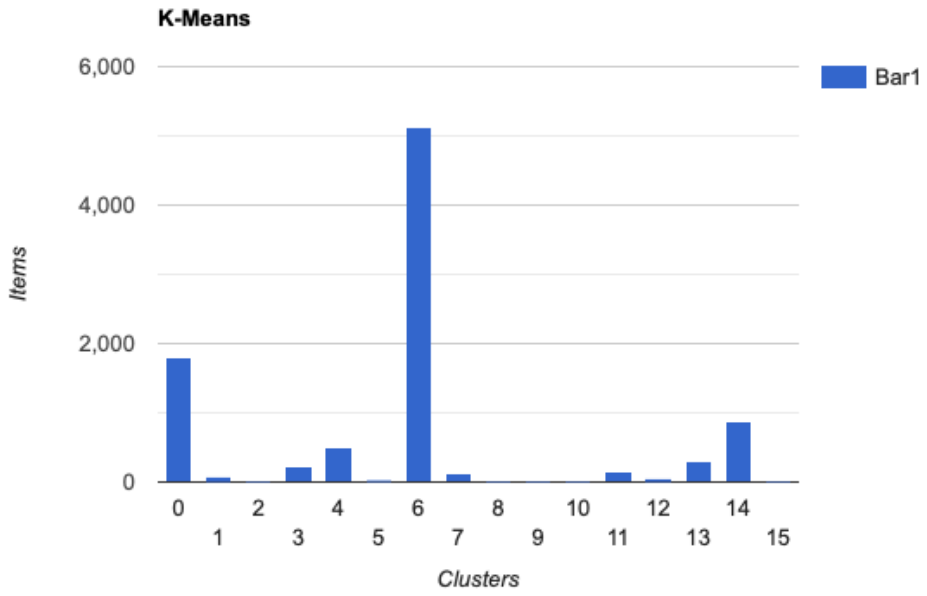


Figure 27: Business task K-Means cluster distribution

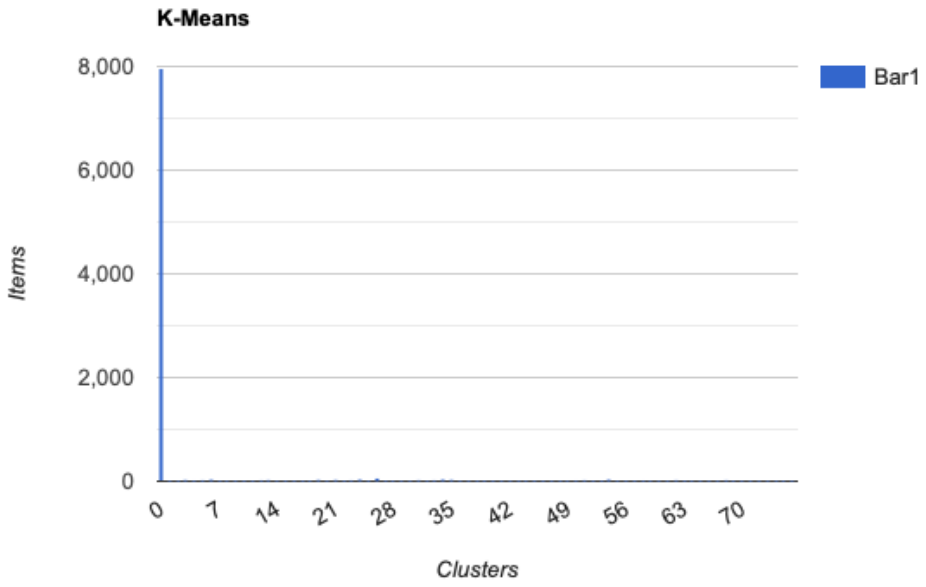


Figure 28: Business task DBSCAN cluster distribution

4 Discussion and Conclusions

In the following sections, we will discuss the final consideration of this project.

4.1 Discussion of the results obtained and limitations of the method

As explained in the section 3.4, after comparing 7 different algorithms, the best was found to be SVC with an accuracy of 85%. However, it must be taken into account that the test for the first task was performed on 2000 rows to speed up the process on our computers. Using the whole dataset with 7,000,000 is likely to come out with different results, in which case it is only

necessary to export the model of the algorithm with the best performance. Instead, for user task the test was performed on 20000 rows and for business task the test was performed on 150000. On the other hand, concerning K-Means and DBSCAN, for the users task it is possible to view the comparative graphs in the section 3.5.2, while for the business task it is possible to view through the web page with the geographic comparison of the markers related to the premises. Again, using the full datasets might yield different results.

4.2 Future work: how you intend to carry on the work

Because the Yelp Open Dataset is large, it is possible, with its data and with different artificial intelligence techniques, to perform different tasks as we have seen with the other solutions in the literature (Section 1.3). Regarding machine learning specifically, among other ideas we have:

- Using the *df_checkin* dataset, which provides us with data on the presence of people inside the premises, it is possible, for example, to estimate the most frequented places or times;
- Using a combination of multiple datasets, the liking of different locales can be detected;
- Using the photo dataset can be done tasks of image classification and image recognition.

References

[1] Nabiha Asghar. Yelp dataset challenge: Review rating prediction. 2016.

[2] Bhanu Prakash Reddy Guda, Mashrin Srivastava, and Deep Karkhanis. Sentiment analysis: Predicting yelp scores. *arXiv preprint arXiv:2201.07999*, 2022.

[3] Sindhu Hegde, Supriya Satyappanavar, and Shankar Setty. Restaurant setup business analysis using yelp dataset. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2342–2348. IEEE, 2017.

[4] Xiaopeng Lu, Jiaming Qu, Yongxing Jiang, and Yanbing Zhao. Should i invest it? predicting future success of yelp restaurants. In *Proceedings of the Practice and Experience on Advanced Research Computing*, pages 1–6. 2018.

[5] Li Yanrong. Prediction of useful reviews on yelp dataset.