Homework 3 - Data Mining - Sapienza

Ivan Fardin 1747864

December 6th, 2020

Contents

1 Problem 1

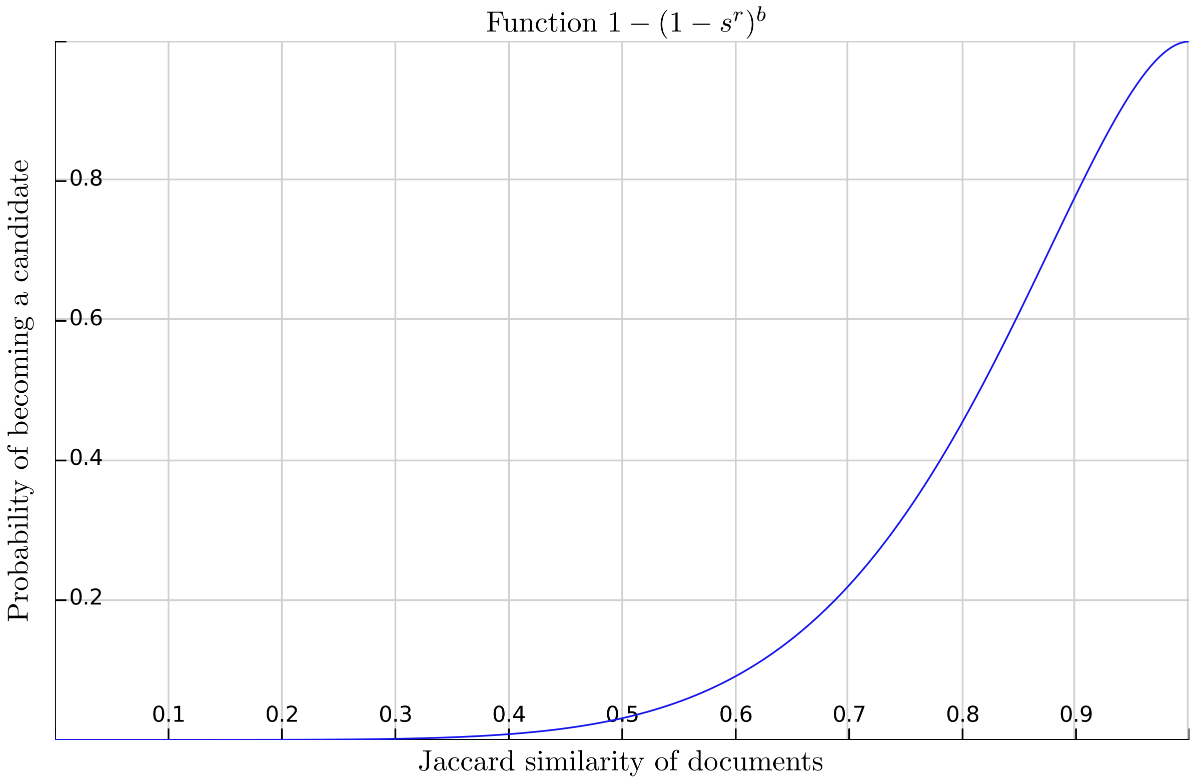
This problem requires implementing nearest-neighbor search for text documents using the locality-sensitive hashing (LSH) technique that allows us to focus on pairs that are likely to be similar, without having to look at all pairs.

Since we want to find Amazon products (downloaded in the previous hw) that are near duplicates, in *main.py* I start loading the *products.tsv* file and extracting every product description from it.

Then, I perform the shingling which converts each document (product description) into a set of the hashes of the shingles of length 10 characters. This phase is implemented in the *Shingling* class in *shingling.py*.

The next step is minhashing, which is a way to convert large sets into much smaller representations, called signatures.
The length n of each signature is equal to $n = b \cdot r$ where b represents the bands and r the rows used in the LSH algorithm. Since, we assume that two products are near duplicates if the Jaccard coefficient of their shingle sets is at least 80%, I found values for r and b that can give us the desired behavior and then the size of each signature.

$$\frac{1}{b}^{\frac{1}{r}} \approx 0.8 \Rightarrow \frac{1}{2}^{\frac{1}{6}} \approx 0.8 \Rightarrow n = b \cdot r = 12$$



The signature consists of n minhashing values where the permutations are simulated using n randomly chosen hash functions. This phase is implemented in the *MinwiseHashing* class in *minwiseHashing.py*.

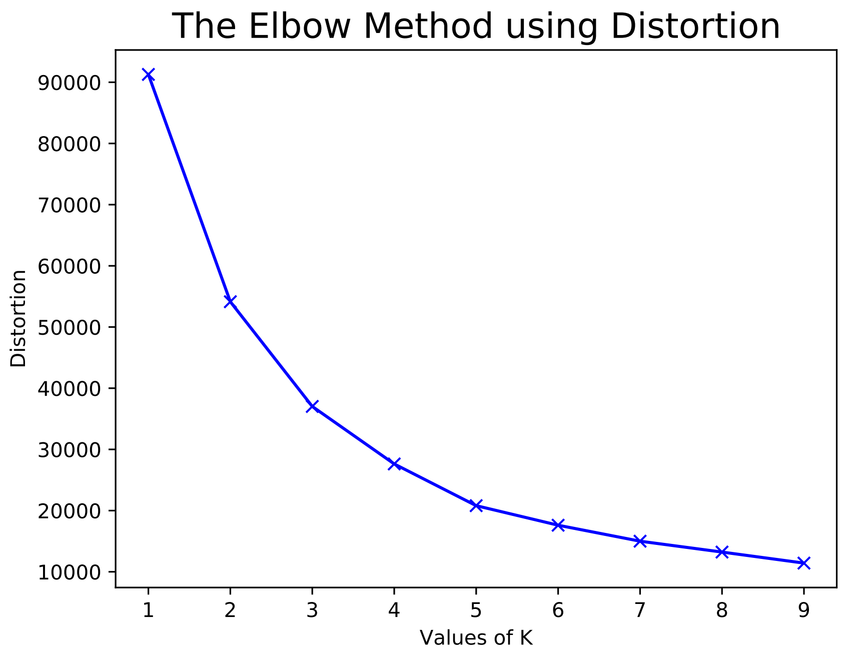
Finally, I apply the bucketing idea inherent in LSH to signatures, so that, given a collection of minwise hash signatures of a set of documents, it finds all the documents pairs that are near each other (similar items are more likely to be hashed to the same bucket than dissimilar items are).
This final phase is implemented in the *LocalitySensitiveHashing* class in *localitySensitiveHashing.py*.

To test the LSH algorithm, I implemented in *utils.py* a function that given the shingles of each of the documents, finds the nearest neighbors by comparing all the shingle sets with each other.
Then I report the number of duplicates found in both cases, the size of the intersection along with the time required to compute the near duplicates in either case.

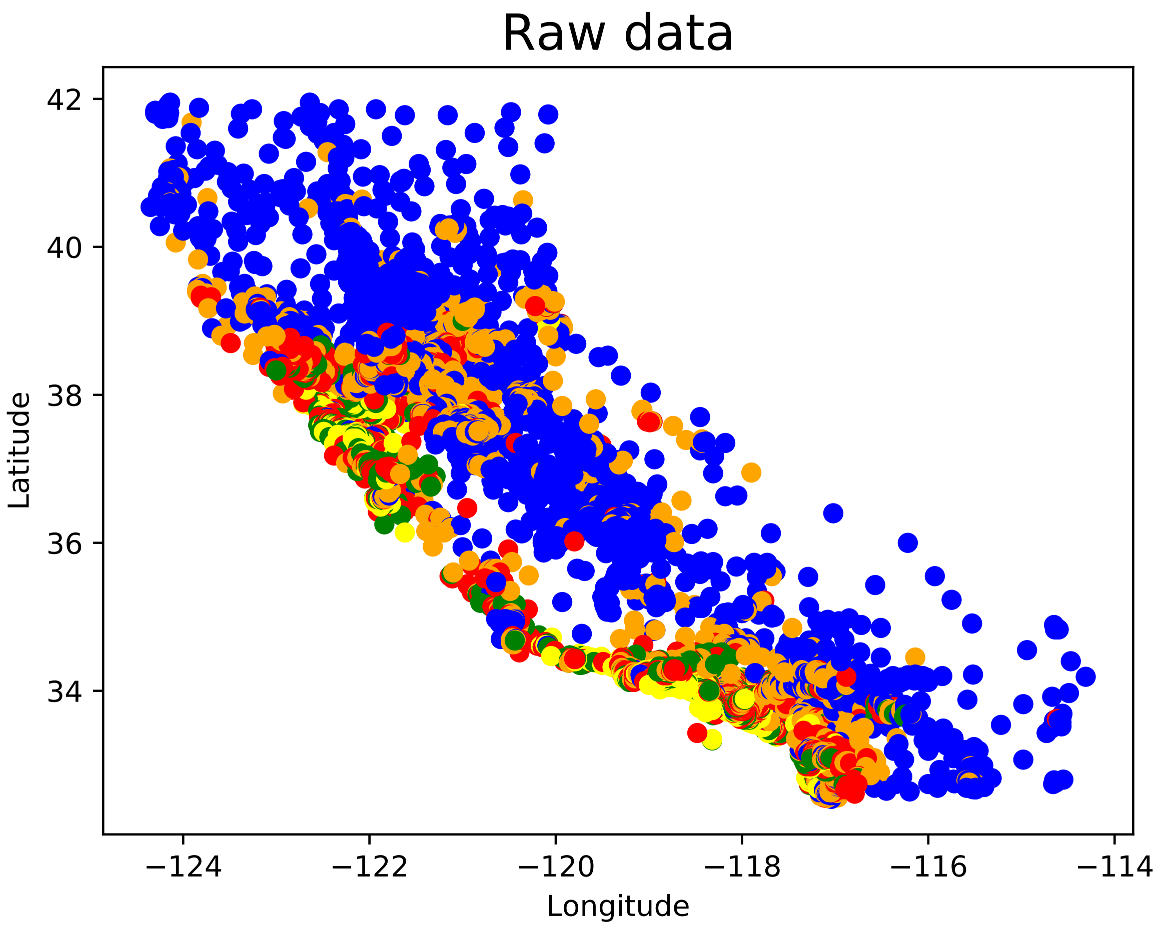
2 Problem 2

To cluster the samples in the California Housing Prices dataset, I used the **k-means++** algorithm and the **elbow method** for finding the most suitable number of clusters k. Moreover, I evaluated clustering performance based on the Silhouette Score and visualizing the clusters in a longitude and latitude coordinates and coloring each cluster a different color.

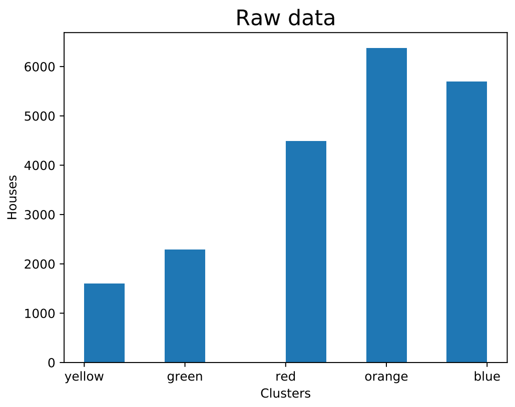
First, in *clustering.py*, I perform clustering just on raw data, so I only drop categorical columns and any row that contains a missing value. The elbow method suggests that the most suitable number of clusters k is 5.



The corresponding Silhouette Score on raw data is about 0.577 and the clustering looks very confusing.



The size of each cluster is



Then, I repeat performing the clustering using some feature engineering technique on the data. This time by imputating (instead of dropping) any row that contains a missing value, filling categorical missing values with a new category called *Other* and missing numerical values with the medians of the columns.

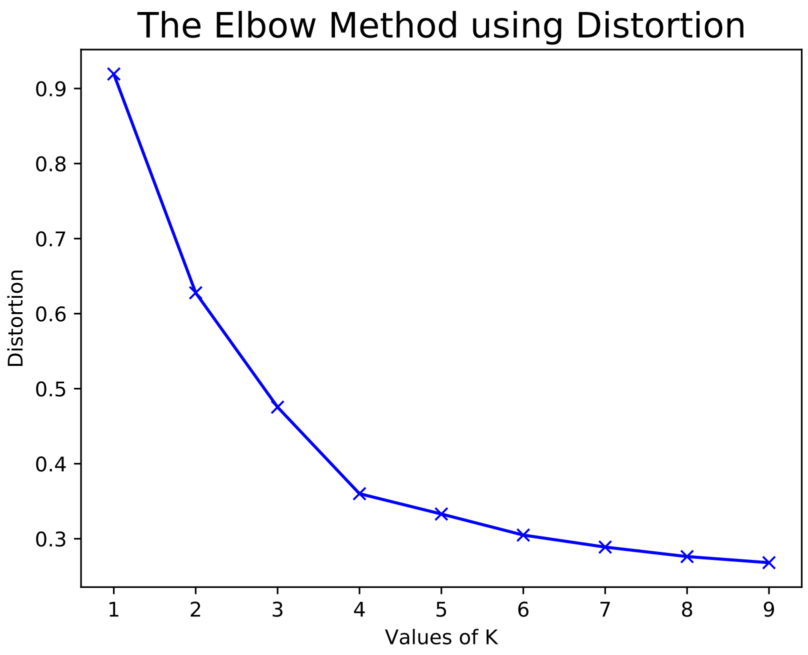
So, I encode categorical columns (ocean proximity in this case) values as a one-hot numeric array in order to use this information that with just raw data I was forced to drop.

As a further step, I remove outliers from each column (except the encoded categorical ones) by assuming outliers all values having a distance from the mean greater than 3 times the standard deviation.

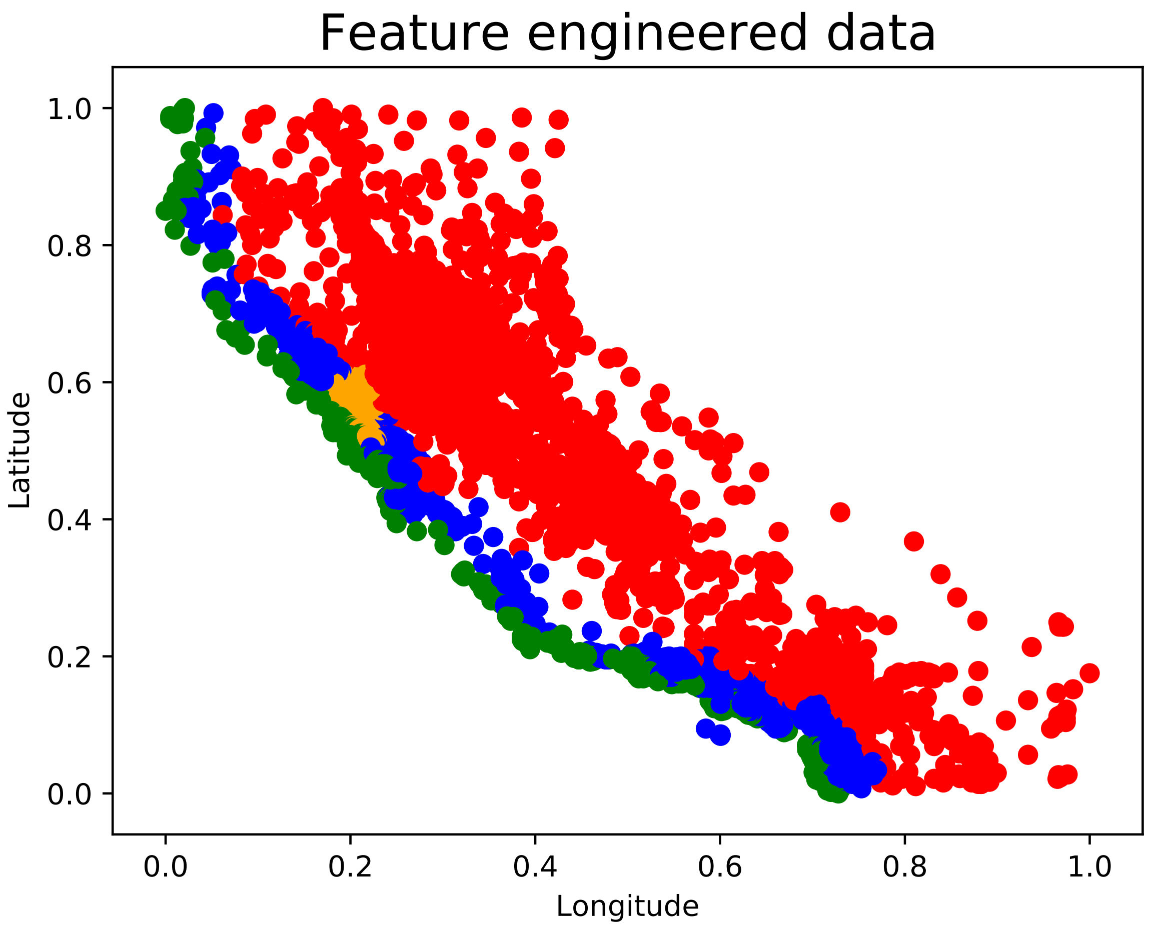
Then, I try to approximate the data distribution of each column (except the longitude, latitude and encoded categorical ones) to the normal distribution by log transforming each value to reduce their skewness.

Eventually, I normalize data (except the encoded categorical ones) because the continuous features become identical in terms of range after a scaling process.

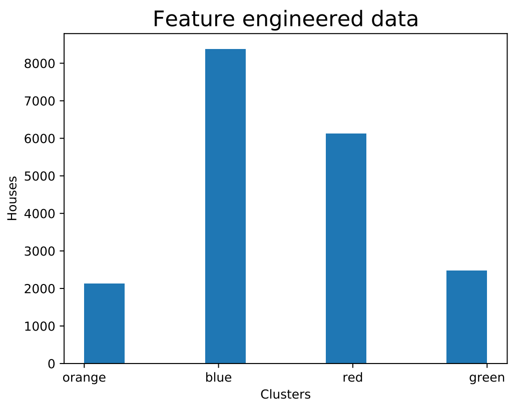
The elbow method suggests that the most suitable number of clusters k is 4.



The corresponding Silhouette Score on data is about 0.666 and the clustering looks reasonable.



The size of each cluster is



Comparing the obtained clusters in the two versions, in the feature engineered case, they are clearly divided into three bands:

- green = houses on the coast
- red = houses away from the coast
- blue = houses in the middle between green and red

plus a well-defined orange group concentrated in an area within the blue band.
In the raw data case instead, even if there is an additional cluster that might suggest a better clustering, clusters appear almost random or grouped together without an understandable pattern.

Furthermore, the elbow curve shows a huge improvement in terms of distortion (0.9 vs 90000) and steeper shape with k between 2 and 4.

Finally, the better clustering with feature engineered data doesn't cost in terms of running time of the k-means++ algorithm since in both versions it takes about 1.5 seconds with the aforementioned clustering slightly faster than the other.