University of St Andrews

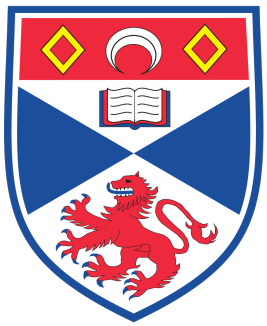
CS4099 Senior Honours Project

User Stylometry Association

Author:
Victor Igodifo

Supervisor:
Dr Juan Ye

April 12, 2019



Abstract

There's an inherent variation of writing styles from one person to the next. I aim to classify the fundamental style each person possesses, associating those with a similar style with one another. Employing a means to identify the factors which define the underlying notion of "style", I will ultimately be able to take any textual input and process it in such a manner where it can be identified as to belonging to a certain stylistic group; related to other pieces of text having similar innate characteristics.

I believe that classifying user's based on their own fundamental styles regarding their communication, whether written or audible (not covered by this report) is rife with potential. Along with understanding the features which dictate such styles, this could be the first step to a natural language generation system parameterised by users themselves.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is NN,NNN* words long, including project specification and plan.

Change the word count

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bonafide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Contents

1 Introduction

Bixby or Cortana. Let's have two people pick their poison; respectively they ask:

- "Shall I depart for my previously arranged appointment promptly?"
- "I have a meeting soon, should I leave soon?"

Now, disregarding the response from the virtual assistant, in fact, disregarding the assistant chosen entirely as well, these two questions are undoubtedly distinct. Yet, identical for all intents and purposes. Commercial products such as the aforementioned Bixby, and Cortana, as well as their competitors such as Alexa can recognise the true intent of a sentence incredibly well. Even the two previous questions will be seen asl, both asking, in essence "Am I late for my meeting?". Additionally, this project will ignore the underlying equivalence of the questions and focus on their other noticeable aspect. Their distinctness.

This project aims to differentiate the two questions. I'll look at the questions as if they were text; not the audio waveforms generated by human speech, as it's the content, rather than the rhythm and prosody, that I'm concerned with. They're stylistically different. Shorter words, yet more in the sentence. Less repeated words, and a greater variety of word length. But, do we need that if it just leads to more confusion? Either way, this project will aim to identify the aspects which dictate the entire style of each question, and of any sentence in general. This project will provide a means of associating users based on their writing style - their ***stylometry***.

User Stylometry Association (USA) takes the defining attributes from both questions and uses that as a way to cluster the users - or authors, if you will. Word length, vocabulary richness, frequency of words appearing n times, all of these can be used with a multitude of other features to identify the literary style of a user. Its these features that form a users stylometry. Standard classification tasks are commonplace in authorship assignment, USA instead follows the route of clustering, making it easier for parent systems using USA to serve groups of users in a unique manner, making applications feel much more personal.

1.1 Objectives

This project proved to evolve and take a direction which differed from that specified in the DOER; thus the objectives also changed accordingly, with some remaining relevant.

1.1.1 Primary Objectives

These objectives are the fundamentals for this project; they are the bare minimum that I aim to achieve. As with each objective to follow, they are expressed in relatively high-level terms - these objectives can very easily be split up into smaller, more manageable objectives.

- A Sequence-to-Sequence RNN (or the like) to allow for a model to distinguish between users

- A program capable of correlating users who possess a similar literary style together
  - Encompassed within such a program would be the means to process any provided input to appropriately associate an unseen user with known users
  - A basic program would be able to group these users based on solely their literary style - content will not be considered

1.1.2 Secondary Objectives

Whilst not critical to the overall intended functionality of the system to be developed, these objectives are considered to be meaningful additions for the system, provided that the fundamentals are adequately implemented.

- Allow for an unknown user to be presented with text which has been determined as stylistically similar to their own
- Expand the feature domain to consider more involved aspects of stylometry, particularly sentence structure and punctuation use
- Calculate the similarity between entities within a group and determine the principal features shared amongst the group's members
- Allow for the input of speech in the program
  - Stylometry will then be applied on the parsed text
  - The audio waveforms will not be used, would lead to the capture of other similarities such as dialects

1.1.3 Tertiary Objectives

Such objectives are highly unlikely to be completed within the allocated time as they will only be considered upon the completion of those before it. In essence, they can be deemed pretty ambitious.

- Transform any given text such that it takes on the style of any other text (or trained user)
  - This would be a system analogous to Luan et al. [10] - except with text.
  - The source sentence will be modified such that it takes on the style of a reference sentence, whilst preserving its meaning
- Provide a speech synthesiser to read aloud the stylistically similar text.

2 Context Survey

Classification tasks are commonplace throughout machine learning. Typically these tasks focus upon the identification of a single class and ensuring that created models are capable of predicting the correct class with respectable frequency. It's this aspect, the identification of a single user, where this project deviates. Instead, I aim to predict classes of users, rather than precise users, with these groups of users being related based on their literary style.

Stylometry is the statistical analysis of literary style, working on the knowledge that the same author maintains a consistent writing style throughout their work [7, 9]. Work regarding stylometric classification is prominently used for authorship assignment for anonymous or even disputed text [1, 9, 11]. Thus leading it to be an ideal method of uniquely identifying an author. However, the means as to which this identification is executed proves relevant for this project to identify a group of users sharing a similar literary style.

2.1 Author Identification

The use of stylometry to classify anonymous piece of text is a rather typical approach. This is seen in Narayanan et al [11]. The identification of the author for any given piece of text, even those which are supposedly anonymous, can have an everlasting on the author, especially if they are a whistle-blower on the internet - as mentioned in their paper. Using attributes surrounding written text, such as punctuation, part-of-speech tags, and capitalisation, Narayanan et al could successfully identify the author of a piece of text 20% of the time. Given a dataset containing 100,000 possible authors, such a feat is rather impressive.

Unlike their intention, this project does not set out to highlight the apparent lack of anonymity authors have on the internet. Instead, the challenge is to group authors on their stylistic similarities (however, identifying authors from unlabelled text would be a trivial modification). Nevertheless, the use of stylometry as a means of authorship assignment used in Narayanan proved to be an ideal method as to determine the stylistic similarities between users for this project.

2.2 Similarity Detection

Rightly being seen as a clustering task by Abbasi and Chen [1], similarity detection is a self-descriptive process, occurring between authors in a dataset. This contrasts classification as clustering focuses on grouping entities together based on some similar attributes. As discussed prior with respect to Narayanan et al. [11] in section 2.1, classification aims to assign an unlabelled input (anonymous piece of text) to one of many previously trained upon entities. Figure 1 depicts the difference between identification and similarity detection.

Similarity detection is seen by Abbasi and Chen as a way to correlate multiple pieces of anonymous text to a single author, based on some threshold value [1]. In general, I would personally disagree with this notion that text which may be stylistically similar can immediately be associated with one unique author - that is unless a sufficiently large feature

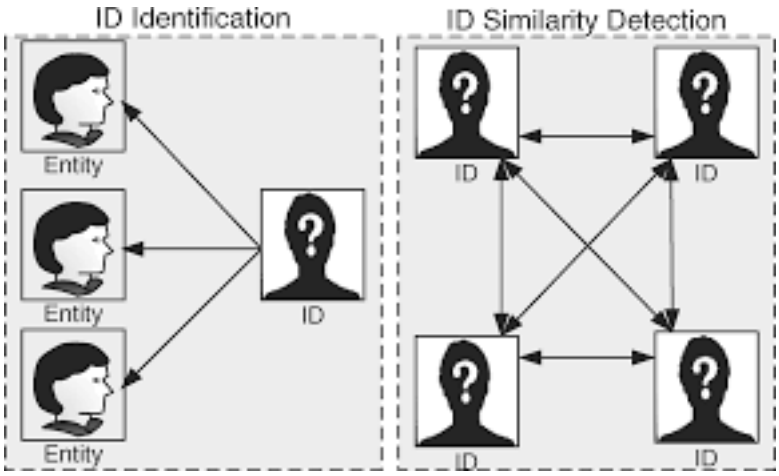


Figure 1: Identification vs Similarity Detection as illustrated by Abbasi and Chen [1]

set is being used (which they claim). This assumption does not apply to this project; instead, given text that is stylistically similar to another, I assume that there is no ulterior significance, such as the text belonging to the same author, behind their similarity. This is ultimately overruled however if my system undoubtedly believes the text must belong to the same author.

I do, however, sympathise with Abbasi and Chen's proclaimed importance for similarity detection within stylometric systems. I propose that similarity detection could either be accomplished between different authors, or different pieces of text.

I aim to cluster users who resemble one another into a group, known as an ***equivalence class***, where the style of one user is representative of the rest of the group. This is my view of similarity detection in the former case. It follows that the similarity detection to be employed here will be parameterised by an understanding, and correlation, between a given literary style and an author. Subsequent analysis pertaining to this detection would arise from a trained model's ambiguity as to identifying an author from an anonymous text.

The latter method of similarity detection works between anonymous text instead; this is the concept proposed by Abbasi and Chen. This true clustering task would group text together based entirely on their stylometric features. Dissimilar text would be, mathematically, further away, whilst similar text would be closer together, forming a cluster. It's these clusters which they deduce as authors - each text in the cluster was written by the same person. A notion which, whilst plausible in the correct circumstances, does not generalise to my situation as previously mentioned.

3 Requirements Specification

3.1 Preface

The requirements of the system to be implemented are derived from the project objectives as defined in section 1.1. These requirements come in two flavours, ***functional*** and ***non-functional***. Functional requirements focus on the behaviour of the system and the functionality it provides. Non-functional requirements are alternatively concerned with the operation of the system, such as in regards to reliability and security.

When discussing the development of the system with my supervisor, Erica, requirements for the next stage of development would be defined each week. I would then organise these requirements in *Trello* and use my board as a means of deciding upon which requirement to satisfy at any given time. This developmental approach is discussed in detail in section 4.

3.2 Functional Requirements

Process datasets in CSV files

A fundamental part of the system; models cannot be trained without a source dataset. This encompasses the reading of datasets into memory and the subsequent filtering of users to comply with later cross-validation.

Allow for feature extraction of a dataset

Given a dataset as a CSV file, features from a body of text can be extracted and assigned to the corresponding user (label). These features and labels would later be used for training/testing.

Reduce the feature dimensionality

Use a form of dimension reduction on the extracted feature set to convert all the features down to a smaller number of representative features. Section 6.3 covers this in more detail.

Utilise k-folds Cross Validation

Used for training the classifier models (SVM, Artificial Neural Network, and Random Forest). This succeeds appropriate filtering of the dataset. Allows for a model to be trained effectively, even if the provided dataset is small.

Ensure the best model is always used

From an evaluation of the trained models, the model which proved to have the greatest accuracy is to be used to form predictions on later supplied user input. This model will also be saved as a binary blob to be loaded into memory instead of being retrained (cf. section 3.3).

Allow equivalence class-based predictions

Rather than predicting against a single user, predict the equivalence class of users. Sections 6.4.2 and 7.3.2 discuss this in detail.

3.3 Non-functional Requirements

Models must be reusable

Being reusable would prevent the models needing to be retrained for each invocation of the system. Lacking in model re-usability would still lead to a complete system, however, every time the system is booted up a new model will need to be trained to make predictions. The amount of time wasted on retraining a model so often would be ludicrous.

Predictions should remain consistent

Predictions should not be easily/repeatedly mistaken as being pure guesses. Accomplishing such non-deterministic behaviour should be regarded as a failure; at that point, the system is equivalent to an elaborate random number generator. Evaluating this is simple - the same input should always lead to the same, if not reasonably similar, output.

The system must be environment independent

I laud the benefits that may be reaped from this system later on in this report (section 10.1). If I truly stand by the stated benefits this may provide, their realisation should not be hampered by an incompatibility to successfully run this system. Additionally, the results generated must be identical across operating systems and computer specifications.

4 Software Engineering Process

4.1 Methodology

Given the weekly meetings I had with my supervisor, the development methodology for this project quickly delved into a pseudo agile development process. Each week with Erica we would discuss the progress that I had made that week as well as the work to be completed for the upcoming week; ultimately, leading me to adopt a work-flow akin to sprints. Some weeks would involve the demonstration of results I had gotten from my experimental procedures, to be discussed in a later section. Other weeks had multiple meetings, these weeks would follow points of sub-standard, or unrelated work on my behalf in regards to what we had discussed prior. These weeks were incredibly useful as to ensuring that I maintained the favourable momentum I had at the beginning.

It would be safe to say that early on in the project I was rather ambitious as to what I could achieve in the allocated time. I had envisioned multiple components for this project which, if I had been more stubborn and yielded to my eagerness, would have left me with a flaky, barely-working mess. Following an agile-esque approach allowed me to instead focus on tasks that could be reasonably accomplished, whilst ensuring that the work carried out was dependent on previously completed tasks.

Ultimately, the process I ended up following came about as a consequence of my adaptation to my scheduled meetings with my supervisor. This by no means left me at a disadvantage compared to the alternative of a meticulously planned process. Actually, initially adapting to an agile process subsequently gave me the confidence that I could continue development in this manner. I would be able to change to different demands/requirements rapidly, which occurred later on in the development of the project (to be discussed in section 9), in contrast to being tied down to rigid system requirements as seen in a waterfall design methodology.

4.2 Tools

4.2.1 Trello

It's only natural for task management to become a problem if no process exists to regulate tasks to be carried out. Alleviating the problem of working on too many tasks, and keeping a record of completed tasks, was handled by Trello. At its heart, Trello can be interpreted as a glorified to-do list.

The list-making website/application was introduced to my development process in early December and radically improved how I worked on this project. Prior to it's inclusion, I had used an incredibly naive approach to task management. A rudimentary list in OneNote, and a mental note as to what should be accomplished next. This was far from ideal. The old adage "a picture is paints a thousand words" is suitable here to depict the improvement of Trello over a basic list. Have a look at Figure 2.

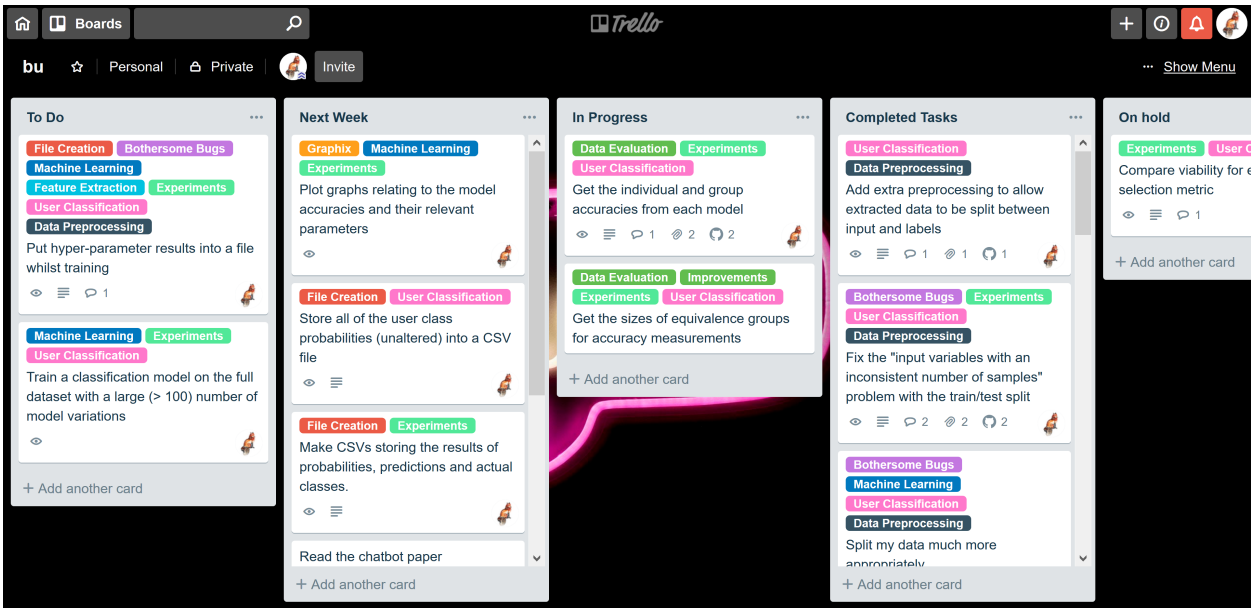


Figure 2: The Trello board created for task management

4.2.2 Git and GitHub

The distributed version control system Git allows for the management of changes to the written codebase. Git was used to ensure that I would always have a record of the changes that have been made to the codebase; vital in the event that a serious mistake had been made during development. It's branching mechanism also proves itself useful. When deciding to work on an experimental feature, I could create a new branch to ensure that I always have a stable, working copy of my code, despite any feature I may introduce.

GitHub is a hosting service which I used to host the codebase for this project. This served as a way to backup all of my code in the unlikely event that I would lose it locally. Using a private repository, GitHub proved to be an important tool as it allowed me to transfer an up-to-date codebase to any destination I desired over the internet. Often this was between my own laptop to a dedicated host machine kindly set up for me to use with this project, and avoiding a slow copying process using a command such as `scp`.

5 Ethics

While various datasets are used in this project, none possess any private or sensitive user information which required prior approval. Additionally, whilst this project is user-centric, the project itself does not pose any potential to cause distress towards users. The potential to learn from users should not also be considered a breach of ethics as this learning would be based entirely on the user's interaction with the project, as opposed to any additional sources which may be seen as invasive.

6 Design

6.1 Overarching Structure

Upon the system receiving user input, the first aspect encountered is an undercomplete Convolutional AutoEncoder. This AutoEncoder processes the input in a manner allowing for dimension reduction. Such reduced data is the foundation for subsequent classification tasks of the system.

Classification is a prerequisite to the transformation of text with respect to a specific user's vocabulary. This ensures that any form of transformation can be carried out with a reasonable degree of accuracy. Such a classification task in question may be executed by a variety of means:

- Support Vector Machines (SVMs)
- Random Forests
- Artificial Neural Networks (ANNs)

My approach will utilise SVMs for their relatively simplistic nature. This SVM will be connected to the aforementioned Convolutional AutoEncoder to ensure that it is trained in accordance to the reduced data and can thus perform the classification task on reduced input.

The notion of Equivalence Classes were defined such that users who shared stylometric aspects were, for all intents and purposes, seen as equivalent. As per standard classification tasks, the likelihood of a text belonging to a particular user was expressed as a probability, such that the sum of all these probabilities is 1.

6.2 Data Preprocessing

6.2.1 The Dataset

6.2.1.1 Description

Whilst the focus of Zhan's thesis focused intently on community-driven question-answer (QA) sites [14], I found her reasoning behind each dataset she considered to use particularly insightful. Whilst I never required to model the interaction between users, I found these QA datasets as incredibly useful due to the typical length of the posts. Hence, I also went to the StackExchange QA collective of websites to get the data I would use.

From a consistently updated data dump relating to each StackExchange QA community, I decided I would use those from the *worldbuilding* community - a place for those who are fascinated about the construction of imaginary worlds. I had personally visited this website on occasion prior to the commencement of this project, so I was aware of the detail some posts may possess. Imagine a world where everything I say is a lie - how do I make you believe anything I say? Now, imagine all the conditions leading to my inability to tell the truth, the restrictions in place against you as to prevent you from cheating the system, and

now imagine explaining that in a mere sentence or two [3]. It's this inherent effort and attention to detail by the worldbuilding community that made me realise it would be an ideal candidate for the training data.

The worldbuilding community was not the only community that I had utilised for this project. As described in section 6.3, I employ a ***Convolutional AutoEncoder*** to perform dimensionality reduction of the features to be extracted from the text of the posts in the used dataset. This autoencoder must be trained on similar data prior to it being able to perform dimensionality reduction. As a result, I also used the *serverfault* dataset, as mentioned in Zhan's thesis to train the autoencoder. The choice of this community came about of a lack of familiarity with any other community which was as detailed as that of worldbuilding, and the fact that Zhan had achieved satisfactory results with the serverfault dataset.

6.2.1.2 Data Retrieval

The StackExchange data was downloaded as XML files. These files contained entries as different nodes within the tree-like file, a contrast to the rows of data seen in CSVs. Given the use of libraries accustomed to working with CSV files - numpy and pandas in particular - converting these files to an equivalent CSV was a necessity.

The fact that pandas could process a dictionary (a set of key-value pairs) and interpret one as a DataFrame, walking through the XML files and parsing the nodes to create these key-value pairs appeared to be an appropriate method. Each node in the XML tree had multiple attributes corresponding to the fields and values of the dataset. With each node forming an entity in the dataset, every attribute could then be stored as a key-value pair in a dictionary. This dictionary would then become an equivalent representation of the entity. A DataFrame can be created from an array of these dictionaries and upon its creation, a CSV file is exported from the DataFrame and saved for future use wherever necessary.

6.2.2 Data Filtering

With the utilisation of k-folds cross-validation during the training of a classifier, the newly created CSV files must undergo filtering, such that only records which have a sufficient number of occurrences are preserved. In other words, as the datasets being worked with involve users posting comments and submitting posts, the dataset must be filtered to ensure that only users who appear k times throughout the corpus are kept.

Completing such a task was no more complicated than simply creating a frequency distribution in relation to each of the User IDs present within the dataset. Following its construction, achieved simply by iterating over the data and counting occurrences, each ID which does not meet the specified threshold frequency would be filtered out from the original dataset. Thus, leaving a pruned dataset suitable for training purposes, as each of the remaining entries pertain to an adequately active user of the correlating StackExchange site.

The data filtering process as a whole is a relatively simple process, especially in the context of the various other techniques and algorithms employed throughout the project. Its undemanding nature however, does not subject it to being a meaningless process; instead, the

Dataset	Initial Number of Posts	Initial Number of Users	Final Number of Posts	Final Number of Users
World Building	106,663	76,563	11,429	2230
serverfault	705,213	85,253	400,013	9777

Table 1: The size of the utilised datasets before and after they had been filtered down to only appropriate records.

act of filtering the dataset proved to be a crucial and repeated process during development. As opposed to constantly developing against the full dataset, I developed a means of caching the entire dataset, avoiding reading the entire CSV file line-by-line into memory each time, to allow for the quick generation of an arbitrarily-sized subset of the dataset for development purposes.

This proved incredibly beneficial when the core functionality of other project components needed to be tested. Especially as many components were not reliant on having an extensive proportion of the dataset or records present in order to effectively evaluate the success of their operation as a part of the system at large. Hence, running tests against these components could be carried out using a fraction of the dataset, even something as small as 0.5%, instead of the complete amount.

The data filtering process is undoubtedly an important portion of the over-arching Data Preprocessing task. Being variable in nature regarding the criteria which preserved records must satisfy, the number of permutations possible are enormous. Ultimately, I decided upon a threshold of 5 occurrences within the original dataset for a record to be preserved; this is in accordance to the number of folds I later use for my k-folds cross-validation algorithm to train classifiers, which is also 5. Table depicts the characteristics regarding the filtered dataset.

Once an appropriate subset of the original dataset has been acquired, the next step in preprocessing the data is the act of feature extraction. Allowing for the competent use of the data with a classification model.

6.2.3 Feature Extraction

The act of feature extraction is a means of being able to accurately portray the original data by "reducing" the overall semantics of the data; in a manner where it can instead be expressed as high-dimensional numerical vectors. These vectors are an inescapable piece in the training process of a classification model. They form the foundation of the model's ability to recognise similarities and discern stark variations in user stylometry.

Creating one of these high-dimensional numeric vectors which could satisfactorily represent its correlated record, will need to be done with features capable of predicting the target class as accurately as possible. Given the nature of this project, and the shear amount of target classes, expecting an accuracy >40% or so would be rather absurd; instead an accuracy circa 20% would be considered good.

Category	Description	Number of relevant features
Corpus Structure	Relating to the number of words and characters in a corpus, along with their corresponding averages within the corpus	4
Vocabulary Richness	Features dictating, as a whole, the abundance of variety within a corpus's vocabulary, or a lack thereof.	6
Stop Words	Can be considered a sub-category for Vocabulary Richness. Features identifying the frequency of some of the most used words in the English language.	293

Table 2: Feature sets for input data

Taking into account the effectively infinite number of possible subsets of each potential feature to be extracted from the text, evaluating the effectiveness of each subset would take too long to prove beneficial. Determining which feature set to use for extraction, as discovered by this evaluation of each permutation of every potentially extractable feature, could therefore be quickly dismissed as being a fruitless endeavour, which may not lead to a substantial performance gain. Alternatively, I decided to base my feature set around that used by Narayanan et al. [11]; ensuring that any feature I used could capture the literary style of the corpus. The result lead to the categories in Table 2 summarising the complete feature set.

The features constructing these feature categories were the sole input that the classifier model would receive as training input. The reliance on this feature set spawned from the confidence that they would be able to accurately represent the raw textual data the features are dependent on.

6.3 Convolutional Autoencoder

Autoencoders are a means of taking some input and creating a new abstract representation of the original input; in my case, a representation of a lower dimensionality [2, 4, 12]. This dimension reduction, thus makes the autoencoder ***undercomplete*** as the original features have been reduced to a compressed equivalent [4]. This is kind of autoencoder was used for this project.

Compressing the input data is achieved by one part of the autoencoder - the encoder - whilst the reconstruction of the compressed input is carried out by another part - the decoder.

Both the encoder and decoder are constructed with neural nets [6]. The encoder for an undercomplete autoencoder requires the number of outputs to be, ideally, much less than the number of inputs, while the inverse is true for the decoder. As a prerequisite, the number

of encoder inputs must equal the number of decoder outputs and the number of encoder outputs must equal the number of decoder inputs to allow for correspondence between the encoding and decoding processes.

Encoding the original data is a lossy process (some data is lost in compression), which has the potential benefit of eliminating noise when learning a compressed representation. Rather obviously, this could also lend itself to the possibility of losing/ignoring important connections between the original features.

Understanding the operation of autoencoders, the one I decided to be beneficial for this project would be a Convolutional AutoEncoder. These autoencoders utilise a Convolutional Neural Network (convnet) for both the encoding and decoding processes. These convnets utilise a number of weights forming a ***kernel*** to apply the mathematical convolution operation across the inputs, rather than associating weights between each input and hidden neuron as seen in a standard neural network [5]. This means the same weight can be applied to various inputs in the layer, known as a convolutional layer.

The outputs of this convolutional layer are down-sampled using Max-Pooling layers to compress the data within the encoder. Inside the decoder, inputs to these layers are up-sampled with Upsampling layers to decompress the data [2].

Given the acquisition of the numerical vectors representing the original textual input, I chose to pursue this route of further dimension reduction. Throughout my research, I was unable to find other papers utilising any subsequent form of dimension reduction following the feature extraction process. Instead I maintained my anticipation for the system to be able to further abstract away from the original input and form unseen connections between features, when the previously stated characteristics of autoencoders are considered.

6.4 Classification

6.4.1 Overview

As stated in Section 6.1, the classification task is achieved via the use of either a Support Vector Machine, Random Forest, or an Artificial Neural Network. To ensure ideal predictions, a hyper-parameter search is conducted on each architecture, aiming to find an ideal hyper-parameter distribution for each model. The hyper-parameter algorithm is described in detail later on in section 7.3.1.

The aim of classification is to identify a group to which a user most accurately relates to, given their textual input. Such a group contains other users sharing some relation to the user being classified such that the entire group identifies with a computationally identical (as determined by this system) writing style. Using the system components described in preceding sections, the input is converted from raw text, to high-dimensional numerical vectors, which are subsequently compressed. It's these vectors - this abstracted representation of the original input - which determine the predicted group.

Given the fact that this is a classification problem, each ***label*** - in this case, each user who has been used to train the model - is given a probability upon the execution of a prediction.

This label probability relates to the likelihood that the given input can be identified as that label. On its own, these probabilities are not particularly useful for identifying a given style of writing, instead being more suited to attempting to identify individual users - such work is covered primarily by Narayanan et al. [11], and by Abbasi and Chen [1] with a lesser weighting. This does not associate that user to any particular writing style whatsoever. Thus, to allow for an association between individual users and a given writing style, I introduced the concept of ***equivalence classes***.

6.4.2 Equivalence Classes

An equivalence class of users can be interpreted as a subset of the original labels which have some properties signifying a relationship between the elements of the subset [13]. This system defines equivalence classes based on the probabilities calculated for each label once the original data is predicted. This is as opposed to forming equivalence classes of users based on their features, akin to the relationship-signifying properties mentioned prior.

As a result of forming equivalence classes based on the predicted probabilities rather than the features themselves, creating multiple equivalence classes for each of the possible writing styles is a difficult, near impossible, task to accomplish with reasonable results. Instead, taking inspiration from Zhan [14], only two equivalence classes are formed - one for those users of a similar writing style, and one for those who are not. Any equivalence class which may be stylistically similar will not be considered and are disregarded into the latter equivalence class. The distinction between the two classes may be formed based on one of three algorithms:

- Jump Points
- Score Distributions
- n^{th} Percentile

each, rather naturally, forming different equivalence classes. Each of the algorithms have the commonality that they all work on a sorted list of predicted probabilities in descending order.

Each equivalence class generation method will be encountered later on in this report (in section 8.3) during experimentation of the system and its performance evaluation.

6.4.2.1 Jump Points

This algorithm aims to find the point within the sorted list where the difference between consecutive elements is the largest. The index of this point is used to define the two equivalence classes. Each probability and their corresponding label to the left of the index is maintained as the equivalence class the input belongs to. Whilst everything to the right of the index inclusive is not considered within the equivalence class.

6.4.2.2 Score Distributions

This algorithm calculates the cumulative sum of the sorted probability list, starting from the label with the greatest probability then descending, such that the sum is greater than or equal to a specified percent. In other words, given 90% (0.9) as a parameter, the equivalence class will be constructed from all of the labels whose cumulative probability sum totals, or exceeds with the last label, 90%. Remaining labels are thus not considered a part of the equivalence class.

6.4.2.3 $\mathbf{n^{th}}$ Percentile

Given a parameter, n (I'll use 95 as an example), this algorithm defines the equivalence class as all of the labels within the 95^{th} percentile. As seen in later experiments, this lends itself to larger equivalence classes as the size of the class is fixed; this algorithm is independent of the values of the predicted probabilities. Given 100 labels, using the 95^{th} percentile as an equivalence class, only the 5 highest probability labels form the equivalence class.

7 Implementation

Leading on from the Design of the system (section 6), we arrive at its implementation of previously discussed aspects. To accomplish this system, a wide variety of tools were required. Some of these tools tended towards the administrative side of the projcet rather than development-oriented (Trello and Git) and were previously mentioned in section 4.2. Whilst a multitude of development tools were also incorporated into this project. Along with a delve into some of the algorithms I employed, this section highlights the latter tools.

7.1 Python

This project is composed of a mixture of fields. Primarily computational linguistics, with a hint of machine learning sprinkled throughout. As a result, the choice of programming language was uncontested. With the sheer number of packages available from its package manager *pip*, especially the AI-focused packages, implementing this project in any other language would have been naive. With pip I could create a file, `requirements.txt`, to specify all the packages/libraries that I would need to implement this project. Subsequently running the command `pip3 install -r requirements.txt` downloads the packages locally, ready for development.

Ultimately, Python with the help of various packages makes implementing intricate problems a simple task. Reading a CSV and storing it into a robust data structure is accomplished in one line with *pandas*. Running a hyper-parameter search and training an Artificial Neural Network is elementary with *scikit-learn*. Tasks such as these could easily be several hundreds of lines long if I were to implement them myself in another language, whether that be *JavaScript*, *C++*, or *Java*.

Throughout development I had used *Python 3.6.8*, as it was compatible with Tensorflow, a library which I had anticipated to use. The latest version of pip was always used; at the time of writing, this is v19.0.3.

7.2 Docker

The containerisation platform *Docker* allows for consistency of program execution, independent of the host computer. Adding Docker to the project was a decision I made in late December (the 30^{th} to be precise) for the sole reason that I wanted to guarantee consistent execution between computers. Fortunately, Docker came with the added benefit of allowing me to make my code incredibly portable - a welcome bonus indeed. This particularly came in handy when running my code on a host computer. Despite developing locally on Solus and Windows, and the host machine running Ubuntu, a Docker container ensured I would not fall subject to the "it works on my machine" phenomenon.

Let's examine Docker's consistency to program execution. This is achieved via the use of ***containers***. Simply put, containers are lightweight Virtual Machines. Many nuanced

differences exist between the two, such as communication and data sharing between containers, however they are out-of-scope for the time being. These containers can "simulate" an operating system allowing for any command carried out within the container to act as if it were running on the specified operating system. This is system independent. Given a Fedora container, that container will operate as if it were a Fedora distribution regardless of the host machine's specifications or operating system. As containers run operating systems internally, this allows them access to nearly any operation available on the specified OS, including their native package managers for installing programs and command-line utilities.

When using Docker, a special file, known as a ***Dockerfile***, was created. This Dockerfile specified the configuration for another aspect of Docker, an ***image***. Docker images are analogous to a blueprint for a container, they describe the foundation (base image) the container is built on, the steps needed to set up the internal container environment and the default behaviour of the container. My Dockerfile can be seen in the root folder of the project (submitted alongside this report); it's configuration can be summarised as follows:

- Use the Python 3.6.8 base image
  - A base image is simply a foundation/environment to work on. The Python base image allows for the building of another image that depends on Python; providing access to the language and other useful features for Python development such as pip.
  - Apart from extremely basic images and OS images, those describing containers for OSs, each image is dependent on some non-empty base image.
- Create a folder for the project in the container.
- Download and set up all the packages needed for the project.
- Install extra commands to the container that may be useful.

Following the configuration and creation of an image, a container suitable for running the project would be built.

7.2.1 Virtual Environments

Before I had incorporated Docker into my development process and containerised the running of my code, I had utilised ***virtual environments***. Avoiding the entire overhead of simulating a completely different OS to the host machine, virtual environments work on top of the host. They are additionally less robust, instead only being used to maintain the packages/libraries that a project is dependent on. The `virtualenv` package available from pip was my chosen implementation method.

Speaking in a general sense, activating a virtual environment then allows for the installation and management of packages pertaining to the project associated with the environment. Installing any packages through pip installs them to a virtual environment folder; keeping them separate from packages installed outside the virtual environment, which instead apply

to any project on the machine.

Both virtual environments and Docker are, arguably, as portable as each other; provided the codebase is at the destination machine and both Docker and the virtualenv package are installed *globally* that is. Not satisfying this would likely lead to Docker being advantageous. With these prerequisites, an image to build a container can easily be pulled from Docker Hub (a hosting service for Docker images, akin to GitHub) to the intended configurations. Whilst a virtual environment can be *initialised* with a single command; additional set up may be required afterwards. This leads on to my next point.

Despite virtual environments being incredibly lightweight in comparison, I found myself favouring Docker mainly for its ease-of-use and its reliability. Given an image, unless the image itself is modified, the container will run as the original author intended, installing any extra necessities there may be.

Given a virtual environment, even though the initialisation is comparably simple, it's the user's responsibility to remember to install any and all program requirements, an automatic process for an image, as long as it had been configured by the original author. The hereditary nature of virtual environments is also an issue as the original development machine may be accustomed to one version of Python, say Python 3.7, whilst the other may only be on Python 2.7. Again, an image imposes a uniform version on both machines, ensuring they are capable of running the code and it runs as intended.

It's these aforementioned reasons that kept me with Docker, despite the possibility of it being overkill. I do however still maintain the virtual environment I used. It's only ever used for the plotting of graphs (seen in section 8) because Docker containers are *headless* - they don't have a GUI. Plots can still be created and saved within the Docker container, but I instead chose to use a virtual environment here as I display the plots graphically before choosing to save them.

7.3 Classifiers

As stated in section 6.1, the classifier used may be one of a SVM, an Artificial Neural Network, or a Random Forest. The *scikit-learn* library was used to implement each algorithm given its simplicity. While the *keras* and *Tensorflow* libraries are undoubtedly more flexible and "powerful" for creating neural networks, I opted to use those available in scikit for consistency and maximum code re-use.

7.3.1 Hyper-parameter selection and training

Hyper-parameters are parameters dictating the functionality of a model. Unlike parameters (weights), they aren't learnt by the model and have to be manually defined prior to training. These hyper-parameters are the focus of this section. Even though the algorithms used to create classifiers are all distinct, the process for each is identical, thanks to the continued use of scikit.

I employ a ***hyper-parameter search*** for each algorithm as to determine the combination of hyper-parameters which is the most beneficial for accurately predicting the correct

class. This search takes a parameter distribution to search over, training models based on each of the 3 algorithms over the parameter distribution. These hyper-parameters for the models being trained are uniquely assigned, in correspondence to the possible permutations specified by the parameter distribution. Either every permutation of the parameter distribution, or n permutations are used. The trained models are given a testing score, which is used to determine which model is the best; simultaneously identifying the best possible parameter distribution.

7.3.2 Predictions

Predictions are carried out on input vectors generated from the process described in section 6.2. Instead of using the standard `predict` method available to scikit models, the alternative `predict_proba` method is favoured to make these predictions. As its name loosely implies, this method returns the probability distribution of users, relating users to the likelihood that the input vector represents them.

Whilst the `predict` method is satisfactory for predicting a single user, I instead aim to predict, and create, the equivalence class, rather than individual users themselves. By predicting the equivalence class, not only am I able to achieve higher accuracy, I am able to correlate a user to a specific stylometry. Equivalence classes are determined by the probabilities resulting from the predictions (as seen in section 6.4.2) and are impossible to determine from the prediction of a single user, as returned by the `predict` method. Associating a user with a specific writing style would not be possible without the implementation of these equivalence classes. These classes group users based on whether they also associate with the determined stylometry of the input user.

The entire association of stylometry between users is achieved through ambiguity. Whilst a typically undesired feature, ambiguity is the central component regarding the grouping of users of similar stylometric attributes. Only when a noticeable difference occurs, then can a reasonable distinction be made in style. Forming the equivalence classes based on this difference, and subsequently predicting against the equivalence class thus allows for this stylometry classification.

8 Experimentation

8.1 Preface

Selecting the best possible hyper-parameter combination for a model is the difference between a network correctly identifying the style of a literary genius such as Roald Dahl, or mis-classifying the same text as belonging to a coffee-deprived CS student. I measure model accuracy with respect to generated equivalence classes. Hypothetically, I stumble across a model with 100% accuracy. Did I overfit the data? Am I testing against the training set? Or is every label a part of the same equivalence class? A solution to problems such as hyper-parameter selection, the deduction of the most efficient loss function for the convolutional autoencoder, and the performance of each equivalence class generation methods can be found empirically. That's the principal focus for this section.

8.2 Feature Subset Influence

8.2.1 Overview

A convolutional autoencoder is used as a part of the system structure, as described in section 6.3. This autoencoder transforms the input vector in a non-linear manner; as stated previously, this is to downsample the data and reduce its dimensionality.

Principal Component Analysis (PCA) is another means of dimension reduction except, unlike the use of an autoencoder, it is a linear reduction. A principal component is a linear combination on a vector containing j variables - think of this vector as a single record in a dataset. PCA finds principal components from the original vector, with these principal components being ordered of decreasing variance regarding predictions. In other words, they are organised by their importance, successive principal components account for less of the variation of predictions. The k most important principal components as the reduced feature set [8].

Explaining the variance of the data with PCA can be done by analysing the principal components as these linear combinations are directly-defined in terms of the dataset features. The dimensionality reduction from an autoencoder is more abstract - it is essentially a black box where the autoencoder learns the most salient features internally (Chapter 14 Section 1 of Goodfellow et al. [4]). This leaves me with little knowledge as to which features are most influential in the predictive process. Which leads me to this experiment.

8.2.2 Methodology

Proposed by Erica, this experiment trains the entire system on only a subset of the features. Using the same categories specified in Table 2 under Section 6.2.3 (Corpus Structure, Vocabulary Richness and Stop Words), models would be trained on each category, as well as the complete feature set, and be evaluated based on their correctness.

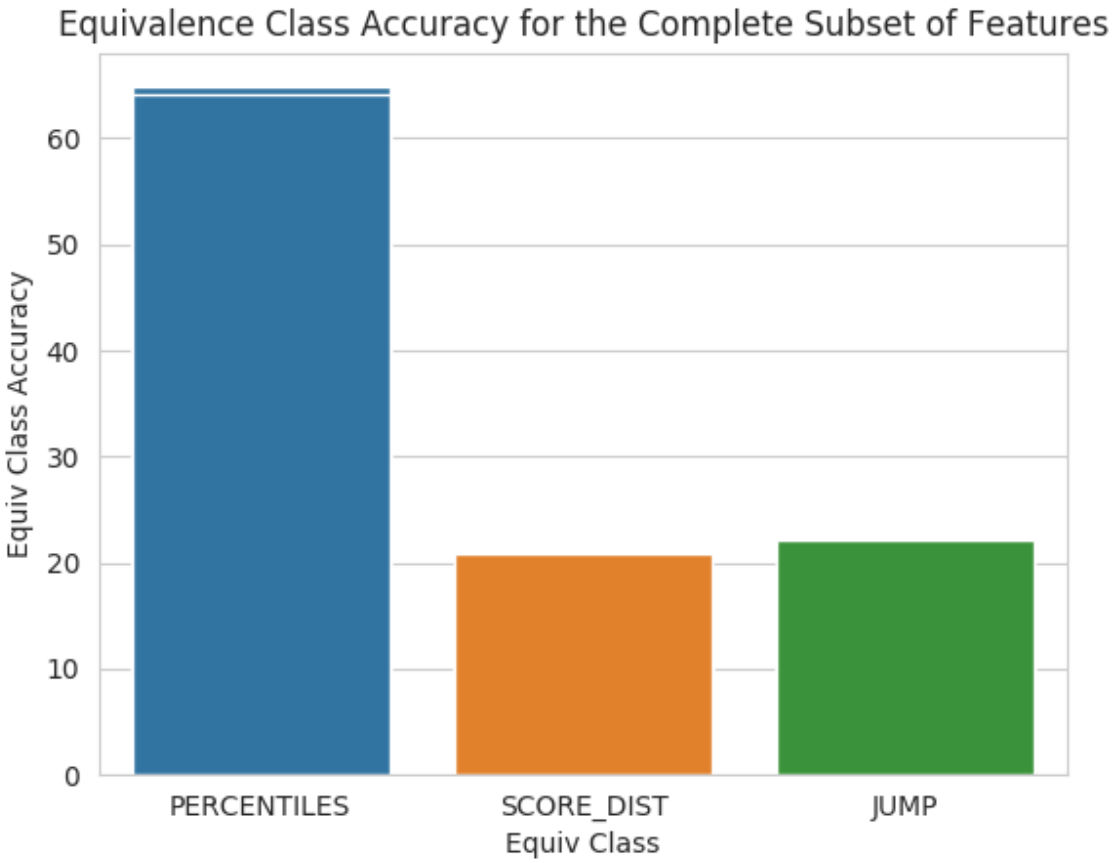


Figure 3: The accuracy of a model trained on the entire feature set.

8.2.3 Results

To be seen in subsequent experiments, graphs were plotted to visualise the results. We'll examine the accuracy (pertaining to correct predictions for each equivalence class methods) for a model trained on the complete set of features (Figure 3), and for a model trained on the *Vocabulary Richness* subset of features (Figure 4). These two figures were chosen as they depict the highest and lowest model accuracy in regards to the predicting equivalence classes. The n^{th} Percentile algorithm seems largely unaffected by the difference in features; the only discernible difference comes from the other algorithms.

Table 3 reports the accuracies obtained from this experiment. There are some interesting results reported, notably concerning the accuracies from the stop words feature set. Laramée [9], and Holmes and Kardos [7] both mention the influence stop words (synonymously called function words in both citations) has on a user's stylometry. Yet I did not expect the results that followed.

Using only a feature set composed of stop words, the highest user accuracy, that is the

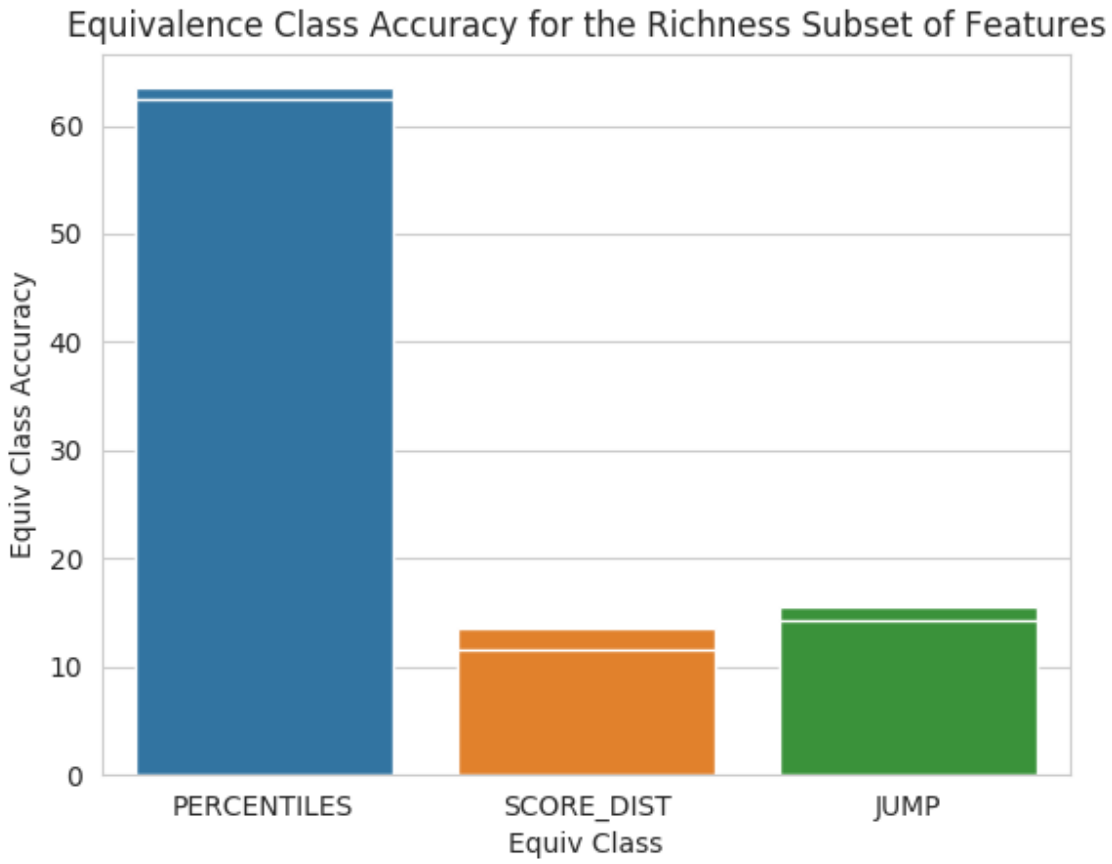


Figure 4: The accuracy of a model trained on only the features which model the vocabulary richness of a corpus.

Feature Set	User Accuracy	Jump Points Accuracy	Score Distribution Accuracy	n^{th} Percentile Accuracy
Averages (Corpus Structure)	10.87450%	15.52419%	13.55847%	63.48286%
Vocabulary Richness	9.71522%	14.25151%	11.59274%	62.42440%
Stop Words	18.58619%	20.72833%	20.09829%	56.56502%
Averages & Richness	11.92036%	17.02369%	14.31452%	64.83115%
Averages & Stop Words	16.63306%	19.99748%	19.35484%	61.51714%
Richness & Stop Words	16.21724%	20.01008%	18.38458%	63.30645%
Complete Feature Set	18.08216%	22.17742%	20.76613%	64.08770%

Table 3: Tabulation of the reported accuracies for each feature set permutation as discovered by the Feature Subset Influence experiment (cf. Section 8.2)

accuracy to predict an exact user; not the equivalence class they belong to, was recorded. The 18.58619% accuracy even surpasses the accuracy obtained from the complete feature set - the second highest accuracy, which only reached 18.08216%. With a substantial user accuracy, anyone would logically assume that the model trained only on stop words in a corpus would be the best model; achieving the highest equivalence class accuracies for each algorithm should be trivial. Reality instead takes a slightly different route. Placing 2^{nd} behind the complete feature set for both the Jump Points and Score Distributions algorithms would reflect the expected result. The model coming in last place for the n^{th} Percentile algorithm *slightly* deviates from what I anticipated.

I'm unable to provide an explanation as to why the stop words model is even beaten by the vocabulary richness model, which accomplished the abysmal user accuracy of 9.71522%. Initial assumptions would point to an inverse relation between user accuracy and n^{th} Percentile accuracy as the vocabulary richness, averages, and averages & richness models have the worst user accuracies, yet maintain very competitive n^{th} Percentile accuracies; placing 5^{th}, 3^{rd}, and 1^{st} respectively. Such a theory does not apply to the complete feature set as it maintained its, expected, dominance over every other algorithm, coming 2^{nd} in this regard.

8.2.4 Conclusion

I set off intending to determine which features were most influential as to the predictions made by the model regarding a user's equivalence class

8.3 Equivalence Classes

8.3.1 Overview

Section 6.4.2 describes the 3 different equivalence class generation algorithms employed to detect similarities between users. Each algorithm determines the cut-off point for each equivalence class in a unique manner: Jump Points based on the largest difference, Score Distribution based on the cumulative probability, and n^{th} Percentile utilising a fixed number - for this experiment it is set to use the 95^{th} percentile. Given that I evaluate the viability of a model with respect to its performance against generated equivalence classes, it's vital that these classes are appropriately constructed to avoid the possible overestimation of what could be a poor model.

8.3.2 Methodology

In essence, this experiment aims to compare the different equivalence class methods as to the size of the classes that they form and their corresponding accuracy. Whilst the principal focus here is to evaluate the difference between equivalence class algorithms, this experiment loosely also evaluates mildly different hyper-parameter distributions. These distributions are applied over a SVM model. This model was used to predict a testing set, different to the data it was trained on, and report on the results it had predicted. These results were generated for each instance within the test set and included information such as the equivalence class generation method used, the class size, and whether the class was successfully predicted.

8.3.3 Results

Naturally, I was curious as to which equivalence class proved to provide the greatest accuracy - so this was the first graph that I plotted. The results in Figure 5 shows the accuracy distribution for each equivalence class. Wider points in the graph indicate more observations for the matching accuracy, as seen on the y-axis. One can clearly observe that the n^{th} Percentiles algorithm maintains the largest average accuracy of the 3 implemented algorithms; it also achieves the greatest spread of the algorithms. An alternative visualisation of the same result can be seen in Figure 6. This figure additionally depicts which SVM kernel was used and its respective accuracy.

With hyper-parameter evaluation playing a supporting role in this experiment, I was opened up to opportunity to examine the correlation between a model's ability to predict an individual user, as well as its ability to predict an equivalence class. However, this lead to incredibly uninteresting results - showing no correlation between the two - as seen in Figure 7.

8.3.4 Conclusion

This experiment proved incredibly useful in determining which equivalence class algorithm proved reliable in predicting user classes. Utilising n^{th} Percentiles over Jump Points and Score Distributions is uncontested. Even when the substantial variance is taken into account, this algorithm still proves superior as the upper and lower bounds contributing to

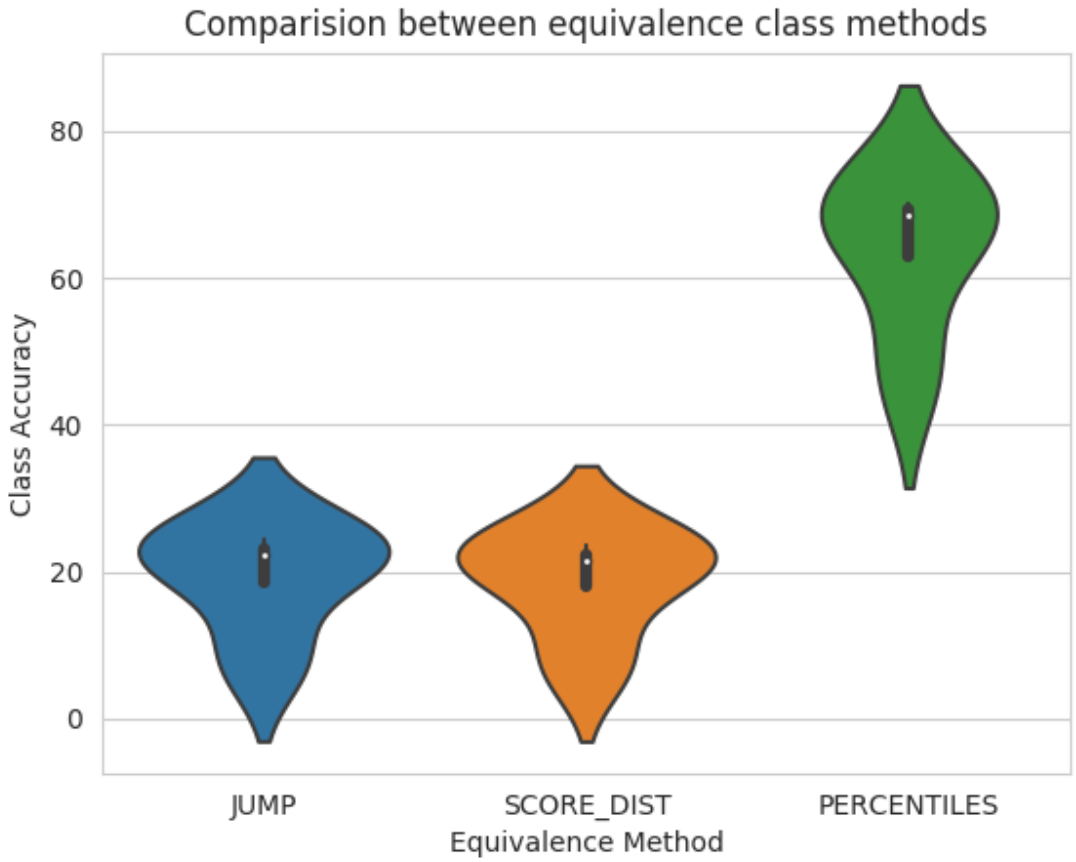


Figure 5: Equivalence class algorithm accuracy distributions

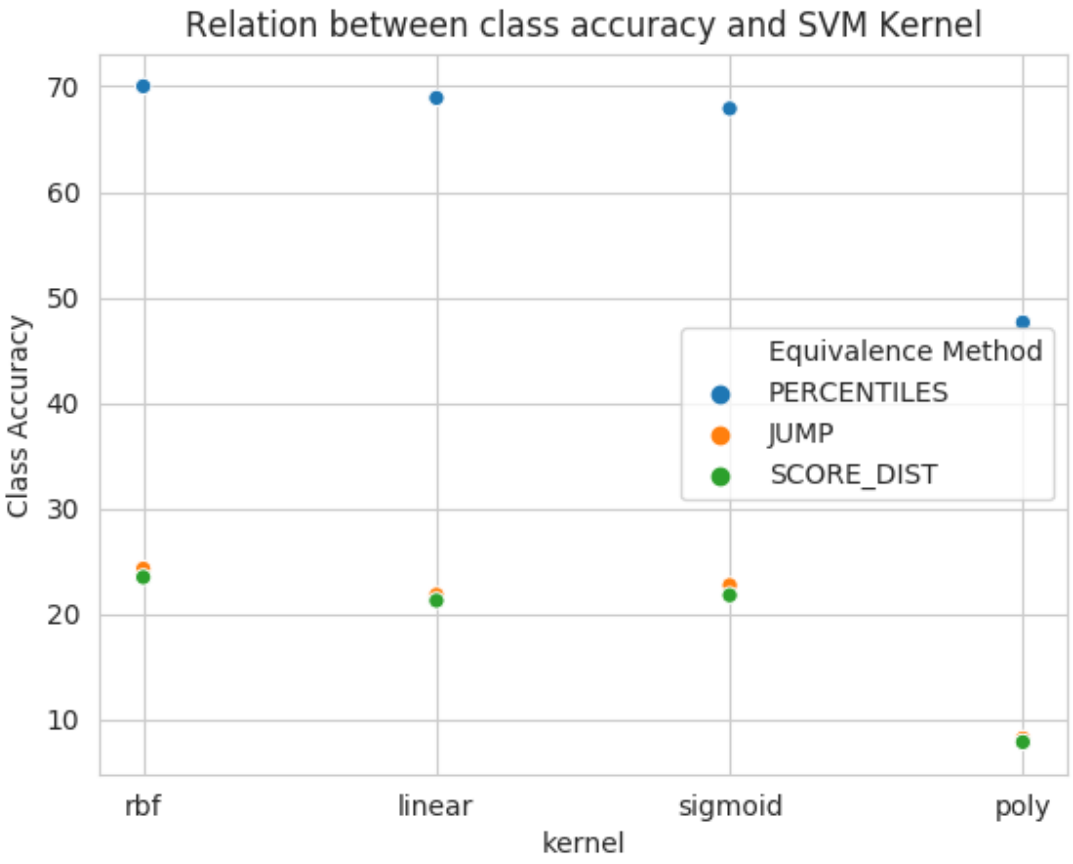


Figure 6: Alternative visualisation of equivalence class accuracy distributions; including information on the SVM kernel employed.

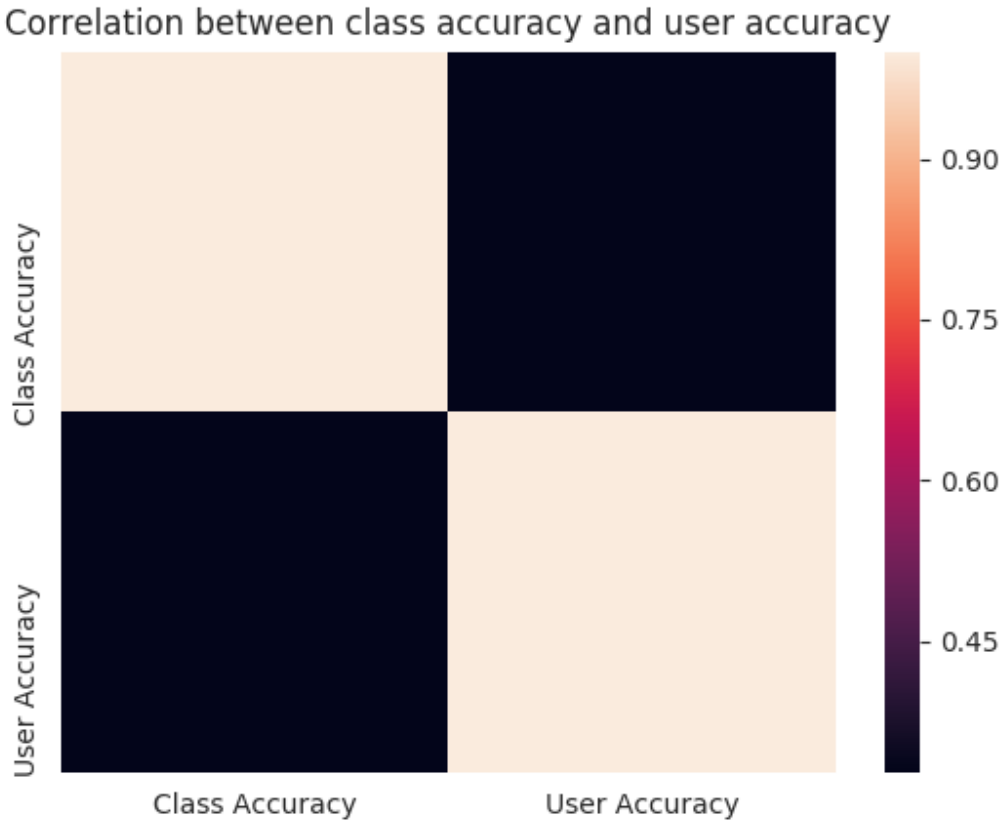


Figure 7: Visualising the lack of correlation between user accuracy and equivalence class accuracy. As a surprise to nobody whatsoever, each are strongly correlated to themselves.

this variance are yet shifted, to a lesser extent nonetheless, in the other algorithms under the same environment.

8.4 Prediction Variance

8.4.1 Overview

The last couple sentences of the previous experiment are rather misleading. In a world where the dominance of the n^{th} Percentile equivalence class algorithm is truly uncontested, there is no need for this experiment.

Naively, I accepted the supposed performance gain by using the n^{th} Percentile algorithm after being seduced by its rather attractive vanity metric. An accuracy unmatched by its competitors. After all, I had not thought to retrieve the precision and recall of the models at the moment the experiment was being carried out - why wouldn't I believe this algorithm was vastly superior?

Erica proposed that the observed difference in accuracy could be as a result of equivalence class sizes, a straightforward proposition in hindsight, yet it still eluded me. She later made the point where the use of equivalence classes would not prove beneficial for the clustering task I aim to achieve if the classes generated were composed of a substantial proportion of the possible labels. Understanding her point of view, I decided to examine the relationship between equivalence class size and their accuracy.

8.4.2 Methodology

The overall methodology for this experiment resembles that defined for the previous equivalence class experiment under section 8.3. Some slight changes are present to make understanding the relation of accuracy and equivalence class sizes the principal focus of the experiment. SVMs were chosen as the classifier - the model's hyper-parameters being set as those which performed the best during the training process. The model will be used to predict unseen testing data, with these predictions then being evaluated to generate competent results. To clarify, this model was trained on the entire feature set.

8.4.3 Results

As the equivalence class experiment shows in section 8.3, the n^{th} Percentile equivalence class algorithm leads to superior accuracy over its two competitors. The subsequent graphs depict this rivalry in an alternate view as I examine the relationship between equivalence class size and the accuracy. To generate the results, I determined accuracy as the proportion times that the encountered equivalence class size contained the user being predicted. Figure 8 depicts the relationship.

Prominently, there is only one green dot on the graph. The results utilised 7936 predictions for each equivalence class; with a constant value of n (as used), the n^{th} Percentile equivalence class algorithm will only recognise a single class size - hence the sole green dot. At the very top of the graph, there are two blue dots for the Jump Points algorithm - class sizes that attained 100% accuracy. This was observed for the classes of size 13 and 16,

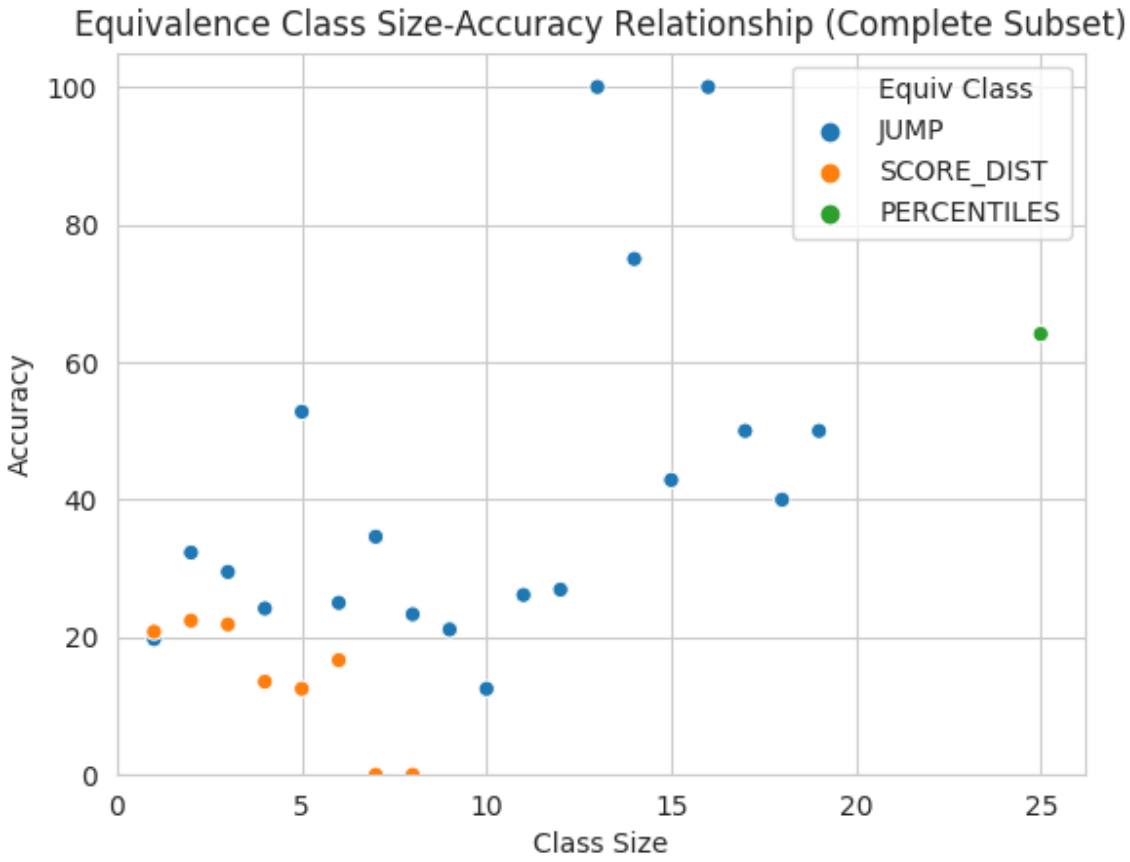


Figure 8: Equivalence class size plotted against their accuracy.

however it is important to note that these class sizes were observed 4 times and twice respectively. Contrasting the 100% accuracy of Jump Points are two orange dots from the Score Distribution equivalence class method. These occurred for class sizes of 7 and 8 elements, excusably, these classes were only observed 4 times and once respectively.

An upward trend in the size-accuracy relationship is observed from the computed results and is best depicted in Figure 9. Such a trend is wholly expected. The size of the equivalence classes is determined by the users who compose the class. First, the user who has the highest predicted probability joins the class, they're followed by the next highest, then the next, and so on. The algorithms only decide on a cutoff point as to when no more users can join the class. If this cutoff point is generous and allows for a larger class, the likelihood of true user of the predicted input joining the class increases in tandem. This likelihood increases cumulatively until the equivalence class size is the same as the total number of trained labels; at which point, the accuracy is 100%. This would be fantastic, but it doesn't help with similarity detection; even users who are diametrically opposed to one another are seen as equivalent.

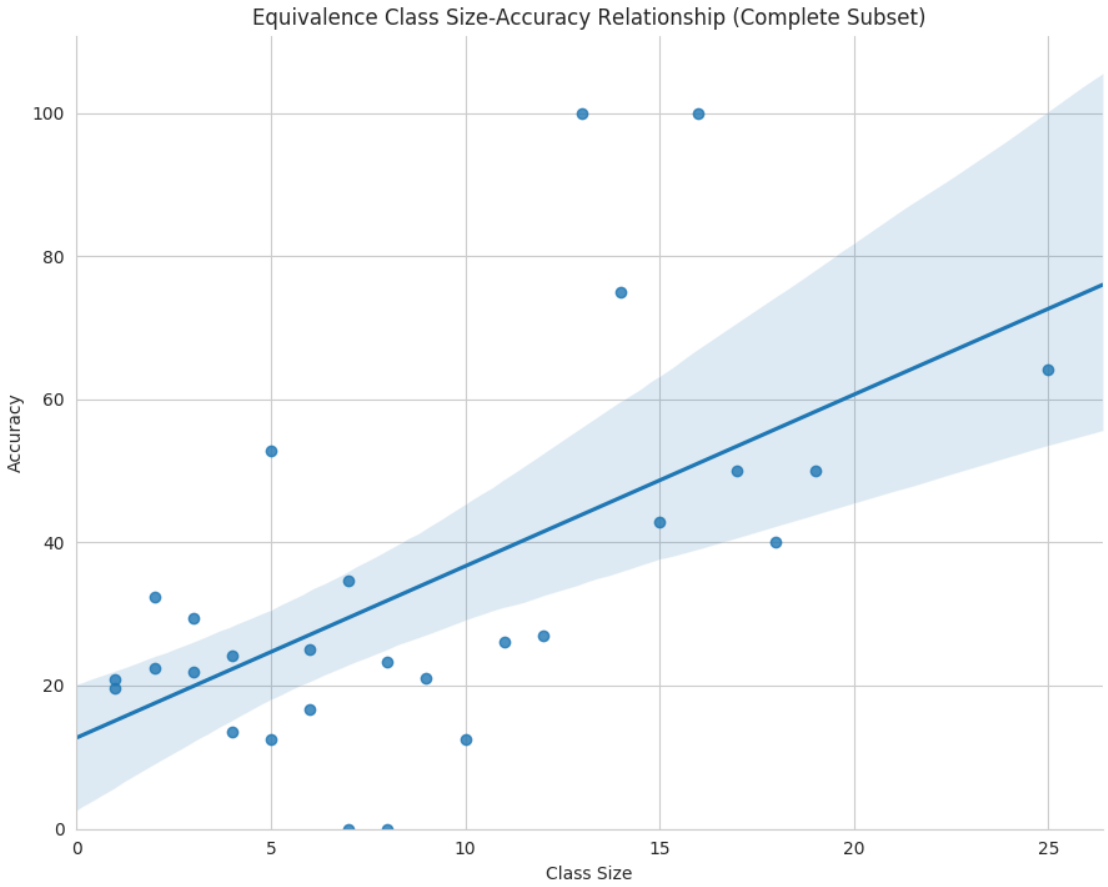


Figure 9: Depiction of the upward accuracy trend given equivalence class size.

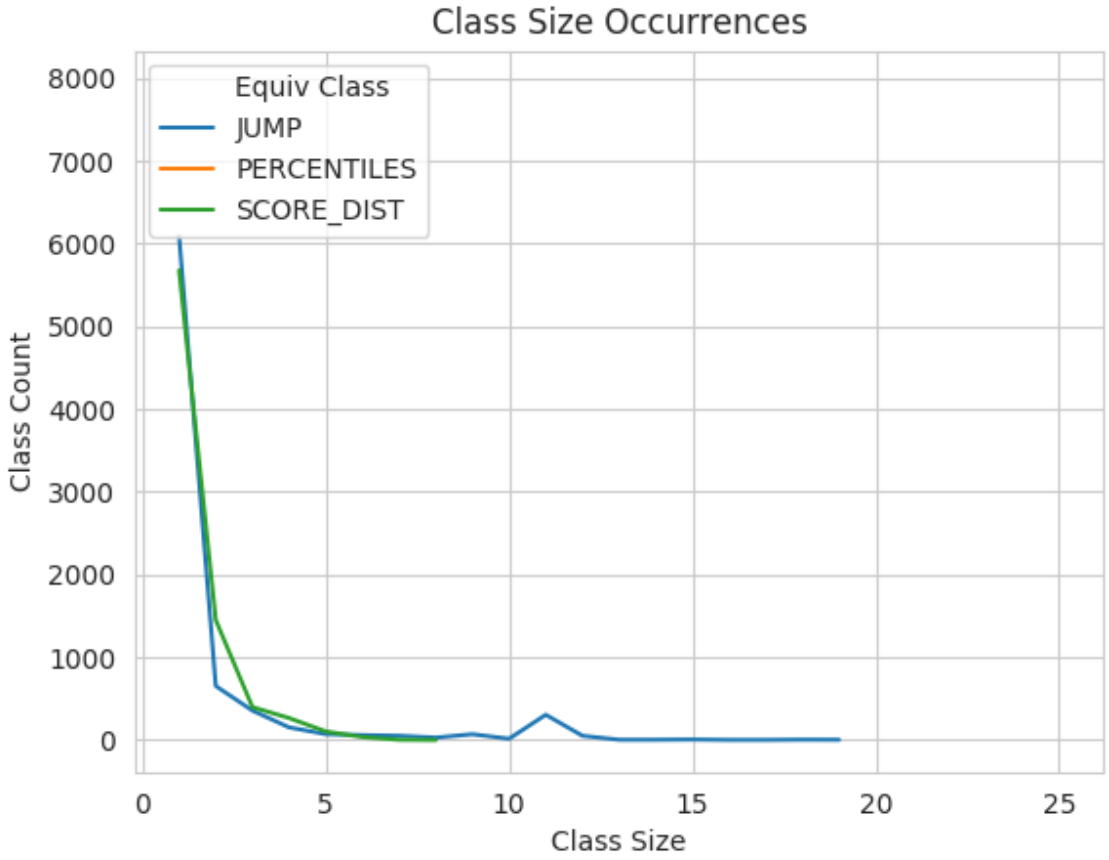


Figure 10: Sizes of equivalence classes for the generation algorithms

An interesting observation is made when the data is examined much more carefully; we find out that the Jump Points and Score Distribution algorithms tend to have incredibly small equivalence class sizes. As seen in Figure 10, The Score Distribution algorithm does not exceed a class size of 8, an incredible 89.806% of the observed results came from a class size of only 1 or 2 users. Jump Points showed a similar story, albeit less extreme. While 76.701% of the observed results came from a class of size 1, the observed class sizes were more numerous and went as high as 19. Unexpectedly, a class size of 11 users composed 3.856% of the observations, only behind classes of size 1, 2 (8.228%), and 3 (4.486%).

8.4.4 Conclusion

Without an intricate analysis into the cause of the substantial performance difference between the class generation algorithms, one could be excused for blindly assuming that the n^{th} Percentile algorithm is the superior generation algorithm. As evident in the results of this experiment, this supposed excellence is nothing more than an attribution to a larger equivalence class size in comparison. Ultimately, the use of the n^{th} Percentile algorithm can

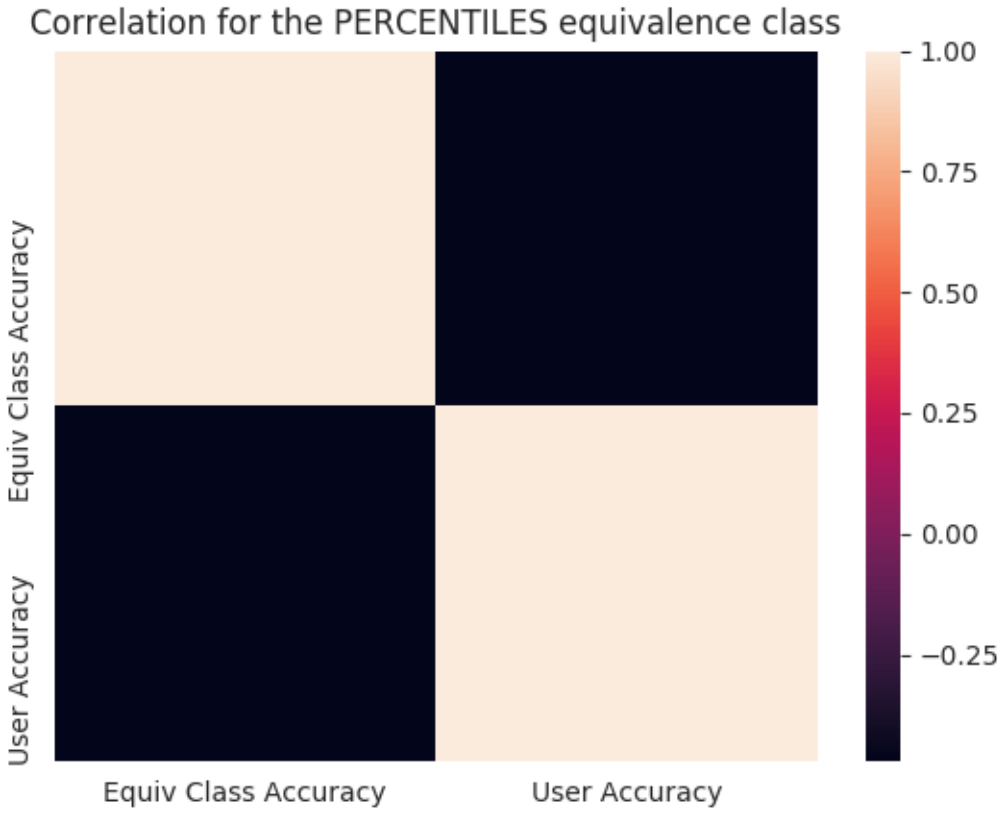


Figure 11: Correlation between the n^{th} Percentile equivalence class generation method and the user accuracy. As dictated by the legend on the right, there is a negative correlation between the two.

be deemed as detrimental, despite its high accuracy. The creation of an equivalence class of a constant size does not model the potential intimate connections between users in a class, as is possible from the Jump Points and Score Distribution algorithms. This is only compounded by the discovery of the lack of correlation between the n^{th} Percentile algorithm and a model's ability to predict a user (see Figure 11).

9 Evaluation and Critical Appraisal

9.1 User-Centric Natural Language Generation

Whilst being able to identify user stylometry, and classifying users based on a their written style was successfully achieved, the entire scope of this project was in fact much larger. Back in the Abstract, I mentioned how identifying user stylometry could be a prerequisite for, and in hindsight - a major component of, a natural language generation system which was parameterised by users themselves. This project was intended to be such a system; originally titled "User-Centric Natural Language Generation" - stylised as UC-NLG.

9.1.1 Initial Objectives

9.1.1.1 Primary

- A Sequence-to-Sequence RNN (or the like) to allow for the distinguishing users apart, for the purposes of appropriating system responses
- A program capable of providing responses to a restricted domain of queries, which are influenced by factors surrounding individual users
- A means of processing speech to use with said program, utilising NLP to identify user intents
- A means of outputting information to the user, such as through the use of a Speech Synthesis program/API

9.1.1.2 Secondary

- Manipulate a Speech Synthesiser to emulate a user's dialect
- Integration of a Virtual Assistant with UC-NLG
  - i.e. interact with an Alexa or Google Assistant API, for example, to create an app which can be called using the relevant Virtual Assistant, but uses a response generated through UC-NLG
- Expansion of the domain of queries for UC-NLG
- Allowing UC-NLG to be contextual
  - Ensures that users may ask follow-up queries, which are resolved in the context of previous queries

9.1.1.3 Objectives Summary

As can be interpreted from these objectives, the initial project was intended to be heavily audio focused. In fact I had not considered written stylometry whatsoever; it was only introduced in the early stages of the project when discussing the nature of the project with my supervisor Erica, and another lecturer Dr Mark-Jan Nederhof. Unanimously, it was agreed that the analysis of user speech to classify users based on attributes such as their pitch, prosody, and rhythm, would be considerably more difficult to accomplish in comparison to the analysis of written stylometry. This difficulty alone did not prevent me from pursing UC-NLG; instead I decided to shift the weighting much more to the written side. I also believed that if I could accomplish this aspect, I could parse user speech into sentences and use those, instead of the actual audio waveforms, to form a rudimentary speech-influenced system. Evidently, this was never realised.

9.2 User Stylometry Association

Even though the current project deviates from my initial ambition of UC-NLG, the classification which I did accomplish is a seemingly novel idea. Other completed work surrounding stylometry that I have encountered are either focused on the identification of single users, or predominantly discuss the history of stylometry [7].

9.2.1 Completed Objectives

Of the objectives specified in section 1.1, the following were successfully completed.

- A Sequence-to-Sequence RNN (or the like) to allow for a model to distinguish between users
- A program capable of correlating users who possess a similar literary style together

Both of the primary objectives specified were successfully implemented. As I previously mentioned, I had expected to satisfy these objectives at the very least.

The first objective was not completed as I had initially anticipated - using a Sequence-to-Sequence RNN. I had believed that it would be possible to extract features regarding the literary style of text using such a network. However, very early on into the project's development, it was realised that this would not produce a desired result; thus, I focused on performing feature extraction on the text in favour of creating an abstraction of the text generated by a RNN. Considering the lack of context used within this system, this objective would be better suited to the initial project definition, mentioned in section 9.1.

The second objective is, effectively, an overview of the entire implemented system.

9.2.2 Relevant Work

Both Narayanan et al. [11], and Abbasi and Chen [1] proved to have work akin to the system developed. Whilst both apply stylometric techniques to assign/correlate authorship to anonymous users on the internet, the techniques they employed proved beneficial to clustering stylistically similar users/text.

Both papers focused on identification, Abbasi and Chen, however, also highlighted the problem of similarity detection; stating it as a research gap in stylometry. Our applications of similarity detection differ, as I mentioned in section 2.2, leading this system to be a novel approach in the field by identifying similar authors as opposed to clustering related text.

The novelty of my approach unfortunately does not provide myself with a directly-related published baseline for comparison. Writeprints suffices rather well. Similarity detection is accomplished using Karhunen-Loeve transforms and Principal Component Analysis to determine the similarity between anonymous pieces of text - a significantly more advanced technique than the equivalence classes I utilise.

The method I used proved to be adequately effective in clustering users into an stylometry-determined equivalence class. Arguably, this may be seen as "pseudo-clustering" - clusters are not directly-dictated by the features possessed by users; they're formed based on model prediction ambiguity. Even though clustering is achieved, it is not executed in a manner which calculates the the similarity in the process; a nice feature, even though it is not fundamental.

10 Conclusions

10.1 Future Work

10.1.1 Preface

As a complete system, the association of stylometry between different users works well. Given the grouping of users into equivalence classes - ultimately deciding the similarity between them - work can be derived from these classes to parameterise systems based on classes/a class a user falls into. Such a utilisation can prove beneficial to "personalising" applications, particularly those which are text-heavy. Imagine a text-based adventure game where the persona of your main antagonist is the yin to your yang - how cool would that be? Well, the potential uses for User Stylometry Association extend further than a cliché good vs evil game.

10.1.2 Feature Set

The extracted feature set can satisfactorily group users into equivalence classes based on the stylometric attributes they may possess. However, this does not imply that the utilised feature set is complete/ideal. A wider feature set encompassing more stylometric attributes such as sentence structure, part-of-speech tag n-grams, and punctuation use, would prove beneficial. Narayanan et al. [11] discovered that the information gain per feature (calculated using Shannon entropy and random variables dependent on the corpus) is, for all intents and purposes, equal in utility. Therefore, the addition of more features to capture more information is elementary. The exact features being used are important, but the difference in influence each possesses over the final is rather insignificant.

Nevertheless, a large feature set does pose the potential risk of being able to predict individual users with a greater conviction. More confidence in predictions of a model shouldn't be considered a disadvantage; in practice, this is an obvious benefit. However, a greater confidence in a prediction would lead to less ambiguity - the principal component in determining the equivalence classes in the current implementation. This would undoubtedly lead to smaller equivalence classes, thus devaluing the current similarity detection employed by this system.

The potential benefits that a larger feature set provides cannot be ignored. Whilst smaller equivalence classes will be recorded, these classes will ensure that each entity within the class is tightly related to one another. Alternatively, to accommodate for a larger feature set, the means of similarity detection could take on a much more involved approach. This does not necessarily have to resemble the approach taken in Writeprints [1] - any approach which is feature-centric; even aggregating the use of currently employed equivalence class generation methods onto the features, could prove remarkably useful.

10.1.3 Natural Language Generation

Meticulously described in section 9.1, User Stylometry Association (USA) could be applied to a natural language generation system which generates text in a manner which

resembles some reference user. Despite the undeniable difference between each person, we are all too familiar with the monotony that is automated telephone menus, the repetitiveness surrounding virtual assistants and the broad generalisations in supposed targeted advertising.

Our attitudes and vocabulary vary depending on the person we're speaking to. We live in an age where chess grand-masters and 9 dan professional Go players are bested by computers - surely we're capable of uniquely tailoring some strings to users? Aspects which could be seen as intimidating to users such as virtual assistants or the future of strong AI/Artificial General Intelligence can be made approachable and relatable. After all, you do you find more relatable than yourself? Ignoring the potentially creepy aspects of a significantly advanced version of Alexa mimicking your entire linguistic style, eliminating the dreary programmatically-determined style of linguistic-heavy systems could greatly improve their user experience.

10.2 Closing Notes

Perhaps there exists an amateur author somewhere in the world. Yet to be discovered they feel that they are truly revolutionary - one of a kind - unmatched by any to come before them and anyone to follow. But how can they be so sure? Now, they can appropriately find out whether they can match the originality of J.R.R. Tolkien or if they are no better than a knock-off William Shakespeare.

Admittedly, amateur authors discovering whether they're just an imitation of another is not the primary audience of this project. No, the potential uses for this project are aplenty as described in the previous section. As its own entity, this project is an suitable means of clustering users of similar writing style with one another. Such clustering and similarity detection was noticeably absent in other works regarding stylometry as pointed out in Writeprints [1]; this project aids in filling that hole.

I managed to accomplish the fundamental objectives that were established in the very first section (1.1). There is no doubt in my mind that the incomplete secondary and tertiary could become a reality, whether I implement them as a part of further studies or they are adopted by a third-party who is, hopefully, somewhat inspired by the nature of this project. With the advent of the internet there is an unfathomable amount of written text at our fingertips. Who knows how the field of stylometry could evolve with all that data?

References

[1] Ahmed Abbasi and Hsiu-chin Chen. "Writeprints: A Stylometric Approach to Identity-level Identification and Similarity Detection in Cyberspace". In: *ACM Transactions on Information Systems* 26 (Jan. 2008), pp. 1–29. DOI: 10.1145/1344411.1344413.

[2] Francois Chollet. In: (May 2016). URL: https://blog.keras.io/building-autoencoders-in-keras.html.

[3] *Everything Bob says is false. How does he get people to trust him?* URL: https://worldbuilding.stackexchange.com/questions/142465/everything-bob-says-is-false-how-does-he-get-people-to-trust-him.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 14.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 9.

[6] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 0036-8075. DOI: 10.1126/science.1127647. eprint: https://science.sciencemag.org/content/313/5786/504.full.pdf. URL: https://science.sciencemag.org/content/313/5786/504.

[7] David Holmes and Judit Kardos. "Who Was the Author? An Introduction to Stylometry". In: *CHANCE* 16 (Mar. 2003). DOI: 10.1080/09332480.2003.10554842.

[8] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[9] François Dominic Laramée. *Introduction to stylometry with Python*. URL: https://programminghistorian.org/en/lessons/introduction-to-stylometry-with-python.

[10] Fujun Luan et al. "Deep Photo Style Transfer". In: *CoRR* abs/1703.07511 (2017). arXiv: 1703.07511. URL: http://arxiv.org/abs/1703.07511.

[11] A. Narayanan et al. "On the Feasibility of Internet-Scale Author Identification". In: *2012 IEEE Symposium on Security and Privacy*. May 2012, pp. 300–314. DOI: 10.1109/SP.2012.46.

[12] Andrew Ng. *CS294A Lecture Notes*. 2011. URL: https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf.

[13] Eric W. Weisstein. *Equivalence Class*. URL: http://mathworld.wolfram.com/EquivalenceClass.html.

[14] Sidi Zhan. *Protecting Online Privacy through Self-Disclosure and User Identifiability*. June 2018.