Code for My heart will go on

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

[2]:
```
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]:
```
train.describe()
```

[3]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

T8

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

```
median of age is 28.0
```

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

T9

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

```
Embarked Mode is S
```

[7]:
```
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]:
```
print('Sex Mode is', sex_mode := train['Sex'].mode()[0])
```

```
Sex Mode is male
```

[9]:
```
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 0x1287b3c40>
```

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

test['Embarked'] = test['Embarked'].fillna(embark_mode)
```

```
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(sex_mode)
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 $(x_1, x_1^2, x_2 \ldots)$

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

[27]:
```
    0.003390521142094702
```