Author Attribution through Linear SVC

Hudson Strauss, Helen Wang, Juan Alfredo ”JoJo” Torres Molina
Ids: (1617457), (1634974), (1626252)
Team 6

June 11, 2025

Abstract

Author classification, the task of identifying the writer of a text, has important real-world applications in fields such as forensic linguistics and security analysis. In this project, we aim to classify the authors of emails within the Enron Corpus, a large and diverse dataset consisting of email communications from over 150 Enron employees. To focus our analysis on genuine writing style, we restricted our dataset to sent emails only, removing signatures, quoted text, and URLs to prevent models from learning superficial cues like names or email handles.

Preprocessing steps included tokenization, lemmatization, stemming, and stop-word filtering to ensure the text representation reflects meaningful linguistic patterns. We transformed the processed text using TF-IDF vectorization and trained several classification models, including Linear SVC, Logistic Regression, Ridge Classifier, and Multinomial Naive Bayes. Model performance was evaluated and compared to determine the most effective approach for author identification in this context. This experiment resulted in a Linear SVC classification model with an accuracy score of 0.8245 of identifying the author of a given email from the Enron email dataset.

1 Introduction

The problem which we are seeking to explore in this paper is author classification, or being able to find the author of a text by its contents. Our approach was to first look at the data set we targeted. In this work, we explore the author classification problem using a real-world dataset: the Enron Corpus.

The Enron Corpus is a large collection of over 500,000 emails generated by employees of the Enron Corporation. To focus our study on genuine authorship characteristics, we limited our analysis to the sent items folders only—emails that were actually composed by users. This subset includes 37,664 messages from 136 different users, yielding an average of approximately 278 emails per user. While this is a reduction in total data, it ensures that we are training on original user-generated content, which is critical for the author identification task. However, the scope of this project is based around predicting the author.

Before training, we performed a series of preprocessing steps on the email content. These included parsing the email bodies to remove quoted text, signatures, and URLs, then applying text normalization techniques such as lemmatization, stemming, and stop-word filtering to emphasize stylistic features. Each email was labeled by its sender, forming the basis for supervised learning.

We split the dataset into training and testing sets, reserving 30% of the data for evaluation. To transform the text into a format suitable for machine learning, we used Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to extract meaningful features from the processed emails. [KY04] We then trained and evaluated several classification models, comparing their performance in predicting the author of unseen emails..

This paper presents our methodology, experiments, and analysis of different machine learning approaches to author classification using the Enron email dataset.

2 Background and Related Work

- Background/Related Work: This section discusses relevant literature for your project.
  writing inspo https://ceas.cc/papers-2004/136.pdf

***writing support** https://www.sciencedirect.com/topics/computer-science/vector-space-models https://www.sciencedirect.com/science/article/pii/S1319157814000573 https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/AF1-1.pdf https://onlinelibrary.wiley.com/doi/full/10.1111/exsy.12136

3 Approach

3.1 Pre-Processing

This is where we broke apart the data and the emails within. Like mentioned above we separated the data into sent emails. We are identifying the authors of emails and this was the logical data to target. We then used the python email parser to separate the body of outgoing email threads. We then cleaned the message body. Initially targeting the removal of all links, whitespace, and erroneous punctuation. We removed all stop words, stemmed the data, and lemmatized the rest. Then converted the remaining words in the text into tokens for that file. These tokens were paired with the author of the email. For data preprocessing, we had stripped each email of URLs, punctuations, and collapsed whitespaces. We also tokenized each word, first by stemming and then lemmatizing the result. Following this, we utilized the TfidfVectorizer from the sklearn library in order to transform the processed data into a matrix. Then, we filtered the data again by removing any authors and corresponding samples that appeared less than two times. After transforming the filtered data as TF-IDF vectors, it was split into training and testing sets. These sets were then used for training models.

We then separated our data set into 30% test data and the rest would be used for training.

3.2 Feature Extraction

For our feature extraction we followed literature recommendation [KY04]. TF-IDF represented each document as a TF-IDF vector. We capped the frequent terms at 20,000.

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

$$idf(t,D) = log \frac{N}{|\{d \in D : t \in d\}|}$$

Figure 1: TF-IDF Calculation

To do this we leveraged SK-Learn's TfidfVectorizer functionality. TF–IDF transforms each document into a vector of term weights that reflect how important a word is in that document relative to the whole data set. The term frequency (TF) component boosts words that appear often in a document, while the inverse document frequency (IDF) component down-weights words that appear in many documents. By combining TF and IDF, TF–IDF highlights terms that are both frequent in a given document and distinctive across the whole collection. With the goal of making it easier for our classifier to focus on the most informative words with the greatest insight.

3.3 Classification

For our classification method we tried many different approaches. We will compare them in our experimentation section. After many methods our best preforming method was the Linear SVC method.

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1} \max(0, y_i(w^T \phi(x_i) + b)).$$

Figure 2: LinearSVC Equation

Linear SVC seeks a single, straight decision boundary that separates the classes by maximizing the distance between the boundary and the nearest points of each class. Concretely, it learns weights **w** and bias **b** so that the decision function

$$y_i(w^\top \phi(x) + b)$$

can correctly label training examples while keeping the margin $1/\|w\|$ as large as possible, trading off margin size against misclassification with a hinge-loss penalty.[sci25b] Focusing on the points closest to the boundary, a Linear SVC aims to generalize well with new data, run efficiently on high-dimensional feature sets, and produces sparse models that highlight the most discriminative features. Meshing well with our TF-IDF selection method. To support our work we leveraged sklearn's LinearSVC functionality.[sci25a]

3.4 Evaluation

We assess our author-attribution model using several metrics so that we can capture both its overall accuracy and its behavior in individual classes. Accuracy measures the fraction of messages correctly labeled, giving a high-level sense of overall performance, but it can be misleading when some authors dominate the dataset. To account for size and class imbalances, we computed the precision, which tells us, for each author, the proportion of predicted messages that truly belong to them or how "confident" our model is when it makes an attribution. We also calculated the recall, which tells us the proportion of an author's true messages that our model successfully identifies or how "complete" its coverage is. Because there is often a trade-off between precision and recall, optimizing one can hurt the other, we combined them into the F1-score. Mostly however, accuracy was our driving factor for quicker evaluation.

4 Experiments

We had experimented with several models: linear regression, multinomial naive bayes, ridge classification, stochastic gradient descent classification, passive aggressive classification, K neighbors classification, decision trees and linear support vector classification (linear SVC).[dev25a] We also tried using a shallow neural network. We found that linear SVC had the highest accuracy of these models, followed by ridge classification.

TF-IDF:

Initially, we had used the default parameters for the TfidfVectorizer[dev25b]. Although this had resulted in an accuracy of 0.8064 for our best model, we later altered the parameters to optimize the accuracy even further. We modified the parameters ngram_range, max_df, and min_df primarily. Changing the ngram_range to (2,2), which means only bigrams, resulted in an accuracy score drop from 0.80 to 0.77 for our best model. Changing it to (1,2), meaning both unigrams and bigrams resulted in a 0.02 increase in the accuracy score. Altering the max df resulted in no changes in the accuracy score, while altering the min df did. Changing min df to 2 did show a little improvement to the accuracy score when looking at four decimal places while changing it to 5 had even more of an improvement (also only visible at four decimal places).

Linear SVC:

We tested different C values for the accuracy of the linear SVC and found that C= 0.4 gave the best accuracy score of 0.8245 and the F1 score of 0.8234. After finding that linear SVC produced the most accurate model, we had attempted to optimize its parameters. The default value for the regularization parameter (C) is 1 [sci25a], producing an accuracy of 0.8012. In order to optimize C, we looped through values starting from 0.1 to 1.7 incremented by 0.1 each time. This resulted in finding a C value of 0.5 to be the best, producing an accuracy of 0.8064. We also attempted to change the class weight to be balanced, to reduce the bias towards more frequent authors. This resulted in the accuracy score being lower than 0.8. We also attempted to change the loss from the default squared hinge to the hinge, but that also resulted in worse accuracy scores. After modifying the TfidfVectorizer parameters, the accuracy score improved by about 0.2 and the optimal C value changed to 0.4, resulting in a score of 0.8245.

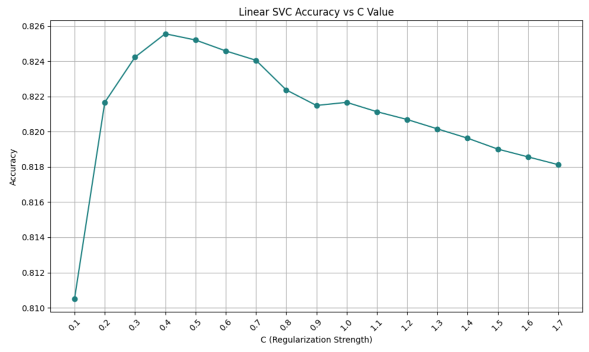
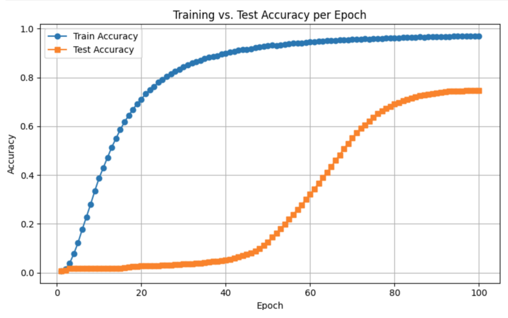


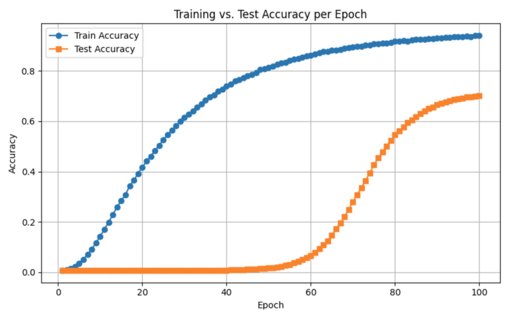
Figure 3: Linear SVC Accuracy vs C value

Shallow Neural Network:

Although this project focuses on more traditional machine learning methods, we still wanted to explore using Shallow Neural Networks, and try to use them as a benchmark for the other models so we could find out the hypothetical maximum accuracy score; However, as we found out, Neural Networks were not as accurate as our other methods. We used a neural network model with 256 nodes in the first hidden layer and 128 in the second hidden layer. Then we played around with the alpha value, as can be seen in figures 4 through 6, and got a maximum accuracy score of 0.77 and maximum with alpha =0.0001.



(a) Neural Network Training vs Test accuracy, $\alpha = 0.0001$



(b) Neural Network, $\alpha = 0.001$

Figure 4: Comparison of NN accuracy for two different learning rates

We also made an attempt with a more complicated model that used 2048 nodes in the first hidden layer and 1024 nodes on the second hidden layer, and it got an accuracy score of 0.7832.

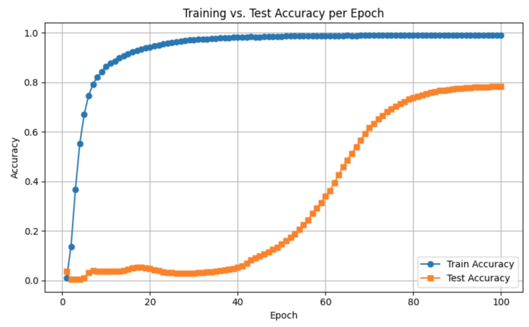


Figure 5: Higher complexity neural network

Ridge Classifier:

Ridge classification had also produced a relatively high accuracy score. Initially, the score was 0.7951. We tried to optimize the learning rate, also looping from 0.1 to 1.7 in increments of 0.1. The default alpha, 1.0, had produced the best score[dev25c]. However, after altering the parameters of the TfidfVectorizer, ridge classification produced an accuracy score of 0.8185. We then ran the same again, this time resulting in alpha being optimal at 1.3, 1.4, and 1.5, all of which resulted in an accuracy score of 0.8195.

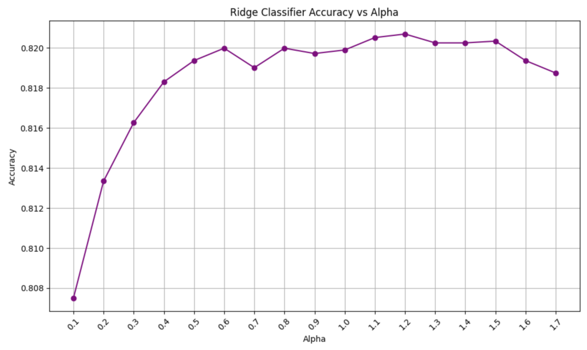


Figure 6: Ridge classifier Accuracy vs alpha

KNN:

To find the optimal value of the parameter k for this model, we simply trained 100 models with different k values in a range of 1 through 100. We used this method because using a knn model was very time efficient for our data. Our first experiment was using the default similarity measure,The best score was at k= 50, with an accuracy = 0.61713426 We tried some experiments that reprocessed the Tf-ID vectorized data but it resulted in a tremendous decrease in accuracy, as seen in Figure 9.

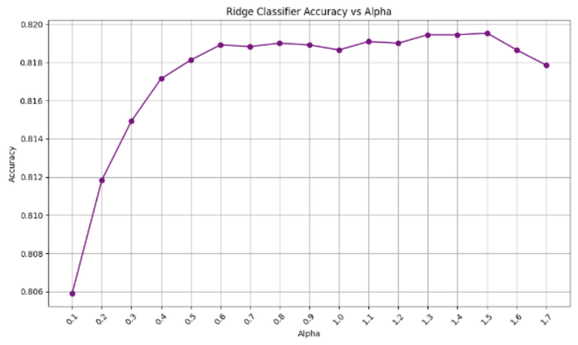


Figure 7: Accuracy for knn model after Tf-Id change

We then switched to using cosine similarity proximity measure, and we saw that k values after 42 converged in an accuracy score of about 0.653 with the highest accuracy value of 0.65342 and F1 score of 0.6372 at k=50.

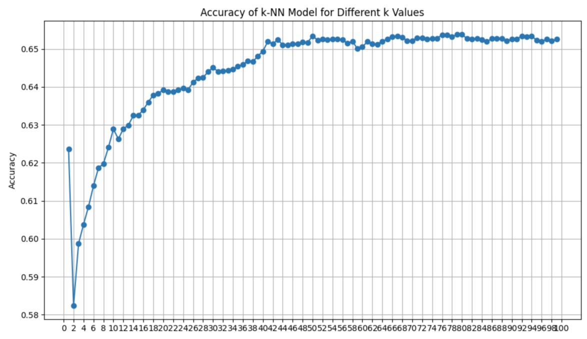
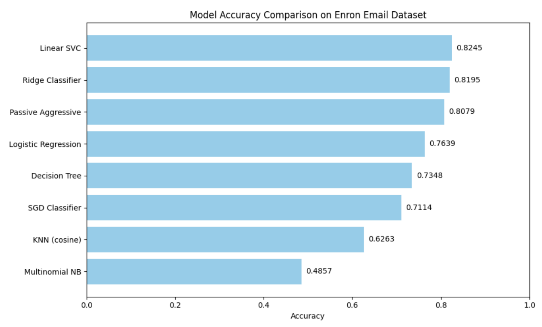


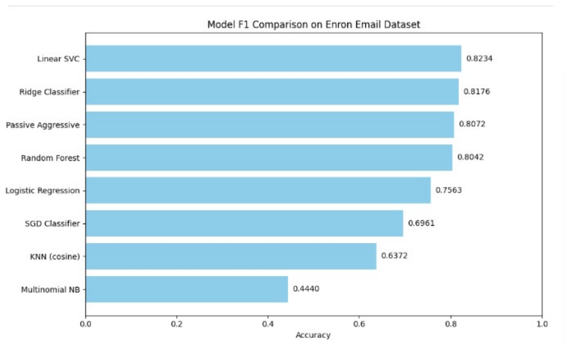
Figure 8: Accuracy for knn by k value using cosine similarity

5 Model Comparison

Our best performing model was Linear SVC, followed by Ridge Classifier, and Passive Aggressive, close behind. Logistic regression and SGD classifier lagged a bit behind, but KNN and Multinomial NB turned out to be very inaccurate methods even after our experimentations.



(a) Accuracy comparison between the models



(b) F1 score comparison between the models

Figure 9: Model performance comparisons

6 Conclusion

We had learned how to process textual data such as emails and process it in order to build a classification model that can predict who wrote an email by its content with a probability of 0.8245 using the LinearSVC Classification model, with the context of the Enron email database. This experiment also showed the importance of preprocessing the data as it had a significant effect on the accuracy of our model, as well as model choice as there had been widely different accuracy scores depending on the model.

In terms of future ideas, we could have experimented more with feature extraction and what data we incorporated. There was a lot more that we could have done in terms of feature extraction, as we only used emails from the sent items folder. While this had made labels much easier to retrieve, it had also reduced the number of available data that could have been incorporated. Furthermore, some of the emails that we used for this project were not useful and added noise to the data. There could have been more done in the process of cleaning and preparing the data for training, such as feature reduction or similar techniques. We had attempted to conduct PCA, but that simply lowered our accuracy scores without substantially increasing the speed of our models. Future researchers might also want to try the model with different sizes of training sets, both in terms of the sheer number of emails but also the number of users.

References

[dev25a] scikit-learn developers. Linear models. https://scikit-learn.org/stable/modules/linear_model.html, 2025. Accessed: 2025-06-11.

[dev25b] scikit-learn developers. sklearn.feature_extraction.text.tfidfvectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html, 2025. Accessed: 2025-06-11.

[dev25c] scikit-learn developers. sklearn.linear_model.ridgeclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html, 2025. Accessed: 2025-06-11.

[KY04] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. *Machine Learning: ECML 2004. Lecture Notes in Computer Science*, 3201:217–226, 2004.

[sci25a] scikit-learn Developers. sklearn.svm.linearsvc — linear support vector classification. Online documentation, https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html, 2025. Accessed: 2025-06-10.

[sci25b] scikit-learn Developers. Support vector machines. Online documentation, https://scikit-learn.org/stable/modules/svm.html, 2025. Accessed: 2025-06-10.