Data Mining Exam

Stefano Baroni and Matteo Mormile, **IT University of Copenhagen**
https://github.itu.dk/matmo/DataMining2023.git

1 Introduction

The scope of the project is to conduct an in-depth analysis of incident data in New York City, using advanced data mining techniques to answer critical questions.
The primary goals are:

- **Predicting Missing Motivations**
  Developing a Random Forest classification model to predict unspecified motivations associated with collisions, taking advantage of the collective decision-making capabilities of multiple decision trees for improved accuracy.

- **Uncovering Collisions Patterns through Clustering**
  Using advanced clustering algorithms to uncover patterns linking collisions based on motivations, as well as categorizing collisions with shared characteristics to reveal hidden structures and associations.

2 Data Description

For our research we needed a dataset that contained a fairly well-stocked history of car-to-car crashes. Most cities, thankfully, allow free access to this information data, and we chose, from the available datasets, one that represents crashes that occurred in New York City between 2016 and 2023. [1]
The dataset, which was originally structured as a comma separated value file, contains information about each accident for each row, such as the date and time of the accident, the borough, the zip code, the latitude and longitude, the location, which is nothing more than the combination of the previous two, the street name, the nearest cross name, the number of people injured and killed, categorized by type as pedestrian, cyclist, or motorist and, for each vehicle involved, the reasons for the collision and the car's models.
In order to use and manipulate the dataset in our phyton code, we decided to use the pandas library, which allows us to directly load our csv file inside a dataframe. A Pandas DataFrame is a two-dimensional data structure that organizes data into rows and columns, enabling powerful operations for cleaning, manipulation, and analysis.

3 Data Preparation

However, before we can view our data and analyze it to be able to extract important information, we must make some changes in order to make the best use of it.

The first thing we did was reduce the number of columns in which the accident's street name was specified. We noticed that the street name was sometimes reported in the `ON STREET NAME` column and sometimes in the `OFF STREET NAME` one, so we decided to merge these values in the first one in order to remove the second one. We also removed the `LOCATION` column, which was simply a column derived from the `LATITUDE` and `LONGITUDE` columns.

```
replace = lambda row: row['OFF STREET NAME']
 if pd.isna(row['ON STREET NAME']) else row['ON STREET NAME']
df2['ON STREET NAME'] = df2.apply(replace, axis=1)
df2 = df2.drop(columns = ['LOCATION','OFF STREET NAME'])
```

Listing 1: Merge of off street name inside on street name and drop location

Proceeding with our research, we discovered that the values of latitude and longitude, zip code, and borough were indicated for some streets but not for others. As a consequence, we decided to create a dictionary with the streets name as the key and the terms mentioned above as the values. We were able to reconstruct 497,442 rows by first creating the dictionary and then searching for rows with missing values in the dataframe.
Despite the reconstruction, we noticed that some zip codes were still made of only spaces. By using the strip function to remove the spaces and then removing the rows where the zip code appeared empty, we were able to remove 42 malformed rows.
After achieving this result, we came across rows with the opposite problem: there were data on latitude and longitude but not on the street name where the incident occurred. Since latitude and longitude are measured in degrees, repeating the process in reverse with the dictionary would introduce a margin of deviation. To avoid this, we decided to use an OpenStreetMap system [2] that allows us to receive information about a specific location via an API. By doing this, we were able to complete the missing street name information for 585 rows.

```
def get_street_name(lat, lon):
  if pd.notna(lat) and pd.notna(lon):
    location = geolocator.reverse((lat, lon), language='en')
    if location and 'address' in location.raw:
      return location.raw['address'].get('road', None)
  return None
```

Listing 2: Function to obtain street name given latitude and longitude

Our dataset also includes information on people who were injured or killed in the accident, categorizing them as pedestrians, bicyclists, or motorists. There are also `NUMBER OF PERSONS INJURED` and `NUMBER OF PERSONS KILLED` columns, which are totals of the previous categories. However, there are errors or missing values present, so we have fixed these numbers by recalculating the sum.
Our final step in data preprocessing was to remove any incidents that had incorrect values, such as latitude and longitude. We constructed a polygon using the major geographic points bordering the city of New York and filtered the values in our dataset so that they respected this boundary. By doing this we were able to remove 4398 outliers.

4 Data Visualization

Before we can derive information from our data and after we have cleaned it, we need to evaluate it to see how the values are distributed within our dataset and whether they can be useful for our research. We used several graphs to represent the various values in our dataset in order to achieve this.
The first thing we did was look at how the number of incidents changed over time. We obtained a graph representing the number of incidents by grouping them by month. We can see how the unusual phenomenon of the covid pandemic resulted in a sudden drop in incidents, followed by an almost periodical behavior of incidents that appears to have decreased compared to previous years. Moreover we can notice how in general the number of incidents tend to drop around the end of every year.

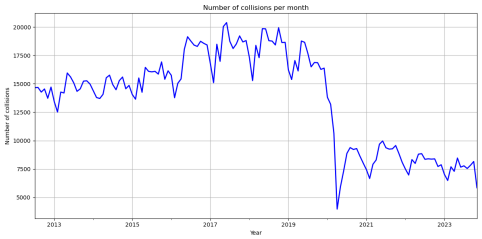

Fig. 1: Number of collisions per month

We also decided to examine how the incidents were distributed across different New York neighborhoods, using a groupby and displaying the results in a histogram, showing how the Queens district appears to be the most dangerous one.

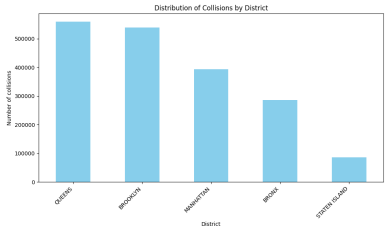

Fig. 2: Number of collisions per neighborhoods

We created a heatmap of New York City to visualize the distribution of accidents and verify that their location referred to streets in order to guarantee that the latitude and longitude data were correct.

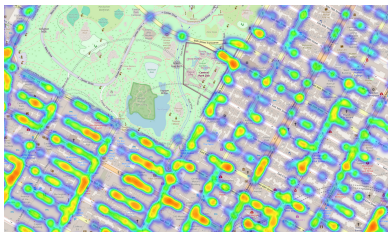

Fig. 3: Heatmap of car collisions

We even showed how the injured or killed people were divided between pedestrians, cyclists, and motorists. We chose a stacked bar chart to represent this data, with each column representing the number of injured or killed individuals for each year. The column is then divided into colored sections that represent the different categories.

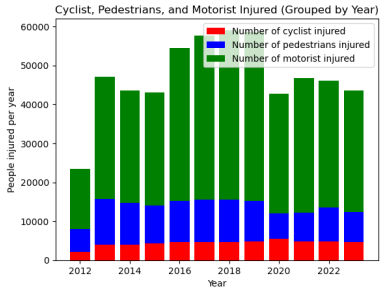

Fig. 4: Distribution of injured people

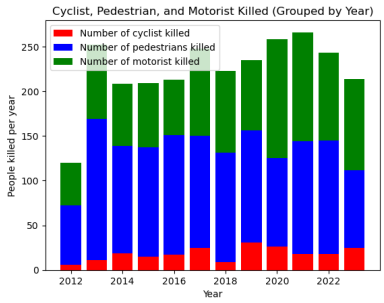

Fig. 5: Distribution of killed people

Finally we wanted to learn about the causes of accidents and how they were distributed.

	Value	Occurrences	Percentage
0	Unspecified	639680	34.399831
1	Driver Inattention/Distraction	376623	20.253514
2	Failure to Yield Right-of-Way	116131	6.245133
3	Following Too Closely	97673	5.252524
4	Backing Unsafely	71043	3.820453
5	Other Vehicular	58061	3.122325
6	Passing or Lane Usage Improper	53311	2.866886
7	Passing Too Closely	48983	2.634140
8	Turning Improperly	43843	2.357729
9	Unsafe Lane Changing	35907	1.930957

Fig. 6: Distribution of accident motivations

The discovery of a large number of unspecified motivations specifically guided us in the early stages of our research.

5 Data Processing and Knowledge Extraction

5.1 Analysis of collision factors

The majority of the factors that contributed to the accident are `unspecified`, as the Figure 6 illustrates. Apart from that, significant numbers of others are sporadic and can be identified from others by slight variations in writing or by similarities. Examples of reasons that could be combined are `Cell Phone (hand-held)` and `Cell Phone (hands-free)`, as is always improper use of the phone in both scenarios. We also discovered meaningless categories like "1" and "80," which fortunately don't come up too often, so we chose to eliminate them.

5.2 Dimentionality Reduction through Word2Vec

From the above observations, we then tried to reduce the number of motivation for collisions by grouping possible definitions using the algorithm Word2Vec. Word2Vec is a widely used algorithm for generating word embeddings, which are vector representations of words. In our case code, we chose the Continuous Bag of Words (CBOW) model, as evidenced by the parameter setting sg=0 during the Word2Vec model initialization:

```
# Train Word2Vec model
model = Word2Vec(sentences, vector_size=100,
                 window=1, min_count=1, workers=4, sg=0)
threshold_similarity = 0.55
```

Listing 3: Use Word2Vec to reduce the number of unique contributing factor

```
Similar motivations to join:
passing too closely : ['following too closely']
reaction to uninvolved vehicle:
     ['reaction to other uninvolved vehicle']
pavement defective : ['tow hitch defective']
driverless/runaway vehicle : ['vehicle damaged']
cell phone(hands-free):['cell phone(hand-held)']
```

In the CBOW model, the algorithm predicts a target word based on its context, utilizing tokenized and lowercased sentences for training. The neural network employed by Word2Vec learns to associate words with their context, generating vector representations that capture semantic relationships.

The code computes the cosine similarity between the vectors of different motivations, using a threshold similarity value of 0.55 to identify pairs of motivations considered similar. The cosine similarity is based on the cosine distance, which is the cosine measure of the angle between the two vectors representing the words.

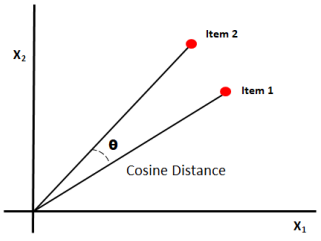

Fig. 7: Cosine distance

The identified similar motivations are then stored in a dictionary for further analysis.

```
if similarity > threshold_similarity:
    # Merge token to original motivation
    mot1_str = ' '.join(motivation1)
    mot2_str = ' '.join(motivation2)
    if mot2_str not in similar_values_dict:
        similar_values_dict[mot1_str].append(mot2_str)
```

Listing 4: Save the similar motivation into a dictionary

5.2.1 Manual Dimentionality Reduction

We also decided to manually reduce the list of possible collision causes by creating a dictionary similar to the one produced by the Word2Vec algorithm by associating similar motivations by hand.

5.3 Result of Dimentionality Reduction

After creating our dictionaries manually and through Word2Vec, we used a function that updates the values in the dataframe with the keys associated with those values in the dictionary.

```
def replace_values(value, dictionary):
    for key, values in dictionary.items():
        if value in values:
            return key
    return value

df['CONTRIBUTING ...'] = df['CONTRIBUTING ...'] \
    .apply(lambda x: replace_values(x, similar_values_dict))

df_man['CONTRIBUTING ...'] = df['CONTRIBUTING ...'] \
    .apply(lambda x: replace_values(x, manual_dictionary))
```

Listing 5: Apply the reduction to our dataframe

By doing this we went from 57 initial unique values to 54 in the case of reduction with Word2Vec and 27 with manual reduction.

5.4 Supervised Classification for Missing Motivation

We chose to use a supervised classification model to address the problem of most unspecified accident causes, as shown in the Figure 6. Among the various algorithms available, including Support Vector Machine and Decision Tree, we chose Random Forest. We opted for the latter since SVM needed excessive resources to analyze the nearly 1.900.000 data points, and Decision Tree was significantly less accurate than Random Forest.

5.4.1 Random Forest Overview

Random Forest is a learning algorithm used for classification and regression tasks that contains several decision trees on various subsets of the given dataset and takes the average of them to improve the predictive accuracy. It is based on the concept of ensemble learning which is a process of combining multiple classifiers to solve a complex problem and improve the performance of the model (see [3] for more).

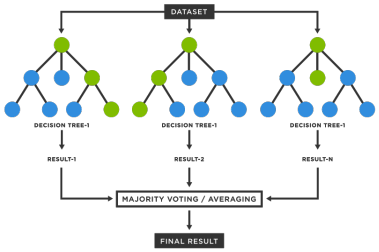

Fig. 8: Random Forest Rappresentation [4]

In our case we will use a Classifier and not a Regressor because the value we need to identify, the motivation for the collision, is a label and its values belong to a finite, discrete set. Instead, a Regressor algorithm is used when the value to be predicted is a continuous numerical value, in our dataset could be, for example, the latitude of the incident.

5.4.2 Apply Random Forest to different DataFrame

Before we used Random Forest to reconstruct the unspecified values of accidents, we conducted an accuracy analysis on the different dataframes that we obtained with manual reduction and Word2Vec. We applied the same function, which contains before the classified some necessary steps for our data, to the different dataframes and analyzed the accuracy to select the best one to use for the effective reconstruction of the data.
In fact, before we can generate the random forest model, we must first select the features, which are the variables or factors that will be used to make predictions or classify data, and ensure that they are in numeric format. The features we will use to build the model are `BOROUGH`, `ZIP CODE`, `ON STREET NAME`, `NUMBER OF PERSONS INJURED`, `NUMBER OF PERSONS KILLED`. As previously stated, we will use a Label Encoder to convert the features that are strings into numeric format. The Label Encoder will assign to each category a unique number sequentially.

```
le = LabelEncoder()
df['BOROUGH'] = le.fit_transform(df['BOROUGH'])
df['ON STREET NAME'] = le.fit_transform(df['ON STREET NAME'])
```

Listing 6: Label Encoder on string values

After that, we can divide our dataset into train and test data, build the model, train it with the train set, and calculate the accuracy of the predictions by comparing the predicted results to the test set's actual results.

```
# Create the model
model = RandomForestClassifier(random_state=42)
# Train the model
model.fit(train_df[features], train_df['CONTRIBUTING ...'])
# Make prediction
predictions = model.predict(test_df[features])
# Calculate accuracy
accuracy = accuracy_score(test_df['CONTR ...'], predictions)
```

Listing 7: Create and Test the Random Forest model

When we apply this algorithm to different dataframes, we can see that in the case of the dataframe where we reduced the number of values for the reasons for the accidents, the prediction reached an accuracy of 35,7% while in the original dataframe the accuracy stops at only 28%.

5.4.3 Random Forest Tuning

Once we realized that the dataset with the reductions made by hand was the best, we shifted our attention to the algorithm, specifically the parameters it uses to create the model. These parameteres are called `hyperparameters` and are used to either enhance the performance and predictive power of models or to make the model faster. The hyperparameters that we will use to enhance the predictive power are:

- **n_estimators**: Number of trees built by the algorithm before averaging the products.
- **max_depth**: Length of the longest path from the root node to a leaf node. The depth of a tree is a measure of how many splits it makes before arriving at a prediction.

We will make the prediction of missing values using the following three different combinations:

```
param_sets = [
    {'n_estimators': 50, 'max_depth': 10},
    {'n_estimators': 100, 'max_depth': 20},
    {'n_estimators': 150, 'max_depth': 30}
]
```

Listing 8: Random Forest hyperparameters

We will make sure that train and test data will be the same for each combinations by setting a constant random state during data splitting. We also set a random_state variable to the Random Forest model(see also chapter 5.4.2) in order to reproduce the same simulations otherwise we would get different models every time. We obtained the following results by applying the model with different parameters to our manually reduced dataframe.

```
{'n_estimators': 50, 'max_depth': 10}
Accuracy: 0.39490156052091147
{'n_estimators': 100, 'max_depth': 20}
Accuracy: 0.38862586439413505
{'n_estimators': 150, 'max_depth': 30}
Accuracy: 0.36040367769729914
```

Against all expectations the simpler model seems to behave better, however, the strange result is due to the distribution of collision factor classes.
In fact, analyzing the forecasts made by the various models, we noticed how the first simply predicted almost always (90%) the same label, which turns out to be the one most present in the dataset after the unspecified one. We then try to make a classification of the missing specifications by balancing the classes, namely, to provide a fair number of examples for each label in order to teach the model to classify them better.

```
labels = df['CONTRIBUTING FACTOR VEHICLE 1']
# Get unique classes
unique_classes = np.unique(labels)
# Calculate class weights
class_weights = compute_class_weight("balanced",
                    classes=unique_classes, y=labels)
# Create model with balanced classes
model = RandomForestClassifier(class_weight=dict(
        zip(np.unique(labels),class_weights)),
        n_estimators=n_estimators,max_depth=max_depth,
        random_state=42)
```

Listing 9: Random Forest balanced classes

We got far worse results than expected by balancing the classes:

```
{'n_estimators': 50, 'max_depth': 10}
Accuracy: 0.05019327176510614
{'n_estimators': 100, 'max_depth': 20}
Accuracy: 0.09197112605910059
{'n_estimators': 150, 'max_depth': 30}
Accuracy: 0.09755407715294089
```

5.4.4 *Analysis Summary and Results*

In order to predict values for missing accident motivation labels, we first investigated how to automatically reduce the number of possible labels using an algorithm such as word2vec. Following that, we worked with the manually reduced dataframe to see how changing model hyperparameters like the number of estimators and maximum depth affects accuracy. Finally, we used class balancing to create a model based on more uniform values, but with a significant decrease in model accuracy. As a result, the best model obtained has only a 9,7% prediction accuracy. The balanced predictor's poor performance can be attributed to two major factors. To begin, the presence of a very imbalanced class distribution, in which some classes (contributing factors) had only a few instances, hurt the model's ability to learn and predict accurately. Second, due to their uniformity, the model's features were deemed insufficient, limiting the model's ability to capture meaningful patterns. The interaction of these factors produced less-than-desirable predictive accuracy results. Despite the fact that the case without balanced classes fails to detect suitable prediction patterns, the case with the accuracy of 36% was found to be the best for the continuum of our research. In fact, it succeeds in both predicting values other than the main *driver inattention*, but at the same time maintaining the same distribution of classes.

5.5 Uncovering Incident Patterns through Clustering

After reconstructing the motivations for the whole dataset we wanted to investigate common patterns in collisions in order to find the most critical ones. To achieve this goal our idea is to use a clustering technique so as to understand the main factors of different car accident. Unfortunately due to the high number of columns before proceeding with a clustering algorithm we had to perform a second operation of dimensionality reduction. This was needed since the K-Medoids algorithm is sensitive to high dimensional data. A comparison between normal K-Means, K-means with PCA and K-Medoids will be discussed later in the report since we prefer first to introduce all the methodologies and then discuss the comparison.

5.5.1 *Principal Component Analysis*

We agreed that PCA was the best option in order to perform the operation of dimensionality reduction. Basically PCA takes the dataset as input, computes the eigenvalues and eigenvectors of the whole dataset and finds the principal components. This are useful since the goal is to catch the maximum variance of the original data just by considering the principal components that contains the highest variance. The eigenvectors obtained through this process define the orientation and direction of every principal component and are associated to a single eigenvalue. On the other hand eigenvalues represents the variance linked to each principal component. The higher the value of the eigenvector, the higher the variance linked to the principal component. By combining eigenvectors and eigenvalues it is possible to find the components with the highest variance along with their directions. The goal of the process, as already said, it to keep a few amount of principal components that still represent a sufficient portion of the original variance of the dataset. Since the dataset is too big we had to decide how to shrink it, in order to have enough computational power to compute the PCA. We decided to focus on the pedestrians, filtering out the rows that had no pedestrians injured or killed.

```
df = df[(df['NUMBER OF PEDESTRIANS INJURED'] > 0) | \
(df['NUMBER OF PEDESTRIANS KILLED'] > 0)]
```

Listing 10: Filters on pedestrians

In order to apply PCA we have to have dense data and, since the algorithm is based on matrix multiplications, we had to encode all the string columns into numbers. We started by identifying numerical and categorical columns:

```
cat_cols = df.select_dtypes(include=['object']).columns
num_cols = df.select_dtypes(include=['float64', 'int64']) \
           .columns
```

Listing 11: Identification of numerical and categorical columns

And then we applied an encoder and a scaler to them. For the categorical columns that contain strings, we applied the One Hot Encoder which creates binary columns for each category without assigning them an order.

```
cat_encoder = OneHotEncoder(drop='first', sparse=False, \
              handle_unknown='ignore')
X_cat_encoded = cate_encoder.fit_transform(df[cat_cols])
```

Listing 12: Application of Encoder on categorical columns

For the numerical columns instead, we applied a scalar function to create numerical columns with comparable scales.

```
num_scaler = StandardScaler()
X_num_scaled = num_scaler.fit_transform(df[num_cols])
```

Listing 13: Application of Standar Scaler

At this point we can combine the resulting columns and apply the PCA to obtain the principal components:

```
pca = PCA(n_components=len(features))
X_transformed = pca.fit_transform(X_combined)

eigenvalues = pca.explained_variance_
sorted_eigenvalues_indices = np.argsort(eigenvalues)[::-1]

sorted_eigenvalues = eigenvalues[sorted_eigenvalues_indices]
cumulative_variance_ratio = np.cumsum(sorted_eigenvalues) /
np.sum(sorted_eigenvalues)
```

Listing 14: Application of PCA

After ordering the eigenvalues from the highest to the lowest it is possible to compute the cumulative variance ratio and find out how many components are needed to keep a certain percentage of the variance of the original data. We found out that out of 16 initial feature, we only needed 9 principal components to have a total variance of 90Now that we have found the number of principal component needed

we can consider the matrix `X_transformed` and use it to find the transformed data using only 9 components. In fact the matrix will have a number of rows equal to the initial number of observations, and the number of columns equal to the number of components given as input to the PCA algorithm.

```
num_components = percentage_90
X_pca = X_transformed[:, :num_components]
```

Listing 15: Extraction of transformed data with only 9 principal components

In this way the columns of the matrix are reduced to 9, the first 9 principal components. All of the process was useful to reduce the dimensionality of the data, making easier for the clustering algorithm to identify different clusters.

5.5.2 K-Medoids Algorithm: How it Works and Parameters

Now that we have prepared our data, we are ready to apply K-Medoids algorithm. Since we had difficulty with k-means in identifying correct clusters, we opted for k-medoids which uses medoids instead of centroids. A medoid is an existing data point that is considered as the center of the cluster and the distances are computed from it.
First of all we have to determine the optimal number of clusters. To do that we analyzed the results given by two different metrics:

- Elbow Method
- Silhouette Method

The first one consists of applying the K-Medoids algorithm to an increasing number of k and, for each iteration, compute the Sum of Squared Errors that measure how well the data fits into the clusters(for a detailed explanation see also [5]).
This is simply done by the code below:

```
inertia_values = []
    for k in range(1, 30):
        kmedoids = KMedoids(n_clusters=k, random_state=42)
        kmedoids.fit(data)
        inertia_values.append(kmedoids.inertia_)
```

Listing 16: Elbow Method Code

Once all the inertia values are computed, which are the SSE mentioned above, we plot the results and look for the elbow in the graph.

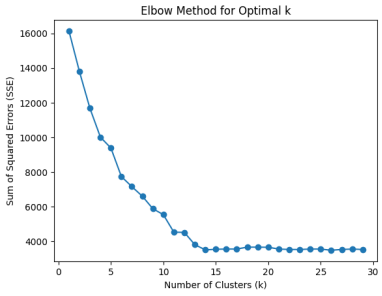

Fig. 9: Elbow Graph for K-Medoids

As we can see the elbow is clearly visible starting from k = 11 or 12, even though 13 and 14 could be correct choices too.
The method helps in visually identifying the best choice for the cluster making evident that at some point the increment is not really necessary. It is also important to remember that this is an empirical approach whose results are decided visually based on the plot generated and, although good results can be derived from it, it is important to test different methodologies and compare the results. For this reason we analyzed even the silhouette method and compared the results in the end.

The silhouette method, just like the elbow, apply the K-Medoids algorithm to an increasing number of k but differently from the elbow method computes the silhouette value for each single element and then computes the silhouette average value for the entire dataset.
The underlying code computes the silhouette value:

```
silhouette_scores = []
possible_k_values = range(2,21)
for k in possible_k_values:
kmedoids = KMedoids(n_clusters=k, random_state=42)
labels = kmedoids.fit_predict(data)
silhouette_avg = silhouette_score(data, labels)
silhouette_scores.append(silhouette_avg)
```

Listing 17: Silhouette Method Code

The silhouette score is used to evaluate how well elements within the same cluster are grouped together and how distinct each cluster is from the other. The score ranges between -1 and +1 where +1 specify that the elements are correctly positioned inside that cluster, while -1 indicates that some misassignment could have happened. In our analysis below we can observe some interesting results:

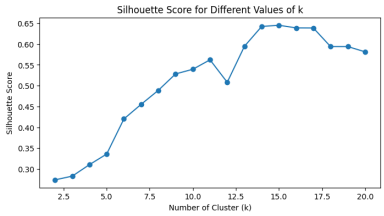

Fig. 10: Silhouette Score Graph for K-Medoids

Where:

```
The best number of cluster (k) is: 15
With a score of 0.6385800546949424
```

It is interesting to see that for k equal 15 we have the maximum, meaning that the assignment of cluster is the most optimal and 15 is the best choice for the number of clusters. This result is slightly different from the one obtained in the elbow method and for our analysis we chose to use the result derived from the silhouette method. In order to choose which one to use we found two interesting articles that confronts the two different metrics and come to the conclusion that due to its heuristic nature and the high dimensionality of the data, the elbow method is not recommended while the silhouette method can help more in identifying clusters especially with a complex dataset (sources [6], [7]). Once the k was determined we applied the K-Medoids with k = 15:

```
num_clusters = k
kmedoids = KMedoids(n_clusters=num_clusters, random_state=42)
kmedoids.fit(X)
clusters = kmedoids.labels_
data['Cluster'] = clusters
```

Listing 18: K-Medoids Code

This code apply the K-Medoids using the Manhattan distance to our data and assign, to the original dataset, the cluster to which a specific observation belongs to. Due to the fact that our data are in 8 dimensions the visual results printed on a 2D plane was not informative enough, for this reason we tried to plot a 3D representation of the first three principal components that allows to identify some of the clusters:

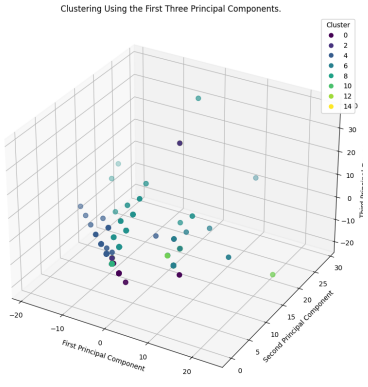

Fig. 11: 3D Clusters representation

We even have created an interactive version of this 3D representation the helps in better visualizing the clusters. We can notice how the clusters are not really distinguishable and the representation seems to be a bit confusing. For this reason we have conducted an analysis on the variables of each clusters to find out patterns.

5.5.3 Performance Comparison

After having introduced all the algorithms we want to explain why we decided to apply PCA and then use K-Medoids. First of all due to the high dimensions of the data and the numerous observations it was unfeasible for us to represent a comparison between the complete data so, in order to make the comparison as fair as possible, we sampled a total of 15000 observations. Below we can seen the comparison between K-Means with PCA and normal K-Means:

1) As already said, the first thing we can notice is that PCA helps incredibly in doing the computations and lets us use a higher number of observations, making the research more complete.

2) It is possible to notice that, with the same number of clusters, the Sum of Squared Errors is much lower when we apply PCA. This means that the points are near the centroids and the identified clusters are therefore more compact, which is not the case for k-means without PCA.

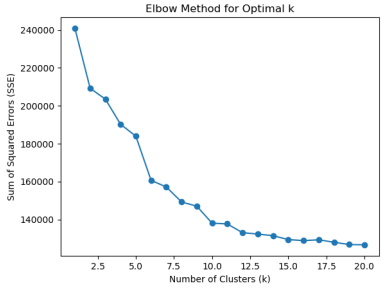

Fig. 12: Elbow Method Without PCA for K-Means

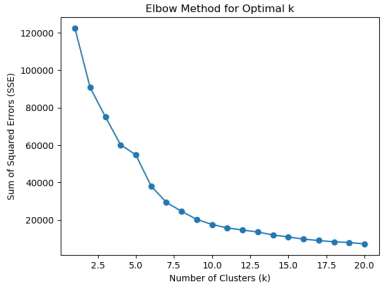

Fig. 13: Elbow Method With PCA for K-Means

3) In the silhouette case instead we can notice how, with the same cluster, the silhouette score is much lower. A silhouette score near 0 specifies that the observations are near the decision boundaries between two different clusters which creates uncertainties about which clusters each observation belongs to. This is not the case of the k-means with PCA where the scores are much better.

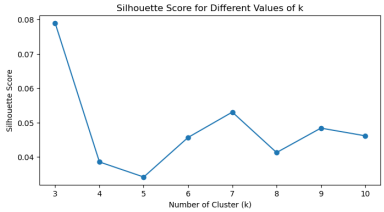

Fig. 14: Silhouette Method Without PCA for K-Means

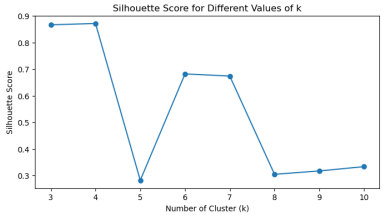

Fig. 15: Silhouette Method With PCA for K-Means

For all this reasons we decided not to further investigate the cluster created without the application of PCA since the resulting clusters would have been uncertain and confusing. Instead the reason behind the use of K-Medoids is that the results obtained with the K-Means algorithm were not satisfying. The first cluster identified was in fact composed

of more than 70.000 points out of a total of 75.000 points. We can see how cluster 0 basically contains all the accidents in every borough from the graph below:

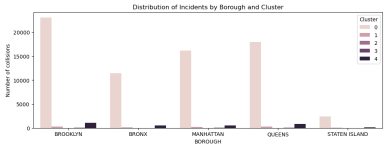

Fig. 16: Inicdents grouped by Borough and Cluster

The application of K-Medoids, on the contrary, is able to find out 15 different clusters that are analyzed below. Moreover we have found an article that analyzes the two approaches and concludes that K-Medoids performs better with large datasets, source [8]. It is important to say that since K-Medoids is computationally more expensive than K-Means, the final results discussed below have been obtained by selecting a portion of the original dataset containing the years: 2020, 2021 and 2022.

5.5.4 *Resulting Cluster Analysis*

To analyze the cluster we tried to investigate the frequency of the number of collisions by month across the three years, obtaining the following result:

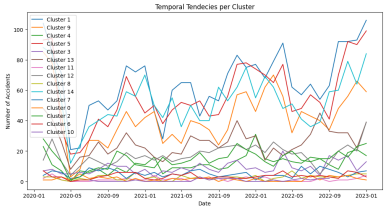

Fig. 17: Temporal Analysis for each Cluster

We can see how different clusters identifies different time trends so our idea to find pattern was to divide the clusters in order to identify three different macro groups. Every group is characterized by the monthly number of collisions. If a specific clusters has, for example, more than 4 months in which the total number of incidents is higher than 50, then that clusters identifies a high severity pattern. If the number of months is below that but at least 3 months are between 20 and 50 number of incidents, then we have a medium severity cluster. And finally the remaining clusters become low severity clusters. The final results is:

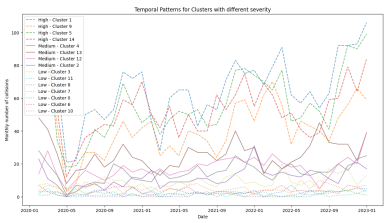

Fig. 18: Temporal Analysis for each Cluster divided by severity

Now we can analyze the clusters that belong to a certain severity class in order to find some patterns. By looking at `CRASH TIME`, `BOROUGH`, `ON STREET NAME`, `VEHICLE TYPE CODE 1` and `CONTRIBUTING FACTOR VEHICLE 1` we have found out some interesting results:

Fig. 19: High severity cluster analysis

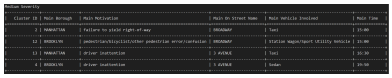

Fig. 20: Medium severity cluster analysis

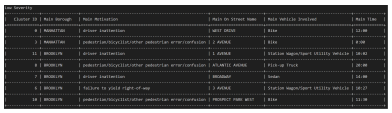

Fig. 21: Low severity cluster analysis

The tables illustrate the most common element for each column in each cluster and we can see how Brooklyn and Queens are the most dangerous borough and the majority of incidents is caused by vehicle like Sedan and Station Wagons. Moreover information like the time of the day helps in understanding the most dangerous hours. The other tables instead, show different data. In the case of medium severity we can notice how a lot of incidents are caused by taxi and we can even see how the time is different from the hours in the first table. Finally we can notice in the last table, how a lot of collisions involved bicycle, again during different times of the day. The results can be analyzed in this way to find out specific situations that contribute to daily accident.

6 Societal Impact and Privacy Aspects

The results obtained highlight how the Brooklyn area is the most dangerous due to a high number of collisions. The main reason behind this being driver inattention could be the starting point for awareness and educational campaign and could bring to the introduction of additional controls along those streets. For what concern the other results we can see how accidents involving bicycles are a non-negligible part of total accidents, thus some bike lane could be created or improved in the specified streets. It is important to say that, although collisions are classified by severity, we don't mean to say that low severity collisions are less important than the high severity one. The severity of the patterns reflects how in certain zone the number of incidents is higher than others and suggest insights that can be used, for example, by the local police authorities to increase controls. For what concern privacy, no specific detail about specific collisions is given out through this study, in order to ensure the privacy of all the person involved in specific car collision. Moreover during the study we didn't make use of sensitive data like plate's number or sensitive information of the drivers and pedestrians involved, like names and surnames.

7 Conclusion

We can conclude that, thanks to the cluster analysis, we were able to find some interesting patterns that occur, and we were able to characterize the collisions by identifying recurrent patterns that can be used as the starting point in seeking solutions that limit the number and impact of incidents. The data obtained, then, can be analyzed again by focusing on different components to identify additional hidden patterns. However, we must keep in mind that the results were obtained on a limited subset of data, due to computational reasons, which however contain as mentioned above at least one wounded pedestrian. Further analysis could be done on larger portions of data that contain a greater amount of information, although the ability of the clustering algorithms would have to be evaluated, since the more the number of data and dimensionality increases, the more difficult it becomes to identify different clusters. Finally, additional analysis could be done on the motivations behind each incident since driver inattention appears to be a too general motivation. One could for example investigate the factors that lead a driver to be distracted perhaps by comparing with other motivations already within the dataset. In this way more specific motivations could lead to different patterns and to the application of different measures to prevent the number of accidents even more.

Appendix

Chapter	Author
1	Matteo and Stefano
2	Matteo
3	Matteo and Stefano
4	Matteo
5.1 - 5.4	Matteo
5.5 - 5.5.4	Stefano
6	Stefano
7	Matteo and Stefano

TABLE 1: **Job Division**

All members contributed to writing the code for this project. Likewise, review and content selection for this report was done in collaboration by all members of the group. Thus, the content presented has been evaluated by all members and selected in collaboration.

References

[1] Dataset from the official website of the United States government https://catalog.data.gov/dataset/motor-vehicle-collisions-crashes

[2] API use Nominatim from OpenStreetMap https://nominatim.openstreetmap.org/

[3] Random Forest Algorithm Explanation https://www.simplilearn.com/tutorials/machine-learning-tutorial/random-forest-algorithm

[4] Random Forest Algorithm Image https://www.spotfire.com/glossary/what-is-a-random-forest

[5] K-Means, Elbow and Silhouette Methods Explanation https://uc-r.github.io/kmeans_clustering

[6] Deciphering Optimal Clusters: Elbow Method vs. Silhouette Method, Megha Natarajan, 2023 https://medium.com/@megha.natarajan/deciphering-optimal-clusters-elbow-method-vs-silhouette-method-7e311c604201

[7] Silhouette Method — Better than Elbow Method to find Optimal Clusters, Satyam Kumar, 2020 https://towardsdatascience.com/silhouette-method-better-than-elbow-method-to-find-optimal-clusters-378d62ff6891

[8] Analysis and Approach: K-Means and K-Medoids Data Mining Algorithms , Dr. Aishwarya Batra https://www.apiit.edu.in/downloads/all%20chapters/CHAPTER-59.pdf