Chayut Archamongkol
6637307621

Homework 1 Clustering and Regression

Instructions

Answer the questions and upload your answers to courseville. Answers can be in Thai or English. Answers can be either typed or handwritten and scanned. the assignment is divided into several small tasks. Each task is weighted equally (marked with **T**). For this assignment, each task is awarded 1 points. There are also optional tasks (marked with **OT**) counts for 0.5 points each.

Metrics

In a population where the amount of cats is equal to the amount of dogs. Considering the following classification results from a classifier.

Model A	Predicted dog	Predicted cat
Actual dog	30	20
Actual cat	10	40

T1. What is the accuracy of Model A? $= \frac{30+40}{30+20+10+40} = \frac{70}{100} = 0.70$ #

T2. Consider cats as 'class 1' (positive) and dogs as 'class 0' (negative), calculate the precision, recall, and F1. $\text{precision} = \frac{TP}{\#\text{predicted pos}} = \frac{40}{60} = 0.67$ # $\text{recall} = \frac{TP}{\#\text{actual pos}} = \frac{40}{50} = 0.80$ #

$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = \frac{2(0.8)(0.67)}{0.8+0.67} = 0.73$ #

T3. Consider class cat as 'class 0' and class dog as 'class 1', calculate the precision, recall, and F1. $\text{precision} = \frac{TP}{\#\text{predicted pos}} = \frac{30}{40} = 0.75$ # $\text{recall} = \frac{TP}{\#\text{actual pos}} = \frac{30}{50} = 0.60$ # $F_1 = \frac{2(0.75)(0.6)}{0.75+0.6} = 0.67$ #

It is important to specify the 'positive' class when you calculate precision, recall, and F1. If there are more than two classes, it is usually done in a one-versus-all setting where one class is considered positive and the rest of the classes are considered negative.

T4. Now consider a lopsided population where there are 80% ?? cats. What is the accuracy of Model A? Using dog as the positive class, what is the precision, recall, and F1? Explain how and why these numbers change (or does not change) from the previous questions.

$\text{Accuracy} = \frac{68}{100} = 0.68$ # $\text{Precision} = \frac{15}{20} = \frac{3}{4}$ $\text{Recall} = \frac{15}{42} = \frac{5}{14}$ $F_1 = \frac{2}{\frac{14}{5}+\frac{4}{3}} = \frac{15}{31}$ #

change. because the ratio has change

OT1. Consider the equations for accuracy and F1

$$
\begin{aligned}
Accuracy &= \frac{TP+TN}{TP+TN+FP+FN} \\
F1 &= \frac{2TP}{2TP+FP+FN}
\end{aligned}
\tag{1}
$$

When will accuracy be equal, greater, or less than F1?

$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$ $F_1 = \frac{TP+TP}{TP+TP+FP+FN}$

let $k_1 = TP+FP+FN$, $k_2 = TP$

$\text{Accuracy} = \frac{k_2+TN}{k_1+TN} = \frac{k_1+TN-k_1+k_2}{k_1+TN} = 1 - \frac{k_1-k_2}{k_1+TN}$

$F_1 = \frac{k_2+TP}{k_1+TP} = \frac{k_1+TP-k_1+k_2}{k_1+TP} = 1 - \frac{k_1-k_2}{k_1+TP}$

[illegible] $\frac{k_1-k_2}{k_1+TN}$, $\frac{k_1-k_2}{k_1+TP}$

accuracy equals to F1 : TN = TP #
accuracy less than F1 : TN > TP #
accuracy more than F1 : TN < TP #

Homework1

February 7, 2026

0.1 K-Means

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

class KMeans:
    def __init__(self, init):
        self.data = np.array([(1, 2), (3, 3), (2, 2), (8, 8), (6, 6), (7, 7),␣
↪(-3, -3), (-2, -4), (-7, -7)])
        self.init = init
        self.old_centroids = []
        self.new_centroids = init.copy()
        self.group = {}

    def cluster(self, display=True):
        i = 0
        while (self.new_centroids != self.old_centroids):
            self.old_centroids = self.new_centroids

            centroids = np.array(self.old_centroids)
            self.group = {}
            for each_data in self.data:
                # Vectorization Optimized
                point = np.array(each_data)
                dist = np.sqrt(np.sum((centroids - point) ** 2, axis=1))
                group_of_each_data = np.argmin(dist)

                if (group_of_each_data not in self.group):
                    self.group[group_of_each_data] = []
                self.group[group_of_each_data].append(each_data)

                # Traditional Way
                # min_distance = 1e9
                # group_of_each_data = -1
                # for i in range(len(self.old_centroids)):
                #     current_distance = ( (self.old_centroids[i][0] -␣
↪each_data[0]) ** 2 + (self.old_centroids[i][1] - each_data[1]) ** 2 ) ** 0.5
```

```
                #     if (current_distance < min_distance):
                #         group_of_each_data = i
                #         min_distance = current_distance

                # if (group_of_each_data not in self.group):
                #     self.group[group_of_each_data] = []

                # self.group[group_of_each_data].append(each_data)

            self.new_centroids = []
            for each in self.group.keys():
                centroid_x = sum(t[0] for t in self.group[each]) / len(self.
↪group[each])
                centroid_y = sum(t[1] for t in self.group[each]) / len(self.
↪group[each])
                self.new_centroids.append((centroid_x, centroid_y))

            i += 1

            if (display):
                print(f"--- Round {i} ---")
                print(f"Centroids: {self.new_centroids}")

        if (display):
            for cluster_id, points in self.group.items():
                xs, ys = zip(*points)
                plt.scatter(xs, ys, label=f"Cluster {cluster_id}")

            plt.xlabel("X")
            plt.ylabel("Y")
            plt.show()
```

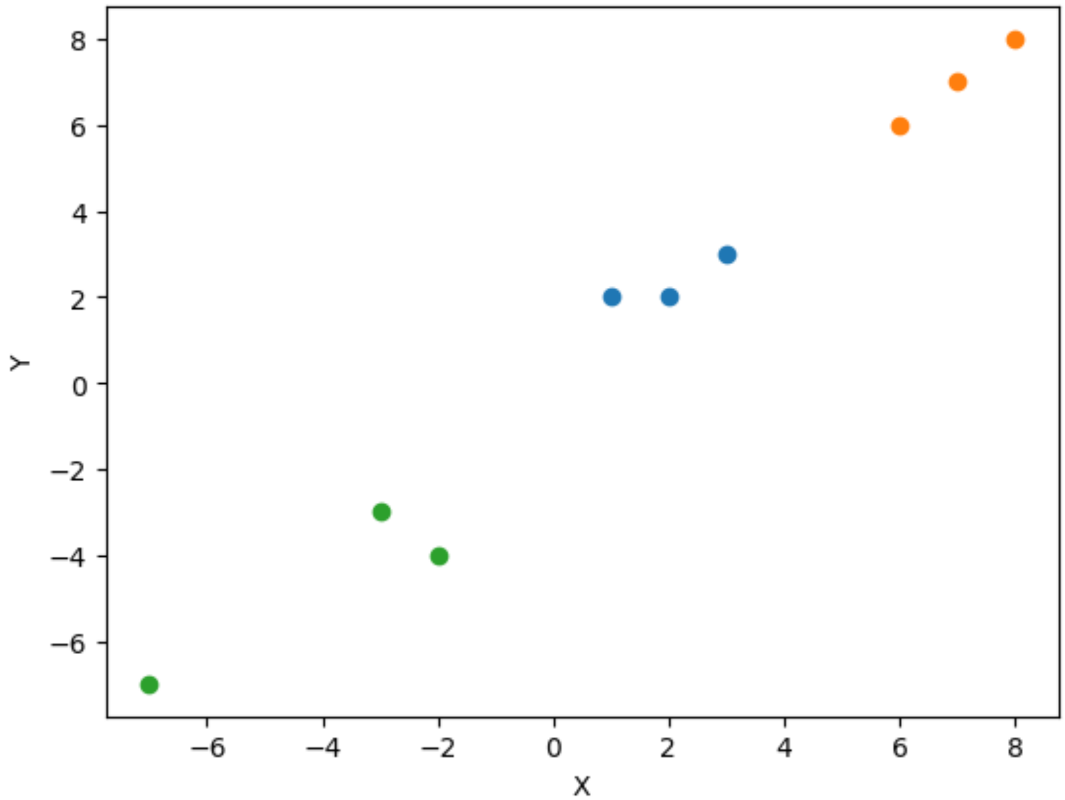
0.1.1 T5. starting points are (3,3), (2,2), and (-3,-3).

[2]:
```
init_t5 = [(3, 3), (2, 2), (-3, -3)]

km_t5 = KMeans(init_t5)
km_t5.cluster()
```

```
--- Round 1 ---
Centroids: [(np.float64(1.5), np.float64(2.0)), (np.float64(6.0),
np.float64(6.0)), (np.float64(-4.0), np.float64(-4.666666666666667))]
--- Round 2 ---
Centroids: [(np.float64(2.0), np.float64(2.3333333333333335)), (np.float64(7.0),
np.float64(7.0)), (np.float64(-4.0), np.float64(-4.666666666666667))]
--- Round 3 ---
Centroids: [(np.float64(2.0), np.float64(2.3333333333333335)), (np.float64(7.0),
```

```
np.float64(7.0)), (np.float64(-4.0), np.float64(-4.666666666666667))]
```

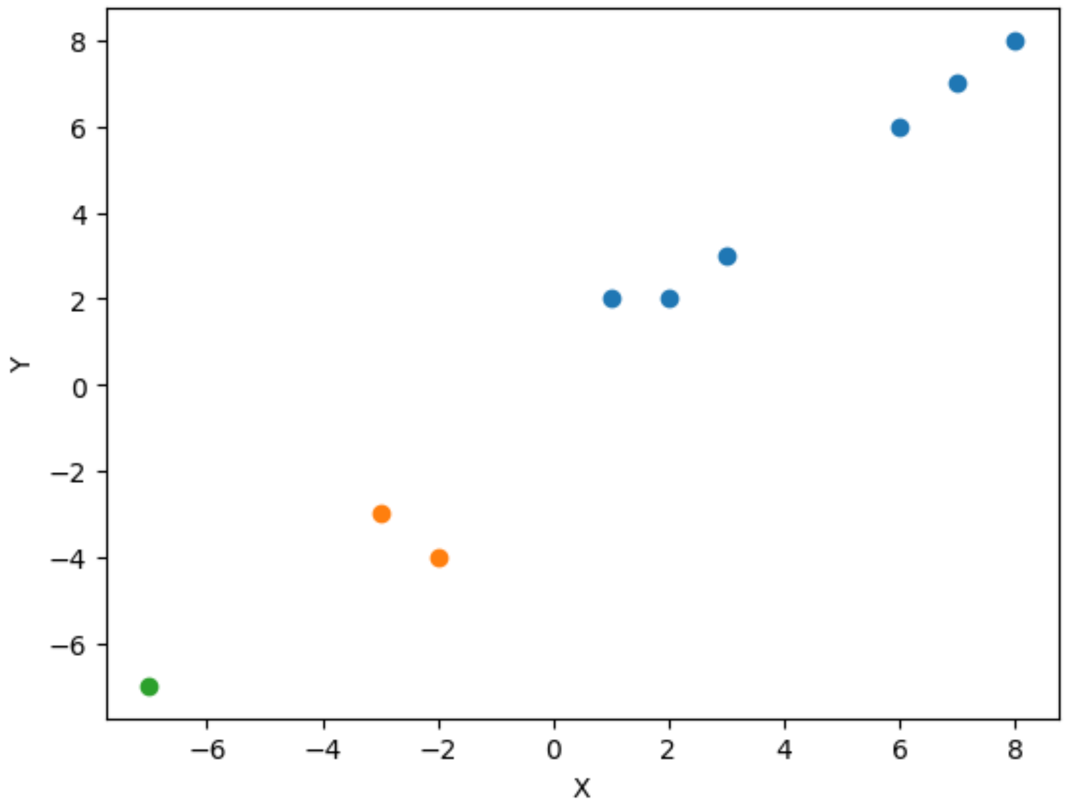


0.1.2 T6. starting points are (-3,-3), (2,2), and (-7,-7),

[3]:
```
init_t6 = [(-3, -3), (2, 2), (-7, -7)]

km_t6 = KMeans(init_t6)
km_t6.cluster()
```

```
--- Round 1 ---
Centroids: [(np.float64(4.5), np.float64(4.666666666666667)), (np.float64(-2.5),
np.float64(-3.5)), (np.float64(-7.0), np.float64(-7.0))]
--- Round 2 ---
Centroids: [(np.float64(4.5), np.float64(4.666666666666667)), (np.float64(-2.5),
np.float64(-3.5)), (np.float64(-7.0), np.float64(-7.0))]
```



0.1.3 T7. Between the two starting set of points in the previous two questions, which one do you think is better? How would you measure the 'goodness' quality of a set of starting points?

0.1.4 In general, it is important to try different sets of starting points when doing k-means.

```
For me, the better one is the first one.
The 'goodness' may measure with the variance
in the group
```

0.1.5 OT2. What would be the best K for this question? Describe your reasoning.

```
For me, it's 4 because the distance between (-7, -7)
to (-3, -3) and (-2, -4) is too large to be in the
same group. The distances are estimately equal to
distances between centroids of each group.
```

0.2 Regression

[4]:
```
train_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.
  ↪csv"
train = pd.read_csv(train_url)

test_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv"
test = pd.read_csv(test_url)
```

[5]:
```
train.describe()
```

[5]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

0.2.1 T8. Median of the training datasets?

[6]:
```
train["Age"].median()
```

[6]:
```
np.float64(28.0)
```

28

0.2.2 T9. Mode of Embarked

[7]:
```
train["Embarked"].mode().iloc[0]
```

[7]:
```
'S'
```

"S" which is 0

0.2.3 T10.

[8]:
```
test
```

[8]:
```
     PassengerId  Pclass                                          Name  \
0            892       3                              Kelly, Mr. James
1            893       3              Wilkes, Mrs. James (Ellen Needs)
2            894       2                     Myles, Mr. Thomas Francis
3            895       3                              Wirz, Mr. Albert
4            896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
..           …     …                                           …
413         1305       3                            Spector, Mr. Woolf
414         1306       1                  Oliva y Ocana, Dona. Fermina
415         1307       3                  Saether, Mr. Simon Sivertsen
416         1308       3                           Ware, Mr. Frederick
417         1309       3                      Peter, Master. Michael J

        Sex   Age  SibSp  Parch              Ticket      Fare Cabin Embarked
0      male  34.5      0      0              330911    7.8292   NaN        Q
1    female  47.0      1      0              363272    7.0000   NaN        S
2      male  62.0      0      0              240276    9.6875   NaN        Q
3      male  27.0      0      0              315154    8.6625   NaN        S
4    female  22.0      1      1             3101298   12.2875   NaN        S
..      …   …    …    …                 …       …   …      …
413    male   NaN      0      0           A.5. 3236    8.0500   NaN        S
414  female  39.0      0      0            PC 17758  108.9000  C105        C
415    male  38.5      0      0  SOTON/O.Q. 3101262    7.2500   NaN        S
416    male   NaN      0      0              359309    8.0500   NaN        S
417    male   NaN      1      1                2668   22.3583   NaN        C

[418 rows x 11 columns]
```

[9]:
```python
import pandas as pd
import numpy as np
import os

class LogisticRegression:
    def __init__(self, train, test, feature_cols, target_col, learning_rate=0.
↪001, iters=100000, threshold=0.5):
        self.train = train
        self.test = test
        self.feature_cols = feature_cols
        self.target_col = target_col
        self.learning_rate = learning_rate
        self.iters = iters
        self.threshold = threshold
        self.theta = np.zeros((len(self.feature_cols), 1)) # column vector
```

```python
    def fit(self):
        self.process()
        x = np.array(self.train[self.feature_cols].values)
        y = np.array(self.train[self.target_col].values).reshape(-1, 1)
        for i in range(self.iters):
            self.theta += self.learning_rate / x.shape[0] * (x.T @ (y - self.h(x @ self.theta)))

        print(f"trained with learning rate: {self.learning_rate} and iterations: {self.iters}")

    def predict(self, title="submission", directory="submission"):
        self.test["Survived"] = self.h(np.array(self.test[self.feature_cols].values) @ self.theta)
        self.test.loc[self.test["Survived"] < self.threshold, "Survived"] = 0
        self.test.loc[self.test["Survived"] >= self.threshold, "Survived"] = 1
        self.test["Survived"] = self.test["Survived"].astype(int)

        os.makedirs(directory, exist_ok=True)
        self.test[["PassengerId", "Survived"]].to_csv(f"{directory}/{title}.csv", index=False)
        print(f"Saved to {directory}/{title}.csv")

    def params(self):
        print(self.theta)

    def h(self, x):
        return 1 / (1 + np.exp(-x))

    def process(self):
        self.train["Age"] = self.train["Age"].fillna(self.train["Age"].median())
        self.test["Age"] = self.test["Age"].fillna(self.train["Age"].median())

        self.train.loc[self.train["Embarked"] == "S", "Embarked"] = 0
        self.train.loc[self.train["Embarked"] == "C", "Embarked"] = 1
        self.train.loc[self.train["Embarked"] == "Q", "Embarked"] = 2

        self.test.loc[self.test["Embarked"] == "S", "Embarked"] = 0
        self.test.loc[self.test["Embarked"] == "C", "Embarked"] = 1
        self.test.loc[self.test["Embarked"] == "Q", "Embarked"] = 2

        self.train["Embarked"] = self.train["Embarked"].fillna(self.train["Embarked"].mode().iloc[0])
        self.test["Embarked"] = self.test["Embarked"].fillna(self.train["Embarked"].mode().iloc[0])
```

```
        self.train.loc[self.train["Sex"] == "male", "Sex"] = 0
        self.train.loc[self.train["Sex"] == "female", "Sex"] = 1

        self.test.loc[self.test["Sex"] == "male", "Sex"] = 0
        self.test.loc[self.test["Sex"] == "female", "Sex"] = 1
        # print(self.test[self.feature_cols].dtypes)

        self.train[self.feature_cols] = self.train[self.feature_cols].
↪astype(float)
        self.test[self.feature_cols] = self.test[self.feature_cols].
↪astype(float)
```

[10]:
```
lr = LogisticRegression(train, test, ["Pclass","Sex","Age","Embarked"],␣
↪"Survived",iters=100000)
lr.fit()
lr.params()
lr.predict(title="T10_100k_iters")
```

```
C:\Users\chyut\AppData\Local\Temp\ipykernel_22544\3690440150.py:53:
FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is
deprecated and will change in a future version. Call
result.infer_objects(copy=False) instead. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  self.train["Embarked"] =
self.train["Embarked"].fillna(self.train["Embarked"].mode().iloc[0])
C:\Users\chyut\AppData\Local\Temp\ipykernel_22544\3690440150.py:54:
FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is
deprecated and will change in a future version. Call
result.infer_objects(copy=False) instead. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  self.test["Embarked"] =
self.test["Embarked"].fillna(self.train["Embarked"].mode().iloc[0])
```

```
trained with learning rate: 0.001 and iterations: 100000
[[-0.70281072]
 [ 2.59431049]
 [-0.00447441]
 [ 0.35126993]]
Saved to submission/T10_100k_iters.csv
```

[11]:
```
lr = LogisticRegression(train, test, ["Pclass","Sex","Age","Embarked"],␣
↪"Survived",iters=2000000)
lr.fit()
lr.params()
lr.predict(title="T10_2m_iters")
```

```
trained with learning rate: 0.001 and iterations: 2000000
[[-0.71333809]
 [ 2.65049353]
```

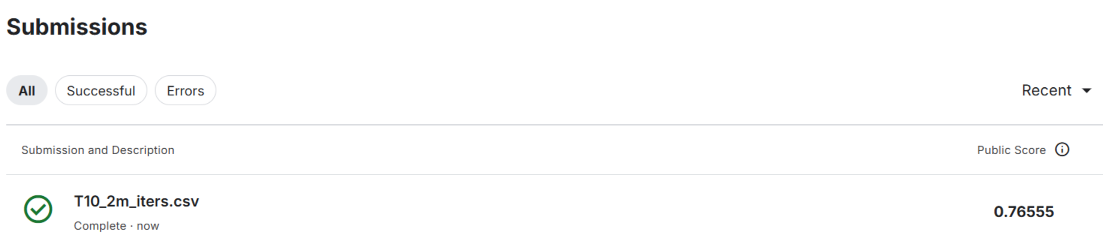
```
 [-0.00447566]
 [ 0.35092113]]
Saved to submission/T10_2m_iters.csv
```

[12]:
```python
print(pd.read_csv("submission/T10_2m_iters.csv").set_index("PassengerId"))
```

```
             Survived
PassengerId
892                 0
893                 1
894                 0
895                 0
896                 1
…                 …
1305                0
1306                1
1307                0
1308                0
1309                0

[418 rows x 1 columns]
```

0.2.4 T11.



0.2.5 T12.

[13]:
```python
train_high = train.copy()
train_high["Embarked**2"] = train_high["Embarked"] ** 2
train_high["Age*Pclass"] = train_high["Age"] * train_high["Pclass"]
train_high["Age**2"] = train_high["Age"] ** 2
train_high["Age*Sex"] = train_high["Age"] * train_high["Sex"]

test_high = test.copy()
test_high["Embarked**2"] = test_high["Embarked"] ** 2
test_high["Age*Pclass"] = test_high["Age"] * test_high["Pclass"]
test_high["Age**2"] = test_high["Age"] ** 2
test_high["Age*Sex"] = test_high["Age"] * test_high["Sex"]
```

[14]:
```python
```

```
lr = LogisticRegression(train_high, test_high, ["Pclass","Sex","Age","Embarked", "Embarked**2", "Age*Pclass", "Age**2", "Age*Sex"], "Survived",iters=100000)
lr.fit()
lr.params()
lr.predict(title="T12_100k_iters")
```

```
C:\Users\chyut\AppData\Local\Temp\ipykernel_22544\3690440150.py:39:
RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-x))
```

```
trained with learning rate: 0.001 and iterations: 100000
[[-0.52522612]
 [-2.49243163]
 [ 2.86031408]
 [ 0.8270162 ]
 [ 0.7425793 ]
 [-1.47554321]
 [ 0.01012895]
 [ 7.54272117]]
Saved to submission/T12_100k_iters.csv
```

Submission and Description	Public Score
T13_100k_iters.csv Complete · 1h ago	0.75358
T12_100k_iters.csv Complete · 1h ago	0.58851

0.2.6 T13.

[15]:
```
lr = LogisticRegression(train_high, test_high, ["Age", "Sex"], "Survived",iters=100000)
lr.fit()
lr.params()
lr.predict(title="T13_100k_iters")
```

```
trained with learning rate: 0.001 and iterations: 100000
[[-0.04000972]
 [ 2.21648735]]
Saved to submission/T13_100k_iters.csv
```

Submission and Description	Public Score
T13_100k_iters.csv Complete · 1h ago	0.75358
T12_100k_iters.csv Complete · 1h ago	0.58851

0.2.7 OT3.

[16]:
```
import pandas as pd
import numpy as np
import os

class LinearRegression:
    def __init__(self, train, test, feature_cols, target_col, learning_rate=0.
 ↪001, iters=100000, threshold=0.5):
        self.train = train
        self.test = test
        self.feature_cols = feature_cols
        self.target_col = target_col
        self.learning_rate = learning_rate
        self.iters = iters
        self.threshold = threshold
        self.theta = np.zeros((len(self.feature_cols), 1)) # column vector

    def fit(self):
        self.process()
        x = np.array(self.train[self.feature_cols].values)
        y = np.array(self.train[self.target_col].values).reshape(-1, 1)
        for i in range(self.iters):
            self.theta += self.learning_rate / x.shape[0] * (x.T @ (y - (x @␣
 ↪self.theta)))

        print(f"trained with learning rate: {self.learning_rate} and iterations:
 ↪ {self.iters}")

    def params(self):
        print(self.theta)
        return self.theta

    def process(self):
        self.train["Age"] = self.train["Age"].fillna(self.train["Age"].median())
        self.test["Age"] = self.test["Age"].fillna(self.train["Age"].median())

        self.train.loc[self.train["Embarked"] == "S", "Embarked"] = 0
        self.train.loc[self.train["Embarked"] == "C", "Embarked"] = 1
        self.train.loc[self.train["Embarked"] == "Q", "Embarked"] = 2

        self.test.loc[self.test["Embarked"] == "S", "Embarked"] = 0
        self.test.loc[self.test["Embarked"] == "C", "Embarked"] = 1
        self.test.loc[self.test["Embarked"] == "Q", "Embarked"] = 2

        self.train["Embarked"] = self.train["Embarked"].fillna(self.
 ↪train["Embarked"].mode().iloc[0])
```

```
        self.test["Embarked"] = self.test["Embarked"].fillna(self.
 ↪train["Embarked"].mode().iloc[0])

        self.train.loc[self.train["Sex"] == "male", "Sex"] = 0
        self.train.loc[self.train["Sex"] == "female", "Sex"] = 1

        self.test.loc[self.test["Sex"] == "male", "Sex"] = 0
        self.test.loc[self.test["Sex"] == "female", "Sex"] = 1
        # print(self.test[self.feature_cols].dtypes)

        self.train[self.feature_cols] = self.train[self.feature_cols].
 ↪astype(float)
        self.test[self.feature_cols] = self.test[self.feature_cols].
 ↪astype(float)
```

[17]:
```
lr = LinearRegression(train, test, ["Pclass","Sex","Age","Embarked"],␣
 ↪"Survived",iters=1000000)
lr.fit()
theta_ot3 = lr.params()
```

```
trained with learning rate: 0.001 and iterations: 1000000
[[-0.01411427]
 [ 0.60420619]
 [ 0.00501483]
 [ 0.06116326]]
```

0.2.8 OT4.

[18]:
```
x = np.array(train[["Pclass","Sex","Age","Embarked"]])
y = np.array(train["Survived"]).reshape((-1, 1))
theta_ot4 = np.linalg.inv(x.T @ x) @ (x.T @ y)
theta_ot4
```

[18]:
```
array([[-0.01411427],
       [ 0.60420619],
       [ 0.00501483],
       [ 0.06116326]])
```

[19]:
```
mse = np.sum((theta_ot3 - theta_ot4) ** 2) / theta_ot3.shape[1]
mse
```

[19]:
```
np.float64(5.471194131828056e-26)
```

OT5. $\nabla_A \operatorname{tr}(AB) = B^T$

จาก $(AB)_{ij} = \sum_m A_{im} B_{mj}$

as trace is diagonal sum, so it is $\sum_k (AB)_{kk} = \sum_k \sum_m A_{km} B_{mk} = \sum_i \sum_j A_{ij} B_{ji}$

ให้ $f = (\sum_i \sum_j A_{ij} B_{ji})$

$$\nabla_A \operatorname{tr}(AB) = \frac{\partial}{\partial A}\left(\sum_i \sum_j A_{ij} B_{ji}\right) = \frac{\partial f}{\partial A} = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \frac{\partial f}{\partial A_{12}} & \cdots & \frac{\partial f}{\partial A_{1N}} \\ \vdots & \ddots & & \vdots \\ \frac{\partial f}{\partial A_{N1}} & \cdots & & \frac{\partial f}{\partial A_{NN}} \end{bmatrix}$$

ในตำแหน่งที่ (i,j) ทุกพจน์เป็น 0 หมด ยกเว้นพจน์ที่คูณกับ A_{ij} นั่นคือ B_{ji}

$$= \begin{bmatrix} B_{11} & B_{21} & \cdots & B_{N1} \\ \vdots & \ddots & & \vdots \\ B_{1N} & \cdots & & B_{NN} \end{bmatrix} = B^T \ \#$$

OT6. $\nabla_{A^T} f(A) = (\nabla_A f(A))^T$

ให้ $G = \nabla_A f(A)$ ที่ทำให้ $G_{ij} = \frac{\partial}{\partial A_{ij}} f$

สำหรับ $G^T = (\nabla_A f(A))^T$ ที่ทำให้ $G^T_{ij} = (\nabla_A f(A))^T_{ij} = \frac{\partial}{\partial A_{ji}} f$

กำหนด $B = A^T$; $B_{ji} = A_{ij}$

$$\nabla_B f_{ji} = \nabla_{A^T} f_{ji} = \frac{\partial f}{\partial B_{ji}} = \frac{\partial f}{\partial A_{ij}}$$

$$\therefore \nabla_{A^T} f_{ij} = \frac{\partial}{\partial A_{ji}} f$$

$\therefore$ พบว่า $\nabla_{A^T} f = (\nabla_A f(A))^T$ #