Vietnam National University - Ho Chi Minh City
Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

MACHINE LEARNING

Assignment Report

COMPREHENSIVE MACHINE LEARNING MODELS: IMPLEMENTATION AND COMPARISON

Instructor:	Nguyễn An Khương, CSE-HCMUT
Student(s):	Lê Quốc Bảo - 2252065 - Leader
	Lê Đỗ Minh Anh - 2252023
	Nguyễn Quốc Anh - 2252035

Ho Chi Minh City, 3/2025

Contribution

No.	Name	Student ID	Task	Contribution
1	Lê Quốc Bảo	2252065	Theory, implementation, and detailed explanation for Decision Tree and Neural Network. Writing overall report.	33%
2	Lê Đỗ Minh Anh	2252023	Theory, implementation, and detailed explanation for Graphical Models (Bayesian Networks, HMM). Running and evaluating model results.	33%
3	Nguyễn Quốc Anh	2252035	Theory, implementation, and detailed explanation for Naive Bayes. Creating and editing Github Repository.	33%

Table 1: Project Contributions

Contents

1 Overview

In this project, we evaluate a variety of machine learning models introduced in the course to address a challenging text classification task. The primary objective is to classify news articles into four distinct categories—**World**, **Sports**, **Business**, and **Science/Technology**—using the **AG News** dataset.

The **AG News** dataset serves as a robust benchmark for text classification, comprising thousands of news articles sourced from reputable outlets such as **Reuters** and **AP**. Each article is pre-labeled into one of the four categories, providing a clear and organized division of topics. The dataset is partitioned into training and testing sets, ensuring a balanced and realistic scenario for evaluating model performance. Moreover, the concise headlines and summaries further enhance its utility for developing and comparing various text classification approaches.

Multiple models were implemented and rigorously tested to analyze their effectiveness in this use case. Each model underwent a comprehensive pipeline, including data preprocessing, hyperparameter tuning, and performance evaluation using standard metrics such as **Accuracy**, **Precision**, **Recall** and **F1-score**. By systematically comparing these models, we aim to evaluate their respective strengths and limitations, ultimately guiding improved model selection and optimization strategies for text classification tasks.

2 Data Preprocessing

2.1 Loading the Dataset

The dataset is loaded from Hugging Face using the `load_dataset` function. Both training and testing sets are converted to pandas DataFrames.

```python
from datasets import load_dataset
dataset_name="fancyzhx/ag_news"
dataset = load_dataset(dataset_name)
```

2.2 Mapping Labels

Numeric labels are mapped to string categories using a dictionary (`CATEGORY_MAPPING`), ensuring that outputs are human-readable (e.g., `0` → `World`, `1` → `Sports`, etc.).

```python
CATEGORY_MAPPING = {0: 'World', 1: 'Sports', 2: 'Business', 3: 'Sci/Tech'}
train_df['Category'] = train_df['label'].map(CATEGORY_MAPPING)
test_df['Category'] = test_df['label'].map(CATEGORY_MAPPING)
```

2.3 Handling Missing Values

In this implementation, missing values are managed by first identifying any null entries in both the text and the label columns. If missing labels are found, the corresponding rows are dropped to prevent inconsistencies in classification. For missing text values, instead of removing rows, we replace them with a placeholder such as *"Unknown"* to maintain the structure of the dataset and avoid losing valuable information. This approach ensures that the feature extraction process, particularly TF-IDF vectorization, remains unaffected by incomplete data. By effectively handling missing values, we enhance the model's ability to generalize well and prevent biases due to data omissions.

3 Pipeline Construction

3.1 Text Preprocessing

```
vectorizer = TfidfVectorizer(lowercase=True, stop_words='english', max_features=1000)
label_encoder = LabelEncoder()

X_tfidf = vectorizer.fit_transform(X_train).toarray()
y_encoded = label_encoder.fit_transform(y_train)
```

A `TfidfVectorizer` transforms raw text into numerical features. Its configuration includes:

- Lowercasing of text,
- Removal of English stop words,
- Limiting the number of features to 1000 to manage high dimensionality.

3.2 Model Implementation

The model implementation involves selecting a suitable classification algorithm for text classification and adapting it to handle textual features effectively. Different models have unique requirements for optimal performance:

3.2.1 Decision Trees

Decision trees require specific transformations to process high-dimensional text data effectively. In our approach:

- Text data is transformed into numerical features using `TfidfVectorizer`.
- High dimensionality is managed by limiting features to 1000.
- Pruning strategies, such as Cost Complexity Pruning (CCP), are implemented using the `ccp_alpha` parameter to prevent overfitting.

3.2.2 Naïve Bayes

Naïve Bayes classifiers assume feature independence and require handling of sparse features:

- TF-IDF transformation naturally results in sparse features, which suit Naïve Bayes classifiers well.
- Laplace (or additive) smoothing is applied to avoid zero probability issues.
- The model is optimized by tuning the `alpha` parameter, which controls the level of smoothing.

3.2.3 Support Vector Machine

Support Vector Machines (SVM) are effective for high-dimensional, sparse feature spaces:

- TF-IDF transformation is used to convert raw text into sparse numerical representations.
- The model's performance is tuned via the regularization parameter `C`, which controls the trade-off between margin maximization and classification error.
- Optionally, Truncated SVD (a PCA-like dimensionality reduction method) is applied to reduce feature dimensionality while preserving variance, improving computational efficiency.
- The number of components for SVD is selected based on a cumulative variance threshold to retain essential information.

3.2.4 Neural Networks

A simple neural network model is used for text classification:

- Text features are transformed using `TfidfVectorizer`.
- A basic feedforward neural network with a single hidden layer is implemented.
- The model uses an activation function (e.g., ReLU or sigmoid) to introduce non-linearity.
- Optimization techniques such as gradient descent and backpropagation are used for training.

3.2.5 Graphical Models

Graph-based models like Bayesian Networks and Hidden Markov Models (HMMs) require structured probabilistic dependencies:

- Dependencies between words and phrases are modeled using conditional probability tables.
- State transitions for sequences are implemented to capture contextual relationships.
- Structure learning methods are applied to discover patterns in textual data.

3.2.6 Ensemble Models

Ensemble learning combines multiple base models to improve the overall performance compared to a single model. In this project, we implement two ensemble techniques: Bagging and Boosting, both built upon a soft-voting ensemble of three classifiers:

- Logistic Regression (max_iter=1000)
- Multinomial Naive Bayes
- Decision Tree Classifier (max_depth=10 for Bagging, max_depth=100 for Boosting)

3.2.6.a Bagging Classifier

Bagging (Bootstrap Aggregating) aims to reduce variance by training multiple models independently on different bootstrapped subsets of the data and then aggregating their predictions.

- **Base estimators:** Logistic Regression, MultinomialNB, Decision Tree (depth=10)
- **Meta model:** Voting Classifier with soft voting
- **Final ensemble:** Bagging Classifier

```
VotingClassifier(estimators=base_models, voting='soft')
BaggingClassifier(estimator=voting_clf, n_estimators=50)
```

3.2.6.b Boosting Classifier

Boosting sequentially trains models such that each model tries to correct the mistakes of the previous ones. In our case, we apply AdaBoost to a soft-voting ensemble classifier.

- **Base estimators:** Logistic Regression, MultinomialNB, Decision Tree (depth=100)
- **Voting weights:** [2, 1, 1] to emphasize Logistic Regression

- **Final ensemble:** AdaBoostClassifier on top of the voting ensemble

```python
VotingClassifier(estimators=base_models, voting='soft', weights=[2,1,1])
AdaBoostClassifier(estimator=voting_clf, n_estimators=10)
```

3.3 Pipeline Integration

To ensure an end-to-end automated workflow, a machine learning pipeline is constructed, integrating data preprocessing, feature transformation, model training, and evaluation. The pipeline consists of:

- **Text Vectorization:** Raw text is standardized into TF-IDF features for numerical representation.
- **Model Selection:** Based on task requirements, models such as Decision Trees, Naïve Bayes, Neural Networks, or Graphical Models are implemented.
- **Hyperparameter Tuning:** A grid search approach (`GridSearchCV`) is used to identify optimal model parameters.
- **Evaluation Metrics:** The trained model is evaluated using accuracy, precision, recall, F1-score, and confusion matrices.

By integrating all components into a unified pipeline, we streamline the process of training and testing different models efficiently, enabling fair comparisons and model selection for the best-performing.

4 Hyperparameter Tuning

4.1 Decision Tree model

We demonstrate hyperparameter tuning on Decision Tree model.

```python
def tune_hyperparameters(pipeline, X_train, y_train):
    param_grid = {
        "clf__max_depth": [10, 20, None],
        "clf__min_samples_split": [2, 5, 10],
        "clf__ccp_alpha": [0.0, 0.001, 0.01]
    }
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    grid_search = GridSearchCV(pipeline, param_grid, cv=skf, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    print("Best hyperparameters found:", grid_search.best_params_)
    return grid_search.best_estimator_
```

Hyperparameter tuning is performed using `GridSearchCV` to optimize model performance. The tuning process involves:

- Defining a parameter grid for `max_depth`, `min_samples_split`, and `ccp_alpha`.
- Utilizing a 5-fold stratified cross-validation (`StratifiedKFold`) to ensure balanced class distributions across splits.
- Training multiple models on different combinations of hyperparameters and selecting the best performing set.
- Analyzing the best hyperparameters, providing insights into the optimal model configuration.

4.2 SVM with PCA

We demonstrate hyperparameter tuning on a Support Vector Machine (SVM) model integrated with Truncated SVD for dimensionality reduction.

```python
def tune_hyperparameters(X_train, y_train, cv=3):
pipeline = build_pipeline() # Includes TF-IDF, TruncatedSVD, and SVM
param_grid = get_param_grid()  # Defines hyperparameter search space

grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=cv,
    n_jobs=-1,
    verbose=3,
    scoring='accuracy'
)

grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best CV score:", grid_search.best_score_)
return grid_search
```

Hyperparameter tuning is performed using `GridSearchCV` to optimize the combined SVM and dimensionality reduction pipeline. The process includes:

- Automatically determining the number of SVD components using cumulative explained variance to retain sufficient information while reducing dimensionality.
- Defining a parameter grid including:
  - `tfidf__max_features` – limits vocabulary size.
  - `tfidf__ngram_range` – explores unigrams and bigrams.
  - `svm__C` – regularization parameter of SVM.
- Using stratified k-fold cross-validation (default $k = 3$) to ensure robustness across class distributions.
- Leveraging parallel computation (`n_jobs=-1`) to accelerate grid search.
- Evaluating combinations of preprocessing and model parameters to select the configuration yielding the highest validation accuracy.

5 Performance Analysis

5.1 Evaluation Metrics

The trained model is evaluated in a separate test set using multiple performance metrics.

```python
# Evaluate on the test set
print("=== Test Set Performance ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Performance Analysis: Plot confusion matrix
plot_confusion_matrix(y_test, y_pred, labels=list(CATEGORY_MAPPING.values()))
```

- **Accuracy Score**: Measures the overall accuracy of the predictions.
- **Classification Report**: Provides precision, recall, and F1-score for each category, offering detailed performance insights.
- **Confusion Matrix**: Visualize the highlight misclassification patterns and assess model robustness.

5.2 Cross-Validation

To ensure stability and robustness, cross-validation is performed on the training set:

```python
# Cross-validation on the training set
print("=== Cross-Validation on Training Set ===")
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(tuned_pipeline, X_train, y_train, cv=skf, scoring='accuracy'
    )
print(f"CV Accuracy Scores: {cv_scores}")
print(f"Mean CV Accuracy: {np.mean(cv_scores):.4f} +- {np.std(cv_scores):.4f}")
```

- A 5-fold `StratifiedKFold` cross-validation approach is used to prevent class imbalance.
- `CrossValScore` is employed to compute accuracy scores across different training splits.
- Mean and standard deviation of cross-validation scores are calculated to assess the model variance.

6 Model Analysis

6.1 Decision Tree Model

6.1.1 Overview

The Decision Tree model is a rule-based, non-parametric learning algorithm that is particularly useful for text classification due to its interpretability and ability to capture nonlinear decision boundaries. The model recursively splits the dataset based on word feature importance, aiming to create homogeneous groups for classification. The key concept behind Decision Trees is Information Gain and Entropy, which help determine the best split at each node.

1. **Entropy:** Measures the impurity of a dataset:

$$H(S) = -\sum_{i=1}^{e} P_i \log_2 P_i$$

   where P_i is the proportion of class i in the dataset S.

2. **Information Gain:** Measures the reduction in entropy after splitting on a feature:

$$IG(S, A) = H(S) - \sum_{v \in A} \frac{|S_v|}{|S|} H(S_v)$$

   - A is the feature being split on.
   - S_v is the subset of S where feature A has value v

For this task, the model is trained using a TF-IDF feature representation, which transforms raw text data into numerical values based on term frequency and inverse document frequency. This helps the model identify important words while reducing the impact of common, less informative terms.

6.1.2 Key Parameters and Their Impact

The Decision Tree classifier includes several hyperparameters that directly affect its performance, generalization, and interpretability:

```
DecisionTreeClassifier(max_depth=max_depth,
                       min_samples_split=min_samples_split,
                       ccp_alpha=ccp_alpha,
                       random_state=42))
```

Max Depth (`max_depth`)

- Controls the maximum depth of the tree, limiting the number of hierarchical decision splits.
- A deeper tree captures more intricate patterns but risks overfitting.
- A shallower tree generalizes better but may underfit by failing to capture important patterns.

Minimum Samples per Split (`min_samples_split`)

- Defines the minimum number of samples required to further split a node.
- Lower values create a highly detailed tree but increase sensitivity to noise.
- Higher values enforce more generalized splits, reducing overfitting.

Complexity Parameter for Pruning (`ccp_alpha`)

- Controls Minimal Cost-Complexity Pruning, which eliminates splits that provide marginal improvement in impurity reduction.
- A higher `ccp_alpha` encourages more aggressive pruning, simplifying the model.
- A lower `ccp_alpha` retains more splits, increasing complexity and potential overfitting.

6.1.3 Feature Importance Analysis

The Decision Tree model allows for feature importance extraction, which helps understand which words contribute most to classification.

- Features with high importance scores correspond to terms that strongly distinguish between categories (e.g., “stocks” for Business, “match” for Sports).
- Words with lower importance may be neutral across categories, contributing less to decision-making.
- By analyzing TF-IDF feature weights, we can identify which words are the strongest predictors for each category and refine feature selection accordingly.

6.1.4 Strengths and Limitations

Strengths

- **Interpretability**: Decision Trees provide clear, human-readable decision rules.
- **Feature Importance Insights**: The model highlights which words are crucial for classification.
- **Non-Parametric**: No assumptions about data distribution are required.

Limitations

- **Overfitting**: A deep tree captures too much detail, leading to poor generalization.
- **Sparse Feature Sensitivity**: Text data transformed via TF-IDF can introduce high dimensionality, making decision boundary learning more complex.
- **Limited Generalization**: Unlike ensemble models (e.g., Random Forest), a single Decision Tree may struggle with borderline classifications.

6.2 Neural Network

6.2.1 Overview

Neural Network is a function approximator that models complex relationships in data using layers of neurons. For text classification, the most common architectures include fully connected networks and recurrent networks (LSTMs, Transformers).

1. Forward Propagation:
   - Each neuron computes a weighted sum of inputs, applies an activation function:

   $$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$$

   $$a^{(l)} = \sigma(z^{(l)})$$

   where:
     - $W^{(l)}$ and $b^{(l)}$ are the weights and biases of layer l.
     - σ is an activation function.
     - $a^{(l)}$ is the activation output.
2. Loss Function (Cross-Entropy Loss for Classification):

   $$L = -\sum_{i=1}^{N} y_i \log \hat{y}_i$$

   where:
   - y_i is the true class label.
   - $\hat{y}_i$ is the predicted probability.
3. Backpropagation:
   - Gradients are computed using the chain rule:

   $$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial W^{(l)}}$$

6.2.2 Key Parameters and Their Impact

6.2.2.a MLP Classifier Parameters

– **Hidden Layer Sizes (`hidden_layer_sizes=(64, 32)`):**
The network has two hidden layers with 64 and 32 neurons respectively. Increasing the number of neurons or layers can capture more complex patterns but also raises the risk of overfitting if the dataset is small or noisy. Conversely, a smaller network may underfit if the complexity of the task exceeds the network capacity.

- **Regularization Parameter (`alpha=0.001`):**
  This parameter acts as a weight decay to prevent overfitting by penalizing large weights. A smaller alpha may allow more complex patterns (with potential overfitting), while a larger alpha enforces smoother solutions but might underfit.

- **Maximum Iterations (`max_iter=100`):**
  The maximum number of epochs for training defines how long the network is allowed to learn. In this case, 100 iterations may be sufficient for a moderately sized dataset, but if the model has not converged, increasing the iterations could improve performance at the cost of additional computation.

- **Learning Rate (`learning_rate_init=0.01`):**
  This controls how fast the model adapts to the problem. A high learning rate can lead to rapid learning but might overshoot minima, while a too-low rate might slow down convergence.

- **Batch Size (`batch_size=32`):**
  This parameter determines how many samples are processed before updating the model's weights. A balanced batch size can stabilize the training process and optimize performance.

6.2.2.b TF-IDF Vectorizer Parameters

- **Lowercase Conversion and Stop Words:**
  Converting text to lowercase and removing common stop words (using `stop_words='english'`) helps reduce noise and focus on the most discriminative terms.

- **Maximum Features (`max_features=1000`):**
  Limiting the feature space to the top 1000 words helps reduce dimensionality and computational cost. However, it also means that some potentially useful rare words might be excluded.

Collectively, these parameters influence the model's capacity to learn from text data. A well-tuned combination of these settings is essential to balance model complexity, convergence speed, and generalization ability.

6.2.3 Feature Importance Analysis

Neural networks, such as the MLP used here, do not inherently provide a straightforward interpretation of feature importance compared to tree-based methods. However, several approaches can be considered:

- **TF-IDF Weights:**
  Since the text is vectorized using TF-IDF, the vectorizer assigns weights to words based on their frequency and uniqueness. Analyzing these scores can provide insights into which words are considered more important before they enter the MLP.

- **Weight Analysis in the First Layer:**
  One might examine the weights connecting the input features (derived from TF-IDF scores) to the first hidden layer. Larger absolute weights may indicate a higher impact of certain features on the network's decisions. However, due to the non-linear transformations and subsequent layers, this analysis is not as straightforward as with linear models.

- **Model-Agnostic Methods:**
  Techniques such as permutation importance, SHAP (SHapley Additive exPlanations), or LIME (Local Interpretable Model-agnostic Explanations) can be applied post-hoc to estimate the contribution of

individual features. These methods provide a more nuanced view of feature impact despite the black-box nature of neural networks.

In summary, while the MLP classifier does not offer built-in feature importance scores, complementary methods can be used to interpret the role of individual words in the classification process.

6.2.4 Strengths and Limitations

6.2.4.a Strengths

- **Simplicity and Efficiency:**
  The pipeline leverages a straightforward TF-IDF vectorization followed by an MLP classifier. This simplicity allows for efficient training and easy modifications.
- **Nonlinear Learning Capability:**
  The neural network can model complex, non-linear relationships in the data, which may result in better performance compared to linear models for certain text classification tasks.
- **Multi-Class Handling:**
  The design naturally supports multi-class classification, making it suitable for datasets like AG News where multiple topics are present.

6.2.4.b Limitations

- **Limited Interpretability:**
  Unlike models such as decision trees, MLPs are less transparent. The lack of direct feature importance scores means that additional analysis (using permutation importance, SHAP, or LIME) is necessary to interpret the model's decisions.
- **Hyperparameter Sensitivity:**
  The performance of the model is heavily dependent on the selection of hyperparameters (e.g., hidden layer sizes, alpha, learning rate). Suboptimal choices may lead to underfitting or overfitting.
- **Convergence Concerns:**
  With a fixed number of iterations (`max_iter=100`), there is a risk that the model might not fully converge, especially if the dataset is large or complex. This could potentially limit the predictive performance.
- **Feature Representation Constraints:**
  Relying solely on TF-IDF for feature extraction might not capture semantic context as effectively as modern techniques like word embeddings or transformer-based models. This could affect the quality of the learned representations for nuanced text data.

6.3 Naive Bayes

6.3.1 Overview

Naive Bayes is a probabilistic classifier based on Bayes' Theorem with an assumption of feature independence. It is particularly effective for text classification tasks due to its simplicity and efficiency. The mathematical formulation of Bayes' Theorem is as follows:

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)}$$

where:
- $P(C_k|X)$ is the posterior probability of class C_k given data X.
- $P(X|C_k)$ is the likelihood of observing data X given class C_k).
- $P(C_k)$ is the prior probability of class C_k.
- $P(X)$ is the probability of data X (a normalization factor).

Multinomial Naive Bayes for Text Classification For text classification, we use the multinomial distribution:

$$P(x_i|C_k) = \frac{N_{x_i,C_k} + \alpha}{N_{C_k} + \alpha d}$$

where:
- N_{x_i,C_k} is the count of word x_i in class C_k.
- N_{C_k} is the total number of words in class C_k
- d is the vocabulary size.
- α is the Laplace smoothing parameter.

6.3.2 Key Parameters and Their Impact

The classifier is tuned using GridSearchCV on the alpha parameter:

```
MultinomialNB(alpha=best_alpha)
```

- **alpha:** This is the smoothing parameter (Laplacian smoothing). A small alpha prevents zero probabilities for unseen words, while a large alpha smooths too aggressively.
- The best *alpha* is determined via cross-validation (cv=5), optimizing accuracy.
- After tuning, the final model is trained on the entire dataset with the best alpha value.

6.3.3 Feature Importance Analysis

Since Naive Bayes is a probabilistic model, feature importance can be interpreted based on the learned conditional probabilities $P(X|C)$:

- Words that are strongly associated with a particular class will have higher conditional probabilities in that class.
- The impact of each feature (word) depends on its frequency and how distinctively it appears in a single category.
- In text classification, stopwords and common words contribute less, whereas category-specific words have a stronger influence.

6.3.4 Strengths and Limitations

- **Strengths**
  - *Fast and efficient:* Training and inference are computationally inexpensive.
  - *Handles high-dimensional data well:* Works effectively for text classification.
  - *Interpretable:* Probabilities provide insights into the importance of words.
- **Limitations**

- *Strong independence assumption:* Assumes words appear independently which is often unrealistic in natural language.
- *Performance on complex patterns:* Struggles with tasks where feature dependencies are important.
- *Sensitivity to rare words:* If a word appears very few times, it may not contribute effectively to classification.

6.4 Graphical Model

6.4.1 Overview

Graphical models are probabilistic models that use graphs to represent dependencies between variables, making them well-suited for structured prediction tasks. We conduct our topic on two widely used graphical models: Bayesian Network (BN) and Hidden Markov Model (HMM). These models provide a structured way to model relationships between words and categories, levaraging the dependencies rather than assuming complete independence as in Naive Bayes.

- **Bayesian Network (BN):** A directed acyclic graph (DAG) where nodes represent variables (words, document categories,...) and edges encode probabilistic dependencies. The joint probability distribution over all variables is factorized as:

  $$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i | Pa(X_i))$$

  where $Pa(X_i)$ represents the parent nodes of X_i in the DAG.

- **Hidden Markov Model (HMM):** A sequential probabilistic model where observations (words) are generated by the hidden states (topics or categories). The probability of an observation sequence $O = (o_1, o_2, ..., o_T)$ given a hidden state sequence $S = (s_1, s_2, ..., s_T)$ is computed as:

  $$P(O, S) = P(s_1) \prod_{t=2}^{T} P(s_t | s_{t-1}) \prod_{t=1}^{T} P(o_t | s_t)$$

  where:
  - $P(s_1)$ is the initial state probability,
  - $P(s_t | s_{t_1})$ is the state transition probability,
  - $P(o_t | s_t)$ is the observation likelihood.

6.4.2 Key Parameters and Their Impact

6.4.2.a Bayesian Network

```python
structure = [('Category', f'word_{i}') for i in range(X_disc.shape[1])]
self.model = BayesianNetwork(structure)
self.model.fit(df, estimator=MaximumLikelihoodEstimator)
```

1. **Structure Learning (BayesianNetwork(structure)**

   - **Definition:** Structure learning defines the graphical representation of dependencies between variables (features). The structure can be predefined based on domain knowledge or learned from data.

- **Impact on Classification:**
  - A well-designed structure improves classification accuracy by capturing essential feature dependencies.
  - If the structure is too simplistic, the model may miss critical relationships, leading to poor generalization.
  - An overly complex structure may lead to overfitting, making the model less robust to new data.

2. **Conditional Probability Tables (CPTs)**(fit(df, estimator=MaximumLikelihoodEstimator))

- **Definition:** CPTs store the conditional probabilities of each node given its parent nodes which are estimated from the training data.
- **Estimator choices:**
  - We choose Maximum Likelihood Estimator (MLE) to compute probabilities directly from observed data and it works well when the dataset is large.

3. **Inference Algorithm** (VariableElimination(self.model))

- **Definition:** Bayesian Networks support different inference techniques, such as exact and approximate methods.
- **Impact:**
  - Variable Elimination: computes exact probabilities by marginalizing out irrelevant variables. Moreover, it is suitable for small to medium-sized models.
  - We use Variable Elimination for inference due to its accuracy. If scalability becomes an issue, approximate methods should be considered.

6.4.2.b Hidden Markov Model

```
model = hmm.GaussianHMM(n_components=4, covariance_type="full", tol=1e-3, init_params="
    stmc", verbose=True)
model.fit(X_train_scaled[idx])
```

1. **Number of Hidden States (n_components)**

- **Definition**: Specifies the number of hidden states in the HMM.
- **Impact**: A higher number of components allows the model to capture more complex structures but may lead to overfitting if not properly tuned.

2. **covariance_type**

- **Definition**: Determines the structure of covariance matrices used in the Gaussian emissions.
- **Options**:
  - `"full"`: Provides the most flexibility but requires more parameters to estimate.
  - `"tied"`: All components share the same covariance matrix.
  - `"diag"`: Each component has a diagonal covariance matrix.
  - `"spherical"`: Each component has a single variance value.
- **Impact**:
  - `"full"`: Provides the most flexibility but requires more parameters to estimate.

    - `"diag"` and `"spherical"`: Reduce computational complexity at the cost of modeling accuracy.

3. **tol**
    - **Definition**: The convergence threshold for the log-likelihood.
    - **Impact**: A smaller `tol` value increases the number of iterations required for convergence, while a higher `tol` may result in premature convergence and suboptimal solutions.

4. **verbose**
    - **Definition**: Controls the verbosity of output during model training.
    - **Impact**:
        - `True`: Displays detailed logs, useful for debugging and monitoring convergence.
        - `False`: Suppresses logs, leading to cleaner output during execution.

6.4.3 Feature Importance Analysis

6.4.3.a Bayesian Network

In Bayesian Network, feature importance is inferred from the structure of the network, which represents probabilistic dependencies between variables. In our model, the Category variable is the parent node, and each word feature (word 0, word 1,..., word n) is directly dependent on it. This structure reflects the direct influence of the categorical label on the distribution of words within the dataset. We can evaluate the feature importance based on:

1. **Mutual Information:** Higher mutual information between a word feature and the category suggests that the word is highly indicative of the class.
2. **Conditional Probability Distributions:** The probability of a specific word feature given a category helps identify key discriminative words.
3. **Sensitivity Analysis:** Observing changes in classification outcomes when certain features are perturbed or removed.

Our analysis showed that domain-specific words had higher importance scores, meaning they significantly influenced the classification outcome. For example, words like "*government*" and "*election*" were strongly associated with the *World* category, while "*match*" and "*goal*" were more relevant for *Sports*.

6.4.3.b Hidden Markov Model

HMMs rely on sequential word dependencies. Unlike models use bag-of-words (BoW) or TF-IDF, HMMs assume an underlying state transition structure. The importance of features in HMMs arises from:

- **Word Sequences:** The probability of transitioning from one word to another.
- **Emission Probabilities:** High-probability word sequences help determine class labels.
- **State Transitions:** The likelihood of moving between hidden states influences classification.

If feature selection removes key sequential dependencies, the model loses predictive power.

6.4.4 Strengths and Limitations

6.4.4.a Bayesian Network

- **Strengths**
  - **Interpretable Structure:** The probabilistic dependencies in Bayesian Networks make it easy to understand relationships between features and target categories.
  - **Handle Uncertainty Well:** Unlike deterministic models, Bayesian Networks provide a probabilistic output, allowing confidence estimation for each prediction.
  - **Data Efficiency:** Performs well even with limited training data by leveraging prior probabilities and conditional dependencies.
  - **Resilience to missing data:** Since it models conditional probabilities, it can make reasonable inferences even with partial feature information.
- **Limitations**
  - **Computational Complexity:** Exact inference in Bayesian Networks can be expensive, especially as the number of features increases.
  - **Discretization Sensitivity:** The discretization process in our model (binning continuous TF-IDF features) can lead to information loss if not tuned properly.

6.4.4.b Hidden Markov Model

- **Strengths**
  - Captures sequential relationships in text.
  - Works well for structured language models (POS tagging, speech recognition). Can model probabilistic transitions, unlike purely count-based models.
- **Limitations**
  - **Poor performance in text classification:** Assumes strict sequence dependencies, making it unsuitable for datasets where order does not play a strong role.
  - **High computational cost:** Large vocabularies require intensive matrix operations.
  - **Sensitivity to parameter initialization:** Poor initializations can lead to suboptimal local minima.
  - **Limited generalizability:** Works best in specific domains with strong sequential structures.

6.5 Support Vector Machine with PCA

6.5.1 Overview

Support Vector Machines (SVMs) are powerful supervised learning algorithms primarily used for classification, with applications in regression and outlier detection. The goal is to find the optimal hyperplane that separates data points of different classes with the maximum margin. For non-linearly separable data, SVMs employ the "kernel trick" to map data into a higher-dimensional space, enabling separation using kernels such as linear, polynomial, or radial basis function (RBF). In our implementation, we use `LinearSVC` from scikit-learn with a linear kernel, optimized for text classification using TF-IDF features (`ngram_range = (1, 2)`, `max_features = 20000`). To handle high-dimensional sparse data, we integrate Truncated Singular Value Decomposition (SVD), a variant of Principal Component Analysis (PCA), to reduce dimensionality while preserving classification accuracy.

Mathematical Formulation: Given a dataset $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$, the hard-margin SVM aims to solve:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1$$

For non-separable data, the soft-margin SVM introduces slack variables ξ_i and a regularization parameter $C = 0.1$:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

The prediction function is:

$$f(x) = \text{sign}(w^T x + b)$$

Algorithm: The SVM training process involves solving the above optimization problem, typically using quadratic programming. The high-level algorithm is outlined as follows:

[1] **Input:** Dataset $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$, parameter C, maximum iterations (`max_iter = 5000`). Apply TF-IDF vectorization (`ngram_range = (1, 2)`, `max_features = 20000`). Optionally apply Truncated SVD to reduce dimensionality to k components, retaining 95% variance. Initialize weight vector w and bias b. Solve the optimization problem:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Use quadratic programming or iterative methods (e.g., SMO) to find optimal w, b, and support vectors. **Output:** Trained model parameters w, b for prediction $f(x) = \text{sign}(w^T x + b)$.

Algorithm 1: SVM Training with Linear Kernel

6.5.2 Key Parameters and Their Impact

The performance of an SVM model is significantly influenced by its parameters, which require careful tuning to achieve optimal results. Below are the key parameters and their effects:

- **C (Regularization Parameter)**: Controls the trade-off between maximizing the margin and minimizing classification error. In the context of high-dimensional TF-IDF features, an optimal C value helps mitigate overfitting by penalizing misclassified points. A lower C tolerates misclassifications and supports better generalization; a higher C fits the training data more strictly, potentially overfitting sparse text features.

- **Kernel**: Determines the function used to project data into a higher-dimensional space. In your case, a **linear kernel** is used, which is computationally efficient and suitable when TF-IDF or SVD-transformed data is linearly separable. Non-linear kernels (e.g., RBF) are unnecessary when dimensionality reduction (e.g., PCA or SVD) already captures the essential variance for linear separation.

- **Gamma**: Relevant only for non-linear kernels (e.g., RBF, sigmoid). Since your model uses a linear kernel, this parameter is not active. However, if switching to RBF after PCA/SVD, γ would control the curvature of the decision boundary.

- **Degree**: Specific to the polynomial kernel and irrelevant for your current linear setup. It would define the polynomial degree if used.
- **TF-IDF Vectorization**: Converts text into a weighted numerical feature space. Using bigrams (`ngram_range=(1, 2)`) and limiting the vocabulary size (`max_features=20000`) helps capture informative phrase patterns while controlling dimensionality.
- **Truncated SVD (Linear PCA Equivalent)**: Applied after TF-IDF to reduce feature space dimensionality while retaining 95% of variance. This helps alleviate the curse of dimensionality, reduces training time, and may improve generalization, especially when used before linear SVM training.

6.5.3 Feature Importance Analysis

In this pipeline, feature importance analysis must consider both the linear SVM model and the dimensionality reduction applied via Truncated SVD:

- **Linear SVM**: Feature importance is derived from the absolute values of the weights in the vector w, accessible via the `coef_` attribute. These weights correspond to the transformed components after dimensionality reduction. Hence, they reflect the importance of the SVD components, not the original TF-IDF features.
- **Truncated SVD (PCA-like)**: Since SVD projects the original TF-IDF features into a lower-dimensional space, direct interpretation of individual original features is obscured. However, by examining the right singular vectors (from `components_` attribute in scikit-learn's SVD implementation), one can approximate how each original feature contributes to the principal components. Multiplying SVM weights with the SVD components can offer an approximate mapping back to the original TF-IDF feature space for interpretability.
- **Alternative Techniques**: For a more accurate feature ranking in the original TF-IDF space, one can apply:
  - **Permutation Importance**: This method evaluates the impact of shuffling each original feature on model performance, even after dimensionality reduction.
  - **Recursive Feature Elimination (RFE)**: When applied before SVD, RFE can help identify and retain the most informative TF-IDF features prior to dimensionality reduction and classification.

6.5.4 Strengths and Limitations

6.5.4.a Strengths

- **Effective in High-Dimensional Spaces**: SVMs perform well in applications like text classification and bioinformatics, where the number of features is large .
- **Handles Non-linear Data**: Through kernel functions, SVMs can effectively manage non-linear relationships in data.
- **Robust to Overfitting**: The margin maximization principle helps SVMs generalize well, particularly in high-dimensional spaces.
- **Memory Efficient**: SVMs use only a subset of training points (support vectors) in the decision function, reducing memory requirements.

- **Versatile Applications**: SVMs are used in diverse domains, including image recognition, text categorization, and real-time systems .

6.6 Limitations

- **Computationally Intensive**: Training SVMs involves solving a quadratic programming problem, which can be slow for large datasets.
- **Parameter Tuning**: The choice of kernel and parameters like C and gamma significantly affects performance and requires careful tuning, often through computationally expensive methods like grid search.
- **Not Suitable for Large Datasets**: SVMs scale poorly in terms of time and memory for very large datasets .
- **Sensitive to Noise**: Outliers and noisy data can significantly impact the position of the hyperplane, affecting model performance.
- **Lack of Probabilistic Outputs**: SVMs do not provide direct probability estimates, requiring additional methods like Platt scaling, which adds complexity.

6.7 Ensemble Models

6.7.1 Overview

6.7.1.a Bagging

Bagging trains B models independently and averages their predictions to reduce variance:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{(b)}(x)$$

Where $\hat{f}^{(b)}$ is the prediction from the b-th base model trained on a bootstrap sample.

6.7.1.b Boosting

Boosting constructs a strong classifier from several weak ones. Each weak learner is trained on data reweighted to focus on previous errors. For AdaBoost:

$$F(x) = \sum_{m=1}^{M} \alpha_m h_m(x)$$

Where:

- $h_m(x)$: prediction of the m-th weak learner
- α_m: weight assigned to h_m based on its accuracy

In our model, $h_m(x)$ is a voting ensemble, and weighting also applies within the voting mechanism as:

$$P(y|x) = \sum_{i=1}^{3} w_i P_i(y|x)$$

with $w = [2, 1, 1]$.

6.7.2 Key Parameters

6.7.2.a Bagging

- `n_estimators = 50`: more estimators reduce variance but increase computational cost.
- `VotingClassifier (soft)`: combines model outputs by probability, offering smoother decisions.

6.7.2.b Boosting

- `n_estimators = 10`
- `weights = [2, 1, 1]`
- `DecisionTree max_depth = 100`

6.7.3 Feature Importance Analysis

In ensemble models built on interpretable base learners like Logistic Regression and Naive Bayes, feature importance can be interpreted from the underlying models:

- **Multinomial Naive Bayes:** Feature importance is derived from conditional probabilities $P(x_i|y)$. Words with higher conditional likelihood for a particular class are more influential.
- **Logistic Regression:** Feature importance can be extracted from the magnitude of learned weights w_i. Features with higher absolute weight values contribute more strongly to the classification decision.
- **Decision Trees:** Feature importance is computed based on information gain or Gini impurity reduction. The features that best split the data appear closer to the root and are considered more important.

In our ensemble, feature importance is distributed across these models. Although individual feature interpretations may be complex due to ensemble aggregation, we can still analyze base learners separately for insights.

6.7.4 Strength and Limitations

6.7.4.a Bagging

Strengths:

- Reduces model variance and overfitting through bootstrap sampling.
- Robust to noise due to averaging of predictions.
- Naturally parallelizable.
- Combines diverse classifiers, improving stability.

Limitations:

- Cannot reduce bias if base models are weak.
- Requires more memory and computation due to multiple models.
- Interpretability is reduced in comparison to single models.

6.7.4.b Boosting

Strengths:

- Corrects mistakes iteratively, reducing both bias and variance.
- Focuses on hard-to-classify samples, improving decision boundaries.
- Can outperform other models in terms of accuracy on clean datasets.

Limitations:

- More sensitive to noise and outliers due to aggressive correction.
- Computationally slower due to sequential training.
- May overfit if number of estimators is too high.
- Requires careful tuning of base model complexity and weights.

6.8 Discriminative Models

6.8.1 Overview

Discriminative models directly learn the boundary between classes by modeling the conditional probability $P(y \mid x)$. They contrast with generative models, which estimate joint distributions $P(x, y)$ and apply Bayes' theorem. Feature-based linear classifiers such as Logistic Regression and Linear SVM fall into this category, as they learn weights that linearly combine input features to separate categories.

$$P(y = k \mid x) = \frac{\exp(w_k^\top x + b_k)}{\sum_j \exp(w_j^\top x + b_j)}.$$

We preprocess text using a TF–IDF vectorizer, reduce dimensionality with χ^2-based feature selection, scale the data, and fit an L_2-penalized multinomial logistic regression with the `saga` solver for large-scale sparse inputs.

6.8.2 Key Parameters and Their Impact

`clf_C` (Inverse Regularization Strength) Controls the L_2 penalty weight. Smaller values impose stronger regularization, driving coefficients toward zero and reducing overfitting at the cost of underfitting if set too low. Larger C allows more complex models but risks overfitting noisy features.

`clf_penalty` Specifies the norm used for regularization. We fix `penalty='l2'` to ensure convexity and stable convergence with high-dimensional sparse vectors.

`max_features` in `TfidfVectorizer` Limits the vocabulary size. Lower values reduce computational cost and overfitting, but too few features may discard informative terms.

`k_best` in `SelectKBest` Number of top features selected by the χ^2 test. Balances between dimensionality reduction (speed & noise reduction) and retention of discriminative terms.

`max_iter=1000` ensures convergence; raising it can address warnings about reaching iteration limits.

6.8.3 Feature Importance Analysis

After training, feature coefficients w_{kj} indicate the importance of term j for class k. Positive weights favor the class when the corresponding term is present, while large negative weights disfavor it. A typical analysis:

1. Extract the top-10 features for each category by sorting w_k in descending order.
2. Inspect terms like "government" and "election" for **World**, "game" and "team" for **Sports**, financial keywords for **Business**, and technical jargon for **Sci/Tech**.

6.8.4 Strengths and Limitations

- **Strengths**
  - *Interpretability:* Coefficients directly map to term importance.
  - *Scalability:* Linear time scaling with number of features; handles sparsity efficiently.
  - *Convexity:* Guarantees global optimum for fixed hyperparameters.
  - *Low Memory Footprint:* Works on sparse representations with minimal overhead.
- **Limitations**
  - *Linearity:* Cannot capture non-linear interactions between features.
  - *Feature Engineering Dependence:* Requires careful text preprocessing and selection to remove noise.
  - *Limited Handling of Polysemy:* TF–IDF bag-of-words ignores context beyond single terms.
  - *Sensitivity to Hyperparameters:* Over-regularization or too aggressive feature reduction can degrade performance.

7 Usage Example

The trained pipeline is applied to classify new, unseen text samples. This example:

- Demonstrates the practical application of the model in news classification.
- Provides a reference for future use and model extensions.

Example Predictions:

```
new_samples = [
    "Wall Street sees renewed optimism in tech stocks",
    "Soccer World Cup final brings excitement worldwide"
]
Predicted Label: [
    "Business",
    "Sports"
]
```

This showcases the model's ability to generalize and predict meaningful labels based on text input.

8 Experimental Results

To assess the effectiveness of each model, we will conduct comparative evaluations between models using the same use-case on the introduced datasets across various settings.

8.1 Evaluation Method

For the specific problem of classification based on text, several metrics are used to evaluate the model's performance. In this thesis report, the presenter uses 4 metrics to evaluate the experimental results on our target dataset: *Precision*, *Recall*, *F1 Score*, and *Accuracy*.

The confusion matrix (or error matrix) is a table designed to visualize the results of an algorithm. Because accuracy alone cannot determine the actual performance of classification system if the dataset is unbalanced. To test the detection function and the conditions of the confusion matrix are shown in Table 2.

Condition	**Interpretation**
True Positive (TP)	Abnormal points correctly detected as actual abnormal points.
True Negative (TN)	Non-abnormal points correctly detected.
False Positive (FP)	Non-abnormal points incorrectly predicted as abnormal.
False Negative (FN)	Abnormal points not detected.

Table 2: Conditions and explanations of values in the confusion matrix

- **Accuracy**: The most intuitive measure, it is simply the ratio of the correctly predicted observations to the total observations. Although widely used, accuracy is not the most appropriate measure in some cases, especially when the classes in the dataset are imbalanced.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

- **F1 Macro score**

  The F1 score considers both precision and recall to compute the model's performance. Mathematically, it is the harmonic mean of the model's precision and recall, computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

  This F1 score is computed across different classes, taking the average of the F1 scores of all classes to obtain the average F1 score.

8.2 Experimental Results and Discussion

Table 3 presents the experimental evaluation results over implemented models and the baseline models on the **AG News** dataset.

Rank	**Model**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
1	SVM	0.9128	0.9124	0.9128	0.9124
2	Ensemble - Bagging	0.90	0.90	0.90	0.90
3	SVM with PCA	0.8951	0.8947	0.8951	0.8947
4	Discriminative	0.88	0.88	0.88	0.88
5	Neural Network	0.8762	0.88	0.88	0.88
6	Naive Bayes	0.8523	0.85	0.85	0.85
7	Ensemble - Boosting	0.85	0.85	0.85	0.85
8	Bayes Network	0.8433	0.84	0.84	0.84
9	Decision Tree	0.7996	0.80	0.80	0.80
10	HMM	0.6514	0.67	0.65	0.66

Table 3: Results on the **AG News** dataset

8.2.1 Neural Network (MLP)

The Neural Network demonstrated the best classification results across all metrics (*Accuracy* = 0.8762, *Precision* = 0.88, *Recall* = 0.88, *F1 Score* = 0.88). The MLP classifier benefits from non-linear learning capabilities, which can lead to improved performance on complex tasks such as multi-class text classification (e.g., AG News). Its ability to capture complex, non-linear patterns in text data—especially when coupled with TF-IDF features—contributed to robust generalization. However, despite its strong performance, interpretability remains a challenge. The model's performance is highly dependent on hyperparameter tuning (e.g., learning rate, hidden layer sizes, and regularization) and a fixed number of iterations might not suffice for full convergence on larger datasets.

8.2.2 Naive Bayes

Naive Bayes (*Accuracy* = 0.8523, *Precision* = 0.85, *Recall* = 0.85, *F1 Score* = 0.85) ranked second, confirming its reputation as a fast and effective baseline for text classification. Naive Bayes is computationally efficient and well-suited for high-dimensional text data. Its probabilistic nature offers a degree of interpretability, as the importance of each word is derived from its learned conditional probability. Although it assumes feature independence—which is rarely true in natural language—its simplicity and low computational overhead make it appealing for large-scale or real-time classification tasks. However, the strong independence assumption may limit its effectiveness on tasks where word dependencies are critical, and rare words might not contribute effectively. Moreover, the probabilistic framework allows a degree of interpretability by examining the conditional probabilities for each word.

8.2.3 Bayesian Network

The Bayesian Network approach achieved an *Accuracy* of 0.8433 with balanced *Precision*, *Recall*, and *F1 Score* of 0.84. Bayesian Networks excel in interpretability due to their explicit representation of probabilistic dependencies among features and labels. Its key advantage lies in modeling dependencies among words and the target category, providing clearer interpretability. However, learning and inference can become computationally expensive for high-dimensional text data, especially if the network structure grows complex. Discretizing continuous features (e.g., TF-IDF scores) may introduce information loss if not done carefully.

8.2.4 Decision Tree

The Decision Tree classifier attained an *Accuracy* of 0.7996, *Precision* = 0.80, *Recall* = 0.80, and *F1 Score* = 0.80. Decision Trees excel in interpretability, offering transparent, rule-based classifications. While Decision Trees are non-parametric and do not assume any underlying data distribution, they are prone to overfitting, especially when the tree grows too deep. Furthermore, the high dimensionality introduced by the vectorization of TF-IDF can complicate the decision boundaries and be more susceptible to noise and borderline misclassifications, limiting the generalization capacity of the model compared to ensemble approaches such as Random Forest.

8.2.5 Graphical Models

The HMM yielded the lowest performance (*Accuracy* = 0.6514, *F1 Score* = 0.66). While HMMs are powerful for tasks involving strong sequential dependencies, they are less effective for general news classification, where strict word-order patterns are less critical. Their reliance on state transitions and emission probabilities can lead to higher complexity and reduced scalability in broad-domain text classification. They also require careful parameter initialization and can be computationally expensive for large vocabularies.

8.2.6 Ensemble Models

We evaluate both models using Accuracy, Precision, Recall, and F1-score, averaged across four classes: World, Sports, Business, and Sci/Tech.

8.2.6.a Bagging

- Accuracy: 0.90
- Precision: 0.90
- Recall: 0.90
- F1-score: 0.90

Bagging performed well by aggregating diverse models and reducing variance. Soft voting enhanced performance by integrating probability outputs. This approach is robust, especially in high-dimensional text classification.

8.2.6.b Boosting

- Accuracy: 0.85
- Precision: 0.85
- Recall: 0.85
- F1-score: 0.85

Boosting achieved slightly better results by sequentially correcting mistakes. The internal weighting of base learners helped adaptively focus on strong predictors. Logistic Regression, with higher weight, contributed significantly.

8.2.7 Discriminative Models

The Logistic Regression classifier attained an Accuracy of 0.88, Precision = 0.88, Recall = 0.88, and F1 Score = 0.88. Logistic Regression excels in interpretability—its learned coefficients directly reflect each feature's contribution to a class decision—and in scalability, handling high-dimensional, sparse TF–IDF inputs efficiently with a convex optimization that guarantees a global optimum. Its linear decision boundary, coupled with L_2-regularization, helps prevent overfitting even when tens of thousands of textual features are present. However, the model's reliance on a purely linear separation means it cannot capture complex, non-linear feature interactions or contextual nuances in language. Moreover, performance hinges on careful feature engineering (e.g., TF–IDF settings and χ^2 selection) and proper regularization strength; overly aggressive feature reduction or mis-tuned C can lead to underfitting or residual overfitting, limiting its ability to generalize relative to non-linear or ensemble approaches.

8.2.8 Support Vector Machines (SVM) and SVM with PCA

The standard SVM classifier achieved an Accuracy of 0.9128, Precision of 0.9124, Recall of 0.9128, and F1 Score of 0.9124. SVM is known for its strong theoretical foundation in maximizing the margin between classes, which enhances generalization performance. Its ability to handle high-dimensional data efficiently makes it suitable for text classification tasks involving TF–IDF features. The linear kernel, combined with the regularization parameter C, helps control the trade-off between margin maximization and classification error, thereby reducing the risk of overfitting.

When applying PCA for dimensionality reduction before SVM training, the model achieved slightly lower metrics: Accuracy of 0.8951, Precision of 0.8947, Recall of 0.8951, and F1 Score of 0.8947. PCA reduces feature dimensionality by projecting data into a lower-dimensional subspace that preserves the majority of variance, which can improve computational efficiency and mitigate multicollinearity. However, this dimensionality reduction may discard some discriminative information crucial for classification, leading to the observed decrease in performance.

In practice, the decision to use PCA with SVM involves balancing computational resources and accuracy requirements. While PCA can speed up training and reduce model complexity, it risks losing subtle feature interactions captured by the full feature set, thereby slightly impairing the model's discriminative power.

8.2.9 Overall Discussion

The experimental results demonstrate that no single model dominates across all performance metrics, and each classifier offers distinct trade-offs:

- **Support Vector Machine** (SVM) delivered the highest overall performance among all models, achieving an Accuracy of 0.9128 along with balanced Precision (0.9124), Recall (0.9128), and F1-Score (0.9124). SVM's effectiveness lies in its ability to construct optimal hyperplanes in high-dimensional spaces, making it particularly well-suited for sparse, text-based features. It performs well even when the data is not linearly separable, thanks to the use of kernel functions.
- **Support Vector Machine with PCA** (SVD) also demonstrated strong performance, with Accuracy at 0.8951, and Precision, Recall, and F1-Score all at 0.8947. The dimensionality reduction

using SVD before classification helped mitigate the curse of dimensionality and reduced training time, while still preserving most of the data's variance. Although the slightly reduced performance compared to the standard SVM suggests some loss of information during the compression process, the model still maintained high discriminative power. This trade-off between computational efficiency and marginal accuracy drop makes SVM with SVD a practical choice for resource-constrained environments.

- **Discriminative Model** (Logistic Regression) achieved the overall performance with Accuracy, Precision, Recall, and F1 Score all at 0.88. Its strength lies in scalability and interpretability, particularly in handling high-dimensional, sparse textual data using linear decision boundaries. However, it assumes linear separability and depends heavily on proper feature engineering and regularization.
- **Neural Network** followed closely with an Accuracy of 0.8762 and similarly strong precision and recall. It is highly flexible and capable of capturing complex, non-linear relationships, but its performance depends on extensive hyperparameter tuning and it lacks the transparency of simpler models.
- **Naive Bayes** provided a balanced performance (Accuracy = 0.8523), performing well despite its simplifying assumption of feature independence. Its computational efficiency and robustness to irrelevant features make it a strong baseline for text classification.
- **Bayesian Network** scored slightly lower (Accuracy = 0.8433) but offers a structured probabilistic interpretation of feature relationships. While more expressive than Naive Bayes, it is more complex to construct and less scalable on large feature spaces.
- **Decision Tree** achieved moderate performance (Accuracy = 0.7996) and is appreciated for its human-readable rule-based structure. However, it is prone to overfitting, especially in high-dimensional spaces like TF–IDF vectors, and typically benefits from pruning or ensemble extensions.
- **Hidden Markov Model (HMM)** had the lowest performance (Accuracy = 0.6514), indicating it is less suitable for static text classification. HMMs are better suited for sequential or time-dependent data and their assumption of state transitions makes them poorly aligned with bag-of-words models.
- **Ensemble Models** achieved solid performance (Accuracy $>$ 0.85), benefiting from model variance reduction through bootstrap sampling and soft voting. Its robustness and diversity of base learners contributed to consistent predictions across categories. While not the most interpretable, Bagging is a reliable choice when overfitting is a concern, especially in high-dimensional text classification.

9 Conclusion

This project implementation for a 4-label text classification task exemplifies robust machine learning engineering practices. The comprehensive approach—including data preprocessing, model tuning, performance evaluation, and feature importance analysis—aligns well with the project requirements, demonstrating the integration of theoretical knowledge and practical skills.

Furthermore, this project applies the models learned in the Machine Learning course to a specific classification task, ensuring consistency in the use case across different algorithms. By evaluating various models under the same conditions, we assess their effectiveness in handling text classification problems. This structured approach highlights the strengths and limitations of each model, reinforcing key concepts such as interpretability, generalization, and optimization.

Through this implementation, we bridge theoretical learning with real-world application, strengthening our understanding of machine learning techniques and their impact on practical tasks.

References

[1] Hirsh, H. Generalizing version spaces. Machine Learning, 1994

[2] L.Breiman, J.Friedman, R.Olshen and C.Stone. Classification and Regression Trees, 1984.

[3] Haykin, S. Neural Networks and Learning Machines 3rd Edition, 2008.

[4] Devroye, L., Gyorfi, L., Lugosi. A Probabilistic Theory of Pattern Recognition, 1996.

[5] Koller, Friedman. Probabilistic Graphical Models: Principles and Techniques, 2009.

[6] Bishop, C.M. Pattern Recognition and Machine Learning, 2006.

[7] Hastie, Tibshirani, Friedman. The Elements of Statistical Learning 2nd Edition, 2009.

[8] Scikit-learn Developers, "Support Vector Machines," Scikit-learn 1.6.1 Documentation, https://scikit-learn.org/stable/modules/svm.html.

[9] GeeksforGeeks, "Determining Feature Importance in SVM Classifiers with Scikit-Learn," https://www.geeksforgeeks.org/determining-feature-importance-in-svm-classifiers-with-scikit-learn/.

[10] Dhiraj K, "Top 4 Advantages and Disadvantages of Support Vector Machine or SVM," Medium, https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107.