AI/ML Bootcamp: Foundations and Applications

Week 2: Linear Classifiers, Logistic Regression, Bias-Variance Trade-off, and Regularization

Mohammad Saidur Rahman
msrahman3@utep.edu
Department of Computer Science
University of Texas at El Paso (UTEP)

Mohammad Ishtiaque Rahman
mrahman5@umbc.edu
Department of Computer Science
Pennsylvania Western University

Contents

1 Introduction to Week 2 Topics in the Context of Machine Learning

In this week, we will explore fundamental machine learning techniques that are widely used for classification tasks: **Linear Classifiers** and **Logistic Regression**. Additionally, we will cover core concepts like the **Bias-Variance Trade-off** and **Regularization**, which help in understanding the performance and generalization of machine learning models.

These concepts are essential for building accurate and interpretable models that can classify data and predict outcomes in various fields. Understanding when and why to use these techniques is key to solving different types of problems in machine learning.

2 Linear Classifiers

2.1 Why Linear Classifiers?

Linear classifiers are one of the simplest and most interpretable models in machine learning. They work well when the relationship between input features and output labels is approximately linear. A linear classifier uses a straight line (or hyperplane in higher dimensions) to separate data points into different classes.

The utility of linear classifiers lies in their simplicity and efficiency. They are particularly useful when:

- The data is linearly separable, meaning a straight line can divide the data into distinct classes.
- The model needs to be interpretable, where the decision boundary is easy to understand and explain.
- The dataset is large, and computational efficiency is important.

2.2 Mathematical Representation

The general form of a linear classifier is:

$$y = \mathbf{w}^T \mathbf{x} + b$$

Where:

- $\mathbf{w}$ is the weight vector that determines the orientation of the decision boundary.
- $\mathbf{x}$ is the input feature vector.
- b is the bias term that adjusts the position of the boundary.
- y is the predicted class label.

The classifier predicts a class label by applying a threshold to the output. In binary classification, the decision rule is:

$$\hat{y} = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0 \end{cases}$$

This simple decision rule is effective for linearly separable problems. The decision boundary, where $\mathbf{w}^T\mathbf{x} + b = 0$, is the point where the model is equally likely to classify an input as either class.

2.3 What Types of Problems Can Be Solved by Linear Classifiers?

Linear classifiers are effective for problems where the relationship between the input features and the class labels is linear or nearly linear. Some example problems include:

- **Binary classification problems**: Classifying whether a patient has a disease or not, based on their medical records (e.g., Pima Indians Diabetes dataset [4]).
- **Document classification**: Classifying emails as spam or not spam based on word frequencies (e.g., SMS Spam dataset).
- **Image recognition**: Classifying images of handwritten digits as 0–9 (e.g., MNIST dataset).

Although linear classifiers are simple, they may not perform well on complex, non-linear datasets. In such cases, we need more advanced techniques like logistic regression or support vector machines.

2.4 Example: Classifying Iris Flowers

Let's use the Iris dataset [1] to demonstrate how a linear classifier works.

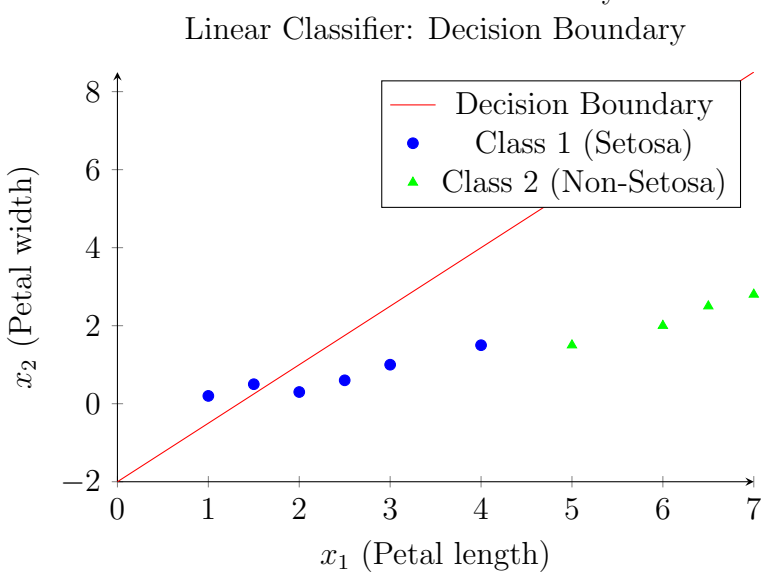
```python
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
import numpy as np

# Load the Iris dataset
iris = load_iris()
X = iris["data"][:, (2, 3)]  # Petal length and width
y = (iris["target"] == 0).astype(np.int32)  # Setosa vs non-Setosa

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Train a linear classifier
clf = SGDClassifier(loss='hinge', random_state=42)
clf.fit(X_train, y_train)

# Test the classifier
y_pred = clf.predict(X_test)
print("Predictions:", y_pred)
```

Listing 1: Example: Linear Classifier for Iris Dataset

This example classifies iris flowers as Setosa or non-Setosa using a linear classifier (Stochastic Gradient Descent with hinge loss).

2.5 Visualization: Linear Decision Boundary

Below is a visualization of the decision boundary for a linear classifier:



The decision boundary separates Setosa from non-Setosa flowers based on petal length and width.

2.6 Limitations of Linear Classifiers

Linear classifiers are limited to datasets that are linearly separable. In real-world problems, many datasets are non-linear, requiring more sophisticated models. When the data cannot be separated by a straight line or hyperplane, a more flexible classifier such as logistic regression or support vector machines (SVM) may be needed.

3 Logistic Regression

3.1 Why Logistic Regression?

Logistic regression extends the idea of linear classifiers by predicting probabilities for classification tasks. It is particularly useful for binary classification problems where the output is a probability between 0 and 1. Unlike linear regression, which can predict continuous values, logistic regression is designed to handle classification tasks by modeling the probability of a binary outcome.

We use logistic regression when:

- The relationship between the features and the target variable is not strictly linear.
- The task requires probabilities rather than exact predictions.
- The output is binary or categorical.

3.2 Mathematical Representation of Logistic Regression

In logistic regression, the linear combination of input features is passed through the **sigmoid function**, which maps any real number to a probability between 0 and 1.

The hypothesis for logistic regression is:

$$h_\theta(x) = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

Where:

- $\mathbf{w}^T\mathbf{x} + b$ is the linear combination of input features.
- $\sigma(z)$ is the sigmoid function:
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
- $h_\theta(x)$ is the predicted probability of the positive class.

3.3 Why Not Use Linear Regression for Classification?

Linear regression is not ideal for binary classification problems because:

- **Output range**: Linear regression can predict any real number, but for classification tasks, we need probabilities between 0 and 1.

- **Boundaries**: Logistic regression applies a sigmoid function to the linear output, which forces the prediction into a range of probabilities.
- **Error measure**: Linear regression uses Mean Squared Error (MSE) as its loss function, which is not suited for binary classification. Logistic regression uses the **log-loss** (binary cross-entropy), which better handles the probabilistic nature of classification.

Logistic regression is preferred when we need probabilistic outputs, and it performs better than linear regression in cases where the relationship between features and target is not linear.

3.4 Step-by-Step Approach to Logistic Regression

1. **Model hypothesis**: The model outputs a probability, using the sigmoid function, of the input being in the positive class. 2. **Logistic function**: The linear combination of inputs is transformed using the sigmoid function, $\sigma(z) = \frac{1}{1+e^{-z}}$, to map it between 0 and 1. 3. **Decision boundary**: For binary classification, if $\sigma(z) \geq 0.5$, the class label is predicted as 1; otherwise, it is predicted as 0. 4. **Loss function**: Logistic regression minimizes the log-loss (binary cross-entropy):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

5. **Gradient descent**: The model parameters are updated iteratively to minimize the log-loss, similar to linear regression but using the gradient of the log-loss function.

3.5 Example: Logistic Regression with Titanic Dataset

Let's use logistic regression to predict the survival of passengers in the Titanic dataset [2].

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Titanic dataset (source: Kaggle)
titanic = pd.read_csv('https://path/to/titanic.csv')
X = titanic[['Pclass', 'Age', 'Fare', 'Sex']]  # Features
X['Sex'] = X['Sex'].map({'male': 0, 'female': 1})  # Convert categorical
    to numerical
y = titanic['Survived']  # Target

# Fill missing values and scale features
X['Age'].fillna(X['Age'].mean(), inplace=True)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size
    =0.2, random_state=42)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)
print("Predictions:", y_pred)
```

Listing 2: Example: Logistic Regression for Titanic Dataset

In this example, we predict whether Titanic passengers survived based on features like age, gender, and class. Logistic regression is chosen because it predicts the probability of survival.

3.6 When to Use Logistic Regression?

Logistic regression is most useful when:

- The target variable is binary (e.g., survived/not survived, disease/no disease).
- We are interested in predicting the probability of an event.
- We want a simple, interpretable model.

It is especially effective in binary classification problems where the relationship between features and target is not strictly linear, but the decision boundary can still be approximated by a linear combination of features.

3.7 Additional Datasets for Logistic Regression

Some additional datasets that are commonly used for logistic regression tasks include:

- **Heart Disease Dataset** [3]: Used to predict the presence of heart disease based on patient data.
- **Adult Income Dataset** [5]: Classifies individuals' income as greater than or less than $50K.

4 Bias-Variance Trade-off

4.1 What is the Bias-Variance Trade-off?

The **bias-variance trade-off** is a fundamental concept in machine learning that deals with a model's performance on unseen data. The goal is to find the right balance between two sources of error:

- **Bias**: The error introduced by approximating a real-world problem with a simplified model. High bias can lead to underfitting, where the model is too simple to capture the patterns in the data.
- **Variance**: The error introduced by the model's sensitivity to small fluctuations in the training data. High variance can lead to overfitting, where the model captures noise in the training data rather than the underlying pattern.

4.2 Mathematical Understanding of Bias and Variance

Formally, the expected error of a model can be decomposed as:

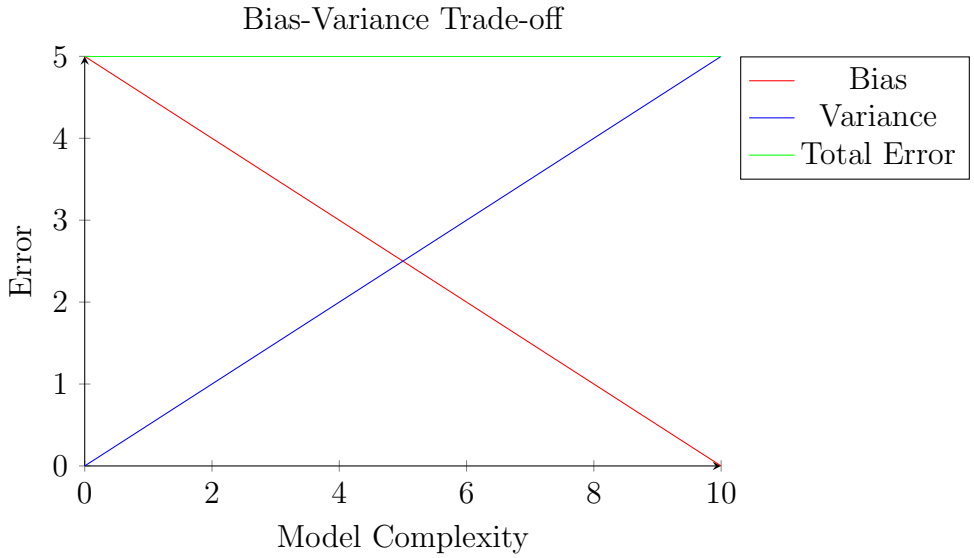
$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Where:

- **Bias** measures how far the model's predictions are from the true target values on average.
- **Variance** measures the variability of the model predictions for different training sets.
- **Irreducible Error** refers to the noise inherent in the data that cannot be reduced by any model.

The bias-variance trade-off implies that increasing model complexity reduces bias but increases variance, and vice versa. The goal is to minimize both to achieve the lowest possible error.

4.3 Graphical Representation of Bias-Variance Trade-off



This graph shows how error changes with model complexity. As complexity increases, bias decreases but variance increases. The total error is minimized at an optimal point where the model is neither too simple nor too complex.

4.4 Example of Bias-Variance Trade-off

Consider the case of fitting a polynomial to a dataset. If the polynomial degree is too low (e.g., a straight line), the model will have high bias and underfit the data. If the polynomial degree is too high, the model will fit the noise in the data, resulting in high variance and overfitting.

In practice, regularization techniques (discussed next) help to control the model complexity and balance the bias-variance trade-off.

5 Regularization

5.1 What is Regularization?

Regularization is a technique used to prevent overfitting by penalizing the complexity of the model. It introduces additional constraints to the model to ensure that it generalizes well to unseen data. There are two common types of regularization:

- **L1 Regularization (Lasso)**: Adds a penalty proportional to the absolute values of the model's coefficients. It can shrink some coefficients to zero, effectively performing feature selection.
- **L2 Regularization (Ridge)**: Adds a penalty proportional to the square of the model's coefficients. It tends to shrink the coefficients but usually keeps all of them in the model.

The regularized loss function for linear regression with L2 regularization (Ridge) is:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

Where:

- λ is the regularization parameter, controlling the strength of the penalty. A higher value of λ reduces model complexity but can also increase bias.
- θ_j are the model's parameters (coefficients).

5.2 Example: Regularized Logistic Regression with Breast Cancer Dataset

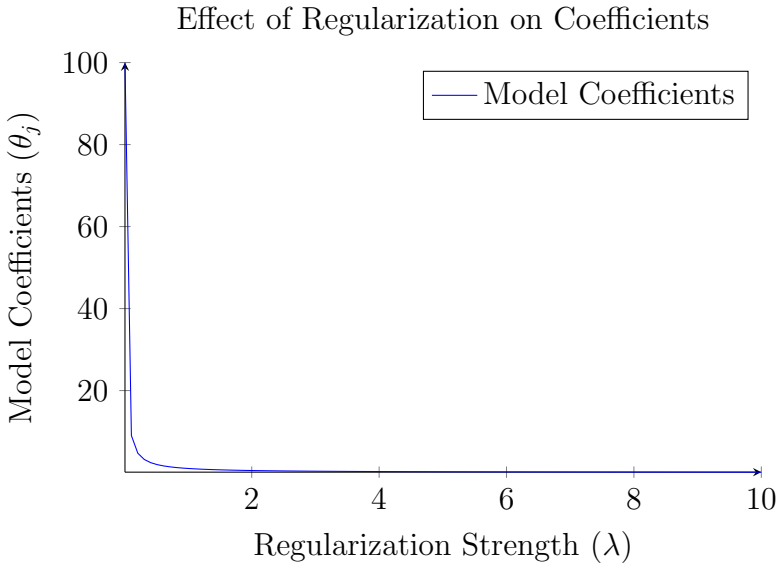
We apply regularization to logistic regression using the Breast Cancer dataset [6] to predict whether a tumor is benign or malignant.

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train regularized logistic regression (L2 regularization)
model = LogisticRegression(penalty='l2', C=1.0)  # C is the inverse of
    regularization strength
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)
print("Predictions:", y_pred)
```

Listing 3: Example: Regularized Logistic Regression for Breast Cancer Dataset

In this case, we use L2 regularization to prevent overfitting, which forces the model to generalize better to unseen data.

5.3 Visualizing the Effect of Regularization



This graph shows how the strength of regularization (λ) affects the size of the model's coefficients. As λ increases, the coefficients shrink, reducing the model's complexity and preventing overfitting.

6 Conclusion

In this week, we explored essential machine learning concepts such as linear classifiers, logistic regression, the bias-variance trade-off, and regularization. These techniques are crucial for building models that generalize well to unseen data. By understanding when and why to apply these methods, we can solve a variety of classification problems, avoid overfitting, and ensure that our models make accurate predictions. These tools form the backbone of modern machine learning and are applicable across a wide range of industries and domains.

References

[1] Ronald Aylmer Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[2] Kaggle. Titanic: Machine learning from disaster. `https://www.kaggle.com/c/titanic/data`, 2012.

[3] UCI Machine Learning Repository. Heart disease dataset. `https://archive.ics.uci.edu/ml/datasets/Heart+Disease`, 1988.

[4] UCI Machine Learning Repository. Pima indians diabetes database. `https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database`, 1988.

[5] UCI Machine Learning Repository. Adult income dataset. `https://archive.ics.uci.edu/ml/datasets/adult`, 1996.

[6] William H Wolberg and Olvi L Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, 87(23):9193–9196, 1990.

A Common Statistical Tests and Measurement Scores

Understanding standard values in basic statistical analysis is crucial. Below are some of the most commonly used statistical tests along with their standard measurement scores and interpretations.

A.1 T-Test (Student's T-Test)

Purpose: Compares the means of two groups.
P-value: A probability that measures evidence against the null hypothesis.

- P-value < 0.05: Reject the null hypothesis (statistically significant difference).
- P-value ≥ 0.05: Fail to reject the null hypothesis.

T-statistic: Measures the size of the difference relative to the variation in the sample data.

A.2 ANOVA (Analysis of Variance)

Purpose: Compares the means of three or more groups.
P-value: Same as with the T-test.
F-statistic: Ratio of variances, comparing between-group variance to within-group variance.

- Higher F-statistic: More variance between groups than within.

A.3 Chi-Square Test

Purpose: Tests the association between categorical variables.
P-value: Same interpretation as other tests.
Chi-square statistic: Measures the discrepancy between observed and expected frequencies.

- Larger Chi-square statistic: Greater difference between expected and observed frequencies.

A.4 Linear Regression

Purpose: Models the relationship between a dependent and one or more independent variables.

- **P-value for coefficients**: Tests significance of independent variables in predicting the dependent variable.

- **R-Squared (R^2)**: Proportion of variance in the dependent variable explained by the model.
  - R^2 close to 1: The model explains most of the variance.
  - R^2 close to 0: The model explains little variance.
- **Adjusted R-Squared**: Adjusts R^2 for the number of predictors, higher is better.

A.5 Logistic Regression

Purpose: Predicts a binary outcome.

- **P-value for coefficients**: Similar to linear regression.
- **Odds Ratio (OR)**: Represents the change in odds for a unit increase in the predictor.
  - OR > 1: Predictor increases the odds of the event.
  - OR < 1: Predictor decreases the odds of the event.
- **Pseudo R-Squared (McFadden's R^2)**: Measures model fit (lower than traditional R^2).

A.6 Correlation Coefficient (Pearson/Spearman)

Purpose: Measures strength and direction of the relationship between two variables.

- **Correlation Coefficient (r)**: Ranges from -1 to 1.
  - r = 1: Perfect positive correlation.
  - r = -1: Perfect negative correlation.
  - r = 0: No correlation.
- **P-value**: Indicates statistical significance of the correlation.

A.7 Mann-Whitney U Test

Purpose: Compares differences between two independent groups (non-parametric).

- **P-value**: Same interpretation as other tests.
- **U statistic**: Rank-based difference between groups.

A.8 Wilcoxon Signed-Rank Test

Purpose: Compares two related samples or repeated measurements.

- **P-value**: Same interpretation as other tests.
- **W-statistic**: Sum of the signed ranks of differences between matched pairs.

A.9 Kruskal-Wallis Test

Purpose: Non-parametric test comparing three or more groups.

- **P-value**: Same as for other tests.
- **H-statistic**: Measures rank-based differences across groups.

A.10 Cox Proportional Hazards Model

Purpose: Examines the association between survival time and one or more predictors.

- **P-value for coefficients**: Tests the significance of predictors for survival time.
- **Hazard Ratio (HR)**: Represents the effect of a predictor on the hazard.
  - HR > 1: Increased hazard.
  - HR < 1: Decreased hazard.

A.11 Fisher's Exact Test

Purpose: Tests the association between two categorical variables in small sample sizes.

- **P-value**: If P-value < 0.05, there is a significant association between variables.

Key Measurement Scores:

- **P-value**: Typically, P-value < 0.05 indicates statistical significance.
- **R-Squared ($\mathbf{R^2}$)**: Higher is better for model fit.
- **Adjusted R-Squared**: Adjusts for the number of predictors, higher is better.
- **Odds Ratio (OR)**: OR > 1 indicates increased likelihood of the outcome.
- **Correlation Coefficient (r)**: Values closer to -1 or 1 show stronger correlation.