CPSC 490
SENIOR PROJECT

Authorship Attribution, Natural Language Processing, And Beyond

Advisor: Professor Dana Angluin
Department of Computer Science

By Lien Le Hong Tran

I. ABSTRACT

Authorship attribution is the problem of analyzing and characterizing an individual's writing style in order to determine the most likely author for an unlabeled body of natural language text. In recent years, the problem has been approached from a statistical angle by employing machine learning techniques. This senior project aims to explore various learning models to solve authorship attribution in conjunction with a number of linguistic methods in order to understand what governs writing style and what features play significant roles in distinguishing different authors.

Natural language generation is the problem of generating language computationally. This is a relatively new field, which in recent years has taken the advantage of powerful machine learning techniques to produce text. This project will examine a specific learning model employing recurrent neural networks to generate text. The text will then be evaluated by models that were previously trained to predict authorship. The combined application of these problems shows interesting results.

II. INTRODUCTION

Authorship attribution has long been an important problem in linguistics. Stamatatos provides a survey of the field in 2009 from a linguistic point of view. With the recent development of machine learning, authorship attribution has become a task to which an enormous amount of computational methods are applied. Koppel et al. explains the general problems of authorship attribution, as well as computational methods used to solve them in their survey from 2009.

Machine Learning has become the most cutting edge set of techniques used today to deal with the massive corpus of data that the world is generating, especially language and writing. Due to the breadth of the field, there are very few survey papers on the topic. Instead, machine learning techniques are often discussed in context of being applied to a certain domain of data to accomplish a specific task, such as categorizing images or learning word representations.

This project's original goal is to explore the literature for both of the above fields, and combine them in order learn patterns and analyze the writing of three classic writers: Jane Austen, Charles Dickens, and Mark Twain. It will then aim to answer one fundamental question: *Can we distinguish between the styles of writing of these authors using machine learning techniques?* Relevant works on the topic include Dieredich et al. 2003, Koppel et al. 2005, Stanczyk and Cyran 2007.

When it became clear that distinguishing the styles of these authors was doable, I began adding questions and problems that I would like to tackle during the course of this project. Therefore, the next portion of the project tries to answer the following question: *Can we generate language in the style of a particular author using machine learning?* Relevant works on this topic include Mikolov et al. 2010 and Lin et al. 2015.

The rest of the project consists of extensions that result from combining techniques discovered from the previous two parts. They are experiments that I carried out in order to better understand machine learning, writing styles, and the relationships between them.

III. PREPARATION

1. Learning and Research

I spent a significant portion of the project acquainting myself with the literature and applications of linguistics and machine learning, as both were relatively new to my knowledge. I had a particular vision of what problems I wanted to learn about and solve, but no idea of the existing research or challenges that researchers faced as they tackle these problems.

Linguistics

There are many software packages online that can linguistically break down natural language. In particular, I looked at NLTK, word2vec and spaCy to investigate what tools they delivered that would be useful to the project. Out of these, I eventually used spaCy to do part-of-speech tagging and sentence boundary detection for the final application.

Machine Learning

The reason I was interested in machine learning was because it has recently provided the state-of-the-art techniques to process language. I went through a number of tutorials and online resources to learn about the basics, including Michael Nielsen's online book "Neural Networks and Deep Learning" and Caltech's "Learning From Data" lectures by Professor Yaser Abu-Mostafa. I investigated several machine learning packages, including caffe, Torch, and Theano, besides serious attempts to read and understand the surrounding literature. In Spring 2016, I took Professor Taylor Arnold's STAT 365 "Data Mining and Machine Learning" course. The course taught me how to apply machine learning techniques using these readily available online software packages, as well as introduced me to a number of key papers on the topic.

2. Data

Project Gutenberg, an online library of free e-books, provided all the text data for this project. Because I was primarily interested in style of writing, I chose novels from three authors who

wrote natively in the English language: Jane Austen, Charles Dickens and Mark Twain. It would be possible to conduct the project with non-English writers as well. However, this would require open-source software packages for processing text in other languages—which are difficult to find—or training models on translated texts, which would defeat the purpose of learning writing styles.

	Jane Austen	Charles Dickens	Mark Twain
Size (Megabytes)	3.9	9.4	3.3
Number of lines	10196	37903	12671
Number of words	745757	9802754	635273

3. Dependencies and Deliverables

Python 2.7, Python's **numpy**, **pandas** and **sklearn** packages, **spaCy** and **PyEnchant** libraries, and **Torch** were installed for the execution of this project. The project's deliverables are a library of functions that processes literary texts into .csv data files, and uses them to train various machine learning models.

IV. PROJECTS

1. Distinguishing styles of writing

The first portion of the project is authorship attribution, aimed at training models to distinguish styles of writing between Austen, Dickens and Twain. More precisely, these models are binary models trained to predict whether or not the author that they were trained on wrote a certain piece of text. A typical test would entail presenting text to a model trained on a particular author, such as Austen, and seeing whether it returns 0 or 1. If the answer is 0, the model predicts that Austen did not write that text; if the answer is 1, then Austen wrote it.

This also means that individually, each model is unfit for identifying *who* wrote a piece of text. To answer that question, one would need to run all models on that text, and return the author of the model that answers in the affirmative. In the unfortunate case where more than one model returns an affirmative result, then a possible solution would be to pick a random author from those proposed, or get the answers in probabilities (if possible) and pick the model that produced the highest probability. If none of the models were trained on the actual author, then the ideal scenario is all models answer in the negative—which should happen if the data was representative and models trained well.

1.1 Methods and Preparation

Data Representation

One particular challenge with using machine learning for natural language processing is that machine learning models can only interpret numbers—something that language is not. Therefore, a significant portion of this project is spent on looking at methods of representing language numerically—or in other words, methods to convert text into vectors and matrices of data that encodes information in the text. Current research in the field has yet figured out a way to efficiently and completely encode words into vectors—one either has to lose some dimension of language, such as vocabulary or meaning, in return for data compactness, or deal with vectors in very high dimensions, which may require vast amounts of data and computing power to learn sufficiently. For this project, I followed the former approach.

I broke down language using two linguistic structures: part-of-speech tags and function words. Each produced a different set of features that I could use as input to my models. Therefore, I trained my models separately on each set of features, as well as their combined set.

Part-of-speech tag pairs

Part-of-speech tags denote the grammatical function that each word plays in a sentence. Part-of-speech tag pairs, as I will use them in this project, are pairs of part-of-speech tags that appear consecutively in a sentence. For example, in the sentence "*She went.*" the three

corresponding part-of-speech tags are [NOUN VERB PUNCT]. These construct 2 part-of-speech tag pairs, NOUN-VERB and VERB-PUNCT.

Part-of-speech tag pairs, although devoid of meaning, provide information about the grammatical structure of any given text. Models that learn from data characterized by part-of-speech tag pairs thus learn to categorize writing by segregating portions in the multidimensional space of grammar representation.

I used the spaCy library in Python to help me convert words into their part-of-speech tags. There are a total of 19 part-of-speech tags in the spaCy library. Out of these 19 tags, I created 19 x 19 = 361 pairs of possible consecutive part-of-speech tags and used them as features. The features count how many occurrences of each pair occur in a piece of text.

Something I could have done on top of this to make the model more robust is to eliminate part-of-speech pairs that never occur. This would have reduced the number of features in the data, which is desirable because it means reducing the sparseness of data in the representation space. Sparseness of data often leads to underfitting in machine learning techniques. That said, it could be possible that removing part-of-speech pairs that never occur wouldn't affect the results at all because all the values for those pairs are zero. In that case, the only advantage would be computational efficiency.

Function words

Function words are, by definition, words that do not contain content—as opposed to content words. Instead, they connect and create relationships between other content words. They could be understood of as part of the supporting framework for grammar and semantics.

I referred to the list of function words found on John and Muriel Higgins' website[1] for my feature set. I used it simply because it was the most accessible source online. The original list of 321 function words included contractions, where an example of a contraction is ***don't***—a contraction of ***do not***. However, since spaCy was capable of breaking contractions into their constituent parts, I removed the contractions and reduced the list to 280 words. Each word is a feature. That feature tracks the number of occurrences of that word in every piece of text that is a data point.

In retrospect, I should have condensed the Higgins' list of function words, simply because I disagreed that some of them were at all important, such as 'first', 'second', or 'third'. This would have helped reduce the number of features in the data—not unlike the case with part-of-speech tags.

[1] http://myweb.tiscali.co.uk/wordscape/museum/funcword.html

Data points

The collection of texts from Jane Austen, Charles Dickens and Mark Twain is a plentiful database of natural language. However, the texts need to be organized as multiple data points to be processed by machine learning models, one by one, to incrementally improve its model and better approximate the representational space for each category. Therefore, I divided all the texts into non-overlapping 10-sentence chunks and used each as a separate data point.

One could have used fewer sentences for each data point, e.g. 5 sentences, to produce twice as many data points at the cost of fewer features measured for each data point. To increase the number of data points, another approach could have been to allow overlapping sentences, i.e. the previous data point of 10 sentences overlaps with the next data point of 10 sentences for 5 of the sentences. This would have generated the same amount of data as dividing the texts into non-overlapping 5-sentence chunks without compromising feature measurements. Since the features only measure occurrences of features, data points generated from overlapping sentences would most likely not have been overwhelmingly similar to each other, while still individually capturing continuity of style.

1.2 Models and Implementation

Random forests

A random forest is a collection of decision trees. Decision trees learn to separate categories of data by identifying features that separate these categories. A collection of decision trees trained separately can then vote for the most likely category when faced with a prediction task, creating a forest. The random element comes in how each tree is trained. Usually, each tree can only train on a small random subset of all the features in order to minimize high correlation between trees in the forest.

Random forests, besides being able to be trained to classify whether or not an author wrote a piece of text—a binary task—could be trained to identify multiple authors as well. One interesting experiment would be to train random forests to predict who wrote a certain piece of text by having it learn the styles of multiple authors at once. However, since I wanted to use support vector machines and compare the results, I did not exploit this feature of random forests.

Support vector machines

Support vector machines can only carry out binary classification tasks. They accomplish this by trying to find a hyperplane in the feature space that separates two categories of data from each other. Their specialty is the ability to deal with high-dimensional spaces without exponentially increasing required computational power.

Due to these characteristics, support vector machines are suited to answer the binary question of whether or not an author wrote an unseen piece of text. They would most likely work better than other non-neural models with the vectors that I generate from analyzing part-of-speech tag pairs and function words, which have several hundred features each.

Implementation

a. Data pre-processing

The original novels from Project Gutenberg came in .html format. I reformatted them into .txt format.

I wrote a script (data_clean.py) to filter the .txt files of particular characters, or sequences of characters. In other words, I standardized the .txt files so that spaCy can process them at a later stage with more accuracy. For example, I needed to eliminate multiple consecutive new lines that results from reformatting .html into .txt, since spaCy for some obscure reason would tag this part-of-speech as ADJ (adjective). Other filters include deleting 'Chapter' and 'Volume', underscores that were used to signify italics in the text, Gutenberg notes, notes for illustrations (done manually for some novels that had them).

b. Data processing

To create the data for machine learning models, I wrote scripts to turn folders of .txt novels into .csv files. One script turned novels into counts of part-of-speech tag pairs; the other turned novels into counts of function words.

The process for each script is as follows:

- For each folder, traverse all files.
- For each file, separate the text into 10-sentence chunks by using spaCy's sentence boundary detector.
- For every chunk, turn sentences into part-of-speech tag pairs with the help of spaCy, or filter out function words.
- Count part-of-speech tag pairs or function words; put these numbers into a vector.
- Turn the final collection of vectors into a `pandas` dataframe, and export to .csv.

c. Model parameters

My model-training scripts import .csv files for their data. Both random forests and support vector machines are supervised models, which means that they need data to tell them which are the two types, and what types are the particular data points that train on. Therefore, depending on which author's writing style I want to learn, I need to edit the script to mark that author's 'type' parameter, marking 1 for the author whose style I want to learn, and 0 for the data from all other authors' data points.

The project was never intended to explore how much accuracy one could get with distinguishing these authors. Rather, it aimed at exploring whether we *can* distinguish between these authors with decent accuracy. Therefore, I did not spend time tuning my models. The parameters below were run for all training tasks that I did on distinguishing styles of writing.

Random forests:	175 trees, each trained on a subset of 15 features.
Support vector machines:	default settings for sklearn's svm.SVM() model.

Out of the entire data set, I used 25% as test data and 75% as training data (default setting).

1.3 Results

Multiple reruns of the models on test data show some fluctuations in random forest model, but none in support vector machines. The data set was unevenly split between all authors, as shown on page 4. More specifically, there were 2726 data points for Austen, 5694 data points for Dickens, and 2758 data points for Twain. Results are displayed in the tables below.

Part-of-speech tag pairs	Jane Austen	Charles Dickens	Mark Twain
Random Forests	0.861	0.833	0.865
Support Vector Machines	0.897	0.857	0.903

Function words	Jane Austen	Charles Dickens	Mark Twain
Random Forests	0.898	0.838	0.862
Support Vector Machines	0.936	0.864	0.910

Combined	Jane Austen	Charles Dickens	Mark Twain
Random Forests	0.898	0.854	0.875
Support Vector Machines	0.942	0.909	0.935

1.4 Analyzing style using Random Forests

One particular feature of random forests is that it is possible to see the ranking of importance for all features. Therefore, I can figure out what the random forests deem as the most distinguishing features of each author when pitted against the other two authors.

Below are samples of the ten most distinguishing features for each author. It is important to note that the order of importance varies for each run, due to random forests' randomization. Thus the term '*samples*'.

Jane Austen's 10 most distinguishing features

- she 0.0264363733062
- her 0.0259856072855
- CONJ-VERB 0.0179374043836
- be 0.0173356659824
- must 0.0162157038719
- PUNCT-VERB 0.0150229411784
- ADV-ADJ 0.0140277767659
- herself 0.0133087586464
- very 0.0132134102585
- PUNCT-SPACE 0.0130553712746

Charles Dickens' 10 most distinguishing features

- PUNCT-VERB 0.0483842085148
- NOUN-PUNCT 0.0359666346531
- PUNCT-PUNCT 0.0294989807895
- PUNCT-ADP 0.0207091271091
- PUNCT-SPACE 0.0161603367598
- ADP-DET 0.0126785795154
- PUNCT-NOUN 0.012353578433
- ADV-PUNCT 0.0117416161885
- me 0.0116850716046
- SPACE-PUNCT 0.0114369300976

Mark Twain's most distinguishing features

- PUNCT-VERB 0.0334190281827
- PUNCT-ADP 0.0238454942259
- NOUN-PUNCT 0.0196812287667

- NOUN-CONJ 0.0162292828107
- ADP-ADJ 0.0161981748681
- have 0.015774725304
- PUNCT-PUNCT 0.0144975765855
- ADJ-NOUN 0.0141694199635
- around 0.0134944771271
- her 0.0128228408786

It is important to understand that these most important features are determined *with respect* to the non-affirmative data—data that represents what the target of learning *is not*. In this case, that data is text from the two other authors. If I were to include other authors' writing in the training data, the resulting most important features for each author would most likely be different. This will become clearer in part 5 of this project.

1.5 Discussion

The classification results are particularly good: all models achieve a correct classification rate of above 80% for part-of-speech tags features and function word features, and above 85% for the combined case. Particularly, support vector machines for the combined case achieved above 90% for all authors. With this, I can conclude that they distinguish between Austen, Dickens and Twain writing fairly well.

The superior results of the combined features make sense: with more features, it is easier to identify the distinguishing features for each author. However, there is a dimensionality trade-off: with more dimensions, the data points become incredibly sparse, and it will most likely require more data to train the models well. I did not increase the amount of training or testing data that I had in my data set for the combined features. I have speculated about how the results of the exercise may be somewhat less dependable for this reason. However, it is difficult to test. One possibility is to get more text from these authors. To do so, I will most likely need to reassess how I split the novels into data points so as to generate more data points all together from the same amount of text.

An additional observation that I made while repeatedly training these models, however, is that the rate for random forests tend to fluctuate, while the rate for support vector machines were always constant, given the same model parameters and same data set. The random forest case makes sense: for each iteration, the set of features that each tree is trained on is randomized, thus resulting in different rates every time. The algorithm for support vector machines is fixed and iterative with no randomization.

2. Generating text in particular styles

The second portion of the project is natural language generation in particular styles of writing. I investigated whether or not it is possible for computer to learn how to write like a certain person, or in a given style. Given the data collected for the previous portion of this project, I applied this question to Austen, Dickens and Twain. The goal is to utilize the writing from any author, and let a computer learn patterns that appear in that writing, so as to be able to reproduce those patterns by itself.

This problem of generating text in particular styles is completely different from distinguishing styles of writing. When distinguishing styles of writing, as in the previous part, it was possible to trim down the data to the most important structure, i.e. grammar, and try to find distinctive patterns from there. The task of learning what it different about each style does not require understanding the text holistically, as it is; we only need to collect data from certain aspects of the text in order to start some sort of analysis for patterns.

The same features do not apply when it comes to generating text. If we looked at the problem from the perspective of understanding substructures and then try to put them together, the task will be immensely difficult. For example, simply knowing the part-of-speech tag pair patterns or function word patterns cannot help generate text. If we were to want to use that knowledge, we will also need to learn how to deduce words from speech tags in some probabilistic fashion—which is not impossible, but certainly challenging—before combining them into a generator.

With this in mind, I researched more generally for what people have used to generate texts. What I found and settled on was a character-level recurrent neural network, which is carefully and clearly expounded in Andrej Karpathy's blog post *The Unreasonable Effectiveness of Recurrent Neural Networks*[2].

2.1 Methods and Preparation

Recurrent Neural Networks

Recurrent neural networks are neural networks that consider inputs *temporally*. In other words, they digest inputs in a stream rather than all at once, and learn the context of each input with respect to what came before and after it. They are not limited by variable input sizes, which, for example, aren't a problem for images, but would be a problem with words and sentences. Recurrent neural networks do this by processing each input with both a weight matrix and the output of the network from the previous input.

[2] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Character-level training on text

For this project, my recurrent neural network trained on text at a character level. This means that each input is a character in the text, both alphabetical and non-alphabetical characters (space, quotations, comma, etc.). The network learns the probabilities of each character based on the characters that came before it in the learning process. Then, given a random starting seed, this probability model can generate characters, one at a time, based on the probabilities that result from the weight vector and the previous input.

2.2 Models and Implementation

Preprocessing

This part of the project did not require an extensive amount of data processing. Karpathy's model needed only one input.txt file that included all the text for a particular author. I concatenated all the novels for each author, and generated an input.txt file for each.

Had the model been word-level, some thought should be given to determining a good way to denote the boundary between 2 novels—we do not want the words of one novel affect another. However, even if that were the case, it might have been statistically insignificant. But the model is character-based. The effect of consecutive novels should be trivial because the character model learns only local patterns that are unlikely to extend beyond one or two sentences. That said, that is difficult to say for certain without further careful investigation.

Parameters and Time

The parameters I used to train models for each author were as follows:

RNN size:	200
Number of layers:	2
Drop out rate:	0.5

- RNN size is the size of vectors used to process each input. The larger the data set, the bigger this size should be. I could have varied the size for each author because I had different amount of text for each author, but I decided to standardize it. This makes it easier to measure variations in the results, if needed.
- The number of layers also depends on data set size. Usually, each layer adds processing power for more complex and abstract patterns. The recommended number of layers is usually 2 to 3.
- Drop out rate is the probability at which each weight drops out of the computation for an input, in order to decrease high correlation between the weights.

The parameters that I used to generate text from each model were as follows:

Length:	3000 characters
Temperature:	1

- Length is the number of characters to generate for each chunk. I supposed that 3000 would be enough to make at least 10 sentences. After generating these 3000-character texts, I would trim and filter them to make sure the whole text comprises of exactly 10 sentences.
- Temperature is the conservativeness of the generator. The lower the temperature, the more conservative the model gets. When temperature is at 0.1, the generator usually produces only 1 repeated sentence. Higher temperatures allow the model to take more chances and increase diversity of results.

It took about 1 day to train the generator for each author.

2.3 Results

In the following, I present the generation results of training each model at various stages. All generations were made at temperature 1.

a. *Jane Austen*

After 1 iteration:

"Af her fenerace, she ditnessed, "Whate good was of preper well and fell to Every, that the writtle wastter that canne ness to disporuse; and if it would I here a sitters no only not five the eather for learn, upon my hoor of a consicourion of slefnering."
"I should give firetely his leet of a pirtance, she salmund doing, "This is same with after undorneess to surucg, might deselve, was not; and of Hasturach, was good was an anyone her."
"I for the wanted evur with so had any manders expect who de such the receinly surpuming-tolretcow the plite side mevening of from havoured. I cannct of a sotruly, my distand her ganlul, brother with the agant of the tumery of himself."

After 10 iterations:

"You would almost be honour on the ornighting in better for it, but therefore seems hardly more of daughter, past accomple, but that you thought the Ballers means! And by, I could be right, scelling her. I was for the pertroms! Nole and be near anybody how eeads up and yet, every danger, which loved for and own so perwoms, I shall never be miserous off vain to this not been assuring to it; she had not understand; and everything would soon be an idea of such a value. Mr. Elton would for such a great days of the two troubles which she saw I have not the new propicible of love of the same conevicul. Toons to say while she had had governess you again well have the gretter would be storm, with a notive of her driving him as letter, for were the impossibilityformer-own wimanlons, you do not was. Mr Elliot's sorrighing too a little greater common, never seemed, which had not seen them gentlered mimprace.
What he was difficultificainely wattachonable, in a much letter Mrs. Darcy?"

After 50 iteration:

"I find, and catch."
"That's this poverty of farther striken and ulletal. I should we it, with grown off a terious party time when so often quite the two language, come in I was in the same triumph. Myself has disposed of the young man of the time less pleasant arlus, and there most feeling. You have in this right friend on making it. This has a very day upon my life, the symptom of her own very Miss Crawford, I am satisfied in Miss Dashwood, but, if you have their poordness at hands and plain than it was kept down to her; for she is very tired, he seemed about the greatest collection of rising young weather."

b. *Charles Dickens*

After 1 iteration:

And him him treak). Her and like the lothforem. At his kist to tonverming face shaming-a little ground at the meay with ret each and head, to peeming begwand him,. "You have a paster cheeted, I nexteain to this more instartony shovenly though, my sughing of your being upon Mrs. Tarkel, no say before you. No dops no firne groing in coucton, undich, the is this it. Now thought, for Hoba-most offercoming some'! "That you know the sarlidge will beer?'"
Is the wayment, bocked for when he was beant was Mr, entered.
seed with her stosilyes, in sught, his baadting of the stmate he light them me forch the tixe somely dron his stay, nike reptation, his earning, and she would have returned for my wick of his own.
She had not searpressed pursuing nips broat.
You to great yis upcall me.'

After 10 iterations:

' Lot the forge."
"What Dolly. You are ever arm, three of quiet, Betterliveling right to on one early, that I have called not up to be distrusted in other; and money has been out of your reneditake, whether they part them a garden, (to always prison into light, happy, if guincian it is a limb as he has fully terrageffess of no dulf os more in susticion from the table, looking one piticent, and reason her next face. The awacher word has not that as now that I had vading it to be rose. Something lacker in a learuge. Set have found I couldn't make it, it take to my good Wemmick, dread. He had great distinctible to go schilly far would dare as well, a better for made consmation for a very day). Then he must be twill, I dreamed, run fiance her fitting bour standing became heavy arm of which, the papornity of who felt many.

After 50 iterations:

'My little thing it is as what you glanced stirn.'
'But I am not reposed as like that?'
'But he was stalled on, on some pace of weak, and it is. Didn't you speak as he would in men hot into the would like a happiness.
'My dear to call, to be away. I retain that we know, goes that I was in evoses, I shall make no good door, I'm a days until the gate without took the utmost pleasant one your being toine to our one and say else, rumpled.

c. *Mark Twain*

After 1 iteration:

"I all it, yardy of the reekin. Man up resinply perkon).
He not have the enedint doon, but Jelwloyed had hore Masulian plop, of here. Through it was have castened unrendle presisudad and hose in the starm."
"Mellars's mle's. I'd made nring, Laures busk in that bight and you with one let a can as that hussed "
"Wrile allow go to dinnant out proate me my dighty man in fleo what a means I we was the durtnerss, to it by oull as e, saad he well smilse she is armiatoon obentmence. Well? It was not. I was gone whole present for the winlem" but Jound juited kight. He to turn and in the stain while she men pabed his jnater'ble frule, up it and lens the just efficf very light to trry is one a jird him was nothing."
"Back to be away. I six dack and heart."

After 10 iterations:

"It is rose and "for but he changed ut Pordee," said in a-betters, and tied as lates and sleep to be have really perpored at the questions were true! Even the esticked great balk Mock his mitting, as crobs, with the coeder hot friendly the roin the bill and slat and turned and boand as he was pity just out through sevin of the worth by her ten excitement of life with their floor, fun and then then the norsell means a thing and dear. He said, we always some house.

After 50 iterations:

"Yes!"
The body appreached himself to the others, and says:
"What was such a bill made. I'm a full sun, and they has thinks a grower to wun, but I'm considerable. You have made the purple lights. You know how to need," said now, "Had I'll amount to breading."
"But she was going to ring with it. I lenters I quit for a wife. You believe this first every invisiviss agin, I was too that, I warn't only a thousand sinceral mighty days, you embot into the tramp. I am just for dollars in it. Whilst your Beal Clay da fatis on it, filleds of his first near seven on her command at her astriffing along of curse now, looking after that fut to be trampled for the breift, through that ceckne feftings for the feeling so one of them. His master never might get enough to assemble a month; there he has seen so buck.

2.4 Discussion

Looking at the various stages of training, we can see the progress that the model makes on learning the context of each character.

After 1 iteration, the generated texts appear only as random chains of characters. However, with closer examination, it is possible to see that the model has learned the general word lengths, that punctuation should always be followed by space. Some short words are already present, such as *he*, *she*, and *and*.

After 10 iterations, longer words such as 'disappointment' or 'excitement' have been learned. A large portion of the words are not English words, but many are valid words. The words that are invalid quite often have parts that resemble English words, such as 'tion' or 'ness' or 'cked'.

After 50 iterations, the generated texts are not perfect. Some sentences still have invalid words, but this number has decreased drastically compared to the same model at 10 iterations. Some of the complete, valid sentences are quite surprisingly grammatically correct. However, very few generated sentences actually convey any coherent meaning.

This shows that the model has learned tremendously well the local contexts that characters appear in. For most sentences, various individual portions make sense, but together they do not combine into meaningful sentences. Many of the short sentences make sense by themselves, but not in context of other sentences. If using bigger learning vectors and more data could expand the size of local contexts, perhaps some more meaning can appear.

Between the authors, Charles Dickens had the most amount of text, and Mark Twain the least (in terms of megabytes). This is apparent in the resulting generated texts: Austen and Dickens models generate fewer invalid words than the Twain model.

3. Combining predictor and generator models from part 1 and 2

The idea behind this experiment is to combine the models trained in the previous 2 parts. Given that I have trained models that can tell if writing is written by a particular author, and models that can generate texts that look like they were written by a particular author, I can feed the output of the latter into the former to see what happens.

3.1 Methods and Preparation

For this portion, I needed to download and install a spell checker for Python called PyEnchant in order to correct spellings for the generated texts. spaCy itself can assign part-of-speech tags for invalid words, but this process is close to random and often ends up tagging all invalid words as nouns. Therefore, I used a spell checker to reduce randomness.

I generated 50 pieces of text from each model for Austen, Dickens and Twain. Then, I processed each of these with a script (gen_to_data.py) that turns all 50 .txt pieces into a data frame. The steps are as follows:

- Load in the text
- Spell check and correct all words not found in PyEnchant's dictionary
- Find the first period (.) and start the text beginning at next character
- Find the next 10 sentences
- Turn them into part-of-speech tag pairs or filter function words
- Count appearances of all features
- Export all to .csv

3.2 Models and Implementation

After having trained the models for Austen, Dickens and Twain on the real authors' texts, I make them predict authorship for the generated texts. The point to keep in mind is that each of these models is predicting on binary classes—whether a text was or was not written by the author it trained on. More specifically, generated Austen text is fed into the Austen model as Austen text—the result should be 1 (yes, written by Austen). If the model misclassifies, it means that either the generated text is not similar to the real author, or the model's separating hyperplane could somehow capture the difference between real and fake Austen.

3.3 Results

Random Forests (combined features)	Jane Austen	Charles Dickens	Mark Twain
Real author	0.9	0.858	0.872
Computer	0.953	0.94	0.8

SVMs (combined features)	Jane Austen	Charles Dickens	Mark Twain
Real author	0.942	0.909	0.935
Computer	0.987	0.967	0.98

3.4 Discussion

The above results show that for the most part, both random forests and support vector machines recognize the generated texts as being written by its target author. In fact, they recognize the generated texts better than real texts, meaning that the authorship attribution model perceives the generated texts as more representative and more substantially in the style of the target author.

There is one outlier case, which is the random forest model for Mark Twain. It is difficult to understand why this happens, and particularly so with random forests, where I cannot quite think of them as finding separating hyperplanes between Twain and the rest of the authors. There are a lot of variables to consider when investigating the cause of this outlier: the accuracy of the Twain generator, especially when I did not have a lot of data for Twain's writing; the accuracy of the Twain predictor; the most important parameters for distinguishing Twain, possibly overlapping with the other authors; a combination of all of the above. In short, I do not have a satisfactory hypothesis or answer to explain this issue.

The support vector machines, however, do spectacularly. This result is most likely a consequence of how support vector machines work—they separate the feature space with a hyperplane, and perform classification based on where the writing falls in that space. The fact that the generated texts were more correctly classified than the real texts mean that the generated texts fall more deeply in the region of Austen, and wander less into the separating region of the hyperplane.

One thing to remember while looking at these features is the fact that the training data is simply grammatical representations: part-of-speech tags and functions words. From these results, we can make an additional evaluation about the generator models. That is, not only did they learn to generate words in the English language, they somehow learned to generate them in the grammatical structure of their target authors as well. I did not predict this result, especially when the generated texts appeared rather incoherent and meaningless.

4. Distinguishing authors and impostor generator models

The results of distinguishing generated texts in lieu of real writing prompted me to ask the question: Are the differences between the writing of an author and the generated text in the style of that author distinguishable? This portion of the project is dedicated to answering that question by training models on only the target author and texts generated in the style of that author.

4.1 Methods and Preparation

In order to carry out this task, I needed a large corpus of generated texts—enough to serve as training data. Therefore, I wrote a shell script to generate 2750 texts using the Austen generator model. For this portion of the project, I only tested with Austen's writing and models trained in her style.

4.2 Models and Implementation

The shell script used to generate texts in Austen's style seeded each text with the iterator integer *i*. Naturally, the seeds were from 1 to 2750.

I used only Austen's novels and the 2750 generated texts as training and test data.

4.3 Results

15 most distinguishing features from Random Forests

(order varies on every run)

ADV-ADJ	0.0310919442828
my	0.0308208675058
i	0.0305351621943
the	0.0260296151029
ADP-DET	0.0249008750905
DET-NOUN	0.0233411050948
VERB-ADV	0.0219411220231
her	0.021794348856
his	0.0202436863022
me	0.0189474908937
he	0.0169017789124
ADV-NOUN	0.0164204095632
NOUN-NOUN	0.0161944543935
NOUN-PUNCT	0.0149710961533
DET-VERB	0.0147159670721

Combined features	Random Forests	Support Vector Machines
Jane Austen	0.975	0.984

Note: Classification rate varies mildly for Random Forests.

4.4 Discussion

Both random forests and support vector machines perform well on distinguishing between real Austen writing and generated Austen. This is incredibly surprising in context of the results from part 3 of this project. However, it proves something about the styles in their representational space. From the perspective of support vector machines, although real Austen writing and generated Austen may fall on the same side of some hyperplane separating Austen and Dickens/Twain, they can still fall on different sides of another hyperplane, which is the one that results from training an SVM on only real Austen writing and generated Austen text.

I have been unable to perform much analysis on the most distinguishing features. Nevertheless, they are interesting information to observe.

5. Distinguishing all authors and the Austen generator

As a natural extension of part 4, I wanted to know if these learning models can distinguish not only Austen from Dickens and Twain but also the impostor Austen text generated by the recurrent network model—all in the same space.

5.1 Preparations and Model

All the training data in this portion of the project have already been collected and generated in the previous parts.

The models I used are trained similarly to part 1, with the exception that now there are 4 input authors: Austen, Dickens, Twain, and Impostor Austen (generated). From the models trained from this data, I can target any 4 of these 4 inputs in order to train a model on them, and observe whether or not that target is easily separable from the rest of the authors.

5.2 Results

Combined features	Jane Austen	Charles Dickens	Mark Twain	Computer Austen
Random Forests	0.895	0.825	0.896	0.918
Support Vector Machines	0.950	0.886	0.943	0.966

5.3 Discussion

Compared to results in part 1, the classification rates for all real authors decreased. In other words, introducing a new author, namely an Austen impostor in the case, made it more difficult for the model to classify other authors. This mostly likely means that it is now harder to find a hyperplane that fits all styles of writing precisely. That said, the results are still generally pretty good.

One thing to deduce from this is that these models' performances decrease as more diverse styles of writing are introduced in the training set. One possible deduction from this is that it is probably difficult, then, to train a binary classifier for one author so that it can, very completely, distinguish that author from all other possible authors. As we can see in part 3, the classifier could not tell that the generated text was not the real author at all—it classified most of them as written by its target author. More generally, this means that unless a style of writing is introduced in the training set, it is likely that the model will not be able to correctly classify it, especially if it is similar to the target author.

The next step would be to train models on all real writing and all generated texts. Another idea is to introduce authors that are similar to current authors, such as Charlotte or Emily Bronte, to see whether it is at all easy for the models to distinguish them from Jane Austen. However, I did not have enough time to execute those experiments.

V. CONCLUSION, QUESTIONS AND FUTURE WORK

1. Conclusions

Through the various exercises and experiments in machine learning to distinguish and generate texts in the style of particular authors, I have learned the following things.

The task of distinguishing styles of writing between authors is pretty well solved with relatively simple learning models (random forests and support vector machines). However, it would still be difficult to train a robust learning model to distinguish one style of writing and use that to test on a large corpus of written texts. This is because the model very much depends on the training data. If the training data comprises only a limited number of styles of writing, especially to represent what the target author is *not*, the model will not be able to extrapolate well. For example, if we want to train a model to distinguish the style of Homer—given that our documents of Homer writing for training data is authentically Homer—and then run it on all the unidentified texts we find on Greek history, it is not certain that it would be able to tell the difference between who is Homer and who isn't, especially if the writing is similar to Homer's. In the ideal world, what we want to do is to find a hyperplane that uniquely separates Homer from *all other possible writers*—be they

ancient or contemporary. This requires that our training data knows them in advance and represents this space sufficiently. This, of course, is impossible.

The method of generating texts in a particular style is particularly good at learning local patterns. These patterns include not only words, but also grammatical structures. However, because the learning is local, it is difficult to learn patterns of meaning. This would require the ability to see patterns on longer ranges—at the sentence and paragraph levels. We could potentially train the same recurrent network model not on character level, but word level. This would increase the range of context considered. However, the trade-off is in dimensionality. The vector for learning probabilities for words would need to be large enough to encompass the English dictionary, which would then require an enormous source of training data in order to learn sufficiently well.

All in all, machine learning, although incredibly powerful, has very significant limitations. These limitations lie mostly in the availability of training data. For my particular project, the three big questions that constantly come up are:

How do I represent the text numerically?
Do I have enough data?
How do I measure how well my models are doing?

The first question was dealt with differently for different tasks. To distinguish styles of writing, it was sufficient to filter only some grammatical structure data from the text. To generate writing, it was impossible to learn on a vector of all words. This would require both a large amount of memory to contain all the computation, a large amount of data that I did not have, and either a large amount of computing power or time. Therefore, it made more sense to simplify and learn at the character level.

The second question presents two issues: one in consideration of the first, and one in consideration of completeness.

The first issue comes from the fact that the more features we feed the model, the more data we need to learn sufficiently. This arises out of the fact that more features creates a higher dimensional space, and when data in that space is sparse, the hyperplanes that models learn to tease out from the data may not represent categories as well as desired, possibly leading to poor extrapolation to unseen data.

The second issue comes from the fact that if we do not have data that is representative of the patterns we want to learn, the model will not extrapolate well. This is separate from the sparseness of data problem. It is possible to have a lot of data points in a lower dimensional

space, and yet still make poor predictions, simply because the model was not provided the data for a category that existed outside of the data set. Unless a category is *known* in the training data, the model itself will never be able to come up with a classification that correctly corresponds to reality—a limitation of supervised learning.

The third question is the greatest challenge. For all of my models, it has been difficult to know whether they are doing well or poorly, and why they are doing well or poorly. It is difficult to assess whether the data set had too many features or whether the training data needed more authors. There are no direct observables—all observations must be made indirectly through the results of training each model. In order to get answers to these questions, the only means is to conduct additional experiments with different number of features or data sets. I did not experiment with different sets of features other than part-of-speech tags, function words, and their combination. I only experimented with different data sets.

On the linguistics side, I learned that semantics is difficult to represent numerically. There are literature and applications that can encode words into vectors of meaning, such as **word2vec**, but machine learning has yet to be able to utilize this representation due to the high dimensionality of the word vectors.

2. Unanswered questions

What could have been done with word2vec and spaCy word vectors?
There are a lot of readily available tools to encode meanings of words into vectors. It would be interesting to think about and experiment with working with these incredibly robust data sources.

Were the training data sets sufficiently large?
This is the eternal question of machine learning. Although I have used cross-validation in my models, but it would also have been a good idea to try early stopping or graphing accuracy rates in order to assess different parameters and see how they translate to different amounts of data. Some more background in statistics would preferable for this task, as well.

What did the writing data sets actually look like?
One thing I would have liked to do is perform principal component analysis on the data sets for Austen, Dickens, Twain and their generator models. This would facilitate the process of explaining what was happening in the data and provide more insights to the results I was getting for different portions of my project.

3. Future Work and Applications

Distinguishing good versus bad writing

One possible extension of the authorship attribution problem that I think would be extremely interesting to see is differentiating between good writing and bad writing. Of course, this would require a good source of data, such as a database of A's and C's for university essays, or accepted and rejected scripts at publishing houses—both of which are difficult to secure. Another question is, are there any tangible differences between good and bad writing? It is not possible to say for certain. That said, such a model, if it exists, could help students to figure out ways to write better by looking at the most important features, and editors to save much more time by running a predictor for classifying quality instead of spending time to read and evaluate writing.

Word-level networks

Generation of language in particular styles is a burgeoning field with many applications. Through this project, I've only seen one small aspect of this field using recurrent neural networks at the character level. With more computing power and efficient software packages, it would interesting to train generators at the word level, even if only on a limited set of vocabulary. Some literature has already mentioned and attempted sentence-level modeling, giving rise to sentence-level and paragraph-level coherence (Lin et al.).

Neural Style Effect for Language

For machine learning in the image domain, it is already possible to combine the content and style of two different images together to make an entirely new image. This technique is known as the neural style effect (Gatys et al.). Is it possible to do the same for language—to combine the content of an essay with the style of a particular author in order to produce an entirely different essay conveying the same message in the style of that author? The current answer is not yet. We do not yet know enough about semantics in order to carry out such a task. But it would certainly be fun to investigate.

VI. INSTRUCTIONS FOR CODE EXECUTION

1. How to run scripts to distinguish between styles of writing

Python, Python's **numpy**, **pandas** and **sklearn** packages are required for this section.

To generate .csv data files suitable for machine learning for a target author, organize all text files of writing by that author into one folder. Then use:

author_to_pos_csv.py	:	part-of-speech tags features
author_to_func_lem_csv.py	:	function word features

These files call other functions in novel_to_pos.py or novel_to_func_lem.py, which were written to perform other modular tasks, such as turning a text file into a data frame.

The following files train random forests and support vector machines on various data sets.

learning.py	:	training on Austen, Dickens, Twain
learning_auth_comp.py	:	training on Austen, computer Austen
learning_all_comp.py	:	training on all authors & computer Austen

To set target author, edit the file's initial variables. Set target author as 1, others as 0.
To see most important features from random forests, set see_rd_features as True.

2. How to run scripts to generate text in particular styles of writing

Torch and Lua are required for this section.
Before using Torch, type source ~/.profile in the command line.

I followed the instructions on Andrej Karpathy's blog and Github repository to train models to generate texts. Specific command line instructions, both to train the model and generate texts are found on Readme.md on Karpathy's Github.

Here is an example command to train a model on Austen:

```
th train.lua -data_dir data/austen -rnn_size 200 -num_layers 2 -dropout 0.5
```

Here is an example command to generate a sample text:

```
th sample.lua -length 3000 cv/twain_2/lm_lstm_epoch18.55_1.2688.t7
```

To generate multiple samples, I wrote the script sample_gen.sh.

3. How to run scripts to train models on generated texts

The most important part of this portion is to know how to clean data for generated texts. Since the generated texts contain many errors and do not come in complete sentences, they must be processed slightly differently using gen_to_data.py. The script should turn a folder of generated texts into a .csv file.

WORKS CITED

Stamatatos, E., 2009. *A Survey of Modern Authorship Attribution Models*.

Koppel, M. *et al*, 2009. *Computational Methods in Authorship Attribution*.

Diederich, J. *et al*, 2003. *Authorship Attribution with Support Vector Machines*.

Koppel, M. *et al*, 2005. *Measuring Differentiability: Unmasking Pseudonymous Authors*.

Stanczyk, U., Cyran K., 2007. *Machine Learning Approach to Authorship Attribution of Literary Texts.*

Mikolov, T, 2010. *Recurrent Neural Network based Language Model*.

Lin, R. *et al*, 2015. *Hierarchical Recurrent Neural Network for Document Modeling*.

Gatys, L., Ecker, A., Bethge, M., 2015. *A Neural Algorithm of Artistic Style*.