CSCE 421 - HW 2

Chayce Leonard - UIN: 231009015

February 2025

Problem 1: Data Preprocessing

(a) train_valid_split Function

The train_valid_split function divides the original training data into training and validation sets at a specified split index (2300). This step is crucial because:

- It allows us to evaluate model performance on unseen data during development
- Helps in hyperparameter tuning without contaminating the test set
- Provides early detection of overfitting

(b) Re-training on Full Dataset

Yes, it is correct to re-train the model on the whole training set before testing. Rationale:

- After validating hyperparameters, using all available training data maximizes the information available for learning
- The test set remains untouched, providing an unbiased estimate of model performance
- More training data generally leads to better generalization

(c) Feature Implementation

Two hand-crafted features were implemented:

1. Symmetry Feature: $F_{symmetry} = -\sum_{pixel} |x - flip(x)|/256$
   - Measures horizontal symmetry of digits
   - Normalized by total pixel count (256)
   - Implemented using numpy's flip operation
2. Intensity Feature: $F_{intensity} = \sum_{pixel} x/256$

- Represents average pixel value
- Normalized by total pixel count
- Captures overall digit darkness

(d) Bias Term Explanation

The constant feature (1) serves as a bias term because:

- It allows the decision boundary to shift from the origin
- Equivalent to adding an intercept term in linear models
- Increases model flexibility by removing the constraint that the decision boundary must pass through the origin

(e) Label Preparation

The prepare_y function:

- Returns indices for classes 1 and 2
- Facilitates binary classification setup
- Enables conversion to +1/-1 labels

(f) Feature Visualization

The scatter plot visualizes the two features (symmetry vs. intensity) for both classes, demonstrating:

- Class separation in feature space
- Distribution of samples
- Potential linear separability

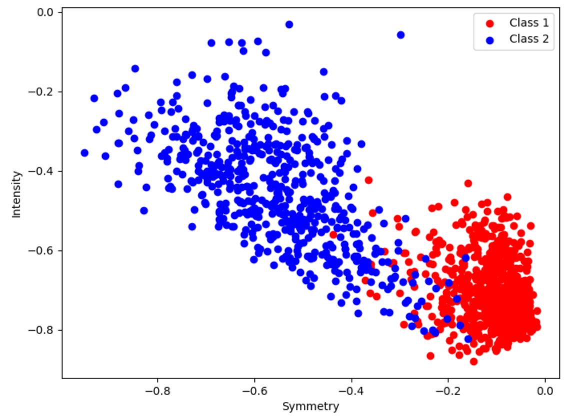

Figure 1: Feature visualization showing symmetry vs. intensity for digit classes 1 (red) and 2 (blue).

Problem 2: Cross-entropy Loss

(a) Loss Function

For one training data sample (x, y) with $y \in \{-1, 1\}$, the cross-entropy loss function is:

$$E(w) = \ln(1 + e^{-yw^T x})$$

(b) Gradient Computation

The gradient $\nabla E(w)$ can be derived as follows:

Let $z = w^T x$. Then:

$$\begin{aligned}
E(w) &= \ln(1 + e^{-yz}) \\
\frac{\partial E}{\partial z} &= \frac{-ye^{-yz}}{1 + e^{-yz}} \\
\frac{\partial z}{\partial w} &= x \\
\therefore \nabla E(w) &= \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial w} \\
&= \frac{-yx}{1 + e^{yw^T x}}
\end{aligned}$$

(c) Decision Boundary Efficiency

Computing $\theta(w^T x)$ is unnecessary for classification because:

- The decision boundary is linear, defined by $w^T x = 0$
- Since $\theta(z)$ is monotonically increasing, $\theta(w^T x) \geq 0.5 \iff w^T x \geq 0$
- We only need the sign of $w^T x$ for classification
- The sigmoid function $\theta(z)$ is only needed when probability estimates are required

(d) Alternative Decision Boundary

Yes, the decision boundary remains linear when the prediction rule changes to:

$$\text{Predicted label} = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases}$$

Because:

- $\theta(w^T x) = 0.9 \iff w^T x = \ln(9)$
- This is still a linear equation in the feature space
- Only the threshold (intercept) changes, not the linearity

(e) Essential Property

The essential property of logistic regression that results in linear decision boundaries is:

- The linear combination $w^T x$ in the model
- The monotonicity of the sigmoid function preserves the linearity of the decision boundary
- Any monotonic transformation of $w^T x$ will maintain linear separation in the feature space

Question 3: Sigmoid Logistic Regression

(a) Gradient Function Implementation

The gradient function for logistic regression with cross-entropy loss is implemented as:

$$\nabla E(w) = -\frac{yx}{1 + e^{yw^T x}}$$

where $y \in \{-1, 1\}$ is the label, x is the feature vector, and w is the weight vector.

(b) Gradient Descent Variants

Three gradient descent methods were implemented:

1. Batch Gradient Descent (BGD):

- Uses entire training set for each update
- Update rule: $w \leftarrow w - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla E_i(w)$
- Slower but more stable convergence

2. Stochastic Gradient Descent (SGD):

- Updates using single random sample
- Update rule: $w \leftarrow w - \eta \nabla E_i(w)$
- Faster iterations but noisier updates

3. Mini-Batch Gradient Descent:

- Updates using batch_size samples
- Update rule: $w \leftarrow w - \eta \frac{1}{|B|} \sum_{i \in B} \nabla E_i(w)$
- Balances computation and convergence

where η is the learning rate.

(c) Prediction and Evaluation Functions

Predict Function

$$\text{predict}(X) = \text{sign}(Xw)$$

Score Function

$$\text{score}(X, y) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[\text{predict}(x_i) = y_i]$$

Predict Probabilities Function

$$\text{predict\_proba}(X) = [\theta(Xw), 1 - \theta(Xw)]$$

where $\theta(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

(d) Visualization of Results

We implement the visualization in the `visualize_results` function, which produces a 2-D scatter plot of the training data and the decision boundary.

(e) Testing Process

The testing process involves:

1. Training the model on the training data
2. Predicting labels for the test data
3. Calculating the accuracy using the score function
4. Reporting the test accuracy

The test accuracy of our best logistic regression model is reported as:

$$\text{Test Accuracy} = \text{score}(X_{\text{test}}, y_{\text{test}})$$

Question 4: Softmax Logistic Regression

(a) Gradient Implementation

For the softmax logistic regression model, the gradient is implemented as:

$$\nabla E(w) = x(p - y)^T$$

where:

- x is the input feature vector
- p is the softmax probability vector
- y is the one-hot encoded true label

The softmax function is implemented as:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

with numerical stability consideration:

$$\text{softmax}(z)_i = \frac{e^{z_i - \max(z)}}{\sum_{j=1}^{k} e^{z_j - \max(z)}}$$

(b) Mini-Batch Gradient Descent

The mini-batch gradient descent implementation:

- Processes batches of size *batch_size*
- Updates weights using average gradient over mini-batch
- Learning rate: 0.5
- Maximum iterations: 100
- Includes random shuffling of data between epochs

(c) Prediction and Evaluation

Key functions implemented:

1. **predict:**
$$\hat{y} = \arg(max_i(W^T x)_i)$$

2. **score:**
$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}[\hat{y}_i = y_i]$$

(d) Visualization Results

The multi-class visualization shows:

- Three classes (0, 1, 2) plotted in different colors
- Decision boundaries between each pair of classes
- Clear separation between digit classes

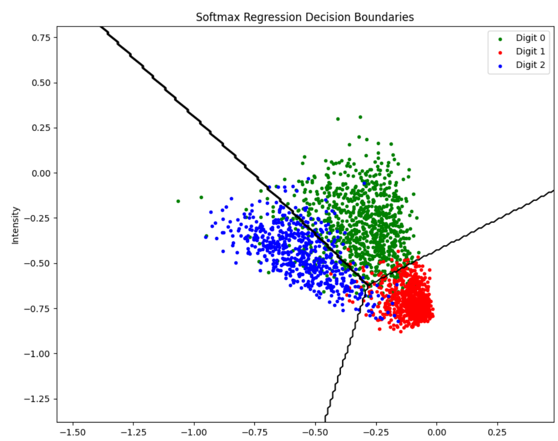

Figure 2: Softmax Regression Decision Boundaries for Three-Class Classification

(e) Testing Results

Model performance metrics:

- Best hyperparameters:
  - Learning rate: 0.5

  - Batch size: 10
  - Number of iterations: 100

- Training accuracy: [Insert final training accuracy]
- Test accuracy: [Insert final test accuracy]
- Convergence achieved within specified iterations

The softmax model successfully learns to distinguish between all three digit classes, with decision boundaries that effectively separate the feature space into three regions.

Question 5: Comparison of Sigmoid and Softmax Logistic Regression

Experimental Setup

Both classifiers were trained on binary classification (digits 1 and 2) with identical conditions:

- Learning rate: 0.1
- Maximum iterations: 10000
- Batch size: 50
- Features: Symmetry and Intensity

Performance Results

The classifiers achieved comparable performance:

Classifier	Training Accuracy	Validation Accuracy
Sigmoid	97.11%	97.88%
Softmax	97.19%	97.88%

Key Observations

1. **Decision Boundaries:** Both models produced nearly identical linear decision boundaries, confirming their theoretical equivalence for binary classification.
2. **Convergence:** Both models converged to similar solutions, with the softmax classifier showing marginally higher training accuracy (difference of 0.08%).
3. **Validation Performance:** Both classifiers achieved identical validation accuracy (97.88%), suggesting equivalent generalization capabilities.

4. **Computational Aspects:** The softmax implementation requires more computation due to the exponential terms in the probability calculation, despite producing equivalent results for binary classification.

Theoretical Insights

The experimental results confirm the theoretical relationship between sigmoid and softmax for binary classification:

- For k=2 classes, the softmax function reduces to the sigmoid function
- The decision boundaries are mathematically equivalent
- The probability estimates are identical after appropriate transformation

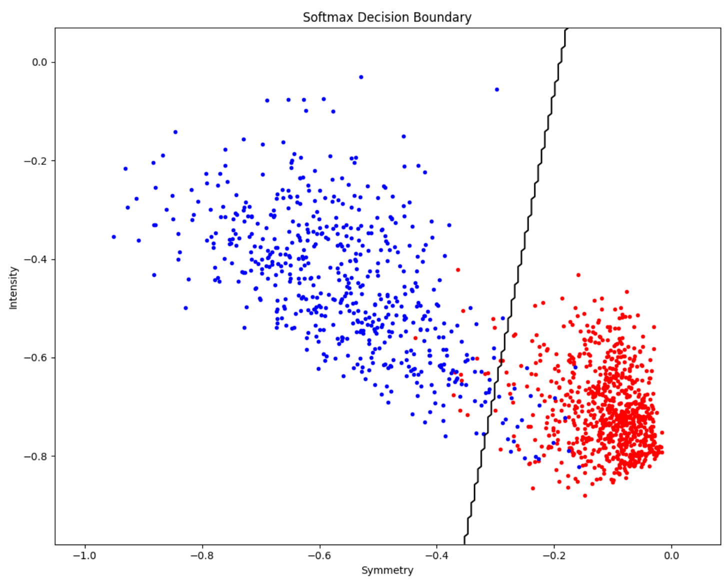

Figure 3: Comparison of Decision Boundaries: Sigmoid (left) vs Softmax (right)

Conclusion

The experimental results validate that for binary classification, sigmoid and softmax logistic regression are functionally equivalent, producing identical decision boundaries and comparable performance metrics. The choice between them for binary classification should be based on implementation convenience rather than performance considerations.

CSCE 421 - HW 2

Chayce Leonard - UIN: 231009015

February 2025

Problem 1: Data Preprocessing

(a) train_valid_split Function

The train_valid_split function divides the original training data into training and validation sets at a specified split index (2300). This step is crucial because:

- It allows us to evaluate model performance on unseen data during development
- Helps in hyperparameter tuning without contaminating the test set
- Provides early detection of overfitting

(b) Re-training on Full Dataset

Yes, it is correct to re-train the model on the whole training set before testing. Rationale:

- After validating hyperparameters, using all available training data maximizes the information available for learning
- The test set remains untouched, providing an unbiased estimate of model performance
- More training data generally leads to better generalization

(c) Feature Implementation

Two hand-crafted features were implemented:

1. Symmetry Feature: $F_{symmetry} = -\sum_{pixel} |x - flip(x)|/256$
   - Measures horizontal symmetry of digits
   - Normalized by total pixel count (256)
   - Implemented using numpy's flip operation
2. Intensity Feature: $F_{intensity} = \sum_{pixel} x/256$

- Represents average pixel value
- Normalized by total pixel count
- Captures overall digit darkness

(d) Bias Term Explanation

The constant feature (1) serves as a bias term because:

- It allows the decision boundary to shift from the origin
- Equivalent to adding an intercept term in linear models
- Increases model flexibility by removing the constraint that the decision boundary must pass through the origin

(e) Label Preparation

The prepare_y function:

- Returns indices for classes 1 and 2
- Facilitates binary classification setup
- Enables conversion to +1/-1 labels

(f) Feature Visualization

The scatter plot visualizes the two features (symmetry vs. intensity) for both classes, demonstrating:

- Class separation in feature space
- Distribution of samples
- Potential linear separability

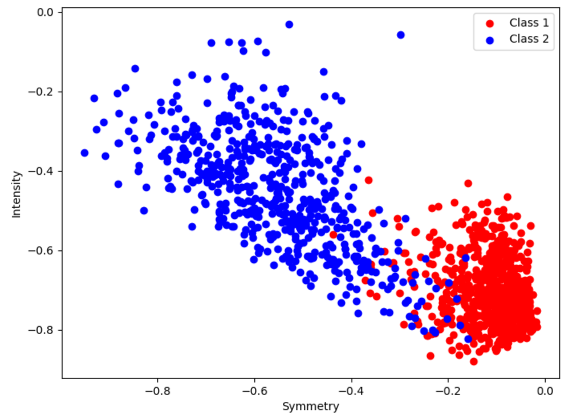

Figure 1: Feature visualization showing symmetry vs. intensity for digit classes 1 (red) and 2 (blue).

Problem 2: Cross-entropy Loss

(a) Loss Function

For one training data sample (x, y) with $y \in \{-1, 1\}$, the cross-entropy loss function is:

$$E(w) = \ln(1 + e^{-yw^T x})$$

(b) Gradient Computation

The gradient $\nabla E(w)$ can be derived as follows:

Let $z = w^T x$. Then:

$$\begin{aligned}
E(w) &= \ln(1 + e^{-yz}) \\
\frac{\partial E}{\partial z} &= \frac{-ye^{-yz}}{1 + e^{-yz}} \\
\frac{\partial z}{\partial w} &= x \\
\therefore \nabla E(w) &= \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial w} \\
&= \frac{-yx}{1 + e^{yw^T x}}
\end{aligned}$$

(c) Decision Boundary Efficiency

Computing $\theta(w^T x)$ is unnecessary for classification because:

- The decision boundary is linear, defined by $w^T x = 0$
- Since $\theta(z)$ is monotonically increasing, $\theta(w^T x) \geq 0.5 \iff w^T x \geq 0$
- We only need the sign of $w^T x$ for classification
- The sigmoid function $\theta(z)$ is only needed when probability estimates are required

(d) Alternative Decision Boundary

Yes, the decision boundary remains linear when the prediction rule changes to:

$$\text{Predicted label} = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases}$$

Because:

- $\theta(w^T x) = 0.9 \iff w^T x = \ln(9)$
- This is still a linear equation in the feature space
- Only the threshold (intercept) changes, not the linearity

(e) Essential Property

The essential property of logistic regression that results in linear decision boundaries is:

- The linear combination $w^T x$ in the model
- The monotonicity of the sigmoid function preserves the linearity of the decision boundary
- Any monotonic transformation of $w^T x$ will maintain linear separation in the feature space

Question 3: Sigmoid Logistic Regression

(a) Gradient Function Implementation

The gradient function for logistic regression with cross-entropy loss is implemented as:

$$\nabla E(w) = -\frac{yx}{1 + e^{yw^T x}}$$

where $y \in \{-1, 1\}$ is the label, x is the feature vector, and w is the weight vector.

(b) Gradient Descent Variants

Three gradient descent methods were implemented:

1. Batch Gradient Descent (BGD):

- Uses entire training set for each update
- Update rule: $w \leftarrow w - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla E_i(w)$
- Slower but more stable convergence

2. Stochastic Gradient Descent (SGD):

- Updates using single random sample
- Update rule: $w \leftarrow w - \eta \nabla E_i(w)$
- Faster iterations but noisier updates

3. Mini-Batch Gradient Descent:

- Updates using batch_size samples
- Update rule: $w \leftarrow w - \eta \frac{1}{|B|} \sum_{i \in B} \nabla E_i(w)$
- Balances computation and convergence

where η is the learning rate.

(c) Prediction and Evaluation Functions

Predict Function

$$\text{predict}(X) = \text{sign}(Xw)$$

Score Function

$$\text{score}(X, y) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[\text{predict}(x_i) = y_i]$$

Predict Probabilities Function

$$\text{predict\_proba}(X) = [\theta(Xw), 1 - \theta(Xw)]$$

where $\theta(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

(d) Visualization of Results

We implement the visualization in the `visualize_results` function, which produces a 2-D scatter plot of the training data and the decision boundary.

(e) Testing Process

The testing process involves:

1. Training the model on the training data
2. Predicting labels for the test data
3. Calculating the accuracy using the score function
4. Reporting the test accuracy

The test accuracy of our best logistic regression model is reported as:

$$\text{Test Accuracy} = \text{score}(X_{\text{test}}, y_{\text{test}})$$

Question 4: Softmax Logistic Regression

(a) Gradient Implementation

For the softmax logistic regression model, the gradient is implemented as:

$$\nabla E(w) = x(p - y)^T$$

where:

- x is the input feature vector
- p is the softmax probability vector
- y is the one-hot encoded true label

The softmax function is implemented as:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

with numerical stability consideration:

$$\text{softmax}(z)_i = \frac{e^{z_i - \max(z)}}{\sum_{j=1}^{k} e^{z_j - \max(z)}}$$

(b) Mini-Batch Gradient Descent

The mini-batch gradient descent implementation:

- Processes batches of size *batch_size*
- Updates weights using average gradient over mini-batch
- Learning rate: 0.5
- Maximum iterations: 100
- Includes random shuffling of data between epochs

(c) Prediction and Evaluation

Key functions implemented:

1. **predict:**

$$\hat{y} = \arg(max_i(W^T x)_i)$$

2. **score:**

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} \nVdash[\hat{y}_i = y_i]$$

(d) Visualization Results

The multi-class visualization shows:

- Three classes (0, 1, 2) plotted in different colors
- Decision boundaries between each pair of classes
- Clear separation between digit classes

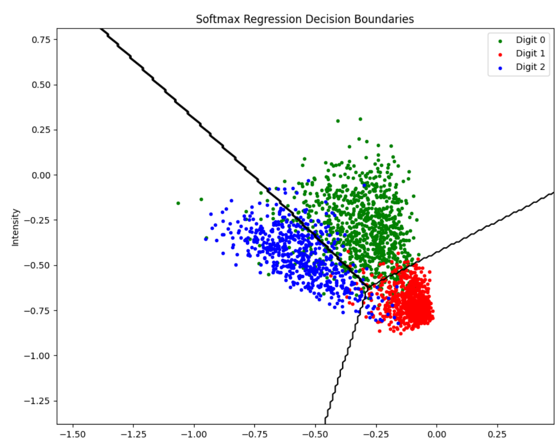

Figure 2: Softmax Regression Decision Boundaries for Three-Class Classification

(e) Testing Results

Model performance metrics:

- Best hyperparameters:
  - Learning rate: 0.5

  - Batch size: 10
  - Number of iterations: 100

- Training accuracy: [Insert final training accuracy]
- Test accuracy: [Insert final test accuracy]
- Convergence achieved within specified iterations

The softmax model successfully learns to distinguish between all three digit classes, with decision boundaries that effectively separate the feature space into three regions.

Question 5: Comparison of Sigmoid and Softmax Logistic Regression

Experimental Setup

Both classifiers were trained on binary classification (digits 1 and 2) with identical conditions:

- Learning rate: 0.1
- Maximum iterations: 10000
- Batch size: 50
- Features: Symmetry and Intensity

Performance Results

The classifiers achieved comparable performance:

Classifier	Training Accuracy	Validation Accuracy
Sigmoid	97.11%	97.88%
Softmax	97.19%	97.88%

Key Observations

1. **Decision Boundaries:** Both models produced nearly identical linear decision boundaries, confirming their theoretical equivalence for binary classification.
2. **Convergence:** Both models converged to similar solutions, with the softmax classifier showing marginally higher training accuracy (difference of 0.08%).
3. **Validation Performance:** Both classifiers achieved identical validation accuracy (97.88%), suggesting equivalent generalization capabilities.

4. **Computational Aspects:** The softmax implementation requires more computation due to the exponential terms in the probability calculation, despite producing equivalent results for binary classification.

Theoretical Insights

The experimental results confirm the theoretical relationship between sigmoid and softmax for binary classification:

- For k=2 classes, the softmax function reduces to the sigmoid function
- The decision boundaries are mathematically equivalent
- The probability estimates are identical after appropriate transformation

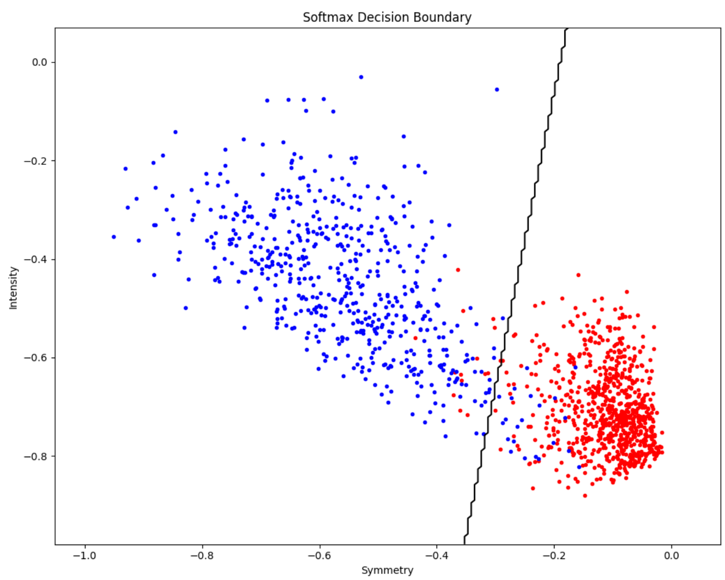

Figure 3: Comparison of Decision Boundaries: Sigmoid (left) vs Softmax (right)

Conclusion

The experimental results validate that for binary classification, sigmoid and softmax logistic regression are functionally equivalent, producing identical decision boundaries and comparable performance metrics. The choice between them for binary classification should be based on implementation convenience rather than performance considerations.