Implementation of Binary Classification

Jiaqi Yin

***Index Terms*—Classification, NLP**

I. Introduction

In this paper, I will implement several binary state of art classification on two types of datasets: Bank Markeking Campaign Suscription in Sec. II and Youtube comment spam dectection in Sec. III

II. Dataset 1: Bank Marketing Campaign Subscription

In the following sections, I will describe the dataset, preprocessing steps, implemented models, hyperparameter tuning, results analysis, and sensitivity analysis by feature reduction.

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 sensitivity analysis through feature reduction.

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.

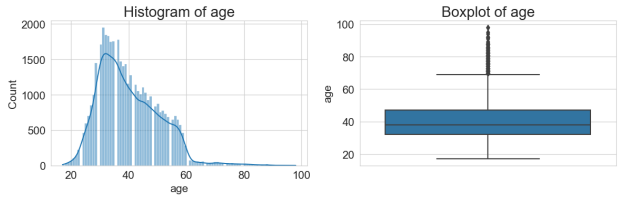
Numerical features contain some outliers that could distort the model, and most are skewed; see in Fig 1. To improve performance, I implemented the following steps:

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

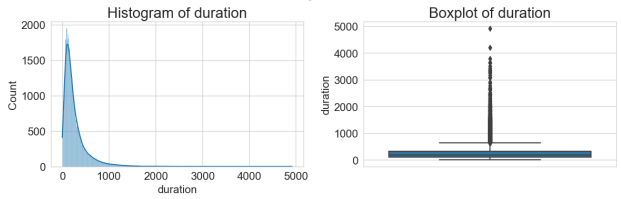
For categorical features, I applied one-hot encoding to convert them into numerical format.

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 Section.

For model training and testing, I split the dataset into a training set and a testing set with a 4:1 ratio. For hyperparameter tuning, I employed five-fold cross-validation on the training set.



(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.

D. Model Implementation and Hyperparameter Tuning

I implemented four models using the scikit-learn library: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Neural Network (NN), and Gradient Boosting Tree (GBT). For all models, I used 5-fold cross-validation for hyperparameter tuning. 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 contral 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, and 'sigmoid' activation for the output layer. 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.

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.

I also leveraged the feature importance from GBT to perform feature reduction, which will be discussed in a later section. This approach capitalizes on the boosting tree's ability to learn which combination of features significantly reduces the fitting error.

E. Results and Analysis

To verify my hypothese, I have conducted three cases of analysis:

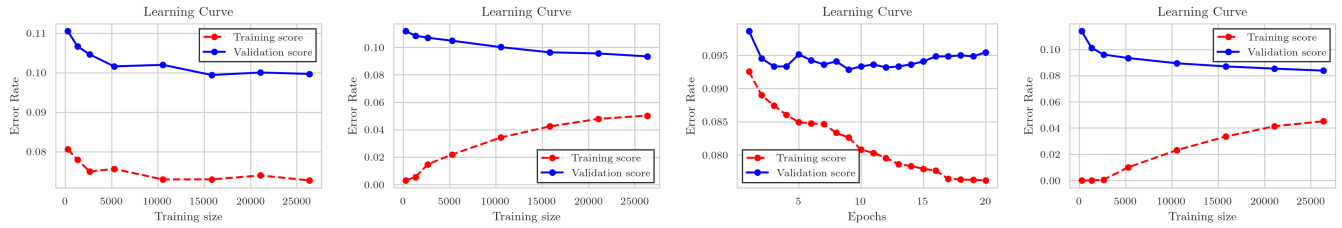
1) Full features without oversampling;
2) Feature reduction with GBT;
3) Oversampling the minority class.

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 iterations. To understand the model complexity, I have also monitored the performance with different hyperparameters.

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.

2) Case 2: Feature reduction with GBT:

3) Case 3: Oversampling the minority class:



(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 F1-score.

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=rbf, $\gamma = 0.1$	0.667	0.524
NN	[32, 25], adaptive, 0.05	0.611	0.567
GBT	max depth 5, estiamtor 50, subsample 1	0.671	0.602

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

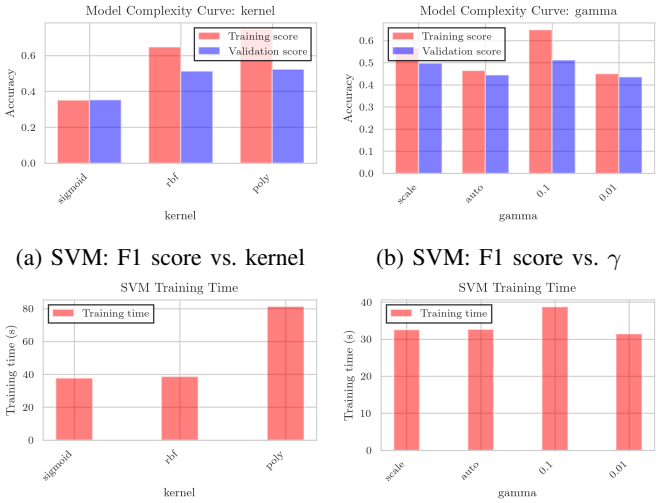
(c) KNN: Training time vs. neighbors (d) KNN: Training time vs. weights

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.

TABLE II: Feature Importance

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

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



(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.

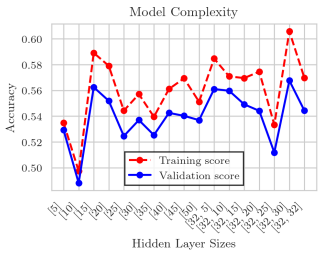
Case No	Model	Accuracy	Precision	Recall	F1
1	KNN	0.902	0.609	0.419	0.497
	SVM	0.915	0.658	0.494	0.564
	NN	0.921	0.649	0.643	0.646
	GBT	**0.923**	**0.679**	**0.591**	**0.632**
2	KNN	0.906	0.598	0.487	0.537
	SVM	0.915	**0.686**	0.437	0.534
	NN	0.920	0.645	0.629	0.637
	GBT	**0.923**	0.682	**0.587**	**0.631**
3	KNN	0.898	0.564	0.478	0.517
	SVM	0.911	0.680	0.421	0.520
	NN	0.912	0.657	0.484	0.557
	GBT	**0.918**	**0.671**	**0.554**	**0.607**

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

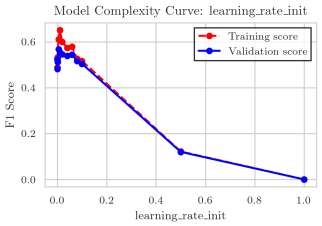
III. Dataset 2: YouTube Comments Spam Detection

A. Data Description and Rationale

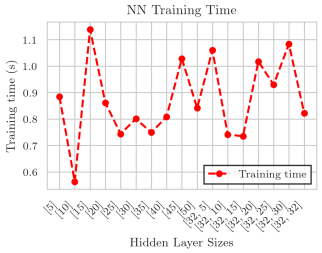
The YouTube Comments Spam dataset is a public collection comprising 1,956 real comments from five different videos that were among the 10 most viewed during the collection period



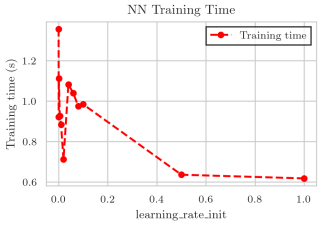
(a) NN: F1 score vs. hidden layer sizes



(b) NN: F1 score vs. learning rate

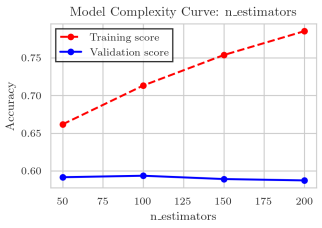


(c) NN: Training time vs. hidden layer sizes

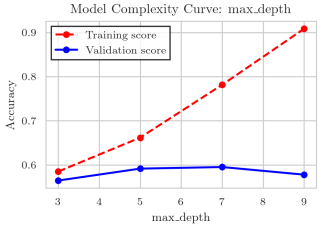


(d) NN: Training time vs. learning rate

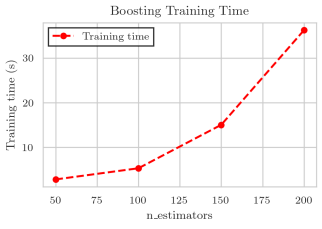
Fig. 5: NN hyperparameter tuning in Case 1. The x-axis represents the hyperparameter values, and the y-axis represents the F1-score or training time.



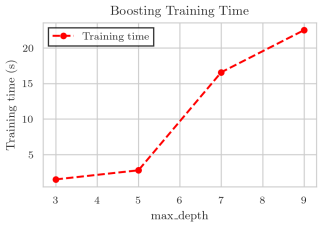
(a) GBT: F1 score vs. n estimators



(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.

[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.

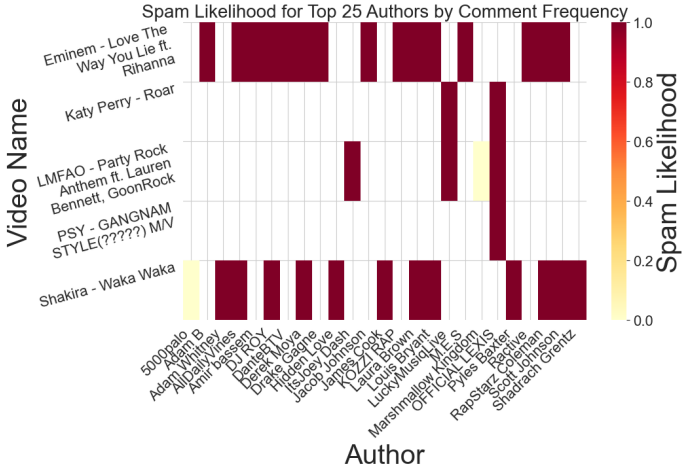


Fig. 7: Heatmap showing the spam likelihood for the top 25 authors by comment frequency. The y-axis represents the video names, and the x-axis represents the authors. The color intensity indicates the likelihood of spam, with a color gradient from yellow (low likelihood) to red (high likelihood).

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 embeddings 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 stopwords 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. 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. This model produces embeddings with 768 dimensions, capturing rich semantic information from the input text. The MiniLM-based model ('paraphrase-MiniLM-L3-v2') uses a deep self-attention mechanism to learn language representations, generating more compact embeddings with 384 dimensions while still maintaining high performance on various NLP tasks.

After preprocessing and embedding, I split my dataset into training and testing sets using an 80-20 split ratio. This allows us to train my models on a substantial portion of the data while retaining a significant amount for testing to ensure robust evaluation of my models' performance.

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.

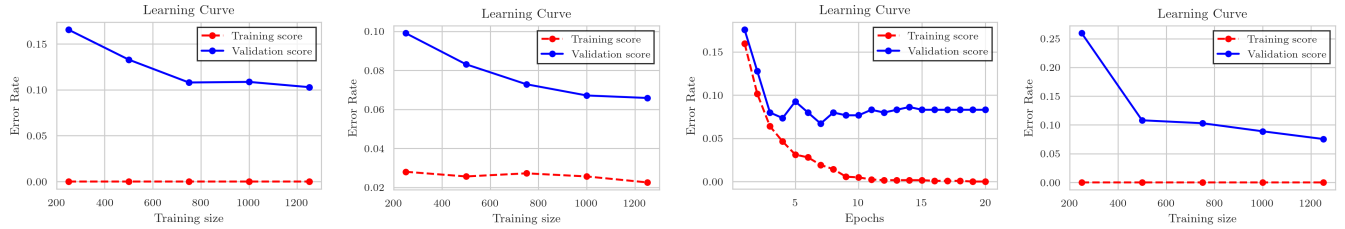
All three cases have undergone the same hyperparameter tuning process with 5-fold cross-validation.

E. Results and Analysis

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

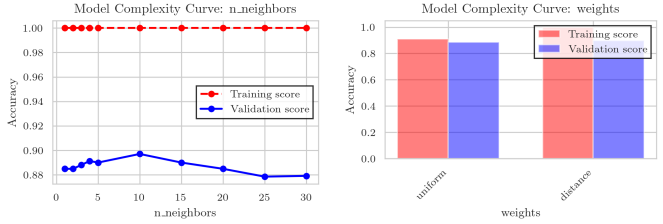


(a) KNN (b) SVM (c) NN (d) GBT

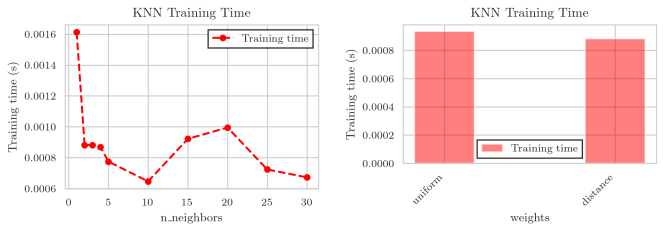
Fig. 8: Learning curves for KNN, SVM, NN, GBT in Case 3. The x-axis represents the number of samples, and the y-axis represents the F1-score.

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

TABLE V: 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.

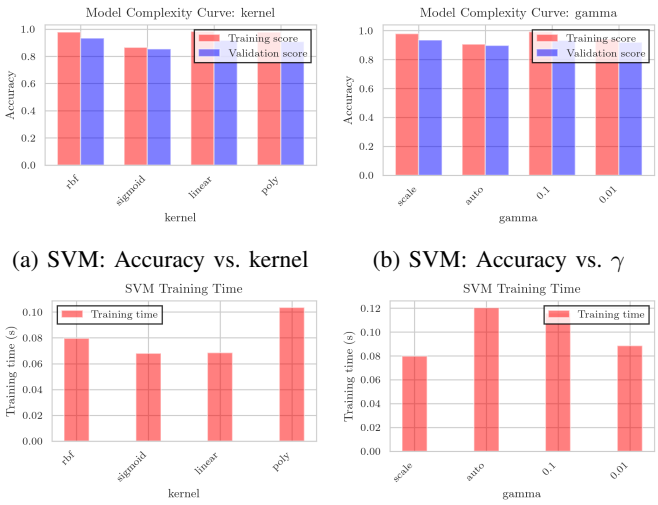


(a) KNN: Accuracy vs. neighbors (b) KNN: Accuracy vs. weights



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

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



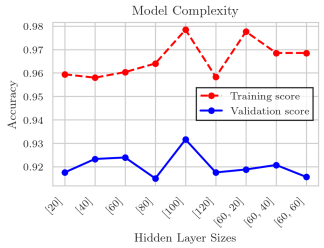
(a) SVM: Accuracy vs. kernel (b) SVM: Accuracy vs. γ

(c) SVM: Training time vs. kernel (d) SVM: Training time vs. γ

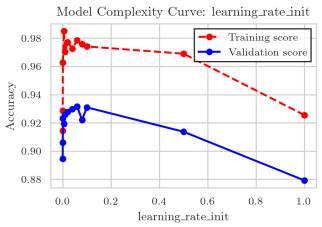
Fig. 10: SVM hyperparameter tuning in Case 3. The x-axis represents the hyperparameter values, and the y-axis represents the accuracy or training time.

Case No	Model	Accuracy	Precision	Recall	F1
1	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**
2	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
3	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

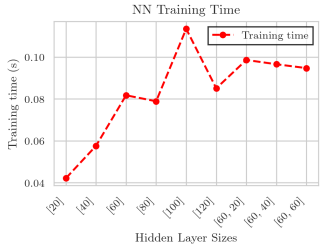
TABLE VI: Updated Comparison of Model Performance Metrics on Testing Set. The table shows the accuracy, precision, recall, and F1 score for each model in three cases.



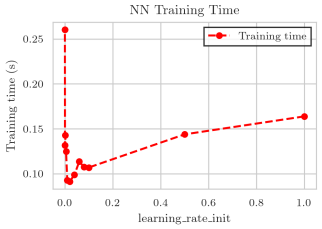
(a) NN: Accuracy vs. hidden layer sizes



(b) NN: Accuracy vs. learning rate

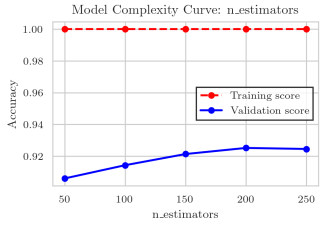


(c) NN: Training time vs. hidden layer sizes

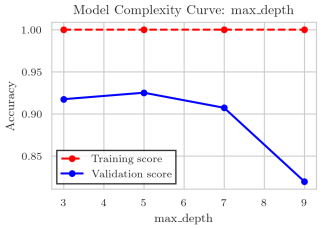


(d) NN: Training time vs. learning rate

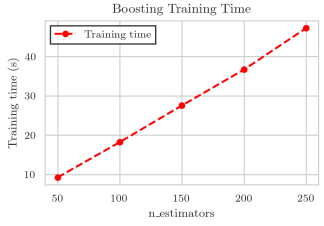
Fig. 11: NN hyperparameter tuning in Case 3. The x-axis represents the hyperparameter values, and the y-axis represents the accuracy or training time.



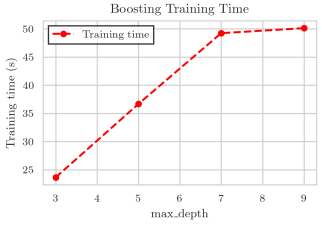
(a) GBT: Accuracy vs. n estimators



(b) GBT: Accuracy vs. max depth



(c) GBT: Training time vs. n estimators



(d) GBT: Training time 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.