Spring 2019

Building A Classification Model Using Affinity Propagation

Christopher R. Klecker

BUILDING A CLASSIFICATION MODEL USING AFFINITY PROPAGATION

by

CHRISTOPHER KLECKER

(Under the Direction of Ashraf Saad)

ABSTRACT

Regular classification of data includes a training set and test set. For example for Naïve Bayes, Artificial Neural Networks, and Support Vector Machines, each classifier employs the whole training set to train itself. This thesis will explore the possibility of using a condensed form of the training set in order to get a comparable classification accuracy. The technique explored in this thesis will use a clustering algorithm to explore which data records can be labeled as exemplar, or a quality of multiple records. For example, is it possible to compress say 50 records into one single record? Can a single record represent all 50 records and train a classifier similarly? This thesis aims to explore the idea of what can label a data record as exemplar, what are the concepts that extract the qualities of a dataset, and how to check the information gain of one set of compressed data over another set of compressed data. This thesis will explore using affinity propagation, categorical data, exploring entropy within cluster sets, and testing the compressed data using Cosine Similarity as a classifier.

INDEX WORDS: Classification, Clustering, Clustering analysis, Condensed dataset, Prediction model, Affinity propagation, Damping factor, Preference value, Categorical data, Elbow method

BUILDING A CLASSIFICATION MODEL USING AFFINITY PROPAGATION

by

CHRISTOPHER KLECKER

B.S., Purdue University, 1997

A Thesis Submitted to the Graduate Faculty of Georgia Southern University

in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

BUILDING A CLASSIFICATION MODEL USING AFFINITY PROPAGATION

by

CHRISTOPHER KLECKER

Electronic Version Approved:
May 2019

ACKNOWLEDGMENTS

I would like to thank Dr. Ashraf Saad for his direction and advisory role in the creation of my project and thesis. I would like to thank Dr. Ray Hashemi for his time and class instruction which aided in the theories used in this thesis. I would like to thank Jeffery Young, Brennen Lemieux, my friends, and family for their support and help in the building and writing of this thesis. I would like to thank Saint Vincent College, PA, and William Barnes, the assistant Director of Saint Vincent College for their resources in this thesis, particularly the student enrollment database which was used as an example to test the technique described in this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Data Mining and Clustering

Data Mining is the use of methodologies to discover patterns in a data set. Many of these methodologies are an amalgamation of approaches to machine learning and statistics[1]. Clustering is one such methodology which groups data into meaningful categories often times employing statistical models such as conditional probability or measuring the distance of actual data points to discover groups in physical space. Each cluster describes a similarity between the data points existing in the cluster. This can be desirable in the context of customer data to find similarities in customers' attributes. It can be used to discover basic qualities of customer groups and target those qualities. Clustering is an exploratory technique[2], which aims to discover an optimal grouping of data points. Clustering seeks what features of the dataset will drive the shape and number of the resulting clusters which can either be predetermined or discovered naturally. The overall goal is to optimize the groups of data points so that meaningful patterns emerge. For example, a particular pattern might be what are the majority values for each attribute in each cluster?

Most clustering algorithms work from a similarity matrix which defines a "distance" or relationship of records from each other using the features of the records. The features of the record correspond to the attribute values of each record. By analyzing two records in our dataset and performing a similarity function, a similarity value is returned. A higher similarity value indicates the two records are highly similar, whereas a lower similarity value indicates the two records are highly dissimilar. Each record is compared to all other records, except itself, and a similarity is calculated and stored resulting in a similarity matrix[3]. Features of

1 Frans Coenen, "Data Mining: Past, Present and Future," The Knowledge Engineering Review 26, no. 1 (2011): 25-29, doi:10.1017/s0269888910000378.

2 Vipin Kumar et al., "Cluster Analysis: Basic Concepts and Methods," in Introduction to Data Mining, 2nd ed. (New York, NY: Pearson Education, 2005).

3 Daniel Lawson and Daniel Falush, "Similarity Matrices and Clustering Algorithms for Population Identification Using Genetic Data," FineSTRUCTURE, March 1, 2012, , accessed April 15, 2019, https://people.maths.bris.ac.uk/~madjl/finestructure/.

records can be categorical, numerical, or both. In the case of numeric data similarities between records calculates their distance from each other using a distance equation such as Euclidian Squared Distance, for example. Furthermore the centroids of numerical data describe the overall location and shape of the data on a plane. Datasets containing categorical features must use alternate methods to calculate a distance as categorical features do not physically map to any dimension of a plane.

Clustering algorithms output a cluster set which contain the data points mapped to a particular cluster. Each cluster contains a centroid, a center of the cluster if the features of the data are numerical. For categorical features, referring to the centroid as the "center," is not applicable. The centroid of a cluster with categorical features is described by the features which represent the majority of features found in the cluster from the data assignments. Because this thesis works with categorical data and explores clustering and classification of categorical data, this thesis will concentrate on clustering algorithms that can handle categorical features and return clusters and centroids that describe the features of the cluster through the features of the centroid. These centroids may prove useful in summarizing a dataset, therefore acting as training data for a classification model.

Classification is the task of assigning objects to a predefined category or class[4]. Classification models require input data, or a collection of records that are sent to an indicator decision function which chooses the "best" decision based on some optimization method known as a prediction model[5]. Prediction models can range from numeric input to train an Artificial Neural Network, or from determining a potential feature of an attribute that is unknown. These are called predictive models. Predictive Models are used in data mining to analyze historical and current data in order to generate a model for predicting future outcomes[6].

[4] Vladimir S. Cherkassky and Filip Mulier, Learning from Data: Concepts, Theory, and Methods, 2nd ed. (Hoboken, NJ: IEEE Press/Wiley-Interscience, 2007).

[5] Ibid.

[6] Hal Kalechofsky, "HomeA Simple Framework for Building Predictive Models," M Squared Consulting, September 2016, , accessed April 15, 2019, http://www.msquared.com/.

The Purpose of This Study

The purpose of this study is to illustrate a new technique of taking a whole dataset, applying a clustering algorithm with respect to its class attribute to condense the data and apply a classification on this condensed data form. The algorithm will split the original dataset into two datasets with respect to the class features of the dataset. Each dataset is applied to a clustering algorithm which will output centroids and the labels for each centroid, that is, matching records to specific clusters. The centroids outputted from the clustering algorithm will be used as a classification model which describes the features of each class through these centroids. The features represent the most relevant records therefore creating a condensed dataset of just the relevant records. The prediction function then takes test records, records not included in the initial clustering algorithm, and are applied to each compressed dataset representing each class whereas the classifier will return a class using Cosine Similarity. This classification model takes a summary of the dataset and is still able to function as a classifier, in some cases, only 1% of the actual data is used. Figure 1 shows an illustration of the technique proposed for this thesis.

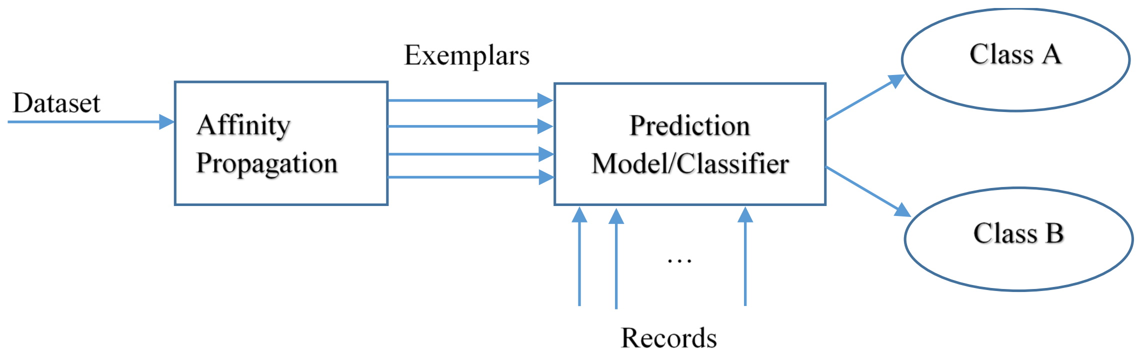


Figure 1: Graphical Representation of the Affinity Propagation Prediction Model

This thesis will explain the clustering algorithm used, affinity propagation, and will explain the technique used to construct and compute the similarity matrix for categorical features. It will discuss methods of calculating a preference value and damping factor to be used with affinity propagation, which will determine the number of clusters and the integrity of the clustered result. This thesis will discuss datasets that can work with this technique and datasets that are not applicable. It will discuss the prediction model, how it will predict new records and the classification feature of the new record. It is the goal of this thesis to have an accuracy of 90% or higher, especially for a classifier that is binary, or contains only two classes. A classifier which predicts around 50% for a binary classifier, or two classes, is no better than chance.

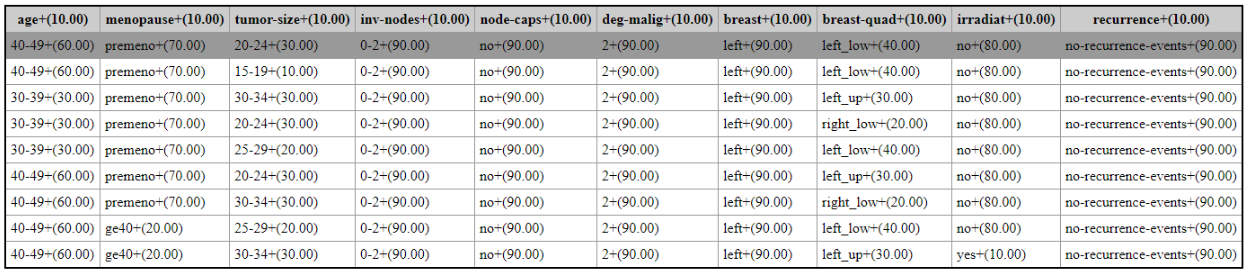
A few purposes of the creation of this classifier are to create a classifier that works with categorical data without having to perform a “One Hot Conversion”. One Hot is a technique to convert categorical data into a vector representation which is then used instead of the categorical name. The problem with this conversion is in retrieving the categorical values back. Also all records are converted to be a long vector string which is difficult for a human to understand the features of the record. This technique doesn’t alter the categorical values in any way. The centroids returned are actually the categorical strings themselves. Because of this feature, this technique provides a fully transparent classifier, and a fully transparent cluster set. The centroids which describe the features of the cluster set can be understood easily. This technique could also be used as a means to fill in missing attribute data. Currently this technique is written to classify only binary classifiers, however it can easily be augmented to have N classes. The centroids created for the classifier can also be saved and used later. It will also show how centroids created as a training set can exist for records that cover a whole year. That is, running this classifier against 2017 data, and outputting a training set, the very same training set can be used for 2018 with high accuracy results.

CHAPTER 2

CLUSTERING WITH AFFINITY PROPAGATION

What is Affinity Propagation?

Affinity propagation is a modern unsupervised clustering algorithm that "takes as input a collection of real-valued similarities between data points, where the similarity $s(i, k)$ indicates how well the data point k is suited to be the exemplar for data point i [7]. Affinity propagation can also be called "Exemplar Based Clustering" as it returns actual data points as exemplars over arbitrary averaged data values within the cluster itself. The output of affinity propagation therefore becomes a compressed version of the overall data where the exemplars represent data points that define the properties found in the assignments. This is shown in Figure 2.



age+(10.00)	menopause+(10.00)	tumor-size+(10.00)	inv-nodes+(10.00)	node-caps+(10.00)	deg-malig+(10.00)	breast+(10.00)	breast-quad+(10.00)	irradiat+(10.00)	recurrence+(10.00)
40-49+(60.00)	premeno+(70.00)	20-24+(30.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	left_low+(40.00)	no+(80.00)	no-recurrence-events+(90.00)
40-49+(60.00)	premeno+(70.00)	15-19+(10.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	left_low+(40.00)	no+(80.00)	no-recurrence-events+(90.00)
30-39+(30.00)	premeno+(70.00)	30-34+(30.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	left_up+(30.00)	no+(80.00)	no-recurrence-events+(90.00)
30-39+(30.00)	premeno+(70.00)	20-24+(30.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	right_low+(20.00)	no+(80.00)	no-recurrence-events+(90.00)
30-39+(30.00)	premeno+(70.00)	25-29+(20.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	left_low+(40.00)	no+(80.00)	no-recurrence-events+(90.00)
40-49+(60.00)	premeno+(70.00)	20-24+(30.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	left_up+(30.00)	no+(80.00)	no-recurrence-events+(90.00)
40-49+(60.00)	premeno+(70.00)	30-34+(30.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	right_low+(20.00)	no+(80.00)	no-recurrence-events+(90.00)
40-49+(60.00)	ge40+(20.00)	25-29+(20.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	left_low+(40.00)	no+(80.00)	no-recurrence-events+(90.00)
40-49+(60.00)	ge40+(20.00)	30-34+(30.00)	0-2+(90.00)	no+(90.00)	2+(90.00)	left+(90.00)	left_up+(30.00)	yes+(10.00)	no-recurrence-events+(90.00)

Figure 2: Cluster Example From affinity propagation

The exemplar is shown as the first record after the header. Notice how the features of the exemplar contain a majority of features in the respective attribute. Not all features in the attribute of the cluster are the same. For example, the attribute for "Breast-Quad" contains 3 different features. The value "left-low" appears in the attribute the most, therefore this value should be the feature value represented in the exemplar. The goal of affinity propagation is to discover these data points, or exemplars, which represent the whole of the dataset. It does this through finding the maximal value of the responsibility and availability of each data point by passing messages between each data point until a maximum availability and

[7] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," Science 315, no. 5814 (2007): , doi:10.1126/science.1136800.

responsibility is achieved. The responsibility of a data point is defined by a measure of how the data point is suited to be assigned to an exemplar, and the availability of a data point is the measurement of how the point is suited to be an exemplar[8]. This procedure is illustrated in Figure 3.

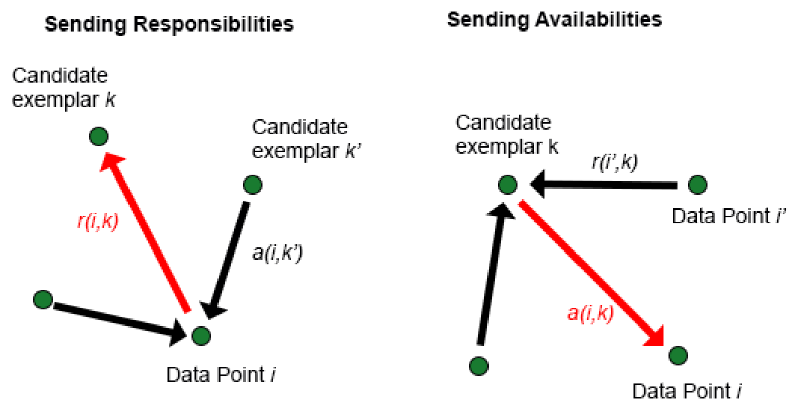


Figure 3: Message Passing in Affinity Propagation.

Affinity propagation runs iteratively through each data point passing messages and singling out exemplars until a convergence is achieved. This is when the availabilities and responsibilities for each data point no longer update. At each iteration exemplars will emerge, and clusters will form. When convergence is reached a set of exemplars is selected and the clusters take their final shape. Convergence of affinity propagation is shown in Figure 4 [9].

In affinity propagation clustering happens naturally. The algorithm does not take an initialization of K and outputs exactly K clusters for the dataset. The number of clusters is determined through a series of equations that update the responsibility and availability of each data point in reference to all other data points. There is a possibility that convergence will not happen. If this is the case, the number of clusters and data assignments are determined by when the algorithm terminates, not on any convergence. It is not necessarily true that the resultant clusters are incorrect, however because clustering did not happen

[8] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," Science 315, no. 5814 (2007): , doi:10.1126/science.1136800.
[9] ibid

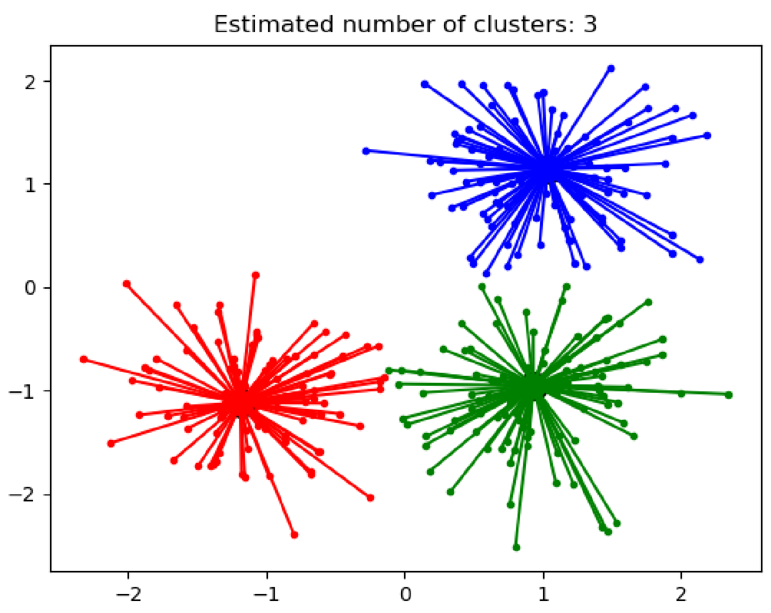


Figure 4: Convergence in Affinity Propagation.

naturally, the result will not be the same if affinity propagation is run again as the termination of the algorithm happens in the middle of data points switching between assignments and being or not being an exemplar to a cluster. Therefore, convergence is an important aspect to this technique and convergence will be sought when determining exemplars.

There are a couple of methods which can help with affinity propagation reach convergence: raise the damping factor or raise the number of iterations. The damping factor decreases the oscillations resulting from the message passing between data nodes. The damping factor smooths and normalizes the new values for each data point making convergence easier to achieve. Increasing the number iterations can also help if convergence does not happen even with a high damping factor, however increasing the iteration for a clustering result which did converge will not change the clustering result in any way. affinity propagation clusters naturally and will stop when convergence is reached. By increasing the number of iterations it only makes it more possible for convergence to be reached if the number of iterations necessary is high. If convergence is reached in 150 iterations and the max iterations is 200, raising the number of iterations will not matter in the clustering result as convergence happened at 150 iterations. However if the iterations were lowered below 150, this will cause the algorithm to not converge which will result in a premature clustering result. Therefore, although raising the number of iterations for affinity propagation will not affect a clustering result if it converges at a lower iteration point, lowering the number of iterations can prevent convergence from happening.

Why Use Affinity Propagation?

The classification function of this project contains exemplars from the cluster set for each class. It is important for this technique to have a set of centroids that are constant, that is, each time we run the clustering algorithm the same output is returned. If exemplar features are not constant it is difficult to be assured the exemplars used are a good representation of the dataset. In contrast, clustering algorithms like K-Means and BIRCH will select random data points which are initialized as centroid points, however each iteration will shift the centroid from its current position to its new position to represent it as being centroid to its data assignments. This results in a different clustering set each time such clustering algorithms are run[10]. Additionally, because centroids of clusters created by these algorithms like K-Means and BIRCH are most likely not actual records in our dataset, therefore additional work must be done to get the features of the centroids from surrounding data points in the cluster. Affinity propagation eliminates this step, returning exemplars, centroids which are actual data points, therefore no additional work is needed to extract the features of the centroid.

Accuracy is also very important in the clustering algorithm due to the centroids needing to represent as best as possible the data assignments of the cluster. Therefore choosing an algorithm that results in low error is preferable and affinity propagation will return errors that are lower compared with other clustering algorithms[11]. For these reasons, this thesis will focus on affinity propagation to create the clusters from the datasets generated and use its exemplars for our prediction model function.

[10] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," ACM Computing Surveys 31, no. 3 (1999): , doi:10.1145/331499.331504.

[11] John Trono, Dave Kronenberg, and Patrick Redmond, "Affinity Propagation, and Other Data Clustering Techniques," Affinity Propagation, and Other Data Clustering Techniques, , accessed April 15, 2019, http://academics.smcvt.edu/.

Algorithm for Affinity Propagation

Like most clustering algorithms, affinity propagation computes the clusters by using a similarity matrix the user constructs beforehand. The similarity matrix is symmetrical, an N x N matrix, where N is the number of records being clustered. Each value in the similarity matrix, (N_p, N_q), where p is *record p* and q is *record q*, is calculated by finding the distance between record p and record q. The construction of this similarity matrix will be discussed in the next section.

Affinity propagation generates two additional matrices during initialization called R for responsibilities, and A for availabilities. Each matrix is also symmetrical like the Similarity Matrix, and exactly the same size. Each index pair in each matrix is associated to each other. *R[i, j]* is associated with *A[i, j]* and *S[i, j]*. The diagonal represents self-availability for the availability matrix, and self-responsibility for the responsibility matrix, and the preference value for the dataset is stored in the diagonal of the similarity matrix.

Affinity propagation calculates responsibilities for each data point i and stores this in matrix R using the formula shown in Figure 5 [12]. This finds the maximum value of all data points from the availability matrix A at $A(i, k')$ where *k'* is not data point *k*, added to all $s(i, k')$ where *k'* is not data point

$$r(i,k) \leftarrow s(i,k) - \max_{k' s.t. k' \neq k} \{a(i,k') + s(i,k')\}$$

Figure 5: Equation to calculate Responsibilities.

k.

Accessing the similarity matrix value at matrix location (i, k), all other data points k' where k' is not k are accessed from the availability matrix and similarity matrix and added together. Messages do not get passed to themselves. We are looking for the maximum value returned for this summation with each message passed. This maximum value found is subtracted from the similarity matrix value $S(i, k)$. This

[12] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," Science 315, no. 5814 (2007): , doi:10.1126/science.1136800

value is then stored as the responsibility value at $R(i, k)$. We increment i to the next data point and the process is repeated until all data points have been calculated. The algorithm to calculate all responsibilities

Algorithm 1: UpdateResponsibilities for each data point i in Matrix R

```
 1. function UpdateResponsibilities(S, R, A)
    Input: Similarity Matrix S, Responsibility Matrix R, Availability Matrix A
    Output: Responsibility Matrix R
 2. dampingFactor = 0 > x > 1
 3. for each record i do
 4.   for each record j do
 5.      for each record k that is not j do
 6.        value = A[i][k] + S[i][k];
 7.        if value is maximum then
 8.          max = value
 9.        R[i][k] = R[i][k] + dampingFactor + (1 - dampingFactor) * S[i][j] - max);
10. return R
```

Figure 6: Algorithm to Update Responsibility Matrix

for each data point *i* with respect to data point k is shown in Figure 6 [13].

Algorithm 1 shows the usage of the damping factor variable which exists to limit oscillations during this process of updating Responsibilities which can help the algorithm converge more efficiently. The authors of affinity propagation recommend a damping factor of .5. For situations where convergence does not occur with a damping factor of .5, raise the damping factor until convergence occurs. The highest damping factor can be .9.

Once all responsibilities are calculated for matrix R, availabilities for matrix A are calculated for each data point. The availability equations for affinity propagation are shown in Figure 7 and 8 [14].

$$a(i, k) \leftarrow \; min\{0, r(k, k) + \sum_{i' s.t. i' \notin \{i,k\}} \max\{0, r(i', k)\}\}$$

Figure 7: Equation for calculating availability of data record i.

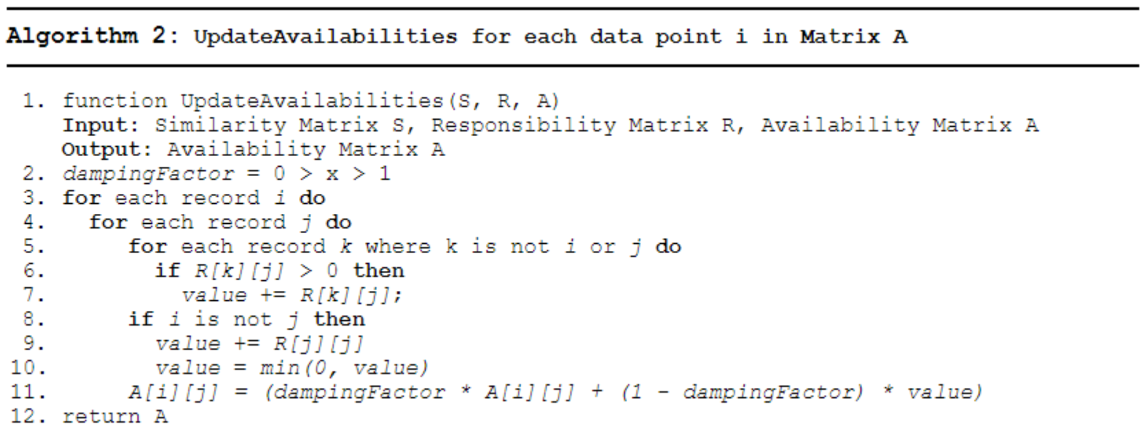
[13] "Affinity Propagation - Java," Cognitive Foundry, , accessed April 15, 2019, https://foundry.sandia.gov/releases/latest/javadoc-api/index.html.

[14] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," Science 315, no. 5814 (2007): , doi:10.1126/science.1136800

$$a(k,k) \leftarrow \sum_{i' s.t.\ i' \neq k} \max\{0, r(i',k)\}$$

Figure 8: Equation for self-availability of data record k.

The summation of all maximum positive values for all responsibilities of $r(i',k)$ where *i'* are all data points not *i* (or 0, whichever value is greatest) is assigned to the self-availability, the diagonal of the availability matrix. This value is also added to the self-responsibility, the diagonal of the responsibility matrix. The availably of point *i* with respect to point *k* is assigned the minimum value of the summation or 0, whichever is smaller. All points *i* are iterated through until all availabilities and self-availabilities are calculated. The algorithm for finding availabilities for all data points *i* with respect to data point *k* is shown in Figure 9 [15].



Algorithm 2: UpdateAvailabilities for each data point i in Matrix A

```
function UpdateAvailabilities(S, R, A)
Input: Similarity Matrix S, Responsibility Matrix R, Availability Matrix A
Output: Availability Matrix A
dampingFactor = 0 > x > 1
for each record i do
  for each record j do
     for each record k where k is not i or j do
       if R[k][j] > 0 then
         value += R[k][j];
     if i is not j then
       value += R[j][j]
       value = min(0, value)
     A[i][j] = (dampingFactor * A[i][j] + (1 - dampingFactor) * value)
return A
```

Figure 9: Algorithm for Updating Availability Matrix

The total likelihood that data point *i* is assigned then to data point *k* is the sum of the self-availability and self-responsibility values. If the value is not zero, data point *k* can be assigned to data point *i*. At each iteration several exemplars start to emerge. affinity propagation continues to iterate and update responsibilities and availabilities until convergence occurs which results in a final set of exemplars. Assignments are made by checking each matching matrix field in availability and responsibility. The sum

[15] "Affinity Propagation - Java," Cognitive Foundry, , accessed April 15, 2019, https://foundry.sandia.gov/releases/latest/javadoc-api/index.html.

of the availability and responsibility therefore determine if record *i* is assigned to exemplar record *j*. The algorithm for updating data point *i* to cluster *k* is shown in Figure 10 [16].

Algorithm 3: UpdateAssignments for each data point i

```
 1. function UpdateAssignments (R, A);
    Input Responsibility Matrix R, and Availability Matrix A
    Output none
 2. Initialize sum to 0.0
 3. for each record i
 4.   Initialize assignment to -1
 5.   for each record j
 6.     value = A[i][j] + R[i][j]
 7.     if assignment < 0 or value > maximum
 8.       assignment = j
 9.       maximum = value
10. Assign value assignment to cluster i
```

Figure 10: Algorithm for Updating Assignment Record j to Cluster record i

The time complexity for affinity propagation requires $O(k*n^2)$[17] where k is the number of iterations and n is the number of data points. This time complexity is important as data which contains a large number of records can take a substantial amount of time to converge. Furthermore, construction of a similarity matrix for large dataset will process slowly.

A Similarity Matrix for Categorical Data

The similarity matrix holds a measurement of similarity or dissimilarity between records in an N x N matrix where N is the number of records in the dataset. Datasets with numeric features are quite simple to calculate similarity as such records can be plotted in N-Dimensional space and the physical distance between records can be the measurement of their similarity. This distance can be measured using such equations like Euclidian Squared Distance or Manhattan Distance[18] to name a few. Categorical data, data

[16] 16 "Affinity Propagation - Java," Cognitive Foundry, , accessed April 15, 2019, https://foundry.sandia.gov/releases/latest/javadoc-api/index.html.

[17] R. Refianti, A.B. Mutiara, and S. Gunawan, "Time Complexity Comparison Between Affinity Propagation Algorithms," Journal of Theoretical and Applied Information Technology 95, no. 7 (April 15, 2007):

[18] Paul Barrett, "Euclidean Distance Raw, Normalized, and Double-scaled Coefficients," The Technical Whitepaper Series, September 2005, , accessed April 15, 2019, https://www.pbarrett.net/techpapers/euclid.pdf.

whose features are categories or strings, presents a challenge in that categorical values cannot be represented as existing in some physical space like numeric data. For example, expressing the distance between Male and Female cannot be achieved by plotting Male and Female onto a plane. However, if we recognize that categorical data is dataset dependent, that is the dataset used will contain only the categories included in each attribute, it may be possible to think of a distance between two categorical values with respect to all categories in the dataset.

This approach, written by Amir Ahmad and Lipika Dey titled, “A K-Mean Clustering Algorithm for Mixed Numeric and Categorical Data”[19], uses a conditional probability approach to find a similarity measurement between two categorical values x and y in Attribute i. The algorithm calculates the co-occurrence of values x and y in Ai with all other categorical values in Attributes j. where j does not equal i. For example, I have three attributes where Attribute A has the following categories: Male, Female, and Undeclared, Attribute B has the following categories, Yes, and No, and Attribute C has the following categories, House, Apartment, and Condo. To find the distance between Male and Female, I find the conditional probability of Yes, given Male, and Yes given Female. Then I find the conditional probability of No, given Male, and No given Female, and continue this process until all conditional probabilities are computed for all attribute features. This measurement of co-occurrence is calculated by finding the maximum of the conditional probability of X given Aj(Category) and Y given Aj(Category). We first define the distance between values x and y of attribute I with respect to Attribute j shown in Figure 11 [20].

Definition 1: $\delta^{ij}(x, y) = P_i(\omega \mid x) + P_i(\sim\omega \mid y)$

Figure 11: Distance Between Attribute values x and y for Ai with respect to Attribute Aj.

where w is a subset of values and ~w is the complement of w, of Aj which maximizes the value of $P_i(\omega \mid x) + P_i(\sim\omega \mid y)$. For example, for x = Male, and y = Female, the distance between Male and Female

[19] Amir Ahmad and Lipika Dey, "A K-mean Clustering Algorithm for Mixed Numeric and Categorical Data," Data & Knowledge Engineering 63, no. 2 (2007): , doi:10.1016/j.datak.2007.03.016.

[20] ibid

requires the conditional probability of all categories in Attributes j ≠ i. If the categories in Attribute j are Yes, and No, then, $\delta^{ij}(Male, Female) = max(P_1(Yes|Male), P_1(Yes|Female)) + max(P_1(No|Male), P_1(No|Female))$.

Definition 2 [21], shown in Figure 12, expands the equation of finding the maximum of the conditional probabilities of our first definition to find the maximum of the conditional probabilities for all attribute features. The total of all maximum conditional probability values is then divided by the number of attributes – 1. This is the "distance" for this categorical pair. For example, if we expanded our first example to find the maximum probability of all attribute features and the number of attributes is 10, then $\delta(Male, Female) = \left(\frac{1}{10-1}\right)\sum_{j=1\ldots10, i\neq j}\delta^{ij}(Male, Female)$, where j represents the attribute other than Attribute I.

Definition 2: $\delta(x, y) = \left(\frac{1}{m-1}\right)\sum_{j=1\ldots m, i\neq j}\delta^{ij}(x, y)$

Figure 12: Equation for distance between values x and y in any given attribute Ai.

Because this technique will not use numeric attributes, definition 3 of Ahmad and Lipika's paper is altered to not include numeric attributes. Therefore, definition 3 [22] is rewritten as the sum of all distance pairs between two categories as the similarity of two records: D1, and D2. This equation is shown in Figure 13.

Definition 3: $Dist(D1, D2) = \sum_{t=1}^{m}(\delta(X_t, Y_t))^2$

Figure 13: Equation for the distance between record D1 and D2

where m represents all categorical attributes. If data contains numeric values for attributes, this thesis will convert the attribute into bins, where the bin name will represent the categorical value for the binned data itself. How binning will be performed on the attribute is task dependent and must be done as to not over simplify the distribution of the values in the attribute. The reason for eliminating the process of converting

[21] Amir Ahmad and Lipika Dey, "A K-mean Clustering Algorithm for Mixed Numeric and Categorical Data," Data & Knowledge Engineering 63, no. 2 (2007): , doi:10.1016/j.datak.2007.03.016
[22] ibid

numeric values using Ahmad and Dey's approach is that the technique did not perform well with any clustering algorithm, especially affinity propagation. It created clusters with large errors. In order to simply the process, this thesis eliminates numeric attributes by binning them manually.

Preference Value

The preference value is the priori suitability of a point *i* to serve as an exemplar[23]. The Preference value is set to the diagonal of the similarity matrix. Data points that maximize the sum of the availability and responsibility and exceed this preference value are defined as an exemplar. Preference values that are too low will produce many clusters, as most if not all data points are going to exceed a low preference value. Preferences that are high in value produce low number of clusters. Because of this the authors of affinity propagation suggest the preference value be set around the median value found in the similarity matrix[24]. In reference to the classification technique affinity propagation raises the question of how accurate are the clusters returned, and will the resultant exemplars serve the purpose of producing a high enough accuracy with as few clusters as possible?

This is the challenge to affinity propagation raising the importance of finding an ideal preference value. If there are too many centroids then the compression of the dataset is minimal. If there are too few centroids then the compression of the dataset is too extreme. Therefore it is necessary to isolate the preference value and find an ideal setting.

Isolating this preference value can be achieved in a few ways. One way is to analyze the data to see if the clusters contain good patterns and meaning. If not, the preference is changed, forming a new cluster set until meaning is achieved. For small data this could be plausible, yet for large data, analysis in this fashion is not practical. Another method automates this process by calculating information gain in the

[23] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," Science 315, no. 5814 (2007): , doi:10.1126/science.1136800

[24] ibid

resultant cluster set with respect to another resultant cluster set from a different preference value. This method is called The Elbow Method.

Evaluating a preference value using the Elbow Method

The Elbow Method calculates the variance of change. The challenge is to find something within affinity propagation which can represent change between different cluster results. One element of change can be explained as a function of the number of clusters with respect to the error of the clusters. The elbow method can then use entropy to ascertain a method of information gain to choose a number of clusters so that adding more clusters does not give a better model of the data[25]. Entropy characterizes the amount of information found within our cluster result, whereas information gain is the amount of information gained in the addition of another process. One way to perceive information gain from a clustered result is to analyze the average error in the clustered dataset. When any clustering algorithm creates a set of resultant clusters, there is always error in the clusters, represented by the distance of the data point in the cluster from its centroid. In the case of categorical values this will be the value of the similarity matrix $\mathrm{M(i, k)}$ where i is the data point and k is the exemplar. The entropy from one clustered result to another is the difference of the average error between each cluster result. If the error delta is small, the entropy is small and thus the information gain is small. We want to maximize the information gain by finding where the error has a significant delta. This might not be the largest delta if a specific run of affinity propagation did not result in convergence. Clustering a dataset without convergence can lead to distortion in the cluster set and thus effect the error. For example, if affinity propagation does not converge for preference value -2, but does for preference value 1, a cluster set result may have an error much higher or lower than 1. It can result in a spike or valley in the resultant line graph that will distort our measurements. If this happens, this method will ignore that preference value as convergence is desirable in affinity propagation. The Elbow method will show the entropy of each clustered result to discover the cluster set which will have the greatest

[25] Purnima Bholowalia and Arvind Kumar, "EBK-Means: A Clustering Technique Based on Elbow Method and K-Means in WSN," International Journal of Computer Applications 105, no. 9 (November 2014): , https://pdfs.semanticscholar.org/5771/aa21b2e151f3d93ba0a5f12d023a0bfcf28b.pdf.

information gain. From this clustered result a preference value should be related. It is the ideal preference value.

An example can give some insight to this method. Let us say we have a similarity matrix defined from categorical data using the technique of categorical difference pairs above and now we desire to cluster this dataset. Taking the median of the similarity matrix and expanding the value for a range of about 10 to 15 preference values, we can run affinity propagation for each preference value of the dataset. For each run of affinity propagation, we calculate the average error of the total clustered dataset as well as store the number of clusters and the preference value which produced this number of clusters.

An example illustration is shown in Figure 14. In this example the x-axis represents the number of clusters which represents a preference value range of -8 to 5 and y-axis represents the error the cluster result set produced. Each value of the x-axis represents affinity propagation run with a specified preference value which returns the number of clusters on the x-axis. The line chart shows, starting with the lowest number of clusters, the error is large and eventually as the cluster number increases the delta in the error decreases. From cluster number 74 to cluster number 241 we see very little change in error, thus a low entropy from adding more clusters starting with a result of 74 clusters. From the line graph we can show a “Goldilocks Zone” where the entropy of error is the highest compared to all clustering results. This zone represents affinity propagation converging at 17, 22, and 74 clusters. It would be up to the individual to decide which clustering amount is preferable to the problem at hand. That is, a user might feel 74 clusters is too many, therefore selecting 22 or 17 as preferable. The decision of what preference value to use is task dependent. If our clustered result is to be used for a prediction model, using 74 exemplars might be acceptable. In the case of a predictive model, it might be beneficial to try all three convergence results in our graph and calculate accuracy.

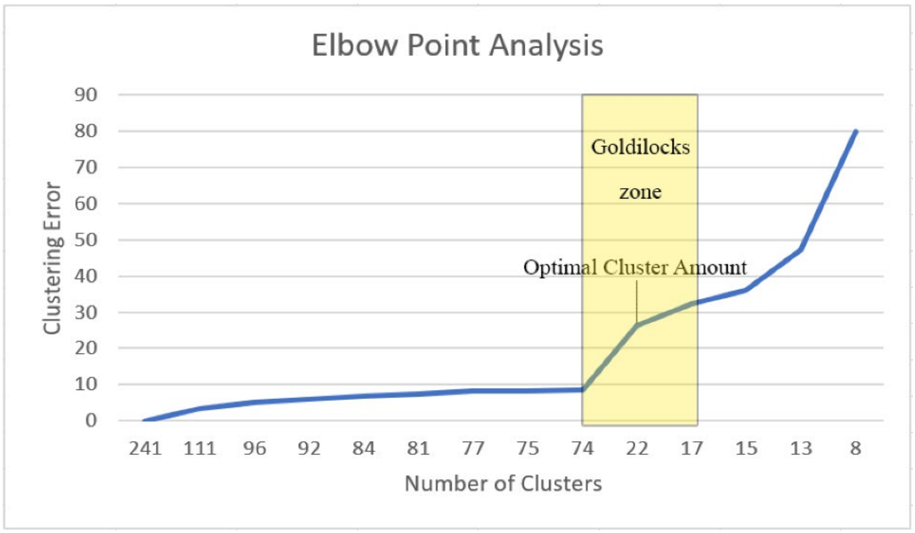


Figure 15: Elbow Method analysis of 3400 records of Data

Running affinity propagation 10 to 15 times for any given dataset is tedious, thus it would be beneficial if there was some way to automate it. By using the Law of Cosines, we can design an algorithm that runs AP between a range of preference values and measures the change of error by measuring the angle at a point in the line. The equation of the law of Cosines and its application to our graph is shown in Figure 15.

$$a^2 = b^2 + c^2 - 2bc\cos A$$

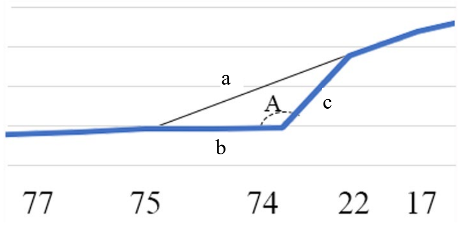


Figure 14: Application of Law of Cosines to measure entropy between cluster sets

Whichever point produces an angle that is small we save for consideration as smaller angles represent high entropy. We can then analyze each point where the elbow measurement was significant, and we can get a preference value that will produce for us the number of clusters at that elbow point. This technique will be used to get the preference value when running affinity propagation on any dataset.

Summary of Technique

The entire technique described in this chapter can be summarized as our training function for our classifier. That is, the process of training our classifier uses affinity propagation to get exemplar data from datasets. The entire process is shown in the algorithm in Figure 16.

Algorithm: Training A Classification Model Using Affinity Propagation (binary classifier)

```
input δ:Path to File, ε: Attribute containing classifiers, γ: Training/Test Data Fraction
output A Set of Arrays which hold the centroids (exemplars) for each clustered dataset with respect to the classifiers
FileCSV = Read_CSV_File(δ)
Train, Test = Split_Train_Test(FileCSV, γ)
Class = Split_On_Classifiers(Train, ε)
foreach item cls in Class
    SimiliarityMatrix = Create_Similarity_Matrix(cls)
    append SimiliarityMatrix to SimMatrix_Array
foreach sim in SimMatrix_Array
    Pref = Find_Preference_From_Elbow_Method(sim)
    append Pref to Pref_Array
foreach i in SimMatrix_Array
    Exemplars = Run_Affinity_Propagation(SimMatrix_Array[i], Pref[i], DampingFactor = .9)
    append Exemplars to Exemplars_Array
return Exemplars_Array
```

Figure 16: Algorithm: Training for classification using affinity propagation (binary classifier)

The algorithm trains the data by loading the dataset which is in CSV format, comma delimited, then splits the data into two sets: a training set and test set. It sets the test set aside and splits the training data set again into two datasets (or more datasets) which reflect the classes we are matching against. It takes each class dataset and constructs a similarity matrix for each. Each similarity matrix is sent to a method to calculate the ideal preference value using the elbow method and returns that preference value for that similarity matrix. affinity propagation is then run against each similarity matrix with the preference paired to that similarity matrix. affinity propagation returns the exemplars we are after. We save the exemplars from each class to a file which we will use for our testing. This file serves as the memory for our classifier as this thesis will show exemplars from earlier training sets work on datasets in the future. These files could be expanded as well with new exemplars added and others removed based on newer models constructed.

CHAPTER 3

CLASSIFICATION MODEL AND PREDICTION FUNCTION

Classification Using Affinity Propagation Exemplars

The classification process is quite simple as the training process has already been completed. The training data represents exemplar data for each class. This exemplar data describes potential clusters which exist for that class. The classifier will take each record and determine the probability of match for each class. Whatever class returns the highest probability of match, becomes the class. This is novel compared to Naïve Bayes, and Decision Trees which use the whole dataset to calculate conditional probabilities for records belonging to a particular class then uses the same process to calculate the test data. A great feature contained in Naïve Bayes and Decision Tree classifiers is the total transparency of the classification function. For businesses it is important to not only know to what class a customer may belong, but the features of the customer that placed it into the class. Classifiers like Artificial Neural Networks, and Support Vector Machines have high accuracy, but very little transparency when it comes to how the classes are determined. In the case of ANNs and SVMs the transparency is analogous to a black box, there is no way to determine practically how the classification was made, other than to trust the computations that performed the classification. Full transparency is sought in this thesis as it provides practical knowledge of what constitutes the qualities of being defined in a particular class, and the output of the clustering result set provides full transparency of the qualities which make up each feature set.

The basic idea behind the classifier is to calculate a probability of match for each class and classify the record based on which class returns the highest probability of match. The classification process uses an algorithm known as Cosine Similarity. The equation for Cosine Similarity is shown in Figure 17 [26].

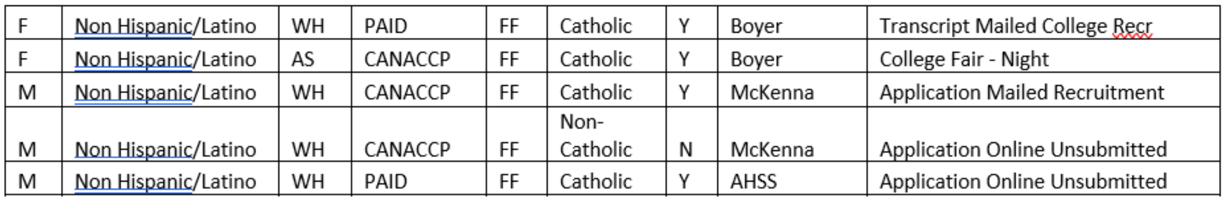
$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Figure 17: Cosine Similarity Equation

[26] Edel Garcia, "Cosine Similarity Tutorial," Minerazzi, April 10, 2005, , accessed April 15, 2019, http://www.minerazzi.com/tutorials/cosine-similarity-tutorial.pdf.

Cosine Similarity uses vector dot products which would mean having to convert each record into a vector representation, however because of how we can represent the categorical data, the following process produces the exact same result as Cosine Similarity every time. The process is as follows.

Figures 18 shows one class set for “Applied”. Another class set exists for all records “No Decision”. Figure 19 shows a test record we want to test against each class set.



F	Non Hispanic/Latino	WH	PAID	FF	Catholic	Y	Boyer	Transcript Mailed College Recr
F	Non Hispanic/Latino	AS	CANACCP	FF	Catholic	Y	Boyer	College Fair - Night
M	Non Hispanic/Latino	WH	CANACCP	FF	Catholic	Y	McKenna	Application Mailed Recruitment
M	Non Hispanic/Latino	WH	CANACCP	FF	Non-Catholic	N	McKenna	Application Online Unsubmitted
M	Non Hispanic/Latino	WH	PAID	FF	Catholic	Y	AHSS	Application Online Unsubmitted

Figure 18: Example of exemplars from affinity propagation Clustering

F	Non Hispanic/Latino	WH	PAID	FF	Catholic	Y	Boyer	Transcript Mailed College Recr

Figure 19: Record to test classifier

If we check each attribute of the test record against each attribute of the first record we see that 6 of the 9 attributes match, giving us 9/9 for the probability of match. The second record returns 6/9 for probability of match, and so on. If we were to use Cosine Similarity we would have to convert the test record to a vector. One method of conversion is called “One Hot Conversion”. This converts features of an attribute to a binary vector representation where one of the binary values is toggled as 1. For example, the first attribute contains 2 features, therefore the binary representation contains two bits: male = 01, and female = 10. One of the bits is toggled to 1. Attribute 3 contains 6 features, therefore the binary representation contains 6 bits: white = 100000, black = 010000, Asian = 001000, and so on. If this conversion extends the length of the record for each attribute, a vector representation will emerge containing only 1s and 0s. The number of 1s in the vector will always be the number of attributes in the record according to One Hot Conversion.

Taking the equation in Figure 17, the numerator is a dot product of the test record vector and the record in the training set we are matching against. If the vector representation of the test record is

(A) 00100000 100 01000 10000 01 001 010 0010 01

The spacing clearly represents the delineation between the attribute features. Normally this vector is all one single binary value. The first training record is represented as

(B) 00100000 100 01000 10000 01 001 010 0010 01

The dot product is $A \cdot B$ which takes each A_i and B_i and multiplies them together. If either vector value is zero the result is zero, therefore only the vector values where both are 1 remain. In this example we will have the following result from the dot product of A and B.

001000001000100010000010010100010001

The denominator is the magnitude of the vector A multiplied by the magnitude of the vector B. The magnitude is measured by summing the number of 1s in our vector. For A and B they will always be the same because each vector is a representation of each attribute feature. Each record contains the same number of attribute features; thus the magnitude of A and B are always the same. Thus plugging our values into the equation for Figure 17 we have the following,

$$\frac{9}{\sqrt{9}\sqrt{9}} = \frac{9}{3 * 3} = \frac{9}{9} = 1$$

This is the same value we got from our initial matching algorithm. Applying this method to the second training record we get the following,

$$\frac{6}{\sqrt{9}\sqrt{9}} = \frac{6}{9} = .6667$$

This application of Cosine Similarity can be done for each training record. A mean must be taken so as to normalize the value in case either class set is not equal or contains more records than the other. The class set which returns a higher probability of match will be the class the record will be classified as.

Algorithm of Technique

The technique proposed in this thesis can be represented through the following algorithm shown in Figure 19: The testing function takes each record and looks for a class of exemplars that returns the highest probability of match. This exemplar class becomes the prediction class and this is compared to the actual class the record belongs. It is important to realize the testing data was not a part of the training set and was therefore not a part of the clustering algorithm. It is also worth noting that the exemplars saved during the process of creation, the exemplars which are compared with the test records according to the algorithm above, are saved to a file. This file can be edited by removing exemplars that are not "important" and adding new exemplars if a new dataset is received. For example if you clustered a dataset for 2017 then received a new dataset for 2018, it is possible to combine the datasets together. Once a training set is computed, it is not required to run a clustering of the dataset again. All new records can be compared to a new dataset. This thesis will show how testing a new dataset on a previous year's dataset has highly favorable results.

Algorithm: Classification_Model (binary classifier)

```
Input: α: Test_Database, βExemplar_Classes
Output: Results of Classification of Records
Make sure Test_Database has classifiers as last attribute
1  foreach record in α
2      FalsePositive,Positive,Negative,TrueNegative = 0
3      ProbNo = 0.0
4      ProbYes = 0.0
5      foreach k in β[No] //Negative class
6          ProbNo = ProbNo + CompareRecords(record, β[No][k])//how many attributes match / number of attributes
7      foreach k in β[Yes] //Positive class
8          ProbYes = ProbYes + CompareRecords(record, β[Yes][k])
9      ProbYes = ProbYes / length of β[No]
10      ProbYes = ProbYes / length of β[No]

11      if MaxProbYes is higher and classifier is Yes then increment Positive
12      if MaxProbYes is higher and classifier is No then increment TrueNegative
13      if MaxProbNo is higher and classifier is No then increment Negative
14      if MaxProbNo is higher and classifier is Yes then increment FalsePositive
15      if MaxProbNo = MaxProbYes and classifier is Yes then increment Positive//rare
16      if MaxProbNo = MaxProbNo and classifier is No then increment Negative//rare
17  end
18  print results
```

Figure 20: Algorithm for Classification Model (for binary classifiers only)

CHAPTER 4

ILLUSTRATING OUR PREDICTION MODEL ON VARIOUS DATASETS

Breast Cancer

To help illustrate this algorithm and technique this thesis will use four datasets: three from UCI data repository and a real world dataset containing student enrollment data from a university of 4000 students. The first dataset this thesis will explore is a dataset for Breast Cancer. Taken from the UCI Repository, the dataset contains 286 records, 9 attributes and all data records are categorical[27].

The results using this technique, shown in table 1, were not favorable.

BREAST CANCER DATASET #1

	AP Classification Model
PRECISION	28%
RECALL	19%
ACCURACY	61%
FSCORE	23%

Table 1: Breast Cancer Dataset #1 Results classified with AP Classification

A reason for the unfavorable results is most likely due to the lack of data and therefore the lack of exemplars to represent the data. Clustering algorithms need a good representation of data to develop the clusters that help explain the data. If there isn't enough information, the clusters become sparse and the error will be significantly higher. The overall accuracy of the clusters for this training set averaged around 70% which is not very good, but okay.

To compare these results with another classifier I have chosen Naïve Bayes due to its similarities and its ease of use with categorical data. I also compared the results to using Cosine Similarity on the entire set, which therefore compares the use of the same classifier with two different training sets: a whole training

[27] Milan Soklic and Matjaz Zwitter, "Breast Cancer Data Set," UCI Machine Learning Repository, July 11, 1988, , accessed April 15, 2019, https://archive.ics.uci.edu/ml/datasets/Breast Cancer.

set and a compressed training set. Naïve Bayes did fare better at 61%, shown in Table 2, thus Naïve Bayes appears stronger with fewer records on a potentially rich dataset. Cosine Similarity also returned very good results from using the whole dataset. The lesson from this dataset shows that the prediction function fails when the training data is too small. When we have a richer feature set and more records our prediction function improves dramatically..

BREAST CANCER DATASET #1

	Naïve Bayes	**Cosine Similarity**
PRECISION	28%	42%
RECALL	19%	55%
ACCURACY	61%	72%
FSCORE	23%	48%

Table 2: Breast Cancer Dataset Results Classified with Naive Bayes and Cosine Similarity

Breast Cancer Dataset #2

A second dataset for breast cancer exists based on a Wisconsin study. The dataset contains 699 records, 10 attributes, and a binary classifier for benign or malignant. Created by Dr. William H. Wolberg from the university of Wisconsin, Madison in 1992[28], the results of the technique, shown in table 3, on this dataset are far more favorable.

BREAST CANCER DATASET #2

	AP Classification Model
PRECISION	86%
RECALL	98%
ACCURACY	88%
FSCORE	92%

Table 3: Breast Cancer Dataset #2 Results with AP Classification

[28] William H. Wolberg, "Breast Cancer Wisconsin (Original) Data Set," UCI Machine Learning Repository, July 15, 1992, , accessed April 15, 2019, https://archive.ics.uci.edu/ml/datasets/Breast Cancer Wisconsin (Original).

This dataset shows with enough records and features, it will improve the accuracy of the classifier. This goes without saying as any dataset will need enough data for any classifier to have a higher accuracy. Also a dataset where the features are well represented helps to increase not only accuracy, but a well-defined cluster set. Running the same dataset on Naïve Bayes produced results shown in Table 4. Naïve Bayes fared well with its prediction model, however our classification model uses less data and results in a higher accuracy.

What is more interesting are the results for cosine similarity. Comparing these results to our classifier the measurements are comparable. The difference between the two classifications is that the AP Classification training set used 37% of the original dataset used by this run of Cosine Similarity. Although my accuracy for AP Classification is 3% lower, the fact the numbers are so close shows potential.

BREAST CANCER DATASET #2

	Naïve Bayes	**Cosine Similarity**
PRECISION	83%	88%
RECALL	94%	99%
ACCURACY	84%	91%
FSCORE	88%	93%

Table 4: Breast Cancer Dataset #2 Results with Naive Bayes Classification

Mushroom Dataset

The next dataset contains information about various mushrooms also from the UCI Machine Learning Repository. It is a dataset which contains 8124 records, 22 attributes, with all data stored as categorical values. The records themselves are drawn from the Audubon Society Field Guide to North

American Mushrooms[29]. The classification is binary: poisonous or edible. The classification technique presented in this thesis provided the following results shown in Table 5.

MUSHROOM DATASET

	AP Classification Model
PRECISION	85%
RECALL	97%
ACCURACY	90%
FSCORE	91%

Table 5: Mushroom Dataset Results using AP Classifier

An accuracy of 90% is excellent indeed, more so recall of 97% demonstrates low False Positive results from this analysis which are less desirable than True Negatives. Also analysis of the data shows a perfect 50% split in the classification data. Further analysis into the output of the algorithm itself showed the number of exemplars produced for each classification was around 140. The training set is roughly 5686 records with roughly 50% of the records in each classification. This means the exemplars themselves represent about 5% of the original data, with the test set represent 2500 records. As the classification shows, having a complete training set where the exemplars indeed represent the data assignments well, we can build a prediction model that uses a fraction of the data, and still classify the data well, and in some cases perfectly.

[29] Jeff Schlimmer, "Mushroom Data Set," UCI Machine Learning Repository, April 27, 1987, , accessed April 15, 2019, https://archive.ics.uci.edu/ml/datasets/Mushroom.

Applying the same data to Naïve Bayes and Cosine Similarity, the results are shown in table 6.

MUSHROOM SET

	Naïve Bayes	**Cosine Similarity**
PRECISION	56%	83%
RECALL	93%	96%
ACCURACY	59%	88%
FSCORE	70%	89%

Table 6: Mushroom Dataset Results Using Naive Bayes Classification

From this analysis we see that our AP classifier, which is using only 5% of the original training set, is now out performing Cosine Similarity which uses the whole training dataset. Let's do one more test with student enrollment data.

Student Enrollment Data

A final dataset comes from a private institution, Saint Vincent College in Latrobe Pennsylvania. The records consist of around 19000 enrollment records with 15 attributes. The data was truncated to 8000 records as 17000 out of the 19000 records are related to a single class feature. This would have biased the data greatly. Therefore the data was truncated so the class distribution is 25% for "Applied" and 75% for "No Decision". Removing more records with the Attribute feature "No Decision" in my mind might corrupt the data. The idea of this prediction is to decide if a student who fills out an inquiry form will formally apply to the school. The results of this test are shown in table 7.

STUDENT ENROLLMENT DATASET

	AP Classification Model
PRECISION	98%
RECALL	77%
ACCURACY	94%
FSCORE	86%

Table 7: Student Enrollment Dataset Results using affinity propagation Classification

The results in testing this same dataset using Naïve Bayes and Cosine Similarity on the whole training set are shown in table 8.

STUDENT ENROLLMENT DATASET

	Naïve Bayes	**Cosine Similarity**
PRECISION		97%
RECALL		65%
ACCURACY	77%	90%
FSCORE		78%

Table 8: Student Enrollment Dataset Results Using Naive Bayes classification

The AP Classifier is using only 1% of the training set used by Cosine Similarity in the results shown in Table 8. The AP classifier is again out performing a classification of the whole dataset.

One additional test is to use the training set saved from the student enrollment cluster set from 2017 and apply a dataset from 2018 to see how accurate future records can be classified using older exemplars. The one unique feature of this technique is that it saves the training data in a file. This file can be edited in order to improve the quality of the training set. The training file is initialized with a single data set at first, however it is possible to run another dataset through affinity propagation to compile another set of exemplars and include these exemplars into the training set. The following test run does not add or alter the training set from 2017 in any way and is run against the entirety of 2018 dataset containing the same attributes, features, and about 19000 records. The results of this run are shown in table 9.

STUDENT ENROLLMENT DATASET

	AP Classification Model
PRECISION	97%
RECALL	74%
ACCURACY	93%
FSCORE	84%

Table 9: Student Enrollment Dataset from 2018 applied as Test data against 2017 Training set

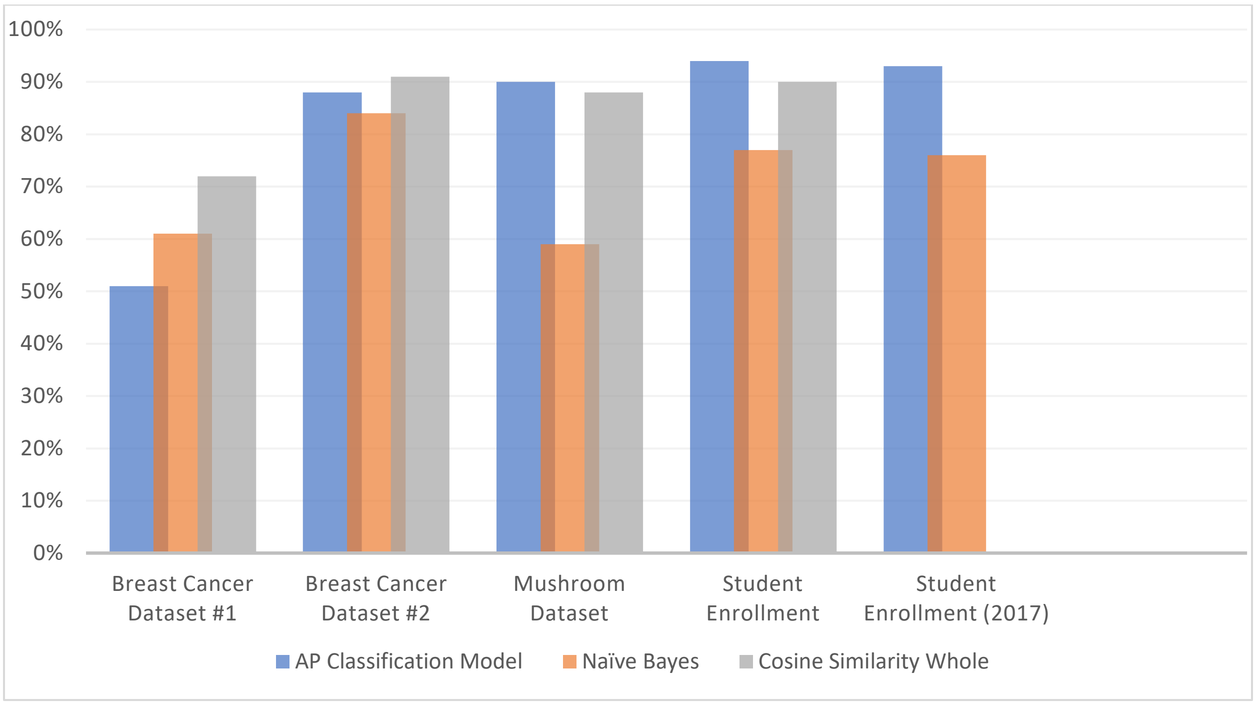


Figure 21: Comparison of accuracy between 3 classification models

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

Conclusions

This thesis illustrates a novel way to build a training set for classification purposes by using a reduced dataset represented as exemplars from the output of affinity propagation as a training set. It shows that even a subset of the dataset, as long as this subset contains the most important records, measured by their return as exemplars from a clustered output, will serve as training data for a classifier and will classify as well as Naïve Bayes and Cosine Similarity on the whole training set. This thesis also shows that the exemplars chosen from a dataset at one point, will still represent new data records for the same dataset for at least one year into the future with an accuracy of over 90%. This illustrates that the exemplars from a given year of data can still act as exemplars for another given year of data. However, as data can fluctuate over time, the accuracy of the training data for a given year will start to produce lower accuracies over time. This can be addressed by creating new exemplars for additional data.

Another key feature to this technique is the training data itself. The training data is saved to a file which is sent to a classification model. This file can be edited. It can be added to, and even exemplars removed. Additional logic could be added to the file to show which exemplars are getting the most hits from new records. The exemplars that are getting fewer hits, could be exemplars worth removing. Clustering new data with new exemplars outputted can be added to the training file as well. If this is done, procedures to make sure that no duplicates exist and that there is a clear delineation between the exemplars of all classes would need to be done. A process was performed for the binary exemplar data to ensure that no exemplar matches another exemplar in the other class.

This technique is also transparent. That is, how it classifies records is intuitive to the user by reading the training file which contains all exemplars containing the feature information from the dataset unaltered. Each exemplar array describes the qualities which make up the class. This information can be used easily to explain features in the data using real words and not numeric values such as the output returned from other classifiers as Artificial Neural Networks and Support Vector Machines.

Some of the negative features of the technique are shown with the breast cancer data where small datasets do not give a good accuracy result. The overhead in creating the exemplars is extensive, however once the training set is created it does not have to be recreated. As shown with the student data for 2018 which was tested with exemplars created using 2017, the accuracy for prediction was still very strong. Also classification is biased towards the clustering of the data itself. This occurs in other classifiers as well where the distribution of records for the classifications are not even or close to even. For example, if the number of records which contains class = A is 10% of the whole dataset, the dataset needs to be truncated. This is the same for other classifiers where the number of records which contain information for each class needs to be evenly distributed. Finally, this technique is not tested for numeric data. However given the classifier is built for discrete values, and small features sets for attributes, datasets containing numeric attributes like age or distance will not cluster well unless these numerical attributes are “binned”, that is specific ranges are translated into a category or bin. All datasets for this technique are tested using only datasets containing full categorical values. If any attributes were numeric, they were converted to categories before running affinity propagation.

Future Work

Expanding this technique to be able to include numeric attributes would lend itself well to many more datasets. The categorical algorithm used to calculate a categorical pair’s distance can be expanded to include numerical values and can be found in Ahmad and Dey’s paper[30]. Other techniques not explored are adding to the exemplar records based on clustering of another dataset. Benchmarking this technique against

[30] Amir Ahmad and Lipika Dey, "A K-mean Clustering Algorithm for Mixed Numeric and Categorical Data," Data & Knowledge Engineering 63, no. 2 (2007): , doi:10.1016/j.datak.2007.03.016.

other classifiers such as Artificial Neural Networks, and Decision Trees to calculate where it stands with accuracy.

Related to the affinity propagation, other ways to zero in on an ideal preference value, and an ideal damping factor could help increase accuracy. Either value is highly volatile to the creation of clusters so that a value that is not ideal can result in distorted cluster sets or not enough clusters or too many clusters to describe the dataset. This extension of the algorithm lends itself well to the idea of adaptive techniques especially related to affinity propagation. Because of affinity propagation strong connection to the values of the preference and damping factor, it is possible to get entirely different cluster returns, and in some cases no cluster returns at all. By applying adaptive techniques which can gauge the damping factor's effect on convergence, and the preference value's effect on the quality of the returned clusters, it is possible to increase the accuracy of the exemplars chosen and thus increase the performance and accuracy of the classifier.

What if other clustering algorithms were used? KMeans was not used as it does not provide clear cut features when it returns its centroids. The centroids for KMeans are not actual data points. However a data point could be extracted using a few methods. For example, it may be possible to get a data record that is an exemplar to the cluster by finding the closest record to the centroid. However what if the cluster is sparse? Then it may be necessary to "build" an exemplar from the majority of values found in all data assignments. Either way, this process will take more time in constructing the training dataset. KMeans out performs affinity propagation in relation to the execution time. That is affinity propagation can be slow in reaching convergence, however KMeans reaches convergence quickly, therefore for very large dataset, it may be more practical to use an adapted technique of using KMeans over affinity propagation to get the exemplars of clusters from a dataset. affinity propagation starts to get impractical if you have around 10,000 records, therefore depending on the speed of the computer, this could mean a matter of minutes to get exemplars over a matter of days.

REFERENCES

"Affinity Propagation - Java." Cognitive Foundry. Accessed April 15, 2019. https://foundry.sandia.gov/releases/latest/javadoc-api/index.html.

Ahmad, Amir, and Lipika Dey. "A K-mean Clustering Algorithm for Mixed Numeric and Categorical Data." *Data & Knowledge Engineering*63, no. 2 (2007): 503-27. doi:10.1016/j.datak.2007.03.016.

Ahmad, Amir, and Lipika Dey. "A K-mean Clustering Algorithm for Mixed Numeric and Categorical Data." *Data & Knowledge Engineering*63, no. 2 (2007): 503-27. doi:10.1016/j.datak.2007.03.016.

Barrett, Paul. "Euclidean Distance Raw, Normalized, and Double-scaled Coefficients." The Technical Whitepaper Series. September 2005. Accessed April 15, 2019. https://www.pbarrett.net/techpapers/euclid.pdf.

Bholowalia, Purnima, and Arvind Kumar. "EBK-Means: A Clustering Technique Based on Elbow Method and K-Means in WSN." *International Journal of Computer Applications*105, no. 9 (November 2014): 17-24. https://pdfs.semanticscholar.org/5771/aa21b2e151f3d93ba0a5f12d023a0bfcf28b.pdf.

Cherkassky, Vladimir S., and Filip Mulier. *Learning from Data: Concepts, Theory, and Methods*. 2nd ed. Hoboken, NJ: IEEE Press/Wiley-Interscience, 2007.

Coenen, Frans. "Data Mining: Past, Present and Future." *The Knowledge Engineering Review*26, no. 1 (2011): 25-29. doi:10.1017/s0269888910000378.

Frey, B. J., and D. Dueck. "Clustering by Passing Messages Between Data Points." *Science*315, no. 5814 (2007): 972-76. doi:10.1126/science.1136800.

Garcia, Edel. "Cosine Similarity Tutorial." Minerazzi. April 10, 2005. Accessed April 15, 2019. http://www.minerazzi.com/tutorials/cosine-similarity-tutorial.pdf.

Jain, A. K., M. N. Murty, and P. J. Flynn. "Data Clustering: A Review." *ACM Computing Surveys*31, no. 3 (1999): 264-323. doi:10.1145/331499.331504.

Kalechofsky, Hal. "HomeA Simple Framework for Building Predictive Models." M Squared Consulting. September 2016. Accessed April 15, 2019. http://www.msquared.com/.

Kumar, Vipin, Pan-Ning Tan, Michael Steinbach, and Anuj Karpatne. "Cluster Analysis: Basic Concepts and Methods." In *Introduction to Data Mining*, 525-612. 2nd ed. New York, NY: Pearson Education, 2005.

Lawson, Daniel, and Daniel Falush. "Similarity Matrices and Clustering Algorithms for Population Identification Using Genetic Data." FineSTRUCTURE. March 1, 2012. Accessed April 15, 2019. https://people.maths.bris.ac.uk/~madjl/finestructure/.

Refianti, R., A.B. Mutiara, and S. Gunawan. "Time Complexity Comparison Between Affinity Propagation Algorithms." *Journal of Theoretical and Applied Information Technology*95, no. 7 (April 15, 2007): 1497-505.

Schlimmer, Jeff. "Mushroom Data Set." UCI Machine Learning Repository. April 27, 1987. Accessed April 15, 2019. https://archive.ics.uci.edu/ml/datasets/Mushroom.

Soklic, Milan, and Matjaz Zwitter. "Breast Cancer Data Set." UCI Machine Learning Repository. July 11, 1988. Accessed April 15, 2019. https://archive.ics.uci.edu/ml/datasets/Breast Cancer.

Trono, John, Dave Kronenberg, and Patrick Redmond. "Affinity Propagation, and Other Data Clustering Techniques." Affinity Propagation, and Other Data Clustering Techniques. Accessed April 15, 2019. http://academics.smcvt.edu/.

Wolberg, William H. "Breast Cancer Wisconsin (Original) Data Set." UCI Machine Learning Repository. July 15, 1992. Accessed April 15, 2019. https://archive.ics.uci.edu/ml/datasets/Breast Cancer Wisconsin (Original).

APPENDIX

PYTHON CODE FROM PROJECT

```
from sklearn.cluster import AffinityPropagation
import numpy as np
import pandas as pd
import time
import math
import random
import matplotlib
matplotlib.use('TkAgg') #For OSX
from tkinter import *

class AttributeInformation:
    "This object contains attribute information including all attribute distance pairs and the 
significance of the attribute"
    significance = 0.0
    distancePairs = {} #using dict syntax [("value1:value2", value), ("value1|value2", value)]
    attributeName = ""
    attributeType = ""
    WithRespectToAttributej = ""

    def setDistancePair(self, Valuex, Valuey, Value):
        stringValue = Valuex + ":"+ Valuey
        self.distancePairs[stringValue] = Value

    def getDistancePair(self, Valuex, Valuey):
        stringValue = Valuex + ":"+ Valuey
        return self.distancePairs[stringValue]

    def setSignificance(self, S):
        sum = 0
        lenS = len(S)

        for key in self.distancePairs:
            sum = sum + float(self.distancePairs[key])

        if len(S) > 1:
            self.significance = sum / ((lenS * (lenS - 1)) / 2)
        else:
            self.significance = 0

    def getSignificance(self):
        return self.significance


########################################################################
# This allows me to create an attribute object, and return the object
def New_AttributeInformation(distances, S, attributeName, attributeType):
    ob = AttributeInformation();
    ob.distancePairs = distances
    ob.setSignificance(S)
    ob.attributeName = attributeName
    ob.attributeType = attributeType

    return ob

########################################################################
# Reads my similarity matrix from SimilarityMatrixFileName using
# SimilarityMatrixDelimiter as the delimiter.
def ReadSimilarityMatrix(SimilarityMatrixFileName, SimilarityMatrixDelimiter):
    return np.loadtxt(SimilarityMatrixFileName, delimiter=SimilarityMatrixDelimiter);


########################################################################
# Compute Affinity Propagation using the similarity Matrix simData and
# the preference preferenceValue
def ComputeAffinityPropagation(preferenceValue, simData):
    af = AffinityPropagation(preference=preferenceValue,
                            affinity='precomputed',
                            verbose=False,
                            max_iter=300,
                            damping=.9)
    af.fit(simData)

    return af


########################################################################
# This function was created specifically to place all commands that will
# run this program in AP in one place.
def RunAffinityPropagationCommands(Matrix, SimFile, preference, ResultsFilename):

    print("Reading Similarity Matrix from File")
    Matrix = ReadSimilarityMatrix(SimFile, ",")

    print("Running Affinity Propagation")
    start = time.time()
    af = ComputeAffinityPropagation(preference, Matrix)
    end = time.time()

    cluster_centers_indices = af.cluster_centers_indices_
    labels                  = af.labels_
    n_clusters_             = len(cluster_centers_indices)
    print('Estimated number of clusters: %d' % n_clusters_)

    WriteResultsToFile(ResultsFilename, labels, cluster_centers_indices)



########################################################################
#Calcuates the accuracy of the cluster by adding together all the
#similarity values from the assignment to the centroid.
def GetTotalAccuracyOfClusters(FullDBInClusterFormAffinity):
    FullTotalAccuracy = 0.0

    #For each Cluster
    for i in range(len(FullDBInClusterFormAffinity)):

        Attributes          = FullDBInClusterFormAffinity[i][0]
        a                   = np.array(FullDBInClusterFormAffinity[i])
        TotalAccuracy       = 0.0
        count               = 0

        #For each attribute of assignment in cluster
        for k in range(len(FullDBInClusterFormAffinity[i][1])):

            #Get all values in column of attribute
            col = a[:,k]

            #get the value our centroid contains
            value = FullDBInClusterFormAffinity[i][1][k]

            #Because we can have mixed values, this only works for categories in
            #attributes, therefore we wouldn't do this with a numeric attribute.
            if(Attributes[k] in AttributesStrings):

                count = count + 1

                #accuracy is the number of times the value appears in the column
                #over the length of column.
                PercentageAccuracy      = len(np.where(col == value)[0]) / len(col)

                #StrPercentageAccuracy   = "{0:.5f}".format(PercentageAccuracy  )

                #Add this to our total accuracy of the cluster. We will do this for
                #all attributes to get the total accuracy of the cluster.
                TotalAccuracy           = TotalAccuracy + PercentageAccuracy
        #Get the average total accuracy of the cluster and add that to the
        #total accuracy for all clusters.
        FullTotalAccuracy = FullTotalAccuracy + (TotalAccuracy / count)

    #Return the average of all cluster errors.
    return FullTotalAccuracy / len(FullDBInClusterFormAffinity)

########################################################################
# This creates an html file which contains as much statistical information
# about the clusters, what assignments are included, the actual information
# from the database, the exemplars, the cluster error, the percentage of each
# centroid in the cluster, etc.
def CreateHTMLFileWithClusters(FullDBInClusterFormAffinity, FileName, TrainingDB):

    f = open(FileName, "w")

    f.write("<html>")
    f.write("<head>")
    f.write("<style>")
    f.write("table{border-collapse:collapse;border:2px solid black;}")
    f.write("th, td{padding: 5px;border-collapse:collapse;border:1px solid #999;}")
    f.write(".header{background-color:#CCC;}")
    f.write(".headerg{background-color:#lightgreen;}")
    f.write(".exemplar{background-color:#999;}")
    f.write(".exemplarg{background-color:#lightgreen;}")
    f.write(".regularg{background-color:#lightgreen;}")
    f.write("</style>")
    f.write("<body>")

    f.write("<h1>Show Cluster Statistics</h1>")

    FullTotalAccuracy = 0.0

    FullDisplay = []

    for i in range(len(FullDBInClusterFormAffinity)):
        f.write("<h2>Cluster "+str((i+1))+"</h2>")

        NumberOfAssignments = len(FullDBInClusterFormAffinity[i])
        PercentOfTotal      = NumberOfAssignments / len(TrainingDB)

        f.write("<div>Number of Assignments: " + str(NumberOfAssignments)+"</div>")
        f.write("<div>Percent of Total: " + str(PercentOfTotal)+"</div>")
        a               = np.array(FullDBInClusterFormAffinity[i])
        Attributes      = FullDBInClusterFormAffinity[i][0]
        TotalAccuracy   = 0.0
        count           = 0
        dictionary      = {}
        for k in range(len(FullDBInClusterFormAffinity[i][1])):

            col = a[:,k]
            value = FullDBInClusterFormAffinity[i][1][k]
            column = FullDBInClusterFormAffinity[i][0][k]

            dictionary[str(column)+"exemplar"] = {}
            count = count + 1
            b                       = np.where(col == value)
            NumberOfExemplarValue   = len(b[0])
            PercentageAccuracy      = NumberOfExemplarValue / (len(col)-1)
            StrPercentageAccuracy   = "{0:.5f}".format(PercentageAccuracy  )
            TotalAccuracy           = TotalAccuracy + PercentageAccuracy
            f.write("<div><b>("+str(column)+":"+str(value)+")        Accuracy:        "        +
str(StrPercentageAccuracy)+"</b></div>")
            col = np.delete(col, 0)
            columnValues = np.unique(col)
            dictionary[str(column)+"exemplar"][value] = PercentageAccuracy

            for j in range(len(columnValues)):
                if(columnValues[j] is not value):
                    dictionary[str(column)] = {}
                b = np.where(columnValues[j] == col)
                NumberOfExemplarValue = len(b[0])
                PercentageAccuracy = NumberOfExemplarValue / (len(col)-1)
                dictionary[str(column)][str(columnValues[j])] = PercentageAccuracy

        FullDisplay.append(dictionary)

        FullTotalAccuracy = FullTotalAccuracy + (TotalAccuracy / count)
        TotalAccuracy = "{0:.5f}".format(TotalAccuracy / count)
        f.write("<div>Total Accuracy: " + str(TotalAccuracy)+"</div>")

    f.write("<h2>Full Total Accuracy: " + str(FullTotalAccuracy / len(FullDBInClusterFormAffinity))
+"</h2>")

    f.write("<h1> Table of Results </h1>")

    strValue = "<table>"
    for i in range(len(FullDisplay)):
        x = FullDisplay[i]
        strValue += "<tr>"
        max = 0.0
        alternateKey = ""
        alternateKeyValue = ""
        for key in x:
            if("exemplar" in key):
                max = 0.0
                exemplarInfo = x[key]
                val = ""
                for key2 in exemplarInfo:
                    ExemplarAccuracy = exemplarInfo[key2]
                    max = ExemplarAccuracy
                    val = key2

                w = "{0:.3f}".format(ExemplarAccuracy)

                key = str(key)[:-8]
                strValue += "<td><b>"+str(key)+"("+str(val)+"):"+w+"</b>"
            else:
                info = x[key]
                for key2 in info:
                    y = info[key2]
                    if y > max:
                        max = y
                        alternateKey = key2
                        alternateKeyValue = y

                if(alternateKey != ""):
                    w = "{0:.3f}".format(max)
                    strValue += "<br>"+str(key2)+":"+w+"</td>"
                else:
                    strValue += "</td>"

        strValue += "</tr>"

    strValue += "</table>"

    f.write(strValue)

    f.write("<h1>Full Detail on Clusters</h1>")

    for i in range(len(FullDBInClusterFormAffinity)):
        strValue = "<h2>Cluster " + str(i+1) + "</h2>"
        strValue += "<table>"
        a = np.array(FullDBInClusterFormAffinity[i])


        for j in range(len(FullDBInClusterFormAffinity[i])):
            strValue += "<tr>"
            for k in range(len(FullDBInClusterFormAffinity[i][j])):
                column = FullDBInClusterFormAffinity[i][0][k]
                TotalAccuracy = "N/A"

                col = a[:,k]
                value = FullDBInClusterFormAffinity[i][j][k]

                b = np.where(col == value)
                accuracy = len(b[0]) / len(col)
                TotalAccuracy = "{0:.2f}".format(accuracy*100)

                if j == 0:
                    strValue           +=           "<th           class=\"header\">"           +
str(FullDBInClusterFormAffinity[i][j][k]) + "+("+TotalAccuracy+") </th>"

                elif j == 1:
                    strValue               +=              "<td              class=\"exemplar\">"+
str(FullDBInClusterFormAffinity[i][j][k]) + "+("+TotalAccuracy+") </td>"

                else:
                    strValue               +=              "<td               class=\"regular\">"+
str(FullDBInClusterFormAffinity[i][j][k]) + "+("+TotalAccuracy+") </td>"


            strValue += "</tr>"


        strValue += "</table>"

        f.write(strValue)

    f.write("</body>")
    f.write("</html>")
    f.close()
    print("HTML File of results created!")


    return FullDisplay

###############################################################################
# Writes the results of the clustering to a file.
def WriteResultsToFile(Filename, labels, clusterCenters):

    f = open(Filename, "w")

    f.write("\n")
    for i in range(len(labels)):
        if i == 0:  f.write(str(labels[i]))
        else:       f.write(","+str(labels[i]))

    f.write("\n")
    for i in range(len(clusterCenters)):
        if i == 0:  f.write(str(clusterCenters[i]))
        else:       f.write(","+str(clusterCenters[i]))

    f.close()
    print("Results Written to File")
    return

###############################################################################
# Read results of a cluster from a file.
def ReadResultsFromFile(Filename):

    ResultsFile     = pd.read_csv(Filename, sep=",")
    labels          = ResultsFile.columns
    ClustersWhole   = ResultsFile.iloc[0]
    Clusters        = []

    labels          = labels.tolist()
    ClustersWhole   = ClustersWhole.tolist()

    for i in range(len(ClustersWhole)):
        if(math.isnan(ClustersWhole[i]) == False):
            Clusters.append(ClustersWhole[i])

    ListOfClusters = []
    for i in range(len(Clusters)):
        ClusterInfo = []
        for j in range(len(labels)):
            if(int(math.floor(float(labels[j]))) == i):
                ClusterInfo.append(j)

        ListOfClusters.append(ClusterInfo)

    return np.array(ListOfClusters)


##############################################################################
# Read Database
def ReadDatabase(FileName):
    return pd.read_csv(FileName, sep=",")


##############################################################################
# Get all unique category values from each column of the database.
def getCategoricalValuesForEachAttribute(dbSubsetData):

    if len(dbSubsetData) > 0:
        categoriesForAttribute = []
        for column in dbSubsetData:

            categoriesForAttribute.append(dbSubsetData[column].unique())

        return categoriesForAttribute
    else:
        return null


##############################################################################
# Performs algorithm for computing distance with respect to number of data points.
def ALGODistance(dbSubsetData, Columns):

    categories = getCategoricalValuesForEachAttribute(dbSubsetData)

    distancePairsDict = {}

    for i in range(dbSubsetData.shape[1]):

        start           = time.time()
        distancePairs   = {}

        for j in range(len(categories[i])):
            for k in range(len(categories[i])):
                if j != k:
                    count           = 0
                    distance        = 0
                    sum             = 0
                    Valuex          = str(categories[i][j])
                    Valuey          = str(categories[i][k])

                    strValue1       = Valuex+":"+Valuey
                    strValue2       = Valuey+":"+Valuex

                    if strValue1 not in distancePairs and strValue2 not in distancePairs:

                        for m in range(dbSubsetData.shape[1]):

                            if i != m:
                                distanceim = findMax(Valuex, Valuey, categories[m], dbSubsetData,
m, i)

                                sum = sum + distanceim
                                count = count + 1
                        if(count > 0):
                            distance = sum / (count - 1)

                        distancePairs[strValue1] = distance
                        distancePairs[strValue2] = distance


        end = time.time()
        print("This took",(end-start),"(s)")
        distancePairsDict[Columns[i]] = distancePairs

    return distancePairsDict


##############################################################################
# finds the maximum value of Pi(w/x) + Pi(w/y), without actually considering
# all the element of the power set
def findMax(Ax, Ay, CategoriesForAj, df, m, n):

    distance = 0
    W = []
    complementW = []
    Vj = len(CategoriesForAj)
    dfnp = df.values

    for i in range(Vj):

        #Conditional probability of CategoriesForAj[i] given Ax and
        #CategoriesForAj[i] given Ay

        Px = 0
        Py = 0

        a = np.where((dfnp[:,m] == CategoriesForAj[i]) & (dfnp[:,n] == Ax))[0]
        b = np.where(dfnp[:,n] == Ax)[0]
        if(len(b) > 0): Px = len(a) / len(b)

        a = np.where((dfnp[:,m] == CategoriesForAj[i]) & (dfnp[:,n] == Ay))[0]
        b = np.where(dfnp[:,n] == Ay)[0]
        if(len(b) > 0): Py = len(a) / len(b)

        if Px > Py:
            W.append(CategoriesForAj[i]) #   add u[t] to w'
            distance = distance + Px
        else:
           complementW.append(CategoriesForAj[i]) #   add u[t] to ~w'
           distance = distance + Py

    distance = distance - 1

    return distance

##############################################################################
# Creates our similarity matrix using our new distanceObjectList and the
# dbsubsetdata frame

def CreateSimilarityMatrix(DB, distancePairsDict, SimDataFile):

    columns         = DB.columns.values
    nparray         = DB.values
    outcome         = nparray[:,len(columns) - 1]
    classes         = np.unique(outcome)
    FileNameArray   = []
    classArray = []
    SubSets = []

    for cls in classes:
        cls = str(cls)
        row_indices = np.where(outcome == str(cls))[0]
        subset      = nparray[row_indices, :]
        SubSets.append(subset)
        Matrix      = np.full((subset.shape[0], subset.shape[0]), 0.0)

        print("Creating Similarity Matrix")

        for i in range(len(subset)): #record1

            start = time.time()
            Record1 = subset[i]

            for j in range(i+1, len(subset)): #record2
                Record2 = subset[j]
                Similarity    =    GetDistanceBetweenTwoRecords(Record1,    Record2,    columns, 
distancePairsDict)

                #The similarities are all very small numbers which might be why the preference 
value needs to be so precise.
                #Let's multiply the value by a scalar 10 to give the values more range.
                Matrix[i][j] = Similarity * 10.0
                Matrix[j][i] = Similarity * 10.0
            end = time.time()
            print("Iteration",i,"took",end-start,"(s)")


        print("Writing Similarity Matrix to File")
        SimFile = SimDataFile + "_" + cls + ".csv"
        FileNameArray.append(SimFile)
        classArray.append(cls)
        FileWritten = WriteSimilarityMatrixToFile(Matrix, SimFile)

        if FileWritten is True:
            print("Similarity File is Written")

    return FileNameArray, classArray, SubSets

def GetDistanceBetweenTwoRecords(Record1, Record2, columns, distancePairsDict):

    distance = 0.0

    for i in range(len(Record1)):
        columnName = columns[i]
        Record1Value = Record1[i]
        Record2Value = Record2[i]
        if(Record1Value != Record2Value):

            strValue    = str(str(Record1Value)+":"+str(Record2Value))
            strValue2   = str(str(Record2Value)+":"+str(Record1Value))
            val         = 0.0

            if(strValue in distancePairsDict[columnName]):
                val = float(distancePairsDict[columnName][strValue])
            if(strValue2 in distancePairsDict[columnName]):
                val = float(distancePairsDict[columnName][strValue2])

                distance = distance + ((val**2) * -1);

    return distance

def WriteSimilarityMatrixToFile(Matrix, FileName):

    f = open(FileName, "w")

    for i in range(len(Matrix)):
        for j in range(len(Matrix[0])):
            if j == 0:
                f.write(str(Matrix[i][j]))
            else:
                val = "," + str(Matrix[i][j])
                f.write(val)
        f.write("\n")

    f.close()
    return True


def RunSeriesOfPreferences(lowValue, highValue, SimFile, increment):

    print("Reading Similarity Matrix from File")
    Matrix = ReadSimilarityMatrix(SimFile, ",")

    Errors = []
    Clusters = []
    Preference = []

    while lowValue < highValue:

        af = ComputeAffinityPropagation(lowValue, Matrix)
        cluster_centers_indices = af.cluster_centers_indices_
        labels = af.labels_
        n_clusters_ = len(cluster_centers_indices)
        if n_clusters_ not in Clusters:

            Error = GetErrorForClusters(cluster_centers_indices, labels, Matrix)
            if(len(Errors) and Error > Errors[len(Errors)-1]):
                i = 0
            else:
                Clusters.append(n_clusters_)
                Errors.append(Error)
                Preference.append(lowValue)
                #print("Number of Clusters: ",n_clusters_,"\tPreference Value",lowValue,"\tTotal
Error:",Error)

        lowValue = lowValue + increment


    atIndex, min, optimalNumber = LocateElbowPoint(Errors, Clusters)

    while(atIndex>=0):
        if((Clusters[atIndex] / n_clusters_) > .40):
            atIndex = atIndex - 1
        else:
            break

    #print("Errors:", Errors)
    #print("Clusters:", Clusters)
    #print("Preferences:", Preference)
    #print("Elbow Found at index", atIndex,"value of ", Errors[atIndex], "thus Optimal number of
clusters is", Clusters[atIndex])

    return Preference[atIndex], Clusters[atIndex]

def GetErrorForClusters(Centroids, Labels, Matrix):

    TotalError = 0
    for i in range(len(Centroids)):

        count = 0
        Sum = 0
        for j in range(len(Labels)):

            if(i == Labels[j]):
                Sum = Sum + (Matrix[i][j] * -1)
                count = count + 1

        ErrorForCluster = Sum / count
        TotalError = TotalError + ErrorForCluster


    NumberOfClusters = len(Centroids)
    Mean = (NumberOfClusters * (NumberOfClusters - 1)) / 2
    TotalError = TotalError / Mean
    return TotalError * 100 #This is just to zoom out to get a better range of values.
```

```
##############################################################################
# What I am simply doing here is getting the inverse cosine of the angle of a
# vertex point. This angle tells me the variance of change at this point. What
# I am looking for is the lowest value, representing a small angle, representing
# a large variance of change.
# fall backs. Normally elbow points are easy to find and are usually located
# nearer to the y-axis. This approach doesn't eyeball the data so the computer
# is simulating what I would consider the elbow point without me actually
# verifying it.

def LocateElbowPoint(Errors, Clusters):

    min = 180
    atIndex = -1
    for i in range(1, len(Clusters)-1):
        A = i
        B = i-1
        C = i+1

        distanceP12 = ((Errors[A] - Errors[B])**2 + (Clusters[A] - Clusters[B])**2)**.5
        distanceP13 = ((Errors[A] - Errors[C])**2 + (Clusters[A] - Clusters[C])**2)**.5
        distanceP23 = ((Errors[B] - Errors[C])**2 + (Clusters[B] - Clusters[C])**2)**.5

        value = (distanceP12**2 + distanceP13**2 - distanceP23**2) / (2 * distanceP12 * distanceP13)
        angle = math.acos(value)
        angle = np.degrees(angle)
        if(angle < min):
            min = angle
            atIndex = i

    return atIndex, min, Clusters[atIndex]

def GenerateFullDataObjectInClusterForm(ListOfClusters, TrainingDB, FileName):

    DB = pd.read_csv(FileName, sep=",")

    DataFrame   = []

    for i in range(len(ListOfClusters)):
        Row = []
        Row.append(DB.columns.values)
        Col = []
        for j in range(len(ListOfClusters[i])):
            Row.append(TrainingDB[int(ListOfClusters[i][j])])

        DataFrame.append(Row)

    return np.array(DataFrame)


def CompareArrays(array1, array2):

    numberCorrect = 0
    for i in range(len(array1)-1):
        if(array1[i] == array2[i]):
            numberCorrect = numberCorrect + 1

    prob = (numberCorrect / len(array2))
    return prob


def ClassificationOfData(FullDataBase = "", Yes="", No="", dataTesting=""):

    #start with database values
    FalsePositive   = 0
    Positive        = 0
    Negative        = 0
    TrueNegative    = 0

    DB = ReadDatabase(FullDataBase)
```

```
if(dataTesting == "StudentEnrollment"):
    import ClusteringInfo as cl
elif(dataTesting == "BreastCancer1"):
    import BreastCancerClassifier as cl
elif(dataTesting == "BreastCancer2"):
    import BreastCancerClassifier as cl
elif(dataTesting == "Mushrooms"):
    import mushroominfo as cl


for i in range(len(DB)):

    a = DB.iloc[i].values
    classifier = a[len(a)-1]
    MaxProbNo = 0.0
    MaxProbYes = 0.0
    ProbNo = 0.0
    ProbYes = 0.0

    #if(classifier == No):
    for k in range(len(cl.NoArray)):
        ProbNo = ProbNo + CompareArrays(a, cl.NoArray[k])
        #if(MaxProbNo < ProbNo):
        #    MaxProbNo = ProbNo
    ProbNo = ProbNo / len(cl.NoArray)

    #elif(classifier == Yes):
    for k in range(len(cl.YesArray)):
        ProbYes = ProbYes + CompareArrays(a, cl.YesArray[k])
        #if(MaxProbYes < ProbYes):
        #    MaxProbYes = ProbYes

    ProbYes = ProbYes / len(cl.YesArray)

    print("Prediction details", ProbNo, ProbYes, classifier)

    if(ProbYes > ProbNo):
        if(classifier == Yes):
            Positive = Positive + 1
        else:
            TrueNegative = TrueNegative + 1
    elif(ProbYes < ProbNo):
        if(classifier == No):
            Negative = Negative + 1
        else:
            FalsePositive = FalsePositive + 1
    else:
        if(classifier == Yes):
            Positive = Positive + 1
        else:
            Negative = Negative + 1



print("Scores for Full Database Prediction")
print("Positive",Positive)
print("Negative",Negative)
print("False Positive", FalsePositive)
print("True Negative", TrueNegative)

Precision = Positive / (Positive + TrueNegative)
Recall = Positive / (Positive + FalsePositive)
Accuracy = (Positive + Negative) / (Positive + Negative + FalsePositive + TrueNegative)
FScore = (2*(Precision * Recall) / (Precision + Recall))
print("Precision", Precision)
print("Recall", Recall)
print("Accuracy", Accuracy)
print("FScore", FScore)

return Precision, Recall, Accuracy, FScore
```

```
def DefineDistancePairs(DB, Columns):

    #Collect all distance pairs for my subset.
    distancePairsDict = {}

    print("Getting ALGO Distances")
    distancePairsDict = ALGODistance(DB, Columns)

    return distancePairsDict


def DataSplit(FullDB, frac, classAttribute):
    msk         = np.random.rand(len(FullDB)) <= frac
    TrainingDB  = FullDB[msk].values
    TestDB      = FullDB[~msk].values

    if(classAttribute == "END"):
        classIndex = len(FullDB.columns)-1
    else:
        classIndex = 0

    classColumn = TrainingDB[:,classIndex]
    classes     = np.unique(classColumn)
    rows, cols  = np.shape(TrainingDB)

    for cls in classes:
        row_indices = np.where(classColumn == cls)[0]
        while((len(row_indices) / rows)>.70):
            index       = random.randint(0,len(row_indices)-1)
            TrainingDB  = np.delete(TrainingDB, index, axis=0)
            classColumn = np.delete(classColumn, index, axis=0)
            row_indices = np.where(classColumn == cls)[0]

    classColumn = TestDB[:,classIndex]
    classes     = np.unique(classColumn)
    rows, cols  = np.shape(TestDB)

    for cls in classes:
        row_indices = np.where(classColumn == cls)[0]
        while((len(row_indices) / rows)>.70):
            index       = random.randint(0,len(row_indices)-1)
            TestDB      = np.delete(TestDB, index, axis=0)
            classColumn = np.delete(classColumn, index, axis=0)
            row_indices = np.where(classColumn == cls)[0]


    return TrainingDB, TestDB, FullDB.columns

def WriteDBToFile(TrainingDB, TestDB, Columns, FileName, path):

    FileName = FileName.replace(" ", "")
    FullName = path+"/Training_"+FileName+".csv"

    f = open(FullName, "w")
    for i in range(len(Columns)):
        if(i==0):
            f.write(Columns[i])
        else:
            f.write(","+Columns[i])

    f.write("\n")
    for i in range(len(TrainingDB)):
        for j in range(len(TrainingDB[i])):
            if(j==0):
                f.write(str(TrainingDB[i][j]))
            else:
                f.write(","+str(TrainingDB[i][j]))

        if(i != len(TrainingDB)-1):
            f.write("\n")
    f.close()

    FileName = FileName.replace(" ", "")
    FullName = path+"/Test_"+FileName+".csv"

    f = open(FullName, "w")
    for i in range(len(Columns)):
        if(i==0):
            f.write(Columns[i])
        else:
            f.write(","+Columns[i])

    f.write("\n")
    for i in range(len(TestDB)):
        for j in range(len(TestDB[i])):
            if(j==0):
                f.write(str(TestDB[i][j]))
            else:
                f.write(","+str(TestDB[i][j]))

        if(i != len(TestDB)-1):
            f.write("\n")

    f.close()

def WriteInformationFile(WriteFile, FullDisplayArray):

    f = open(WriteFile, 'w')
    count = 0
    for FullDisplay in FullDisplayArray:

        if(count == 0):
            f.write("YesArray = [")
        else:
            f.write("NoArray = [")

        for i in range(len(FullDisplay)):
            x = FullDisplay[i]
            strValue = "["
            count = 0
            for key in x:
                if("exemplar" in key):
                    exemplarInfo = x[key]
                    for key2 in exemplarInfo:
                        if(count == 0):
                            strValue += "\""+str(key2)+"\""
                        else:
                            strValue += ","+"\""+str(key2)+"\""

                        count = count + 1

            strValue += "],"
            f.write(strValue)
            f.write("\n")

        f.write("]")
        f.write("\n")
    f.close()
import matplotlib
matplotlib.use('TkAgg') #For OSX
from tkinter import *
from tkinter import ttk
import klecker_thesis as kt
import numpy as np
import pandas as pd
import time, threading

def SetStudentEnrollmentDataset():
    kt.FullDataBase = "StudentEnrollment/2017/studentenrollment_2017.csv"
    return "StudentEnrollment"
```

```
def SetBreastCancerDataset():
    kt.FullDataBase = "BreastCancer/breastcancerfull.csv"
    return "BreastCancer"

def SetBreastCancerDataset2():
    kt.FullDataBase = "BreastCancer/breastcancer.csv"
    return "BreastCancer"

def SetMushroom():
    kt.FullDataBase = "mushroom/mushrooms.csv"
    return "mushroom"


def Main(GetSimilarityMatrix,RunAffinityPropagation,ShowAPResultsToHTML,
        RunASeriesOfPreferences,PerformClassification,dataTesting, TrainingSize):

    #this is used for naming file purposes.
    Matrix                 = []    #similarity matrix
    Yes                    = ""
    No                     = ""
    Path                   = ""
    Columns                = []
    writeFile              = ""

    if(dataTesting == "Student Enrollment"):
        Yes         = "Applied"
        No          = "No-Decision"
        Path        = SetStudentEnrollmentDataset()
        writeFile   = "ClusteringInfo.py"

    elif(dataTesting == "Breast Cancer 1"):
        Yes         = "recurrence-events"
        No          = "no-recurrence-events"
        Path        = SetBreastCancerDataset()
        writeFile   = "BreastCancerClassifier.py"

    elif(dataTesting == "Breast Cancer 2"):
        Yes         = "J2"
        No          = "J4"
        Path        = SetBreastCancerDataset2()
        writeFile   = "BreastCancerClassifier.py"

    elif(dataTesting == "Mushrooms"):
        Yes         = "e"
        No          = "p"
        Path        = SetMushroom()
        writeFile   = "mushroominfo.py"

    ###############################################################################
    # main reads the database, and calculates the similarity matrix
    dataTesting     = dataTesting.replace(" ", "")

    Precision   = 0.0
    Recall      = 0.0
    Accuracy    = 0.0
    FScore      = 0.0
    RangeValue  = 1

    for i in range(0, RangeValue):
        FileNameArray   = []
        classes         = []
        SubSets         = []
        FileName        = ""

        if(GetSimilarityMatrix == 1):
            FileName                    = Path + "/Training_"+dataTesting+".csv"
            kt.SimilarityMatrixFileName = Path + "/Training_SimMatrix_"+dataTesting

            FullDBCSV                   = kt.ReadDatabase(kt.FullDataBase)
            print(TrainingSize)
            TrainingDB, TestDB, Columns = kt.DataSplit(FullDBCSV, (int(TrainingSize)/100), "END")
            kt.WriteDBToFile(TrainingDB, TestDB, Columns, dataTesting, Path)
            TrainingDB                    = kt.ReadDatabase(FileName)
            distanceDict                  = kt.DefineDistancePairs(TrainingDB, Columns)
            FileNameArray, classes, SubSets = kt.CreateSimilarityMatrix(TrainingDB, distanceDict, 
kt.SimilarityMatrixFileName)

        ##############################################################################
        # Runs affinity propagation in a series using different preference values
        preferencesArray = []
        ClustersArray = []

        if(RunASeriesOfPreferences == 1):
            for i in range(len(FileNameArray)):
                PreferenceVal, Clusters = kt.RunSeriesOfPreferences(-8, 2, FileNameArray[i], 1)
                preferencesArray.append(PreferenceVal)
                ClustersArray.append(Clusters)

        ##############################################################################
        # Reads from the similarity file and runs affinity propagation
        ResultsFiles = []
        if(RunAffinityPropagation):
            for i in range(len(FileNameArray)):

                kt.ResultsFileName                                              =     Path     + 
"/Training_Results_"+classes[i]+"_AP_"+dataTesting+".csv"
                ResultsFiles.append(kt.ResultsFileName)
                kt.RunAffinityPropagationCommands(Matrix,  FileNameArray[i],  preferencesArray[i], 
kt.ResultsFileName)


        ##############################################################################
        #Print Results to HTML
        FullDisplayArray = []
        if(ShowAPResultsToHTML == 1):
            for i in range(len(ResultsFiles)):
                kt.HTMLFileName                                                   =    Path    + 
"/Training_HTML_"+classes[i]+"_AP_"+dataTesting+".html"
                ListOfClusters              = kt.ReadResultsFromFile(ResultsFiles[i])
                FullDBInClusterFormAffinity                                                    = 
kt.GenerateFullDataObjectInClusterForm(ListOfClusters, SubSets[i], FileName)
                FullDisplay                                                                    = 
kt.CreateHTMLFileWithClusters(FullDBInClusterFormAffinity, kt.HTMLFileName, SubSets[i])
                FullDisplayArray.append(FullDisplay)

            kt.WriteInformationFile(writeFile, FullDisplayArray)


        ##############################################################################
        # Perform a classification with training data on test data.
        if(PerformClassification):
            #FileName = Path + "/Test_"+dataTesting+".csv"
            FileName = Path + "/2018/studentenrollment_2018.csv"
            p, r, a, f = kt.ClassificationOfData(
                                                FullDataBase=FileName,
                                                Yes=Yes,
                                                No=No,
                                                dataTesting=dataTesting)

            Precision = Precision + p
            Recall = Recall + r
            Accuracy = Accuracy + a
            FScore = FScore + f

    return Precision, Recall, Accuracy, FScore, RangeValue


class MyApp(Tk):

    def __init__(self):
        Tk.__init__(self)
        self.running = False
        fr = Frame(self)
        fr.pack()

        self.GetSimilarityMatrixCheck        = IntVar(value=1)
        self.RunAffinityPropagationCheck     = IntVar(value=1)
        self.ShowAPResultsToHTMLCheck        = IntVar(value=1)
        self.RunASeriesOfPreferencesCheck    = IntVar(value=1)
        self.PredictionModelCheck            = IntVar(value=1)

        self.LeftFrame = Frame(fr)

        self.labelframe2 = LabelFrame(self.LeftFrame, text="How Will we run the application?")
        self.labelframe2.grid(row=0, column=1, padx=(10, 10), pady=(10, 10))

        self.C1 = Checkbutton(self.labelframe2, text = "Get Similarity Matrix", variable = 
self.GetSimilarityMatrixCheck, onvalue = 1, offvalue = 0, height=1, width = 20)
        self.C2 = Checkbutton(self.labelframe2, text = "Run Affinity Propagation", variable = 
self.RunAffinityPropagationCheck, onvalue = 1, offvalue = 0, height=1, width = 20)
        self.C3 = Checkbutton(self.labelframe2, text = "Show AP Results to HTML", variable = 
self.ShowAPResultsToHTMLCheck, onvalue = 1, offvalue = 0, height=1, width = 20)
        self.C4 = Checkbutton(self.labelframe2, text = "Run a Series of Preferences", variable = 
self.RunASeriesOfPreferencesCheck, onvalue = 1, offvalue = 0, height=1, width = 20)
        self.C5 = Checkbutton(self.labelframe2, text = "Prediction Model", variable = 
self.PredictionModelCheck, onvalue = 1, offvalue = 0, height=1, width = 20)

        self.C1.grid(row=0, columnspan=1)
        self.C2.grid(row=1, columnspan=1)
        self.C3.grid(row=2, columnspan=1)
        self.C4.grid(row=3, columnspan=1)
        self.C5.grid(row=4, columnspan=1)

        self.datasetFrame = Frame(self.LeftFrame)
        self.labelTop = Label(self.datasetFrame, text = "What Dataset are we testing?")
        self.labelTop.grid(row=0, column=1)

        self.comboExample = ttk.Combobox(self.datasetFrame,values=["Student Enrollment",
                                                "Breast Cancer 1",
                                                "Breast Cancer 2",
                                                "Mushrooms"])
        self.comboExample.grid(row=1, column=1)
        self.labelTraining = Label(self.datasetFrame, text = "Size of Training set (in percent)?")
        #self.labelTraining.grid(row=2, column=1)
        self.TrainingSize = Text(self.datasetFrame, wrap='word', width=5, height=1)
        #self.TrainingSize.grid(row=3, column=1)
        self.datasetFrame.grid(row=1, column=1,padx=(10, 10), pady=(10, 10))
        self.TrainingSize.insert(END, "70")

        self.buttonRun = Button(self.LeftFrame, text ="Run", command = self.runCallBack)
        self.buttonRun.grid(row=2, column=1)
        self.buttonClear = Button(self.LeftFrame, text ="Clear", command = self.Clear)
        self.buttonClear.grid(row=3, column=1)
        self.LeftFrame.grid(row=0, column=1, rowspan=2, sticky='N',)

        self.chatBox = Scrollbar(fr)
        self.chat = Text(fr, wrap='word', width=50,
                    yscrollcommand=self.chatBox.set)
        self.chatBox.configure(command=self.chat.yview)

        self.chat.grid(row=0, column=2, columnspan=2, sticky='ens', pady=(10, 10))
        self.chatBox.grid(row=0, column=4, sticky='ns', pady=(10, 10))

    def Clear(self):
        self.chat.delete(1.0,END)

    def runCallBack(self):

        self.running = True
        self.chat.insert(END, "Running\n")
        self.updateText()
        Precision,         Recall,         Accuracy,         FScore,         RangeValue        =
Main(self.GetSimilarityMatrixCheck.get(),      #Run Similarity Matrix From database selected above
                                self.RunAffinityPropagationCheck.get(),           #Run   Affinity
Propagation on Similarity Matrix
                                self.ShowAPResultsToHTMLCheck.get(),       #Write Results of AP to
HTML
                                self.RunASeriesOfPreferencesCheck.get(),   #This is used to collect
an Elbow Method series of values
                                self.PredictionModelCheck.get(),         #Perform a classification
on exemplar data
                                self.comboExample.get(),
                                self.TrainingSize.get("1.0", END))

        self.chat.insert(END, "Precision "+str(Precision / RangeValue)+"\n")
        self.chat.insert(END, "Recall "+str(Recall / RangeValue)+"\n")
        self.chat.insert(END, "Accuracy "+str(Accuracy / RangeValue)+"\n")
        self.chat.insert(END, "FScore "+str(FScore / RangeValue)+"\n")

        self.running = False


    def updateText(self):
        if(self.running):
            self.update()
            self.after(10, self.updateText)


if __name__ == "__main__":
    top = MyApp()
    top.mainloop()
    #DrawWindow()


##############################################################################
# Pandas and numpy preferences settings
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=np.inf)
pd.options.mode.chained_assignment = None  # default='warn'
```