Classifying Concord: Machine Learning Meets Transcendentalism

Table of Contents

Introduction

I wanted to show methods for classifying text using a novel dataset rather than one of the toy ones so often used for tutorials. What I settled on eventually was to classify text to attribute it to the original author. The appendix includes the project repository link where all code and data used in this study can be accessed.

Ralph Waldo Emerson and Henry David Thoreau both resided in mid-19th-century Concord, Massachusetts. Their writings and philosophical exchanges contributed to American Transcendentalism, and both produced literary works that are still influential. Despite their shared location and intellectual community, their writing styles differ in identifiable ways.

Walden has always been one of my favorite books and had a big influence on me when I first read it in high school, but Emerson was a different story. I found his writing style wordy and frustrating. Recently I revisited the context of these works when I read *The Transcendentalists and Their World* (Gross, 2021). which sparked the idea to use the text from their now public domain classic works to create some novel machine learning datasets for practice.

This article is a walkthrough of using machine learning classification techniques combined with some preprocessing to distinguish between the writing of Emerson and Thoreau.

We created our dataset from two public domain works, one for each author. For Emerson, we chose *Essays: First Series* (1841), and for Thoreau, there was really no choice but *Walden, and On The Duty Of Civil Disobedience* (1854).

This dataset offers several practical advantages for educational purposes:

1. **Historical context**: The texts come from the same historical period and location.
2. **Controlled variables**: Both authors wrote philosophical prose on related themes.
3. **Clear differences**: Despite similarities in topic, the authors maintain distinct writing styles.
4. **Accessibility**: The works are in the public domain and widely available.

The texts were segmented into passages of 3-5 sentences each, creating a collection of labeled examples for training and testing classification models.

This article examines how different machine learning approaches perform on this classification task. We compare traditional methods, including logistic regression, random forests, and support vector machines, with newer transformer-based models.

The analysis serves both technical and humanities-oriented readers by demonstrating how computational methods can quantify stylistic differences already noted qualitatively.

The Evolution of Machine Learning Classification

Text classification has developed substantially over the years. In the 1960s, early methods relied on hand-crafted rules, then statistical techniques took over. Advanced neural networks joined simpler statistical models in the following years, notably recurrent neural networks. More recently, transformer-based models have gained prominence in this and most other natural language processing (NLP) use cases from 2018 onward.

1. **Rule-based systems** (1960s-1980s): Manually crafted rules and decision trees
2. **Statistical methods** (1990s-2000s): Naïve Bayes, Support Vector Machines, and Logistic Regression
3. **Neural network approaches** (2010s): recurrent neural networks (RNNs) and convolutional neural networks (CNNs)
4. **Transformer models** (2018-present): BERT, GPT, and their derivatives

Each era brought improvements in accuracy and capability. The field expanded from simple categorization to include sentiment analysis, authorship attribution, and stylometry—the quantitative study of writing style.

In our Concord project, we compare methods across this range, from traditional TF-IDF based classification to transformer-based models.

Creating a Unique Literary Dataset

Data Acquisition

First, we selected two representative texts from Project Gutenberg's public domain collection.

This programmatic approach ensures reproducibility and eliminates manual steps.

```
emerson_txt_url = "https://www.gutenberg.org/ebooks/16643.txt.utf-8"
thoreau_txt_url = "https://www.gutenberg.org/ebooks/205.txt.utf-8"

def download_file(url):
    local_filename = Path(url.split('/')[-1])
    result = requests.get(url)
    result.raise_for_status()
    with open(local_filename, "wb") as f:
        f.write(result.content)
    return local_filename

emerson_file = download_file(emerson_txt_url)
thoreau_file = download_file(thoreau_txt_url)
```

This programmatic approach ensures reproducibility and automation, eliminating the need for manual file handling.

Text Preprocessing

Both works contain Project Gutenberg boilerplate text that needs removal. We identify the start of the actual content by searching for the standard "START OF THE PROJECT GUTENBERG EBOOK" marker:

```
def trim_frontmatter(filename):
    with open(filename) as f:
        lines = f.readlines()
```

```
    n_trim_lines = 0
    for i, line in enumerate(lines):
        if "START OF THE PROJECT GUTENBERG EBOOK" in line:
            n_trim_lines = i + 1
            break
    trimmed_lines = lines[n_trim_lines:]
    trimmed_content = '\n'.join(trimmed_lines)
    new_filename = f"trimmed_{filename}"
    with open(new_filename, "w") as f:
        f.write(trimmed_content)
    return new_filename
```

Text Segmentation

We segment the texts into manageable chunks of 3-5 sentences each. This creates a dataset with many examples for training and testing while preserving enough context to capture authorial style. We use spaCy, a powerful and popular natural language processing library, to perform the segmentation.

This function creates randomly-sized windows of 3-5 sentences, providing natural text chunks while introducing variation in the dataset.

```
def segment_doc(filename):
    with open(filename) as f:
        text = f.read()
    doc = nlp(text)
    assert doc.has_annotation("SENT_START")
    sent_dq = deque()
    n = randint(3, 5)
    for sent in doc.sents:
        sent_dq.append(sent)
        if len(sent_dq) > n:
            sent_dq.popleft()
        snippet = " ".join(sent.text for sent in sent_dq)
        yield snippet
        n = randint(3, 5)
        sent_dq.clear()
```

Dataset Creation

We create pandas DataFrames from these segments and label them with their respective authors:

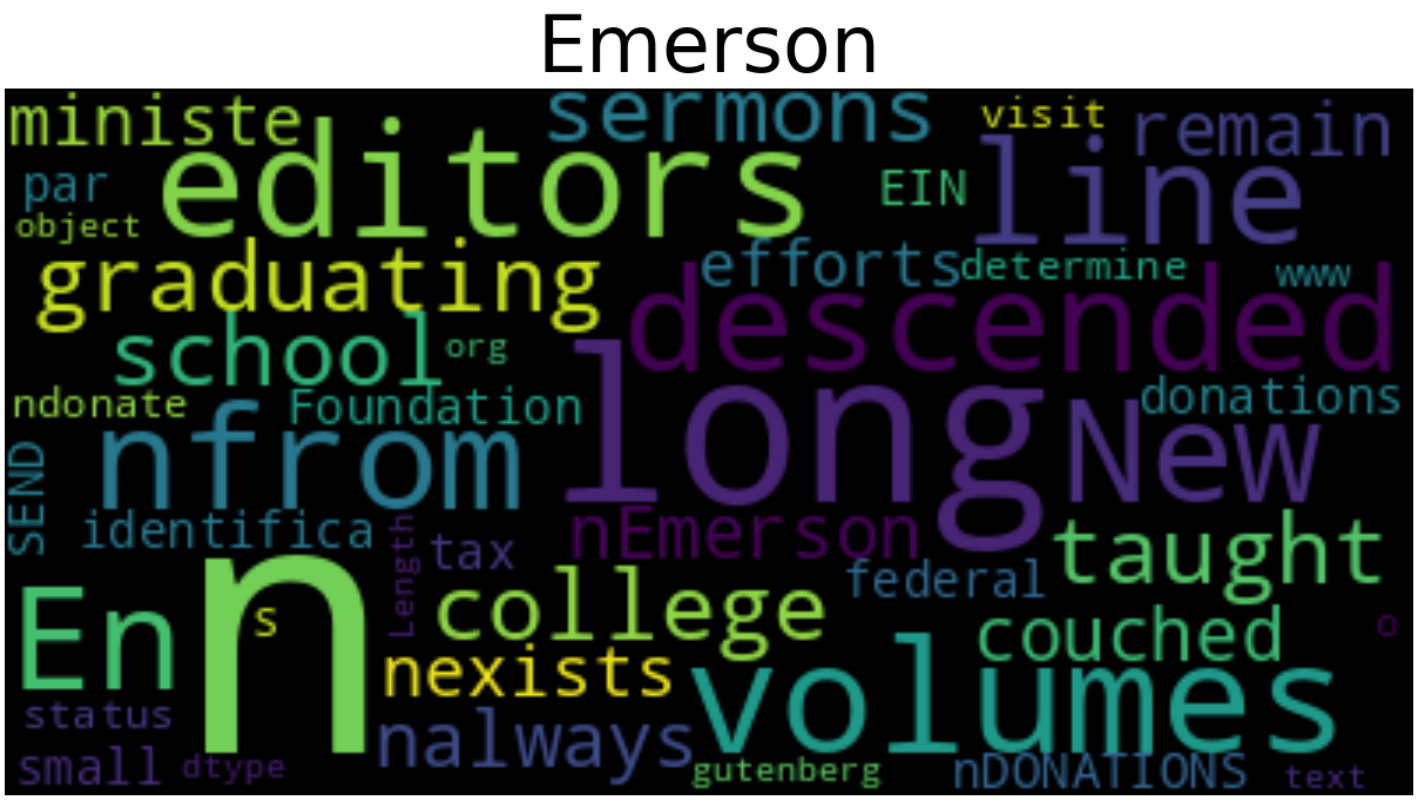
```
emerson_df = dataframe_from_file(trimmed_emerson_file)
emerson_df["label"] = "emerson"

thoreau_df = dataframe_from_file(trimmed_thoreau_file)
thoreau_df["label"] = "thoreau"

combined_df = pd.concat([emerson_df, thoreau_df])
combined_df = shuffle(combined_df, random_state=7919)
```

The result is a dataset of 1,911 text segments (1,064 from Emerson and 847 from Thoreau), each labeled with its author. This balance helps prevent biases in our models.

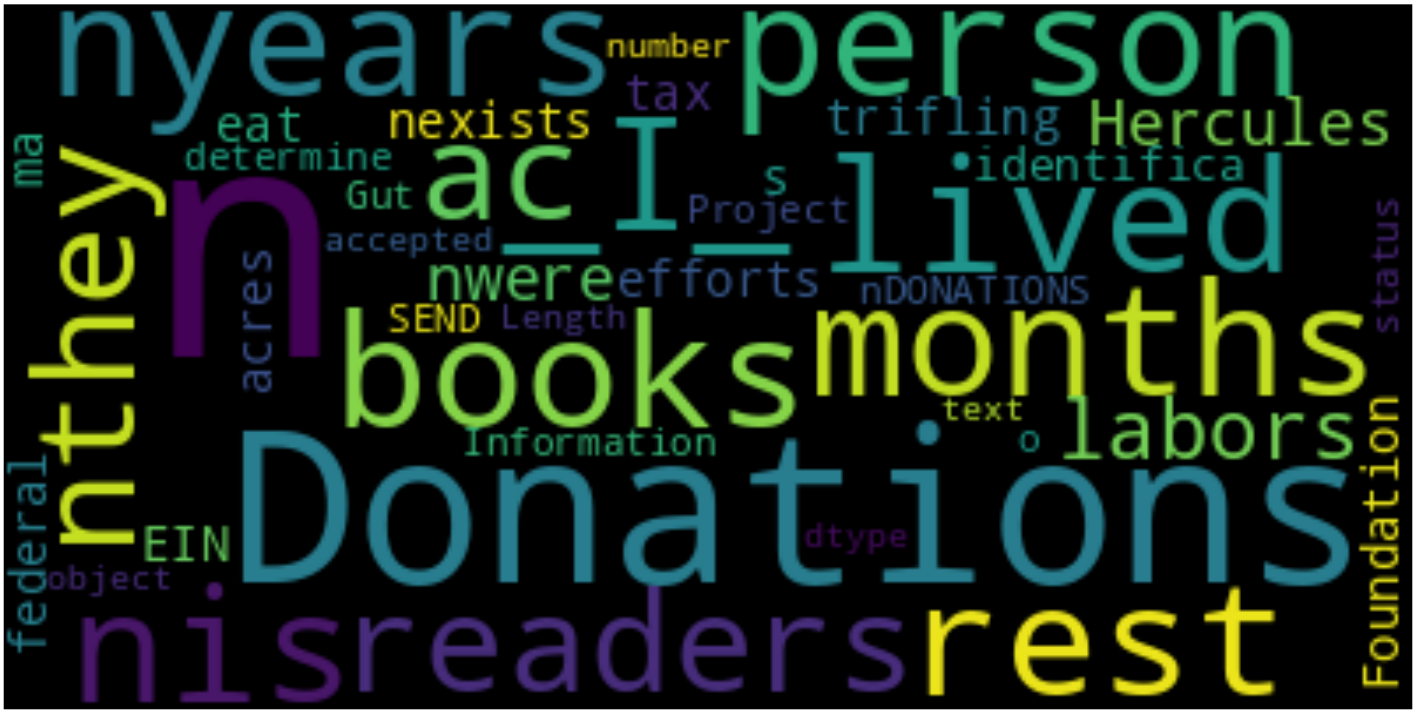
Text Representation Techniques

Visualization with Word Clouds

Before diving into classification, we visualize the most frequent words used by each author through word clouds, excluding common stopwords:



Thoreau



These visualizations already reveal interesting differences in vocabulary. Emerson's text prominently features abstract terms like "soul," "nature," "truth," and "power," reflecting his focus on philosophical concepts. Thoreau's cloud emphasizes more concrete words like "house," "wood," "water," and "life," mirroring his practical observations at Walden Pond.

Text Preprocessing for ML

For machine learning, we preprocess the text using spaCy to remove stopwords and perform lemmatization, which reduces words to their base forms:

```
final_text = []
for index, entry in enumerate(combined_df['text']):
    doc = nlp(entry.lower())
    Final_words = []
    for word in doc:
        if not word.is_stop and not word.is_punct:
            Final_words.append(word.lemma_)
    final_text.append(' '.join(Final_words))

combined_df['final_text'] = final_text
```

TF-IDF Vectorization

For our traditional machine learning models, we convert text to numerical features using Term Frequency-Inverse Document Frequency (TF-IDF) vectorization:

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(combined_df["final_text"])
y = combined_df["label"]
```

TF-IDF represents each text segment as a vector, where each dimension corresponds to a term in the vocabulary. The value combines how frequently the term appears in the document (TF) and how unique it is across all documents (IDF). This balances common words against distinctive ones that might better differentiate the authors.

Traditional ML Classification Models

We split our dataset into training (80%) and test (20%) sets to evaluate model performance:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=4909)
```

Logistic Regression

We begin with logistic regression, a simple but effective linear classifier:

```
lr_model = LogisticRegression(solver='saga', random_state=8102, n_jobs=-2)
lr_model.fit(x_train, y_train)
y_pred = lr_model.predict(x_test)
```

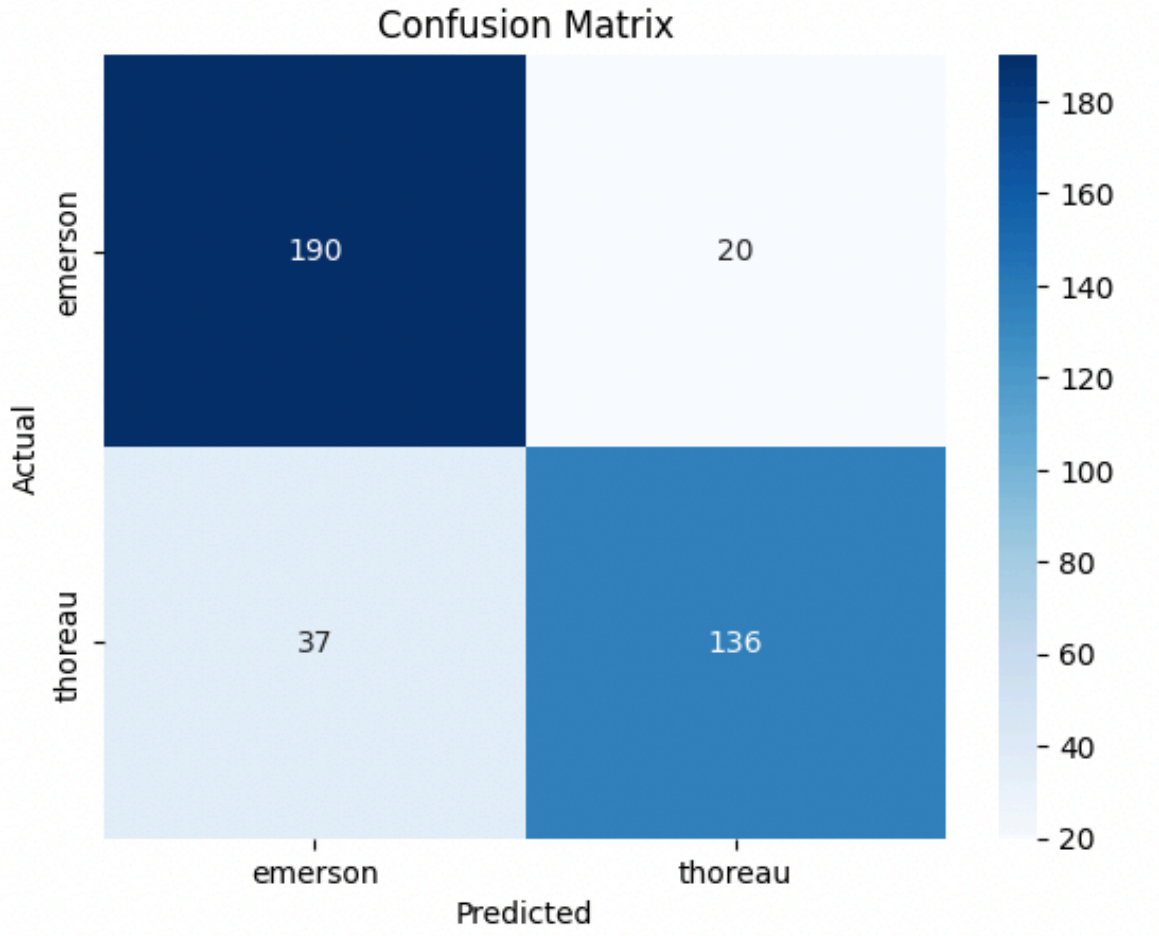
Results:

```
precision    recall  f1-score   support

emerson       0.84      0.90      0.87       210
thoreau       0.87      0.79      0.83       173

accuracy                          0.85       383
```

Logistic Regression

	precision	recall	f1-score	support
emerson	0.84	0.90	0.87	210
thoreau	0.87	0.79	0.83	173
accuracy			0.85	383
macro avg	0.85	0.85	0.85	383
weighted avg	0.85	0.85	0.85	383

Test accuracy: 0.8511749347258486



Logistic regression achieves an impressive 85% accuracy, correctly identifying the author in most cases.

Here we have included confusion matrix. For space we will omit this for most of the models, but they are available in the GitHub repo.

Random Forest

Next, we implement a Random Forest classifier, which creates an ensemble of decision trees:

```
rf = RandomForestClassifier()
rf.fit(x_train, y_train)
y_pred_rf = rf.predict(x_test)
```

Results:

```
precision    recall  f1-score   support

emerson       0.82      0.88      0.85       210
thoreau       0.84      0.76      0.80       173

accuracy                          0.83       383
```

The Random Forest model achieves 83% accuracy, slightly lower than logistic regression.

Support Vector Machine

We also implement a Support Vector Machine with a radial basis function kernel:

```
clf = svm.SVC(kernel='rbf')
clf.fit(x_train, y_train)
y_pred_svm = clf.predict(x_test)
```

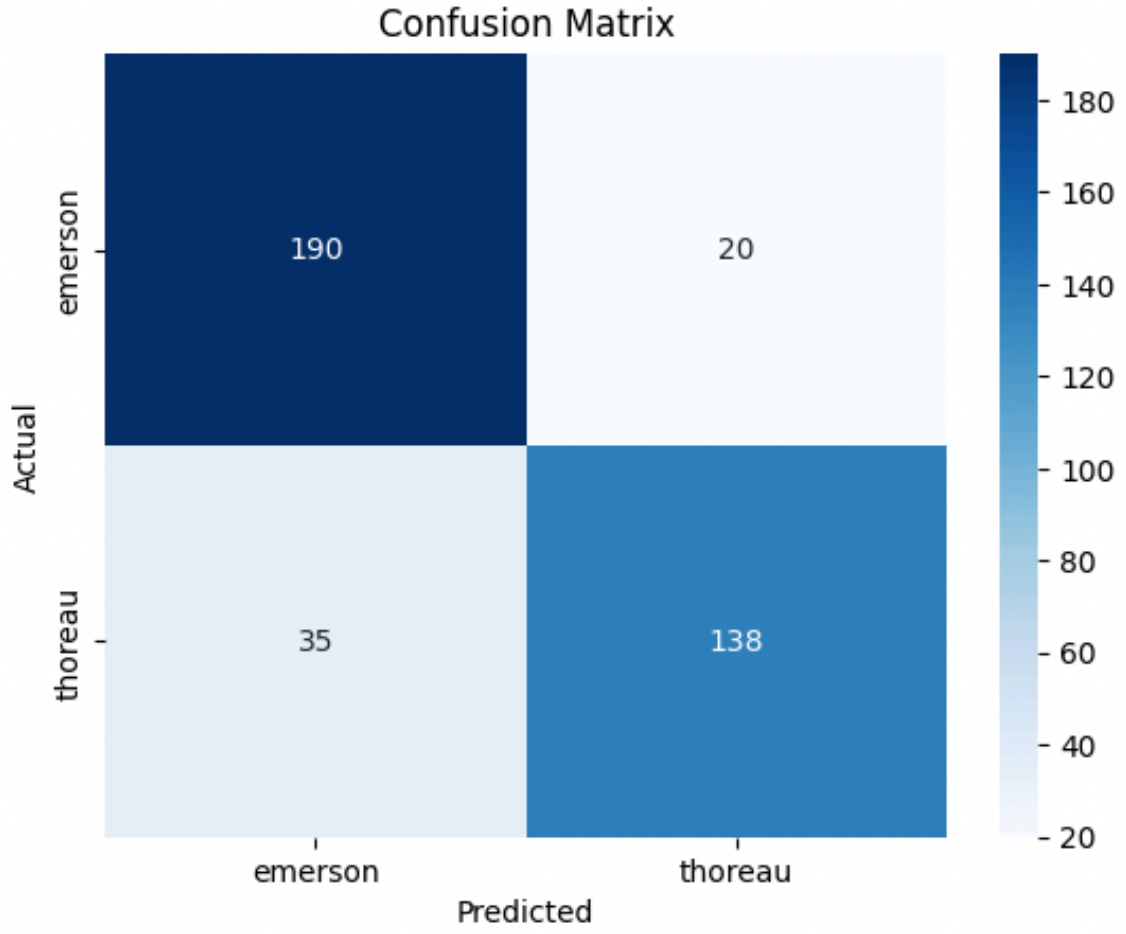
Results:

```
precision    recall  f1-score   support

emerson       0.84      0.90      0.87       210
thoreau       0.87      0.80      0.83       173

accuracy                          0.86       383
```

SVM

	precision	recall	f1-score	support
emerson	0.84	0.90	0.87	210
thoreau	0.87	0.80	0.83	173
accuracy			0.86	383
macro avg	0.86	0.85	0.85	383
weighted avg	0.86	0.86	0.86	383

Test accuracy: 0.856396866840731



The SVM achieves our best traditional model performance at 86% accuracy. I think SVM's are somewhat underappreciated amid all the neural network hype. They train in an amount of time that is quite zippy compared to more complex models that do not always beat them without a lot of training data and more risk of overfitting.

Deep Learning and Transformer Models

Transformer models have revolutionized NLP tasks by capturing complex contextual relationships in text. Here

Transformer models have revolutionized NLP tasks by capturing complex contextual relationships in text. Here we use DistilBERT, a lightweight version of BERT (Bidirectional Encoder Representations from Transformers). In 2025, this is not exactly state of the art, however you can train or fine-tune it easily with modest resources. I did this entire project on Google Colab for free.

Feature Extraction Approach

First, we use DistilBERT as a feature extractor, employing its hidden states as input to traditional classifiers. This method is useful in situations where you may not have enough data to effectively train a model from scratch, but you want something more powerful than simple feature engineering techniques.

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = AutoModel.from_pretrained("distilbert-base-uncased")

# Tokenize text and get hidden states
with torch.no_grad():
    hidden_train = model(**x_train_tok)
    hidden_test = model(**x_test_tok)
    # Get the [CLS] hidden states
    cls_train = hidden_train.last_hidden_state[:,0,:]
    cls_test = hidden_test.last_hidden_state[:,0,:]
```

There is a fair bit of code we are leaving out of the article to get things in a form to make the Transformers library happy here since everything before this point was setup with scikit learn.

We then use these features with our traditional classifiers:

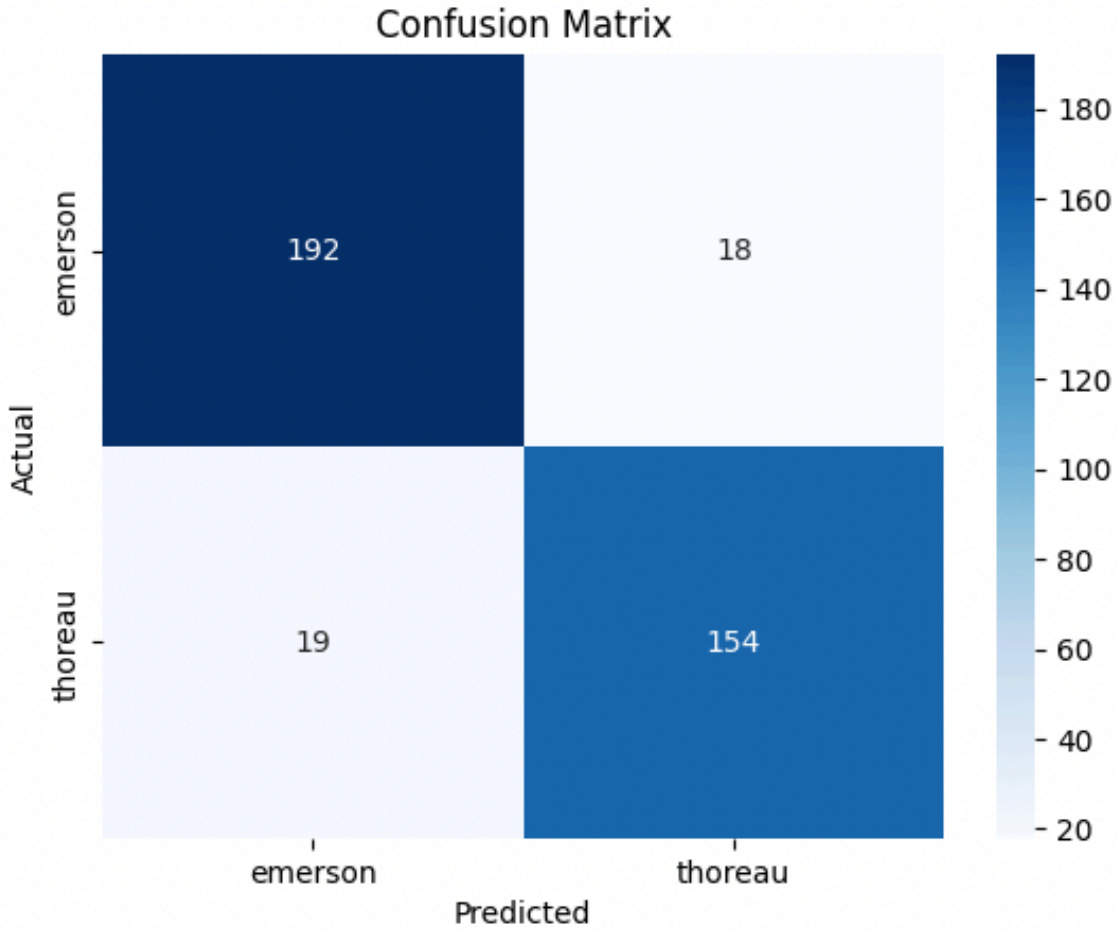
Logistic Regression on DistilBERT hidden states:

```
precision    recall  f1-score   support

emerson       0.91      0.91      0.91       210
thoreau       0.90      0.89      0.89       173

accuracy                          0.90       383
```

Logistic Regression on DistilBERT hidden states

	precision	recall	f1-score	support
emerson	0.91	0.91	0.91	210
thoreau	0.90	0.89	0.89	173
accuracy			0.90	383
macro avg	0.90	0.90	0.90	383
weighted avg	0.90	0.90	0.90	383

Test accuracy: 0.9033942558746736



SVM on DistilBERT hidden states:

	precision	recall	f1-score	support
emerson	0.88	0.92	0.90	210
thoreau	0.90	0.85	0.87	173
accuracy			0.89	383

This hybrid approach shows a significant improvement, with accuracy increasing to 90%

This hybrid approach shows a significant improvement, with accuracy increasing to 90%.

Fine-tuning DistilBERT

And finally, we fine-tune DistilBERT specifically for our classification task

```
from transformers import DistilBertForSequenceClassification

model = DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased', num_labels=2
)

# create our optimizer
from torch.optim import AdamW

optimizer = AdamW(model.parameters(), lr=5e-5)
# a bunch of stuff for preprocessing you can find in GitHub...
```

Here we have left out a bunch of preprocessing code and other boiler plate for brevity which you can find in the repo.

```
from transformers import Trainer, TrainingArguments, DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

training_args = TrainingArguments(
    output_dir="./results",
    learning_rate=2e-4,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    logging_strategy="epoch"
)

# Define Trainer object for training the model
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    data_collator=data_collator,
)
```

```
trainer.train()
```

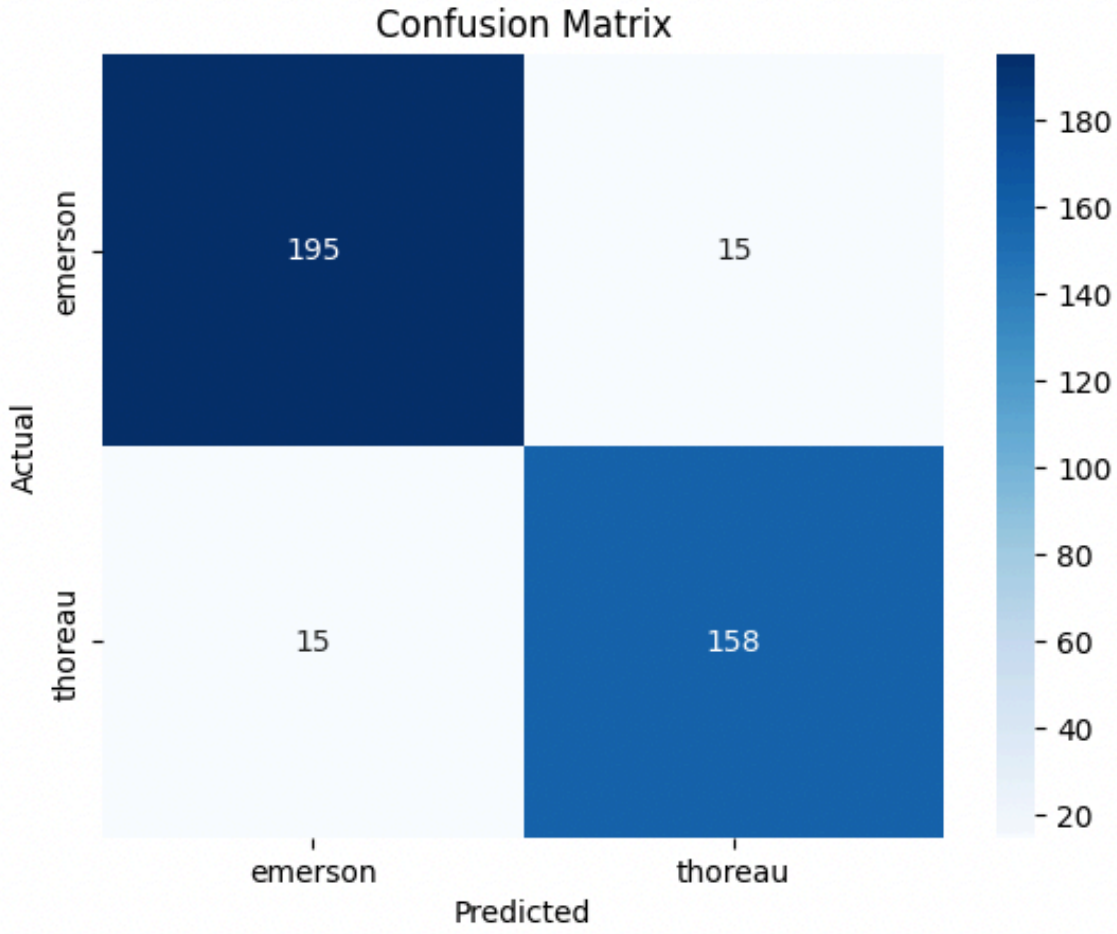
Results:

```
precision    recall  f1-score   support

emerson       0.93      0.93      0.93       210
thoreau       0.91      0.91      0.91       173

accuracy                          0.92       383
```

Fine-tuned DistilBERT

	precision	recall	f1-score	support
0	0.93	0.93	0.93	210
1	0.91	0.91	0.91	173
accuracy			0.92	383
macro avg	0.92	0.92	0.92	383
weighted avg	0.92	0.92	0.92	383

Test accuracy: 0.9216710182767625



The fine-tuned DistilBERT achieves our best performance at 92% accuracy, demonstrating how powerful transformer models are for capturing subtle stylistic differences.

Analysis of Results

Performance Comparison

Performance comparison

Looking at our results across models:

1. Traditional ML models (TF-IDF + classifiers): 83-86% accuracy
2. DistilBERT features + traditional classifiers: 89-90% accuracy
3. Fine-tuned DistilBERT: 92% accuracy

This progression demonstrates the advantages of modern transformer models, which capture contextual information and semantic relationships beyond what bag-of-words approaches can represent.

Misclassification Analysis

Examining misclassified examples provides linguistic insights:

```
for i, (txt, lbl, pred) in enumerate(zip(x_test_trans, y_test_trans, y_pred_trans)):
    if lbl != pred:
        print(f"{lbl=},{pred=}")
        row =(my_cat_labels[lbl], my_cat_labels[pred], txt)
        rows.append(row)
```

Several patterns emerge in the 30 misclassified examples (out of 383):

1. **Project Gutenberg boilerplate**: Many misclassifications involve similar license text that appears in both works
2. **Short segments**: Brief text snippets provide insufficient stylistic markers
3. **Universal themes**: When both authors discuss similar philosophical concepts, the distinction blurs
4. **Quotations**: When either author quotes someone else, their personal style is diminished

One interesting observation is that Thoreau's more practical descriptions are rarely misclassified, while his philosophical musings more frequently get attributed to Emerson. This aligns with our understanding of their writing styles—Thoreau's concrete observations of nature are distinctive, while both authors share transcendentalist philosophical language.

Model Interpretability

While transformer models achieve the highest accuracy, they function as "black boxes" compared to traditional models. With logistic regression, we can examine the coefficients to identify the most influential words for classification:

The most strongly Emerson-associated terms include abstract nouns and adjectives like "soul," "divine," "intellect," and "universal." Thoreau's distinctive vocabulary includes more concrete terms like "pond," "woods," "house," and action verbs reflecting his practical experiences at Walden.

Conclusion

This analysis of Emerson and Thoreau's writing demonstrates the application of machine learning to literary style classification. The results show that computational methods can differentiate between these authors with measurable accuracy. The fine-tuned DistilBERT model achieved 92% accuracy on the test set, while traditional machine learning approaches reached 83-86% accuracy.

The project connects multiple disciplines: computational linguistics, machine learning, and literary studies. For the ML community, it provides a concrete case study in text classification across different algorithmic approaches. For literary scholars, it offers quantitative evidence for stylistic differences previously described through close reading.

The performance progression from traditional to transformer-based models reflects broader developments in natural language processing. TF-IDF vectorization with classic algorithms provided a functional baseline (83-86% accuracy), while transformer models improved on these results (92% accuracy) by capturing contextual relationships between words.

The misclassification analysis revealed specific patterns in the 30 misclassified examples (from 383 total test examples):

1. Project Gutenberg boilerplate text caused confusion for all models
2. Short text segments provided insufficient data for accurate classification
3. Philosophical passages where both authors addressed similar concepts were harder to distinguish
4. Quoted material from other sources disrupted the authors' characteristic styles

Thoreau's nature descriptions and practical observations were rarely misclassified, while his philosophical reflections were sometimes attributed to Emerson. This aligns with the known characteristics of their writing—Thoreau's concrete observations stand distinct from Emerson's typically abstract style.

Future Research Directions

This work suggests several practical extensions:

1. Testing with additional works from each author to determine if the classification accuracy holds across their complete corpus
2. Including other Transcendentalist writers to create a multi-class classification problem
3. Analyzing how each author's style changed over time by training on chronologically labeled texts
4. Creating topic-based classifiers to identify patterns in how each author addressed specific subjects
5. Implementing explainable AI techniques to better understand what aspects of style the transformer models detect

This project demonstrates one practical application of text classification to literary analysis. As NLP methods continue to advance, their application to humanities research can provide new analytical tools while offering challenging test cases for ML development.

The classification results confirm that Emerson and Thoreau maintained distinct writing styles despite their

shared philosophical interests and geographic proximity. These computational findings support established literary scholarship through quantitative measurement of stylistic differences between these significant American authors.

References

Literary Works and Sources

1. Emerson, R.W. (1841). *Essays: First Series*. James Munroe and Company. Retrieved from Project Gutenberg: https://www.gutenberg.org/ebooks/16643
2. Thoreau, H.D. (1854). *Walden, and On The Duty Of Civil Disobedience*. Ticknor and Fields. Retrieved from Project Gutenberg: https://www.gutenberg.org/ebooks/205
3. Richardson, R.D. (1995). *Emerson: The Mind on Fire*. University of California Press.
4. Walls, L.D. (2017). *Henry David Thoreau: A Life*. University of Chicago Press.
5. Buell, L. (2003). *Emerson*. Harvard University Press.

Machine Learning and NLP References

6. Devlin, J., Chang, M.W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of NAACL-HLT 2019*, 4171–4186. https://doi.org/10.18653/v1/N19-1423
7. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
8. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.
9. Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1), 1–47. https://doi.org/10.1145/505282.505283
10. Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
12. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45.

Computational Stylometry and Literary Analysis

13. Burrows, J. (2002). 'Delta': a Measure of Stylistic Difference and a Guide to Likely Authorship. *Literary and Linguistic Computing*, 17(3), 267–287. https://doi.org/10.1093/llc/17.3.267
14. Jockers, M.L. (2013). *Macroanalysis: Digital Methods and Literary History*. University of Illinois Press.
15. Stamatatos, E. (2009). A survey of modern authorship attribution methods. *Journal of the American Society for Information Science and Technology*, 60(3), 538-556. https://doi.org/10.1002/asi.21001
16. Underwood, T. (2019). *Distant Horizons: Digital Evidence and Literary Change*. University of Chicago Press.

Python Libraries and Tools

17. Pandas Development Team (2020). pandas-dev/pandas: Pandas. Zenodo. https://doi.org/10.5281/zenodo.3509134
18. Harris, C.R., Millman, K.J., van der Walt, S.J. et al. (2020). Array programming with NumPy. *Nature*, 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2
19. Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55
20. Waskom, M.L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60), 3021. https://doi.org/10.21105/joss.03021
21. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, 8026-8037.

Appendix: Project Repository

Installation and Usage

An alternative to these steps is to use Google Colab to import the notebook directly from GitHub.

1. Clone the repository:

```
git clone https://github.com/yourusername/classifying_concord.git
cd classifying_concord
```

2. Install required packages:

```
pip install -r requirements.txt
```

3. Download the spaCy English model:

```
python -m spacy download en_core_web_sm
```

4. Run the Jupyter notebook:

```
jupyter notebook supervised_ML_identify_author.ipynb
```