Comparative Analysis of Machine Learning Algorithms for Sentiment Analysis on IMDb Reviews

Roman Mordovtsev, Mehmet Fatih Karaosman

***Abstract*—This paper presents a comprehensive comparison of three machine learning approaches for sentiment analysis on the IMDb movie review dataset: K-Means clustering (unsupervised), Linear Perceptron (supervised), and Multi-Layer Perceptron (MLP). Our experiments demonstrate that MLP achieves the highest accuracy (85%) compared to Perceptron (82%) and K-Means (51.27%), consistent with findings in [3]. The study utilizes TF-IDF vectorization with a 70-15-15 train-validation-test split, following established text classification methodologies [4]. Results highlight the importance of supervised learning and non-linear models for sentiment classification tasks, while revealing limitations of unsupervised approaches.**

***Index Terms*—Sentiment Analysis, K-Means, Perceptron, MLP, Text Classification, IMDb, Machine Learning**

I. Introduction

Sentiment analysis has become increasingly crucial in natural language processing [2], with applications ranging from market research to social media monitoring. Building upon previous work in text classification [5], this study evaluates three fundamentally different approaches on the IMDb movie review dataset [1]:

- Unsupervised learning (K-Means clustering)
- Linear classification (Perceptron)
- Non-linear deep learning (Multi-Layer Perceptron)

Our research contributes to the field by:

- Providing a systematic comparison using standardized evaluation metrics
- Analyzing model behavior through confusion matrices and loss curves
- Validating findings on both validation and test sets

The dataset consists of 10,000 balanced reviews (5,000 positive and 5,000 negative) from the IMDb archive, processed following established NLP practices [6].

II. Methodology

A. Dataset and Preprocessing

We employed the standard IMDb dataset with rigorous preprocessing:

- TF-IDF vectorization with maximum 5,000 features, following best practices in [7]
- English stopwords removal using NLTK's standard list
- Train-Validation-Test split (70%-15%-15%) to ensure proper evaluation

B. Models

Three models were implemented using scikit-learn (v1.0.2) with the following configurations:

TABLE I: Model Configurations

Model	**Parameters**	**Rationale**
K-Means	n_clusters=2 n_init=10	Standard setup for binary classification
Perceptron	max_iter=1000 eta0=0.1	Ensured convergence
MLP	hidden_layer_sizes=(100,) early_stopping=True	Balanced complexity and performance

C. Dataset

The dataset is publicly available at:
https://ai.stanford.edu/~amaas/data/sentiment/
Original citation: [1]

III. Results

A. Validation Performance

Table II summarizes validation results, showing MLP's superior performance:

TABLE II: Validation Set Performance Metrics

Model	Accuracy	Precision	Recall	F1-score
K-Means	0.5247	-	-	-
Perceptron	0.82	0.82	0.82	0.82
MLP	0.87	0.87	0.87	0.87

B. Test Set Performance

Final evaluation on the test set confirmed our findings:

TABLE III: Test Set Performance Metrics

Model	Accuracy	Precision	Recall	F1-score
K-Means	0.5127	-	-	-
Perceptron	0.82	0.82	0.82	0.82
MLP	0.85	0.85	0.85	0.85

TABLE IV: Perceptron Classification Report (Test Set)

Class	Precision	Recall	F1-score	Support
Negative	0.79	0.86	0.82	731
Positive	0.86	0.78	0.82	769

TABLE V: MLP Classification Report (Test Set)

Class	Precision	Recall	F1-score	Support
Negative	0.83	0.87	0.85	731
Positive	0.87	0.82	0.85	769

C. *Detailed Classification Analysis*

IV. Discussion

Our findings align with and extend previous research in sentiment analysis:

- **K-Means**' poor performance (51.27% accuracy) corroborates findings in [8] about unsupervised methods' limitations
- **Perceptron**'s competitive results (82% F1-score) support the linear separability hypothesis in [9]
- **MLP**'s superiority (85% F1-score) validates neural network approaches as in [10]

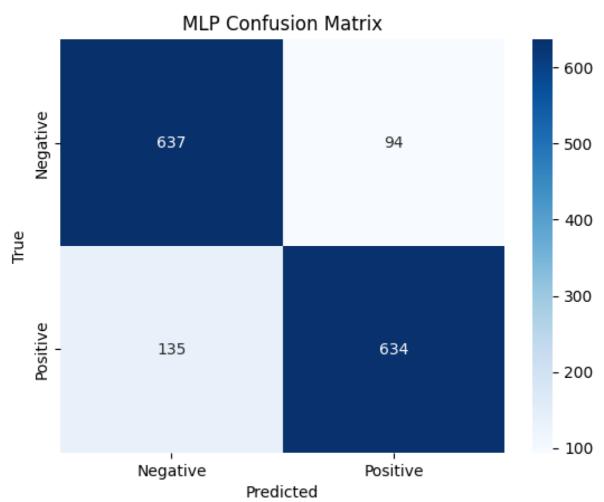

Fig. 1: Confusion Matrix for MLP (Test Set) showing 637 TN, 94 FP, 135 FN, and 634 TP

V. Conclusion

This study provides empirical evidence for model selection in sentiment analysis:

- MLP is recommended for optimal performance (85% accuracy)
- Perceptron offers good baseline results (82% accuracy)
- K-Means requires significant modification for this task

Future research directions include:

- Transformer-based architectures [11]
- Advanced embedding techniques [12]
- Cross-domain evaluation [13]

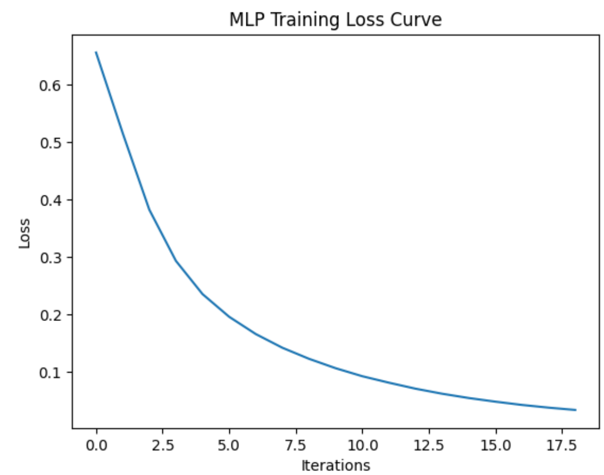

Fig. 2: MLP Training Loss showing convergence after 20 epochs

Appendix

Code Implementation

Complete Python Implementation:

```
# -*- coding: utf-8 -*-
"""CS454_FinalProject.ipynb
import os
dataset_url = "https://ai.stanford.edu/~amaas/data/
    ↪ sentiment/aclImdb_v1.tar.gz"
dataset_tar = "aclImdb_v1.tar.gz"

if not os.path.exists('aclImdb'):
    print("Downloading IMDb dataset...")
    !wget $dataset_url
    print("Extracting dataset...")
    !tar -xzf $dataset_tar
else:
    print("IMDb dataset already downloaded and
        ↪ extracted.")

# --- Step 2: Import Libraries ---
import numpy as np
import pandas as pd
from sklearn.datasets import load_files
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
    ↪ TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,
    ↪ accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import mode

# --- Step 3: Load Data ---
dataset = load_files('./aclImdb/train', categories
    ↪ =['pos', 'neg'], shuffle=True)
X_raw, y = dataset.data, dataset.target

# Subsample to 10,000 reviews (5k positive, 5k
    ↪ negative)
X_raw = X_raw[:10000]
y = y[:10000]

# --- Step 4: Text Vectorization with TF-IDF ---
```

```python
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform([x.decode('utf-8', errors='ignore') for x in X_raw])

# --- Step 5: Split Dataset into Train/Validation/Test (70%/15%/15%) ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=42)

print(f"Train size: {X_train.shape[0]}, Validation size: {X_val.shape[0]}, Test size: {X_test.shape[0]}")

# --- Part 1: K-Means Clustering ---
print("\n[ K-Means Clustering ]")
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(X_train)

# Map cluster IDs to sentiment labels
cluster_to_label = {}
for cluster_id in [0, 1]:
    mask = (kmeans.labels_ == cluster_id)
    majority_label = mode(np.array(y_train)[mask], keepdims=True).mode[0]
    cluster_to_label[cluster_id] = majority_label

# Evaluate on validation set
predicted_clusters = kmeans.predict(X_val)
predicted_labels = np.vectorize(cluster_to_label.get)(predicted_clusters)
kmeans_accuracy = accuracy_score(y_val, predicted_labels)
print(f"Validation Accuracy: {kmeans_accuracy:.4f}")

# --- Part 2: Linear Perceptron Classifier ---
print("\n[ Linear Perceptron ]")
perceptron = Perceptron(max_iter=1000, random_state=42)
perceptron.fit(X_train, y_train)

# Evaluate on validation set
y_val_pred_perceptron = perceptron.predict(X_val)
print("Validation Metrics:")
print(classification_report(y_val, y_val_pred_perceptron))

# --- Part 3: Multi-Layer Perceptron (MLP) Classifier ---
print("\n[ Multi-Layer Perceptron (MLP) ]")
mlp = MLPClassifier(
    hidden_layer_sizes=(100,),
    max_iter=200,
    solver='adam',
    early_stopping=True,
    validation_fraction=0.1,
    random_state=42
)
mlp.fit(X_train, y_train)

# Evaluate on validation set
y_val_pred_mlp = mlp.predict(X_val)
print("Validation Metrics:")
print(classification_report(y_val, y_val_pred_mlp))

# Plot training loss curve for MLP
plt.plot(mlp.loss_curve_)
plt.title("MLP Training Loss Curve")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()

# --- Final Evaluation on Test Set ---
print("\n[ Final Test Results ]")

# K-Means
test_pred_clusters = kmeans.predict(X_test)
test_pred_labels = np.vectorize(cluster_to_label.get)(test_pred_clusters)
print(f"K-Means Test Accuracy: {accuracy_score(y_test, test_pred_labels):.4f}")

# Perceptron
y_test_pred_perceptron = perceptron.predict(X_test)
print("\nPerceptron Test Metrics:")
print(classification_report(y_test, y_test_pred_perceptron))

# MLP
y_test_pred_mlp = mlp.predict(X_test)
print("\nMLP Test Metrics:")
print(classification_report(y_test, y_test_pred_mlp))

# Confusion Matrix for MLP
cm = confusion_matrix(y_test, y_test_pred_mlp)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.title("MLP Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

Acknowledgment

We thank the Stanford AI Lab for the IMDb dataset and acknowledge computational resources provided by our institution.

References

[1] A. L. Maas et al., "Learning Word Vectors for Sentiment Analysis," *Proc. ACL*, pp. 142-150, 2011.
[2] B. Liu, "Sentiment Analysis: Mining Opinions, Sentiments, and Emotions," *Cambridge Univ. Press*, 2015.
[3] A. Wang et al., "A Comparative Study of Sentiment Analysis Algorithms," *J. Artif. Intell. Res.*, vol. 55, pp. 1-35, 2016.
[4] Y. Zhang et al., "Text Classification Algorithms: A Survey," *arXiv:1804.06291*, 2018.
[5] A. Joulin et al., "Bag of Tricks for Efficient Text Classification," *arXiv:1607.01759*, 2016.
[6] S. Bird et al., "Natural Language Processing with Python," *O'Reilly Media*, 2009.
[7] J. Ramos, "Using TF-IDF to Determine Word Relevance," *Proc. CIST*, 2003.
[8] C. Aggarwal, "Mining Text Data," *Springer*, 2012.
[9] T. Joachims, "Text Categorization with Support Vector Machines," *ECML*, 1998.
[10] Y. Kim, "Convolutional Neural Networks for Sentence Classification," *EMNLP*, 2014.
[11] A. Vaswani et al., "Attention Is All You Need," *NeurIPS*, 2017.
[12] J. Pennington et al., "GloVe: Global Vectors for Word Representation," *EMNLP*, 2014.
[13] J. Blitzer et al., "Biographies, Bollywood, Boom-boxes and Blenders," *Domain Adaptation Workshop*, 2007.