CSC 4315: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Week 5, Lecture 9
Logistic Regression and Classification Metrics

Lecture Overview

Logistic regression extends linear models to classification problems. This lecture covers the sigmoid function, binary cross-entropy loss, gradient descent for logistic regression, and essential classification metrics.

Topics Covered:

- Classification vs regression
- The sigmoid (logistic) function
- Binary cross-entropy loss
- Gradient descent for logistic regression
- Classification metrics: accuracy, precision, recall, F1
- Confusion matrix interpretation

1. From Regression to Classification

1.1 The Classification Problem

```
import numpy as np

print("Classification Problem")
print("="*70)

print("Regression vs Classification:")
print("  Regression: Predict continuous values (house prices)")
print("  Classification: Predict discrete categories (spam/not spam)")

print("\nBinary Classification Examples:")
print("  - Email: spam (1) or not spam (0)")
print("  - Medical: disease (1) or healthy (0)")
print("  - Finance: fraud (1) or legitimate (0)")

# Why not use linear regression for classification?
print("\nProblem with Linear Regression for Classification:")
print("  - Output can be < 0 or > 1 (not valid probabilities)")
print("  - Sensitive to outliers")
print("  - Decision boundary is affected by extreme values")

# Example data: exam scores and pass/fail
np.random.seed(42)
scores = np.array([30, 35, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90])
passed = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])

print(f"\nExample: Exam Pass/Fail Prediction")
print(f"Scores: {scores}")
print(f"Passed: {passed}")
print(f"\nTask: Predict if a student will pass based on score")
```

```
Classification Problem
======================================================================
Regression vs Classification:
  Regression: Predict continuous values (house prices)
  Classification: Predict discrete categories (spam/not spam)

Binary Classification Examples:
  - Email: spam (1) or not spam (0)
  - Medical: disease (1) or healthy (0)
  - Finance: fraud (1) or legitimate (0)

Problem with Linear Regression for Classification:
  - Output can be < 0 or > 1 (not valid probabilities)
  - Sensitive to outliers
  - Decision boundary is affected by extreme values

Example: Exam Pass/Fail Prediction
Scores: [30 35 45 50 55 60 65 70 75 80 85 90]
Passed: [0 0 0 0 0 1 1 1 1 1 1 1]

Task: Predict if a student will pass based on score
```

2. The Sigmoid Function

2.1 Squashing Outputs to Probabilities

```python
# The Sigmoid Function
print("The Sigmoid Function")
print("="*70)

def sigmoid(z):
    """
    Sigmoid activation function
    Maps any real number to (0, 1)
    """
    return 1 / (1 + np.exp(-z))

print("Sigmoid function: sigma(z) = 1 / (1 + exp(-z))")
print("\nProperties:")
print("  - Output always between 0 and 1")
print("  - sigma(0) = 0.5")
print("  - Large positive z -> output near 1")
print("  - Large negative z -> output near 0")

# Demonstrate sigmoid
z_values = np.array([-10, -5, -2, -1, 0, 1, 2, 5, 10])
print("\nSigmoid values:")
print(f"{'z':>8} | {'sigma(z)':>10}")
print("-" * 22)
for z in z_values:
    print(f"{z:>8} | {sigmoid(z):>10.6f}")

print("\nLogistic Regression Hypothesis:")
print("  h(x) = sigma(w*x + b) = P(y=1|x)")
print("  Interprets output as probability of class 1")
```

```
The Sigmoid Function
======================================================================
Sigmoid function: sigma(z) = 1 / (1 + exp(-z))

Properties:
  - Output always between 0 and 1
  - sigma(0) = 0.5
  - Large positive z -> output near 1
  - Large negative z -> output near 0

Sigmoid values:
       z |   sigma(z)
----------------------
     -10 |   0.000045
      -5 |   0.006693
      -2 |   0.119203
      -1 |   0.268941
       0 |   0.500000
       1 |   0.731059
       2 |   0.880797
       5 |   0.993307
      10 |   0.999955

Logistic Regression Hypothesis:
  h(x) = sigma(w*x + b) = P(y=1|x)
  Interprets output as probability of class 1
```

3. Binary Cross-Entropy Loss

3.1 Why MSE Doesn't Work

```
# Binary Cross-Entropy Loss
print("Binary Cross-Entropy Loss")
print("="*70)

print("Why not MSE for classification?")
print("  - With sigmoid, MSE creates non-convex cost surface")
print("  - Multiple local minima make optimization hard")

print("\nBinary Cross-Entropy (Log Loss):")
print("  L(y, p) = -[y*log(p) + (1-y)*log(1-p)]")
print("\nFor a dataset:")
print("  J = -(1/m) * sum[y*log(h(x)) + (1-y)*log(1-h(x))]")

def binary_cross_entropy(y_true, y_pred):
    """
    Compute binary cross-entropy loss
    y_pred should be probabilities (0-1)
    """
    # Clip predictions to avoid log(0)
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)

    loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return loss

# Intuition: penalty for wrong predictions
print("\nIntuition:")
print("  If y=1: Loss = -log(p)")
print("    - p close to 1: low loss")
print("    - p close to 0: high loss (penalty!)")
print("  If y=0: Loss = -log(1-p)")
print("    - p close to 0: low loss")
print("    - p close to 1: high loss (penalty!)")

# Example
y_true = np.array([1, 0, 1, 0])
y_pred_good = np.array([0.9, 0.1, 0.8, 0.2])
y_pred_bad = np.array([0.3, 0.7, 0.4, 0.8])

print(f"\nExample:")
print(f"  True labels: {y_true}")
print(f"  Good predictions: {y_pred_good} -> Loss: {binary_cross_entropy(y_true, y_pred_good):.4f}")
print(f"  Bad predictions:  {y_pred_bad} -> Loss: {binary_cross_entropy(y_true, y_pred_bad):.4f}")
```

```
Binary Cross-Entropy Loss
======================================================================
Why not MSE for classification?
  - With sigmoid, MSE creates non-convex cost surface
  - Multiple local minima make optimization hard

Binary Cross-Entropy (Log Loss):
  L(y, p) = -[y*log(p) + (1-y)*log(1-p)]

For a dataset:
  J = -(1/m) * sum[y*log(h(x)) + (1-y)*log(1-h(x))]

Intuition:
  If y=1: Loss = -log(p)
    - p close to 1: low loss
    - p close to 0: high loss (penalty!)
  If y=0: Loss = -log(1-p)
    - p close to 0: low loss
    - p close to 1: high loss (penalty!)

Example:
  True labels: [1 0 1 0]
  Good predictions: [0.9 0.1 0.8 0.2] -> Loss: 0.1643
  Bad predictions:  [0.3 0.7 0.4 0.8] -> Loss: 1.0729
```

4. Logistic Regression Implementation

4.1 Complete Implementation

```
# Complete Logistic Regression
print("Logistic Regression Implementation")
print("="*70)

class LogisticRegression:
    """Logistic Regression from scratch"""

    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.lr = learning_rate
        self.n_iter = n_iterations
        self.weights = None
        self.bias = None
        self.losses = []

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Initialize parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        # Gradient descent
        for _ in range(self.n_iter):
            # Forward pass
            linear = np.dot(X, self.weights) + self.bias
            predictions = sigmoid(linear)

            # Compute gradients
            dw = (1/n_samples) * np.dot(X.T, (predictions - y))
            db = (1/n_samples) * np.sum(predictions - y)

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

            # Track loss
            loss = binary_cross_entropy(y, predictions)
            self.losses.append(loss)

        return self

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

    def predict(self, X, threshold=0.5):
        probabilities = self.predict_proba(X)
        return (probabilities >= threshold).astype(int)

# Prepare data
X = scores.reshape(-1, 1) / 100  # Normalize
y = passed

# Train model
model = LogisticRegression(learning_rate=1.0, n_iterations=1000)
model.fit(X, y)

print(f"Trained model:")
print(f"  Weight: {model.weights[0]:.4f}")
print(f"  Bias: {model.bias:.4f}")
print(f"  Final loss: {model.losses[-1]:.4f}")

# Predictions
probs = model.predict_proba(X)
preds = model.predict(X)
print(f"\nPredictions:")
print(f"  Scores:        {scores}")
print(f"  Actual:        {y}")
print(f"  Predicted:     {preds}")
print(f"  Probabilities: {np.round(probs, 2)}")
```

```
Logistic Regression Implementation
======================================================================
Trained model:
  Weight: 17.9621
  Bias: -9.8643
  Final loss: 0.1742

Predictions:
  Scores:        [30 35 45 50 55 60 65 70 75 80 85 90]
  Actual:        [0 0 0 0 0 1 1 1 1 1 1 1]
  Predicted:     [0 0 0 0 0 1 1 1 1 1 1 1]
  Probabilities: [0.02 0.05 0.21 0.4  0.6  0.78 0.9  0.96 0.99 1.   1.   1.  ]
```

5. Classification Metrics

5.1 Confusion Matrix

```
# Classification Metrics
print("Classification Metrics")
print("="*70)

def confusion_matrix(y_true, y_pred):
    """Compute confusion matrix"""
    tp = np.sum((y_true == 1) & (y_pred == 1))  # True Positives
    tn = np.sum((y_true == 0) & (y_pred == 0))  # True Negatives
    fp = np.sum((y_true == 0) & (y_pred == 1))  # False Positives
    fn = np.sum((y_true == 1) & (y_pred == 0))  # False Negatives
    return tp, tn, fp, fn

# Example with more data
y_true = np.array([1, 0, 1, 1, 0, 1, 0, 0, 1, 0])
y_pred = np.array([1, 0, 1, 0, 0, 1, 1, 0, 1, 0])

tp, tn, fp, fn = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(f"                  Predicted")
print(f"                  Neg   Pos")
print(f"  Actual Neg      {tn:3d}   {fp:3d}")
print(f"         Pos      {fn:3d}   {tp:3d}")

print("\nTerminology:")
print(f"  True Positives (TP): {tp} - Correctly predicted positive")
print(f"  True Negatives (TN): {tn} - Correctly predicted negative")
print(f"  False Positives (FP): {fp} - Incorrectly predicted positive (Type I)")
print(f"  False Negatives (FN): {fn} - Incorrectly predicted negative (Type II)")
```

```
Classification Metrics
======================================================================
Confusion Matrix:
                  Predicted
                  Neg   Pos
  Actual Neg        4     1
         Pos        1     4

Terminology:
  True Positives (TP): 4 - Correctly predicted positive
  True Negatives (TN): 4 - Correctly predicted negative
  False Positives (FP): 1 - Incorrectly predicted positive (Type I)
  False Negatives (FN): 1 - Incorrectly predicted negative (Type II)
```

5.2 Accuracy, Precision, Recall, F1

```
# Core Classification Metrics
print("Core Classification Metrics")
print("="*70)

def compute_metrics(y_true, y_pred):
    tp, tn, fp, fn = confusion_matrix(y_true, y_pred)

    # Accuracy: Overall correctness
    accuracy = (tp + tn) / (tp + tn + fp + fn)

    # Precision: Of predicted positives, how many are correct?
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0

    # Recall (Sensitivity): Of actual positives, how many did we catch?
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0

    # F1 Score: Harmonic mean of precision and recall
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1
    }

metrics = compute_metrics(y_true, y_pred)

print("Metrics computed:")
for name, value in metrics.items():
    print(f"  {name.capitalize():12}: {value:.4f}")

print("\nWhen to use which metric:")
print("  Accuracy:  Good for balanced datasets")
print("  Precision: Important when FP is costly (spam detection)")
print("  Recall:    Important when FN is costly (disease detection)")
print("  F1 Score:  Balance of precision and recall")

# Imbalanced example
print("\n--- Imbalanced Dataset Example ---")
y_true_imb = np.array([0]*90 + [1]*10)  # 90% negative
y_pred_imb = np.array([0]*100)  # Predict all negative

metrics_imb = compute_metrics(y_true_imb, y_pred_imb)
print(f"Predicting all zeros on 90/10 split:")
print(f"  Accuracy:  {metrics_imb['accuracy']:.2f} (misleadingly high!)")
print(f"  Precision: {metrics_imb['precision']:.2f}")
print(f"  Recall:    {metrics_imb['recall']:.2f} (catches none!)")
print(f"  F1:        {metrics_imb['f1']:.2f}")
```

```
Core Classification Metrics
======================================================================
Metrics computed:
  Accuracy    : 0.8000
  Precision   : 0.8000
  Recall      : 0.8000
  F1          : 0.8000

When to use which metric:
  Accuracy:  Good for balanced datasets
  Precision: Important when FP is costly (spam detection)
  Recall:    Important when FN is costly (disease detection)
  F1 Score:  Balance of precision and recall

--- Imbalanced Dataset Example ---
Predicting all zeros on 90/10 split:
  Accuracy:  0.90 (misleadingly high!)
  Precision: 0.00
  Recall:    0.00 (catches none!)
  F1:        0.00
```

Summary

Key Takeaways:

- Logistic regression uses sigmoid to output probabilities
- Binary cross-entropy is the proper loss for classification
- The decision boundary is at probability = 0.5 (threshold)
- Confusion matrix shows TP, TN, FP, FN
- Accuracy can be misleading on imbalanced data
- Precision/Recall trade-off depends on problem context
- F1 score balances precision and recall

Practice Exercises:

1. Implement logistic regression for multiple features
2. Plot the decision boundary for 2D classification
3. Experiment with different classification thresholds
4. Compute ROC curve and AUC score
5. Handle imbalanced datasets with class weights