1 Logistic Regression: Cost, Gradient and Hessian

Sigmoid Function

$$g(z) \;=\; \frac{1}{1+e^{-z}}.$$

Hypothesis Function $h_\theta(x)$

$$h_\theta(x) \;=\; g\big(\theta^T x\big) \;=\; \frac{1}{1+e^{-\theta^T x}}.$$

Given a training set $\{(x^{(i)},\, y^{(i)})\}_{i=1}^m$ with $y^{(i)} \in \{0,1\}$ and $x^{(i)} \in \mathbb{R}^n$, the *average empirical loss* (negative log-likelihood) is

Cost Function $J(\theta)$

$$J(\theta) \;=\; -\frac{1}{m}\sum_{i=1}^m \Big[y^{(i)}\,\log\!\big(h_\theta(x^{(i)})\big) \;+\; \big(1-y^{(i)}\big)\,\log\!\big(1-h_\theta(x^{(i)})\big)\Big].$$

1.1 Gradient of $J(\theta)$

First, write down the gradient component-wise. For each training example i, we have

Gradient Derivation (Part 1)

$$h_\theta(x^{(i)}) \;=\; g\big(\theta^T x^{(i)}\big), \qquad g'(z) \;=\; g(z)\,(1-g(z)).$$

Gradient: Chain-Rule Expansion

$$\nabla_\theta J(\theta) = -\frac{1}{m}\sum_{i=1}^m \Big[y^{(i)}\,\frac{1}{h_\theta(x^{(i)})}\,\nabla_\theta h_\theta(x^{(i)}) \;-\; \big(1-y^{(i)}\big)\,\frac{1}{1-h_\theta(x^{(i)})}\,\nabla_\theta h_\theta(x^{(i)})\Big].$$

Now, since

$$\nabla_\theta\, h_\theta(x^{(i)}) \;=\; g'\big(\theta^T x^{(i)}\big)\,x^{(i)} \;=\; h_\theta(x^{(i)})\,\big(1-h_\theta(x^{(i)})\big)\,x^{(i)},$$

substitute into the above:

Gradient: Simplification

$$\begin{aligned}
\nabla_\theta J(\theta) &= -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)}\,\frac{1}{h_\theta(x^{(i)})} \;-\; \big(1-y^{(i)}\big)\,\frac{1}{1-h_\theta(x^{(i)})}\Big]\Big[h_\theta(x^{(i)})\,\big(1-h_\theta(x^{(i)})\big)\,x^{(i)}\Big] \\
&= -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)}\,\big(1-h_\theta(x^{(i)})\big) \;-\; \big(1-y^{(i)}\big)\,h_\theta(x^{(i)})\Big]\,x^{(i)} \\
&= -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)} \;-\; y^{(i)}\,h_\theta(x^{(i)}) \;-\; h_\theta(x^{(i)}) \;+\; y^{(i)}\,h_\theta(x^{(i)})\Big]\,x^{(i)} \\
&= -\frac{1}{m}\sum_{i=1}^{m}\big[y^{(i)} \;-\; h_\theta(x^{(i)})\big]\,x^{(i)}.
\end{aligned}$$

Hence, in compact vector form:

Gradient of $J(\theta)$

$$\nabla_\theta J(\theta) \;=\; -\frac{1}{m}\sum_{i=1}^{m}\big(y^{(i)} - h_\theta(x^{(i)})\big)\,x^{(i)}.$$

1.2 Hessian of $J(\theta)$

Recall the Hessian is $H = \nabla_\theta^2 J(\theta)$. We can either differentiate the gradient again, or observe directly that

$$\nabla_\theta h_\theta(x^{(i)}) = h_\theta(x^{(i)})\,\big(1-h_\theta(x^{(i)})\big)\,x^{(i)}.$$

When we differentiate $\nabla_\theta J(\theta)$ another time, each term brings down another copy of $x^{(i)}$. Concretely:

Hessian of $J(\theta)$

$$H \;=\; \nabla_\theta^2 J(\theta) = \frac{1}{m}\sum_{i=1}^{m} h_\theta(x^{(i)})\,\big(1-h_\theta(x^{(i)})\big)\,x^{(i)}\,(x^{(i)})^T.$$

—

1.3 Proof that H is Positive Semi-Definite

For any $z \in \mathbb{R}^n$, consider

Positive Semi-Definiteness

$$
\begin{aligned}
z^T H z &= z^T \left[\tfrac{1}{m} \sum_{i=1}^{m} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))\, x^{(i)} (x^{(i)})^T \right] z \\
&= \frac{1}{m} \sum_{i=1}^{m} h_\theta(x^{(i)}) \left(1 - h_\theta(x^{(i)})\right) \left(z^T x^{(i)}\right) \left((x^{(i)})^T z\right) \\
&= \frac{1}{m} \sum_{i=1}^{m} h_\theta(x^{(i)}) \left(1 - h_\theta(x^{(i)})\right) \left(z^T x^{(i)}\right)^2 \ \geq \ 0,
\end{aligned}
$$

showing H is positive semi-definite.

ps1_b

June 6, 2025

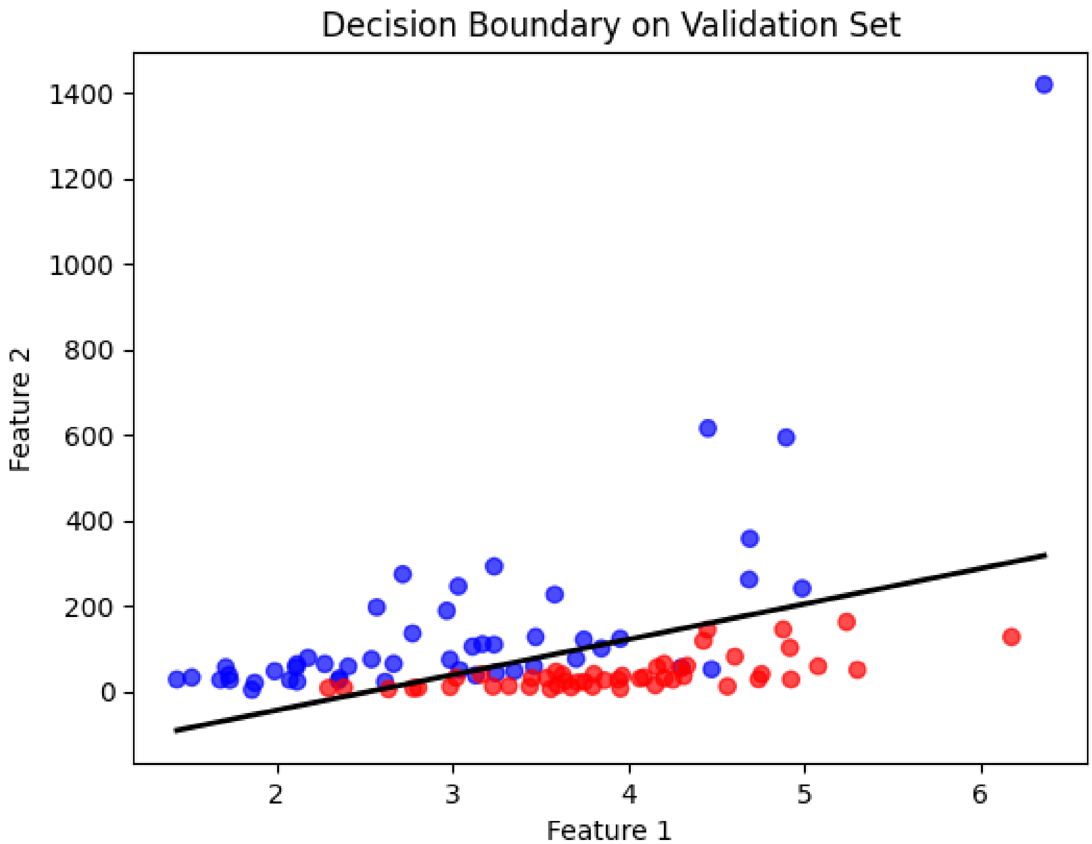
[3]:

```
import numpy as np
import util

from linear_model import LinearModel
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

class LogisticRegression(LinearModel):
    @staticmethod
    def sigmoid(z):
        return np.where(z >= 0, 1 / (1 + np.exp(-z)), np.exp(z) / (1 + np.
↪exp(z)))

    @staticmethod
    def compute_htheta(X, theta):
        return LogisticRegression.sigmoid(np.dot(X, theta))

    @staticmethod
    def compute_grad(X,y,theta):
        m = X.shape[0]
        h = LogisticRegression.compute_htheta(X, theta)        # shape (m,)
        error = y - h                              # shape (m,)
        grad = -(X.T @ error) / m
        return grad

    @staticmethod
    def compute_hessian(X,theta):
        hessian = np.zeros((X.shape[1], X.shape[1]))
        m = X.shape[0]
        h_theta = LogisticRegression.compute_htheta(X, theta)
        for i in range(m):
            hessian += h_theta[i] * (1 - h_theta[i]) * np.outer(X[i], X[i])
        hessian /= m
        return hessian

    def fit(self, x, y):
        # *** START CODE HERE ***
```

```python
        theta = np.zeros(x.shape[1])

        max_iter = 1000
        for _ in range(max_iter):
            gradient_matrix = LogisticRegression.compute_grad(x, y, theta)
            hessian_matrix = LogisticRegression.compute_hessian(x, theta)

            delta  = np.linalg.solve(hessian_matrix, gradient_matrix)
            if np.linalg.norm(delta) < 1e-6:
                break
            theta -= delta

        self.theta = theta.copy()


        # *** END CODE HERE ***

    def predict(self, x):
        """Make a prediction given new inputs x.

        Args:
            x: Inputs of shape (m, n).

        Returns:
            Outputs of shape (m,).
        """
        # *** START CODE HERE ***
        probs = LogisticRegression.compute_htheta(x, self.theta)
        return (probs >= 0.5).astype(int)
        # *** END CODE HERE ***

def main(train_path, eval_path, pred_path):
    """Problem 1(b): Logistic regression with Newton's Method.

    Args:
        train_path: Path to CSV file containing dataset for training.
        eval_path: Path to CSV file containing dataset for evaluation.
        pred_path: Path to save predictions.
    """
    x_train, y_train = util.load_dataset(train_path, add_intercept=True)
    x_eval,  y_eval  = util.load_dataset(eval_path, add_intercept=True)

    # *** START CODE HERE ***
    clf = LogisticRegression()
    clf.fit(x_train, y_train)

    y_pred = clf.predict(x_eval)
```

```python
    np.savetxt(pred_path, y_pred, fmt="%d")

    acc = np.mean(y_pred == y_eval)
    print(f"Eval accuracy = {acc:.4f}")

    print("Confusion Matrix:")
    print(confusion_matrix(y_eval, y_pred))

    print("\nClassification Report:")
    print(classification_report(y_eval, y_pred))

    w0, w1, w2 = clf.theta

    x1_vals = np.linspace(min(x_eval[:, 1]), max(x_eval[:, 1]), 100)
    x2_vals = -(w0 + w1 * x1_vals) / w2

    plt.scatter(x_eval[:, 1], x_eval[:, 2], c=y_eval, cmap="bwr", alpha=0.7)
    plt.plot(x1_vals, x2_vals, color="k", linewidth=2)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("Decision Boundary on Validation Set")
    plt.show()

    # *** END CODE HERE ***
if __name__ == "__main__":
    import sys
    main("./data/ds1_train.csv", "./data/ds1_valid.csv", "./data/ds1_pred.csv")
```

```
Eval accuracy = 0.9000
Confusion Matrix:
[[42  8]
 [ 2 48]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.95      0.84      0.89        50
         1.0       0.86      0.96      0.91        50

    accuracy                           0.90       100
   macro avg       0.91      0.90      0.90       100
weighted avg       0.91      0.90      0.90       100
```


Decision Boundary on Validation Set
1400
1200
1000
800
600
400
200
0
Feature 2
2
3
4
5
6
Feature 1

2 Gaussian Discriminant Analysis: Closed-Form for $p(y = 1 \mid x)$

In GDA, we assume

$$y \sim \text{Bernoulli}(\phi), \quad p(x \mid y = 0) = \mathcal{N}(x;\, \mu_0, \Sigma), \quad p(x \mid y = 1) = \mathcal{N}(x;\, \mu_1, \Sigma),$$

where $\phi \in (0, 1)$, $\mu_0, \mu_1 \in \mathbb{R}^n$, and $\Sigma \in \mathbb{R}^{n\times n}$ is positive-definite and common to both classes. Then by Bayes' rule:

Posterior Probability $p(y = 1 \mid x)$

$$\begin{aligned} p(y = 1 \mid x;\, \phi, \mu_0, \mu_1, \Sigma) &= \frac{p(x \mid y = 1;\, \mu_1, \Sigma)\, p(y = 1;\, \phi)}{p(x \mid y = 1;\, \mu_1, \Sigma)\, p(y = 1;\, \phi) + \, p(x \mid y = 0;\, \mu_0, \Sigma)\, p(y = 0;\, \phi)} \\ &= \frac{1}{1 \; + \; \exp\Bigl(\log\bigl[\frac{p(x|y=0)\, p(y=0)}{p(x|y=1)\, p(y=1)}\bigr]\Bigr)}. \end{aligned}$$

Because $p(x \mid y = 1)$ and $p(x \mid y = 0)$ share the same covariance Σ, we can rewrite the denominator and get a sigmoid form. In particular:

Sigmoid Form of Posterior

$$p(y = 1 \mid x) \; = \; \frac{1}{1 + \frac{p(x|y=0)\, p(y=0)}{p(x|y=1)\, p(y=1)}} \; = \; \frac{1}{1 + \exp\Bigl(\log\bigl[\frac{p(x|y=0)\, p(y=0)}{p(x|y=1)\, p(y=1)}\bigr]\Bigr)}.$$

For a multivariate normal,

Gaussian Density $p(x \mid y = j)$

$$p(x \mid y = j) = \frac{1}{(2\pi)^{n/2}\, |\Sigma|^{1/2}} \; \exp\Bigl\{-\tfrac{1}{2}\,(x - \mu_j)^T \Sigma^{-1}(x - \mu_j)\Bigr\}, \quad j = 0, 1.$$

Therefore, the log-ratio becomes

Log-Ratio of Class-Conditional Terms

$$\log \frac{p(x \mid y=0)\, p(y=0)}{p(x \mid y=1)\, p(y=1)} = -\tfrac{1}{2}\, (x-\mu_0)^T \Sigma^{-1} (x-\mu_0) \;+\; \tfrac{1}{2}\, (x-\mu_1)^T \Sigma^{-1} (x-\mu_1) \;+\; \log \frac{1-\phi}{\phi}.$$

Plugging back into the sigmoid form:

Posterior in Terms of Exponential

$$p(y=1 \mid x) \;=\; \frac{1}{1 + \exp\Bigl(\underbrace{-\tfrac{1}{2}(x-\mu_1)^T \Sigma^{-1} (x-\mu_1) \;+\; \tfrac{1}{2}(x-\mu_0)^T \Sigma^{-1} (x-\mu_0) \;+\; \log \tfrac{1-\phi}{\phi}}_{=-(\theta^T x + \theta_0)}\Bigr)}.$$

Re-arrange the quadratic forms:

Quadratic Expansion

$$\begin{aligned}
&-\tfrac{1}{2}(x-\mu_1)^T \Sigma^{-1} (x-\mu_1) \;+\; \tfrac{1}{2}(x-\mu_0)^T \Sigma^{-1} (x-\mu_0) \;+\; \log \tfrac{1-\phi}{\phi} \\
&= -\tfrac{1}{2}\Bigl[x^T \Sigma^{-1} x - 2\,\mu_1^T \Sigma^{-1} x + \mu_1^T \Sigma^{-1} \mu_1\Bigr] + \tfrac{1}{2}\Bigl[x^T \Sigma^{-1} x - 2\,\mu_0^T \Sigma^{-1} x + \mu_0^T \Sigma^{-1} \mu_0\Bigr] + \log \tfrac{1-\phi}{\phi} \\
&= -\,x^T \Sigma^{-1} x + \mu_1^T \Sigma^{-1} x - \tfrac{1}{2}\,\mu_1^T \Sigma^{-1} \mu_1 + \tfrac{1}{2}\,x^T \Sigma^{-1} x - \mu_0^T \Sigma^{-1} x + \tfrac{1}{2}\,\mu_0^T \Sigma^{-1} \mu_0 + \log \tfrac{1-\phi}{\phi} \\
&= -\,(\mu_1 - \mu_0)^T \Sigma^{-1} x - \Bigl[\tfrac{1}{2}\,\mu_1^T \Sigma^{-1} \mu_1 - \tfrac{1}{2}\,\mu_0^T \Sigma^{-1} \mu_0\Bigr] + \log \tfrac{1-\phi}{\phi}.
\end{aligned}$$

Define

GDA Parameter Definitions

$$\theta \;=\; \Sigma^{-1}\bigl(\mu_1 - \mu_0\bigr), \qquad \theta_0 \;=\; -\tfrac{1}{2}\,\mu_1^T \Sigma^{-1} \mu_1 + \tfrac{1}{2}\,\mu_0^T \Sigma^{-1} \mu_0 - \log\Bigl(\tfrac{1-\phi}{\phi}\Bigr).$$

Then

Final Posterior Form

$$p\bigl(y=1 \mid x;\, \phi, \mu_0, \mu_1, \Sigma\bigr) = \frac{1}{1 + \exp\bigl(-(\theta^T x + \theta_0)\bigr)},$$

$$\theta = \Sigma^{-1}(\mu_1 - \mu_0), \quad \theta_0 = -\tfrac{1}{2}\,\mu_1^T \Sigma^{-1} \mu_1 + \tfrac{1}{2}\,\mu_0^T \Sigma^{-1} \mu_0 - \log\Bigl(\tfrac{1-\phi}{\phi}\Bigr).$$

---

gda

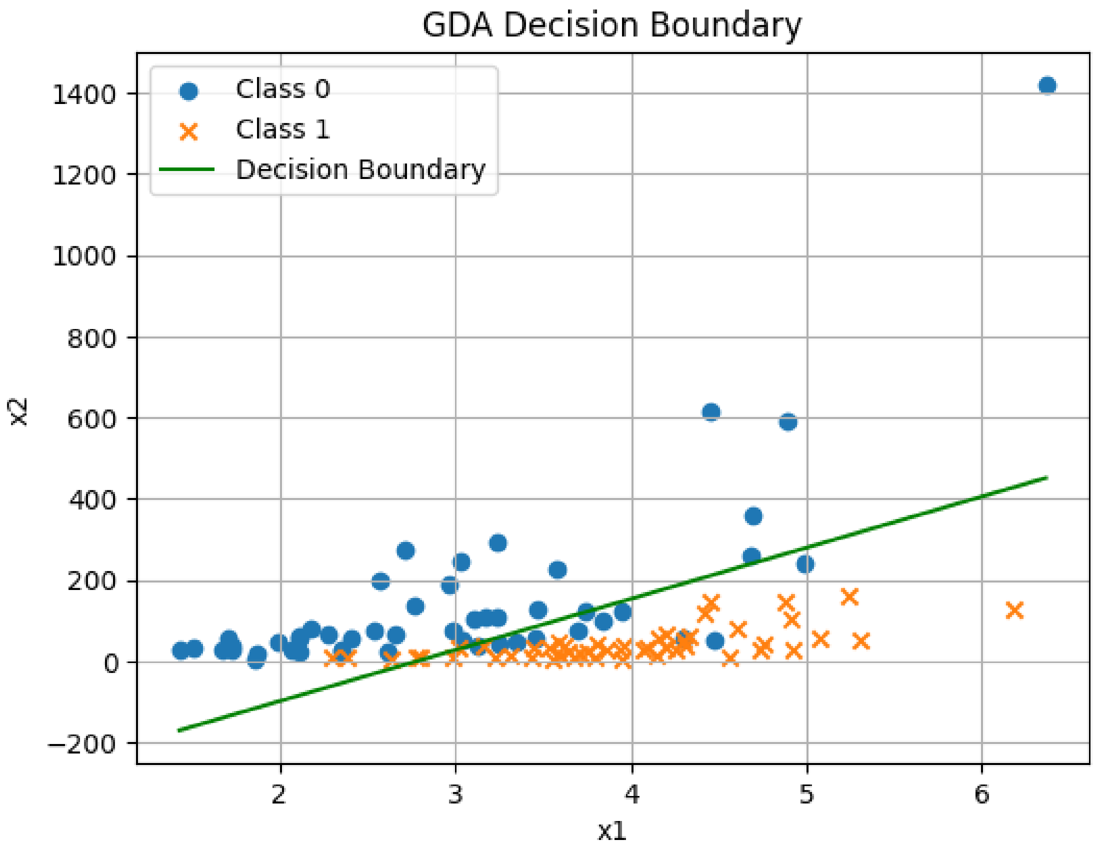
June 6, 2025

[5]:
```python
import numpy as np
import util

from linear_model import LinearModel
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

def plot_decision_boundary(x, y, theta):
    plt.figure()

    # Plot data points
    plt.scatter(x[y==0][:, 0], x[y==0][:, 1], label="Class 0", marker='o')
    plt.scatter(x[y==1][:, 0], x[y==1][:, 1], label="Class 1", marker='x')

    # Plot decision boundary:   +  x  +  x  = 0    x  = -(  +  x ) /
    x1_vals = np.linspace(np.min(x[:,0]), np.max(x[:,0]), 100)
    theta_0, theta_1, theta_2 = theta
    x2_vals = -(theta_0 + theta_1 * x1_vals) / theta_2
    plt.plot(x1_vals, x2_vals, label="Decision Boundary", color='green')

    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.legend()
    plt.title("GDA Decision Boundary")
    plt.grid(True)
    plt.show()


class GDA(LinearModel):
    """Gaussian Discriminant Analysis.

    Example usage:
        > clf = GDA()
        > clf.fit(x_train, y_train)
        > clf.predict(x_eval)
    """
```

```
    def fit(self, x, y):
        """Fit a GDA model to training set given by x and y.

        Args:
            x: Training example inputs. Shape (m, n).
            y: Training example labels. Shape (m,).

        Returns:
            theta: GDA model parameters.
        """
        # *** START CODE HERE ***
        phi = np.mean(y)
        mu_0 = np.mean(x[y == 0], axis=0)
        mu_1 = np.mean(x[y == 1], axis=0)
        cov_matrix = np.zeros((x.shape[1], x.shape[1]))
        for i in range(x.shape[0]):
            if y[i] == 0:
                cov_matrix += np.outer(x[i] - mu_0, x[i] - mu_0)
            else:
                cov_matrix += np.outer(x[i] - mu_1, x[i] - mu_1)
        cov_matrix /= x.shape[0]
        inverse_cov = np.linalg.inv(cov_matrix)

        theta_0 = -((mu_1.T) @ inverse_cov @ mu_1)/2 + (mu_0.T @ inverse_cov @
 ↪mu_0)/2 + np.log(phi / (1 - phi))

        theta = inverse_cov @ (mu_1 - mu_0)

        theta = np.concatenate(([theta_0], theta))
        self.theta = theta



        # *** END CODE HERE ***

    def predict(self, x):
        """Make a prediction given new inputs x.

        Args:
            x: Inputs of shape (m, n).

        Returns:
            Outputs of shape (m,).
        """
        # *** START CODE HERE ***
        if self.theta is None:
```

```
            raise ValueError("Model has not been fitted yet. Call fit() before␣
↪predict().")
        x_new = np.concatenate([np.ones((x.shape[0], 1)), x], axis=1)
        return x_new @ self.theta >= 0

        # *** END CODE HERE
def main(train_path, eval_path, pred_path):
    # Load dataset
    x_train, y_train = util.load_dataset(train_path, add_intercept=False)


    # *** START CODE HERE ***
    x_eval, y_eval = util.load_dataset(eval_path, add_intercept=False)

    clf = GDA()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_eval)
    np.savetxt(pred_path, y_pred)


    acc = accuracy_score(y_eval, y_pred)
    print(f"Accuracy on evaluation set: {acc:.4f}")

    if x_eval.shape[1] == 2:
        plot_decision_boundary(x_eval, y_eval, clf.theta)


    # *** END CODE HERE ***

if __name__ == "__main__":
    main("data/ds1_train.csv", "data/ds1_valid.csv", "output/predictions.txt")
```

```
Accuracy on evaluation set: 0.8300
```


GDA Decision Boundary
Class 0
Class 1
Decision Boundary
x2
1400
1200
1000
800
600
400
200
0
−200
2
3
4
5
6
x1