CSC 587 HW 4

Daniel R. Getty

2024-04-03

Homework 4

1. Given a data tuple having the values ”systems”, ”26_30”, and ”46K_50K” for the attributes department, age, and salary, respectively, what would a naive Bayesian classification of the status according to the data above? Notice that Count column is NOT an attribute. It just tells how many times a row occurs in our database and status is our target variable.

```
# create a data frame
data = {'department': ['sales', 'sales', 'sales', 'systems', 'systems', 'systems', 'systems', 'marketing

df = pd.DataFrame(data)
print(df.to_markdown(tablefmt="grid"))
```

	department	age	salary	status	count
0	sales	31_35	46K_50K	senior	30
1	sales	26_30	26K_30K	junior	40
2	sales	31_35	31K_35K	junior	40
3	systems	21_25	46K_50K	junior	20
4	systems	31_35	66K_70K	senior	5
5	systems	26_30	46K_50K	junior	3
6	systems	41_45	66K_70K	senior	3
7	marketing	36_40	46K_50K	senior	10
8	marketing	31_35	41K_45K	junior	4
9	secretary	46_50	36K_40K	senior	4
10	secretary	26_30	26K_30K	junior	6

```
# df.head()
# print (df)

# calculate the probability of each status
sum_total = sum(df['count'])
print("Total:",sum_total)
```

Total: 165

```
senior_total = sum(df['count'][df['status'] == 'senior'])
print("Senior Total:",senior_total)
```

Senior Total: 52

```
junior_total = sum(df['count'][df['status'] == 'junior'])
print("Junior Total:",junior_total)
```

Junior Total: 113

```
PStaus_Senior = (senior_total / sum_total)

print("P(Staus = Senior)",round(PStaus_Senior,2))
```

P(Staus = Senior) 0.32

```
PStaus_Junior = (junior_total / sum_total)

print("P(Staus = Junior)",round(PStaus_Junior,2))
```

P(Staus = Junior) 0.68

```
# calculate the probability of systems department
systems_total = sum(df['count'][df['department'] == 'systems'])
print("Department Total:",systems_total)
```

Department Total: 31

```
PDept_Systems = (systems_total / sum_total)
print("P(Department = Systems)",round(PDept_Systems,2))
```

P(Department = Systems) 0.19

```
# calculate the probability of age 26_30
age_total = sum(df['count'][df['age'] == '26_30'])
print("Age Total:",age_total)
```

Age Total: 49

```
PAge_26_30 = (age_total / sum_total)
print("P(Age = 26_30)",round(PAge_26_30,2))
```

P(Age = 26_30) 0.3

```
# calculate the probability of salary 46K_50K
salary_total = sum(df['count'][df['salary'] == '46K_50K'])
print("Salary Total:",salary_total)
```

Salary Total: 63

```
PSalary_46K_50K = (salary_total / sum_total)
print("P(Salary = 46K_50K)",round(PSalary_46K_50K,2))
```

P(Salary = 46K_50K) 0.38

```
# calculate the probability of status senior given department systems, age 26_30, and salary 46K_50K
PStatus_Senior_x = (PDept_Systems * PAge_26_30 * PSalary_46K_50K * PStaus_Senior)
print("P(Status = Senior | Department = Systems, Age = 26_30, Salary = 46K_50K)",round(PStatus_Senior_x
```

P(Status = Senior | Department = Systems, Age = 26_30, Salary = 46K_50K) 0.01

```
# calculate the probability of status junior given department systems, age 26_30, and salary 46K_50K
PStatus_Junior_x = (PDept_Systems * PAge_26_30 * PSalary_46K_50K * PStaus_Junior)
print("P(Status = Junior | Department = Systems, Age = 26_30, Salary = 46K_50K)",round(PStatus_Junior_x
```

P(Status = Junior | Department = Systems, Age = 26_30, Salary = 46K_50K) 0.01

2. split your diabetes data into two parts for training and testing purposes. Namely, reserve last 10 rows of the diabetes_train.csv for the test set. Then fit a SVM classifier on the bigger portion of this data and test it on these 10 rows you had reserved. Please feel free to modify existing codes. Notice that you're not going to read diabetes_test.csv anymore since you're going to split the bigger data. Please submit your Python code and your prediction results

```
# read the diabetes data
diabetes = pd.read_csv('diabetes_train.csv')

# split the data into training and testing reserves last 10 rows for testing
test = diabetes.iloc[-10:]
train = diabetes.iloc[:-10]

print("Test Data")
```

Test Data

```
print(test.to_markdown(tablefmt="grid"))
```

	preg	plas	pres	skin	insu	mass	pedi	age	class
748	3	187	70	22	200	36.4	0.408	36	tested_positive
749	6	162	62	0	0	24.3	0.178	50	tested_positive
750	4	136	70	0	0	31.2	1.182	22	tested_positive
751	1	121	78	39	74	39	0.261	28	tested_negative
752	3	108	62	24	0	26	0.223	25	tested_negative
753	0	181	88	44	510	43.3	0.222	26	tested_positive
754	8	154	78	32	0	32.4	0.443	45	tested_positive
755	1	128	88	39	110	36.5	1.057	37	tested_positive
756	7	137	90	41	0	32	0.391	39	tested_negative
757	0	123	72	0	0	36.3	0.258	52	tested_positive

```
print("\n")
```

```
print("Train Data")
```

Train Data

```
dfhead = train.head()
print(dfhead.to_markdown(tablefmt="grid"))
```

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive

```
print("\n")
```

```
# print(train.to_markdown(tablefmt="grid"))

# fit a SVM classifier
from sklearn import svm
clf = svm.SVC()
clf.fit(train.iloc[:,:-1], train.iloc[:,-1])
```

SVC()

```
# test the classifier
prediction = clf.predict(test.iloc[:,:-1])
prediction = pd.DataFrame(prediction, columns=['Prediction'])
# print the prediction results
# print("Predicted Results =",prediction)
print(prediction.to_markdown())
```

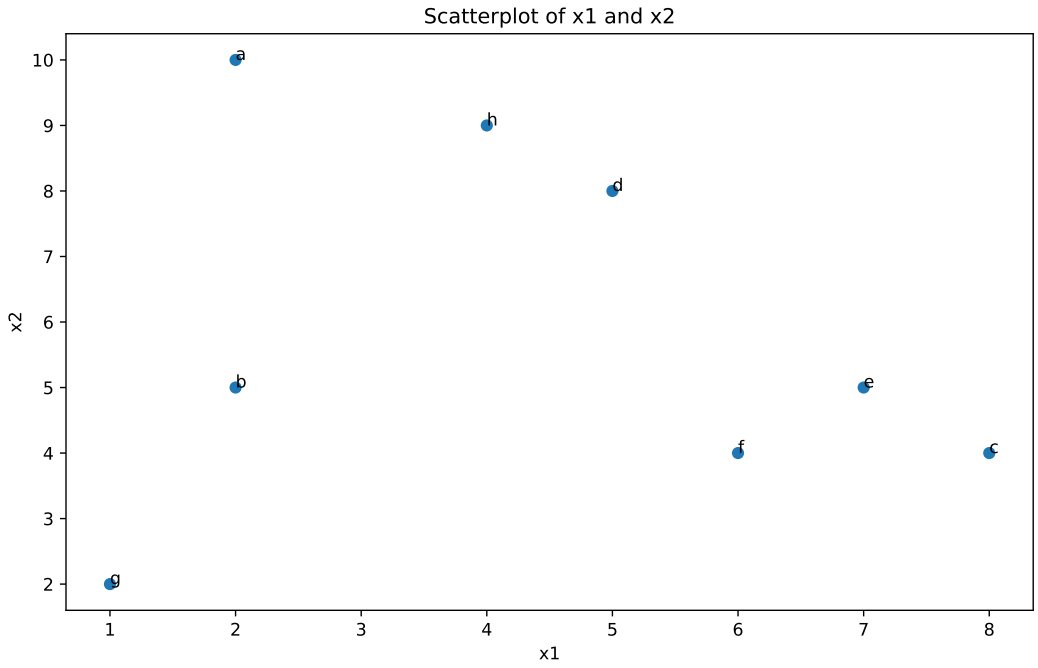
	Prediction
0	tested_positive
1	tested_positive
2	tested_negative
3	tested_negative
4	tested_negative
5	tested_positive
6	tested_positive
7	tested_negative
8	tested_negative
9	tested_negative

3.

```
# create a data frame
classes = ['a','b','c','d','e','f','g','h']
data = {'x1': [2, 2, 8, 5, 7, 6, 1, 4],'x2': [10, 5, 4, 8, 5, 4, 2, 9]}
P3_df = pd.DataFrame(data, index=classes)
# print(P3_df)
print(P3_df.to_markdown(tablefmt="grid"))
```

	x1	x2
a	2	10
b	2	5
c	8	4
d	5	8
e	7	5
f	6	4
g	1	2
h	4	9

```
# build a scatterplot of the df
plt.figure(figsize=(10, 6))
plt.scatter(P3_df['x1'], P3_df['x2'])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Scatterplot of x1 and x2')
for i, txt in enumerate(classes):
    plt.annotate(txt, (P3_df['x1'][i], P3_df['x2'][i]))

plt.show()
```



(a) If h and c are selected as the initial centers for your k-means clustering, assign memberships for other points, and compute the means (centroids) of your initial clusters. You can use Manhattan distance.

```
# initial centers
c1 = P3_df.loc['h']
c2 = P3_df.loc['c']
print(c1)
```

x1 4 x2 9 Name: h, dtype: int64

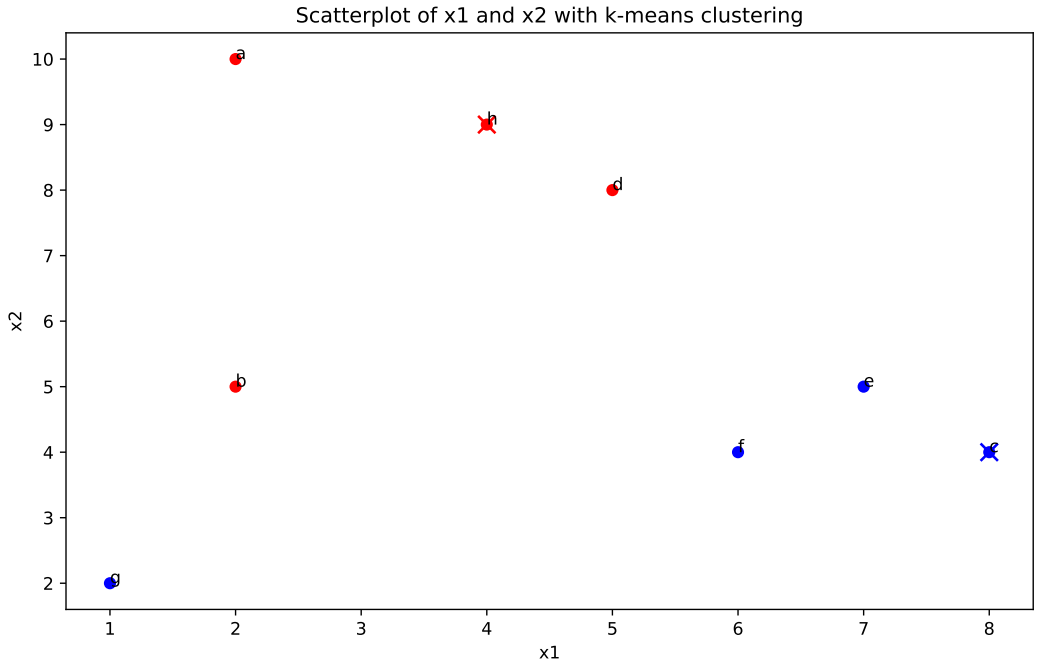
```
print(c2)
```

x1 8 x2 4 Name: c, dtype: int64

```
# assign memberships
P3_df['d_c1'] = np.sqrt((P3_df['x1'] - c1['x1'])**2 + (P3_df['x2'] - c1['x2'])**2)
P3_df['d_c2'] = np.sqrt((P3_df['x1'] - c2['x1'])**2 + (P3_df['x2'] - c2['x2'])**2)
P3_df['mem'] = np.where(P3_df['d_c1'] < P3_df['d_c2'], 'c1', 'c2')

# print(P3_df)
print(P3_df.to_markdown(tablefmt="grid"))
```

	x1	x2	d_c1	d_c2	mem
a	2	10	2.23607	8.48528	c1
b	2	5	4.47214	6.08276	c1
c	8	4	6.40312	0	c2
d	5	8	1.41421	5	c1
e	7	5	5	1.41421	c2
f	6	4	5.38516	2	c2
g	1	2	7.61577	7.28011	c2
h	4	9	0	6.40312	c1

```
# build a visualization of the k-means clustering
plt.figure(figsize=(10, 6))
plt.scatter(P3_df['x1'], P3_df['x2'], c=P3_df['mem'].map({'c1': 'red', 'c2': 'blue'}))
plt.scatter(c1['x1'], c1['x2'], color='red', marker='x', s=100)
plt.scatter(c2['x1'], c2['x2'], color='blue', marker='x', s=100)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Scatterplot of x1 and x2 with k-means clustering')
for i, txt in enumerate(classes):
    plt.annotate(txt, (P3_df['x1'][i], P3_df['x2'][i]))

plt.show()
```



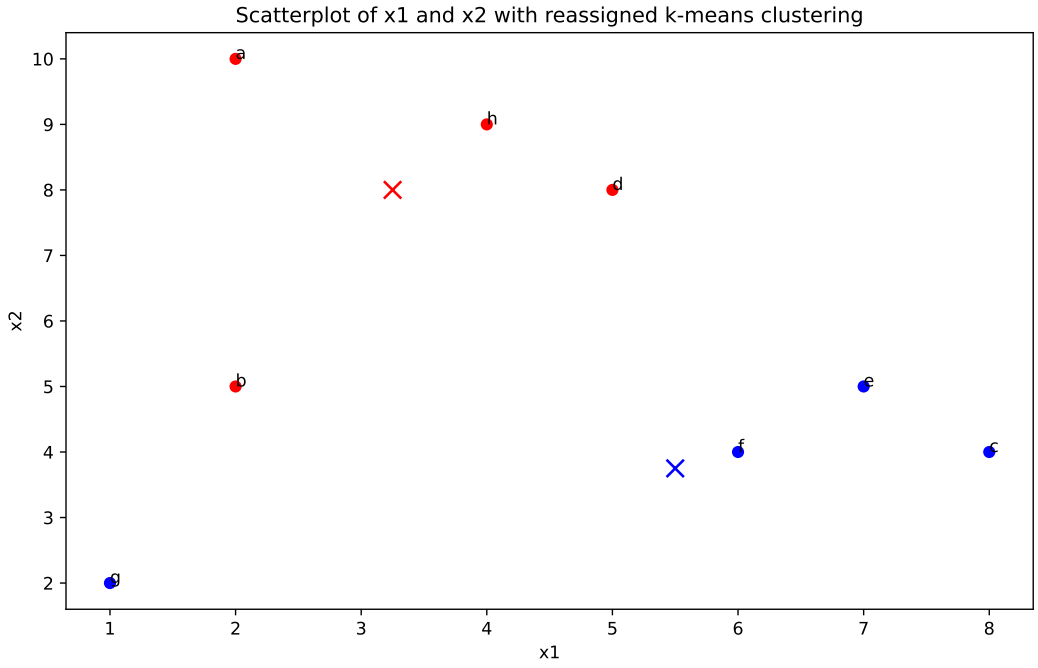
(b) Based on the centroids you found above reassign the memberships by using Manhattan distance

```
# reassign memberships
c1 = P3_df[P3_df['mem'] == 'c1'][['x1', 'x2']].mean()
c2 = P3_df[P3_df['mem'] == 'c2'][['x1', 'x2']].mean()


P3_df['d_c1'] = np.sqrt((P3_df['x1'] - c1['x1'])**2 + (P3_df['x2'] - c1['x2'])**2)
P3_df['d_c2'] = np.sqrt((P3_df['x1'] - c2['x1'])**2 + (P3_df['x2'] - c2['x2'])**2)
P3_df['mem'] = np.where(P3_df['d_c1'] < P3_df['d_c2'], 'c1', 'c2')

# print(P3_df)
print(P3_df.to_markdown(tablefmt="grid"))
```

	x1	x2	d_c1	d_c2	mem
a	2	10	2.3585	7.16327	c1
b	2	5	3.25	3.71652	c1
c	8	4	6.20987	2.51247	c2
d	5	8	1.75	4.27931	c1
e	7	5	4.80234	1.95256	c2
f	6	4	4.85412	0.559017	c2
g	1	2	6.408	4.8283	c2
h	4	9	1.25	5.46008	c1

```
# build a visualization of the reassigned k-means clustering
plt.figure(figsize=(10, 6))
plt.scatter(P3_df['x1'], P3_df['x2'], c=P3_df['mem'].map({'c1': 'red', 'c2': 'blue'}))
```

```
plt.scatter(c1['x1'], c1['x2'], color='red', marker='x', s=100)
plt.scatter(c2['x1'], c2['x2'], color='blue', marker='x', s=100)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Scatterplot of x1 and x2 with reassigned k-means clustering')
for i, txt in enumerate(classes):
    plt.annotate(txt, (P3_df['x1'][i], P3_df['x2'][i]))

plt.show()
```



4. Given the distance matrix below answer the following questions. Notice that this is a distance matrix, meaning the distance between any pair of points can be found by checking the corresponding cell.

```
# create a data frame
classes = ['b','c','d','e','f','g','h']
data = {'a': [5, 8, 4, 7, 7, 8, 2],'b': [0, 6, 4, 5, 4, 3, 4],'c': [0, 0, 5, 1, 2, 7, 6],'d': [0, 0, 0,
P4_df = pd.DataFrame(data, index=classes)
# print(P4_df)
print(P4_df.to_markdown(tablefmt="grid"))
```

	a	b	c	d	e	f	g
b	5	0	0	0	0	0	0
c	8	6	0	0	0	0	0
d	4	4	5	0	0	0	0
e	7	5	1	4	0	0	0
f	7	4	2	4	1	0	0

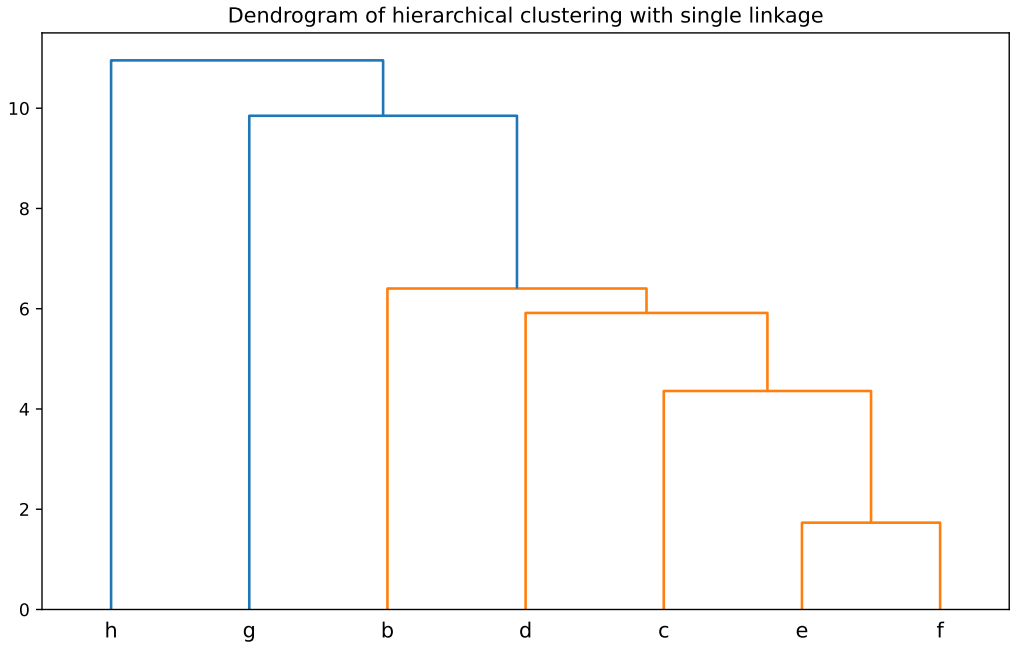
	a	b	c	d	e	f	g
g	8	3	7	7	7	5	0
h	2	4	6	1	5	5	8

(a) Apply the hierarchical clustering algorithm with single linkage to the data above. Draw the final dendrogram.

```
# apply hierarchical clustering with single linkage
from scipy.cluster.hierarchy import linkage, dendrogram
Z = linkage(P4_df, 'single')
plt.figure(figsize=(10, 6))
dendrogram(Z, p=30, truncate_mode=None, color_threshold=None, get_leaves=True, orientation='top', labels
```

{'icoord': [[55.0, 55.0, 65.0, 65.0], [45.0, 45.0, 60.0, 60.0], [35.0, 35.0, 52.5, 52.5], [25.0, 25.0, 43.75, 43.75], [15.0, 15.0, 34.375, 34.375], [5.0, 5.0, 24.6875, 24.6875]], 'dcoord': [[0.0, 1.7320508075688772, 1.7320508075688772, 0.0], [0.0, 4.358898943540674, 4.358898943540674, 1.7320508075688772], [0.0, 5.916079783099616, 5.916079783099616, 4.358898943540674], [0.0, 6.4031242374328485, 6.4031242374328485, 5.916079783099616], [0.0, 9.848857801796104, 9.848857801796104, 6.4031242374328485], [0.0, 10.954451150103322, 10.954451150103322, 9.848857801796104]], 'ivl': ['h', 'g', 'b', 'd', 'c', 'e', 'f'], 'leaves': [6, 5, 0, 2, 1, 3, 4], 'color_list': ['C1', 'C1', 'C1', 'C1', 'C0', 'C0'], 'leaves_color_list': ['C0', 'C0', 'C1', 'C1', 'C1', 'C1', 'C1']}

```
plt.title('Dendrogram of hierarchical clustering with single linkage')
plt.show()
```



(b) Determine whether a point is core based on eps = 6 and 'minPts' = 2. (Recall that a point p is a core point if at least 'minPts' points are within distance eps of it (including p).

```
# determine core points
eps = 6
minPts = 2
core_points = []
for i in range(len(P4_df)):
    if sum(P4_df.iloc[i] <= eps) >= minPts:
        core_points.append(classes[i])

print("Core Points:",core_points)
```

Core Points: [‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’]