HW1

January 24, 2024

Model A	Predicted Dog	Predicted Cat
Actual Dog	30	20
Actual Cat	10	40

T1. Accuracy $= \frac{30+40}{30+20+10+40} = 70\%$

T2. Consider cat as class 1

Precision $= \frac{40}{20+40} = 66.67\%$

Recall $= \frac{40}{10+40} = 80\%$

F1 $= 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{0.6667 \cdot 0.8}{0.6667 + 0.8} = 0.7273$

T3. Consider class cat as class 0

$Precision = \frac{30}{30+10} = 75\%$

Recall $= \frac{30}{30+20} = 60\%$

$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{0.75 \cdot 0.6}{0.75 + 0.6} = 0.6667$

T4. Same model with a new population with 80% cat, consider dog as positive class.
Find accuracy, precision, recall, and F1 score of a model.

Let x is number of new population.
Assume that recall of each class does not change (since it's the most make sense for the same model)

Model A	Predicted Dog	Predicted Cat
Actual Dog	$0.6 \cdot 0.2x$	$0.4 \cdot 0.2x$
Actual Cat	$0.2 \cdot 0.8x$	$0.8 \cdot 0.8x$

Accuracy $= \frac{0.6 \cdot 0.2x + 0.8 \cdot 0.8x}{x} = 76\%$

$$\text{Precision} = \frac{0.6 \cdot 0.2x}{0.6 \cdot 0.2x + 0.2 \cdot 0.8x} = 42.86\%$$

$$\text{Recall} = \frac{0.6 \cdot 0.2x}{0.6 \cdot 0.2x + 0.4 \cdot 0.2x} = 60\%$$

$$\text{F1} = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{0.4286 \cdot 0.6}{0.4286 + 0.6} = 0.5000$$

OT1. let Accuracy = F1

$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{2TP}{2TP + FP + FN}$$

$$2TP^2 + 2TPTN + FPTP + FPTN + FNTP + FNTN = 2TP^2 + 2TPTN + 2TPFP + 2TPFN$$

$$FPTN + FNTN = FPTP + FNTP$$

$$TN(FP + FN) = TP(FP + FN)$$

$$TN = TP$$

∴Accuracy will be equal F1 when $TN = TP$
Same go as greater and less, since all value is positive in inequality.
∴Accuracy will be greater F1 when $TN > TP$
∴Accuracy will be less F1 when $TN < TP$

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

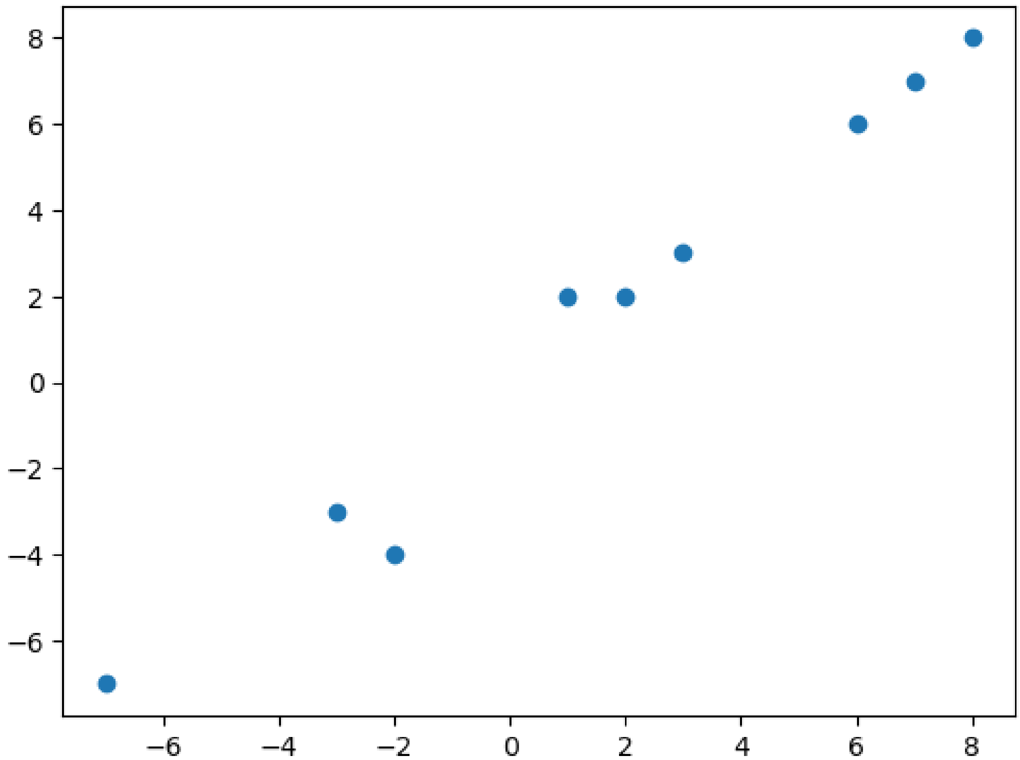
[ ]:
```
# Given data point
x = np.array([1, 3, 2, 8, 6, 7, -3, -2, -7])
y = np.array([2, 3, 2, 8, 6, 7, -3, -4, -7])
ar = np.array([
    [1, 2],
    [3, 3],
    [2, 2],
    [8, 8],
    [6, 6],
    [7, 7],
    [-3, -3],
    [-2, -4],
    [-7, -7],
])
# df.rename({0: 'x', 1: 'y'}, axis=1, inplace=True)
ar
```

[ ]:
```
array([[ 1,  2],
       [ 3,  3],
       [ 2,  2],
       [ 8,  8],
       [ 6,  6],
```

```
       [ 7,  7],
       [-3, -3],
       [-2, -4],
       [-7, -7]])
```

[ ]:
```
plt.scatter(x, y)
```

[ ]: <matplotlib.collections.PathCollection at 0x1904db3e350>

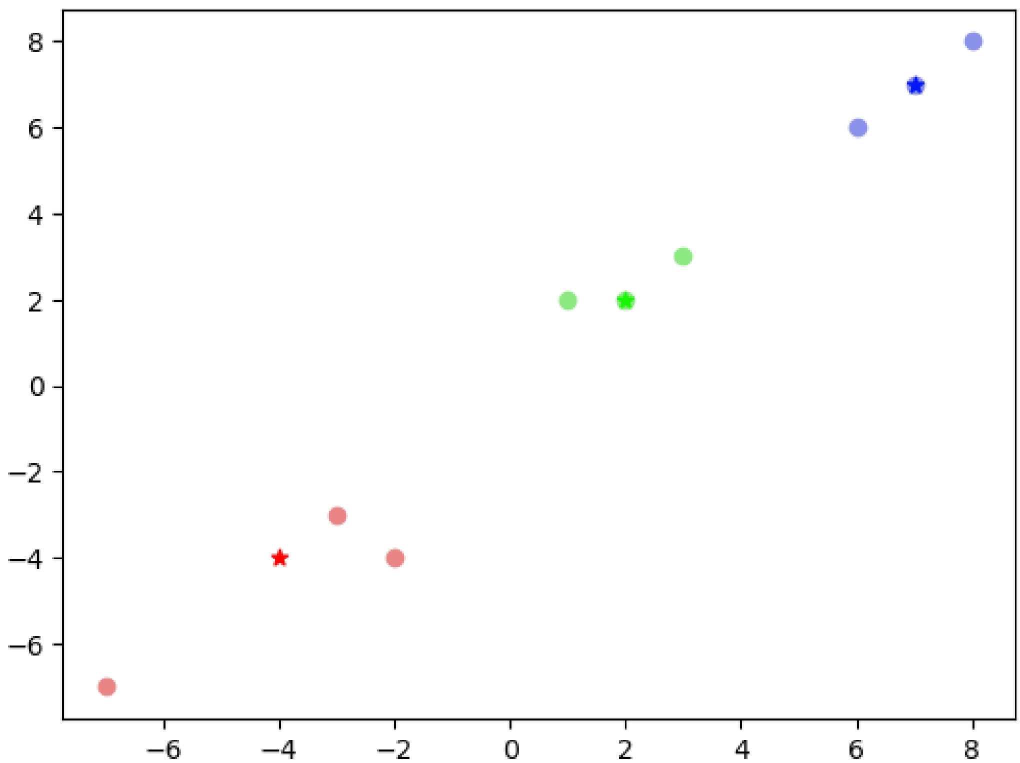


[ ]:
```
# color for plotting
pcolor = ['#e88484', '#8de884', '#8991e8']
ccolor = ['#ff0000', '#17f502', '#0216f5']
```

[ ]:
```
class Kmean():
    def __init__(self, k, stpnt):
        self.k = k
        self.stpnt = np.array(stpnt)

    def assign_points(self, X, centroids):
        # print("Assign points")
        clusters = [[] for i in range(self.k)]
        for p in X:
```

```
            cen = np.argmin(np.linalg.norm(centroids-p, axis=1)) # Euclidian␣
↪distance
            clusters[cen].append(p)
        # for i, cl in enumerate(clusters):
        #     print("cluster", i, ": ", end="")
        #     for p in cl:
        #         print(f"({p[0]}, {p[1]})", end=", ")
        #     print('')
        return clusters

    def update_centroid(self, clusters):
        # print("Update centroids")
        centroids = np.array([[0, 0] for i in range(self.k)])
        for i, cl in enumerate(clusters):
            centroids[i] = np.mean(cl, axis=0)
        # for i, cen in enumerate(centroids):
        #     print("centroid", i, ": ", f"({cen[0]},{cen[1]})")
        return centroids

    def fit(self, X):
        new_centroids = np.array(self.stpnt)
        centroids = np.zeros(self.k)
        i = 1
        while not np.array_equal(new_centroids, centroids):
            centroids = new_centroids
            # print('epoch :',i)
            i += 1
            clusters = self.assign_points(X, centroids)
            new_centroids = self.update_centroid(clusters)
        return clusters, centroids
```

[ ]:
```
# T5 model: starting point at [-3, -3], [2, 2], [3, 3]
model = Kmean(3, [[-3, -3], [2, 2], [3, 3]])
clusters, centroids = model.fit(ar)
for i, cl in enumerate(clusters):
            color = pcolor[i]
            for p in cl:
                plt.scatter(p[0], p[1], color = color)
for i, cen in enumerate(centroids):
            color = ccolor[i]
            plt.scatter(cen[0], cen[1], color = color, marker='*')
plt.show()
# stars are centroids
```

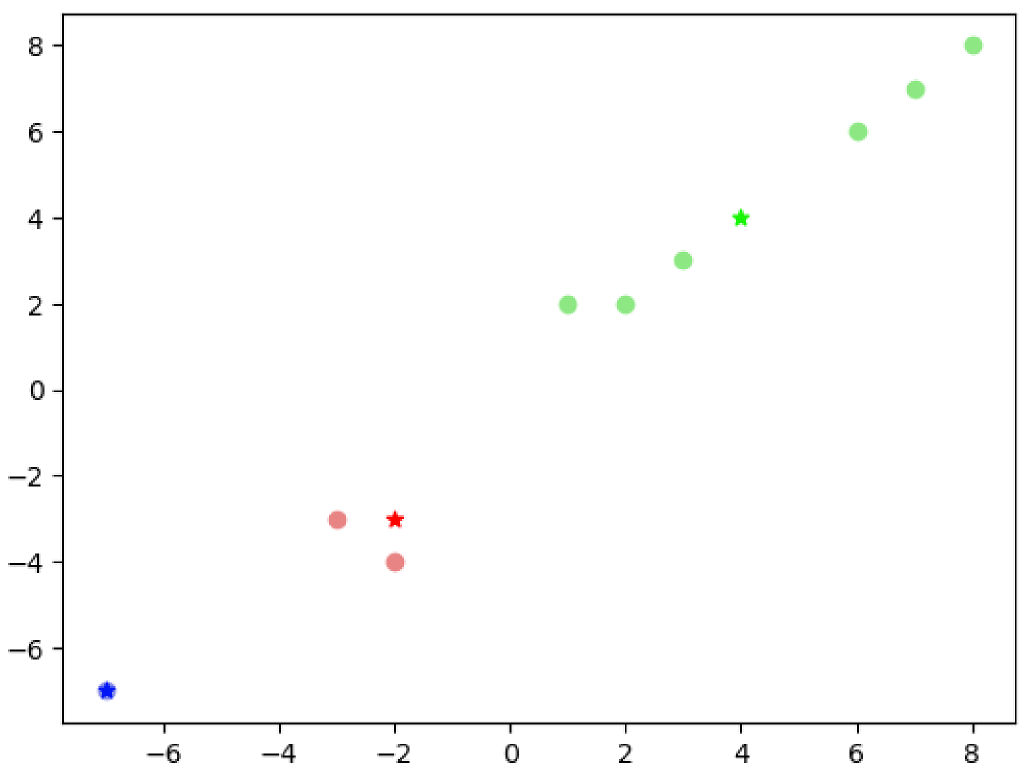


T5:
epoch : 1
Assign points
cluster 0 : (-3, -3), (-2, -4), (-7, -7)
cluster 1 : (1, 2), (2, 2)
cluster 2 : (3, 3), (8, 8), (6, 6), (7, 7)
Update centroids
centroid 0 : (-4, -4)
centroid 1 : (1, 2)
centroid 2 : (6, 6)
epoch : 2
Assign points
cluster 0 : (-3, -3), (-2, -4), (-7, -7)
cluster 1 : (1, 2), (3, 3), (2, 2)
cluster 2 : (8, 8), (6, 6), (7, 7)
Update centroids
centroid 0 : (-4, -4)
centroid 1 : (2, 2)
centroid 2 : (7, 7)
epoch : 3
Assign points
cluster 0 : (-3, -3), (-2, -4), (-7, -7)

cluster 1 : (1, 2), (3, 3), (2, 2)
cluster 2 : (8, 8), (6, 6), (7, 7)
Update centroids
centroid 0 : (-4, -4)
centroid 1 : (2, 2)
centroid 2 : (7, 7)
(output from model below)

T6: As seen from visualization below, starting points from T6 model change output centroids.

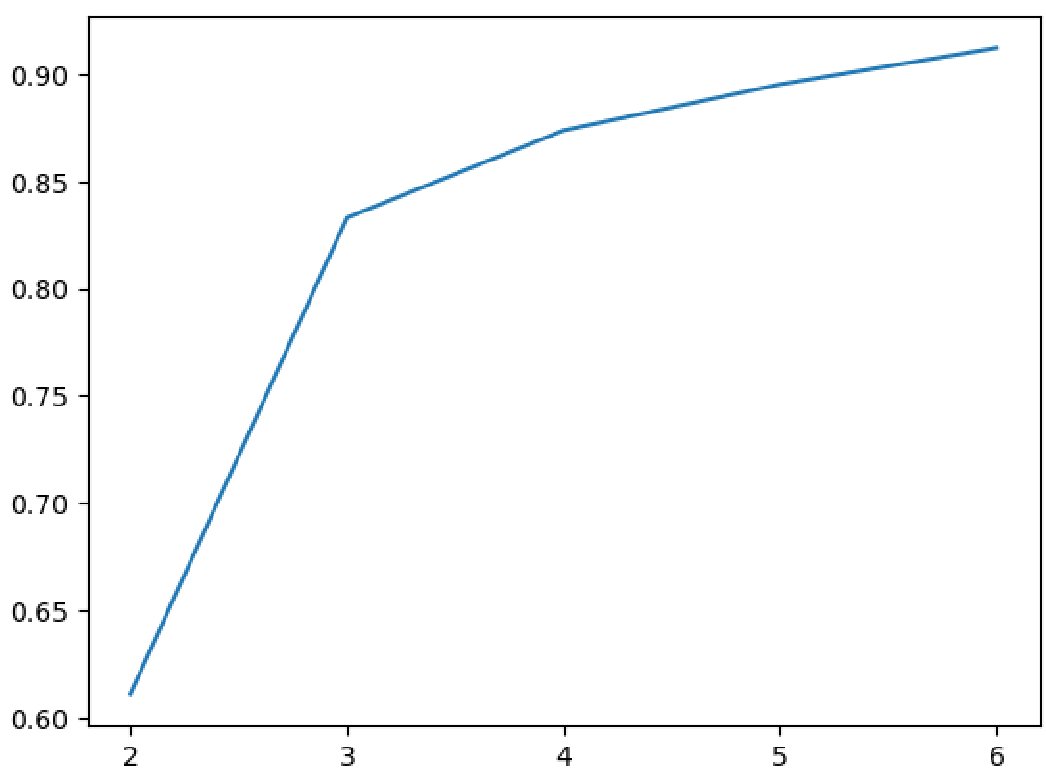
[ ]:
```
# T6 model
model = Kmean(3, [[-3, -3], [2, 2], [-7, -7]])
clusters, centroids = model.fit(ar)
for i, cl in enumerate(clusters):
            color = pcolor[i]
            for p in cl:
                plt.scatter(p[0], p[1], color = color)
for i, cen in enumerate(centroids):
            color = ccolor[i]
            plt.scatter(cen[0], cen[1], color = color, marker='*')
plt.show()
# stars are centroids
```



T7: T5 one is better, I think 'goodness' of starting points is the less initial between cluster variance.

[ ]:
```
# OT2
import random
def foevar(k):
    fvars = []
    allcen = np.mean(ar, axis=0)
    for j in range(100):
        clusters, centroids = Kmean(k, random.sample(ar.tolist(), k)).fit(ar)
        btwvar = 0
        allvar = 0
        for i in range(k):
            btwvar += len(clusters[i]) * ((centroids[i][0] - allcen[0]) ** 2 +␣
↪(centroids[i][1] - allcen[1]) ** 2) /  ar.shape[0]
        for i in range(ar.shape[0]):
            allvar += ((ar[i][0] - allcen[0]) ** 2 + (ar[i][1] - allcen[1]) **␣
↪2) / ar.shape[0]
        frac = btwvar / allvar # fraction of explained variance
        fvars.append(frac)
    return np.mean(fvars)
```

[ ]:
```
plt.plot([(foevar(i)) for i in range(2, 7)])
plt.xticks(range(5), labels=[str(i + 2) for i in range(5)])
plt.show()
```



OT2: Use Elbow method, The best K is 3.

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

[ ]:
```
# T8
median_age = train['Age'].median()
median_age
```

[ ]:
```
28.0
```

T8: Median age of training set is 28.0

[ ]:
```
train["Age"] = train["Age"].fillna(train["Age"].median())
train['Age'].isnull().sum()
```

[ ]:
```
0
```

T9: map Embarked and Sex to numerical value.

[ ]:
```
# T9
train['Embarked'].value_counts(dropna=False)
```

[ ]:
```
S      644
C      168
Q       77
NaN      2
Name: Embarked, dtype: int64
```

[ ]:
```
train['Embarked'] = train['Embarked'].fillna(train['Embarked'].mode()[0])
train['Embarked'].value_counts(dropna=False)
```

[ ]:
```
S    646
C    168
Q     77
Name: Embarked, dtype: int64
```

[ ]:
```
train.loc[train["Embarked"] == "S", "Embarked"] = 0
train.loc[train["Embarked"] == "C", "Embarked"] = 1
train.loc[train["Embarked"] == "Q", "Embarked"] = 2
train['Embarked'].value_counts(dropna=False)
```

[ ]:
```
0    646
1    168
2     77
Name: Embarked, dtype: int64
```

[ ]:
```python
train['Sex'].value_counts(dropna=False)
```

[ ]:
```
male      577
female    314
Name: Sex, dtype: int64
```

[ ]:
```python
train.loc[train['Sex'] == 'male', 'Sex'] = 0
train.loc[train['Sex'] == 'female', 'Sex'] = 1
train['Sex'].value_counts(dropna=False)
```

[ ]:
```
0    577
1    314
Name: Sex, dtype: int64
```

T10: Logistic regression

[ ]:
```python
# start T10
# Preprocess test data
test['Age'].isnull().sum()
```

[ ]:
```
86
```

[ ]:
```python
test["Age"] = test["Age"].fillna(median_age)
test['Age'].isnull().sum()
```

[ ]:
```
0
```

[ ]:
```python
test['Embarked'].value_counts(dropna=False)
```

[ ]:
```
S    270
C    102
Q     46
Name: Embarked, dtype: int64
```

[ ]:
```python
test.loc[test["Embarked"] == "S", "Embarked"] = 0
test.loc[test["Embarked"] == "C", "Embarked"] = 1
test.loc[test["Embarked"] == "Q", "Embarked"] = 2
test['Embarked'].value_counts(dropna=False)
```

[ ]:
```
0    270
1    102
2     46
Name: Embarked, dtype: int64
```

[ ]:
```
test['Sex'].value_counts(dropna=False)
```

[ ]:
```
male      266
female    152
Name: Sex, dtype: int64
```

[ ]:
```
test.loc[test['Sex'] == 'male', 'Sex'] = 0
test.loc[test['Sex'] == 'female', 'Sex'] = 1
test['Sex'].value_counts(dropna=False)
```

[ ]:
```
0    266
1    152
Name: Sex, dtype: int64
```

[ ]:
```
train_X = np.array(train[['Pclass', 'Sex', 'Age', 'Embarked']].values,␣
 ↪dtype=float)
test_X = np.array(test[['Pclass', 'Sex', 'Age', 'Embarked']].values,␣
 ↪dtype=float)
y = np.array(train['Survived'].values, dtype=float)
```

[ ]:
```
# normalize data
mx = np.max(train_X, axis=0)
mn = np.min(train_X, axis=0)
X_norm = (train_X - mn) / (mx - mn)
X_test_norm = (test_X - mn) / (mx - mn)
X_norm
```

[ ]:
```
array([[1.        , 0.        , 0.27117366, 0.        ],
       [0.        , 1.        , 0.4722292 , 0.5       ],
       [1.        , 1.        , 0.32143755, 0.        ],
       …,
       [1.        , 1.        , 0.34656949, 0.        ],
       [0.        , 0.        , 0.32143755, 0.5       ],
       [1.        , 0.        , 0.39683338, 1.        ]])
```

[ ]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# T10
class LogisticRegression():
    def __init__(self, lnr, epoch):
        self.lnr = lnr
        self.epoch = epoch
        self.theta = None # weight

    def fit(self, X, y):
        n_samples, n_attr = X.shape
```

```
        self.theta = np.zeros(n_attr)
        for _ in range(self.epoch):
            y_pred = sigmoid(np.dot(X, self.theta))
            self.theta = self.theta - self.lnr * (np.dot(X.T, (y_pred - y))) / n_samples

    def predict(self, X):
        y_pred = sigmoid(np.dot(X, self.theta))
        return [0 if yi <= 0.5 else 1 for yi in y_pred]

    def accuracy(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)
```

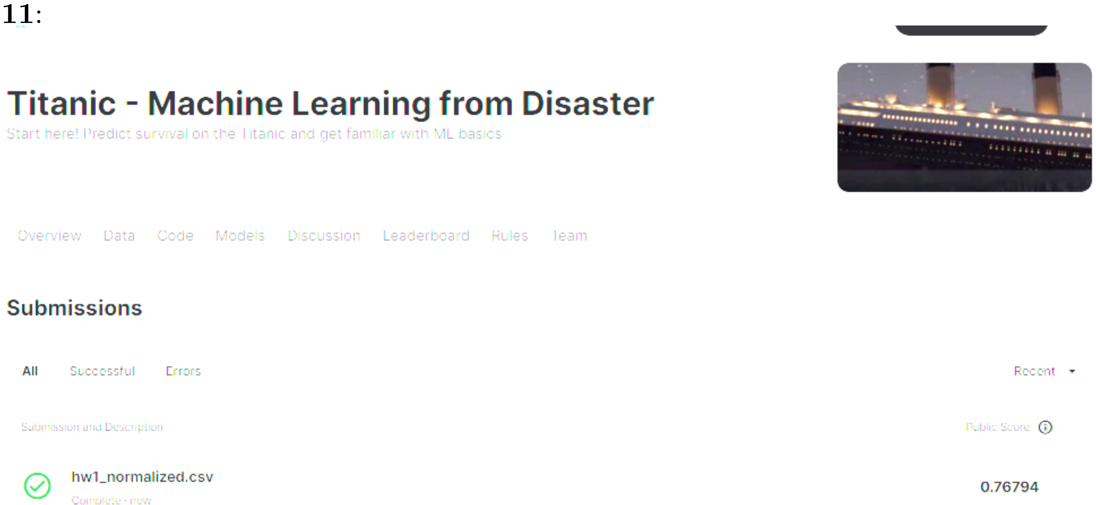
[ ]:
```
model = LogisticRegression(0.01, 100000)
model.fit(X_norm, y)
```

[ ]:
```
print('Training accuracy: ', model.accuracy(X_norm, y))
```

```
Training accuracy:  0.7912457912457912
```

[ ]:
```
pred = model.predict(X_test_norm)
output = pd.DataFrame({'PassengerId': test.PassengerId, 'Survived': pred})
output.to_csv('hw1_normalized.csv', index=False)
```

T11:



[ ]:
```
# T12: add high order feature
age_sq = train['Age'] ** 3
mx = np.max(age_sq, axis=0)
mn = np.min(age_sq, axis=0)
age_sq = (age_sq - mn) / (mx - mn)
highorder_X = np.insert(X_norm, 4, age_sq, axis=1)
age_sq_test = test['Age'] ** 3
```

```
mx = np.max(age_sq_test, axis=0)
mn = np.min(age_sq_test, axis=0)
age_sq_test = (age_sq_test - mn) / (mx - mn)
X_test_highorder = np.insert(X_test_norm, 4, age_sq_test, axis=1)
```

[ ]:
```
highorder_X
```

[ ]:
```
array([[1.        , 0.        , 0.27117366, 0.        , 0.02079673],
       [0.        , 1.        , 0.4722292 , 0.5       , 0.10717175],
       [1.        , 1.        , 0.32143755, 0.        , 0.03432799],
       …,
       [1.        , 1.        , 0.34656949, 0.        , 0.04287486],
       [0.        , 0.        , 0.32143755, 0.5       , 0.03432799],
       [1.        , 0.        , 0.39683338, 1.        , 0.06399986]])
```

[ ]:
```
model2 = LogisticRegression(0.01, 100000)
model2.fit(highorder_X, y)
print('Training accuracy with high order attribute: ', model2.
 ↪accuracy(highorder_X, y))
```

```
Training accuracy with high order attribute:  0.7934904601571269
```

[ ]:
```
pred2 = model2.predict(X_test_highorder)
output2 = pd.DataFrame({'PassengerId': test.PassengerId, 'Survived': pred2})
output2.to_csv('highorder.csv', index=False)
```

Score on kaggle: 0.76794
T12: It got slightly more training accuracy, but perform as good as the old one in test set.

[ ]:
```
# T13: Use just sex and age
model3 = LogisticRegression(0.01, 100000)
X_3 = X_norm[:, 1 : 3] # Sex and Age
X_3_test = X_test_norm[:, 1 : 3]
model3.fit(X_3, y)
model3.accuracy(X_3, y)
```

[ ]:
```
0.7789001122334456
```

[ ]:
```
pred3 = model3.predict(X_3_test)
output3 = pd.DataFrame({'PassengerId': test.PassengerId, 'Survived': pred3})
output3.to_csv('sex_age.csv', index=False)
```

Score on kaggle: 0.75119
T13: got slightly lower accuracy than use 4 attributes in T11.

OT3: Linear regression with gradient descent.

[ ]:
```
class LinearRegression():
    def __init__(self, lnr, epoch):
```

```
        self.lnr = lnr
        self.epoch = epoch
        self.theta = None # weight

    def fit(self, X, y):
        n_samples, n_attr = X.shape
        self.theta = np.zeros(n_attr)
        for _ in range(self.epoch):
            y_pred = np.dot(X, self.theta)
            self.theta = self.theta - self.lnr * (np.dot(X.T, (y_pred - y))) /␣
↪n_samples

    def predict(self, X):
        y_pred = np.dot(X, self.theta)
        return [0 if yi <= 0.5 else 1 for yi in y_pred]

    def accuracy(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)

    def weight(self):
        return self.theta
```

[ ]:
```
linear_gradient = LinearRegression(0.01, 100000)
linear_gradient.fit(X_norm, y)
print('Training accuracy:', linear_gradient.accuracy(X_norm, y))
```

```
Training accuracy: 0.7867564534231201
```

OT4: Linear regression with matrix inversion.

[ ]:
```
class MatrixInversion():
    def __init__(self):
        self.theta = None # weight

    def fit(self, X, y):
        self.theta = np.dot(np.dot(np.linalg.inv(np.dot(X.T, X)), X.T), y)

    def predict(self, X):
        y_pred = np.dot(X, self.theta)
        return [0 if yi <= 0.5 else 1 for yi in y_pred]

    def accuracy(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)

    def weight(self):
        return self.theta
```

[ ]:
```
linear_inversion = MatrixInversion()
linear_inversion.fit(X_norm, y)
print('Training accuracy:', linear_inversion.accuracy(X_norm, y))
```

```
Training accuracy: 0.7867564534231201
```

[ ]:
```
print('MSE of two weight =', np.sum((linear_gradient.weight() -␣
 ↪linear_inversion.weight()) ** 2))
```

```
MSE of two weight = 3.914473604603451e-27
```

which means weights learned from the two methods is similar.

QT 5. $\nabla_A \operatorname{tr} AB = B^T$

$$\operatorname{tr} AB = \operatorname{tr} \begin{bmatrix} \leftarrow & \bar{a}_1 & \rightarrow \\ \leftarrow & \bar{a}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \bar{a}_n & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \bar{b}_1 & \bar{b}_2 & \cdots & \bar{b}_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

$$= \operatorname{tr} \begin{bmatrix} \bar{a}_1^T \bar{b}_1 & \cdots & \bar{a}_1^T \bar{b}_n \\ \vdots & \ddots & \vdots \\ \bar{a}_n^T \bar{b} & \cdots & \bar{a}_n^T \bar{b}_n \end{bmatrix}$$

$$= \sum_{i=1}^{m} a_{1i} b_{i1} + \sum_{i=1}^{m} a_{2i} b_{i2} + \cdots + \sum_{i=1}^{m} a_{ni} b_{in}$$

$$\therefore \frac{\partial \operatorname{tr} AB}{a_{ij}} = b_{ji}$$

$$\therefore \operatorname{tr} AB = B^T$$

QT6 : $\nabla_{A^T} f(A) = (\nabla_A f(A))^T$

$$\nabla_{A^T} f(A) = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \cdots & \frac{\partial f(A)}{\partial A_{n1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{1n}} & \cdots & \frac{\partial f(A)}{\partial A_{nn}} \end{bmatrix}$$

$$= (\nabla_A f(A))^T$$

$$\text{OT 7}: \nabla_A \operatorname{tr} ABA^T C = CAB + C^T A B^T$$

At index m, n

$$(\nabla_A \operatorname{tr} ABA^T C)_{mn} = \sum_i \sum_j \sum_k \sum_l \frac{\partial}{\partial A_{mn}} \Big(\underbrace{A_{ij} B_{jk} A_{lk} C_{li}} \Big)$$

$\neq 0$ when $(i = m$ and $j = n)$
or $(l = m$ and $k = n)$

$i = m, \ j = n \qquad l = m, \ k = n$

$$= \sum_k \sum_l B_{nk} A_{lk} C_{lm} + \sum_i \sum_j A_{ij} B_{jn} C_{mi}$$

$$= \sum_k \sum_l C^T_{ml} A_{lk} B^T_{kn} + \sum_i \sum_j C_{mi} A_{ij} B_{jn}$$

$$= (C^T A B^T + CAB)_{mn}$$

$$\therefore \nabla_A \operatorname{tr} ABA^T C = C^T A B^T + CAB$$