Homework 1

Pana Wanitchollakit 6532136721

Metrics

T1.

$$\text{Accuracy of Model A} = \frac{30+40}{30+20+10+40} = 70\%$$

T2.

$$\text{Precision} = \frac{\sharp\text{True positive}}{\sharp\text{Predicted cat}} = \frac{40}{20+40} = 66.67\%$$

$$\text{Recall} = \frac{\sharp\text{True positive}}{\sharp\text{Actual cat}} = \frac{40}{10+40} = 80\%$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times 0.667 \times 0.8}{0.667 + 0.8} = 72.75\%$$

T3.

$$\text{Precision} = \frac{\sharp\text{True positive}}{\sharp\text{Predicted dog}} = \frac{30}{30+10} = 75\%$$

$$\text{Recall} = \frac{\sharp\text{True positive}}{\sharp\text{Actual dog}} = \frac{30}{30+20} = 60\%$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times 0.75 \times 0.6}{0.75 + 0.6} = 66.67\%$$

T4.

let n be the number of dataset. the proportion of dogs and cats are $0.2n$:$0.8n$

Model A	Predicted dog	Predicted cat
Actual dog	$(0.2n)\frac{30}{30+20} = 0.12n$	$(0.2n)\frac{20}{30+20} = 0.08n$
Actual cat	$(0.8n)\frac{10}{10+40} = 0.16n$	$(0.8n)\frac{40}{10+40} = 0.64n$

Re-calculate Precision, Recall, F1-Score

$$\text{Precision} = \frac{\sharp\text{True positive}}{\sharp\text{Predicted dog}} = \frac{0.12n}{0.12n + 0.16n} = 42.85\%$$

$$\text{Recall} = \frac{\sharp\text{True positive}}{\sharp\text{Actual dog}} = \frac{0.12n}{0.12n + 0.08n} = 60\%$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times 0.4285 \times 0.6}{0.4285 + 0.6} = 49.995\%$$

- Precision is changed because the number of false positives increased (denominator increase).
- Recall is not changed because it is calculated from the dogs only and it proportion is remain the same.
- F1-Score is changed because Precision changed.

OT1.

$$\text{Accuracy} = \text{F1-Score}$$
$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{2TP}{2TP + FP + FN}$$

$$(TP + TN)(2TP + FP + FN) = (2TP)(TP + TN + FP + FN)$$

$$\cancel{2(TP)^2} + \cancel{(TP)(FP)} + \cancel{(TP)(FN)} + \cancel{2(TP)(TN)} + (TN)(FP) + (TN)(FN)$$
$$= \cancel{2(TP)^2} + \cancel{2(TP)(TN)} + \cancel{2}(TP)(FP) + \cancel{2}(TP)(FN)$$

$$(TN)\cancel{(FP + FN)} = (TP)\cancel{(FP + FN)}$$
$$\therefore TN = TP$$

For both greater and less than cases, the inequality can be solved similarly.

- Accuracy = F1-Score if TN = TP
- Accuracy > F1-Score if TN > TP
- Accuracy < F1-Score if TN < TP

Hello Clustering

T5.

1. Attempt 1:
   Centroid 1 at (3.00, 3.00)

   $$\begin{aligned} \textbf{Assign}: &\quad (3.00, 3.00), (8.00, 8.00), (6.00, 6.00), (7.00, 7.00) \\ \textbf{Update}: &\quad (\frac{3.00 + 8.00 + 6.00 + 7.00}{4}, \frac{3.00 + 8.00 + 6.00 + 7.00}{4}) = (6.00, 6.00) \end{aligned}$$

   Centroid 2 at (2.00, 2.00)

   $$\begin{aligned} \textbf{Assign}: &\quad (1.00, 2.00), (2.00, 2.00) \\ \textbf{Update}: &\quad (\frac{1.00 + 2.00}{2}, \frac{2.00 + 2.00}{2}) = (1.50, 2.00) \end{aligned}$$

   Centroid 3 at (-3.00, -3.00)

   $$\begin{aligned} \textbf{Assign}: &\quad (-3.00, -3.00), (-2.00, -4.00), (-7.00, -7.00) \\ \textbf{Update}: &\quad (\frac{-3.00 + -2.00 + -7.00}{3}, \frac{-3.00 + -4.00 + -7.00}{3}) = (-4.00, -4.67) \end{aligned}$$

2. Attempt 2:
   Centroid 1 at (6.00, 6.00)

   $$\begin{aligned} \textbf{Assign}: &\quad (8.00, 8.00), (6.00, 6.00), (7.00, 7.00) \\ \textbf{Update}: &\quad (\frac{8.00 + 6.00 + 7.00}{3}, \frac{8.00 + 6.00 + 7.00}{3}) = (7.00, 7.00) \end{aligned}$$

   Centroid 2 at (1.50, 2.00)

   $$\begin{aligned} \textbf{Assign}: &\quad (1.00, 2.00), (3.00, 3.00), (2.00, 2.00) \\ \textbf{Update}: &\quad (\frac{1.00 + 3.00 + 2.00}{3}, \frac{2.00 + 3.00 + 2.00}{3}) = (2.00, 2.33) \end{aligned}$$

   Centroid 3 at (-4.00, -4.67)

   $$\begin{aligned} \textbf{Assign}: &\quad (-3.00, -3.00), (-2.00, -4.00), (-7.00, -7.00) \\ \textbf{Update}: &\quad (\frac{-3.00 + -2.00 + -7.00}{3}, \frac{-3.00 + -4.00 + -7.00}{3}) = (-4.00, -4.67) \end{aligned}$$

3. Attempt 3:
   Centroid 1 at (7.00, 7.00)

   $$\begin{aligned} \textbf{Assign}: &\quad (8.00, 8.00), (6.00, 6.00), (7.00, 7.00) \\ \textbf{Update}: &\quad (\frac{8.00 + 6.00 + 7.00}{3}, \frac{8.00 + 6.00 + 7.00}{3}) = (7.00, 7.00) \end{aligned}$$

Centroid 2 at (2.00, 2.33)

$$\textbf{Assign}: \quad (1.00, 2.00), (3.00, 3.00), (2.00, 2.00)$$
$$\textbf{Update}: \quad (\frac{1.00 + 3.00 + 2.00}{3}, \frac{2.00 + 3.00 + 2.00}{3}) = (2.00, 2.33)$$

Centroid 3 at (-4.00, -4.67)

$$\textbf{Assign}: \quad (-3.00, -3.00), (-2.00, -4.00), (-7.00, -7.00)$$
$$\textbf{Update}: \quad (\frac{-3.00 + -2.00 + -7.00}{3}, \frac{-3.00 + -4.00 + -7.00}{3}) = (-4.00, -4.67)$$

After 3rd attempt the centroids do not change
$\therefore$ The centroids are (7, 7), (2, 2.33), (-4, -4.67) respectively.

T6.

The centroids changed to (-2.5, -3.5), (4.5, 4.67), (-7, -7)

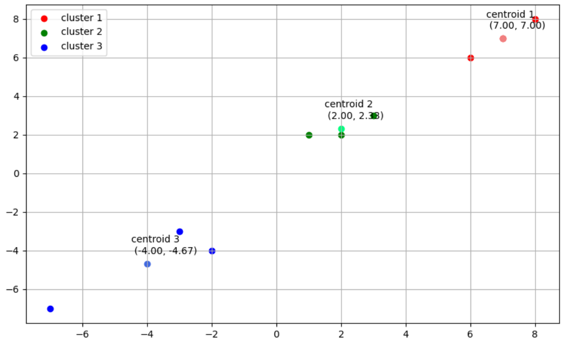


Figure 1: Starting points **T5**

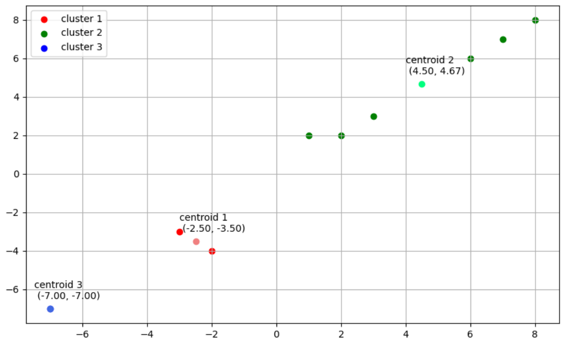


Figure 2: Starting points **T6**

The result of clustering changed, as seen in the figures.

T7.

The starting set from **T5.** is better. Looking at Figure 2, Cluster 2 has a high variance in data, but in Figure 1, Cluster 3 also has a high variance in its cluster, but not higher than in Figure 2. To make it clear, use a **fraction of explained variance** to measure 'goodness' of the starting point.

T5. starting points

$$\begin{aligned}\text{between-cluster variance} &= \frac{388.89}{9-1} = 48.61125\\ \text{all-data variance} &= \frac{418.22}{9-1} = 52.2775\\ \text{fraction of explained variance} &= \frac{388.89}{418.22} = 0.9298\end{aligned}$$

T6. starting points

$$\begin{aligned}\text{between-cluster variance} &= \frac{340.388}{9-1} = 42.548\\ \text{all-data variance} &= \frac{418.22}{9-1} = 52.2775\\ \text{fraction of explained variance} &= \frac{550.638}{628.472} = 0.8138\end{aligned}$$

the starting points from **T5.** has a higher explained variance than **T6.**

OT2.

$K = 4$. From the Figure 1. The cluster 3 has a high variance in its cluster. To reduce the variance by adding a new cluster. For instance, set a staring points to (3, 3), (2, 2), (-3, -3), (-7, -7)

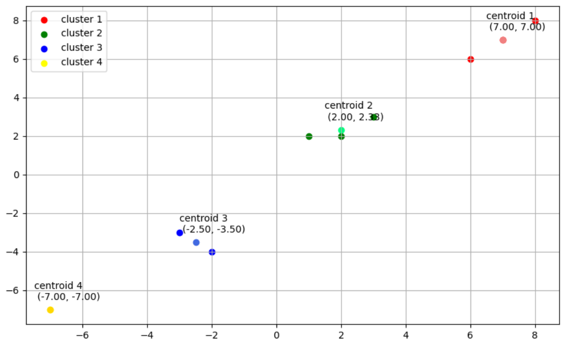


Figure 3: K=4

fraction of explained variance $= \frac{410.55}{418.22} = 0.9816$

My heart will go on [Code below]

T8.

median of Age = 28

T9.

mode of Embarked = S

```
embark_mode = train["Embarked"].mode()[0]
train["Embarked"] = train["Embarked"].fillna(embark_mode)
train.loc[train["Embarked"] == "S", "Embarked"] = 0
train.loc[train["Embarked"] == "C", "Embarked"] = 1
train.loc[train["Embarked"] == "Q", "Embarked"] = 2
```

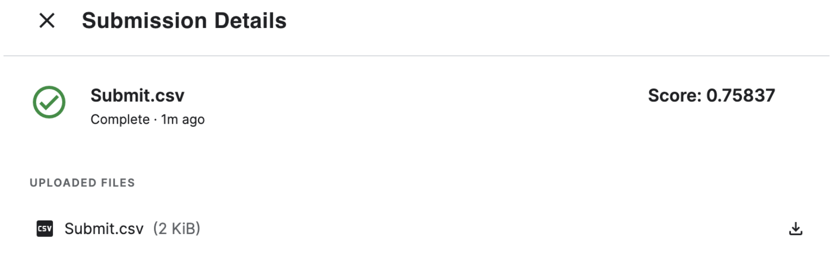
mode of Sex = male

```
sex_mode = train["Sex"].mode()[0]
train["Sex"] = train["Sex"].fillna(sex_mode)
train.loc[train["Sex"] == "male", "Sex"] = 0
train.loc[train["Sex"] == "female", "Sex"] = 1
```

T10.

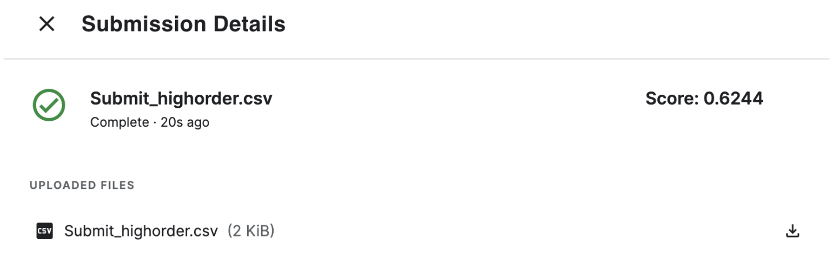
```python
class LogisticRegressionGradient:
    def __init__(self, lr=0.00001, random_state=42, epochs=10_000, threshold=0.5):
        self.lr = lr
        self.random_state = random_state
        self.epochs = epochs
        self.threshold = threshold

    @staticmethod
    def logist(X: np.array):
        X = np.clip(X, -600, 600) # for overflow
        mask = X >= 0
        X[mask] = np.exp(X[mask]) / (1 + np.exp(X[mask]))
        X[~mask] = 1 / (1 + np.exp(-X[~mask]))
        return X

    def fit(self, X: npt.ArrayLike, y: npt.ArrayLike):
        X = np.array(X)
        y = np.array(y)

        np.random.seed(self.random_state)
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias

        self.params = np.random.randn(X.shape[1])

        for _ in range(self.epochs):
            y_pred = self.logist(X @ self.params)
            diff = y - y_pred
            loss = X.T @ diff

            self.params += self.lr * loss

        return self

    def predict(self, X: npt.ArrayLike):
        X = np.array(X)
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias
        return (self.logist(X @ self.params) >= self.threshold).astype(int)
```

T11.



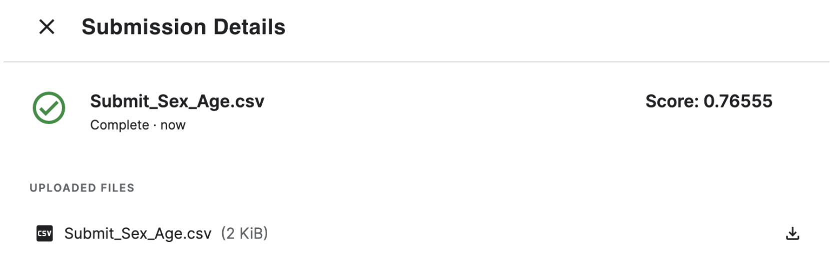
T12.

Adding high order features Age^2 and Age^3
Accuracy on Training set $= 0.79685$
Accuracy on Training set after add high order features $= 0.6274$
Accuracy on Test set after add high order features:



T13.

Accuracy on Training set after use only Sex and Age $= 0.78675$
Accuracy on Test set after use only Sex and Age:



The accuracy of the test set slightly increased.

OT3.

Apply min-max normalize to "Age" feature to prevent overflow

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Linear Regression using Gradient Descent method

```python
class LinearRegressionGradient:
    def __init__(self, lr=0.001, random_state=42, epochs=200_000):
        self.lr = lr
        self.random_state = random_state
        self.epochs = epochs
        self.params = None

    def fit(self, X: npt.ArrayLike, y: npt.ArrayLike):
        np.random.seed(self.random_state)
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias
        self.params = np.random.randn(X.shape[1])

        for _ in range(self.epochs):
            y_pred = X @ self.params
            diff = y - y_pred
            loss = X.T @ diff

            self.params += self.lr / X.shape[0] * loss

        return self

    def predict(self, X: npt.ArrayLike):
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias
        return X @ self.params
```

OT4.

Linear Regression using Matrix Inversion

```python
class LinearRegressionInversion:
    def __init__(self):
        self.params = None


    def fit(self, X: npt.ArrayLike, y: npt.ArrayLike):
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias

        self.params = np.linalg.inv(X.T @ X) @ X.T @ y
        return self

    def predict(self, X: npt.ArrayLike):
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias

        return X @ self.params
```

Weight

Weight	Bias	PClass	Sex	Age	Embarked
Gradient	0.74253777	-0.18302629	0.4945769	-0.35394677	0.04905888
Matrix Inv.	0.77442159	-0.18843944	0.49086711	-0.40222591	0.04911346

Mean squared errors = 0.003390521142094702

[Optional] Fun with matrix algebra

OT5.

$$\nabla_A trAB = B^T$$

Proof.

$$\begin{aligned} tr(AB) &= \sum_{ij} A_{ij} B_{ji} \\ (\nabla_A tr(AB))_{mn} &= \sum_{ij} \frac{\partial}{\partial A_{mn}} A_{ij} B_{ji} \\ &= \sum_{ij} B_{ji} \delta_{im} \delta_{jn} \\ &= B_{nm} \end{aligned}$$

$$\therefore \nabla_A tr(AB) = B^T$$

□

OT6.

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

Proof.

$$\begin{aligned} A^T &= \begin{bmatrix} A_{11} & \cdots & A_{n1} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{nm} \end{bmatrix} \\ \nabla_{A^T} f(A) &= \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \cdots & \frac{\partial f(A)}{\partial A_{n1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \cdots & \frac{\partial f(A)}{\partial A_{nm}} \end{bmatrix} \\ &= (\nabla_A f(A))^T \end{aligned}$$

□

OT7.

$$\nabla_A tr(ABA^TC) = CAB + C^TAB^T$$

1. **Showing arbitrary index of Matrix**

   Proof.

$$\begin{aligned}
\left(\nabla_A tr(ABA^TC)\right)_{mn} &= \sum_{ijkl} \frac{\partial}{\partial A_{mn}} A_{ij}B_{jk}A^T_{kl}C_{li} \\
&= \sum_{ijkl} \frac{\partial}{\partial A_{mn}} A_{ij}B_{jk}A_{lk}C_{li} \\
&= \sum_{ijkl} A_{ij}B_{jk}C_{li}\delta_{lm}\delta_{kn} + \sum_{ijkl} B_{jk}A_{lk}C_{li}\delta_{im}\delta_{jn} \\
&= \sum_{ij} A_{ij}B_{jn}C_{mi} + \sum_{kl} B_{nk}A_{lk}C_{lm} \\
&= \sum_{ij} C_{mi}A_{ij}B_{jn} + \sum_{kl} C^T_{ml}A_{lk}B^T_{kn} \\
&= (CAB)_{mn} + (C^TAB^T)_{mn}
\end{aligned}$$

$$\therefore \nabla_A tr(ABA^TC) = CAB + C^TAB^T$$

□

2. **Derivative Matrix**

   Proof. Let

$$F(A) = AB, G(A) = A^TC$$

Therefore

$$\begin{aligned}
\partial tr(ABA^TC) &= \partial tr(FG) \\
&= tr\left(\partial FG\right) && (\partial(tr(\mathbf{X})) = tr(\partial\mathbf{X})) \\
&= tr\left((\partial F)G\right) + tr(F(\partial G)) && (\partial(\mathbf{XY}) = (\partial\mathbf{X})\mathbf{Y} + \mathbf{X}(\partial\mathbf{Y})) \\
&= tr\left(\left((\partial A)B + \mathbf{0}\right)G\right) + tr\left(F\left((\partial A^T)C + \mathbf{0}\right)\right) && (B, C \text{ are constant}) \\
&= tr\left((\partial A)BG\right) + tr\left(F(\partial A^T)C\right) \\
&= tr\left(BG(\partial A)\right) + tr((\partial A^T)CF) && (tr(\mathbf{ABC}) = tr(\mathbf{BCA}) = tr(\mathbf{CAB})) \\
&= tr\left(BG(\partial A)\right) + tr(F^TC^T(\partial A)) && (tr(\mathbf{A}) = tr(\mathbf{A}^T)) \\
&= tr\left((BG + F^TC^T)\partial A\right) \\
&= tr\left((BA^TC + B^TA^TC^T)\partial A\right)
\end{aligned}$$

From

$$\partial f = tr\left(\left(\frac{\partial f}{\partial X}\right)^T \partial X\right)$$

$$\therefore \nabla_A tr(ABA^TC) = (BA^TC + B^TA^TC^T)^T = C^TAB^T + CAB$$

□

Code for My heart will go on

[1]:
```python
import numpy as np
import numpy.typing as npt
import pandas as pd
```

[2]:
```python
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
```

[3]:
```python
train.describe()
```

[3]:
```
       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
```

T8

[4]:
```python
print('median of age is', age_med := train['Age'].median())
```

```
median of age is 28.0
```

[5]:
```python
train['Age'] = train['Age'].fillna(age_med)
```

T9

[6]:
```python
print('Embarked Mode is', embark_mode := train['Embarked'].mode()[0])
```

```
Embarked Mode is S
```

[7]:
```python
train['Embarked'] = train['Embarked'].fillna(embark_mode)
train.loc[train["Embarked"] == "S", "Embarked"] = 0
train.loc[train["Embarked"] == "C", "Embarked"] = 1
train.loc[train["Embarked"] == "Q", "Embarked"] = 2
```

[8]:
```python
print('Sex Mode is', sex_mode := train['Sex'].mode()[0])
```

```
Sex Mode is male
```

[9]:
```python
train['Sex'] = train['Sex'].fillna(sex_mode)
train.loc[train["Sex"] == "male", "Sex"] = 0
```

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

T10, T11

[10]:
```
class LogisticRegressionGradient:
    def __init__(self, lr=0.00001, random_state=42, epochs=10_000, threshold=0.5):
        self.lr = lr
        self.random_state = random_state
        self.epochs = epochs
        self.threshold = threshold

    @staticmethod
    def logist(X: np.array):
        X = np.clip(X, -600, 600) # for overflow
        mask = X >= 0
        X[mask] = np.exp(X[mask]) / (1 + np.exp(X[mask]))
        X[~mask] = 1 / (1 + np.exp(-X[~mask]))
        return X

    def fit(self, X: npt.ArrayLike, y: npt.ArrayLike):
        X = np.array(X)
        y = np.array(y)

        np.random.seed(self.random_state)
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias

        self.params = np.random.randn(X.shape[1])

        for _ in range(self.epochs):
            y_pred = self.logist(X @ self.params)
            diff = y - y_pred
            loss = X.T @ diff

            self.params += self.lr * loss

        return self

    def predict(self, X: npt.ArrayLike):
        X = np.array(X)
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias
        return (self.logist(X @ self.params) >= self.threshold).astype(int)
```

[11]:
```
X = np.array(train[["Pclass","Sex","Age","Embarked"]].values, dtype = np.float64)
y = np.array(train['Survived'], dtype=np.float64)
```

[12]:
```
lr = LogisticRegressionGradient()
lr.fit(X, y)
```

[12]:
```
<__main__.LogisticRegressionGradient at 0x11a962b00>
```

[13]:
```
test['Age'] = test['Age'].fillna(test['Age'].median())

test['Embarked'] = test['Embarked'].fillna(test['Embarked'].mode()[0])
test.loc[test["Embarked"] == "S", "Embarked"] = 0
test.loc[test["Embarked"] == "C", "Embarked"] = 1
test.loc[test["Embarked"] == "Q", "Embarked"] = 2
```

```
test['Sex'] = test['Sex'].fillna(test['Sex'].mode()[0])
test.loc[test["Sex"] == "male", "Sex"] = 0
test.loc[test["Sex"] == "female", "Sex"] = 1
```

[14]:
```
y_pred = lr.predict(np.array(test[["Pclass","Sex","Age","Embarked"]], dtype=float))

pd.DataFrame({
    'PassengerId': test['PassengerId'],
    'Survived': y_pred
}).to_csv('Submit.csv', index=False)
```

T12

[15]:
```
def accuracy_score(y_test, y_pred):
    if y_test.shape[0] != y_pred.shape[0]:
        raise ValueError("Shape are not equal")
    return (y_test == y_pred).sum() / y_test.shape[0]
```

[16]:
```
y_pred = lr.predict(X)
print('Accuracy score of training set is', accuracy_score(y, y_pred))
```

```
Accuracy score of training set is 0.7968574635241302
```

Add high order feature $(x_1, x_1^2, x_2 \dots)$

[17]:
```
train['Age_squared'] = train['Age'] ** 2
test['Age_squared'] = test['Age'] ** 2
train['Age_Cubic'] = train['Age'] ** 3
test['Age_Cubic'] = test['Age'] ** 3

X_ho_train = np.array(train[["Pclass","Sex","Age", "Age_squared", "Age_Cubic",␣
↪"Embarked"]].values, dtype = np.float64)
X_ho_test = np.array(test[["Pclass","Sex","Age", "Age_squared", "Age_Cubic",␣
↪"Embarked"]].values, dtype = np.float64)

lr_ho = LogisticRegressionGradient().fit(X_ho_train, y)
y_pred_ho_train = lr_ho.predict(X_ho_train)

print(lr_ho.params)

print('Accuracy score of training set with high order feature is',␣
↪accuracy_score(y, y_pred_ho_train))
```

```
[  0.70055359 -12.32966498  11.8377913    9.44359665 318.83403664
 -87.10590108   4.65074534]
Accuracy score of training set with high order feature is 0.6273849607182941
```

[18]:
```
y_pred_ho_test = lr_ho.predict(X_ho_test)
pd.DataFrame({
    'PassengerId': test['PassengerId'],
    'Survived': y_pred_ho_test
}).to_csv('Submit_highorder.csv', index=False)
```

T13

[19]:
```
X_train = np.array(train[["Sex", "Age"]].values, dtype = np.float64)
X_test = np.array(test[["Sex", "Age"]].values, dtype = np.float64)

lr_sa = LogisticRegressionGradient().fit(X_train, y)
y_pred_sa_train = lr_sa.predict(X_train)

print(lr_sa.params)
print('Accuracy score of training set with only Sex and Age is', accuracy_score(y,␣
↪y_pred_sa_train))
```

```
[-1.01863706  2.34645073 -0.01149691]
Accuracy score of training set with only Sex and Age is 0.7867564534231201
```

[20]:
```
y_pred_sa = lr_sa.predict(X_test)
pd.DataFrame({
    'PassengerId': test['PassengerId'],
    'Survived': y_pred_sa
}).to_csv('Submit_Sex_Age.csv', index=False)
```

OT3

[21]:
```
print(X)
```

```
[[ 3.  0. 22.  0.]
 [ 1.  1. 38.  1.]
 [ 3.  1. 26.  0.]
 ...
 [ 3.  1. 28.  0.]
 [ 1.  0. 26.  1.]
 [ 3.  0. 32.  2.]]
```

normalized Age

[22]:
```
mx_age, mn_age = X[:, 2].max(), X[:, 2].min()

def normalize_age(x, mx_age, mn_age):
    return (x - mn_age) / (mx_age - mn_age)

normalize_age_vectorized = np.vectorize(lambda x : normalize_age(x, mx_age, mn_age))
X[:, 2] = normalize_age_vectorized(X[:, 2])
print(X)
```

```
[[3.         0.         0.27117366 0.        ]
 [1.         1.         0.4722292  1.        ]
 [3.         1.         0.32143755 0.        ]
 ...
 [3.         1.         0.34656949 0.        ]
 [1.         0.         0.32143755 1.        ]
 [3.         0.         0.39683338 2.        ]]
```

[23]:
```
class LinearRegressionGradient:
    def __init__(self, lr=0.001, random_state=42, epochs=200_000):
        self.lr = lr
        self.random_state = random_state
        self.epochs = epochs
        self.params = None

    def fit(self, X: npt.ArrayLike, y: npt.ArrayLike):
```

```python
        np.random.seed(self.random_state)
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias
        self.params = np.random.randn(X.shape[1])

        for _ in range(self.epochs):
            y_pred = X @ self.params
            diff = y - y_pred
            loss = X.T @ diff

            self.params += self.lr / X.shape[0] * loss

        return self

    def predict(self, X: npt.ArrayLike):
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias
        return X @ self.params
```

[24]:
```python
params_gradient = LinearRegressionGradient(random_state=0).fit(X, y).params
params_gradient
```

[24]: array([0.74253777, -0.18302629, 0.4945769 , -0.35394677, 0.04905888])

OT4

[25]:
```python
class LinearRegressionInversion:
    def __init__(self):
        self.params = None


    def fit(self, X: npt.ArrayLike, y: npt.ArrayLike):
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias

        self.params = np.linalg.inv(X.T @ X) @ X.T @ y
        return self

    def predict(self, X: npt.ArrayLike):
        X = np.hstack((np.ones(X.shape[0]).reshape(-1, 1), X))  # add bias

        return X @ self.params
```

[26]:
```python
params_matrix_inversion =  LinearRegressionInversion().fit(X, y).params
params_matrix_inversion
```

[26]: array([0.77442159, -0.18843944, 0.49086711, -0.40222591, 0.04911346])

Compute MSE

[27]:
```python
np.power(params_gradient - params_matrix_inversion, 2).sum()
```

[27]: 0.003390521142094702