**Author Classification Using Text Mining and Machine Learning Techniques**

by

Astan, Soner

Dogus University

Class: COME 448

Student Number: 202402001085

Professor: Prof. Dr. Aysun Güran

Submission Date: May 7, 2025

Draft Of Content

[1 Introduction 1](#_Toc196139830)

[1.1 Problem 1](#_Toc196139831)

[1.2 Importance of the topic 1](#_Toc196139832)

[2 Preprocessing 2](#_Toc196139833)

[2.1 The Dataset 2](#_Toc196139834)

[2.2 Preprocessing steps 2](#_Toc196139835)

[2.3 Feature extraction methods 3](#_Toc196139836)

[2.3.1 TF-IDF 3](#_Toc196139837)

[2.3.2 n-grams 4](#_Toc196139838)

[2.3.3 BERT 4](#_Toc196139839)

[3 Experiment 5](#_Toc196139840)

[3.1 Implementation of machine learning models 5](#_Toc196139841)

[3.2 Experimental setup and evaluation methodology 6](#_Toc196139842)

[3.3 Performance comparison 7](#_Toc196139843)

[4 Conclusion 7](#_Toc196139844)

[4.1 Summary 7](#_Toc196139845)

[4.2 Key findings 7](#_Toc196139846)

[4.3 Insights 7](#_Toc196139847)

# Introduction

In this chapter, we introduce the central problem of author classification and explain its relevance in the context of real-world applications. We explore why this topic is important in today's digital age and how machine learning techniques can contribute to solving it. The section also outlines the motivation for this project and the overarching goal of distinguishing authors through data-driven methods.

## 1.1 Problem

Author classification is the task of automatically identifying the author of a given text based on their writing style. This problem is highly relevant in digital forensics, copyright attribution, and automated content moderation. In an era where vast amounts of textual data are published daily, reliable author identification is essential for validating authorship, detecting plagiarism, and analyzing stylistic patterns in various contexts such as literature, academia, and social media.

## Importance of the topic

The ability to accurately classify authors has wide-ranging applications in academia, publishing, legal investigations, and natural language processing research. With the increasing use of AI-generated and anonymous content, author attribution is becoming more critical than ever. Through machine learning and text mining techniques, we aim to develop robust models capable of learning stylistic cues from text and distinguishing authors with high accuracy. This project contributes to the field by evaluating various classical and modern models on Turkish textual data, incorporating both traditional features (TF-IDF, n-grams) and contextual embeddings (BERT).

# Preprocessing

Preprocessing is a critical step in any text classification pipeline, as it transforms unstructured textual data into a numerical format that machine learning models can process effectively. This chapter outlines the dataset used, describes each step taken to clean and structure the text data, and explains the feature extraction methods that were applied. The overall goal was to ensure that the models could accurately learn from stylistic and semantic patterns in the texts, with minimal noise and maximum discriminative power.

## 2.1 The Dataset

The dataset used in this project consists of Turkish text samples written by multiple authors. Each text is annotated with an author label, indicating which individual wrote the passage. For machine learning compatibility and anonymity, these author names were replaced with numerical class labels (e.g., *0*, *1*, *2*, etc.).

To ensure reliable and generalizable evaluation, the dataset was split into training and test subsets using a stratified sampling strategy. This method preserves the distribution of author labels across both subsets, ensuring that each class is equally represented and that no model is biased due to class imbalance in the test data.

## 2.2 Preprocessing steps

Before extracting features from the text data, several preprocessing steps were applied to improve consistency and reduce noise:

* Stopword Removal: A predefined list of Turkish stopwords (e.g., “ve”, “ama”, “bu”) was removed from the text. These frequently used words contribute little to stylistic differentiation and were excluded to improve the quality of extracted features.
* Lowercasing: All text was converted to lowercase to ensure case-insensitive analysis, preventing models from treating “Kitap” and “kitap” as different tokens.
* Character Cleaning: Non-alphabetical characters, such as punctuation and digits, were either removed or ignored during feature extraction.
* Parallel Processing: To optimize performance, both preprocessing and feature extraction were executed in parallel across multiple CPU threads. This significantly reduced runtime and enabled efficient handling of large volumes of data.

The cleaned and processed text data was then transformed into structured features using a combination of classical and modern techniques, which are detailed below.

## 2.3 Feature extraction methods

Feature extraction is the process of converting textual data into numerical representations that capture both syntactic and semantic information. In this project, two main types of features were used: traditional frequency-based methods (TF-IDF and n-grams) and contextual word embeddings from a pre-trained deep learning model (BERT).

### 2.3.1 TF-IDF

TF-IDF (Term Frequency –Inverse Document Frequency) is a classical statistical method that scores words based on their frequency within a document and their rarity across all documents. Terms that appear often in a single text but rarely across the corpus receive a higher weight, indicating their importance to that specific document.

In this project, TF-IDF vectors were constructed using a custom implementation that included:

* Double normalization (0.5): This technique normalized term frequencies to reduce the impact of document length and outliers.
* Smoothed IDF: A variation of IDF calculation that prevents division by zero and stabilizes scores for rare terms.

TF-IDF was computed across both word and character n-grams, providing a flexible representation of stylistic features.

### 2.3.2 n-grams

An n-gram is a contiguous sequence of *n* tokens from a given text. These sequences can be based on either words or characters:

* Word n-grams capture common multi-word phrases and stylistic patterns, such as “çok güzel” (“very beautiful”) or “bu nedenle” (“therefore”).
* Character n-grams detect recurring character patterns, suffixes, or stylistic markers within and across words.

For this project, the following n-gram features have been extracted:

* Word-based n-grams: Unigrams, bigrams, and trigrams (1-gram to 3-gram), with stopword removal.
* Character-based n-grams: Bigrams and trigrams (2-gram to 3-gram), without stopword filtering.

These features provide a robust statistical representation of textual style and are particularly effective for author identification tasks.

### 2.3.3 BERT

BERT (Bidirectional Encoder Representations from Transformers**)** is a state-of-the-art deep learning model developed by Google that captures rich, context-aware embeddings of text. Unlike TF-IDF and n-grams, BERT considers both the left and right context of a word in a sentence, enabling a deeper understanding of semantics and syntax.

In this project, we used the dbmdz/bert-base-turkish-cased model, which is specifically trained on Turkish language data. Each text was tokenized and processed by the BERT model, which generated contextualized embeddings for every token. These token embeddings were then mean-pooled across each sentence to obtain a single dense vector representation for each document.

BERT embeddings were particularly valuable for capturing semantic meaning and syntactic structure that is difficult to model using only frequency-based techniques.

# Experiment

This chapter presents the experimental research process that was conducted in order to evaluate the performance of various machine learning models on the author classification task. It includes the technical implementation of the models, the experimental setup, and the evaluation strategy used to ensure the validity and reliability of the results.

## 3.1 Implementation of machine learning models

To perform author classification, six different supervised machine learning models were implemented and evaluated:

* Random Forest
* Support Vector Machine (SVM)
* XGBoost (Extreme Gradient Boosting)
* Naive Bayes (MultinomialNB)
* Multi-layer Perceptron (MLP)
* Decision Tree

Each model was configured with reasonable default parameters and minimal tuning to ensure comparability and reduce training time. Where applicable, models were configured to handle class imbalance using the class\_weight="balanced" setting. This is especially important in author attribution tasks where the number of texts per author can be imbalanced.

Special treatment was applied to the SVM model using BERT embeddings, where a simple GridSearchCV with 2-fold cross-validation was used to select the best regularization parameter CCC from a limited range. For MLP and SVM models trained on dense features (like BERT), feature scaling was applied using StandardScaler to standardize the input data.

Naive Bayes was skipped for dense representations (i.e., BERT), as it is primarily designed for sparse count-based features like TF-IDF.

All models were implemented using Scikit-learn and XGBoost’s Python API, ensuring reproducibility and integration with common evaluation utilities.

## Experimental setup and evaluation methodology

The evaluation followed a structured experimental procedure:

* Data Splitting: All feature sets were split into training and test sets using a stratified 80/20 split to preserve the distribution of authors across classes. This ensures the representativeness of the test data.
* Feature Sets: The experiments were conducted across six different feature representations:
  + TF-IDF word-level: unigrams (1-gram), bigrams (2-grams), trigrams (3-grams)
  + TF-IDF character-level: 2-grams and 3-grams
  + Contextual embeddings from BERT (precomputed using the Turkish dbmdz/bert-base-turkish-cased model)
* Evaluation Metrics: The following metrics were computed for each model and feature combination:
  + Accuracy and error rate
  + Precision, recall, and F1-score (both per class and weighted average)
  + Positive and negative class metrics separately
* Parallel Execution: To improve runtime efficiency, all training and evaluation tasks were run in parallel using Python’s ThreadPoolExecutor, utilizing all available CPU cores.

All evaluation metrics and plots were saved into structured CSV and image files for further analysis and visualization.

## Performance comparison

This section presents a detailed performance comparison of all trained models across different feature sets. Results are broken down by overall metrics (accuracy, error rate, and weighted F1-score) as well as per-class metrics (precision, recall, and F1-score for positive and negative classes). We also evaluate the balance between precision and recall per class to assess fairness and robustness.

The analysis highlights the relative strengths of different models and provides insights into which combinations of features and algorithms are most effective for the author classification task in Turkish.

Each of the following subsections will focus on a specific metric or comparison and will include interpretations supported by visual plots and summary tables.

# 4 Conclusion

## 4.1 Summary

## 4.2 Key findings

## 4.3 Insights