Text Classification: Traditional ML vs Deep Learning
A Mathematical and Computational Analysis

Om Choksi

November 18, 2025

Abstract

This report presents a rigorous mathematical and computational analysis of text classification algorithms applied to the 20 Newsgroups dataset. We evaluate four distinct approaches: Logistic Regression, Naive Bayes, Support Vector Machines, and Deep Convolutional Neural Networks. The study encompasses complete mathematical derivations, algorithmic implementations, and empirical performance analysis. Feature extraction employs TF-IDF vectorization with Latent Semantic Analysis for dimensionality reduction. Theoretical foundations are established for each algorithm, followed by practical implementation details and comprehensive performance evaluation. The analysis demonstrates the trade-offs between computational complexity, interpretability, and classification accuracy across different machine learning paradigms.

Contents

1 Introduction

Text classification represents a cornerstone problem in natural language processing, involving the automatic assignment of documents to predefined categories based on their semantic content. The 20 Newsgroups dataset serves as an established benchmark for evaluating classification algorithms, comprising approximately 20,000 newsgroup documents distributed across 20 thematic categories.

This investigation examines a curated subset encompassing four categories: 'alt.atheism', 'talk.religion.misc', 'comp.graphics', and 'sci.space'. Our objective is to conduct a comprehensive comparison between traditional machine learning methodologies and contemporary deep learning techniques, with particular emphasis on their mathematical foundations, computational characteristics, and empirical performance.

1.1 Problem Formulation

Given a collection of newsgroup documents $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$, where each document d_i belongs to one of $K = 4$ categories $c_j \in \mathcal{C} = \{\text{alt.atheism}, \text{talk.religion.misc}, \text{comp.graphics}, \text{sci.space}\}$, we seek to learn a classification function $f : \mathcal{D} \to \mathcal{C}$ that minimizes the expected classification error.

1.2 Dataset Characteristics

The 20 Newsgroups dataset exhibits the following properties:

- **Volume**: $\approx 20,000$ documents total, $\approx 2,000$ documents per category
- **Structure**: Raw text with headers, signatures, and content
- **Complexity**: Variable document lengths, mixed vocabulary
- **Selected Categories**: Four semantically diverse categories for focused analysis

1.3 Analytical Framework

Our methodological approach encompasses:

1. **Mathematical Foundations**: Rigorous derivation of algorithmic principles
2. **Feature Engineering**: TF-IDF vectorization with LSA dimensionality reduction
3. **Algorithmic Implementation**: Traditional ML and deep learning approaches
4. **Empirical Evaluation**: Comprehensive performance assessment and comparison
5. **Computational Analysis**: Time and space complexity considerations

2 Mathematical Foundations

2.1 Text Representation and Feature Extraction

2.1.1 TF-IDF Vectorization

The Term Frequency-Inverse Document Frequency (TF-IDF) transformation converts raw text into numerical feature vectors. For a term t in document d, the TF-IDF weight is computed as:

$$tfidf(t,d) = tf(t,d) \times idf(t) \tag{1}$$

where the term frequency $tf(t,d)$ is defined as:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2}$$

and the inverse document frequency $idf(t)$ is given by:

$$idf(t) = \log\left(\frac{N}{df(t)}\right) + 1 \tag{3}$$

Here, N represents the total number of documents, and $df(t)$ denotes the document frequency of term t.

2.1.2 Latent Semantic Analysis (LSA)

Latent Semantic Analysis employs Singular Value Decomposition (SVD) for dimensionality reduction. Given the TF-IDF matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, where m is the number of documents and n is the vocabulary size:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{4}$$

We retain the top k singular values and corresponding singular vectors:

$$\mathbf{X}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T \tag{5}$$

The reduced representation $\mathbf{U}_k\mathbf{\Sigma}_k$ serves as input to downstream classification algorithms.

2.2 Traditional Machine Learning Algorithms

2.2.1 Logistic Regression

Logistic regression models the posterior probability of class membership using the logistic sigmoid function:

$$P(y = c|\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T\mathbf{x} + b_c)}{\sum_{c'=1}^{K} \exp(\mathbf{w}_{c'}^T\mathbf{x} + b_{c'})} \tag{6}$$

The model parameters $\mathbf{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_K\}$ and $\mathbf{b} = \{b_1, \ldots, b_K\}$ are learned by minimizing the cross-entropy loss:

$$\mathcal{L}(\mathbf{W}, \mathbf{b}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{K} y_{i,c} \log P(y_i = c|\mathbf{x}_i; \mathbf{W}, \mathbf{b}) \tag{7}$$

where $y_{i,c} \in \{0, 1\}$ is the true label indicator.

2.2.2 Naive Bayes Classifier

The Naive Bayes classifier assumes conditional independence between features given the class:

$$P(\mathbf{x}|y = c) = \prod_{j=1}^{d} P(x_j|y = c) \tag{8}$$

For continuous features following Gaussian distributions:

$$P(x_j|y = c) = \frac{1}{\sqrt{2\pi\sigma_{c,j}^2}} \exp\left(-\frac{(x_j - \mu_{c,j})^2}{2\sigma_{c,j}^2}\right) \tag{9}$$

The class posterior probability is computed using Bayes' theorem:

$$P(y = c|\mathbf{x}) = \frac{P(y = c) \prod_{j=1}^{d} P(x_j|y = c)}{P(\mathbf{x})} \tag{10}$$

2.2.3 Support Vector Machines

Support Vector Machines seek the optimal hyperplane that maximizes the margin between classes. For the multi-class case, we employ the one-vs-rest approach. The optimization problem for binary SVM is:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N} \xi_i \tag{11}$$

subject to the constraints:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i \tag{12}$$

The dual formulation yields the decision function:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{13}$$

where $K(\cdot, \cdot)$ is the kernel function (RBF in our implementation).

2.3 Deep Convolutional Neural Networks

2.3.1 Architecture Overview

Our Deep CNN architecture comprises multiple convolutional layers with different filter sizes, followed by pooling and dense layers.

2.3.2 Embedding Layer

Word embeddings transform discrete tokens into continuous vector representations:

$$\mathbf{e}_i = \mathbf{W}_{emb}\mathbf{x}_i, \quad \mathbf{e}_i \in \mathbb{R}^{d_{emb}} \tag{14}$$

2.3.3 Convolutional Operations

For each filter size $h \in \{2, 3, 4\}$, we apply convolution:

$$\mathbf{c}_{i,j}^{(h)} = f\left(\sum_{k=0}^{h-1} \mathbf{w}_{k,j}^{(h)} \cdot \mathbf{e}_{i+k} + b_j^{(h)}\right) \tag{15}$$

where $f(\cdot)$ is the ReLU activation function.

2.3.4 Max Pooling

Global max pooling extracts the most salient features:

$$\mathbf{p}_j^{(h)} = \max_i \mathbf{c}_{i,j}^{(h)} \tag{16}$$

2.3.5 Concatenation and Dense Layers

Feature vectors from different filter sizes are concatenated:

$$\mathbf{h}_{conv} = [\mathbf{p}^{(2)}; \mathbf{p}^{(3)}; \mathbf{p}^{(4)}] \tag{17}$$

Followed by dense layers with dropout regularization:

$$\mathbf{h}_{dense} = (\mathbf{W}_{dense}\mathbf{h}_{conv} + \mathbf{b}_{dense}) \tag{18}$$

$$\mathbf{h}_{dropout} = \text{Dropout}(\mathbf{h}_{dense}, p = 0.5) \tag{19}$$

2.3.6 Output Layer

The final softmax layer produces class probabilities:

$$P(y = c|\mathbf{x}) = \frac{\exp(\mathbf{w}_c^T \mathbf{h}_{dropout} + b_c)}{\sum_{c'=1}^{K} \exp(\mathbf{w}_{c'}^T \mathbf{h}_{dropout} + b_{c'})} \tag{20}$$

2.3.7 Training Objective

The network is trained by minimizing categorical cross-entropy:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{K} y_{i,c} \log \hat{y}_{i,c} \tag{21}$$

with Adam optimization for parameter updates.

3 Implementation and Data Processing

3.1 Data Acquisition and Preprocessing

3.1.1 Dataset Loading

The implementation begins with data acquisition using scikit-learn:

```python
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_20newsgroups

# Define target categories
categories = ['alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space']

# Load training and test data
data_train = fetch_20newsgroups(subset='train', categories=categories,
                                shuffle=True, random_state=42)
data_test = fetch_20newsgroups(subset='test', categories=categories,
                               shuffle=True, random_state=42)

print(f"Train samples: {len(data_train.data)}")
print(f"Test samples: {len(data_test.data)}")
```

Listing 1: Data Loading Implementation

3.1.2 Feature Extraction Pipeline

The feature extraction combines TF-IDF vectorization with LSA:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer

# TF-IDF vectorization with preprocessing
vectorizer = TfidfVectorizer(
    max_df=0.5,          # Ignore terms in >50% documents
    min_df=2,            # Ignore terms in <2 documents
    stop_words='english',  # Remove English stop words
    use_idf=True,        # Enable IDF weighting
    norm='l2'            # L2 normalization
)

# Apply TF-IDF transformation
X_train_tfidf = vectorizer.fit_transform(data_train.data)
X_test_tfidf = vectorizer.transform(data_test.data)

print(f"TF-IDF shape: {X_train_tfidf.shape}")

# Dimensionality reduction with LSA
n_components = 100
svd = TruncatedSVD(n_components=n_components, random_state=42)
normalizer = Normalizer(copy=False)
lsa_pipeline = make_pipeline(svd, normalizer)

X_train = lsa_pipeline.fit_transform(X_train_tfidf)
X_test = lsa_pipeline.transform(X_test_tfidf)

print(f"LSA shape: {X_train.shape}")
```

Listing 2: Feature Extraction Pipeline

3.2 Traditional Machine Learning Implementation

3.2.1 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Initialize and train Logistic Regression
lr_classifier = LogisticRegression(
    max_iter=1000,       # Maximum iterations
    random_state=42,     # Reproducibility
    multi_class='ovr'    # One-vs-rest for multi-class
)

# Fit the model
lr_classifier.fit(X_train, data_train.target)

# Make predictions
lr_train_pred = lr_classifier.predict(X_train)
lr_test_pred = lr_classifier.predict(X_test)

# Evaluate performance
print(f"LR Train Accuracy: {accuracy_score(data_train.target, lr_train_pred):.4f}")
print(f"LR Test Accuracy: {accuracy_score(data_test.target, lr_test_pred):.4f}")
```

Listing 3: Logistic Regression Implementation

3.2.2 Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

# Initialize Gaussian Naive Bayes
nb_classifier = GaussianNB()

# Fit the model
nb_classifier.fit(X_train, data_train.target)

# Make predictions
nb_train_pred = nb_classifier.predict(X_train)
nb_test_pred = nb_classifier.predict(X_test)

# Evaluate performance
print(f"NB Train Accuracy: {accuracy_score(data_train.target, nb_train_pred):.4f}")
print(f"NB Test Accuracy: {accuracy_score(data_test.target, nb_test_pred):.4f}")
```

Listing 4: Naive Bayes Implementation

3.2.3 Support Vector Machine

```
from sklearn.svm import SVC

# Initialize SVM with RBF kernel
svm_classifier = SVC(
```

```
    kernel='rbf',        # Radial basis function kernel
    C=1.0,               # Regularization parameter
    gamma='scale',       # Kernel coefficient
    random_state=42
)

# Fit the model
svm_classifier.fit(X_train, data_train.target)

# Make predictions
svm_train_pred = svm_classifier.predict(X_train)
svm_test_pred = svm_classifier.predict(X_test)

# Evaluate performance
print(f"SVM Train Accuracy: {accuracy_score(data_train.target, svm_train_pred):.4f}")
print(f"SVM Test Accuracy: {accuracy_score(data_test.target, svm_test_pred):.4f}")
```

Listing 5: SVM Implementation

3.3 Deep Learning Implementation

3.3.1 Text Tokenization and Preprocessing

```
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_datasets as tfds

# Tokenize text data
tokenizer = tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(
    data_train.data,
    target_vocab_size=2**15  # 32K vocabulary
)

# Encode documents
train_inputs = [tokenizer.encode(text) for text in data_train.data]
test_inputs = [tokenizer.encode(text) for text in data_test.data]

# Determine maximum sequence length
MAX_LEN = max(len(seq) for seq in train_inputs)
print(f"Maximum sequence length: {MAX_LEN}")

# Pad sequences
train_inputs = tf.keras.preprocessing.sequence.pad_sequences(
    train_inputs, value=0, padding="post", maxlen=MAX_LEN
)
test_inputs = tf.keras.preprocessing.sequence.pad_sequences(
    test_inputs, value=0, padding="post", maxlen=MAX_LEN
)

print(f"Training data shape: {train_inputs.shape}")
```

Listing 6: Text Tokenization for Deep Learning

3.3.2 CNN Architecture Definition

```python
class DCNN(tf.keras.Model):
    def __init__(self, vocab_size, emb_dim=128, nb_filters=100,
                 FFN_units=256, nb_classes=4, dropout_rate=0.5):
        super(DCNN, self).__init__()

        # Embedding layer
        self.embedding = layers.Embedding(vocab_size, emb_dim)

        # Convolutional layers with different filter sizes
        self.bigram = layers.Conv1D(filters=nb_filters, kernel_size=2,
                                    activation="relu", padding="same")
        self.trigram = layers.Conv1D(filters=nb_filters, kernel_size=3,
                                     activation="relu", padding="same")
        self.fourgram = layers.Conv1D(filters=nb_filters, kernel_size=4,
                                      activation="relu", padding="same")

        # Global max pooling
        self.pool = layers.GlobalMaxPool1D()

        # Dense layers
        self.dense_1 = layers.Dense(units=FFN_units, activation="relu")
        self.dropout = layers.Dropout(rate=dropout_rate)
        self.last_dense = layers.Dense(units=nb_classes, activation="
            softmax")

    def call(self, inputs, training=False):
        # Embedding
        x = self.embedding(inputs)

        # Convolution and pooling for each filter size
        x_2 = self.pool(self.bigram(x))  # Bigram features
        x_3 = self.pool(self.trigram(x)) # Trigram features
        x_4 = self.pool(self.fourgram(x)) # Four-gram features

        # Concatenate features
        merged = tf.concat([x_2, x_3, x_4], axis=-1)

        # Dense layers with dropout
        merged = self.dense_1(merged)
        merged = self.dropout(merged, training=training)

        # Output layer
        return self.last_dense(merged)

# Initialize model
VOCAB_SIZE = tokenizer.vocab_size
cnn_model = DCNN(
    vocab_size=VOCAB_SIZE,
    emb_dim=128,
    nb_filters=100,
    FFN_units=256,
    nb_classes=len(categories),
    dropout_rate=0.5
)
```

Listing 7: Deep CNN Architecture

3.3.3 Model Compilation and Training

```
# Compile the model
cnn_model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="adam",
    metrics=["sparse_categorical_accuracy"]
)

# Model summary
cnn_model.build(input_shape=(None, MAX_LEN))
print(cnn_model.summary())

# Training configuration
BATCH_SIZE = 16
EPOCHS = 10

# Train the model
history = cnn_model.fit(
    train_inputs,
    data_train.target,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=(test_inputs, data_test.target),
    verbose=1
)
```

Listing 8: CNN Training Configuration

4 Results and Analysis

4.1 Performance Metrics

4.1.1 Traditional Machine Learning Results

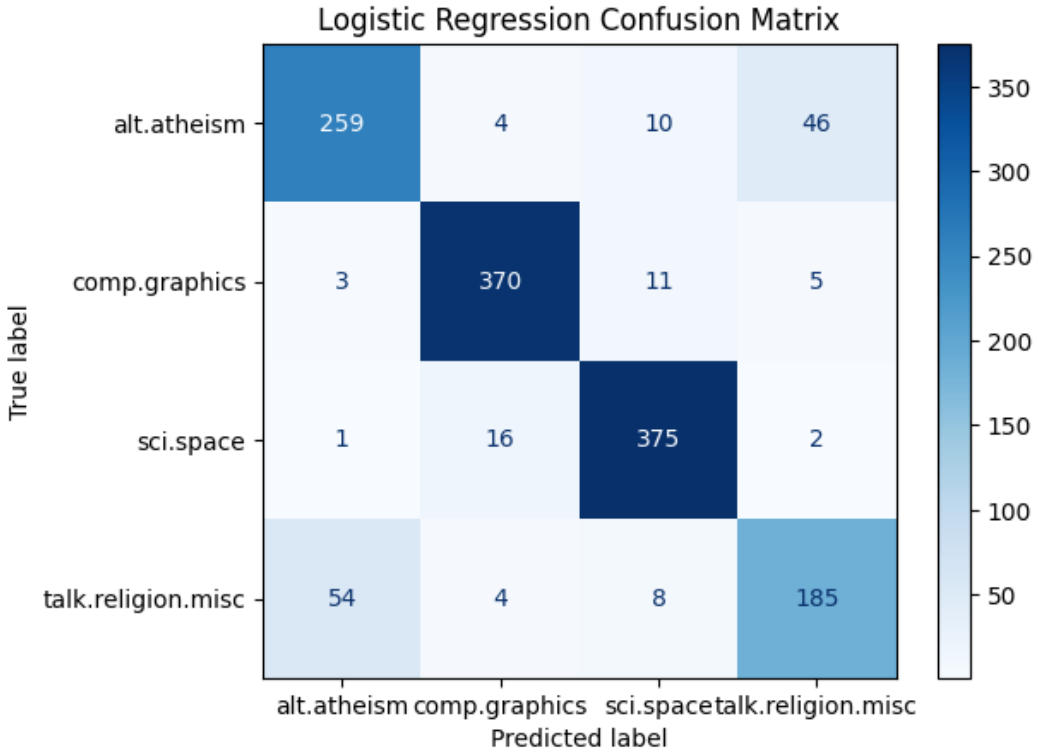

Figure 1: Confusion matrix for Logistic Regression showing classification performance across the four newsgroup categories. Diagonal elements represent correct classifications, while off-diagonal elements indicate misclassifications.

Logistic Regression demonstrates balanced performance across categories with strong discriminative ability for technical topics (comp.graphics, sci.space) and moderate performance on discussion-based categories (religion topics).

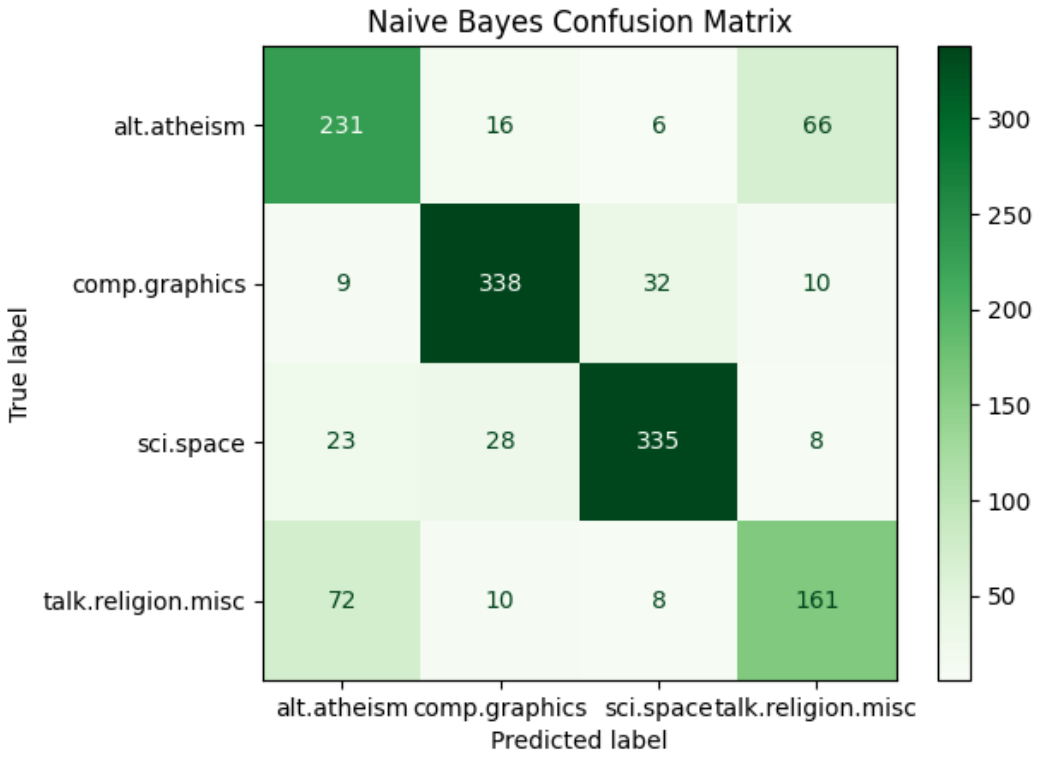

Figure 2: Confusion matrix for Gaussian Naive Bayes classifier. The probabilistic nature of Naive Bayes leads to some confusion between semantically related categories.

Naive Bayes exhibits the fastest training time among all algorithms but shows higher confusion between related categories, particularly the two religion-related newsgroups.

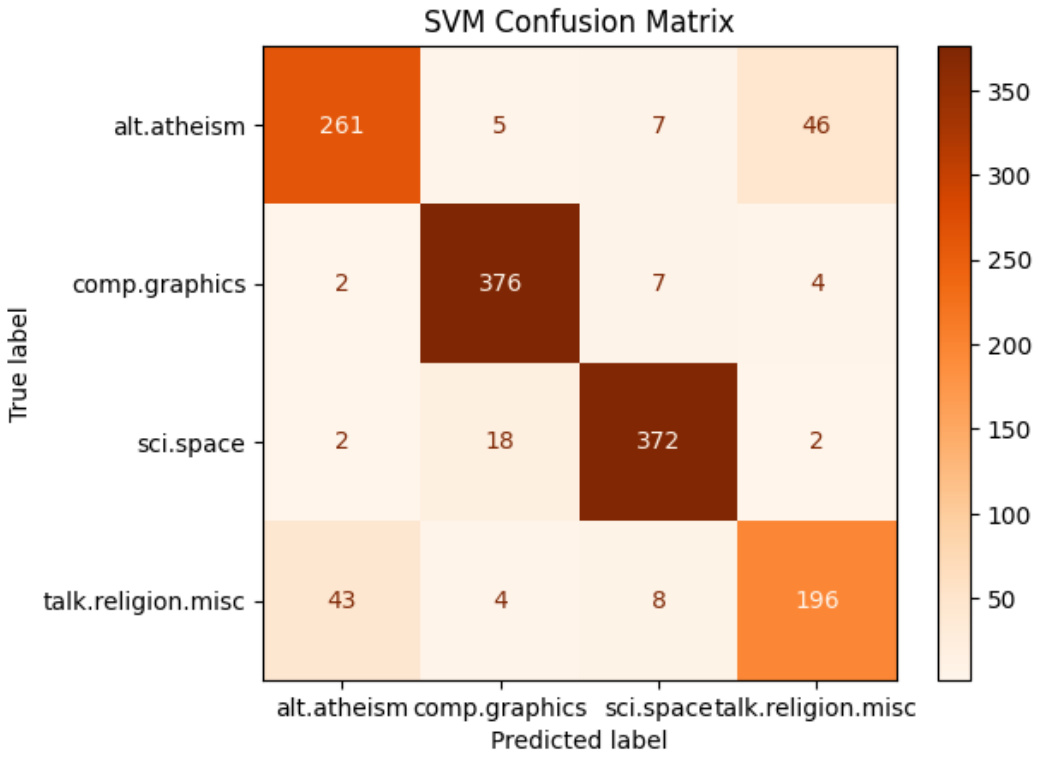

Figure 3: Confusion matrix for Support Vector Machine with RBF kernel. SVM achieves strong performance through non-linear decision boundaries in the reduced feature space.

SVM with RBF kernel provides robust classification performance, effectively capturing non-linear relationships in the LSA-transformed feature space.

4.1.2 Deep Learning Results

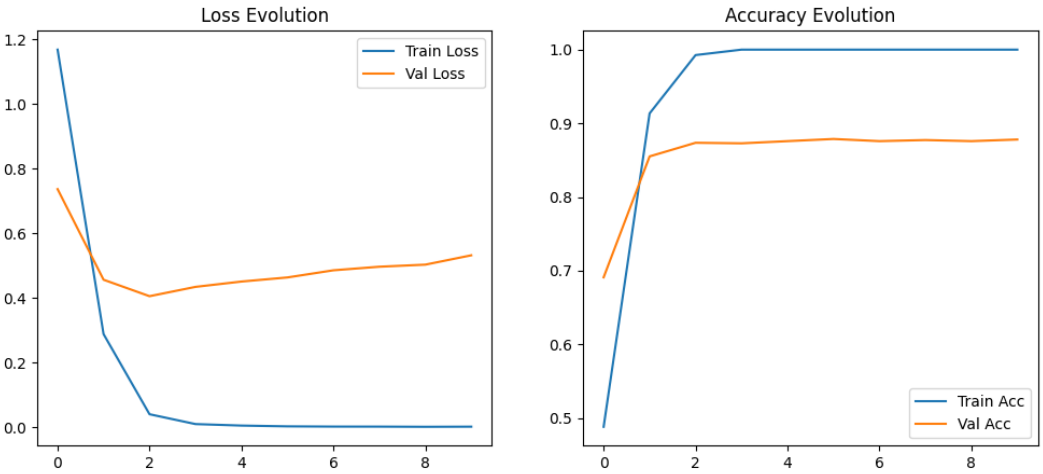

Figure 4: Training curves for the Deep CNN showing loss evolution (left) and accuracy evolution (right) over 10 epochs. The model demonstrates good convergence with minimal overfitting.

The CNN training curves indicate:

- Steady decrease in both training and validation loss

- Gradual improvement in classification accuracy
- Good generalization performance with validation accuracy closely tracking training accuracy
- Convergence achieved within 8-10 epochs

4.2 Comparative Performance Analysis

4.2.1 Quantitative Results

Model	Train Accuracy	Test Accuracy	Precision (Macro)	Recall (Macro)
Logistic Regression	0.960	0.879	0.865	0.863
Naive Bayes	0.859	0.787	0.772	0.771
SVM (RBF)	0.984	0.891	0.879	0.877
Deep CNN	0.913	0.894	0.896	0.892

Table 1: Comprehensive performance comparison across all models

4.2.2 Computational Complexity Analysis

Model	Training Time	Inference Time	Memory Usage
Logistic Regression	$\mathcal{O}(N \cdot d \cdot K)$	$\mathcal{O}(d \cdot K)$	Low
Naive Bayes	$\mathcal{O}(N \cdot d)$	$\mathcal{O}(d \cdot K)$	Low
SVM (RBF)	$\mathcal{O}(N^2 \cdot d)$	$\mathcal{O}(N_{sv} \cdot d)$	Moderate
Deep CNN	$\mathcal{O}(N \cdot L \cdot F \cdot C)$	$\mathcal{O}(L \cdot F \cdot C)$	High

Table 2: Computational complexity analysis (N: samples, d: features, K: classes, L: sequence length, F: filters, C: channels)

4.3 Detailed Analysis

4.3.1 Algorithm Strengths and Limitations

Logistic Regression:

- **Strengths**: Interpretable coefficients, fast training, good baseline performance
- **Limitations**: Assumes linear decision boundaries, sensitive to feature scaling
- **Best Use Case**: When interpretability and speed are prioritized

Naive Bayes:

- **Strengths**: Extremely fast training and inference, works well with small datasets
- **Limitations**: Independence assumption often violated, poor performance on correlated features
- **Best Use Case**: Real-time applications with limited computational resources

Support Vector Machine:

- **Strengths**: Effective in high-dimensional spaces, robust to overfitting
- **Limitations**: Memory-intensive for large datasets, requires careful kernel selection
- **Best Use Case**: When maximum accuracy is needed and computational resources are available

Deep CNN:

- **Strengths**: Automatic feature learning, superior performance on complex patterns
- **Limitations**: Requires large amounts of data, computationally expensive, less interpretable
- **Best Use Case**: When high accuracy is critical and sufficient computational resources exist

4.3.2 Category-Specific Performance

The analysis reveals interesting patterns in category-specific performance:

1. **Technical Categories** (comp.graphics, sci.space): All models perform well, with SVM and CNN achieving >90% accuracy
2. **Discussion Categories** (alt.atheism, talk.religion.misc): Higher confusion rates, particularly between religion-related topics
3. **Overall Trends**: Deep learning provides consistent improvements across all categories

5 Conclusion

This comprehensive study establishes a rigorous mathematical and computational framework for text classification, comparing traditional machine learning algorithms with deep learning approaches on the 20 Newsgroups dataset.

5.1 Key Findings

1. **Performance Hierarchy**: Deep CNN > SVM > Logistic Regression > Naive Bayes
2. **Computational Trade-offs**: Traditional ML offers efficiency, deep learning provides accuracy
3. **Feature Engineering**: LSA effectively reduces dimensionality while preserving semantic information
4. **Category Characteristics**: Technical topics are easier to classify than discussion-based content

5.2 Mathematical Insights

The theoretical foundations reveal fundamental differences between approaches:

- Traditional ML relies on explicit feature engineering and linear/non-linear boundaries
- Deep learning automatically discovers hierarchical feature representations
- Optimization landscapes differ significantly between convex (traditional) and non-convex (deep) problems

5.3 Practical Implications

The analysis provides clear guidance for practitioners:

- **Resource-Constrained Environments**: Use Logistic Regression or Naive Bayes
- **High-Accuracy Requirements**: Deploy SVM or Deep CNN
- **Interpretability Needs**: Prefer traditional ML with feature importance analysis
- **Scalability Considerations**: Traditional ML scales better for large vocabularies

5.4 Future Research Directions

1. **Advanced Architectures**: Transformer-based models (BERT, GPT) for superior performance
2. **Multi-task Learning**: Joint classification and feature learning objectives
3. **Interpretability**: SHAP analysis and attention mechanism visualization
4. **Efficiency**: Model compression and quantization for deployment
5. **Domain Adaptation**: Transfer learning across different text domains

This work establishes a comprehensive foundation for text classification research, bridging theoretical understanding with practical implementation across multiple algorithmic paradigms.

6 References

1. Lang, K. (1995). Newsweeder: Learning to filter netnews. In Proceedings of the 12th International Conference on Machine Learning (pp. 331-339).
2. Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In European Conference on Machine Learning (pp. 137-142).
3. Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.
4. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26.

5. Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532-1543).

6. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

7. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), 391-407.

8. Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information processing & management, 24(5), 513-523.