CMU 10-715: Homework 7 Report

Decision Trees and Unsupervised Learning

Abishek Sridhar (Andrew Id: abisheks)

1 Decision Trees

1.1 Results

a Refer figure 1 and table 1 for the k-fold cross-validation results (for k = 3).

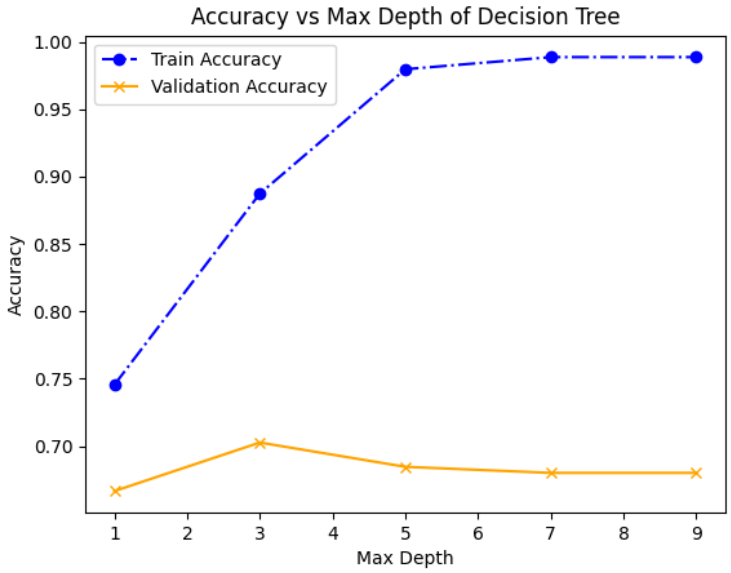


Figure 1: Train and Validation Accuracy vs Max Depth of the Decision Tree

Max Depth	Average Train Accuracy	Average Val Accuracy
1	74.55	66.67
3	88.74	70.27
5	97.97	68.47
7	98.87	68.02
9	98.87	68.02

Table 1: Table showing quantitative results of the k-fold cross-validation (with k = 3)

b We choose best max-depth as 3 since it gives rise to the maximum validation accuracy as observed in figure 1. Refer table 2 for the results of training on the complete training set with max-depth 3.

Max Depth	3
Train Accuracy	85.58
Validation Accuracy	67.27

Table 2: Table showing the results of training on whole training data after choosing the best max depth of 3 from the k-fold cross-validation experiment

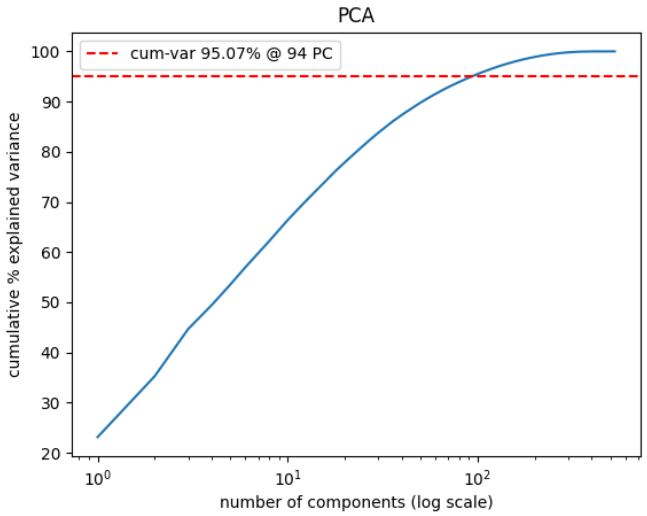
c We observe that the train accuracy increases on increasing the max depth of the decision tree built. This is expected because increasing the maximum depth of decision tree causes it to perform increasingly better on the training data by reducing the possibilities of majority votes and splitting on more features to account for every possible difference in labels among the training samples.
The validation performance however increases, then decreases, and then almost remains the same with increasing max depth. Initially increasing the max depth from 1 to 3 does lead to an increasing expressivity of the decision tree by splitting on more than one feature, but further increasing the max depth and fitting to the training set leads to overfitting and consequent reduction in generalization performance (measured in terms of average validation accuracy).

2 Linkage Based Clustering

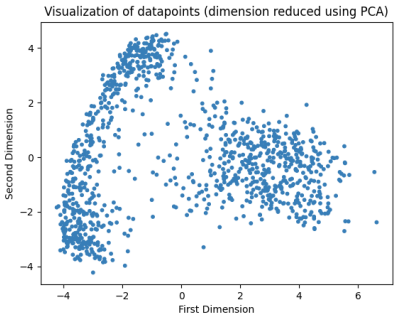
2.1 Results

a I initially pre-processed the data by noticing that quite a few of the columns had no variation in its column values across samples. So I dropped the 256 columns (out of 784) that had (max value - min value) to be less than ϵ (where $\epsilon = 1e - 8$). Following this, I plotted a scatter plot of the 1056 data points by reducing their dimensions to 2 using PCA and obtained plot 2. We can notice roughly about 2 clusters seeming to be present from the PCA scatter plot on the LHS of row 2 in 2. But since the explained variance with 2 components was very less (35.28%) to go by it completely, I decided to also use TSNE dimensionality reduction to visualize the data points. There was an additional reason to motivate the use of TSNE: the number of principal components to explain 95% of the variance in PCA was about 94 (total features being 528), which might indicate the points lie in fairly curved or complex manifolds that require a non-linear dimensionality reduction like TSNE to visualize. From the

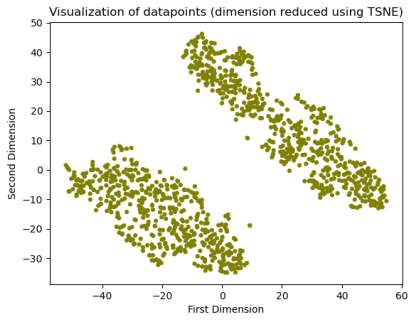
figure 2, I again observed 2 clusters, which validates the observation from 2-component PCA.



(a) PCA on the data points



(b) Visualization of data points using PCA (on 2 components)



(c) Visualization of data points using TSNE (on 2 dimensions)

Figure 2: Exploratory analysis on the data points

To choose the points distance metric and cluster distance metric, I ran the Linkage Clustering Algorithm for the three possible distance functions, three possible cluster distances and for number of clusters = 2. The intuitions I had prior to visualizations were:

- **Regarding Distance Function:** The feature values of all data

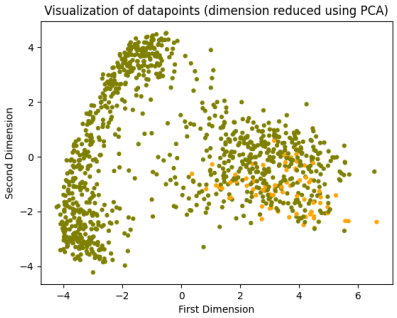
points were in the range of 0 to 1, for all features and no meta information about the features were provided. Manhattan and euclidean distance sort of scales with the dimensions of the datapoints and cosine is generally known to perform better than them for high dimensional vectors, since at very high dimensions the alignment of the vectors tell more information than the actual distance between them. Especially here, we can notice from output in figure 3 that the data point vectors' magnitudes don't vary too considerably (considering the high dimensions), making cosine a marginally better choice in my opinion. Manhattan distance always lags behind euclidean for small-valued vectors, so i expect euclidean to be the next best choice, if at all.

- **Regarding Cluster Distance:** The three linkage modes have different properties: Single (or minimum) linkage tries to form long chained clusters since two points within a cluster can have completely different distances; Complete (or maximum) linkage forms compact clusters since it attempts to keep two points in a cluster within a ball of maximum distance though the ball's internal space can be sparse; Average linkage can form arbitrarily shaped close-knit clusters. Though the PCA plot gives the impression of one long chain and one round cluster, it is not representative of the true points distribution to a great extent given its small explained variance. Generally in higher dimensions, long chain clusters are not very common, especially when the number of data points (1056 here) are not much larger than their dimensions (528). Also, from the earlier observation about vector magnitudes not varying much, I expect the points to form ball-shaped or compact clusters in high dimensions, and feel complete linkage might be slightly better than average linkage (since we ruled out single linkage first).

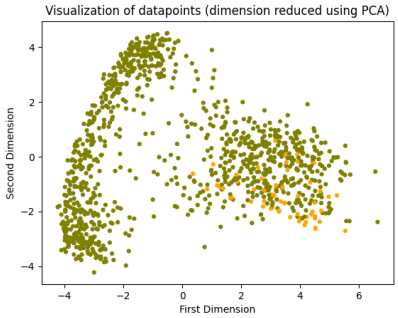
```
Min of column values: 0.0   Max of column values: 1.0
Min of point vector magnitudes: 4.140
Max of point vector magnitudes: 13.174
Mean of point vector magnitudes: 8.042
Variance of point vector magnitudes: 2.395
```

Figure 3: Program output for statistics on the features values and data point vector magnitudes

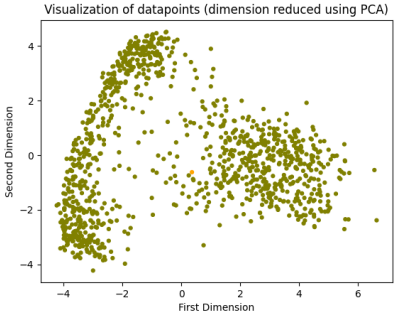
I am attaching the output visualizations of clusters formed by average, complete, and single linkage when set to form 2 clusters and with cosine, euclidean distances in figure 4. I noticed that the distance function almost completely did not matter when forming 2 clusters (I verified this with manhattan distance as well, though the plots are not attached). One



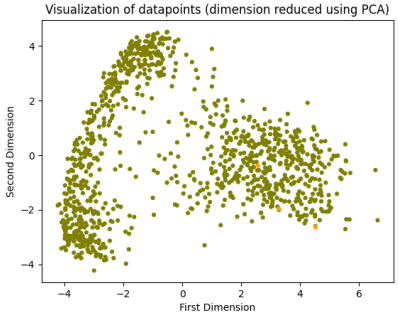
(a) Complete Linkage, Cosine Distance



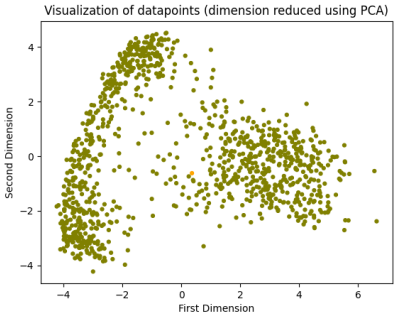
(b) Complete Linkage, Euclidean Distance



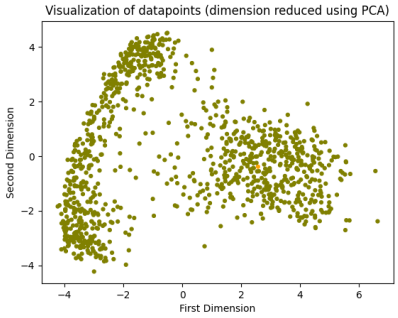
(c) Average Linkage, Cosine Distance



(d) Average Linkage, Euclidean Distance



(e) Single Linkage, Cosine Distance



(f) Single Linkage, Euclidean Distance

Figure 4: Visualization of clustering on data points with no. of clusters = 2

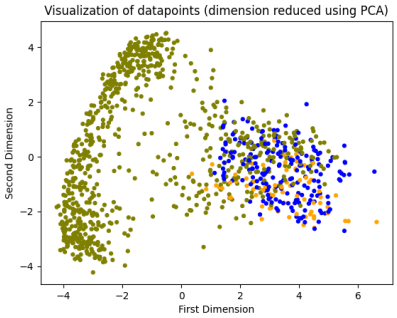
main observation was that max linkage seems better at identifying the

two clusters as we expected, while the other two linkages formed highly skewed pair of clusters. This might indicate the presence of outliers or a tiny set of points from the tail of the data distribution in this sparse set of points. To not let these set of points to come in our way of identifying the two main clusters, I also ran the Linkage Clustering Algorithm on all (points and cluster) distance configurations with number of clusters = 3 and plotted the notable results in figure 5.

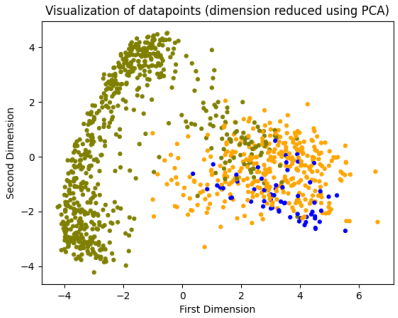
We observe that the distance function does play some part with number of clusters = 3. But again, the manhattan distance fell behind euclidean in every configuration as hypothesized and hence, I have ignored those plots. Also as we expected, single linkage performs poorly in clustering and produces highly skewed clusters. Complete linkage performs well and creates two clusters close to what we expect with the 2D visualization. It also identifies a smaller third cluster within the one on the right that might be outliers. Visually, euclidean distance seems to perform very marginally better than cosine distance for complete linkage setting.
But this time, average linkage with cosine distance seems to perform the best - it almost identified the two clusters one might expect from the 2D plot perfectly, and identifies a single point third cluster. With euclidean distance, it complete fell apart though. This motivated me to examine if it was a one-off case and I ran the clustering by removing one point from the data points set (the last point to be precise). I got the plots demonstrated in figure 6.

The average linkage with cosine distance completely fell apart by removing just one data point! Complete linkage with euclidean distance also was considerably affected by that point's removal. In comparison, complete linkage with cosine distance was fairly robust to the point's removal. The fact that the other configurations were affected by this point hints at the point being an outlier or from the tail, which dominated the distances and consequently the clustering. And the observation that complete linkage with cosine distance was robust to this point implies it identifies the third outlier cluster to a good extent as we wanted and also decently identifies the two visual clusters (especially when we logically merge the second and third clusters both of which are on the right). So, I finally use this configuration (**Complete Linkage, Cosine Distance with number of clusters = 3**) to compare to the true labels.

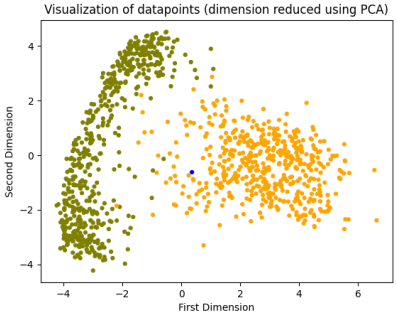
b The true labels' scatter plot and the visualization of the clustering using complete linkage and cosine distance configuration (with number of clusters set to 3) is shown in figure 7. The RHS plot in the second row shows the same plot as LHS, but with the second and third clusters manually merged to consider as a single cluster, which very closely resembles the true labels as is evident. For getting the above plots (as well as for the exploratory data analysis in part a), I perform PCA and reduce the data



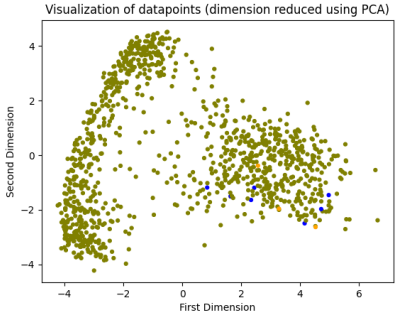
(a) Complete Linkage, Cosine Distance



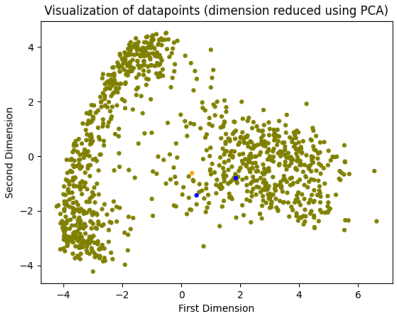
(b) Complete Linkage, Euclidean Distance



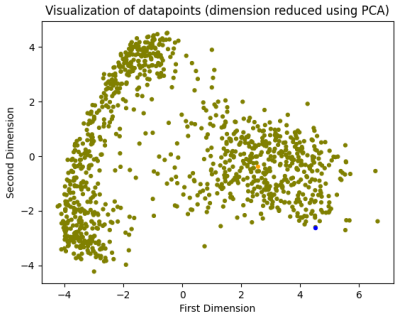
(c) Average Linkage, Cosine Distance



(d) Average Linkage, Euclidean Distance

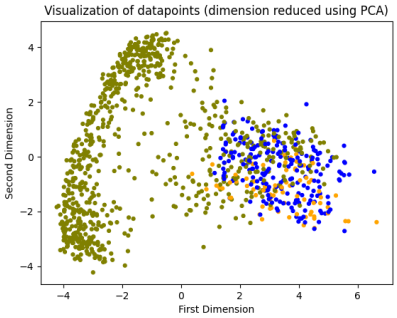


(e) Single Linkage, Cosine Distance

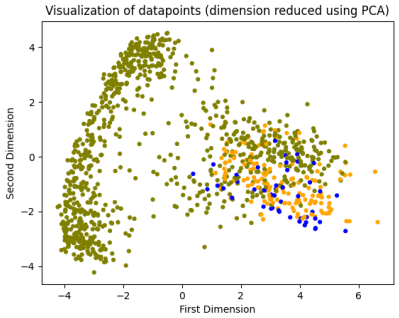


(f) Single Linkage, Euclidean Distance

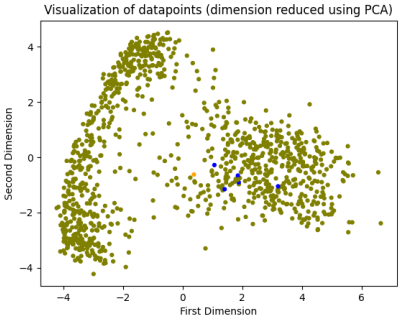
Figure 5: Visualization of clustering on data points with no. of clusters = 3 to 2 dimensions.



(a) Complete Linkage, Cosine Distance



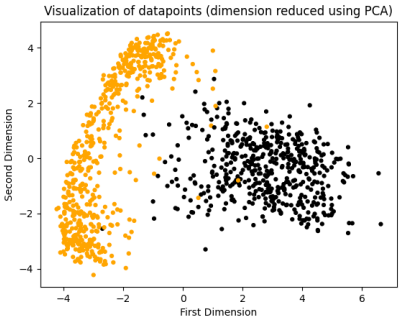
(b) Complete Linkage, Euclidean Distance



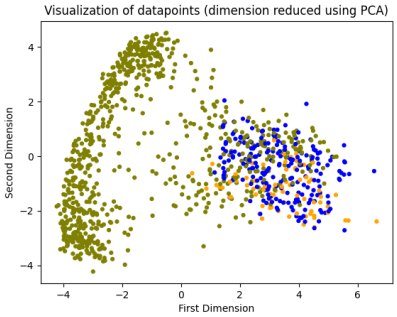
(c) Average Linkage, Cosine Distance

Figure 6: Visualization of clustering on all except the last data point with no. of clusters = 3

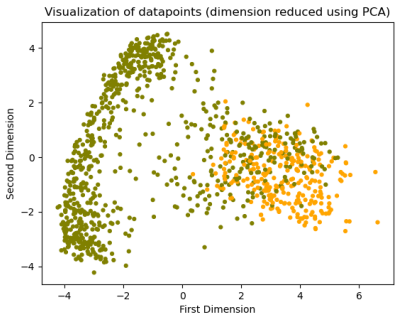
Overall after looking at the true labels, the intuitions about the distances and linkage metrics seem to hold fairly well. Single linkage and manhattan distance with any configuration performs poorly on this data. For high dimensional data where magnitudes of data vectors are fairly constant, cosine similarity seems to be slightly more consistent than euclidean (though robustness depends on the linkage it is tied with). Average clustering with cosine distance is the closest to the true labels among all configurations I tried, but their sensitivity to a data point's removal is of significant concern. Complete linkage also seems to be slightly more consistent than average linkage for this data across other configuration changes, and perform a clustering which is closest to the intuition with number of clusters = 2, and good, robust clustering with euclidean and cosine distances for number of clusters = 3.



(a) True Targets



(b) Complete Linkage, Cosine Distance



(c) Complete Linkage, Cosine Distance with merged second and third clusters

Figure 7: Visualization of true labels and clustering with no. of clusters = 3, cosine distance, and complete linkage

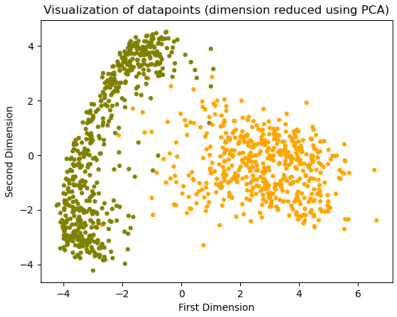
As a side note, I tried **ward** linkage (which is most commonly used in linkage clustering) with euclidean, cosine, and manhattan distances, with number of clusters = 2. Ward's method for cluster distance is given by how much the sum of squares of distances from the respective cluster centers increases when we merge two clusters. Mathematically, it is defined as:

$$D(\mathcal{C}_h, \mathcal{C}_k) = \frac{|\mathcal{C}_h| * |\mathcal{C}_k|}{|\mathcal{C}_h| + |\mathcal{C}_k|} d(\mathbf{m}_h, \mathbf{m}_k)$$

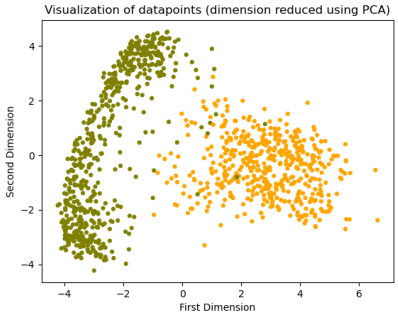
where $\mathbf{m}_h$ and $\mathbf{m}_k$ are the centers of clusters $\mathcal{C}_k$ and $\mathcal{C}_k$ respectively.

From figure 8, we observe that ward linkage with cosine, euclidean, and manhattan distances clusters the data points nearly perfectly even with

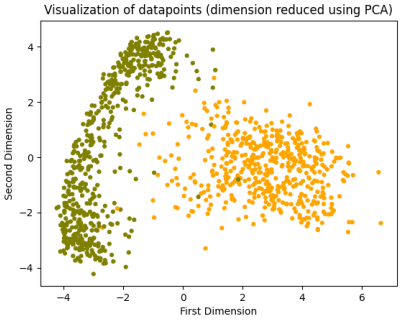
number of clusters = 2 as one would intuitively expect from the 2D plot, and it also matches the true labels to an excellent extent. This linkage proves to be a better method for this scenario and maybe so in any general setting.



(a) Ward Linkage, Cosine Distance



(b) Ward Linkage, Euclidean Distance



(c) Ward Linkage, Manhattan Distance

Figure 8: Visualization of clustering with ward linkage for no. of clusters = 2

3 Code

id3.py

```
import pandas as pd
import numpy as np
from math import log
from random import sample
import os
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.model_selection import KFold
from itertools import takewhile

class Node:
    def __init__(self, depth, node_type='normal', label=None, feature=None, class_dict=None):
        assert (class_dict is None and feature is None) or (class_dict is not None and feature is not None)
        self.depth = depth
        self.node_type = node_type
        self.feature_to_split_on = feature
        self.label = label
        self.children = {value: None for value in class_dict[feature]} if feature is not None else {}

    def add_split_feature(self, feature, class_dict):
        self.feature_to_split_on = feature
        self.children = {value: None for value in class_dict[feature]} if feature is not None else {}


class DecisionTree:
    def __init__(self, X, Y, max_depth, class_dict):
        self.max_depth = max_depth
        self.class_dict = class_dict
        self.root = self.build(X, Y, 0)
        self.features = list(X.columns)

    def build(self, X, Y, depth):
        node = Node(depth, 'root' if depth == 0 else 'normal')
        counter = Counter(Y)
        if counter.most_common(1)[0][1] == len(Y) or depth == self.max_depth:
            node.node_type = 'leaf'
```

```
            all_most_common = list(takewhile(lambda val: val[1] ==
            ↪  counter.most_common(1)[0][1], counter.most_common()))
            all_most_common.sort()
            node.label = all_most_common[0][0]
            return node

        split_feature = self.get_best_feature_to_split_on(X, Y)
        node.add_split_feature(split_feature, self.class_dict)

        for value in self.class_dict[split_feature]:
            eff_idx = X[split_feature] == value
            X_eff, Y_eff = X[eff_idx].loc[:, X.columns != split_feature],
            ↪  Y[eff_idx]
            if len(Y_eff) == 0:
                node.children[value] = Node(
                    depth+1, 'leaf', counter.most_common(1)[0][0]
                )
            else:
                node.children[value] = self.build(X_eff, Y_eff, depth+1)

        return node

    @staticmethod
    def gini_index(y_counts):
        sum_counts = np.sum(y_counts)
        return 1 - np.sum(list(map(lambda x: (x/sum_counts)**2, y_counts)))

    def get_best_feature_to_split_on(self, X, Y):
        features = list(X.columns)
        best_feature = None
        best_gini_gain = -np.inf

        gini_gain_base = self.gini_index([info[1] for info in
        ↪  Counter(Y).most_common()])
        for feature in features:
            gini_gain = gini_gain_base
            for value in self.class_dict[feature]:
                eff_idx = X[feature] == value
                Y_eff = Y[eff_idx]
                y_counts = [info[1] for info in Counter(Y_eff).most_common()]
                gini_gain -= len(Y_eff) * self.gini_index(y_counts) / len(Y)

            if gini_gain > best_gini_gain:
                best_gini_gain = gini_gain
```

```
                best_feature = feature

        return best_feature

    def predict(self, X):
        Y_hat = []
        for _, x in X.iterrows():
            x_features = {}
            for feature in self.features:
                x_features.update({feature: x[feature]})

            Y_hat.append(self.predict_obs(x_features, self.root))

        return Y_hat

    def predict_obs(self, x, node):
        if node.node_type == 'leaf':
            return node.label

        feature_value = x[node.feature_to_split_on]
        return self.predict_obs(x, node.children[feature_value])

    def accuracy(self, Y, Y_hat):
        return np.mean(np.array(Y, dtype=np.bool8) == np.array(Y_hat,
        ↪  dtype=np.bool8))

KFOLD = True
TEST = True

if __name__ == "__main__":
    train = pd.read_csv('Q1_data/train.csv')
    train['target'] = train['target'] == 'recurrence-events'
    test = pd.read_csv('Q1_data/test.csv')
    test['target'] = test['target'] == 'recurrence-events'
    attributes = ['age', 'menopause', ' tumor_size', 'inv_nodes',
    ↪  'node_caps', 'deg_malig', 'breast', 'breast_quad',
                  'irradiant']
    train['deg_malig'] = train['deg_malig'].astype(str)
    test['deg_malig'] = test['deg_malig'].astype(str)

    train = train.sample(frac=1).reset_index(drop=True)

    class_dict = {'age': ["10-19", "20-29", "30-39", "40-49", "50-59",
    ↪  "60-69", "70-79", "80-89", "90-99"],
```

```
                    'menopause': ["lt40", "ge40", "premeno"],
                    ' tumor_size': ["0-4", "5-9", "10-14", "15-19", "20-24",
                    ↪  "25-29", "30-34", "35-39", "40-44", "45-49",
                                    "50-54", "55-59"],
                    'inv_nodes': ["0-2", "3-5", "6-8", "9-11", "12-14",
                    ↪  "15-17", "18-20", "21-23", "24-26", "27-29",
                                    "30-32", "33-35", "36-39"],
                    'node_caps': ["yes", "no"],
                    'deg_malig': ["1", "2", "3"],
                    'breast': ["left", "right"],
                    'breast_quad': ["left_up", "left_low", "right_up",
                    ↪  "right_low", "central"],
                    'irradiant': ["yes", "no"]}

    if KFOLD:
        best_max_depth = -1
        best_val_acc = -1
        max_depths = [1, 3, 5, 7, 9]
        train_accuracies, val_accuracies = [], []
        for max_depth in max_depths:
            kfold_train_acc, kfold_val_acc = [], []
            kf = KFold(n_splits = 3, shuffle=True, random_state=2022)
            for train_index, val_index in kf.split(train):
                X_train, Y_train = train.loc[train_index, train.columns !=
                ↪  'target'], train.loc[train_index, 'target']
                X_val, Y_val = train.loc[val_index, train.columns !=
                ↪  'target'], train.loc[val_index, 'target']
                dt = DecisionTree(X_train, Y_train, max_depth, class_dict)
                Y_train_hat = dt.predict(X_train)
                kfold_train_acc.append(dt.accuracy(Y_train, Y_train_hat))
                Y_val_hat = dt.predict(X_val)
                kfold_val_acc.append(dt.accuracy(Y_val, Y_val_hat))

            train_accuracies.append(np.mean(kfold_train_acc))
            val_accuracies.append(np.mean(kfold_val_acc))
            print(f'Train Accuracy for max-depth {max_depth}:',
            ↪  train_accuracies[-1])
            print(f'Val Accuracy for max-depth {max_depth}:',
            ↪  val_accuracies[-1])
            if val_accuracies[-1] > best_val_acc:
                best_val_acc = val_accuracies[-1]
                best_max_depth = max_depth

        plt.figure(0)
```

```
156         plt.plot(max_depths, train_accuracies, linestyle='-.', color='b',
            ↪  marker='o', label='Train Accuracy')
157         plt.plot(max_depths, val_accuracies, linestyle='-', color='orange',
            ↪  marker='x', label='Validation Accuracy')
158         plt.title('Accuracy vs Max Depth of Decision Tree')
159         plt.ylabel('Accuracy')
160         plt.xlabel('Max Depth')
161         plt.legend(loc='best')
162         plt.savefig(f'./plots/dt_acc_2.png')
163         plt.show()
164
165     if TEST:
166         MAX_DEPTH = 3 if not KFOLD else best_max_depth
167         X_train, Y_train = train.loc[:, train.columns != 'target'],
            ↪  train['target']
168         X_test, Y_test = test.loc[:, test.columns != 'target'], test['target']
169
170         dt = DecisionTree(X_train, Y_train, MAX_DEPTH, class_dict)
171         Y_train_hat = dt.predict(X_train)
172         Y_test_hat = dt.predict(X_test)
173
174         print(f'Final Train Accuracy (after training on all data) with
            ↪  max-depth {MAX_DEPTH}:', dt.accuracy(Y_train, Y_train_hat))
175         print(f'Final Test Accuracy (after training on all data) with
            ↪  max-depth {MAX_DEPTH}:', dt.accuracy(Y_test, Y_test_hat))
```

clustering.py

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import math
5  from sklearn.decomposition import PCA
6  from sklearn.manifold import TSNE
7  from itertools import cycle, islice
8
9  def pts_distance(x, y, mode='euclidean'):
10     if mode.lower() == 'euclidean':
11         return np.sqrt(np.sum((x-y)**2))
12     elif mode.lower() == 'cosine':
13         x_mod = np.linalg.norm(x)
14         y_mod = np.linalg.norm(y)
15         return 1 - np.dot(x/x_mod, y/y_mod)
```

```
    else: # 'manhattan' or 'cityblock'
        return np.sum(np.abs(x-y))

def cluster_distance(X, Y, linkage='single', dist=None, dist_betw_center=None):
    # If dist is not None, we use indices from X and Y to get the distance from dist matrix
    # dist_betw_center is needed for using Ward linkage
    if linkage.lower() in ['single', 'min']:
        return np.min([pts_distance(x, y) if dist is None else dist[x][y] for x in X for y in Y])
    elif linkage.lower() in ['max', 'complete']:
        return np.max([pts_distance(x, y) if dist is None else dist[x][y] for x in X for y in Y])
    elif linkage.lower() == 'ward':
        return (len(X)*len(Y)/(len(X)+len(Y))) * dist_betw_center
    else: # 'average'
        return np.mean([pts_distance(x, y) if dist is None else dist[x][y] for x in X for y in Y])


class LinkageClustering:
    def __init__(self, n_clusters, cluster_dist_linkage='single', pts_dist_mode='euclidean'):
        # assert cluster_dist_linkage != 'ward' or pts_dist_mode == 'euclidean'
        self.n_clusters = n_clusters
        self.cluster_dist_linkage = cluster_dist_linkage
        self.pts_dist_mode = pts_dist_mode

    def fit(self, X):
        if isinstance(X, pd.DataFrame):
            self.X = X.to_numpy()
        else:
            self.X = X

        self.fill_pts_distances()

        self.clusters = {
            id: [id] for id in range(len(X))
        }

        self.distances_ = []
        self.children_ = []
```

```
53
54          self.cluster_distances = {}
55          for id_X, clusterX in self.clusters.items():
56              self.cluster_distances[id_X] = {}
57              for id_Y, clusterY in self.clusters.items():
58                  if id_X > id_Y:
59                      self.cluster_distances[id_X][id_Y] =
                        ↪  self.cluster_distances[id_Y][id_X]
60                      continue
61                  elif id_X == id_Y:
62                      continue
63
64                  dist_betw_center = None
65                  if self.cluster_dist_linkage == 'ward':
66                      m_X = np.mean([self.X[i] for i in self.clusters[id_X]],
                        ↪  axis=0)
67                      m_Y = np.mean([self.X[i] for i in self.clusters[id_Y]],
                        ↪  axis=0)
68                      dist_betw_center = pts_distance(m_X, m_Y,
                        ↪  self.pts_dist_mode)
69                  self.cluster_distances[id_X][id_Y] = cluster_distance(
70                      clusterX, clusterY, self.cluster_dist_linkage,
                        ↪  self.pts_distances, dist_betw_center
71                  )
72
73          while len(self.clusters) > self.n_clusters:
74              id_X, id_Y = self.find_closest_clusters()
75              self.merge_clusters(id_X, id_Y)
76
77          self.re_index_clusters()
78          self.fill_labels_from_clusters()
79
80      def fill_pts_distances(self):
81          self.pts_distances = [
82              [pts_distance(self.X[i], self.X[j], self.pts_dist_mode) for j in
                ↪  range(len(self.X))]
83              for i in range(len(self.X))
84          ]
85
86      def find_closest_clusters(self):
87          best_cluster_pair_idx = (-1, -1)
88          best_distance = np.inf
89          for i in self.clusters:
90              for j in self.clusters:
```

```
                if i >= j:
                    continue
                if self.cluster_distances[i][j] < best_distance:
                    best_distance = self.cluster_distances[i][j]
                    best_cluster_pair_idx = (i, j)

        self.distances_.append(best_distance)
        self.children_.append(list(best_cluster_pair_idx))

        return best_cluster_pair_idx

    def merge_clusters(self, id_X, id_Y):
        for ind in self.clusters:
            if ind == id_X or ind == id_Y:
                continue

            if self.cluster_dist_linkage in ['single', 'min']:
                self.cluster_distances[id_X][ind] =
                ↪  self.cluster_distances[ind][id_X] = min(
                    self.cluster_distances[id_X][ind],
                    ↪  self.cluster_distances[id_Y][ind]
                )
            elif self.cluster_dist_linkage in ['max', 'complete']:
                self.cluster_distances[id_X][ind] =
                ↪  self.cluster_distances[ind][id_X] = max(
                    self.cluster_distances[id_X][ind],
                    ↪  self.cluster_distances[id_Y][ind]
                )
            elif self.cluster_dist_linkage == 'ward':
                m_X = [self.X[i] for i in self.clusters[id_X]]
                m_X.extend([self.X[i] for i in self.clusters[id_Y]])
                m_X = np.mean(m_X, axis=0)
                m_Y = np.mean([self.X[i] for i in self.clusters[ind]], axis=0)
                dist_betw_center = pts_distance(m_X, m_Y, self.pts_dist_mode)
                n1 = len(self.clusters[id_X]) + len(self.clusters[id_Y])
                n2 = len(self.clusters[ind])
                self.cluster_distances[id_X][ind] =
                ↪  self.cluster_distances[ind][id_X] = (n1*n2/(n1+n2)) *\
                    dist_betw_center
            else: # Average Linkage
                nr1 = len(self.clusters[id_X])*len(self.clusters[ind])
                nr2 = len(self.clusters[id_Y])*len(self.clusters[ind])
                dr = nr1+nr2
```

```
                self.cluster_distances[id_X][ind] =
                ↪  self.cluster_distances[ind][id_X] = \
                    (nr1/dr)*self.cluster_distances[id_X][ind] +
                    ↪  (nr2/dr)*self.cluster_distances[id_Y][ind]

            self.cluster_distances[ind].pop(id_Y, None)

        self.cluster_distances[id_X].pop(id_Y, None)
        self.cluster_distances.pop(id_Y, None)
        self.clusters[id_X].extend(self.clusters[id_Y])
        self.clusters.pop(id_Y, None)


    def re_index_clusters(self):
        mapper = {
            old_key: new_key for new_key, old_key in enumerate(self.clusters)
        }
        self.clusters = {
            mapper[old_key]: self.clusters[old_key] for old_key in
            ↪  self.clusters
        }
        self.cluster_distances = {
            mapper[id_X]: {
                mapper[id_Y]: dist for id_Y, dist in distances.items()
            } for id_X, distances in self.cluster_distances.items()
        }

    def fill_labels_from_clusters(self):
        self.cluster_labels = [-1 for _ in range(len(self.X))]
        for cluster_id in self.clusters:
            for X_idx in self.clusters[cluster_id]:
                self.cluster_labels[X_idx] = cluster_id


def scatter_plot(X, y=None, fig_idx=0, merge_outliers=False, tsne=False):
    if y is None:
        y = [0 for _ in range(len(X))]
    if merge_outliers:
        y = [0 if yi == 0 else 1 for yi in y]

    pca = PCA(n_components=2) if not tsne else TSNE(n_components=2)
    X_red = pca.fit_transform(X)

    plt.figure(fig_idx)
```

```
    colors = np.array(
        list(
            islice(
                cycle(
                    [
                        "olive",
                        "orange",
                        "blue",
                        "red",
                        "brown",
                        "mediumseagreen",
                        "#377eb8",
                        "pink",
                        "m",
                        "black"
                    ]
                ),
                int(max(y) + 1),
            )
        )
    )
    plt.scatter(X_red[:, 0], X_red[:, 1], s=12, color=colors[y])
    plt.xlabel('First Dimension')
    plt.ylabel('Second Dimension')
    plt.title('Visualization of datapoints (dimension reduced using PCA)')
    plt.show()
    plt.close()

def pca_plot(variances, threshold=0.95, fig_idx=0):
    plt.figure(fig_idx)
    npc = np.argmax(variances >= threshold)+1
    ax = plt.subplot(1, 1, 1)
    ax.set_xlabel('number of components (log scale)')
    ax.set_ylabel('cumulative % explained variance')
    ax.set_title('PCA')
    ax.semilogx(list(range(1,variances.shape[0]+1)), variances*100)
    ax.axhline(variances[npc-1]*100, c='red', linestyle='dashed',
    ↪  label=f'cum-var {variances[npc-1]*100:.2f}% @ {npc} PC')
    plt.legend()
    plt.show()
    plt.close()


EPS = 1e-8
```

```python
if __name__ == "__main__":
    train = pd.read_csv('Q2_data/train.csv', header=None)
    print(f'Total number of datapoints: {len(train)}')
    # Removing columns that don't have any variations (since they dont
    ↪ contribute to distinguishing datapoints)
    columns = list(train.columns)
    columns_to_drop = [column for column in columns if
    ↪ train[column].max()-train[column].min() < EPS]
    train.drop(columns_to_drop, axis=1, inplace=True)
    print(f'{len(columns_to_drop)} out of {len(columns)} columns dropped due
    ↪ to their zero contribution to distinguishing datapoints')
    min_val = math.inf
    max_val = -math.inf
    for column in train.columns:
        min_val = min(min_val, train[column].min())
        max_val = max(max_val, train[column].max())
    print(f'Min of column values: {min_val} | Max of column values:
    ↪ {max_val}')

    # Stats about vector magnitudes
    X = train.to_numpy()
    mag = np.linalg.norm(X, axis=1)
    print(f'Min of point vector magnitudes: {np.min(mag):.3f}')
    print(f'Max of point vector magnitudes: {np.max(mag):.3f}')
    print(f'Mean of point vector magnitudes: {np.mean(mag):.3f}')
    print(f'Variance of point vector magnitudes: {np.var(mag):.3f}')

    # To plot PCA
    dim_redn = PCA()
    X = dim_redn.fit_transform(train)
    expl_var = np.cumsum(dim_redn.explained_variance_ratio_)
    pca_plot(expl_var, fig_idx=0)

    # To plot true labels
    true_labels = pd.read_csv('Q2_data/labels.csv', header=None)
    true_labels[0] = true_labels[0].astype(int)
    scatter_plot(train, true_labels[0], fig_idx=1)

    # Scatter Plot of Clustering Results
    lc = LinkageClustering(2, 'complete', 'cosine')
    lc.fit(train)
    scatter_plot(train, lc.cluster_labels, fig_idx=2)

```

4 Collaboration Policy

I discussed with Mayank Baranwal (Andrew Id: **mbaranwa**) regarding the observation about sensitivity of the average clustering to removing a single point.