

my-heart-will-go-on

January 14, 2026

1 Homework 1 | Regression Part

1.1 Import Libraries

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

1.2 Loading Data

```
[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.head()
```

```
[3]: PassengerId  Survived  Pclass  \
0             1         0         3
1             2         1         1
2             3         1         3
3             4         1         1
4             5         0         3
```

```

                                Name      Sex  Age  SibSp  \
0                Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                Heikkinen, Miss. Laina    female  26.0      0
3  Futrelle, Mrs. Jacques Heath (Lily May Peel)    female  35.0      1
4                Allen, Mr. William Henry    male  35.0      0
```

```

Parch      Ticket      Fare Cabin Embarked
0      0    A/5 21171   7.2500   NaN        S
1      0    PC 17599  71.2833   C85        C
```

2	0	STON/O2.	3101282	7.9250	NaN	S
3	0		113803	53.1000	C123	S
4	0		373450	8.0500	NaN	S

```
[4]: test.head()
```

```
[4]:
```

	PassengerId	Pclass	Name	Sex	\
0	892	3	Kelly, Mr. James	male	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	
2	894	2	Myles, Mr. Thomas Francis	male	
3	895	3	Wirz, Mr. Albert	male	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	

	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	34.5	0	0	330911	7.8292	NaN	Q
1	47.0	1	0	363272	7.0000	NaN	S
2	62.0	0	0	240276	9.6875	NaN	Q
3	27.0	0	0	315154	8.6625	NaN	S
4	22.0	1	1	3101298	12.2875	NaN	S

1.3 T8.

```
[5]: # print the median age from the training data
median_age = train["Age"].median()
print("The median age of the training set is:", median_age)

# fill the missing values in the "Age" column with the median age
train["Age"] = train["Age"].fillna(median_age)
```

The median age of the training set is: 28.0

1.4 T9.

```
[6]: # print the mode of the "Embarked" column
mode_embarked = train["Embarked"].mode()[0]
print("The mode of the 'Embarked' column is:", mode_embarked)

# fill the missing values in the "Embarked" column with the mode
train["Embarked"] = train["Embarked"].fillna(mode_embarked)
```

The mode of the 'Embarked' column is: S

```
[7]: # get the unique values in the "Embarked" column
unique_embarked = train["Embarked"].unique()
print("The unique values in the 'Embarked' column are:", unique_embarked)
```

```

# map the "Embarked" column to numerical values
train["Embarked"].infer_objects(copy=False)
pd.set_option('future.no_silent_downcasting', True)

embarked_mapping = {}

for i, port in enumerate(unique_embarked):
    train["Embarked"] = train["Embarked"].replace(port, i)

    embarked_mapping[port] = i
    print(f"Mapping '{port}' to {i}")

# print the first 5 entries of the "Embarked" column after mapping
print("The 'Embarked' column after mapping to numerical values:")
print(train["Embarked"].head())

```

The unique values in the 'Embarked' column are: ['S' 'C' 'Q']
Mapping 'S' to 0
Mapping 'C' to 1
Mapping 'Q' to 2
The 'Embarked' column after mapping to numerical values:

0	0
1	1
2	0
3	0
4	0

Name: Embarked, dtype: object

1.4.1 Applied the same code to the “Sex” column

```

[8]: # print the mode of the "Sex" column
mode_sex = train["Sex"].mode()[0]
print("The mode of the 'Sex' column is:", mode_sex)

# fill the missing values in the "Sex" column with the mode
train["Sex"] = train["Sex"].fillna(mode_sex)

```

The mode of the 'Sex' column is: male

```

[9]: # get the unique values in the "Sex" column
unique_sex = train["Sex"].unique()
print("The unique values in the 'Sex' column are:", unique_sex)

# map the "Sex" column to numerical values
train["Sex"].infer_objects(copy=False)
pd.set_option('future.no_silent_downcasting', True)

```

```
sex_mapping = {}

for i, sex in enumerate(unique_sex):
    train["Sex"] = train["Sex"].replace(sex, i)

    sex_mapping[sex] = i
    print(f"Mapping '{sex}' to {i}")

# print the first 5 entries of the "Sex" column after mapping
print("The 'Sex' column after mapping to numerical values:")
print(train["Sex"].head())
```

The unique values in the 'Sex' column are: ['male' 'female']

Mapping 'male' to 0

Mapping 'female' to 1

The 'Sex' column after mapping to numerical values:

```
0    0
1    1
2    1
3    1
4    0
```

Name: Sex, dtype: object

1.5 T10.

```
[10]: train.head()
```

```
[10]:   PassengerId  Survived  Pclass  \
0             1         0        3
1             2         1        1
2             3         1        3
3             4         1        1
4             5         0        3
```

```

                                Name Sex   Age  SibSp  Parch  \
0                        Braund, Mr. Owen Harris    0  22.0    1    0
1  Cumings, Mrs. John Bradley (Florence Briggs Th...    1  38.0    1    0
2                        Heikkinen, Miss. Laina    1  26.0    0    0
3  Futrelle, Mrs. Jacques Heath (Lily May Peel)    1  35.0    1    0
4                        Allen, Mr. William Henry    0  35.0    0    0
```

```

      Ticket      Fare Cabin Embarked
0     A/5 21171   7.2500   NaN        0
1      PC 17599  71.2833   C85        1
2  STON/O2. 3101282   7.9250   NaN        0
3      113803  53.1000  C123        0
4      373450   8.0500   NaN        0
```

```
[11]: # define X and y
X = np.array(train[["Pclass", "Sex", "Age", "Embarked"]].values, dtype=np.
    ↪float32)
y = np.array(train["Survived"].values, dtype=np.float32)
print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (891, 4)
y shape: (891,)
```

```
[12]: # define predict function
def predict(X, theta):
    linear_combination = np.dot(X, theta)
    return linear_combination
```

```
[13]: # reset theta and use a smaller learning rate
np.random.seed(1)
theta = np.random.rand(X.shape[1])
print("Initial theta:", theta)

# use a much smaller learning rate
smaller_learning_rate = 0.001
num_iterations = 1000

print(f"\nUsing learning rate: {smaller_learning_rate}")
print("Running gradient descent...")

# run the corrected gradient descent with smaller learning rate
for iteration in range(num_iterations):
    # compute predictions
    predictions = predict(X, theta)

    # compute errors
    errors = predictions - y

    # compute gradients
    m = X.shape[0]
    gradient_theta = (1 / m) * np.dot(X.T, errors)

    # update parameters (corrected: subtract gradient to minimize loss)
    theta -= smaller_learning_rate * gradient_theta

    mse = np.mean(errors**2)

    # print every 10 iterations to avoid too much output
    if (iteration + 1) % 100 == 0:
```

```

        print(f"Iteration {iteration + 1}: MSE = {mse:.6f}, theta =_
        ↪{[float(round(theta_i, 6)) for theta_i in theta]}")

print(f"\nFinal theta: {[float(round(theta_i, 6)) for theta_i in theta]}")
print(f"Final MSE: {mse:.6f}")

```

Initial theta: [4.17022005e-01 7.20324493e-01 1.14374817e-04 3.02332573e-01]

Using learning rate: 0.001

Running gradient descent...

```

Iteration 100: MSE = 0.484684, theta = [0.330444, 0.710066, -0.019727, 0.283687]
Iteration 200: MSE = 0.382123, theta = [0.261571, 0.701534, -0.015193, 0.267648]
Iteration 300: MSE = 0.314127, theta = [0.205788, 0.694219, -0.011506, 0.253558]
Iteration 400: MSE = 0.268967, theta = [0.160632, 0.687908, -0.008506, 0.241096]
Iteration 500: MSE = 0.238900, theta = [0.124102, 0.682425, -0.006066, 0.229998]
Iteration 600: MSE = 0.218814, theta = [0.094574, 0.677628, -0.00408, 0.220048]
Iteration 700: MSE = 0.205333, theta = [0.070729, 0.6734, -0.002463, 0.211068]
Iteration 800: MSE = 0.196231, theta = [0.051494, 0.669646, -0.001146, 0.202914]
Iteration 900: MSE = 0.190033, theta = [0.035998, 0.666289, -7.3e-05, 0.195464]
Iteration 1000: MSE = 0.185767, theta = [0.023536, 0.663266, 0.000802, 0.188622]

```

Final theta: [0.023536, 0.663266, 0.000802, 0.188622]

Final MSE: 0.185767

1.6 T11.

```

[14]: # use the final theta to make predictions
final_predictions = predict(X, theta)
final_predictions_binary = (final_predictions >= 0.5).astype(int)
accuracy = np.mean(final_predictions_binary == y)
print(f"Training accuracy: {accuracy:.4f}")

```

Training accuracy: 0.7856

```

[15]: # impute missing values in the test set like we did for the training set
median_age_test = test["Age"].median()
test["Age"] = test["Age"].fillna(median_age_test)

mode_embarked_test = test["Embarked"].mode()[0]
test["Embarked"] = test["Embarked"].fillna(mode_embarked_test)

mode_sex_test = test["Sex"].mode()[0]
test["Sex"] = test["Sex"].fillna(mode_sex_test)

# map the "Embarked" and "Sex" column in the test set to numerical values using_
↪the same mappings
test["Embarked"].infer_objects(copy=False)

```

```
test["Sex"].infer_objects(copy=False)
pd.set_option('future.no_silent_downcasting', True)

for port, i in embarked_mapping.items():
    test["Embarked"] = test["Embarked"].replace(port, i)

for sex, i in sex_mapping.items():
    test["Sex"] = test["Sex"].replace(sex, i)
```

```
[16]: # find the matrices of final predictions on testing set
X_test = np.array(test[["Pclass", "Sex", "Age", "Embarked"]].values, dtype=np.
    ↪float32)
test_predictions = predict(X_test, theta)
test_predictions_binary = (test_predictions >= 0.5).astype(int)

print("Test set predictions (first 10):", test_predictions_binary[:10])
```

Test set predictions (first 10): [0 1 0 0 1 0 1 0 1 0]

```
[17]: # save the test predictions to a CSV file
output = pd.DataFrame({'PassengerId': test['PassengerId'], 'Survived':
    ↪test_predictions_binary})
output.to_csv('titanic_test_predictions.csv', index=False)
print("Test set predictions saved to 'titanic_test_predictions.csv'")
```

Test set predictions saved to 'titanic_test_predictions.csv'

1.7 T12.

```
[18]: # define new column, PE, the product of Pclass and Embarked
train_12 = train.copy()
train_12["PE"] = train_12["Pclass"] * train_12["Embarked"]

# define X and y
X = np.array(train_12[["Pclass", "Sex", "Age", "Embarked", "PE"]].values,
    ↪dtype=np.float32)
y = np.array(train_12["Survived"].values, dtype=np.float32)
print("X shape:", X.shape)
print("y shape:", y.shape)
```

X shape: (891, 5)

y shape: (891,)

```
[19]: # reset theta and use a smaller learning rate
np.random.seed(1)
theta = np.random.rand(X.shape[1])
```

```

print("Initial theta:", theta)

# use a much smaller learning rate
smaller_learning_rate = 0.001
num_iterations = 1000

print(f"\nUsing learning rate: {smaller_learning_rate}")
print("Running gradient descent...")

# run the corrected gradient descent with smaller learning rate
for iteration in range(num_iterations):
    # compute predictions
    predictions = predict(X, theta)

    # compute errors
    errors = predictions - y

    # compute gradients
    m = X.shape[0]
    gradient_theta = (1 / m) * np.dot(X.T, errors)

    # update parameters (corrected: subtract gradient to minimize loss)
    theta -= smaller_learning_rate * gradient_theta

    mse = np.mean(errors**2)

    # print every 10 iterations to avoid too much output
    if (iteration + 1) % 100 == 0:
        print(f"Iteration {iteration + 1}: MSE = {mse:.6f}, theta = [
↪{[float(round(theta_i, 6)) for theta_i in theta]}",

print(f"\nFinal theta: {[float(round(theta_i, 6)) for theta_i in theta]}")
print(f"Final MSE: {mse:.6f}")

```

Initial theta: [4.17022005e-01 7.20324493e-01 1.14374817e-04 3.02332573e-01
1.46755891e-01]

Using learning rate: 0.001

Running gradient descent...

Iteration 100: MSE = 0.539801, theta = [0.323194, 0.708745, -0.020362, 0.273278,
0.05195]

Iteration 200: MSE = 0.362548, theta = [0.254492, 0.700147, -0.014448, 0.255667,
-0.007821]

Iteration 300: MSE = 0.275763, theta = [0.202474, 0.69343, -0.01018, 0.245263,
-0.045142]

Iteration 400: MSE = 0.231427, theta = [0.162538, 0.688035, -0.007059, 0.239472,
-0.06789]


```

Iteration 500: MSE = 0.207572, theta = [0.131467, 0.683579, -0.004745, 0.236594,
-0.081255]
Iteration 600: MSE = 0.193982, theta = [0.106993, 0.679801, -0.003004, 0.235522,
-0.088645]
Iteration 700: MSE = 0.185789, theta = [0.087499, 0.676521, -0.001675, 0.235539,
-0.092289]
Iteration 800: MSE = 0.180595, theta = [0.071818, 0.673613, -0.000646, 0.236185,
-0.093635]
Iteration 900: MSE = 0.177165, theta = [0.059095, 0.670991, 0.000161, 0.237167,
-0.093619]
Iteration 1000: MSE = 0.174827, theta = [0.048699, 0.66859, 0.000802, 0.238304,
-0.092839]

```

```

Final theta: [0.048699, 0.66859, 0.000802, 0.238304, -0.092839]
Final MSE: 0.174827

```

```

[20]: # use the final theta to make predictions
final_predictions = predict(X, theta)
final_predictions_binary = (final_predictions >= 0.5).astype(int)
accuracy = np.mean(final_predictions_binary == y)
print(f"Training accuracy: {accuracy:.4f}")

```

Training accuracy: 0.7868

```

[21]: # define new column, PE, the product of Pclass and Embarked
test_12 = test.copy()
test_12["PE"] = test_12["Pclass"] * test_12["Embarked"]

```

```

[22]: # find the matrices of final predictions on testing set
X_test_12 = np.array(test_12[["Pclass", "Sex", "Age", "Embarked", "PE"]].
    ↪ values, dtype=np.float32)
test_predictions = predict(X_test_12, theta)
test_predictions_binary = (test_predictions >= 0.5).astype(int)

print("Test set predictions (first 10):", test_predictions_binary[:10])

```

Test set predictions (first 10): [0 1 0 0 1 0 1 0 1 0]

```

[23]: # save the test predictions to a CSV file
output = pd.DataFrame({'PassengerId': test['PassengerId'], 'Survived':
    ↪ test_predictions_binary})
output.to_csv('titanic_test_predictions_ver_T12.csv', index=False)
print("Test set predictions saved to 'titanic_test_predictions_ver_T12.csv'")

```

Test set predictions saved to 'titanic_test_predictions_ver_T12.csv'

1.8 T13.

```
[24]: # define X and y
X = np.array(train[["Sex", "Age"]].values, dtype=np.float32) # only Sex and Age
y = np.array(train["Survived"].values, dtype=np.float32)
print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (891, 2)
y shape: (891,)
```

```
[25]: # reset theta and use a smaller learning rate
np.random.seed(1)
theta = np.random.rand(X.shape[1])
print("Initial theta:", theta)

# use a much smaller learning rate
smaller_learning_rate = 0.001
num_iterations = 1000

print(f"\nUsing learning rate: {smaller_learning_rate}")
print("Running gradient descent...")

# run the corrected gradient descent with smaller learning rate
for iteration in range(num_iterations):
    # compute predictions
    predictions = predict(X, theta)

    # compute errors
    errors = predictions - y

    # compute gradients
    m = X.shape[0]
    gradient_theta = (1 / m) * np.dot(X.T, errors)

    # update parameters (corrected: subtract gradient to minimize loss)
    theta -= smaller_learning_rate * gradient_theta

    mse = np.mean(errors**2)

    # print every 10 iterations to avoid too much output
    if (iteration + 1) % 100 == 0:
        print(f"Iteration {iteration + 1}: MSE = {mse:.6f}, theta =_
↪{[float(round(theta_i, 6)) for theta_i in theta]}")

print(f"\nFinal theta: {[float(round(theta_i, 6)) for theta_i in theta]}")
print(f"Final MSE: {mse:.6f}")
```

Initial theta: [0.417022 0.72032449]

Using learning rate: 0.001

Running gradient descent...

```
Iteration 100: MSE = 0.182960, theta = [0.415327, 0.006565]
Iteration 200: MSE = 0.182463, theta = [0.420315, 0.006518]
Iteration 300: MSE = 0.181990, theta = [0.425175, 0.006471]
Iteration 400: MSE = 0.181541, theta = [0.429911, 0.006426]
Iteration 500: MSE = 0.181115, theta = [0.434526, 0.006382]
Iteration 600: MSE = 0.180710, theta = [0.439023, 0.006339]
Iteration 700: MSE = 0.180326, theta = [0.443406, 0.006297]
Iteration 800: MSE = 0.179961, theta = [0.447677, 0.006257]
Iteration 900: MSE = 0.179614, theta = [0.451838, 0.006217]
Iteration 1000: MSE = 0.179285, theta = [0.455894, 0.006178]
```

Final theta: [0.455894, 0.006178]

Final MSE: 0.179285

```
[26]: # use the final theta to make predictions
final_predictions = predict(X, theta)
final_predictions_binary = (final_predictions >= 0.5).astype(int)
accuracy = np.mean(final_predictions_binary == y)
print(f"Training accuracy: {accuracy:.4f}")
```

Training accuracy: 0.7733

```
[27]: # find the matrices of final predictions on testing set
X_test = np.array(test[["Sex", "Age"]].values, dtype=np.float32) # only Sex and
    ↪ Age
test_predictions = predict(X_test, theta)
test_predictions_binary = (test_predictions >= 0.5).astype(int)

print("Test set predictions (first 10):", test_predictions_binary[:10])
```

Test set predictions (first 10): [0 1 0 0 1 0 1 0 1 0]

```
[28]: # save the test predictions to a CSV file
output = pd.DataFrame({'PassengerId': test['PassengerId'], 'Survived':
    ↪ test_predictions_binary})
output.to_csv('titanic_test_predictions_ver_T13.csv', index=False)
print("Test set predictions saved to 'titanic_test_predictions_ver_T13.csv'")
```

Test set predictions saved to 'titanic_test_predictions_ver_T13.csv'