CS5785: Applied Machine Learning **(Due: 11/09/19)**

Homework 3

Instructor: Serge Belongie *Names:* Eva Esteban Velasco, Simran Rajpal*, Netids:* epe26, sr2258

1 Programming Exercises

1.1 Sentiment Analysis of Online Reviews

(a) Download and parse the data

For this part, we downloaded yelp_labelled.txt, amazon_cells_labelled.txt and imdb_labelled.txt, and parsed them into 3 different pandas dataframes with 2 columns each: Label and Sentence. We found that the Yelp and Amazon reviews both had 500 0's and 500 1's. However, the Imdb reviews only had 362 0's and 386 1's. Thus, the labels were very close to being balanced (Yelp and Amazon are balanced with ratio 1 and Imdb is close with ratio 1.07). The overall ratio of 1's to 0's is 1.02

(b) Pick preprocessing strategy

To pre-process the data, we converted all the sentences to lower case, so that the same word with some upper case letters and without them would not be detected as different words since we are using strings, which take into account upper/lower case differences.

We also stripped the sentences of stop words because they do not add any meaning, as the same stop words appear in many different sentences.

Additionally, we removed the punctuation because it does not add any meaning to the word analysis exercise we will do in this question.

We also lemmatized all the words because we are interested in knowing which class of words they belong to in order to understand the meaning of the sentence and not whether they are a noun, adjective, etc.

We finally removed words with the conjunction 'n't' because these are also meaningless words like 'isn't' and 'didn't'. They are basically stop words but are not caught by the stop word checker because of the extra 'n't'.

(c) Split training and testing set

We split the datasets into training and testing data using the head() and tail() functions. For each file, we used the first 400 instances for each label as the training data and the remaining 100 instances as testing data. We then concatenated all the training sets together and all the testing sets together, giving 2,348 reviews for training and 600 reviews for testing in total. We used the append() function for this.

(d) Bag of Words model

For this question we could not use the testing set to create the dictionary of unique words because the model needs to be created with the training set so that we can use the testing set as new data to test our model's ability to generalize. If we create the dictionary with the testing data, we are essentially using all the data as training data and would need to look for another set of new data to test the classifier.

To create the Bag of Words model, we first iterated through every sentence in the training set and stored it in a dictionary - of keys unique words and columns the word's counts - with count 0. Then we iterated through the training and testing sets and counted the number of occurrences of each word in the dictionary, updating the count value.

Using the dictionary, we then created one feature vector per review where the ith element of a review's feature vector is the number of occurrences of the ith dictionary word in that review.

Two feature vectors that we uncovered are reported in the file included, '2d.txt'.

(e) Pick postprocessing strategy

We decided to use log normalization as our postprocessing strategy to make data less skewed and focus on relative differences by bringing the feature vectors into the log space. We added the normalization step in the code for d) as we were constructing the feature vectors in order to reduce the number of iterations we need to go through the data.

We decided not to apply mean subtraction and division by standard deviation there are some standard deviations that are equal to zero because the data is sparse, resulting in a divide by zero error. Also, the number of dimensions (words) of the feature vectors is significant, so choosing L1 or L2 norm might result in similar distance measures for sentences with different words, so we discarded these techniques.

(f) Sentiment prediction

For this part, we created the Sentiment_Analysis() function. In this function, we used the existing libraries to train a Logistic Regression (LR) model on the training set and test it on the testing set. We then calculated its accuracy using the accuracy_score() function and created a confusion matrix using the confusion_matrix() function. We also repeated the same processing using a Gaussian Naive Bayes classifier instead. The classification accuracy of the LR classifier was 84.67%, and the classification accuracy of the Naive Bayes classifier was 79.16%. The resulting confusion matrices can be found on Figure 1.

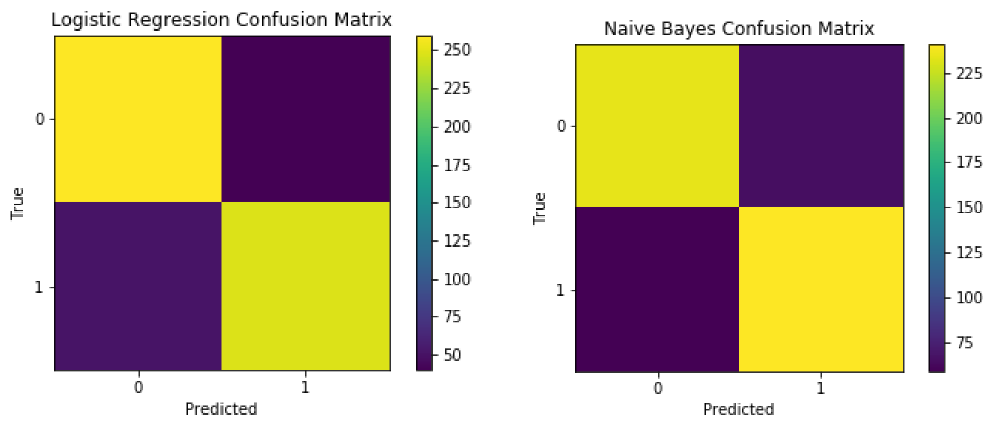


Figure 1: Logistic Regression Confusion Matrix (left) and Naive Bayes Confusion Matrix (right)

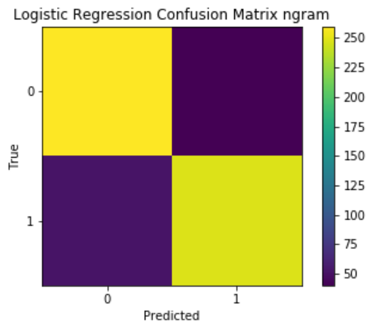
Inspecting the weight vector of logistic regression, we discovered that the following words had the most significant contribution because their weight was greater than 1: beautiful, delight, tender, realize, wonderful, delicious, fantastic, satisfy, perfect, awesome, soundtrack, struggle, 15, 1010, role, cinema, and sturdy.

Inspecting the weight vector of naive bayes, we discovered that the following words had the most significant contribution because their weight was greater than 0.18 (I couldn't use 1 here because there were not any words with weight greater than 1 because of how the weights are given in this model): place, service, food, good, like, go, time, one, get, bad, make, would, work, movie, film, and phone.

The logistic regression model did better and considered different words as most important compared to Naive Bayes.

(g) N-gram model

We then produced a function that would transform the data into N-grams (sections of N words) and convert these N-grams into a bag of words model. We created a 2-gram bag of words model where we took every pair of words in a sentence. For example, the sentence "Hi how are you" would have the 2-gram bag of words [Hi how, how are, are you]. Then we created feature vectors for this bag of words, performed the same post-processing strategy as before for the same reasons, and performed LR and Gaussian Naive Bayes. For LR, the classification accuracy was 69.6%. For Gaussian Naive Bayes, the classification accuracy was 66.7%. The confusion matrix for this is shown in Figure 2.



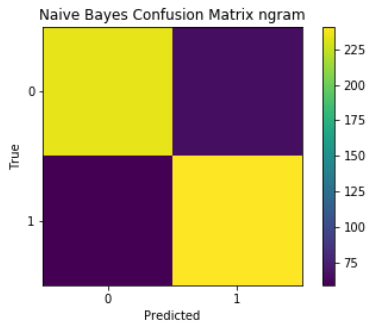


Figure 2: Logistic Regression Confusion Matrix (left) and Naive Bayes Confusion Matrix (right)

We again inspected the weight vectors.

For logistic regression we found that the most significant features were: 'really good', 'one best', 'great place', 'pretty good', 'highly recommend', 'long time', 'phone work', 'great product', 'best phone'. To find these we found weights greater than 0.9.

For naive bayes, after finding words with weight greater than 0.009, we found that the most significant features were: 'go back', 'ive ever', 'would recommend', 'waste time', 'one bad', 'see movie'.

Logistic regression still performed better, however, we can see some overlapping phrases between the two methods.

(h) PCA for Bag of Words model

For the PCA models, we decided to use mean subtracted data because we are using normalized eigenvectors as the pricipal components, so it makes sense to center the data around the mean. To implement PCA, we created a function named my_pca() that performs Singular Value Decomposition (SVD) on the train data and then finds the eigenvectors corresponding to the number of components desired in V. These are the principal components. Then, we found the projection of the train and test sets on these principal components by matrix multiplication. We used these results to show how lowering the dimensions of the data affects our analysis.

We used my_pca() to reduce both the training and testing sets to 10 dimensions first, then to 50 dimensions, and finally to 100 dimensions. We then performed Sentiment Analysis on these sets, using the Sentiment_Analysis() function described above, to train both a Logistic Regression and a Gaussian Naive Bayes (GNB) classifier on the 10-dimensional version of the data, the 50-dimensional version of the data, and finally the 100-dimensional version of the data. For the 10-dimensional reconstruction of the data the accuracy of the LR classifier was 59.5%, and the accuracy of the GNB classifier was 60%. For the 50-dimensional reconstruction of the data the accuracy of the LR classifier was 68.5% and the accuracy of the GNB classifier was 64.8%. For the 100-dimensional reconstruction of the data the accuracy of the LR classifier was 70.5%, and the accuracy of the GNB classifier was 63.8%. As you increase the number of components the better the classifiers get and the logistic regression classifier begins to perform better than GNB.

We also found the words that are the most significant in these models.

Model	PCA 10	PCA 50	PCA 100
Logistic Reg	place, late, may, bank, rick	place, late, bank, beautiful, cakeohhh, hole	cape, velvet, street, beer
GNB	love, holiday	love, cape	love, cakeohhh

Thus, we found that logistic regression wins with more components used in PCA and GNB performs better than logistic regression with less components. However, as you increase the number of components the accuracy of both models get better.

We performed the same thing for the 2-gram bag of words model.

For the 10-dimensional reconstruction of the data the accuracy of the LR classifier was 50.5%, and the accuracy of the GNB classifier was 49.8%. For the 50-dimensional reconstruction of the data the accuracy of the LR classifier was 51% and the accuracy of the GNB classifier was 53.8%. For the 100-dimensional reconstruction of the data the accuracy of the LR classifier was 58.2%, and the accuracy of the GNB classifier was 57.2%. As you increase the number of components the better the classifiers get. It is interesting that logistic regression performs better than GNB for all except 50 component PCA.

We also found the words that are the most significant in these models.

Model	PCA 10	PCA 50	PCA 100
Logistic Reg	bank holiday, rick steve,	selection menu, ravoli chickenwith,	selection menu, ravoli chickenwith,
	steve recommendation,	le interior,beer 23, server attentive	beer 23, server, attentive, good beef,
	selection menu		pita hummus, right sauce, service huge
GNB	stop late, may bank	stop late, cakeohhh stuff	stop late, le interior, hole wall

Thus, we found that logistic regression wins, unless you are at 50 dimensions. As you increase the number of components the accuracy of both models get better. However, for ngram the accuracy is lower than it was for just singular words. This is likely because the ngrams are not really independently acting.

We also see that there are some similarities among the words that influenced the model the most. However, a lot of them are very different.

(g) Algorithms comparison and analysis

The performance of logistic regression is always better than the Gaussian Naive Bayes for the above bag of words models. The best model was just simple bag of words. This gave a logistic regression accuracy of more than 80%. The N-gram model did not do nearly as well as this model and the PCA model also did not produce very desirable results. To start, one reason N-gram model did not work as well as the original bag of words because the highly weighted phrases were mostly positive. There was not a lot of recognition of negative reviews. Also, The N-grams make the searching space more sparse because they are creating more specific searches, so it is difficult for correct classification. Additionally, dimensionality reduction made the model worse because it may have capitalized on the variance due to the way people are writing the reviews but threw away the much of the variance that told whether the review was good or bad. We saw that logistic regression on top of PCA produced words that didn't have positive or negative connotations at all. Without PCA logistic regression weighted several positive and negative words highly.

We learned that the language people use in online reviews tends to be straight forward and specific and sometimes this can result in the reviews not having enough positive or negative connotations unless you look at distributions of single words. The writing expressions between people differ, increasing the variance of the data, so we needed to find a sweet spot with PCA to not eliminate too much of the meaning when stripping away the variance due to different writing styles. We also learned that lemmatizing the sentences is crucial, since there are many expressions that have similar meanings and our calculations would have probably not converged if we had not lemmatized the words.

1.2 Clustering for Text Analysis

(a) Document-wise clustering

For this question, we first read and stored the data in the 'science2k-doc-word.npy' file using the load() function in the numpy library [1], and the data in the 'science2k-titles.txt' file using the open() function. We checked the correct dimensions of the arrays in which the data was stored.

Next, for all the possible values of K in the interval from K = 2 to K = 20, we performed K-means [2] and obtained the silhouette score and the inertia score, which are the two metrics that help us to identify the best K. Figure 3 shows a plot of each k value and its silhouette score. Figure 4 shows a plot of each k value and its inertia score.

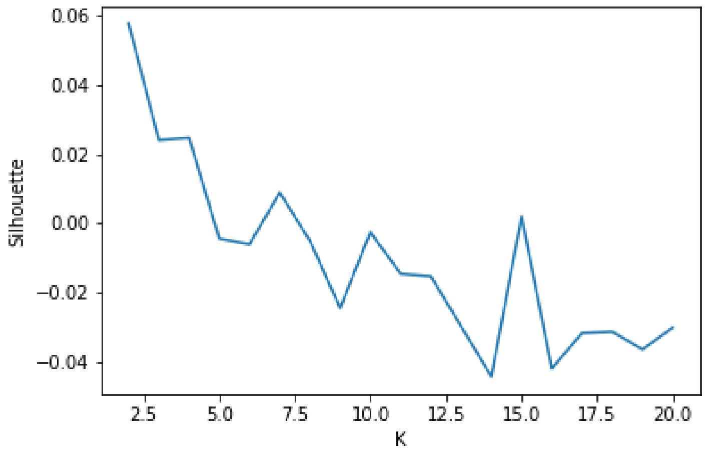


Figure 3: Silhouette scores for different values of k

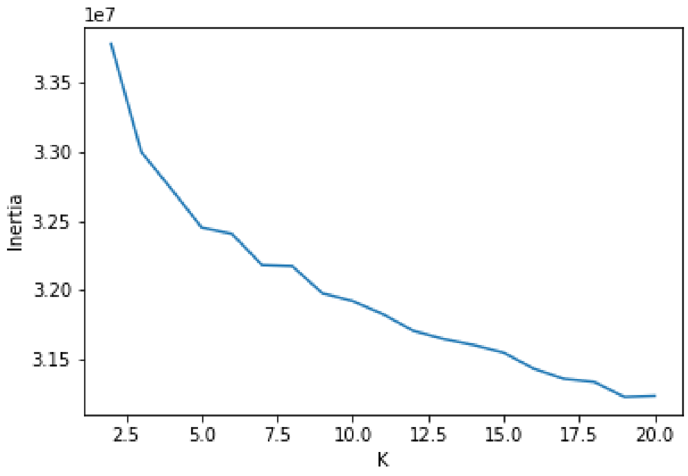


Figure 4: Inertia scores for different values of k

The silhouette score is based on the mean intra-cluster distance and the mean nearest-cluster distance for each sample, a higher silhouette score means that the model is performing better. It is a good metric because it takes into account how similar the points are to their cluster center and how different the points are to other clusters. The inertia value is the sum of squared distances of samples to their closest cluster center.It decreases as we increase the value of K. The best value of k is when you have an "elbow" in the intertia curve, because that is indicating that you have gotten a distance that is close enough to the cluster center. Since the elbow could not be appreciated clearly for this example, we selected the best K as the one that maximized the silhouette score. In this case, the best K was K = 2.

We performed K-means with K = 2, and found the top 10 words in each cluster with respect to the distance for the average over all points. To do this, we found the words associated with the top 10 values of the vector of cluster centers minus the 5467-vector of average values across documents. Additionally, we found the top ten documents that fall closer to each individual cluster center by checking the distance of each document in a cluster to the center of that cluster and finding the top 10 closest ones. The results of this analysis were:

Top ten words in the first cluster:

- expressed
- specific

- binding
- fig
- gene
- proteins
- expression
- cells
- cell
- protein

The top ten words in the second cluster:

- states
- focus
- million
- says
- researchers
- scientists
- world
- field
- year
- years

Top ten documents in the first cluster:

- “A Time for Restraint”
- “Sex Determination in Malaria Parasites”
- “A Potassium Channel Protein Encoded by Chlorella Virus PBCV-1”
- “Cool Glacial Temperatures and Changes in Moisture Source Recorded in Oman Groundwaters”
- “Population Dynamical Consequences of Climate Change for a Small Temperate Songbird”
- “How Climate Change Alters Rhythms of the Wild”
- “Green, Catalytic Oxidation of Alcohols in Water”
- “Giant Birefringent Optics in Multilayer Polymer Mirrors”
- “Oxygen Isotopes and Emerald Trade Routes since Antiquity”
- “Three-Dimensional Direct Imaging of Structural Relaxation near the Colloidal Glass Transition”

Top ten documents in the second cluster:

- “Language Discrimination by Human Newborns and by Cotton-Top Tamarin Monkeys”
- “Thermal, Catalytic, Regiospecific Functionalization of Alkanes”
- “Crossed Nanotube Junctions”
- “Lunar Impact History from 40Ar/39Ar Dating of Glass Spherules”
- “Seeing a World in Grains of Sand”

- "Predictions of Biodiversity Response to Genetically Modified Herbicide-Tolerant Crops"
- "Control of Thickness and Orientation of Solution-Grown Silicon Nanowires"
- "Cholera Dynamics and El Nino-Southern Oscillation"
- "DNA Damage-Induced Activation of p53 by the Checkpoint Kinase Chk2"
- "Saving Venice"

We can see that the top 10 words selected for the first cluster are related to the topic of technical and specialized concepts in science more strongly than the worlds selected for the second cluster. Therefore, the algorithm has managed to identify the most characteristic words in the compilation of science documents, and split the dataset into specialized vs not specialized. In regards to the documents, although the algorithm has grouped many of the titles in climate change and chemical elements on the first cluster compared to the second cluster, this distinction is not as evident as the one presented above for the words. Thus, this algorithm would be useful to understand what the most characteristic words are in the dataset by looking at the words furthest from the average. It could also be useful to separate the most predominant subtopic in a specific topic by looking at the document title selection.

(b) Term-wise clustering

For this section, we read the data from the 'science2k-word-doc.npy' file into a numpy array [1] using the open() function. We then repeated the process explained in (a), obtaining the silhouette at different values of K for K-means. The results can be found on Figure 5. By looking at the maximum silhouette score, we found

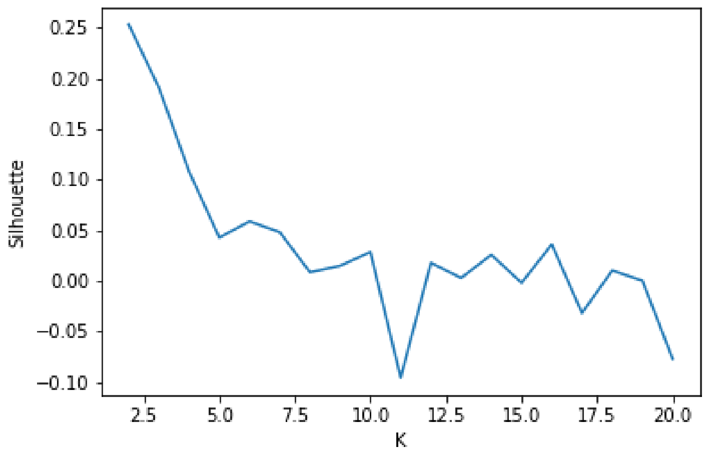


Figure 5: Silhouette scores for different values of K

the best K to be K = 2. We found the top 10 documents in each cluster with respect to the distance for the average over all points. We then reported the top ten words for each cluster. The results of this were:

Top ten documents in the first cluster:

- "Role of the Mouse ank Gene in Control of Tissue Calcification and Arthritis"
- "Function of PI3Kg in Thymocyte Development, T Cell Activation, and Neutrophil Migration"
- "Signaling and Circuitry of Multiple MAPK Pathways Revealed by a Matrix of Global Gene Expression Profiles"
- "Kinesin Superfamily Motor Protein KIF17 and mLin-10 in NMDA Receptor-Containing Vesicle Transport"
- "Central Role for G Protein-Coupled Phosphoinositide 3-Kinase g in Inflammation"

- “The Genome Sequence of Drosophila melanogaster”
- “An Oral Vaccine against NMDAR1 with Efficacy in Experimental Stroke and Epilepsy”
- “Regulated Cleavage of a Contact-Mediated Axon Repellent”
- “Noxa, a BH3-Only Member of the Bcl-2 Family and Candidate Mediator of p53-Induced Apoptosis”
- “Positional Syntenic Cloning and Functional Characterization of the Mammalian Circadian Mutation tau”

Top ten documents in the second cluster:

- “Bastions of Tradition Adapt to Alternative Medicine”
- “NIH, under Pressure, Boosts Minority Health Research”
- “Soaking up Carbon in Forests and Fields”
- “Science Survives in Breakthrough States”
- “Controversy Flares up over NASA Solar Project”
- “Flushing out Nasty Viruses in the Balkans”
- “National Academy of Sciences Elects New Members”
- “Soft Money’s Hard Realities”
- “A New Breed of Scientist-Advocate Emerges”
- “Ground Zero: AIDS Research in Africa”

Top ten words in the first cluster:

- different
- shown
- org
- three
- mice
- control
- two
- www
- cells
- fig

Top ten words in the second cluster:

- model
- expression
- observed
- first
- data
- sciencemag
- science

- start
- www
- protein

We can see that the documents selected for the first cluster have more technical and specialized scientific content than the ones selected for the second cluster, so the algorithm was able to make a clear split between these two groups. Regarding the words, a clear difference between the ones selected for the first cluster and for the second cluster cannot be observed. Some words like ”www” or ”first” barely bring any meaning. Thus, this algorithm would be useful to identify the most specialized document titles within the dataset.

From this, we can conclude that clustering terms is more useful for splitting the dataset in terms of document title categories, whereas clustering documents is more useful for splitting the dataset in terms of word categories.

1.3 EM Algorithm and Implementation

For this question we used the python libraries numpy, sklearn, pandas, and matplotlib [1], [2], [3], [4].

(a) The Alternating Algorithm for k-means is a Special Case of EM

The alternating algorithm for k-means is a special case of EM. The KMeans algorithm starts by initializing the means of the cluster to be some random value in the data set. Similarly, the EM algorithm starts by initializing the means of the clusters to some random value in the data set (or based on KMeans) and by initializing the covariances and weights of the data set. If we say that the GMM/EM fitting uses spherical covariance matrices ($\sigma^2 I$), then driving σ^2 or variance to 0, we have the KMeans algorithm, producing hard assignments.

Since KMeans does not deal with weighting and covariances because it seeks to produce hard assignment, the objective function for the E step is just the current assignment of all of the points.

Then, KMeans seeks to reduce the distance between the points and their cluster center so the objective function for the M step is as follows where x_i is a point and m_k is a cluster center:

$\arg\min \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - m_k||^2$

Basically, we are finding the cluster centers for x_is that minimize the distance of points to their cluster centers.

(b) Parse Data Set

We parsed the data set ignoring the descriptions at the beginning and put it into a Pandas DataFrame. Then, we plotted the data eruptions vs. waiting Figure 6.

(c) Random GMM

(i) We used the sklearn GMM model with random initialization, spherical covariance, and warm_start. We set max_iter to 1 so that every single time that we ran a fitting it only performed 1 iteration of EM. Additionally, we used warm_start = True so that, as we looped through iterations of fitting using gmm.fit until convergence was reached, the model will use previously stored initialization parameters. We plotted the mean vectors on the clusters to show how the convergence occurred. The green vectors are for the first component and the purple are for the second component Figure 7. We can see that both vectors started out kind of in the middle of the plot and moved towards the cluster centers. We continued iterating until the gmm.converged_ parameter returned true (i.e. when the means stopped changing). When the means stop changing is a good termination criteria because this only occurs when there is no better cluster center and we have found the local optimum.

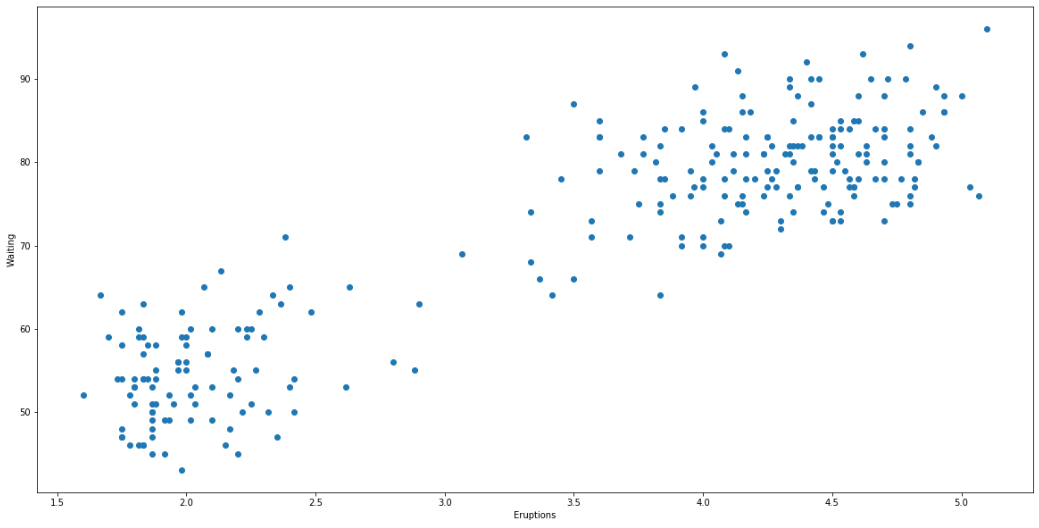


Figure 6: Eruptions vs. Waiting Plot

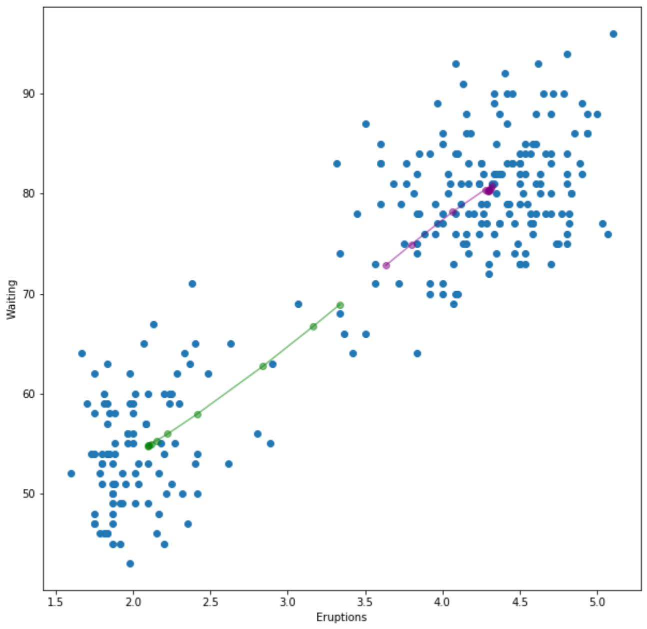


Figure 7: Plot Mean Trajectories by iteration until converge

(ii) We then ran the sklearn GMM model with the default max_iter, random initialization, and spherical covariance 50 times and plotted a histogram of the number of iterations until the model converged. It was interesting that it took few iterations to converge for a lot of iterations and between 8-11 for many iterations Figure 8.

(d) KMeans GMM

(i) We used KMeans with k=2 to determine the initial means and spherical covariance matricies for the GMM. Since the sklearn GMM function takes means, precisions, and weights, we took the inverse of the covariances for

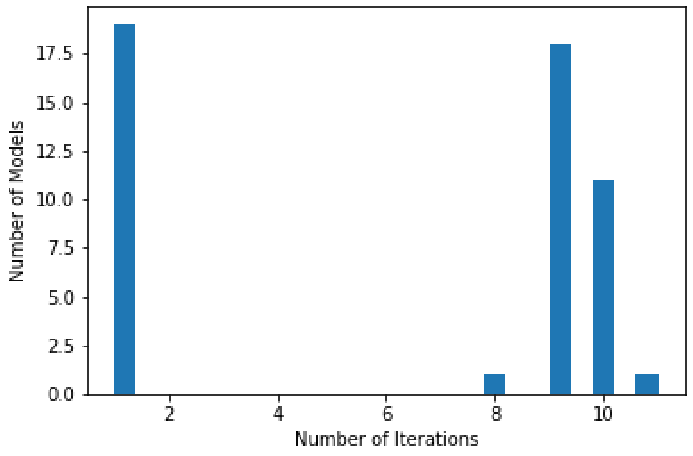


Figure 8: Histogram of Number of Iterations

precisions and we took the fraction of data samples in the kmeans clusters for weights. We again set max_iter=1 and warm_start=True, for the same reason as bofore, to be able to capture each iteration until convergence. Similarly to before, we used the converged_ attribute of the sklearn gmm model as our termination criterion, for the same reason. We again plotted the mean vectors on the clusters to show how the convergence occurred. The green vectors are for the first component and the purple are for the second component Figure 9. We see that this looks very similar to the random case. This could indicate that KMeans did not do an extremely accurate initialization or sklearn's GMM with 'random' parameters is not actually entirely random.

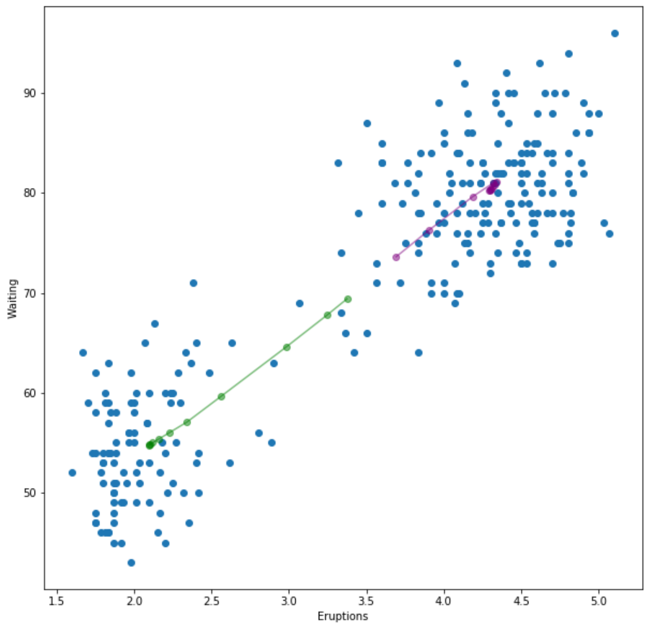


Figure 9: Plot Mean Trajectories by iteration until converge Kmeans-GMM

(ii) We then ran the sklearn GMM model with the default max_iter, our KMeans initializations, and spherical covariance 50 times and plotted a histogram of the number of iterations until the model converged Figure 10. Each time the gmm model was created, a new set of initializations were placed in the model. We can see that unlike the random gmm, this model always takes 6 iterations to converge.

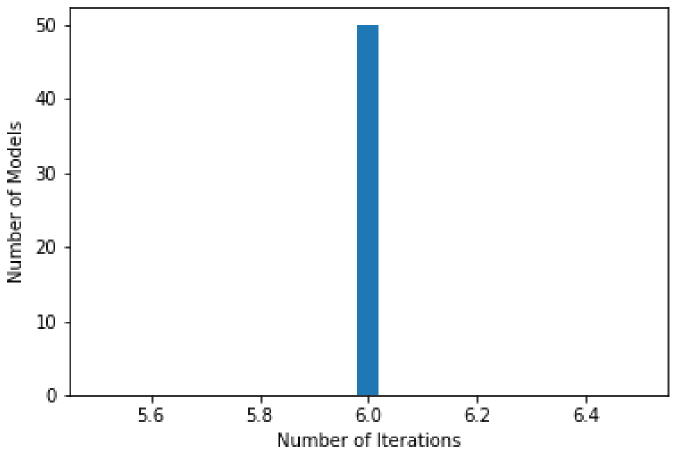


Figure 10: Histogram of Number of Iterations for KMeans

Comparing the algorithm performances of (c) and (d), we can see that both algorithms converge to the same or a very similar cluster center and in a similar trajectory since they start out in about the same location. We found that the random initialization can take very few tries to converge, or it can take very many tries to converge. However, KMeans with k=2 will always take 2 tries to converge.

1.4 Multidimensional Scaling for Genetic Population Differences

(a) Using multidimensional scaling (MDS) to coerce D into a 2-dimensional vector representation

To read the data to analyze for this question, we used the code provided in the homework description document.

(i) For this part we used metric Multidimensional Scaling (MDS). Metric MDS assumes that the data can be represented by vectors along the set of dimensions indicated and that the relationship between the distance and the dissimilarity is inversely proportional. It also assumes the data lives in a low-dimensional manifold. This could cause problems for the situation where the data is clumpy instead of manifoldy. It could also cause problems if, for instance, the principal directions of variation for a dataset are the first 3 dimensions, but we choose to only consider the first 2 dimensions for MDS.

One way to measure how much information is being lost would be to apply MDS for the largest number of dimensions that we are willing to consider, then perform Singular Value Decomposition (SVD) on the resulting data to find the dimensions of greatest variation. Based on this, we can optimize the number of dimensions that best represents the data and apply MDS again taking it into account.

(ii) For this part, we decided to perform 20-dimensional MDS on the data and check the eigenvalues of the result by performing SVD. We used the existing MDS() function in sklearn [2] for this. We then performed SVD using the linalg.svd() function in the library scipy [5]. Looking at the singular values in S (covariance matrix) obtained from SVD, we decided to consider the first 5 dimensions as the ones which give an accurate low-dimensional representation of the data because they were powers of 100. The rest of singular values were powers of 10 and 1, meaning they are much smaller and insignificant compared to the powers of 100. We therefore used this finding to apply MDS again with 5 dimensions instead.

(iii) We applied 2-dimensional MDS on the data. The results can be found on Figure 11.

(b) K-means on 2D embedding

To select the best K, we first performed k-means on the data for K = 2, 3, 4 and 5. We obtained the inertia value for each K value, and plotted it on Figure 12. The ”elbow” of this plot is at K = 6. This is the point

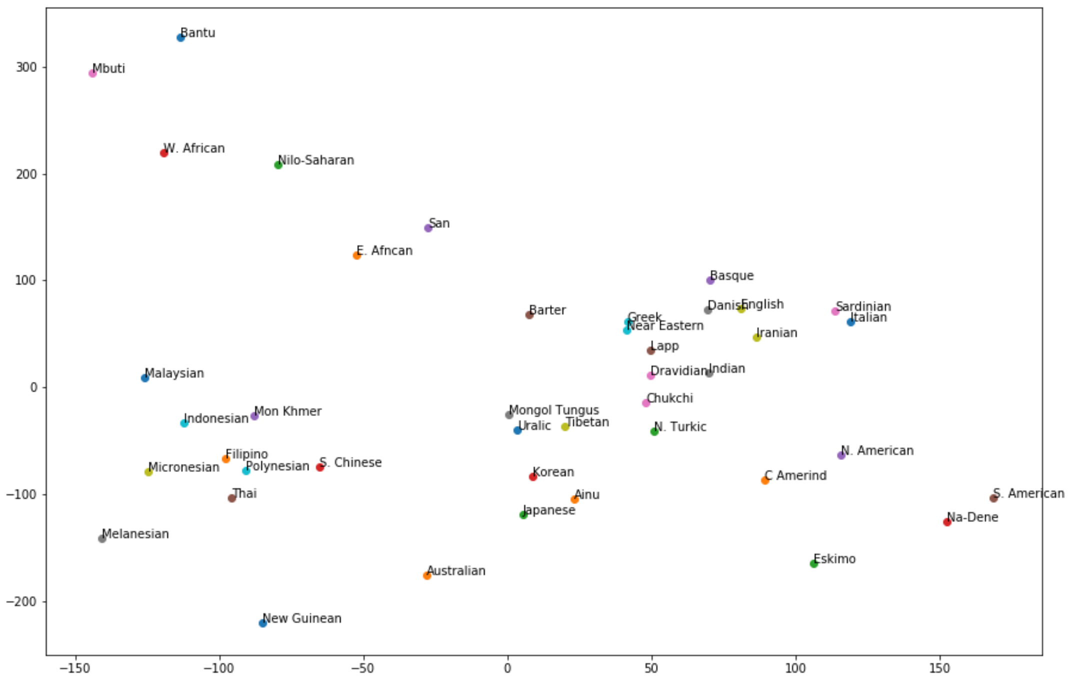


Figure 11: 2-dimensional MDS results

at which we have achieved the best sum of square distance from the cluster center that we can before overfitting. We also found the silhouette score for each value of K. This can be observed on Figure 13. The silhouette scores were all quite high, which made it difficult to choose K. The silhouette plot peaks at K = 2, however, it also peaks similarly at k = 6 like the elbow of the inertia plot tells us. From visually inspecting the data, however, it looked like the data does not fall into 6 classes, so the 2-class class seems more plausible. Therefore, we chose K = 2 as the best K, and run K-means on the data. the K-means clustering results can be found on Figure 14.

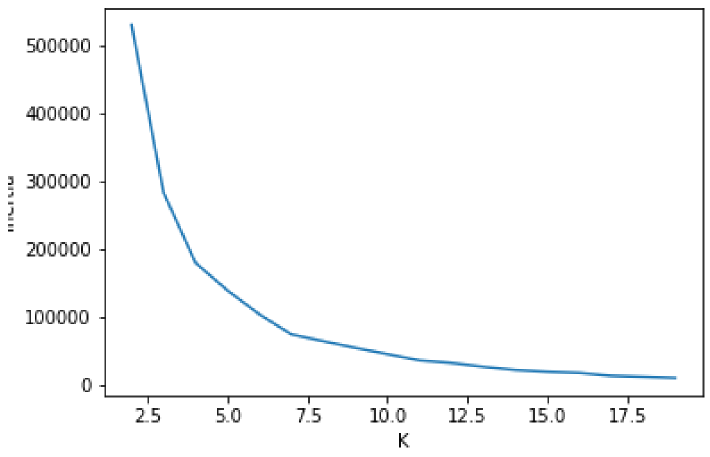


Figure 12: Plot of the inertia score for each value of K

When choosing K = 6, it looks like the clusters cluster by region on Earth pretty well. However, there are

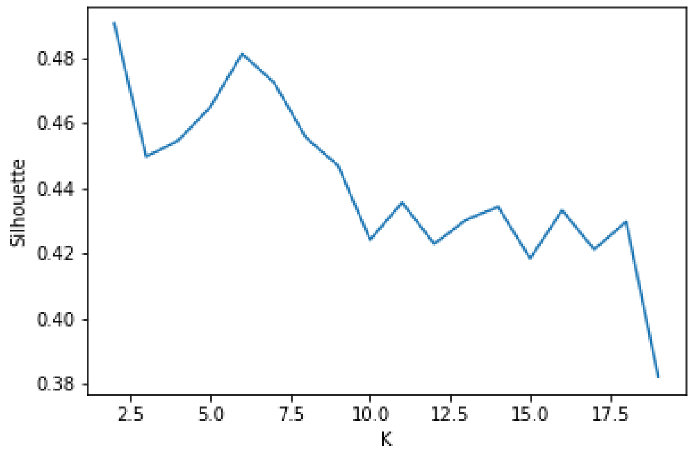


Figure 13: Plot of the silhouette score for each value of K

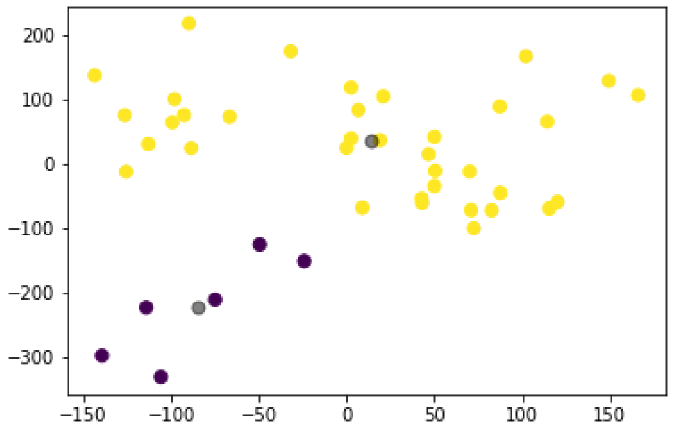


Figure 14: K-means clustering results with best K (K = 2)

some discrepancies such as Indians and Dravidians being clustered with Europeans and Turkish being clustered with some of the East Asian countries. The algorithm does a good job of separating out Americas, Africa, and countries around Australia, but the rest are mixed together, which is likely an effect of the dimensionality reduction.

When choosing K = 2, we do not fully agree with the clustering results because it also looks like there could be 3 clusters, or even more. However, the clustering does a good job of separating the African Countries from the rest of the data. The dimensionality reduction has caused the data to be in a more compacted space impacting how we cluster. The information about what features the distance is being calculated on is vital for understanding and representing the data, as well as for figuring out if the clusters make sense.

(c) Comparing hierarchical clustering with K-means

To complete this question, we used the function cluster.hierarchy() in the library scipy [5]. We obtained a linkage for the data, and used it to obtain the dendrogram. We also created a dictionary with the data labels in order to add labels to the dendrogram. Figure 15 shows the results.

Next, we turned the resulting tree into a flat clustering of points using the fcluster() function in the scipy hierarchy library mentioned above. After trying different thresholds, we found a threshold of t = 601.38 to give the split of the data into two clusters that is closest to the K-means split. The resulting cluster split can be

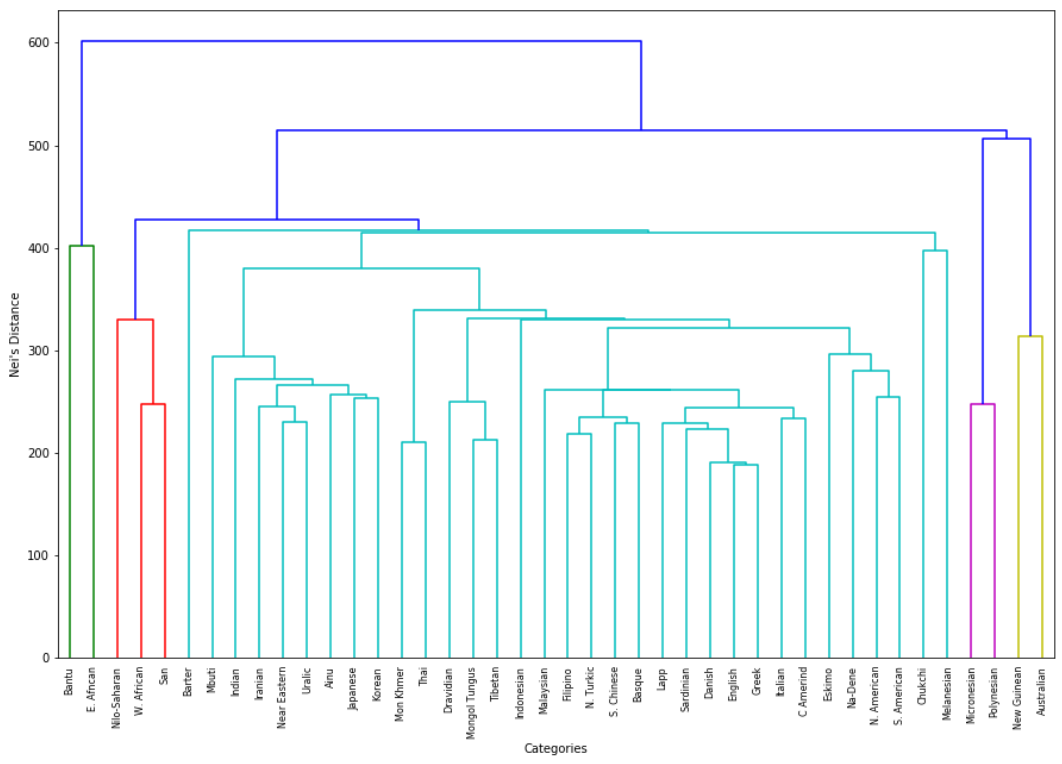


Figure 15: Population dendogram using Nei's distance

observed in Figure 16.

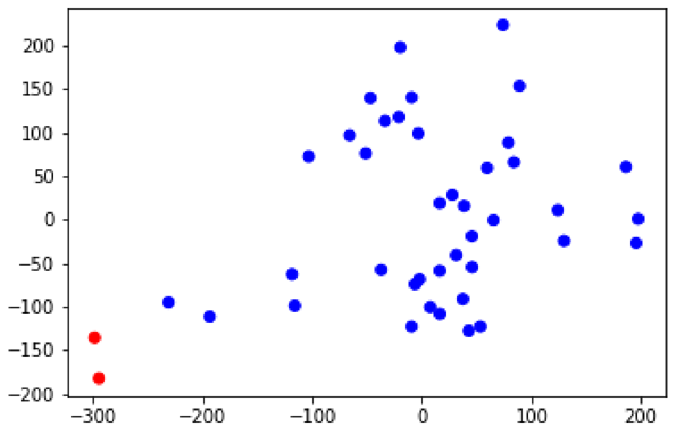


Figure 16: Flat clustering results with a threshold of t = 601.38

Compared to the K-means clustering computer earlier, this technique groups more points into the second cluster. The first cluster in this case only contains 2 points, as opposed to the first cluster in K-means clustering which contains 6 points. Therefore, the K-means clustering results look more realistic and efficient.

(d) Comparing K-medoids with K-means

For this question, we used the k-medoids() function in the pyclustering library [6] to apply k-medoids to the original distance matrix. The results of the clustering are shown on Figure 17.

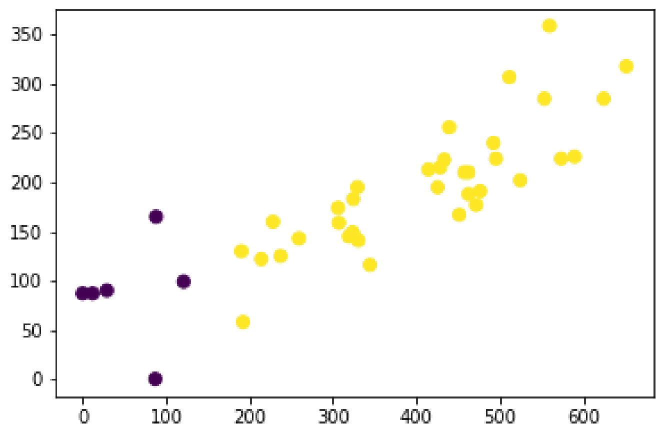


Figure 17: K-medoids clustering of the data using the original distance matrix

We can see that the K-medoids points are less spread in the space than the K-means points. The split into clusters is similar, with the first cluster in both graphs including 6 points that are close to the origin of the graph. However, the 2 clusters in the K-medoids graph are less easily differentiable than the clusters in the k-means graph. Overall, it seems like K-means is a better method for this particular task.

2 Written Exercises

2.1 Decision Trees

(a)

We are replacing each split with a leaf labeled with the more frequent class. Since there are 2 classes, positive and negative, we know that one leaf node is $argmax(p_1, n_1)$ and the other leaf node is $argmax(p_2, n_2)$. This is just saying that the leaf nodes are labeled as p or n depending on which is more frequent.

From this information we can deduce that the number of training mistakes is equivalent to:

$min(p_1, n_1) + min(p_2, n_2)$

Let's look at the weighted impurity function:

$(p_1 + n_1)I(\frac{p_1}{p_1+n_1}) + (p_2 + n_2)I(\frac{p_2}{p_2+n_2}) = (p_1 + n_1)\min\{\frac{p_1}{p_1+n_1}, 1 - \frac{p_1}{p_1+n_1}\} + (p_2 + n_2)\min\{\frac{p_2}{p_2+n_2}, 1 - \frac{p_2}{p_2+n_2}\}$

Since p and n are the only two classes we can write:

$(p_1 + n_1)I(\frac{p_1}{p_1+n_1}) + (p_2 + n_2)I(\frac{p_2}{p_2+n_2}) = (p_1 + n_1)\min\{\frac{p_1}{p_1+n_1}, \frac{n_1}{p_1+n_1}\} + (p_2 + n_2)\min\{\frac{p_2}{p_2+n_2}, \frac{n_2}{p_2+n_2}\}$

We can pull out the denominators to the respective coefficients and we have:

$(p_1 + n_1)I(\frac{p_1}{p_1+n_1}) + (p_2 + n_2)I(\frac{p_2}{p_2+n_2}) = \min\{p_1, n_1\} + \min\{p_2, n_2\}$

We have arrived back at what we though the number of training mistakes was.

QED

(b)

$P(a_1 = 0) = 4/10$
$GiniIndex(a_1 = 0) = 1 - ((2/4)^2 + (2/4)^2) = 0.5$
$P(a_1 = 1) = 6/10$
$GiniIndex(a_1 = 1) = 1 - ((1/6)^2 + (5/6)^2) = 0.278$
$GiniWeighted(a_1) = (4/10)0.5 + (6/10)0.278 = 0.367$

$P(a_2 = 0) = 4/10$
$GiniIndex(a_2 = 0) = 1 - ((1/4)^2 + (3/4)^2) = 0.375$
$P(a_2 = 1) = 6/10$
$GiniIndex(a_2 = 1) = 1 - ((2/6)^2 + (4/6)^2) = 0.444$
$GiniWeighted(a_2) = (4/10)0.375 + (6/10)0.444 = 0.416$

$P(a_3 = 0) = 7/10$
$GiniIndex(a_3 = 0) = 1 - ((3/7)^2 + (4/7)^2) = 0.49$
$P(a_1 = 1) = 3/10$
$GiniIndex(a_3 = 1) = 1 - ((0)^2 + (1)^2) = 0$
$GiniWeighted(a_3) = (7/10)0.49 + (6/10)0 = 0.343$

We see that the minimum Gini Index is 0.343 (smallest) at a_3. Thus, the split that we will use for the root node when using Gini Index is a_3.

$I(a_1)$:
$a_1 = 0$, y = '+' = 2, y = '-' = 2
$a_1 = 1$, y = '+' = 1, y = '-' = 5
$WeightI(a_1) = min\{2, 2\} + min\{1, 5\} = 2 + 1 = 3$

Similarly, we do the same with the others...

$WeightI(a_2) = min\{1, 3\} + min\{2, 4\} = 2 + 1 = 3$

$WeightI(a_3) = min\{3, 4\} + min\{0, 3\} = 3 + 0 = 3$

Since all attributes have the same min-error impurity we can choose the root node split at random. For example, we could choose a_1.

(c)

We need to fulfill the condition:

weighted impurity after splitting < impurity before splitting

Before splitting the impurity is as follows:

$(p_1 + p_2 + n_1 + n_2) * \min\{\frac{p_1+p_2}{p_1+p_2+n_1+n_2}, \frac{n_1+n_2}{p_1+p_2+n_1+n_2}\}$

We can simplify this by moving the denominators out to the coefficient.

$(p_1 + p_2 + n_1 + n_2) * \min\{\frac{p_1+p_2}{p_1+p_2+n_1+n_2}, \frac{n_1+n_2}{p_1+p_2+n_1+n_2}\} = \min\{p_1 + p_2, n_1 + n_2\}$

From (a) we found that the weighted impurity after splitting is equal to $\min\{p_1, n_1\} + \min\{p_2, n_2\}$.

We need to find conditions where:

$\min\{p_1, n_1\} + \min\{p_2, n_2\} < \min\{p_1 + p_2, n_1 + n_2\}$

We know that the above expression fails if $\min\{p_1, n_1\} + \min\{p_2, n_2\} = p_1 + p_2$ or $\min\{p_1, n_1\} + \min\{p_2, n_2\} =$

$n_1 + n_2$. This is because it is no longer an inequality, but an equivalence. Thus, it must be either be true that $p_1 > n_1$ and $n_2 > p_2$ or that $p_1 < n_1$ and $n_2 < p_2$.

Thus, we can conclude that $p_1 + n_2 \neq p_2 + n_1$. Additionally, we need $n_1 \neq p_1$ and $n_2 \neq p_2$ to ensure that the inequality remains less than.

(d)

The min-error impurity is not effective in growing decision trees. This is made clear in (b), when it did not give a choice of attribute to start with when selecting a split. Then, in (c) we saw that we need to impose several unreasonable conditions for it to reduce impurity as it goes from the root to the nodes.

2.2 Bootstrap Aggregation

The probability of each example in the set is given by $\frac{1}{N}$, so all examples have an equal probability of appearing. Given that we have N examples, the probability of choosing any one example is given by $(\frac{1}{N})^N$. Therefore, the probability of not choosing an example is given by $(1 - \frac{1}{N})^N$.

We can now calculate $\lim_{N\to\infty}(1 - \frac{1}{N})^N$.

To calculate this limit, we take the natural logarithm to simplify the limit and then reverse it. First, taking the natural logarithm of the limit we obtain:

$ln(\lim_{N\to\infty}(1 - \frac{1}{N})^N) = \lim_{N\to\infty} ln((1 - \frac{1}{N})^N) = \lim_{N\to\infty} N ln(1 - \frac{1}{N}) =$

$\lim_{N\to\infty} \frac{ln(1-\frac{1}{N})}{\frac{1}{N}}$ Applying L'Hopital rule we get: $\lim_{N\to\infty} \frac{ln(1-\frac{1}{N})}{\frac{1}{N}} = \lim_{N\to\infty} \frac{\frac{1}{1-\frac{1}{N}} \frac{-1}{N^2}}{\frac{-1}{N^2}} = \lim_{N\to\infty} \frac{-N}{N-1} = -1.$

Second, reversing the natural logarithm to find the result x we get:

$ln(x) = -1$ therefore $x = \frac{1}{e}$

This gives:

$\lim_{N\to\infty}(1 - \frac{1}{N})^N = \frac{1}{e}$ Q.E.D

References

[1] T. E. Oliphant, *A guide to NumPy.* Trelgol Publishing USA, 2006, vol. 1.

[2] A. G. e. a. Fabian Pedregosa, Gaël Varoquaux, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, 2011.

[3] W. McKinney, "Data structures for statistical computing in python," *Proceedings of the 9th Python in Science Conference*, pp. 51–56, 2010.

[4] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science Engineering*, 2007.

[5] E. Jones, T. Oliphant, P. Peterson *et al.* (2001) SciPy: Open source scientific tools for Python. "http://www.scipy.org/"[Accessed 2019-06-10].

[6] Lagrange multiplier. "https://pypi.org/project/pyclustering"[Accessed 11-13-2019].