Katyayani
2201AI18
Assignment1 report

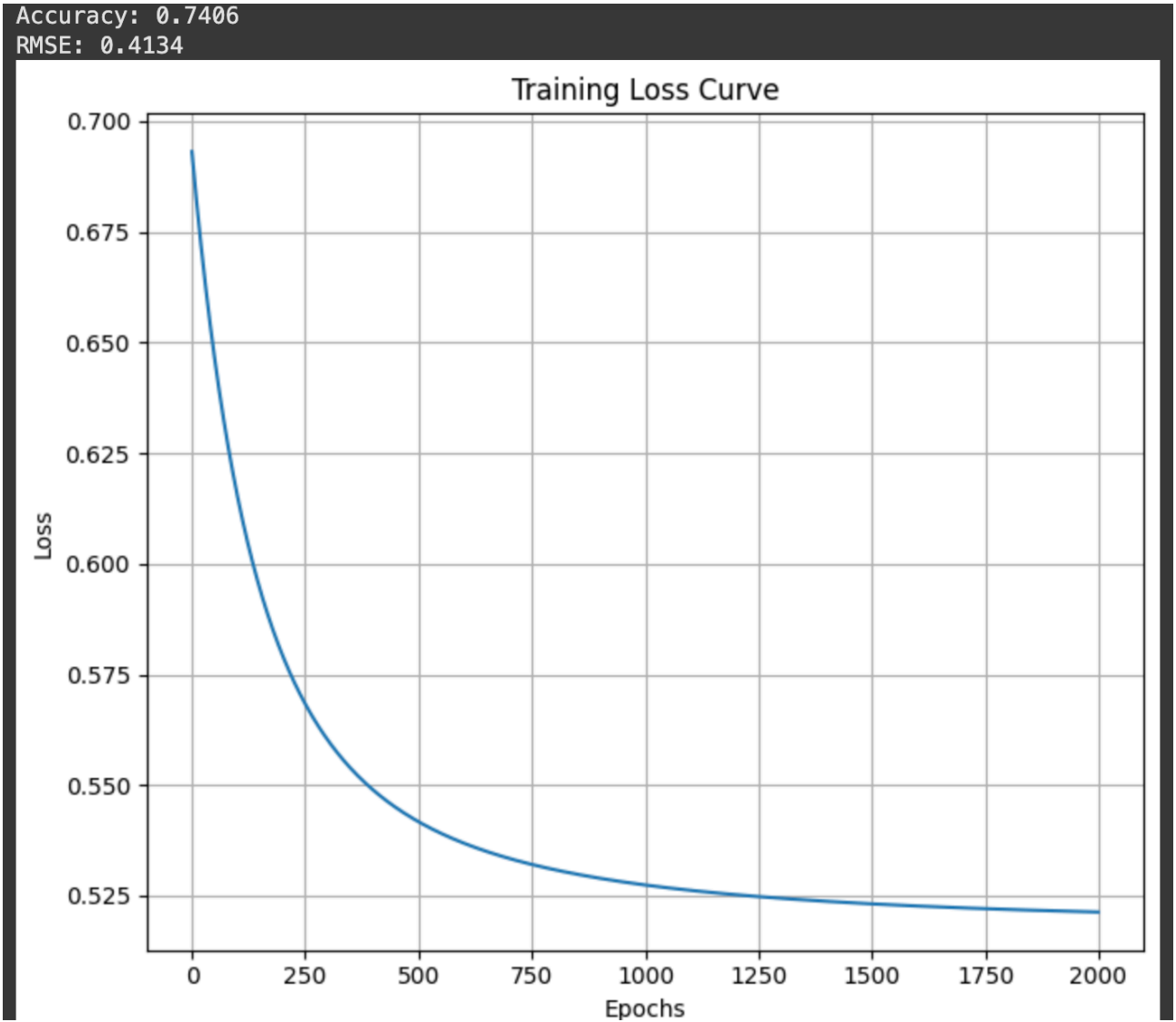
Wine Quality Classification using Logistic Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, mean_s
from math import sqrt

# Load dataset
df = pd.read_csv("winequality-red.csv", sep=";")

X = df.drop("quality", axis=1).values
y = (df["quality"] >= 6).astype(int).values.reshape(-1, 1)   # Good

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic Regression
class LogisticRegression:
    def __init__(self, lr=0.01, epochs=2000):
        self.lr = lr
        self.epochs = epochs

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros((n_features, 1))
        self.bias = 0
        self.losses = []
```

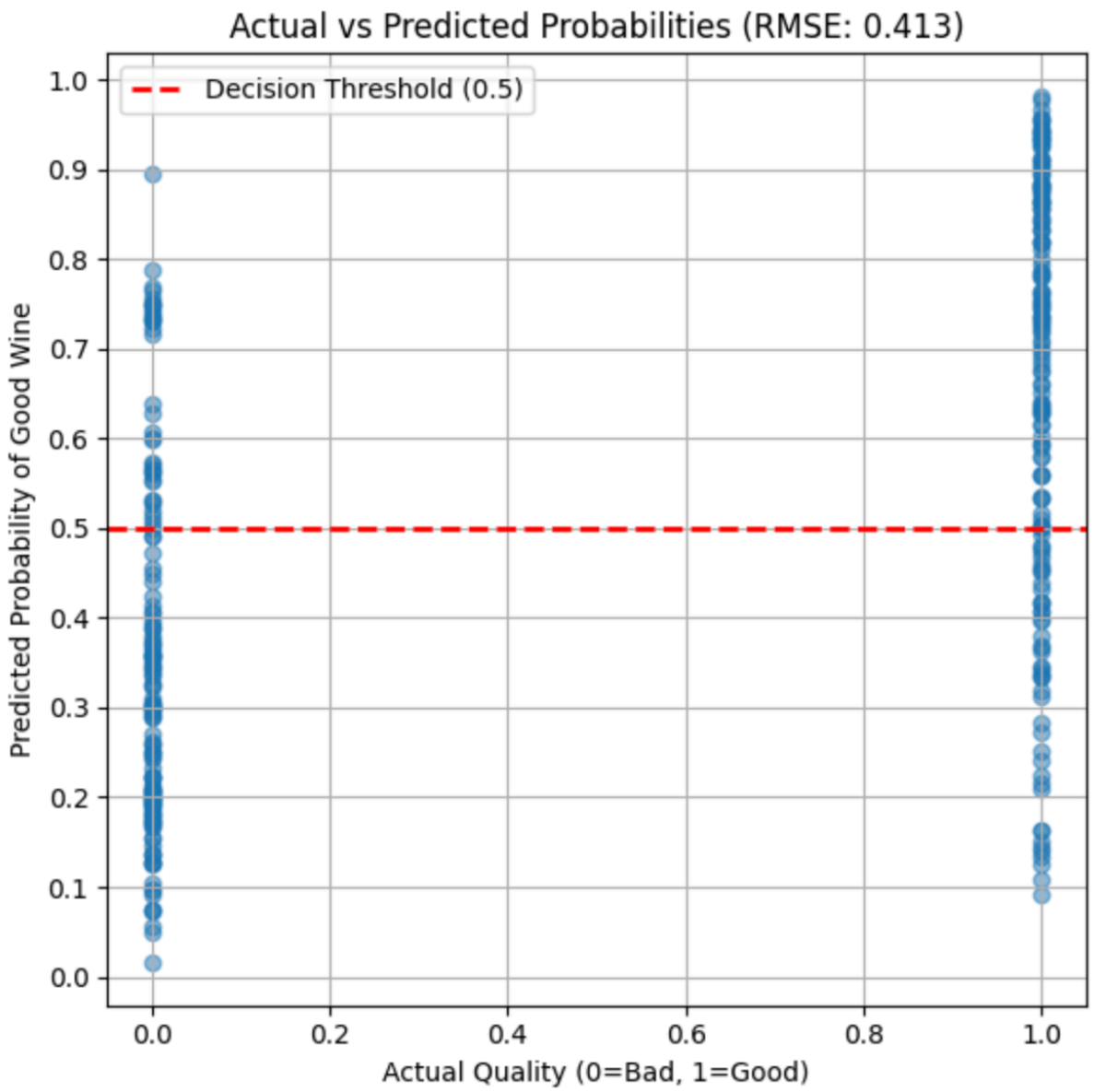
```
        for _ in range(self.epochs):
            # Linear model
            linear_output = np.dot(X, self.weights) + self.bias
            # Apply sigmoid
            y_pred = sigmoid(linear_output)

            # Compute loss (binary cross-entropy)
            loss = -(1/n_samples) * np.sum(
                y*np.log(y_pred+1e-9) + (1-y)*np.log(1-y_pred+1e-9)
            )
            self.losses.append(loss)

            # Gradients
            dw = (1/n_samples) * np.dot(X.T, (y_pred - y))
            db = (1/n_samples) * np.sum(y_pred - y)

            # Update weights
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_pred = sigmoid(linear_output)
        return (y_pred >= 0.5).astype(int)

    def predict_proba(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return sigmoid(linear_output)

# Train model
model = LogisticRegression(lr=0.01, epochs=2000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)

# Accuracy
accuracy = np.mean(y_pred == y_test)
print(f"Accuracy: {accuracy:.4f}")

# Calculate RMSE
rmse = sqrt(mean_squared_error(y_test, y_pred_proba))
print(f"RMSE: {rmse:.4f}")

# Plot loss curve
plt.figure(figsize=(7, 6))
plt.plot(model.losses)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss Curve")
plt.grid(True)
plt.tight_layout()
plt.savefig('loss_curve.png')
plt.show()

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```
# Plot the confusion matrix
plt.figure(figsize=(7, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Bad Wine (0)', 'Good Wine (1)'],
            yticklabels=['Bad Wine (0)', 'Good Wine (1)'])
plt.title('Confusion Matrix - Wine Quality Classification')
plt.ylabel('Actual Quality')
plt.xlabel('Predicted Quality')
plt.tight_layout()
plt.show()

# plot of actual vs predicted probabilities
plt.figure(figsize=(7, 6))
plt.scatter(y_test, y_pred_proba, alpha=0.5)
plt.axhline(y=0.5, color='r', linestyle='--', lw=2, label='Decision
plt.xlabel('Actual Quality (0=Bad, 1=Good)')
plt.ylabel('Predicted Probability of Good Wine')
plt.title('Actual vs Predicted Probabilities (RMSE: {:.3f})'.format
plt.yticks(np.arange(0, 1.1, 0.1))
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Accuracy: 0.7406
RMSE: 0.4134
```




Confusion Matrix - Wine Quality Classification
Actual Quality
Bad Wine (0)
Good Wine (1)
112
37
46
125
Bad Wine (0)
Good Wine (1)
Predicted Quality
120
110
100
90
80
70
60
50
40


Actual vs Predicted Probabilities (RMSE: 0.413)
Decision Threshold (0.5)
Predicted Probability of Good Wine
Actual Quality (0=Bad, 1=Good)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.2
0.4
0.6
0.8
1.0