CMPS242 Machine Learning Final Project Report

Eriq Augustine EAUGUSTI@UCSC.EDU
Student ID: 1116667

Varun Embar VEMBAR@UCSC.EDU
Student ID: 1566148

Dhawal Joharapurkar DJOHARAP@UCSC.EDU
Student ID: 1566168

Xiao Li XLI111@UCSC.EDU
Student ID: 1461332

Team 0: Para-normal Distributions - https://github.com/eriq-augustine/242-2016

1. Problem Statement

Our project is focused around helping business owners find potential competitors if they choose to open a branch in a new location. Using this information the business owners can not only find potential competitors, but also get an idea of how well their business will be received in the new locality.

We say that a "competitor" is any similar business, with the idea that a similar business caters to a similar clientele and is therefore a competitors. Therefore our task involves finding similar businesses in other regions. We do this by first clustering similar businesses. Once we have the clusters, to find similar businesses in a location, we look into the cluster to which a given restaurant belongs to, and then display restaurants in that cluster that are close to the location.

We use the k-medoids algorithm to cluster the businesses as not all attributes are numeric in nature and we need a richer set of dissimilarity scores. We run experiments with various parameter settings, dissimilarity scores, and features and report the rand index on a "gold standard" data set.

2. Algorithm Formulation

We use the K-medoid clustering algorithm to cluster businesses. K-medoids is a clustering technique that tries to minimize the pairwise dissimilarity between data points that are assigned to a cluster and the medoid of that cluster. The medoid is a data point in data set that best represent the cluster center. The medoid is analogous to the centroid in the K-Means algorithm.

The K-Medoids algorithm has the following advantage over K-Means.

- Since we are only using pairwise comparisons, we do not need to be able to calculate the average of a feature. This is better for non-numeric features like strings or sets.
- Since we are only using pairwise comparisons, once we compute the dissimilarity between the data points, we can do away with the actual data points, and this leads to a reduction in the memory usage and makes the computation more efficient.
- Since our medoid is a real data point (as opposed to a centroid), we can use this data point as a "representative" when visualizing our data (or perhaps even sampling it).

Our stopping condition is either when we have run through 10 iterations of clustering (experimentally found to be sufficient with this dataset), or when the current set of clusters is the same as either the last run or the run before it. We compare not just the most recent run so that we can prevent unnecessary runs when the clusters are *jittering*, or when outlier points are constantly shifting their membership back and forth between two or more clusters.

3. Distance Metrics

To compare our features, we use several different distance metrics. In this section, we will discuss only metrics that were used in our final experiments. However, we implemented and explored several different metrics that will be covered in Appendix **??**.

Our distance metrics fall into two categories: numeric and set-based. For numeric, we implemented L1 and L2 distances. For set-based, we implemented Jaccard and Dice.

To make combining the distance of different features together easier, we also attempt to normalize the output of the distance to ideally be in the range 0 to 1.

3.1. L1 (Manhattan) Distance

The Manhattan distance of two numbers x and y is calculated by:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

3.2. L2 (Euclidean) Distance

The Euclidean distance of two numbers x and y is calculated by:

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.3. Jaccard Distance

The Jaccard distance of two set A and B is calculated by:

$$d_j(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

3.4. Dice distance

The Dice distance of two set A and B is calculated by:

$$d_d(A, B) = 1 - \frac{2|A \cap B|}{|A| + |B|}$$

3.5. Normalizations

Because different distance metrics work with different output ranges, we need to normalize the distances metrics to something consistent. For example, the range of Jaccard Distance is between 0 and 1 while the range of Manhattan Distance is 0 to infinity.

To figure out what works best, we will try three different normalization methods:

1. **Raw** - No normalization is applied.

2. **Logarithmic** - Use a logarithm to squash down the value closer to zero. Since we did not want to let values less than 1 grow, we put a hinge at 1 and just let all values less than 1 go to zero.

$$distance(d) = ln(max(1, d))$$

3. **Logistic** - Use the Sigmoid function to squash the distance down to a 0-1 range. Since all our distance metrics return non-negative values, we can ensure that the following returns a value in the range of 0 to 1:

$$distance(d) = (\frac{1}{1 - exp(-d)} - 0.5) * 2$$

4. Features

Our set of features can be divided into three types of features:

- Numeric Features
  - **Star Rating** - The star ratings of a business, rounded to half-stars.
  - **Total Review Count** - The total number of reviews present for the business.
  - **Available Review Count** - The number of reviews available for the business in the data set.
  - **Mean Review Length** - Average length of a review for the business.
  - **Mean Word Length** - Average length of the words in the reviews for the business.
  - **Number of Words** - Total number of words in the reviews of the business.
  - **Mean Words per Review** - Average number of words per review available for the business.

- Descriptive Features
  The dataset also contains some textual data such as attributes, categories, review text, etc. which describe the businesses. We construct features using these texts, but to improve efficiency of our model we encode the string data as a set of unique integers by building maps over all possible values that these features can take. So, our output feature is a set of identifiers of the words present for the business.
  - **Attribute Features** - The dataset contains information about attributes of businesses which describe the operations of the business. They exist as key-value pairs in the data, for example - (WiFi, no), (Delivery, false), (Attire, casual). We squash the key-value pairs together and add these together to create a set of attributes that a business has, which we then use as a feature.
  - **Category Features** - The dataset also contains some categorical information about the business, for example, whether the business is a restaurant, cafe, food place, burgers place, etc. We construct the feature which is the set of categories that the business has assigned to it.

  - **Key Words** - These are words that Yelp has defined to help users in filtering out the businesses that appear in the search results. They're words that delineate businesses as they're mostly categorical words such as restaurant, cafes, etc. We use these key words and look for their occurrences in the reviews of the businesses and return a set of key words that they contain.
  - **Top Words** - This set contains the most frequently occurring words in the reviews of the text, after taking care of the stop words. We used a general English language stop words list containing 562 stop words.

- Temporal Features
  We have two time-related features pertaining to the functioning hours of businesses

  - **Total Hours** - Total number of hours the business is open during the week
  - **Open Hours** - This is a feature encoded information about the functioning hours of the business over the week. We divided the hours in a day in the following way to help us attribute the functioning times to a business.
    * Open between 6AM - 12PM: The restaurant functions in the morning, or serves breakfast.
    * Open between 12PM - 3PM: The restaurant functions in the afternoon, or serves lunch.
    * Open between 5PM - 9PM: The restaurant functions in the evening, or serves dinner.
    * Open between 9PM - 2AM: The restaurant functions post dinner, or late night.

    However, to make our feature more robust, we specify that to encode that a business functions during a specific time-span specified above, it must be open for at least 4 days in a week during those hours. We again return a set of the time-spans that a business operates during as a set.

5. Evaluation

In order to evaluate the correctness of the generated clusters, we create sets of restaurants that are similar. Using these sets as the "gold standard" clusters, we report the Rand Index.

5.1. Rand Index

Rand Index is used in data clustering to measure the similarity between two cluster assignments. Given a set of elements S, and two partitions of S, $X = \{X_1, X_2, \ldots, x_n\}$ and $Y = \{Y_1, Y_2, \ldots, Y_m\}$, we compute the following

- a - No. of pairs that are assigned to same cluster in both X and Y
- b - No. of pairs that are assigned to different clusters in both X and Y
- c - No. of pairs that are assigned to same cluster in X but to different clusters in Y
- d - No. of pairs that are assigned to different clusters in X but to same cluster in Y

The rand index is then given by $R = \frac{a+b}{a+b+c+d}$

5.2. Generating Gold Standard Clusters

Since we do not have the gold standard clusters, we cluster a subset of the data and evaluate the algorithm on these data points.

We looks at various restaurant chains such as "Taco Bell", "Starbucks" etc. that are present in the data and assign all the stores belonging to the same chain as a new cluster. We extracted the top 15 restaurant chains and their details are given in Table **??**. We only generate pairs within a restaurant chain. Since it is not clear if a McDonald's and a Burger King should be in the same or different cluster, we do not look at pairs across various chains.

However if we only have positive pairing, then the rand index can be trivially maximized by assigning all data points to the same cluster. To counteract this, we also need pairs of restaurants that should not be in the same cluster. We collected a list of 285 "fine-dining" restaurants (restaurants which have the highest price range) and create pairs such that, the first restaurant comes from the "fast food" list like "McDonald's" and "Taco Bell" and the other from the fine-dining list. These pairs should not be in the same cluster.

6. Experiments

We use all 3069 restaurants present in the gold standard dataset for our experiments.

There are four different parameters that we can tune in our algorithm:

1. D - the set of features
2. K - the number of clusters
3. F - the function used to normalize various distance metrics
4. S - the distance measure used to measure set similarity

We run multiple experiments, keeping a few of these parameters fixed, and altering others to better understand the sensitivity of each parameter.

Restaurants	No of branches
Starbucks	527
Subway	408
McDonald's	365
Taco Bell	193
Burger King	167
Pizza Hut	159
Wendy's	149
Panda Express	122
Dunkin' Donuts	122
Domino's Pizza	107
KFC	99
Chipotle Mexican Grill	97
Dairy Queen	96
Papa John's Pizza	92
Jack in the Box	81
Fine Dining	100

Table 1. List of restaurant chains

Features	Rand Index
A	0.9070
W	0.8523
N	0.6498
AW	**0.9295**
WN	0.6948
AN	0.7269
NAW	0.8267

Table 2. K = 10, F = logistic, S = Dice

In the first experiment, we modify the feature set keeping other parameters fixed. The possible features are N (*Numeric* features), A (*Attribute* descriptive features (attributes and categories)), W (*Word* descriptive features (key words and top words))[1]. We set K = 10, F to logistic normalization, and S to Dice. The results are shown in table **??**.

We observe that the combination of *Attribute* and *Word* gives the best performance, followed by *Attribute* alone. We also observe that numeric features lead to deterioration of performance.

In the second experiment, we keep the number of clusters and the feature set fixed and alter the normalization function F and the set distance. We use the setting K = 10, and use all the features (*Numeric*, *Attribute*, and *Word*). The results are shown in table **??**.

We observe that the best performance is achieved when we use Jaccard similarity. We also observe that using normalization does not make any difference to the metrics.

[1]Because of time constraints, temporal features were not included in these experiments. Partial results for temporal features can be found in Appendix **??**.

Normalization	Set Distance	Rand Index
Log	Dice	0.685091
Log	Jaccard	0.672223
Logistic	**Dice**	**0.826786**
Logistic	Jaccard	0.785894
None	Dice	0.667862
None	Jaccard	0.667858

Table 3. K = 10, D = NAW

k	Rand Index
8	0.7872
10	**0.8267**
12	0.7780
14	0.7652
16	0.7486
18	0.7464

Table 4. D = NAW, F = logistic, S = Jaccard

In our third experiment, we modify the number of clusters K, keeping the other parameters fixed. We set the set distance to Jaccard and the normalization to logistic and use all the features (*Numeric*, *Attribute*, and *Word*). The metrics are given in table **??**.

We observe that the best setting for K is 10. We also observe that as we increase the number of clusters from 10, the rand index decreases. This could be due to the good clusters being split into many clusters resulting in decreasing rand index. Another reason could be that the number of "same clusters" pairs in the gold standard dataset is much higher than the number of "across cluster" pairs. This could result in rand index preferring fewer and larger clusters over many small clusters.

The full results can be seen in Appendix **??**.

7. Conclusion

Clustering works fairly well for this dataset. The signals that most strongly indicate that two businesses are the same come from the reviews rather than the more structured data like number of stars. Our work with the texttual features was limited to just a few instances of "low-hanging fruit" But if the success of those features is any indication of the richness of the reviews, then the reviews should be the focus of future work.

Appendices

A. Temporal Feature Experiments

Time constraints lead us to not being able to complete the experiments with *Temporal* features. However, we do have paritial results.

It does not look like *Temporal* features are stronger than *Attribute* or *Word* features. However, we see *Temporal* features performing better than most combinations which include *Numeric* features. We already know that the *Numeric* features are detrimental, but it is very interesting that *Temporal* beats it when we consider what is represented by each feature. *Numeric* features contain information such as the restaurants star rating, but *Temporal* features just have information about when a place is open.

Feature Set	K	Scalar Normalization	Set Distance	Rand Index
NAWT	8	Log	Dice	0.694153
NAWT	8	Log	Jaccard	0.686108
NAWT	8	Logistic	Dice	0.774644
NAWT	8	Logistic	Jaccard	0.764851
NAWT	10	Log	Dice	0.700142
NAWT	10	Log	Jaccard	0.695933
NAWT	10	Logistic	Dice	0.761750
NAWT	10	Logistic	Jaccard	0.740966
NAWT	12	Log	Dice	0.694220
NAWT	12	Log	Jaccard	0.698511
NAWT	**12**	**Logistic**	**Dice**	**0.787857**
NAWT	12	Logistic	Jaccard	0.749919
NAWT	14	Log	Dice	0.704690
NAWT	14	Log	Jaccard	0.708275
NAWT	14	Logistic	Dice	0.765841
NAWT	14	Logistic	Jaccard	0.747909
AWT	10	N/A	Jaccard	0.718515
T	10	N/A	Jaccard	0.779835

B. Additional Distance Metrics

We implemented and explored using string similarity metrics to compute distance. However, we were not able to find suitable features to use string distance with.

B.1. Levenshtein Distance

The distance of two strings a and b is evaluate by Levenshtein distance:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j), \\ min = \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} \end{cases} \quad (1)$$

Then we normalize the Levenshtein distance $lev_{a,b}$ by transaction function to the range of [0, 1], l_a and l_b represent the length of 'a' and 'b'.

$$dis = \frac{lev_{a,b}}{\max(l_a, l_b)}$$

B.2. Needleman-Wunsch Distance

The distance of two string 'a' and 'b' is calculated by Needleman-Wunsch algorithm. We calculate the distance by comparing the letters in two strings. The scoring way is as follows:

- Match = 1: two letters a_i and b_j are the same
- Mismatch = -1: two letters a_i and b_j are the different
- Indel = -1:
  - Delete: one letter in the string 'a' aligns to a gap in the string 'b'.
  - Insert: one letter in the string 'b' aligns to a gap in the string 'a'.

The pseudo-code for the Needleman-Wunsch algorithm is as follows:

for i = 0 to length(a) **do**
  $F(i,0) \leftarrow -i$
end for
for j = 0 to length(b) **do**
  $F(0,j) \leftarrow -j$
end for
for i = 1 to length(a) **do**

  for j = 1 to length(b) **do**

    if a[i] == a[j] **then**
      $Match \leftarrow F(i-1,j-1)+1$
    else
      $Mismatch \leftarrow F(i-1,j-1)-1$
    end if
    $Insert \leftarrow F(i,j-1)-1$
    $Delete \leftarrow F(i-1,j)-1$
    $F(i,j) \leftarrow max(Match, Mismatch, Insert, Delete)$
  end for
end for

Then we need to normalize the Needleman-Wunsch distance NWa,b by transaction function to the range of [0, 1]. l_a and l_b are the length of string 'a' and 'b '.

$$dis = \begin{cases} \text{if min(i,j)=0} \\ \text{otherwise} \end{cases} \frac{NWa,b - \max(l_a, l_b)}{-2 * \max(l_a, l_b)}$$

C. Implementation Details

C.1. Code

Our code is hosted on GitHub: https://github.com/eriq-augustine/242-2016. Since we have implemented all our methods ourselves, *numpy* is the only dependency required to run our code.

C.2. Data

To better handle and analyze the Yelp data, we first put it into a relational database. We are using PostgreSQL. Our resulting schema was 12 tables in Boyce-Codd normal form. The create table statements can be found in our repository at data/sql/create.sql.

The script to parse the Yelp data and convert it to SQL insert statements can be found at data/sql/parse.rb.

Having the data in a relational database also gives us the advantage of indexes and precomputations. Non-trivial information like the term frequencies over all reviews can be precomputed and stored in tables for use in more complex features. Additionally, we can tune our feature query with indexes targeted at our specific features. The SQL file that handles precomputations and optimizations can be found at data/sql/optimize.sql.

C.3. Optimizations

Clustering may be a fairly simple task, but it can quickly get very resource intensive.

C.3.1. Memory

The Yelp dataset contains approximately 40000 restaurants (based on the given categories). Since the resulting distance matrix is symmetric matrix and we do not need to keep the values on the diagonal (all points are 0 distance to themselves), that means we need to compute exactly $n \cdot (n-1)/2$, or $40000 \cdot 39999/2 = 799,980,000$.

A naive method would be too keep these values in a map structure comprised of maps to floats. Assuming a float takes up 64 bits and an int takes 64 bits, then this means that a naive representation of the distance matrix would require a float for the value and two integers for the keys which would take up at least: $799,980,000 * (3 * 64) = 153,596,160,000 bits = 17.88 GB$.

This is a but unwieldy for most laptops. We can improve this by using a single array instead of a map. Then, we only need to store the actual distance value and not the keys. This would reduce our size cost down to; $799,980,000 * 64 = 51,198,720,000 bits = 5.96 GB$.

However, we can further reduce our memory cost by choosing a more cost-efficient data type. Since we are normalizing our distance functions to return a value in the range of 0 to 1, we do not expect our distances to grow too large. Therefore, we can feel safe using a smaller data type such as a 16-bit float. This will further reduce the size down to: $799,980,000 * 16 = 12,799,680,000 bits = 1.49 GB$, 1/12 our original cost.

C.3.2. Multiprocessing

Just the precomputation of the distance matrix for our ground truth involves calculating 4,707,846 similarities (which is still nothing when compared to the 799,980,000 required for all restaurants). To speed this up, we took advantage of multiprocessing and shared memory.

D. Full Results

Below are the full results from our experiments on different parameters. The "Feature Set" column describes the features that were included in that run.

- **N** - Numeric Features
- **A** - "Attribute" Descriptive Features (attributes and categories)
- **W** - "Word" Descriptive Features (key words and top words)

Note that scalar normalizations only make sense in the presence of numeric features ('N'), while set distance only makes sense in the presence of non-numeric features ('A' & 'W').

Feature Set	K	Scalar Normalization	Set Distance	Rand Index
NAW	8	Log	Dice	0.660370
NAW	8	Log	Jaccard	0.661637
NAW	8	Logistic	Dice	0.787228
NAW	8	Logistic	Jaccard	0.790557
NAW	8	None	Dice	0.659138
NAW	8	None	Jaccard	0.659138
NAW	10	Log	Dice	0.685091
NAW	10	Log	Jaccard	0.672223
NAW	10	Logistic	Dice	0.826786
NAW	10	Logistic	Jaccard	0.785894
NAW	10	None	Dice	0.667862
NAW	10	None	Jaccard	0.667858
NAW	12	Log	Dice	0.680260
NAW	12	Log	Jaccard	0.677921
NAW	12	Logistic	Dice	0.778079
NAW	12	Logistic	Jaccard	0.763160
NAW	12	None	Dice	0.669829
NAW	12	None	Jaccard	0.669829
NAW	14	Log	Dice	0.681115
NAW	14	Log	Jaccard	0.674080
NAW	14	Logistic	Dice	0.765209
NAW	14	Logistic	Jaccard	0.757906
NAW	14	None	Dice	0.670226
NAW	14	None	Jaccard	0.670226
NAW	16	Log	Dice	0.675980
NAW	16	Log	Jaccard	0.674917
NAW	16	Logistic	Dice	0.748661
NAW	16	Logistic	Jaccard	0.760910
NAW	16	None	Dice	0.670803
NAW	16	None	Jaccard	0.670918
NAW	18	Log	Dice	0.676348
NAW	18	Log	Jaccard	0.686562
NAW	18	Logistic	Dice	0.746411
NAW	18	Logistic	Jaccard	0.757685
NAW	18	None	Dice	0.669489

continued...

Feature Set	K	Scalar Normalization	Set Distance	Rand Index
NAW	18	None	Jaccard	0.669458
AW	10	N/A	Dice	0.929555
AW	10	N/A	Jaccard	0.855210
AW	12	N/A	Dice	0.870459
AW	12	N/A	Jaccard	0.864035
AW	14	N/A	Dice	0.851381
AW	14	N/A	Jaccard	0.842278
AW	16	N/A	Dice	0.850255
AW	16	N/A	Jaccard	0.844351
NA	8	Logistic	Dice	0.742567
NA	8	Logistic	Jaccard	0.770920
NA	10	Log	Dice	0.670728
NA	10	Log	Jaccard	0.671137
NA	10	Logistic	Dice	0.726959
NA	10	Logistic	Jaccard	0.737731
NA	10	None	Dice	0.667858
NA	10	None	Jaccard	0.667858
NA	12	Log	Dice	0.671808
NA	12	Log	Jaccard	0.673073
NA	12	Logistic	Dice	0.723300
NA	12	Logistic	Jaccard	0.729915
NA	12	None	Dice	0.669829
NA	12	None	Jaccard	0.669829
NA	14	Log	Dice	0.667990
NA	14	Log	Jaccard	0.669195
NA	14	Logistic	Dice	0.714399
NA	14	Logistic	Jaccard	0.722061
NA	14	None	Dice	0.670182
NA	14	None	Jaccard	0.670232
NA	16	Log	Dice	0.678210
NA	16	Log	Jaccard	0.676306
NA	16	Logistic	Dice	0.712950
NA	16	Logistic	Jaccard	0.726035
NA	16	None	Dice	0.670561
NA	16	None	Jaccard	0.670614
NA	18	Log	Dice	0.678099
NA	18	Log	Jaccard	0.676272
NA	18	Logistic	Dice	0.708987
NA	18	Logistic	Jaccard	0.719783
NA	18	None	Dice	0.669363
NA	18	None	Jaccard	0.669419
NW	10	Log	Dice	0.669719
NW	10	Log	Jaccard	0.684106
NW	10	Logistic	Dice	0.694825
NW	10	Logistic	Jaccard	0.669269
NW	10	None	Dice	0.667752
NW	10	None	Jaccard	0.667750
NW	12	Log	Dice	0.671185
NW	12	Log	Jaccard	0.684292
NW	12	Logistic	Dice	0.694171
NW	12	Logistic	Jaccard	0.674548
NW	12	None	Dice	0.669814
NW	12	None	Jaccard	0.669810
NW	14	Log	Dice	0.670793
NW	14	Log	Jaccard	0.677972
NW	14	Logistic	Dice	0.694405
NW	14	Logistic	Jaccard	0.675936
NW	14	None	Dice	0.670697
NW	14	None	Jaccard	0.670578
NW	16	Log	Dice	0.672472
NW	16	Log	Jaccard	0.676489
NW	16	Logistic	Dice	0.692137
NW	16	Logistic	Jaccard	0.681405
NW	16	None	Dice	0.670888
NW	16	None	Jaccard	0.670706
NW	18	Logistic	Jaccard	0.681717
A	10	N/A	Dice	0.907089
A	10	N/A	Jaccard	**0.969376**
A	12	N/A	Jaccard	0.910638
A	14	N/A	Dice	0.869441
A	14	N/A	Jaccard	0.911598
A	16	N/A	Dice	0.826885
A	16	N/A	Jaccard	0.907550
N	8	Log	N/A	0.657650
N	8	Logistic	N/A	0.644067
N	10	Log	N/A	0.657301
N	10	Logistic	N/A	0.649856
N	10	None	N/A	0.667856
N	12	Log	N/A	0.669908
N	12	Logistic	N/A	0.665688
N	12	None	N/A	0.669810
N	14	Log	N/A	0.673658
N	14	Logistic	N/A	0.661820
N	14	None	N/A	0.670462
N	16	Log	N/A	0.669040
N	16	Logistic	N/A	0.666306
N	16	None	N/A	0.670721
N	18	Log	N/A	0.669534
N	18	Logistic	N/A	0.661045
N	18	None	N/A	0.669127
W	10	N/A	Dice	0.852379
W	10	N/A	Jaccard	0.816539
W	12	N/A	Dice	0.860230
W	12	N/A	Jaccard	0.815057
W	14	N/A	Dice	0.859987
W	14	N/A	Jaccard	0.814819
W	16	N/A	Dice	0.819333
W	16	N/A	Jaccard	0.798225

continued...