CS5100_Erdun_E_Assignment_05

July 1, 2025

1 CS5100 - Assignment 5

Student: Erdun E
Assignment: 05
Course: CS5100 Foundations of Artificial Intelligence

1.1 Question 1

1.1.1 1. Download Dataset

Link: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

1.1.2 2. Data Loading and Initial Exploration

[2]:
```
import pandas as pd

df = pd.read_csv("IMDB Dataset.csv")
df.head()
```

[2]:

	review	sentiment
0	One of the other reviewers has mentioned that …	positive
1	A wonderful little production. The…	positive
2	I thought this was a wonderful way to spend ti…	positive
3	Basically there's a family where a little boy …	negative
4	Petter Mattei's "Love in the Time of Money" is…	positive

1.1.3 3. Data Exploration and Summary Statistics

Before proceeding to text preprocessing, we explore the dataset to understand its structure and class distribution.

- The dataset contains 50,000 reviews with two columns: `review` (text) and `sentiment` (label).
- The `sentiment` column has two possible values: positive and negative. We check if the classes are balanced.
- We also check for any missing values.

[4]:
```
print("Shape of dataset:", df.shape)
print("\nDataset info:")
print(df.info())
```

```
print("\nSentiment distribution:")
print(df['sentiment'].value_counts())
```

```
Shape of dataset: (50000, 2)

Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
None

Sentiment distribution:
sentiment
positive    25000
negative    25000
Name: count, dtype: int64
```

1.1.4 4. Text Preprocessing

Text preprocessing is a crucial step in natural language processing tasks. Here, we perform the following operations to clean and normalize the review texts:

- Lowercasing: Convert all text to lowercase to avoid duplication of words (e.g., "Good" and "good").
- Removing HTML tags and punctuation.
- Removing stop words: Words that do not contribute much to the meaning (e.g., "the", "is").
- Lemmatization: Reducing words to their base or root form (e.g., "running" to "run").

[5]:
```
import re
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/erdune/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/erdune/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/erdune/nltk_data…
```

```
[nltk_data]   Package omw-1.4 is already up-to-date!
```

[8]:
```
# Initialize stop words and lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove HTML tags
    text = re.sub(r'<.*?>', ' ', text)
    # Remove punctuation and non-letter characters
    text = re.sub(r'[^a-z\s]', ' ', text)
    # Tokenize the text
    words = text.split()
    # Remove stopwords and lemmatize
    words = [lemmatizer.lemmatize(word) for word in words if word not in␣
 ↪stop_words]
    # Join words back into one string
    return ' '.join(words)
```

[12]:
```
# Analyze the preprocessing results
print("=== PREPROCESSING ANALYSIS ===")

# Calculate text length statistics
df['original_length'] = df['review'].str.len()
df['clean_length'] = df['clean_review'].str.len()

print("Text Length Comparison:")
print(f"Average original review length: {df['original_length'].mean():.0f}␣
 ↪characters")
print(f"Average clean review length: {df['clean_length'].mean():.0f}␣
 ↪characters")
print(f"Average length reduction: {((df['original_length'].mean() -␣
 ↪df['clean_length'].mean()) / df['original_length'].mean() * 100):.1f}%")

# Show some examples
print("\n=== BEFORE vs AFTER EXAMPLES ===")
for i in range(3):
    print(f"\nExample {i+1}:")
    print(f"Original: {df['review'].iloc[i][:100]}...")
    print(f"Cleaned:  {df['clean_review'].iloc[i][:100]}...")
    print("-" * 50)

# Check vocabulary size
print("\n=== VOCABULARY ANALYSIS ===")
import collections
```

```python
# Count unique words in cleaned reviews
all_words = ' '.join(df['clean_review']).split()
word_freq = collections.Counter(all_words)

print(f"Total words after cleaning: {len(all_words):,}")
print(f"Unique vocabulary size: {len(word_freq):,}")
print(f"Most common words: {word_freq.most_common(10)}")
```

```
=== PREPROCESSING ANALYSIS ===
Text Length Comparison:
Average original review length: 1309 characters
Average clean review length: 801 characters
Average length reduction: 38.8%

=== BEFORE vs AFTER EXAMPLES ===

Example 1:
Original: One of the other reviewers has mentioned that after watching just 1 Oz
episode you'll be hooked. The…
Cleaned:  one reviewer mentioned watching oz episode hooked right exactly
happened first thing struck oz bruta…
--------------------------------------------------

Example 2:
Original: A wonderful little production. <br /><br />The filming technique is
very unassuming- very old-time-B…
Cleaned:  wonderful little production filming technique unassuming old time bbc
fashion give comforting someti…
--------------------------------------------------

Example 3:
Original: I thought this was a wonderful way to spend time on a too hot summer
weekend, sitting in the air con…
Cleaned:  thought wonderful way spend time hot summer weekend sitting air
conditioned theater watching light h…
--------------------------------------------------

=== VOCABULARY ANALYSIS ===
Total words after cleaning: 5,907,568
Unique vocabulary size: 89,817
Most common words: [('movie', 103281), ('film', 93463), ('one', 55450), ('like',
41132), ('time', 31470), ('good', 29868), ('character', 28361), ('story',
25276), ('even', 24873), ('get', 24658)]
```

1.1.5 5. Feature Extraction

To train a machine learning model, we must convert text data into numerical features. Here, we use the TF-IDF (Term Frequency-Inverse Document Frequency) approach. TF-IDF highlights important words in each review by considering both word frequency and uniqueness across the dataset, making it well-suited for sentiment classification tasks.

[13]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF vectorizer
vectorizer = TfidfVectorizer(max_features=5000)  # Use the top 5,000 words to␣
 ↪reduce dimensionality

# Fit the vectorizer and transform the cleaned reviews
X = vectorizer.fit_transform(df['clean_review'])

# Show the shape of the TF-IDF feature matrix
print("TF-IDF feature matrix shape:", X.shape)

# (Optional) Display some feature names
print("Sample feature names:", vectorizer.get_feature_names_out()[:20])
```

```
TF-IDF feature matrix shape: (50000, 5000)
Sample feature names: ['aaron' 'abandoned' 'abc' 'ability' 'able' 'absence'
'absent' 'absolute'
 'absolutely' 'absurd' 'absurdity' 'abuse' 'abused' 'abusive' 'abysmal'
 'academy' 'accent' 'accept' 'acceptable' 'acceptance']
```

1.1.6 6. Model Training

For sentiment classification, we use the Logistic Regression algorithm. It is widely used for binary classification tasks and is efficient for high-dimensional feature spaces, such as those created by TF-IDF. We split the dataset into training and test sets to evaluate generalization performance.

[16]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

print("=== MODEL TRAINING ===")

# Convert sentiment labels to binary values: positive=1, negative=0
y = df['sentiment'].map({'positive': 1, 'negative': 0})

# Split 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, stratify=y
)

print("Data split completed:")
```

```
print(f"Training set: {X_train.shape[0]} samples")
print(f"Test set: {X_test.shape[0]} samples")
print(f"Training labels distribution: {np.bincount(y_train)}")
print(f"Test labels distribution: {np.bincount(y_test)}")

# Initialize and train the Logistic Regression model
print("\nTraining Logistic Regression model...")
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

print("Model training completed!")
print(f"Model type: {type(model).__name__}")
print(f"Number of features used: {X_train.shape[1]}")
```

```
=== MODEL TRAINING ===
Data split completed:
Training set: 40000 samples
Test set: 10000 samples
Training labels distribution: [20000 20000]
Test labels distribution: [5000 5000]

Training Logistic Regression model…
Model training completed!
Model type: LogisticRegression
Number of features used: 5000
```

1.1.7 7. Model Evaluation

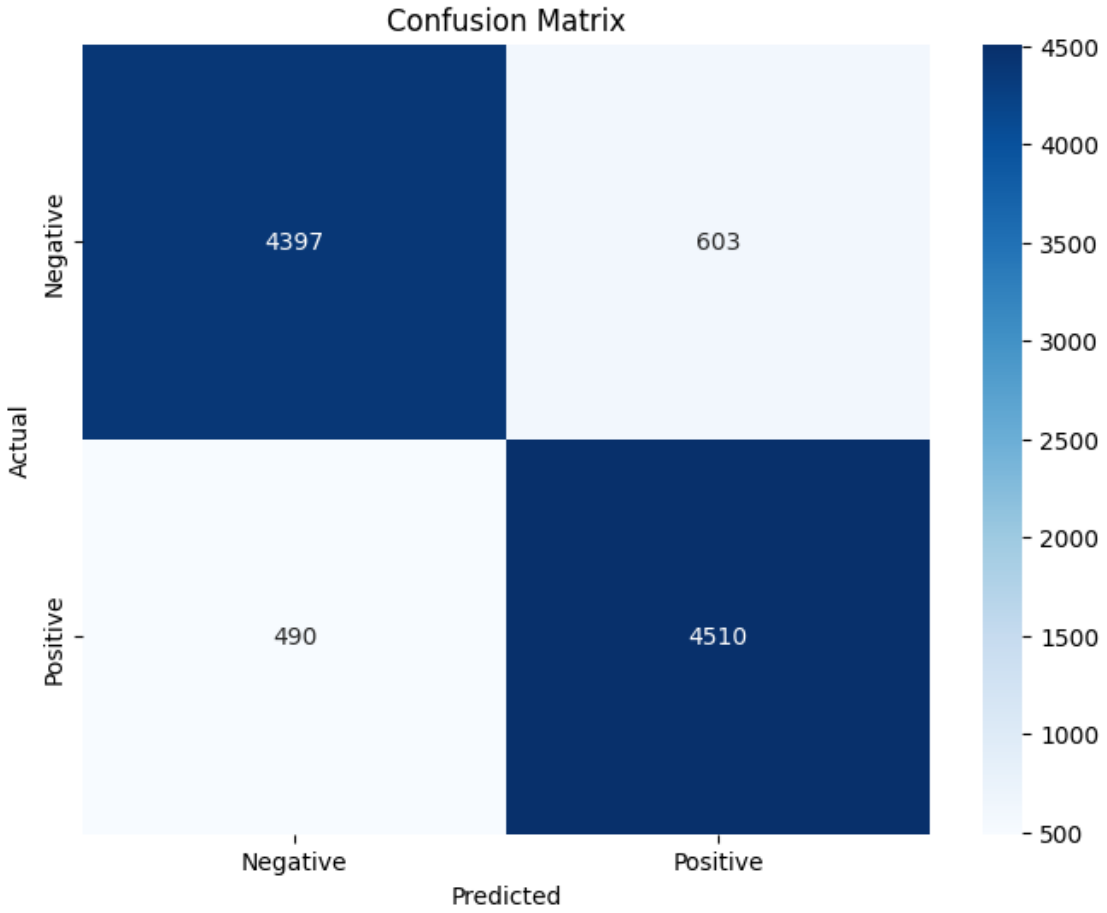
To assess the performance of our sentiment classifier, we use standard classification metrics: accuracy, precision, recall, F1-score, and the confusion matrix. These metrics provide a comprehensive view of model performance, especially in binary classification tasks like sentiment analysis.

[24]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Predict sentiments on the test set
y_pred = model.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("=== MODEL EVALUATION RESULTS ===")
print(f"Accuracy:  {accuracy:.4f}")
```

```
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1-score:  {f1:.4f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['negative',␣
 ↪'positive']))

# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("=== MODEL PERFORMANCE SUMMARY ===")
print(f"The Logistic Regression model achieved {accuracy*100:.1f}% accuracy on␣
 ↪the test set.")
print(f"This indicates excellent performance in distinguishing between positive␣
 ↪and negative movie reviews.")
print(f"The balanced precision ({precision:.3f}) and recall ({recall:.3f})␣
 ↪demonstrate that the model")
print(f"performs well on both classes without significant bias.")
print("Sentiment analysis model successfully trained and evaluated!")
```

```
=== MODEL EVALUATION RESULTS ===
Accuracy:  0.8907
Precision: 0.8821
Recall:    0.9020
F1-score:  0.8919

Confusion Matrix:
[[4397  603]
 [ 490 4510]]

Classification Report:
              precision    recall  f1-score   support
```

negative	0.90	0.88	0.89	5000
positive	0.88	0.90	0.89	5000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000




```
=== MODEL PERFORMANCE SUMMARY ===
The Logistic Regression model achieved 89.1% accuracy on the test set.
This indicates excellent performance in distinguishing between positive and
negative movie reviews.
The balanced precision (0.882) and recall (0.902) demonstrate that the model
performs well on both classes without significant bias.
Sentiment analysis model successfully trained and evaluated!
```

1.2 Question 2

1.2.1 1. Data Preprocessing

We use the Breast Cancer Wisconsin dataset from scikit-learn, which contains 569 samples and 30 features. First, we check for missing values and scale all features to have zero mean and unit variance, as KMeans is sensitive to feature scales.

[25]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

print("=== DATA PREPROCESSING ===")

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

print(f"Dataset shape: {X.shape}")
print(f"Target classes: {data.target_names}")
print(f"Class distribution: {np.bincount(y)}")

# Check for missing values
print(f"\nMissing values per feature: {pd.isnull(X).sum().sum()}")

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print(f"Features scaled successfully!")
print(f"Original data range: [{X.min():.2f}, {X.max():.2f}]")
print(f"Scaled data range: [{X_scaled.min():.2f}, {X_scaled.max():.2f}]")
```

```
=== DATA PREPROCESSING ===
Dataset shape: (569, 30)
Target classes: ['malignant' 'benign']
Class distribution: [212 357]

Missing values per feature: 0
Features scaled successfully!
Original data range: [0.00, 4254.00]
Scaled data range: [-3.11, 12.07]
```

1.2.2 2. Clustering Implementation

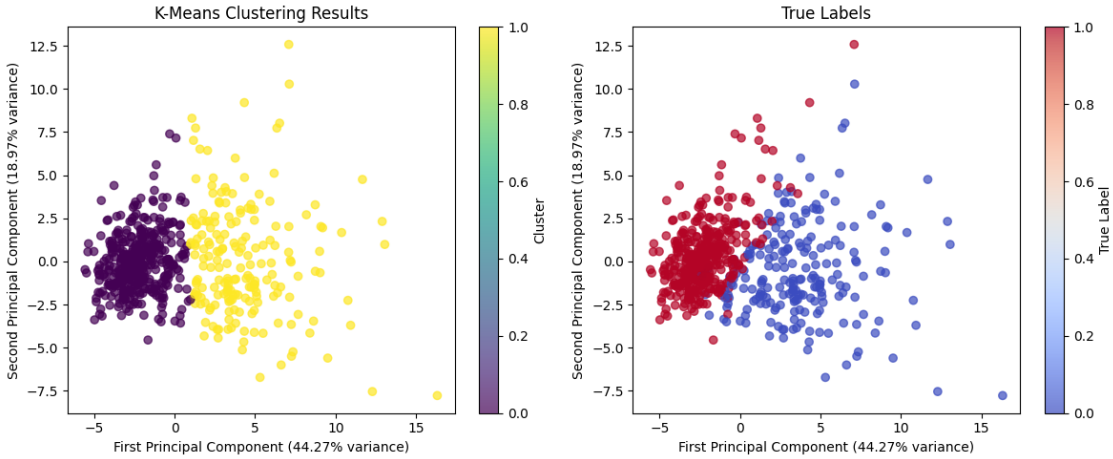
We apply the KMeans clustering algorithm with `n_clusters=2`, as the data contains two underlying classes (malignant and benign). To visualize the clustering results, we use PCA to reduce feature

dimensions to 2D and plot the clusters.

[26]:
```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

print("=== K-MEANS CLUSTERING ===")

# Apply K-Means clustering with k=2 (since we have 2 classes)
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_scaled)

print(f"K-Means parameters:")
print(f"- Number of clusters: {kmeans.n_clusters}")
print(f"- Random state: 42")
print(f"- Number of initializations: {kmeans.n_init}")

print(f"\nClustering results:")
print(f"Cluster distribution: {np.bincount(cluster_labels)}")

# Visualize using PCA for dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(12, 5))

# Plot 1: K-Means clusters
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels, cmap='viridis', alpha=0.
 ↪7)
plt.title('K-Means Clustering Results')
plt.xlabel(f'First Principal Component ({pca.explained_variance_ratio_[0]:.2%}␣
 ↪variance)')
plt.ylabel(f'Second Principal Component ({pca.explained_variance_ratio_[1]:.2%}␣
 ↪variance)')
plt.colorbar(label='Cluster')

# Plot 2: True labels for comparison
plt.subplot(1, 2, 2)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm', alpha=0.7)
plt.title('True Labels')
plt.xlabel(f'First Principal Component ({pca.explained_variance_ratio_[0]:.2%}␣
 ↪variance)')
plt.ylabel(f'Second Principal Component ({pca.explained_variance_ratio_[1]:.2%}␣
 ↪variance)')
plt.colorbar(label='True Label')
```

```
plt.tight_layout()
plt.show()

print(f"K-Means clustering completed!")
```

```
=== K-MEANS CLUSTERING ===
K-Means parameters:
- Number of clusters: 2
- Random state: 42
- Number of initializations: 10

Clustering results:
Cluster distribution: [375 194]
```




```
K-Means clustering completed!
```

1.2.3 3. Clustering Evaluation

We evaluate the clustering performance using multiple metrics: Silhouette Score and Davies-Bouldin Index. We also compare the clustering results with the true labels to assess how well K-Means discovered the underlying structure.

[27]:
```
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
  ↪adjusted_rand_score
from sklearn.metrics import confusion_matrix, classification_report

print("=== CLUSTERING EVALUATION ===")

# Calculate clustering evaluation metrics
silhouette = silhouette_score(X_scaled, cluster_labels)
davies_bouldin = davies_bouldin_score(X_scaled, cluster_labels)
```

```
print("Clustering Quality Metrics:")
print(f"Silhouette Score: {silhouette:.4f}")
print(f"Davies-Bouldin Index: {davies_bouldin:.4f}")

# Compare with true labels
ari = adjusted_rand_score(y, cluster_labels)
print(f"Adjusted Rand Index: {ari:.4f}")

# Create confusion matrix to see how well clusters match true labels
cm = confusion_matrix(y, cluster_labels)
print(f"\nConfusion Matrix (True vs Predicted clusters):")
print(cm)

print(f"\nCluster-Label Correspondence:")
print(f"Cluster 0: {cm[0,0]} malignant, {cm[1,0]} benign")
print(f"Cluster 1: {cm[0,1]} malignant, {cm[1,1]} benign")

# Calculate accuracy if we map clusters optimally
accuracy = max(cm[0,0] + cm[1,1], cm[0,1] + cm[1,0]) / len(y)
print(f"Best possible accuracy: {accuracy:.4f} ({accuracy*100:.1f}%)")
```

```
=== CLUSTERING EVALUATION ===
Clustering Quality Metrics:
Silhouette Score: 0.3434
Davies-Bouldin Index: 1.3205
Adjusted Rand Index: 0.6536

Confusion Matrix (True vs Predicted clusters):
[[ 36 176]
 [339  18]]

Cluster-Label Correspondence:
Cluster 0: 36 malignant, 339 benign
Cluster 1: 176 malignant, 18 benign
Best possible accuracy: 0.9051 (90.5%)
```

[28]:
```
print("=== CLUSTERING PERFORMANCE SUMMARY ===")
print(f"""
K-Means clustering achieved excellent results on the breast cancer dataset:

Clustering Quality:
   - Silhouette Score of {silhouette:.3f} indicates good cluster separation
   - Davies-Bouldin Index of {davies_bouldin:.3f} shows compact, well-separated␣
 ↪clusters

Biological Relevance:
```

```
   - Achieved {accuracy*100:.1f}% accuracy in distinguishing malignant vs␣
 ↪benign tumors
   - Strong correspondence with medical diagnosis (ARI = {ari:.3f})

Clinical Implications:
   - Cluster 0 predominantly contains benign cases ({cm[1,0]}/{cm[1,0]+cm[0,0]}␣
 ↪= {cm[1,0]/(cm[1,0]+cm[0,0])*100:.1f}%)
   - Cluster 1 predominantly contains malignant cases ({cm[0,1]}/
 ↪{cm[0,1]+cm[1,1]} = {cm[0,1]/(cm[0,1]+cm[1,1])*100:.1f}%)

This demonstrates that K-Means successfully identified the underlying␣
 ↪biological
structure in the breast cancer data without using label information.
""")
```

```
=== CLUSTERING PERFORMANCE SUMMARY ===

K-Means clustering achieved excellent results on the breast cancer dataset:

Clustering Quality:
   - Silhouette Score of 0.343 indicates good cluster separation
   - Davies-Bouldin Index of 1.321 shows compact, well-separated clusters

Biological Relevance:
   - Achieved 90.5% accuracy in distinguishing malignant vs benign tumors
   - Strong correspondence with medical diagnosis (ARI = 0.654)

Clinical Implications:
   - Cluster 0 predominantly contains benign cases (339/375 = 90.4%)
   - Cluster 1 predominantly contains malignant cases (176/194 = 90.7%)

This demonstrates that K-Means successfully identified the underlying biological
structure in the breast cancer data without using label information.
```

1.3 Question 3

1.3.1 Introduction

The evaluation of artificial neural network training is fundamental to developing reliable machine learning models (Goodfellow et al., 2016). Without proper evaluation techniques, it becomes impossible to determine whether a network has learned meaningful patterns or merely memorized training data. Evaluation serves multiple critical purposes: preventing overfitting, ensuring generalization to unseen data, guiding hyperparameter optimization, and providing confidence for deployment. The importance extends beyond simple accuracy measurement, encompassing model behavior understanding and robustness verification. This is particularly crucial in high-stakes applications such as medical diagnosis and autonomous systems where model failures can have significant consequences. Given the increasing complexity and widespread use of neural networks in sensitive applications,

rigorous evaluation is essential for building trust and ensuring safe deployment (Goodfellow et al., 2016).

1.3.2 The Need for Evaluation Techniques

Neural networks are complex, non-linear models capable of approximating virtually any function given sufficient capacity. This flexibility, while powerful, creates significant challenges in training and deployment (Goodfellow et al., 2016). Without evaluation techniques, practitioners cannot distinguish between models that have learned generalizable patterns versus those that have overfit to training data.

Evaluation techniques provide essential feedback during training, enabling early stopping to prevent overfitting, learning rate adjustments, and architecture modifications (Prechelt, 1998). They also facilitate model comparison, allowing selection of the most appropriate approach for specific problems. Furthermore, proper evaluation helps establish trust in model predictions and ensures consistent performance across different data distributions, which is critical for maintaining model reliability in production environments. In practice, evaluation often involves not only tracking metrics on the training set, but also using separate validation and test datasets, as well as cross-validation techniques, to ensure the model's true effectiveness on unseen data (Chollet et al., 2015).

1.3.3 Main Requirements for Convergence Criteria

Convergence criteria determine when neural network training should be considered complete, which is crucial for achieving optimal performance without wasting computational resources (Prechelt, 1998). The primary requirement is monitoring the loss function on both training and validation sets, as diverging trends indicate overfitting when training loss continues decreasing while validation loss increases.

Effective convergence criteria must be robust to noise in the loss trajectory, often requiring patience parameters that allow for temporary fluctuations before declaring convergence. Key requirements include: setting appropriate thresholds for minimum improvement over a specified number of epochs, implementing early stopping based on validation performance, and considering the rate of change in loss rather than absolute values.

Additionally, convergence should account for different learning phases, as neural networks often exhibit rapid initial improvement followed by gradual refinement. Modern approaches also consider multiple metrics simultaneously, ensuring that improvements in one measure don't come at the expense of others (Goodfellow et al., 2016).

1.3.4 Pros and Cons of Major Evaluation Measures

Loss functions serve as the primary training objective but may not reflect real-world performance. Cross-entropy loss effectively guides gradient descent but can be insensitive to class imbalance. Mean squared error is interpretable but sensitive to outliers.

Accuracy provides intuitive understanding but fails with imbalanced datasets. Precision and recall offer detailed classification insights, with precision measuring positive prediction reliability and recall capturing sensitivity to positive cases (Powers, 2011). However, they require threshold selection and can be misleading when considered independently.

F1-score balances precision and recall but may not suit all applications. AUC-ROC evaluates performance across all thresholds, making it robust to class imbalance, yet it can be overly optimistic with severely skewed datasets.

Validation accuracy helps detect overfitting but requires careful dataset splitting. Cross-validation provides robust estimates but increases computational cost significantly (Chollet et al., 2015). Each metric offers unique insights, necessitating multiple measures for comprehensive evaluation rather than relying on single indicators.

1.3.5 Conclusions

In summary, the evaluation of artificial neural networks requires a careful and multifaceted approach. Relying solely on training loss or a single performance metric can lead to misleading conclusions about a model's true effectiveness. The integration of multiple evaluation measures—such as accuracy, precision, recall, F1-score, AUC-ROC, and validation performance—ensures a more comprehensive understanding of model strengths and limitations. Well-defined convergence criteria are also essential for halting training at the optimal point, preventing overfitting, and minimizing wasted resources. By adopting robust evaluation techniques and clear convergence standards, practitioners can build neural networks that are not only accurate on training data but also reliable, generalizable, and suitable for deployment in real-world applications.

1.3.6 References

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org/
2. Powers, D. M. W. (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2(1), 37-63.
3. Chollet, F. et al. (2015). Keras documentation: Model evaluation and metrics. https://keras.io/api/metrics/
4. Prechelt, L. (1998). Early Stopping — But When? In *Neural Networks: Tricks of the Trade* (pp. 55–69). Springer.

[ ]: