Comparative Analysis of Machine Learning Models for Bank Marketing and YouTube Spam Detection

Jiaqi Yin (jyin90)

***Index Terms*—Classification, NLP, Transformer,TF-IDF, KNN, SVM, NN, GBT,**

I. Introduction

In this report, I implemented binary classification models on two types of datasets: Bank Marketing Campaign Subscription in Sec. II and YouTube comment spam detection in Sec. III. The first one focus on the tabular data with mixed feature types, while the second one focus on the text data. For each dataset, I conducted data preprocessing, model implementation, hyperparameter tuning, and model evaluation. The goal is to compare the performance of different classification algorithms and understand how they cope with different data types and characteristics.

II. Dataset 1: Bank Marketing Campaign Subscription

A. Data Description and Rationale

The dataset, obtained from Kaggle [1], contains records of marketing phone calls from a Portuguese banking institution. The calls were conducted to persuade bank customers to subscribe to a specific financial product. For each call, customers indicated their willingness to subscribe, serving as an indicator of campaign success.

This dataset presents several interesting characteristics that make it a compelling choice for my analysis:

- Binary Classification Problem: The dataset presents a clear binary outcome (subscription: yes/no), allowing for straightforward evaluation of classification algorithms.
- Large Sample Size: With 41,188 records, the dataset provides ample data for training and testing, enabling more robust model evaluation.
- Class Imbalance: The dataset is imbalanced (yes: 11.3%, no: 88.7%), presenting an additional challenge for model training and evaluation.
- Diverse Feature Types: The dataset contains both numerical and categorical features, requiring the application of various preprocessing techniques.
- Real-World Application: As real-world data from the banking industry, this dataset has practical applications and relevance to current business challenges.

The dataset comprises 41,188 phone call records with 20 features, covering multiple aspects of the customers, such as demographics (e.g., age, job, education), previous campaign contact and outcome, and socio-economic context. The target variable is the campaign outcome: 'yes' or 'no' to subscribing to the product.

B. Hypothesis and Analysis Objectives

Based on the characteristics of this dataset, I hypothesize that:

- Certain demographic features (such as age and job) will be strong predictors of subscription likelihood.
- The class imbalance may pose challenges for some algorithms, potentially requiring techniques like class weighting or resampling.

My analysis will focus on:

- Implementing and comparing the performance of K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Neural Network (NN), and Gradient Boosting Tree (GBT) on this binary classification task.
- Evaluating the impact of various preprocessing techniques, particularly on handling the mixed feature types and class imbalance.
- Analyzing the learning curves and model complexity to understand how each algorithm copes with this large, imbalanced dataset.
- Investigating feature importance and performing through feature reduction.
- Leverage the minority oversampling technique for 'yes' group.

C. Data Preprocessing and Exploration

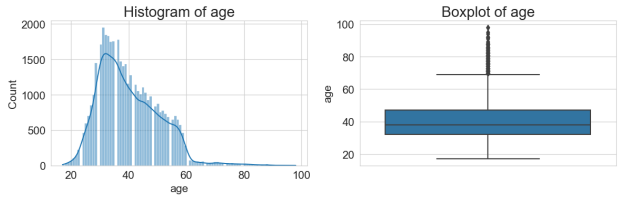
Of the 20 features, 10 are numerical and 10 are categorical. The numerical feature *pdays* (number of days since the client was last contacted from a previous campaign) has a value of 999 for clients not previously contacted, which constitutes the majority. Considering its importance, I converted it into a categorical variable with the following categories: *NoContact, 0-2, 3-7, 8-14,* > 14.

Numerical features contain some outliers that could distort the model, and most are skewed; examples seen in Fig 1. To improve performance, I implemented the following steps:

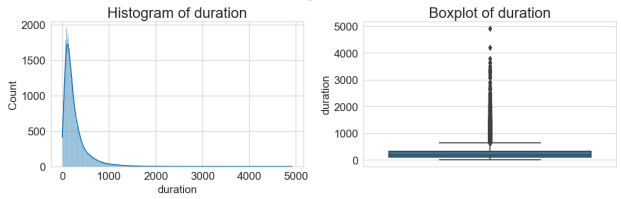
1) Clipped outliers using the IQR method with a 2.0 threshold.
2) Standardized numerical features using StandardScaler.

For categorical features, I applied one-hot encoding to convert them into numerical format.

After the feature preprocess, the dataset contains 67 features. To reduce the feature set, I implemented a feature importance analysis using the Gradient Boosting Tree model which is discussed in Sec. II-E2.



(a) Feature Age Distribution



(b) Feature Duration Distribution

Fig. 1: Numerical feature examples with outliers and skewed distribution. Age is the client's age, and duration is the last contact duration in seconds.

For model training and testing, I split the dataset into a training set and a testing set with a 80:20 ratio. For hyperparameter tuning, I employed five-fold cross-validation on the training set.

D. Model Implementation and Hyperparameter Tuning

I implemented four models using the scikit-learn library. Due to limited computational resources on my local desktop, I had to balance the breadth of hyperparameter exploration with computational feasibility.

1) KNN : I used `KNeighborsClassifier` for KNN implementation. To study the effect of the number of neighbors, I explored the range of $K = \{3, 5, 10, 15, 20, 25, 30\}$ using Euclidean distance. This range was chosen to cover both small and large neighborhood sizes without prior knowledge of the optimal value. I also explored two weight options:

- Uniform: All neighbors have equal weight.
- Distance: Neighbors are weighted by the inverse of their distance, giving closer neighbors greater influence.

2) SVM : I used `sklearn.svm.SVC` for SVM implementation. I explored the following hyperparameters:

- Kernel: 'poly', 'rbf', 'sigmoid'
- γ: 'scale', 'auto', 0.1, 0.01

These kernels are widely used in practice:

- Polynomial (poly): Can model non-linear relationships of higher degree.
- Radial Basis Function (rbf): Effective for non-linear boundaries in many cases.
- Sigmoid: Can produce results similar to certain neural networks.

The γ parameter control the spread of the influence of each point. Lower values leads to a smoother decision boundary, while higher values can lead to overfitting.

3) NN : I used `MLPClassifier` for NN implementation. With 67 total features, I explored various hidden layer configurations:

- One hidden layer: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 neurons
- Two hidden layers: [32, 5], [32, 10], [32, 15], [32, 20], [32, 25], [32, 30], [32, 32] neurons

I used `adam` as the optimization algorithm, `relu` activation for hidden layers. The initial configuration used a constant learning rate of 0.001, batch size of 200, and maximum of 200 epochs. To prevent overfitting, I employed early stopping if the learning metrics not improving after 10 iterations.

Given the importance of learning rate in neural network training, I further explored adaptive learning rates with different initial values: 1.0, 0.5, 0.1, 0.08, 0.06, 0.04, 0.02, 0.01, 0.005, 0.001, 0.0005, 0.0001.

4) GBT : I used `GradientBoostingClassifier` for GBT implementation. I explored the following hyperparameters:

- Number of estimators (boosting stages): 50, 100, 150, 200
- Maximum tree depth: 3, 5, 7, 9
- Subsample ratio: 0.5, 0.7, 0.9, 1.0

These hyperparameters were tuned to balance model complexity and performance. The number of estimators and max depth affect the model's capacity to learn complex patterns, while the subsample ratio helps in reducing overfitting by introducing randomness in the training process.

To verify my hypotheses, I have conducted two cases of analysis:

- Full features without oversampling;
- Oversampling the minority class and reducing features.

In each cases, I have conducted hyperparameter tuning with 5-fold cross-validation. Due to the labeling imbalance, I have used F1-score as the primary metrics for model selection. During the training process, I have also monitored the learning curve to understand the model's performance with different dataset sizes, while except for NN, I monitored the performance with the number of epochs. To understand the model complexity, I have also monitored the performance with different hyperparameters.

E. Results and Analysis

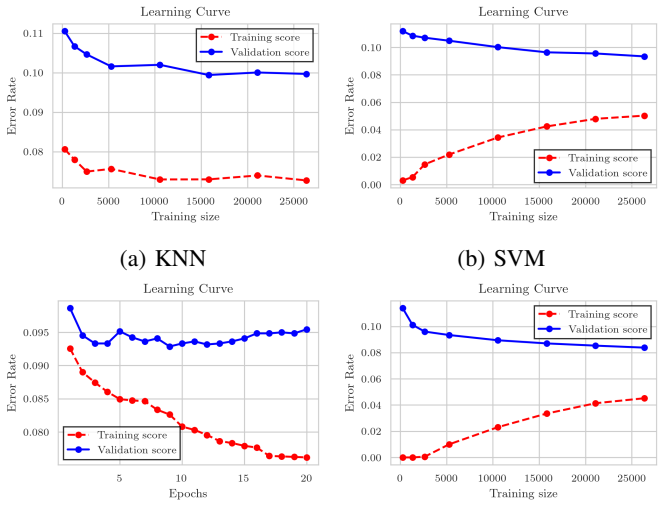
1) Case 1: Full features without oversampling : In the first case, I trained the four different models (KNN, SVM, NN, GBT) on the full feature set without oversampling the minority class. With respective to hyperparameters listed in Sec. II-D, the best hyperparameter for each model in selected with highest F1-score, and the best model is showed in Tab. I. The best model of KNN is with $K = 5$ and uniform weight, SVM is with polynomial kernel and $\gamma = 0.1$, NN is with two hidden layer of [32, 30] and constant learning rate (note that hyperparameters with adaptive learning with inital rate as 0.001 also yield the same validation F1-score), GBT is with max depth 7, 50 estimators, and subsample ratio 0.9. Among

those best models, GBT has the best validation F1-score, while KNN has the worst validation F1-score. It is because KNN is nonparametric model, which does not performance well in high dimension space.

TABLE I: Best model selection in Case 1. It displays the training and validation F1-score, and the best hyperparameters for each model.

Model	Hyperparameters	Training	Validation
KNN	$K = 5$, uniform	0.636	0.477
SVM	kernel=poly, $\gamma = 0.1$	0.746	0.523
NN	[32, 30], lr=0.001	0.611	0.567
GBT	max depth 7, estimator 50, subsample 0.9	0.781	**0.595**

The learning curve of the four best models is showed in Fig. 2. Those curves illustrate how the model's error rate changes as the training set size increases. All four models' learning curves show that the error rate decreases as the training set size / epochs increase, indicating that the model benefits from model data. The training error rate is constantly lower than the validation error rate and the gap between them is large for each model, showing overfitting issue. However, for SVM and GBT, the training and validation error converge with increasing sample size indicates that the two model is making effective use of more data. The relative performance of the models (GBT $>$ NN $>$ SVM $>$ KNN) aligns with their ability to capture complex, non-linear relationships in high-dimensional data.

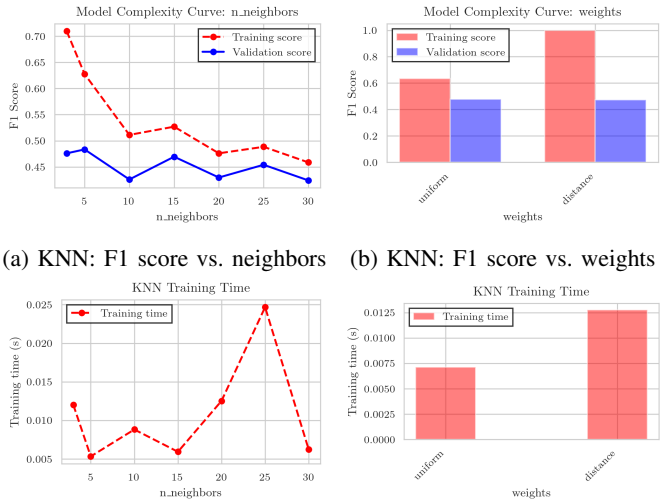


(a) KNN (b) SVM

(c) NN (d) GBT

Fig. 2: Learning curves for KNN, SVM, NN, GBT in Case 1. The x-axis represents the number of samples, and the y-axis represents the error rate.

Fig. 3 shows KNN performance (F1-score) with different number of neighbors and weights. The F1 score generally decreases as the number of neighbors increases. There's a sharp drop in performance after 5 neighbors, followed by a more gradual decline, suggesting that the local patterns in data are more informative than broader trends for this dataset. Uniform weights slightly outperform distance-based weights, indicating that choice of weighting has limited impact on the performance. The training time increases with the number of neighbors, and the distance-based weights require more time to compute than uniform weights. Overall, the relatively low F1 scores (below 0.5) indicate that KNN may not be the most suitable algorithm for this complex, high-dimensional dataset.



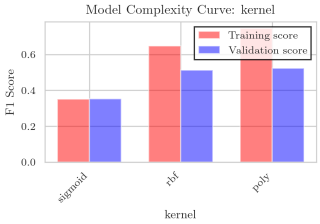
(a) KNN: F1 score vs. neighbors (b) KNN: F1 score vs. weights

(c) KNN: Training time vs. neighbors (d) KNN: Training time vs. weights

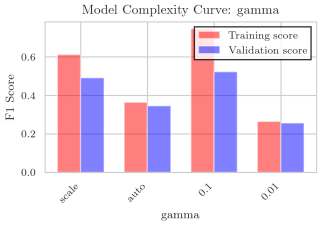
Fig. 3: KNN hyperparameter tuning in Case 1. The x-axis represents the hyperparameter values, and the y-axis represents the F1-score or training time.

Fig. 4 shows SVM F1-score with respect to different kernels and γ. The polynomial (degree = 3) kernel slightly outperforms RBF kernel, and both outperform the sigmoid kernel when $\gamma = 0.1$. This shows that the decision boundary is non-linear. For polynomial kernel, γ as scale or 0.1 have better performance. This suggests that a moderate influence of each training example is beneficial for this dataset, balancing between too localized (high gamma) and too broad (low gamma) impacts. However polynomial kernel takes the longest time to train. In the future, for quick implementation, I may consider using RBF kernel with $\gamma = scale$

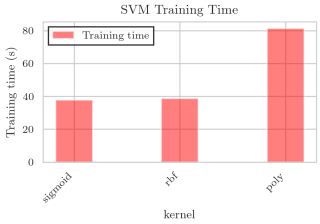
Fig.5 shows the NN performance with respect to hidden layer size and learning rate. With learning rate 0.001, the F1 score increases as the hidden layer size increases, while the relationship is not linear. It indicates that once the number of neurons reaches a certain threshold, the model's performance does not improve significantly. The training time increases with the number of neurons, indicating that larger networks require more computational resources. For learning rate, the performance is generally better with moderate learning rates (around 0.01). Very high learning rates (1.0, 0.5) lead to poor performance, likely missed the optimal weights in training,



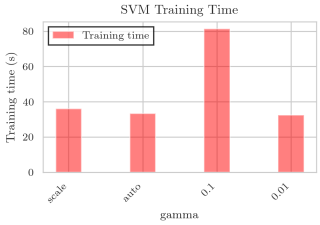
(a) SVM: F1 score vs. kernel



(b) SVM: F1 score vs. γ



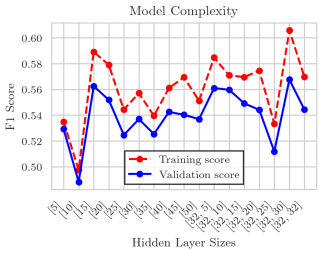
(c) SVM: Training time vs. kernel



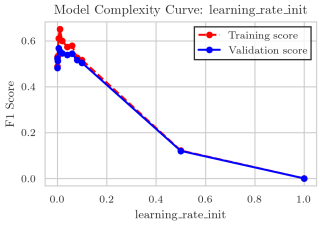
(d) SVM: Training time vs. γ

Fig. 4: SVM hyperparameter tuning in Case 1. The x-axis represents the hyperparameter values, and the y-axis represents the F1-score or training time.

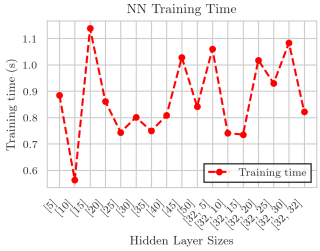
while very low learning rates (0.0001, 0.0005) result in slower convergence and suboptimal performance within the given training time.



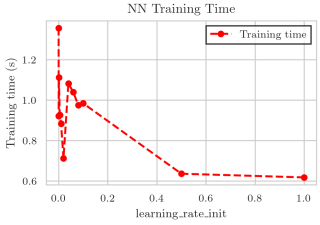
(a) NN: F1 score vs. hidden layer sizes



(b) NN: F1 score vs. learning rate



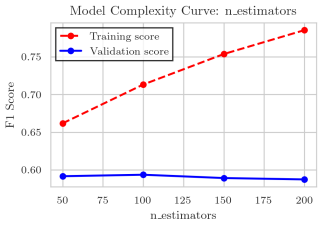
(c) NN: Training time vs. hidden layer sizes



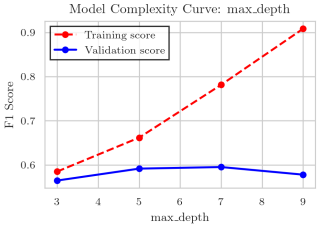
(d) NN: Training time vs. learning rate

Fig. 5: NN hyperparameter tuning in Case 1. The x-axis represents the hyperparameter values, and the y-axis represents the F1-score or training time. Figure 5a has learning rate as 0.001, and Figure 5b has hidden layer sizes as [32, 30].

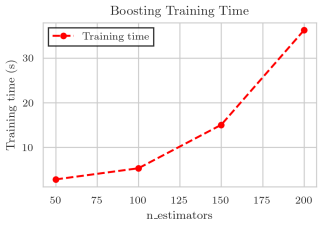
Fig. 6 shows the GBT performance with respect to the number of estimators and max depth. The training F1 score for both hyperparameters increases as the hyperparameter value increases, while the validation F1 score reaches a peak and then decreases. This indicates that the model is overfitting with higher values of these hyperparameters. With 50 estimators and max depth 7, the model achieves the best validation F1 score. This indicates a trade-off between model complexity and generalization ability. Deeper trees can capture more complex patterns in the data, but beyond a certain depth, they may start to overfit. Training time increases linearly with the number of estimators, which is expected as each additional estimator requires training another decision tree. The max depth parameter also affects training time, with deeper trees taking longer to train due to the increased number of splits that need to be evaluated.



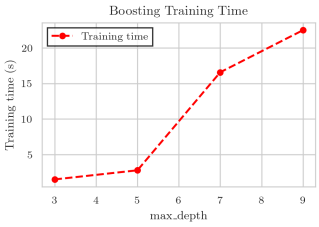
(a) GBT: F1 score vs. n estimators



(b) GBT: F1 score vs. max depth



(c) GBT: Training time vs. n estimators



(d) GBT: Training time vs. max depth

Fig. 6: GBT hyperparameter tuning in Case 1. The x-axis represents the hyperparameter values, and the y-axis represents the F1-score or training time. Fig 6b has max depth as 7 and subsample ratio as 0.9. Fig 6d has n estimators as 50 and subsample ratio as 0.9.

Overall the validation F1 score reaches highest with GBT among all the tested models, indicating that GBT works well with this dataset.

2) Case 2: Feature reduction with GBT and oversampling the minority class : I also leveraged the feature importance from GBT to perform feature reductions. This approach capitalizes on the boosting tree's ability to learn which combination of features significantly reduces the fitting error. The top four features with the importance are showed in Tab. II. Consider the labeling with "yes" takes only 11.3% of the dataset, I also applied the minority oversampling technique to balance the dataset. In this case, I reduced the 20 features to top 14 features, mainly reducing the categorical feature. In the end, after data preprocessing with one-hot encoding, the dataset contains 30 features. The rest of hyperparameter tuning and model selection is the same as Sec. II-E1. The best KNN model is with $K = 3$ and uniform weight; SVM is with

RBF kernel and $\gamma = 0.1$; NN is with one hidden layer of 45 neurons and 0.005 learning rate; GBT is with max depth 5 and 50 estimators.

TABLE II: Feature importance from GBT in Case 2. The table shows the top four features with the highest importance.

Feature	Importance
duration	0.488
nr.employed	0.252
euribor3m	0.086
poutcome_success	0.032

I compared the model performance on the testing set for the two cases, and the results are showed in Tab. III. Generally, the full feature set without oversampling outperforms the reduced feature set with oversampling. GBT and NN tend to have better performance in the full feature cases, likely due to the complexity of the dataset. The oversampling technique may have introduced noise, potentially affecting model performance. KNN consistently shows the worst performance, with F1 scores below 0.5, suggesting that nonparametric models may not be suitable for this complex dataset. Overall, the results indicate that for this particular problem, preserving the full feature set and addressing class imbalance through other means might be more effective than feature reduction and oversampling.

Case No	Model	Accuracy	Precision	Recall	F1
Full features without oversampling	KNN	0.902	0.609	0.419	0.497
	SVM	0.915	0.658	0.494	0.564
	NN	**0.923**	0.665	**0.620**	**0.642**
	GBT	**0.923**	**0.679**	0.591	0.632
Feature reduction with oversampling	KNN	0.898	0.564	0.478	0.517
	SVM	0.911	**0.680**	0.421	0.520
	NN	0.909	0.618	0.549	0.582
	GBT	**0.918**	0.671	**0.554**	**0.607**

TABLE III: Comparison of Model Performance Metrics on Testing Set. The table shows the accuracy, precision, recall, and F1 score for each model in two cases.

III. Dataset 2: YouTube Comments Spam Detection

A. Data Description and Rationale

The YouTube Comments Spam dataset is a public collection comprising 1,956 real comments from five different videos that were among the 10 most viewed during the collection period [2]. This dataset presents an interesting binary classification problem in the domain of natural language processing, with several characteristics that make it a compelling choice for my analysis:

- Balanced Dataset: The dataset contains 951 spam comments and 1,005 non-spam comments, providing a nearly balanced distribution of classes. This balance is crucial for training unbiased models and evaluating their performance accurately.
- Text Classification Challenge: Unlike the Bank Marketing dataset, this problem requires text processing techniques, allowing me to explore a different aspect of machine learning and compare model performances across diverse data types.
- Moderate Sample Size: With 1,956 samples, the dataset is large enough to be meaningful yet small enough to present challenges in model generalization, making it an excellent test for the algorithms' ability to learn from limited data.
- Real-World Application: Spam detection in online platforms is a prevalent issue, making this dataset relevant to current technological challenges.
- Unique User Behavior Patterns: An intriguing aspect of this dataset is that users tend to be consistent in their behavior - they either post all spam or all non-spam comments. This pattern, visualized in Figure 7, adds an interesting dimension to my analysis and may influence my feature engineering and model selection processes.

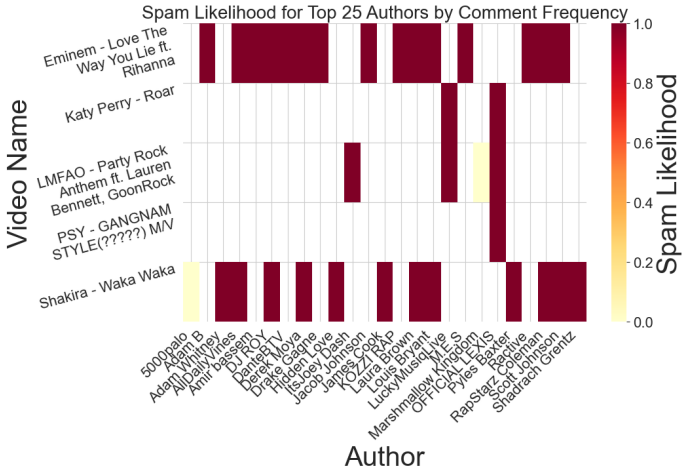


Fig. 7: Heatmap showing the spam likelihood for the top 25 authors by comment frequency. The y-axis represents the video names, and the x-axis represents the authors. The color intensity indicates the likelihood of spam, with a color gradient from yellow (low likelihood) to red (high likelihood).

B. Hypothesis and Analysis Objectives

Based on the characteristics of this dataset, I hypothesize that:

- The consistency in user behavior (all spam or all non-spam) might allow for effective user-based features.
- Text embedding size may impact the model performance, with larger embedding potentially capturing more nuanced information but requiring more data to train effectively.
- Different classification algorithms may perform variably due to the textual nature of the data, with some potentially struggling with the high-dimensional feature space created by text data.

My analysis will focus on:

- Comparing the performance of Neural Networks, Support Vector Machines, and k-Nearest Neighbors, and Gradient Boosting Tree on this text classification task.
- Evaluating the impact of different text preprocessing techniques on model performance.
- Analyzing the learning curves to understand how each algorithm copes with the limited dataset size.
- Investigating the effectiveness of various feature engineering approaches for text data.

Through this analysis, I aim to gain insights into the strengths and weaknesses of each algorithm when applied to text classification tasks, and to understand the unique challenges posed by spam detection in online platforms.

C. Data Preprocessing

The raw data contained six columns: comment ID, author name, comment date, video name, comment content, and the target variable indicating whether the comment is spam or not. To prepare this textual data for my machine learning models, I implemented a comprehensive preprocessing pipeline:

Feature Combination: I combined the `AUTHOR`, `VIDEO_NAME`, and `CONTENT` columns into a single `combined_text` feature. This approach allows my models to consider all available textual information.

Text Cleaning: I applied several standard NLP cleaning techniques to the combined text:

- Converting all text to lowercase to ensure consistency
- Removing non-alphabetic characters and punctuation
- Tokenizing the text into individual words
- Removing common English stop words to reduce noise in the data

Text Vectorization: To convert the cleaned text into a format suitable for my machine learning models, I employed two different embedding methods:

1) Embedding Methods: TF-IDF Vectorization

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure used to evaluate the importance of a word to a document in a collection or corpus [3]. I used scikit-learn's TfidfVectorizer with a maximum of 768 features to create sparse vector representations of my documents.

2) Transformer-based Embedding: For a more advanced representation of the text data, I utilized different pretrained embedding models from the sentence-transformers library. Specifically, I employed 'distilbert-base-nli-mean-tokens' and 'paraphrase-MiniLM-L3-v2' to generate dense vector representations of the documents. The DistilBERT-based model ('distilbert-base-nli-mean-tokens') is a distilled version of BERT, retaining 97% of BERT's language understanding capabilities while being 40% smaller and 60% faster [4]. This model produces embeddings with 768 dimensions, capturing rich semantic information from the input text. The MiniLM-based model ('paraphrase-MiniLM-L3-v2') generates more compact embeddings with 384 dimensions [5].

After preprocessing and embedding, I split my dataset into training and testing sets using an 80-20 split ratio.

D. Model Implementation and Hyperparameter Tuning

I implemented four classification algorithms on the preprocessed YouTube Comments Spam dataset: KNN, SVM, NN, and GBT. I have used the same sklearn framework and default hyperparameters as in Dataset 1, with the following hyperparameters for each model:

1) KNN:

- Number of neighbors: 1, 2, 3, 4, 5, 10, 15, 20, 25, 30
- Weights: uniform, distance

2) SVM:

- Kernel: 'linear', 'poly', 'rbf', 'sigmoid'
- γ: scale, auto, 0.1, 0.01

3) NN: With adaptive training rate, I explored the following hyperparameters:

- One hidden layer: 50, 100, 150, 200, 250, 300, 350 neurons; Two hidden layers: [32, 5], [32, 10], [32, 15], [32, 20], [32, 25], [32, 30], [32, 32] neurons
- Initial learning rate: 1.0, 0.5, 0.1, 0.08, 0.06, 0.04, 0.02, 0.01, 0.005, 0.001, 0.0005, 0.0001

4) GBT:

- Number of estimators: 50, 100, 150, 200, 250
- Maximum tree depth: 10, 15, 20, 25, 30, 35, 40

Different from dataset 1, dataset 2 has relatively balanced classes, which allows me to use accuracy as the primary metric for model selection. To verify my hypothesis, I have conducted three cases of analysis:

- Case 1: TF-IDF vectorization with 768 features;
- Case 2: DistilBERT embeddings with 768 features;
- Case 3: MiniLM-based embeddings with 384 features.

All three cases have undergone the same hyperparameter tuning process with 5-fold cross-validation.

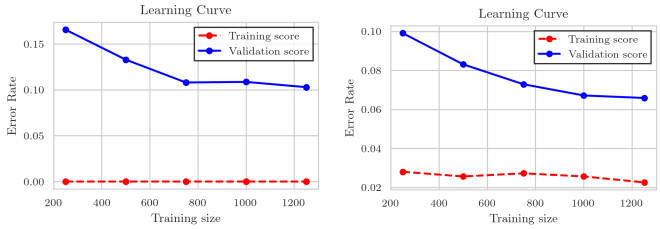
E. Results and Analysis

The best models for each case are showed in Tab. IV, which are is selected with the highest validation acccuracy due to the balanced classes. It displays that the choice of embedding method/size significantly impacts model performance. Case 2 (DistilBERT embeddings) and Case 3 (MiniLM-based embeddings) consistently outperform Case 1 (TF-IDF vectorization) across all models. This suggests that transformer-based embeddings capture more nuanced semantic information from the text data compared to traditional TF-IDF vectorization.

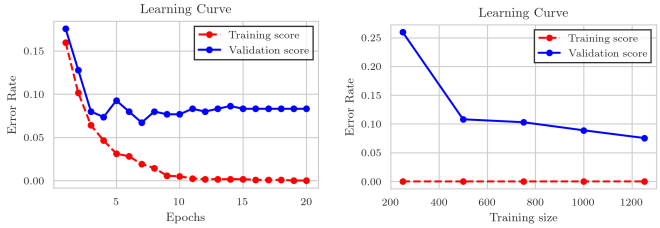
KNN shows the poorest performance among the four models, particularly in Case 1; seen in Tab. IV. Even through the training accuracy is high, the validation accuracy is significantly lower, indicating overfitting. This aligns with our findings from Dataset 1, reinforcing that KNN may not be suitable for high-dimensional data. SVM, NN, and GBT show strong performance, especially in Cases 2 and 3. It demonstrates the three models' ability to capture complex patterns in the embedded text data. The smaller size of embedding in Case 3 does not shows performance drop, indicating that the MiniLM-based embeddings are more efficient in capturing the relevant information for the spam detection task.

Case No	Model	Hyperparameters	Training	Validation
TF-IDF (768 features)	KNN	K=4, distance	0.987	0.827
	SVM	kernel=rbf, $\gamma = scale$	0.974	**0.909**
	NN	[250, 100], lr=0.01	0.953	0.889
	GBT	max depth 10, estimator 50	0.982	0.900
DistilBERT (768 features)	KNN	K=4, distance	1.000	0.885
	SVM	kernel=rbf, $\gamma = 0.01$	0.998	0.916
	NN	[100], lr=0.001	0.960	**0.938**
	GBT	max depth 3, estimator 250	1.000	0.910
MiniLM (384 features)	KNN	K=10, distance	1.000	0.897
	SVM	kernel=rbf, $\gamma = scale$	0.977	**0.934**
	NN	[100], lr=0.06	0.978	0.932
	GBT	max depth 5, estimator 200	1.000	0.925

TABLE IV: Best model selection among the four models in each case. The table displays the training and validation accuracy, along with the best hyperparameters for each model.



(a) KNN (b) SVM



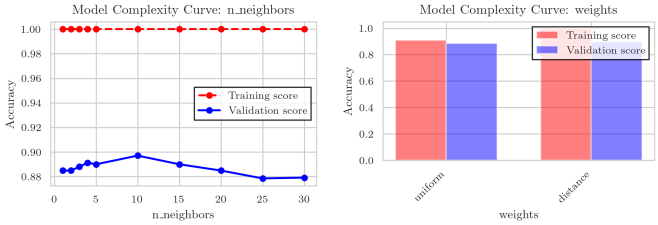
(c) NN (d) GBT

Fig. 8: Learning curves for KNN, SVM, NN, GBT in Case 3. The x-axis represents the number of samples, and the y-axis represents the error rate.

The learning curve in Fig. 8 provide insights into each model's learning behavior. All four models has significant gap between training and validation error, indicating overfitting. The gap is particularly large for KNN, suggesting that the model is not generalizing well to unseen data. For KNN, SVM, and NN, the gaps don't converge as the sample size increases, indicating that those model might benefit from more data to improve generalization. For GBT, the validation score improves with more data, therefore GBT could benefit from additional samples or stronger regularization.

For hyperparameter analysis, I use the training and validation results from Case 3 from Fig. 9 to Fig. 12 to provide insights into the model's behavior. The hyperparameter tuning results for KNN, SVM, NN, and GBT are as follows Sec. III-D. For Dataset 2, I did not list the training time hyperparameter tuning results, as it delivers the same insights as Dataset 1.

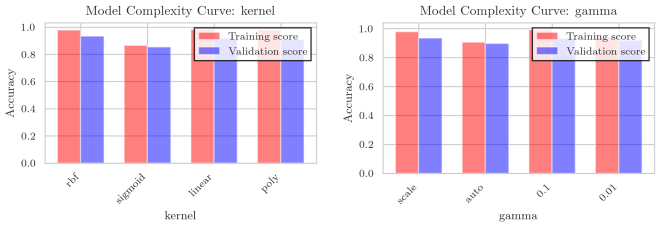
When tuning the number of neighbors and weights for KNN, Fig. 9 shows that the largest accuracy is at K=10 and then declines. It illustrates that considering too many neighbors introduces noise. Distance-based weights consistently outperform uniform weights, indicating that closer neighbors provide more valuable information for classification.



(a) KNN: Accuracy vs. neighbors (b) KNN: Accuracy vs. weights

Fig. 9: KNN hyperparameter tuning in Case 3. The x-axis represents the hyperparameter values, and the y-axis represents the accuracy.

The RBF kernel outperforms than other kernal when setting γ as scale, as shown in Fig. 10. The polynomial kernel has a similar performance to the RBF kernel, but the RBF kernel is more computationally efficient. The γ value of scale has the highest accuracy, indicating that a moderate influence of each training example is beneficial for this dataset.



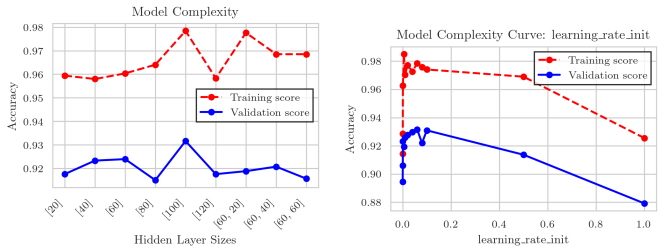
(a) SVM: Accuracy vs. kernel (b) SVM: Accuracy vs. γ

Fig. 10: SVM hyperparameter tuning in Case 3. The x-axis represents the hyperparameter values, and the y-axis represents the accuracy.

In NN hyperparameter tuning, the single hidden layer with 100 neurons and initial learning rate as 0.06 yeild the best performance; seen in Fig. 11. The model is sensitive to learning rate, with optimal performance at rates between 0.01 and 0.1. This indicates the importance of careful learning rate tuning for NN performance. With 648 features, 100 neurons are sufficient to capture the relevant information in the data, which significantly reduces the model's complexity and training time.

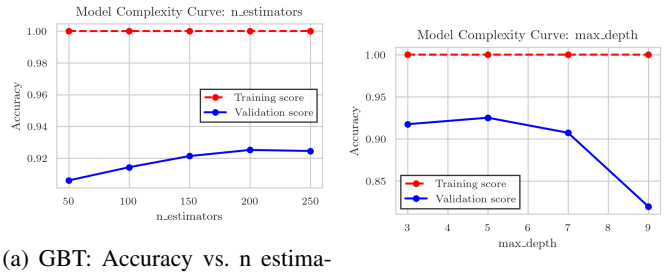
GBT's performance improves with the number of estimators up to about 200, after which returns diminish. Optimal max depth is around 3-5 and the performance drops significantly when the max depth increases, suggesting that deeper trees may lead to overfitting on this dataset.

The Tab. V summarize the model performance on the testing sets for the three cases. In the TF-IDF case, GBT achieves



(a) NN: Accuracy vs. hidden layer sizes (b) NN: Accuracy vs. learning rate

Fig. 11: NN hyperparameter tuning in Case 3. The x-axis represents the hyperparameter values, and the y-axis represents the accuracy or training time.



(a) GBT: Accuracy vs. n estimators (b) GBT: Accuracy vs. max depth

Fig. 12: GBT hyperparameter tuning in Case 3. The x-axis represents the hyperparameter values, and the y-axis represents the accuracy or training time.

the highest accuracy (0.887), precision (0.969) and F1 score (0.894), while NN shows the best recall (0.892) and F1 score (0.894). For DistilBERT embeddings, SVM outperforms other models with the highest accuracy (0.941), recall (0.946), and F1 score (0.948), while NN achieves the best precision (0.965). In the MiniLM embedding case, SVM again leads in accuracy (0.941), recall (0.919) F1 score (0.947), with NN showing the highest precision (0.990). Notably, all models show improved performance with transformer-based embeddings (DistilBERT and MiniLM) compared to TF-IDF, with accuracy improvements ranging from 2% to 10% relatively. This consistent improvement across all models underscores the effectiveness of advanced embedding techniques in capturing relevant features for spam detection. The high precision values, particularly for SVM and NN, indicate that these models are especially effective, which is crucial in spam detection applications.

Unlike the Bank Marketing dataset, the YouTube Comments Spam dataset is more balanced and involves text classification. This leads to some notable differences in model behavior:

- SVM performs exceptionally well on this text classification task, whereas it was outperformed by NN and GBT in the Bank Marketing dataset.
- The impact of embedding methods is significant in text classification tasks. Transformer based embedding methods (DistilBERT and MiniLM) outperform traditional TF-IDF vectorization.
- All models achieve higher accuracy on this dataset compared to the Bank Marketing dataset, possibly due to the more balanced nature of the classes and the effectiveness of text embeddings in capturing relevant features.

Case No	Model	Accuracy	Precision	Recall	F1
TF-IDF (768 features)	KNN	0.821	0.896	0.776	0.832
	SVM	0.882	0.968	0.821	0.888
	NN	0.880	0.896	**0.892**	**0.894**
	GBT	**0.887**	**0.969**	0.830	**0.894**
DistilBERT (768 features)	KNN	0.892	0.933	0.874	0.903
	SVM	**0.941**	0.950	**0.946**	**0.948**
	NN	0.911	**0.965**	0.874	0.917
	GBT	0.908	0.947	0.888	0.917
MiniLM (384 features)	KNN	0.901	0.926	0.897	0.911
	SVM	**0.941**	0.976	**0.919**	**0.947**
	NN	0.926	**0.990**	0.879	0.931
	GBT	0.906	0.939	0.892	0.915

TABLE V: Model Performance Metrics on Testing Set. The table shows the accuracy, precision, recall, and F1 score for each model in three cases.

IV. Conclusion and Discussion

In this report, I conducted a comparative analysis of four machine learning models (KNN, SVM, NN, and GBT) on two distinct datasets: Bank Marketing Campaign Subscription and YouTube Comments Spam Detection. The analysis revealed that KNN generally performs poorly on high-dimensional data, while SVM excels in text classification tasks. GBT and NN work well for tabular datasets, and all models benefit from transformer-based embeddings in text classification tasks.

However, challenges remain in the analysis. For Dataset 1, which is unbalanced, model performance was not optimal. Given more time, additional feature engineering could potentially improve results. In the text classification task, I have not fully explored data preprocessing, which currently combines user comments, author names, and video titles. This approach, while generic, may not be ideal for new users or videos. With additional time, I would like to investigate a more robust method that focuses solely on comment content, potentially leading to a more generalizable solution.

References

[1] Bank Marketing Campaign Subscription data source: https://www.kaggle.com/datasets/pankajbhowmik/bank-marketing-campaign-subscriptions

[2] Youtube Comments Spam data source: https://www.kaggle.com/datasets/ahsenwaheed/youtube-comments-spam-dataset

[3] Wikipedia contributors. (2024, July 26). TfIdf. In Wikipedia, The Free Encyclopedia. Retrieved 10:20, September 22, 2024, from https://en.wikipedia.org/w/index.php?title=Tf

[4] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. ArXiv, abs/1910.01108.

[5] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, & Ming Zhou. (2020). MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers.