Technical Report CS15-06

Unstyle: A Tool for Circumventing Modern Techniques of Authorship Attribution

Alden Page

Submitted to the Faculty of
The Department of Computer Science

Project Director: Dr. Janyl Jumadinova
Second Reader: Dr. Gregory Kapfhammer

Allegheny College
2015

I hereby recognize and pledge to fulfill my responsibilities as defined in the Honor Code, and to maintain the integrity of both myself and the college community as a whole.

Alden Page

ALDEN PAGE. Unstyle: A Tool for Circumventing Modern Techniques of Authorship Attribution.
(Under the direction of Dr. Janyl Jumadinova.)

ABSTRACT

Linguists have developed incredibly effective methods of determining the authorship of anonymous documents with computer assistance. While this has helped answer a number of intriguing questions about disputed authorship, it can also be used as a threat to privacy and anonymity. As a result, there is a surge of interest in a field that researchers have deemed "adversarial stylometry," in which computers are used to assist in the obfuscation of writing style. As of the time of the writing of this document, few tools exist for effectively obfuscating writing style, and none are especially friendly for non-technical users. In order to rectify this, we have developed a tool called Unstyle, which applies modern techniques of authorship attribution to a document and assists the user in anonymizing the document. We have found that Unstyle has roughly identical performance to existing adversarial stylometry tools, is highly stable, and is considerably easier to use than existing tools in this domain.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to stylometry

Stylometry is the practice of extracting *linguistic features* of style from a body of written work called a *corpus.* Stylometry is of interest due to its applicability in the field of authorship attribution, in which researchers attempt to determine the original author of a document for which authorship is disputed [26]. While the term "stylometry" encompasses a wide variety of stylistic traits not necessarily found in writing, for purposes of this thesis, stylometry and authorship attribution will be used interchangeably.

In the context of stylometry, the term "linguistic corpus" tends to refer to a collection of documents by a number of different authors, and "linguistic features" refers to distinct reoccurring patterns in an author's corpora. The number of possible linguistic features in a document is quite large, but only a small number of them have been proven to be useful for authorship attribution in an empirical manner. The complexity of linguistic features ranges anywhere from simple word length metrics to more computationally expensive "deep" metrics, such as syntactic phrase structure. In either case, linguists and computer scientists have made extensive use of machine learning in order to successfully extract, classify, and rank the importance of features for purposes of attribution.

In response to stylometry, there is an emerging field of research focusing on evading modern methods of authorship attribution, deemed *adversarial stylometry* by Brennan et al. [18]. The concept of adversarial stylometry proposes the idea that an attacker aware of stylometric features could modify his or her writing style in order to partially or entirely invalidate the use of authorship attribution analysis through either *imitation*, *translation*, or *obfuscation*, with machine-assisted obfuscation serving as the current most promising method of circumventing stylometry.

1.2 Importance and potential impact

By combining network-level privacy tools like Tor[1] with an effective tool for obfuscating stylistic features, one could have at least a reasonable assurance of total anonymity, both from a behavioral and network perspective.

For most security applications, obfuscating textual style may strike some as borderline paranoid. Indeed, making use of adversarial stylometry is likely overkill for most security applications, and patently absurd in other domains. Whereas other existing privacy-enhancing technologies like encryption have their place in the daily lives of millions of internet users, adversarial stylometry should be reserved for situations in which being identified is a dire risk to personal safety or way of life. As large swathes of online activity are being actively collected and analyzed every day, however, the number of potential beneficiaries of adversarial stylometry tools increases every day.

In mid-2013, before going public with his identity, NSA whistleblower Edward Snowden requested that the Washington Post did not quote him at length out of fear of an adversary using stylometry to assume his identity [14]. Leaks of similar size by other whistleblowers have had serious consequences. Consider the case of fellow whistleblower Chelsea Manning[2], whose suffered long-term detention without trial and, eventually, a life-sentence [25] carried under questional judicial circumstances.

A proper adversarial stylometry tool could significantly lighten the burden of whistleblowers by providing some extra anonymity, with the end result of increasing governmental and corporate accountability.

1.3 Motivating analogy

There are other potential beneficiaries of adversarial stylometry than high-profile whistleblowers with, by some accusations, possibly questionable motivations.

Consider Bob, a theoretical citizen of a repressive regime. Bob has distributed anonymous essays that express dissatisfaction with the state establishment. In addition, Bob has been previously imprisoned for complaining about working conditions. Walter, a state actor whose job is to identify and intern political dissidents, has obtained a copy of the essay and sets about determining the authorship of the document. Walter performs stylometric analysis on the document and, using a database of corpora of previous political agitators, finds that approximately five former political offenders tend to use the particular configuration of function words and n-grams found in the anonymous document. By augmenting this information with the use of traditional investigation techniques such as interviews, interrogation, and intimidation, Walter correctly deduces that Bob wrote the incriminating document and turns

[1]Tor is a software suite designed to assist in hiding the user's location and activity while using the internet. A recent leak revealed that the NSA described Tor as "The king of high-secure, low-latency anonymity."

[2]Formerly Bradley Manning.

him over to the police. Perhaps Walter does not even bother with this step, and simply imprisons the five prime suspects. Had Bob made use of adversarial stylometry tools, he likely would have avoided apprehension by the authorities.

The previous analogy makes some assumptions that likely initially strike the reader as unrealistic. How could Walter have access to Bob's previous linguistic corpus? With the proliferation and public availability of social media, it is less difficult than one might think to assemble a corpus for a particular individual. Bob must have had a public blog completed unrelated to politics, such as a do-it-yourself or cooking themed blog. Since Bob has a record of political agitation, his online activity is archived by the state, and his blog is added to Walter's available corpora. Application of modern techniques of stylometry could potentially associate Bob with the anonymous document, particularly in light of authorship attribution becoming increasingly effective with large corpora and smaller bodies of text.

More mundane examples of the applicability of adversarial stylometry are not difficult to conjure up. Perhaps Alice has witnessed corruption in her company and wishes to write an anonymous editorial to the local newspaper, hoping that her boss Mallory does not use her work emails as a linguistic corpora to expose her identity, endangering her employment status and likely her livelihood.

1.4 Thesis

In order to address security concerns raised by stylometry, we have developed Unstyle. Unstyle is a fast, stable, and easy-to-use tool that applies modern techniques of authorship attribution to user-supplied documents. By using Unstyle, users are made aware of idiosyncracies in their writing that can lead to deanonymization. Furthermore, as users make changes to their documents, they are provided real-time feedback on the status of their anonymity. After Unstyle notifies users that a document has been anonymized, users can be confident that their identities are safe against stylometric attribution attacks that make use of the Basic-9 feature set. Due to the inclusion of a robust feature extraction framework, it is easy to extend Unstyle to make use of other feature extractors and feature sets, meaning that Unstyle has the capacity to be useful even after the development of more advanced methods of authorship attribution.

We support these claims by presenting a brief survey of modern techniques of authorship attribution, giving a thorough technical description of the process through which Unstyle ascertains authorship, evaluating the effectiveness of document classification techniques, discussing the design of the graphical front end, and documenting the process of constructing new feature sets.

Chapter 2

Related Work

The design of a user-friendly adversarial stylometry application requires at least a basic level of background knowledge in a wide variety of domains, including linguistics, stylometry, user experience evaluation, and the broader field of human-computer interaction. This survey of literature should provide a good starting point for those who wish to contribute to the emerging field of adversarial stylometry, or, more particularly, the adversarial stylometry tool itself.

2.1 Early work in stylometry

Pioneering work in stylometry focused on authors of historical significance, with a notable example being studies on the stylistics of the Federalist Papers [29]. 12 of the 85 articles of the Federalist Papers were of disputed authorship, though it was always suspected that either Alexander Hamilton or James Madison wrote them. By taking note of small differences in the choice of function words and making use of Bayesian inference, Mosteller and Wallace were able to say with a high degree of certainty that James Madison was the author of all 12 of the disputed essays. Later studies using more sophisticated machine learning models have overwhelmingly agreed with these findings.

There are countless quantifiable stylistic features that linguists have made use of in studies on stylometry. Only a small number of these features are known to be usable in authorship attribution. A tool hoping to defeat stylometry must therefore focus on these particular features. Stamatos provided a survey of effective methods for authorship attribution, providing context and focus for an adversarial stylometry tool [26].

Perhaps the most successful research in stylometry has focused on the occurrence of *character n-grams*[1] and *function words*[2]. Peng et al. have reported good results

[1]Character n-grams are, informally, groups of characters of n length. Note that n-grams include spaces, often times represented as underscores. For instance, the first 3-gram (or trigram) from this sentence is given by "_Fo".

[2]Function words are connecting words in sentences with little or no lexical information, such as the words "the", "a", and "and".

with n-gram based stylometry [16]. Additionally, Juola reports that one of the best performing algorithms in an authorship attribution competition was based on n-gram representation.

Function words are syntactic features[3] that appear to provide reliable methods of recognizing authorship. Zhao and Zobel showed that syntactic information such as function words are one of the best stylistic features available for distinguishing between authors [30].

2.2 Current state of research in authorship attribution

Studies involving stylometry typically involve a sample size of 10 to 300 authors. In light of constantly improving computational capacity and the public availability of large corpora on the internet, a few studies have attempted to use stylometry on corpora with thousands of potential authors. Narayayan et al. recently showed that internet-scale authorship attribution is feasible to a surprising degree [19]. This study made use of proven stylometric methods on a sample size of 100,000 potential authors. The classifier succeeded in identifying authors approximately 20% of the time. Prolific writers with more than 40 posts on their blog could be identified as much as 35% of the time. While this amount of accuracy is too low to identify authors singlehandly, the classifier can be used in combination with manual analysis of documents to great effect in identifying authors, since the pool of possible authors has been shrunk from 100,000 to 100-200 by a ranking algorithm. Furthermore, a few adjustments to the classifier could improve accuracy to up to 80% by adjusting the sample size slightly. While 20 to 35% accuracy is not a dependable number on its own, large-scale stylometric analysis could be used in combination with other security leaks in order to unmask an anonymous person's identity. This shows that it is quite possible to apply traditional means of stylometry on enormous sample sizes, and further illustrates the importance of proper tools for adversarial stylometry in the preservation of anonymity.

Further challenging assumptions about the requisite number of authors and available works to perform authorship attribution is a study showing that stylometric analysis can be effectively used on Twitter[4] posts [7]. Bhargava et al. showed that it is possible to attain 95% accuracy in authorship attribution with approximately 200 posts in a pool of 10 to 30 users. Given that many active Twitter users have thousands of posts, it appears that stylometry can be performed quite effectively on social media platforms, even with exceptionally short and unrelated samples.

Generally speaking, the most effective stylometry systems make use of supervised

[3]Syntactic information is valued because it is believed that authors may use syntactic patterns subconsciously.

[4]Twitter is a social media platform in which users submit posts ("tweets") consisting of no more than 140 characters.

machine learning models. Although any supervised learning model may be used, Support Vector Machines using Sequential Minimum Optimization (SVM SMO) are the most popular choice due to their speed, accuracy, and resistance to both the curse of dimensionality and overfitting, given that the user parametrizes the machine properly [10]. A typical authorship attribution study using supervised learning will involve the extraction of features from the documents of several suspected authors. The classifier is then trained on these features. Then, features are extracted from the document of unknown authorship. The classifier can then predict the label of the document of unknown authorship given its features. The accuracy of such a classification varies wildly depending on the chosen classifiers, the number of potential authors, and the chosen feature set. Greenstadt et al. achieved 92% classification accuracy in a 13-author setting using an SVM SMO classifier and a variation of the Writeprints. The Writeprints feature set was limited to the top 50 features [18].

A recent study augmented an SVM SMO classifier with an additional verification step. The verifier compares the centroids of set of document's features with the suspect document's features using cosine similarity. This verification step is an important contribution to the field of stylometry in a number of ways. Most forms of stylometry use a "closed world" model, in which it is assumed that the true author is in the set of suspect documents. In real applications, particularly those that are forensic in nature, this is not an acceptable assumption to make, since one can rarely be sure that the suspect author set actually contains the true author. If the true author is not in the suspect set, a misclassification is guaranteed to occur. Adding a verification step makes this an "open world" problem: if the verifier outputs a cosine distance below a certain threshold t derived from the training set, then the suspect author belongs in the suspect set. If cosine distance is above the threshold, the classifier abstains. In addition to significantly increasing the accuracy of document classification, the Classify-Verify algorithm has been shown to be significantly stronger against adversarial corpora[5] We make use of this measure in the implementation of Unstyle [27].

2.3 Current state of adversarial stylometry

Compared to the increasingly sophisticated field of authorship attribution, adversarial stylometry is in its infancy. Few studies have been conducted in this field, with the most notable work having been performed by Greenstadt et al in the development of Anonymouth.

It has been shown that simply imitating another author's writing style significantly impairs the use of some of the more superficial features used in authorship attribution, such as average word length or vocabulary richness [8]. Imitation also works well on more sophisticated forms of stylometric analysis, although to a lesser degree. The

[5] "Adversarial corpora" are documents for which the author has taken measures to obfuscate his or her writing style.

problem with the imitation approach is that it is difficult to apply to writing samples of extended length without ever allowing one's own writing style to "leak" through, possibly resulting in the inadvertent identification of the original author.

One previously proposed method of circumventing authorship attribution was through the application of machine translation from one language to another, and then back to the parent language. Research has shown this supposition to be entirely incorrect. Machine translation from one language to another back to the parent language is not an effective method of hiding writing style [8]. It comes at great cost to the readability of the writing and still fails to hide linguistic features to any reasonable degree. It seems that linguistic features persist through any number of rounds of translation.

The only adversarial stylometry tool that is available to the public is an application called Anonymouth, a tool developed at Drexel University designed primarily with linguistic researchers in mind [18]. Previous research also suggests that the feature set that Anonymouth focuses on may be flawed [11]. While Anonymouth and its developers deserve praise for their pioneering work in adversarial stylometry, Anonymouth is primarily a proof of concept designed for research use rather than a practical tool for obfuscating users' identities. There are a number of self-admitted usability issues, from the instability of the application to expecting users to hunt down and input work by other authors into the application. The project also suffers from a lack of documentation. As a pioneering tool in an immature field, Anonymouth's obfuscation algorithms (described as naive by the designers of Anonymouth) could be more effective in their suggestions and perhaps could be more sensitive to the context of the paper (for instance, Anonymouth should not suggest a user changes tense to third-person in a formal paper.)

2.4 Application design

Unstyle seeks to be useful to even non-technical users with little or no knowledge about sylometry. As such, it is important to take advantage of resources describing high quality design and usability.

Constantine and Lockwood describe a number of principles of user interface design that developers should make use of [13], including:

The structure principle.
: Designers should organize the user interface such that related elements are kept together and unrelelated elements are separated.

The simplicity principle.
: The design should be as simple as possible. Common tasks should be easy, and longer procedures should have shortcuts whenever possible.

The visibility principle.
: Options and materials for a given task should be kept easily visible and free of redundant information. Do not overwhelm the user with unneeded information.

The feedback principle.
Users need to be kept informed of actions, changes of state, and errors that are of relevant interest to the user in unambiguous language.

The tolerance principle.
Designs should be flexible and tolerant of failure, reducing the cost of mistakes and also preventing errors. For example, in the context of Anonymouth, accidentally inputting an incorrect document should be easy to revert in minimal time.

The reuse principle.
Designs must reuse internal and external components and behaviors for the sake of maintaining consistency and purpose, thereby reducing the need for users to waste time deciphering an unfamiliar interface.

Commenting on this list of user interface principles, Scott Ambler notes that, no matter how sophisticated a piece of software is, a sufficiently poor interface will deter anyone from using it [4]. He also lists a considerable number of useful tips and techniques in the field of interface design.

2.5 Paraphrasing and rewriting sentences

Existing methods for paraphrasing and rewriting sentences may be useful for facilitating the goal of automatic obfuscation of documents. The process of paraphrasing contains many similarities with the process of obfuscating (reordering words, substituting certain words with synonyms, etc.) and may prove to be quite effective, although more research must be done before this can be confirmed.

One particularly impressive paraphrasing system is described by Barzilay and Lee, in which multiple-sequence alignment and word lattice pairs were used to facilitate the restructuring of sentences [6]. Two human judges evaluating the output determined that the output paraphrased sentence was of acceptable quality between 78% and 81.4% of the time, as compared to less than 65% with a naive baseline algorithm. Sadly, despite how promising this research is, no proof-of-concept code is available, and the system is imposingly complex. As such, integrating the automatic paraphrasing system into Unstyle is not feasible at this time.

Chapter 3

Method of Approach

3.1 Statement of problem

We use a similar problem statement as Greenstadt et al. [18], although we automatically select documents by other authors, use a stricter definition of anonymity, and introduce a verification step.

A user U wishes to anonymize adocument D. The user selects a set of self-authored documents D_{pre}. The goal is to create a new document D' from D by changing the feature values F so that D' appears to be written by a different author. D_{pre} is used in order to evaluate the anonymity of D'. D' is *semi-anonymized* if a classifier *Clf* trained on D_{pre} and an arbitrary number of documents by other authors O misclassifies D' or if the similarity of D' to D_{pre} is less than threshold t, but there is a probability $>$ random chance that U authored D'.

D' is *fully anonymized* if all of the conditions of semi-anonymization hold, except the probability that U authored document D' is below random chance.

3.2 Performing document classification

In order to anonymize a document, we need to build a model of the author's writing style. We do this by defining a set of *features* found in a number of the user's other documents, and then using a supervised learning model to generalize the author's writing style. After the creation of the model, we generate a threshold t from the author's other features D_{pre}, which we then use to decide whether or not it is likely that author U wrote document D.

3.2.1 Feature extraction

Before training a classifier, we must first extract features from documents in sets D_{pre} and O. We use the Basic-9 Feature Set, described in Figure 3.1 [18]. The Basic-9 Feature set was primarily chosen for its ease of implementation and for purposes of direct comparison with Anonymouth.

Unique Words Count
The Number of unique words in the document after removing punctuation and unifying case.

Complexity
Ratio of unique words to total number of words in the document.

Sentence Count
The number of sentences in the document.

Average Sentence Length
The total number of words divided by the total number of sentences.

Average Syllables in Word
Number of syllables per word divided by the number of words.

Gunning-Fog Readability Index
The Gunning-Fog readability index is given by:

$$0.4\left[\left(\frac{totalwords}{totalsentences}\right)+100\left(\frac{totalcomplexwords}{totalwords}\right)\right],$$

where complex words are words with 3 or more syllables.

Character Space
The total number of characters in the document, including spaces.

Letter Space
The total number of letters (excluding spaces and punctuation.)

Flesch Reading Ease Score
The Flesch reading ease score is given by:

$$206.835-1.015\left(\frac{totalwords}{totalsentences}\right)-84.6\left(\frac{totalsyllables}{totalwords}\right).$$

Figure 3.1: The features contained in the Basic-9 Feature set.

12.02995346	0.25957967	0.95602251	11.63163162
13.11479054	0.04663154	0.1716382	-0.63453393

Figure 3.2: A mock feature matrix containing two document samples and four features. Each number in the matrix corresponds to a feature's value. For instance, all values in the first column could be occurrences of unique words in each of the two sample documents, all values in the second column could be the sentence count in each document, etc.

Let m, n respectively denote the number of documents (or *samples*) in D_{pre} and O, and let s be the number of features in the feature set. We store every feature in a $(m+n) \times s$ matrix called x. x consequently becomes the set of training vectors. See Figure 3.3 for a high-level description of the document extraction process.

The set of labels y has a cardinality of m. For every element of y, defined as y_i, there is a corresponding label l such that l is the author of sample x_i. For instance, if Cormac McCarthy is the author of sample x_1, then we might say that sample x_1 has a label "cm" in order to indicate that McCarthy is the author of the document that these features were extracted from.

3.2.2 Training and classification

We are now ready to train the classifier with our extracted features and labels. Once the classifier has been trained, we have "learned" how to attribute authorship to a document, and can attempt to identify the author of an unknown document.

We choose to use a Support Vector Machine (SVM) with a linear kernel as our classification model, as previous literature has suggested that SVMs perform well in authorship attribution settings, in addition to their inherent resistance against overfitting and the curse of dimensionality. Experimentation revealed small accuracy gains with a linear kernel over a radial basis function (RBF) kernel in our application; additionally, SVMs with a linear kernel are trained considerably faster than their RBF counterparts, so we chose to use a linear kernel. The precise formulation is as follows:

Let *Clf* be a Linear SVM classifier with penalty factor $C = 1.0$. We train *Clf* on (x, y), yielding the learned function $g : X \to Y$, where X is a feature vector and Y is the predicted label of X. We extract the features of D, giving us the feature vector D_{feat} of length f, and then predict the label of document D.

$$label = g(D_{feat})$$

.

Presumably, the predicted label is the label corresponding to the author's identity; else a misclassification has occurred, and the document is already at least somewhat anonymized against the feature set.

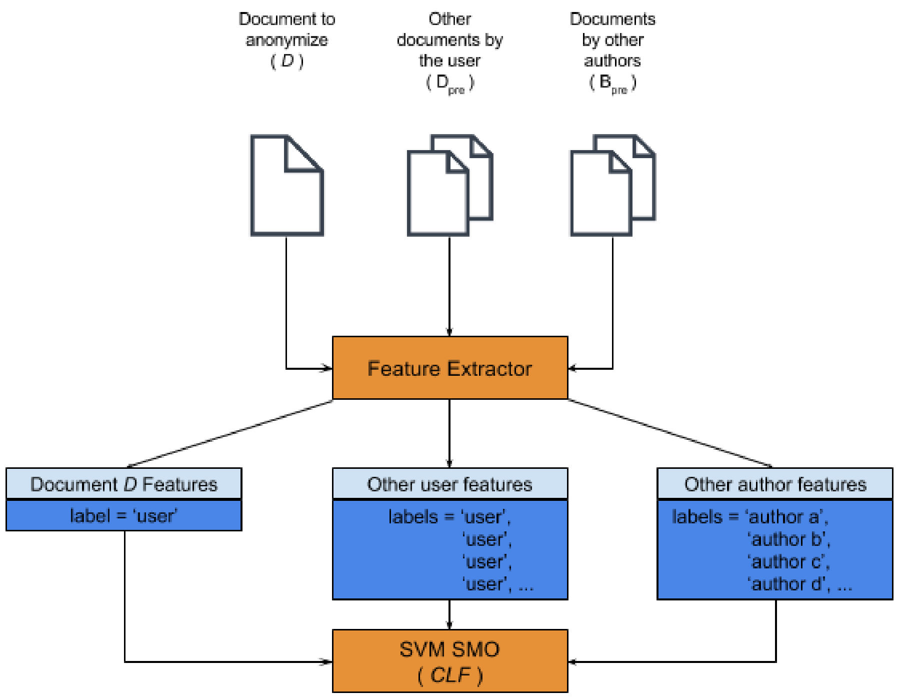

Document to anonymize (D)
Other documents by the user (D_pre)
Documents by other authors (B_pre)
Feature Extractor
Document D Features
label = 'user'
Other user features
labels = 'user', 'user', 'user', 'user', ...
Other author features
labels = 'author a', 'author b', 'author c', 'author d', ...
SVM SMO (CLF)

Figure 3.3: A high-level description of the feature extraction process.

3.2.3 Verification: Computing cosine distance and t

Now that the classifier is trained, we use a verification function to decide whether or not the user U is a likely author of document D. This makes our classification model considerably more robust, as we can notify the user when she has changed her document to the extent that she would not be detected in an "open world" setting, and would cast serious doubt over her authorship in a "closed world" setting.

Verification can be performed by any distance-like function; we use cosine similarity because it has been shown to be effective for stylometric applications [27].

The cosine distance function δ is defined as:

$$\delta(M, F) = \frac{M \cdot F}{\|M\| \cdot \|F\|} = \frac{\sum_{i=1}^{n} m_i f_i}{\sqrt{\sum_{i=1}^{n} m_i^2}\sqrt{\sum_{i=1}^{n} f_i^2}}$$

where M, F are feature vectors, m_i and f_i are the ith entries of M and F, and n is the number of elements in the feature vector. If $\delta(x, y) < \delta(x, z)$, we say that x is closer to y than to z.

To decide whether or not a potential author could have written a document, we check if the cosine distance is below a certain threshold t. The exact process of computing t is defined in Algorithm 1.

Algorithm 1: Computing the threshold

```
input  : Feature matrix of $D_{pre}$
output: The verification threshold t
list;
foreach $x$, $y$ in combinations($D_{pre}$) do
    append CosineDistance($D_{pre}x, D_{pre}y$) to list
end
t = mean(list) + stddeviation(list)
return t
```

If $\delta(D_{pre}$ features, $D') < t$, then we say that the author of D_{pre} is likely an author of D'.

3.2.4 Computation of target features with ak-means

In order to help users anonymize their document, we find a set of "sane" (e.g., reachable) configurations of feature set values that D' could potentially adopt. We then search for a configuration that defeats the classifier, and notify the user of the computed target values.

When we say we are looking for a set of "sane" target features, we mean that we want to have some amount of confidence that it is actually possible for a user to change their document's features to these target values. To do this, we use *ak-means clustering* on every feature. Informally, ak-means clustering is just repeated use of k-means clustering; we repeat k-means clustering until every cluster has at least 3 samples. The purpose of requiring every cluster to have 3 members is that it decreases the probability of a difficult-to-reach target value being chosen. We precisely describe the process of ak-means clustering in Algorithm 2 [18].

Finding possible target values is more challenging than it initially appears, as features in the feature set may have relationships to each other. For instance, "unique words" and "complexity" are obviously intimately related to each other. In order to address this issue, we cluster every feature in O and D_{pre}, and choose the target features from the resulting clusters.

As of now, we naively test random target clusters against *Clf* until the classifier produces a mislabeled result. The drawback of this simple target selection process is that it potentially produces target features quite some distance away from D's feature set. Ideally, the target feature set should be as close as possible to D's feature set while still confounding the classifier.

3.3 Approach to evaluating the classifier's performance

We performed several tests in order to verify the accuracy of the SVM SMO classifier with the Basic-9 feature set. In every test, we use k-fold cross-validation to measure accuracy, where k is equal to the number of documents found in the smallest author-document set, unless the smallest set contains at least 10 authors, in which case we use 10-fold cross-validation. For instance, the Drexel corpus contains 13 authors; each containing between 7 and 11 documents. This means that we compute 7-fold cross-validation on the Drexel set, since the lowest number of documents provided for a single author is 7.

This scheme for selecting k is likely confusing to the reader, yet the nature of our datasets necessitates it, as each label in the Brennan-Greenstadt Corpus has a different number of samples, and k cannot exceed the smallest number of samples found in all of the labels. See the appendix for a more precise description of the process of cross-validation and our selection of k; results are discussed in the next chapter.

Algorithm 2: ak-means algorithm

```
input  : Feature matrix $X$, number of authors in $X$
output: A set of clusters containing at least 3 documents
clusters ← kmeans++(otherDocumentFeatures, number of authors)

foreach cluster in clusters do
    if cluster contains $< 3$ labels then
        return ak-means($X$, number of authors in $X - 1$)
    else
        return clusters
    end
end
```

3.4 Corpora used for evaluations

We use the Brennan-Greenstadt Adversarial Corpus [18] and the Reuters C50 Corpus [17]. The former is a collection of 114 documents, each of approximately 500 words in length, sampled from 13 different authors. The Reuters C50 Corpus contains 5000 documents written by 50 authors. It has been split into training and test data, with both subsets being of the same size.

Chapter 4

Results

4.1 The Unstyle interface

4.1.1 Interface Design

The user is asked to input a single document to anonymize, and a set of documents also written by the user. After the user clicks next, in less than one minute, the user is presented the results of classification and given access to a text editor containing her document. A table with the most important features (features that affect the classifier the most) appears, listing initial and target values of each feature. Clicking on a feature results in a short description of the feature being printed on the screen. As the user edits her document, one of three status messages are displayed in the lower-left corner: not anonymized, semi-anonymized, or anonymized. Each status message is accompanied by a corresponding icon and brief message, clearly distinguishing each kind of anonymity. Once the user has reached a satisfactory amount of anonymity, she may click the "Save As" button and write out her document.

4.1.2 Interface and user experience discussion

In comparison with existing adversarial stylometry tools, Unstyle has an exceptionally easy to use interface, considerably simplifying the process of anonymizing a document. See Figures 4.1 through 4.5 for screenshots of Unstyle.

One important feature of Unstyle is the automatic selection of documents in the training set. The user is only required to input the document to anonymize and a set of other self-authored sample documents, eliminating the cumbersome step of loading and manually labeling documents by other authors. This approximately cuts the time it takes to input documents in half.

Unstyle is highly stable, and is yet to crash on any of the 5114 documents that it has been tested on. Improperly encoded files are automatically sanitized upon loading, eliminating a major source of memory corruption.

Perhaps Unstyle's most useful feature is its real-time classification of the user's document. Every time a character is changed, features are reextracted from the current state of the user's document and tested on a trained classifier. Cosine similarity is

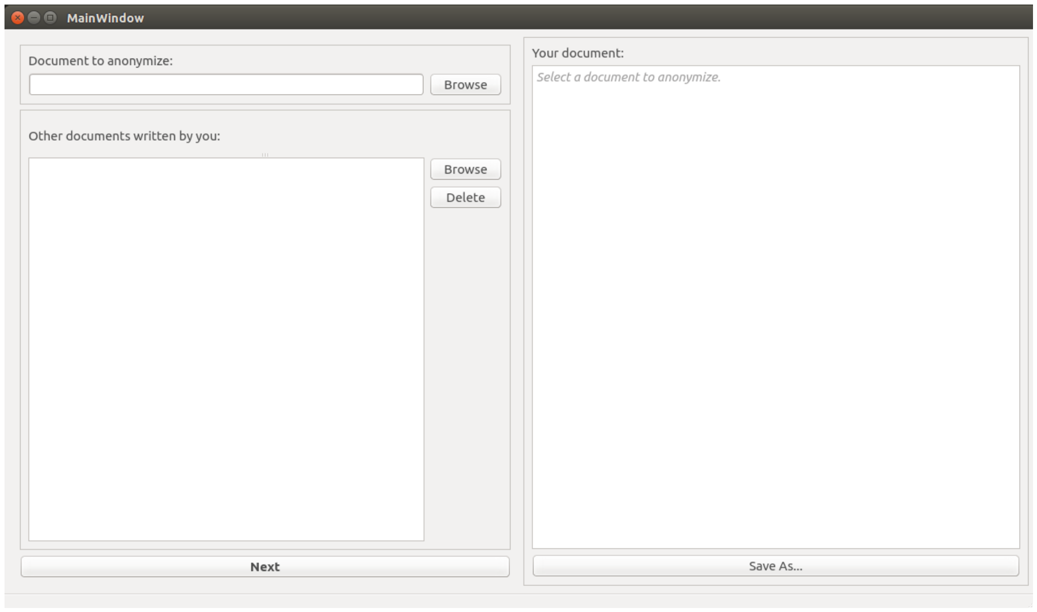


Figure 4.1: The screen shown when Unstyle is started.

recomputed at this time as well. As soon as the author has appropriately anonymized their document, the user receives a simple status message telling him that he may stop editing his document. On a modern machine[1], this process is seamless, and there is no ‘stuttering’ while editing the document.

Unstyle’s status messages are simple, but highly informative. Users are made aware of their “level” of anonymity at every instant.

The Unstyle interface is not perfect, however. The display of target features is, at this time, not particularly useful to the user. See “Evaluation of target features” in this chapter for more details.

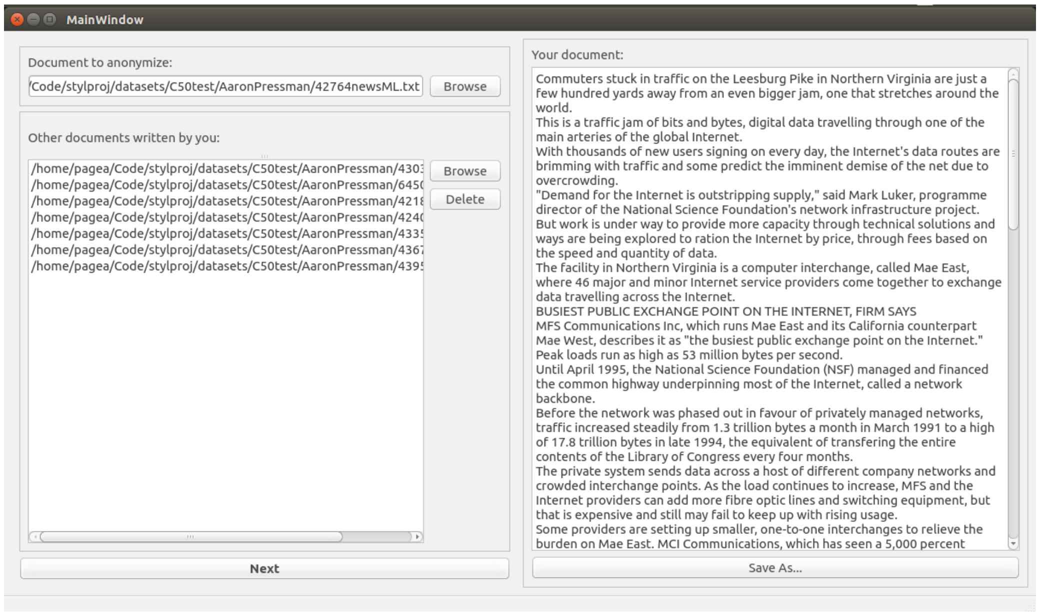
4.2 Evaluating the SVM SMO Classifier

In the first test, we performed 7-fold cross-validation on the entire 13-author Brennan-Greenstadt adversarial corpus. This yielded 63% accuracy.

In the second test, we select 3 random documents from each author in the Drexel corpus to create the training set. We then select 3 random documents *not* in the training set to create the test set, and perform 3-fold cross-validation. This yielded a score of 54%, which is quite close to the reported cross-validated accuracy of Anonymouth’s classification scheme (53.98%). See Figure 4.6 for a confusion matrix[2] associated with this test.

[1]A ‘modern machine’ in this context is defined as possessing equivalent or greater processing power in comparison to an Acer Aspire 5742-6838; Unstyle has not been tested on any other hardware platforms at this time.

[2]For information related to reading confusion matrices, see Appendix A.



Document to anonymize:

'Code/stylproj/datasets/C50test/AaronPressman/42764newsML.txt

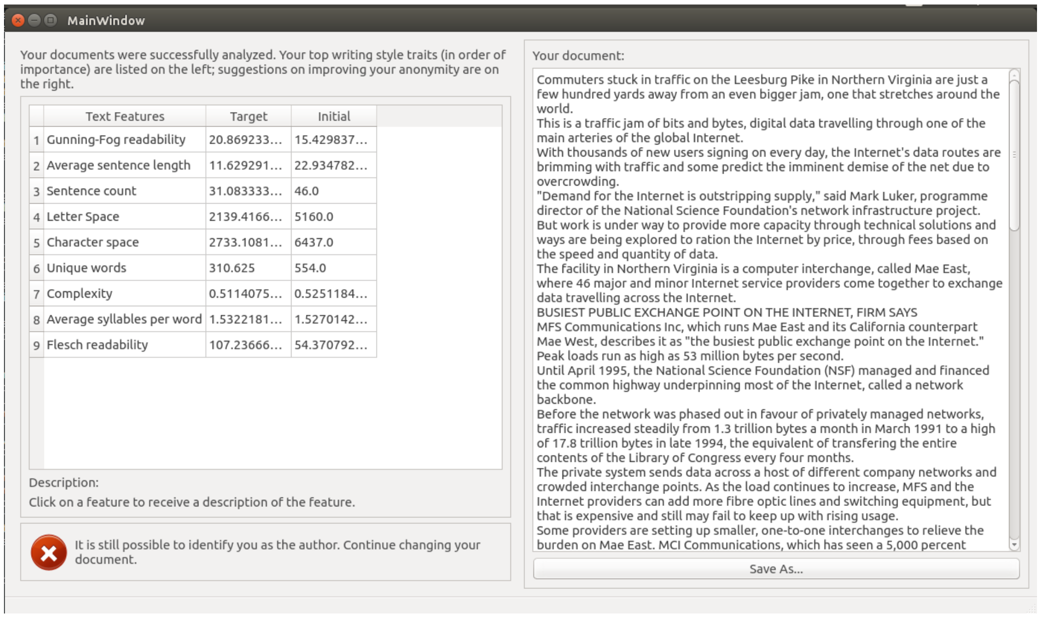
Other documents written by you:

```
/home/pagea/Code/stylproj/datasets/C50test/AaronPressman/430
/home/pagea/Code/stylproj/datasets/C50test/AaronPressman/645
/home/pagea/Code/stylproj/datasets/C50test/AaronPressman/421
/home/pagea/Code/stylproj/datasets/C50test/AaronPressman/424
/home/pagea/Code/stylproj/datasets/C50test/AaronPressman/433
/home/pagea/Code/stylproj/datasets/C50test/AaronPressman/436
/home/pagea/Code/stylproj/datasets/C50test/AaronPressman/439
```

Your document:

Commuters stuck in traffic on the Leesburg Pike in Northern Virginia are just a few hundred yards away from an even bigger jam, one that stretches around the world.
This is a traffic jam of bits and bytes, digital data travelling through one of the main arteries of the global Internet.
With thousands of new users signing on every day, the Internet's data routes are brimming with traffic and some predict the imminent demise of the net due to overcrowding.
"Demand for the Internet is outstripping supply," said Mark Luker, programme director of the National Science Foundation's network infrastructure project.
But work is under way to provide more capacity through technical solutions and ways are being explored to ration the Internet by price, through fees based on the speed and quantity of data.
The facility in Northern Virginia is a computer interchange, called Mae East, where 46 major and minor Internet service providers come together to exchange data travelling across the Internet.
BUSIEST PUBLIC EXCHANGE POINT ON THE INTERNET, FIRM SAYS
MFS Communications Inc, which runs Mae East and its California counterpart Mae West, describes it as "the busiest public exchange point on the Internet."
Peak loads run as high as 53 million bytes per second.
Until April 1995, the National Science Foundation (NSF) managed and financed the common highway underpinning most of the Internet, called a network backbone.
Before the network was phased out in favour of privately managed networks, traffic increased steadily from 1.3 trillion bytes a month in March 1991 to a high of 17.8 trillion bytes in late 1994, the equivalent of transfering the entire contents of the Library of Congress every four months.
The private system sends data across a host of different company networks and crowded interchange points. As the load continues to increase, MFS and the Internet providers can add more fibre optic lines and switching equipment, but that is expensive and still may fail to keep up with rising usage.
Some providers are setting up smaller, one-to-one interchanges to relieve the burden on Mae East. MCI Communications, which has seen a 5,000 percent

Figure 4.2: The user has loaded some documents.



Your documents were successfully analyzed. Your top writing style traits (in order of importance) are listed on the left; suggestions on improving your anonymity are on the right.

	Text Features	Target	Initial
1	Gunning-Fog readability	20.869233...	15.429837...
2	Average sentence length	11.629291...	22.934782...
3	Sentence count	31.083333...	46.0
4	Letter Space	2139.4166...	5160.0
5	Character space	2733.1081...	6437.0
6	Unique words	310.625	554.0
7	Complexity	0.5114075...	0.5251184...
8	Average syllables per word	1.5322181...	1.5270142...
9	Flesch readability	107.23666...	54.370792...

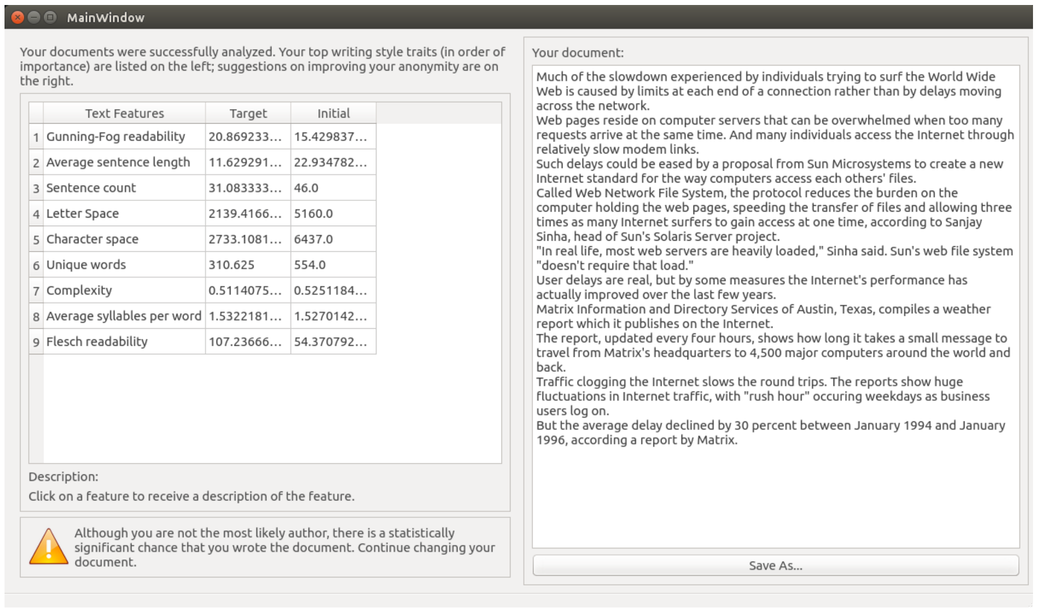
Description:
Click on a feature to receive a description of the feature.

It is still possible to identify you as the author. Continue changing your document.

Your document:

Commuters stuck in traffic on the Leesburg Pike in Northern Virginia are just a few hundred yards away from an even bigger jam, one that stretches around the world.
This is a traffic jam of bits and bytes, digital data travelling through one of the main arteries of the global Internet.
With thousands of new users signing on every day, the Internet's data routes are brimming with traffic and some predict the imminent demise of the net due to overcrowding.
"Demand for the Internet is outstripping supply," said Mark Luker, programme director of the National Science Foundation's network infrastructure project.
But work is under way to provide more capacity through technical solutions and ways are being explored to ration the Internet by price, through fees based on the speed and quantity of data.
The facility in Northern Virginia is a computer interchange, called Mae East, where 46 major and minor Internet service providers come together to exchange data travelling across the Internet.
BUSIEST PUBLIC EXCHANGE POINT ON THE INTERNET, FIRM SAYS
MFS Communications Inc, which runs Mae East and its California counterpart Mae West, describes it as "the busiest public exchange point on the Internet."
Peak loads run as high as 53 million bytes per second.
Until April 1995, the National Science Foundation (NSF) managed and financed the common highway underpinning most of the Internet, called a network backbone.
Before the network was phased out in favour of privately managed networks, traffic increased steadily from 1.3 trillion bytes a month in March 1991 to a high of 17.8 trillion bytes in late 1994, the equivalent of transfering the entire contents of the Library of Congress every four months.
The private system sends data across a host of different company networks and crowded interchange points. As the load continues to increase, MFS and the Internet providers can add more fibre optic lines and switching equipment, but that is expensive and still may fail to keep up with rising usage.
Some providers are setting up smaller, one-to-one interchanges to relieve the burden on Mae East. MCI Communications, which has seen a 5,000 percent

Figure 4.3: The classifier still suspects the user is the author of the input document; a warning is displayed.


MainWindow

Your documents were successfully analyzed. Your top writing style traits (in order of importance) are listed on the left; suggestions on improving your anonymity are on the right.

	Text Features	Target	Initial
1	Gunning-Fog readability	20.869233...	15.429837...
2	Average sentence length	11.629291...	22.934782...
3	Sentence count	31.083333...	46.0
4	Letter Space	2139.4166...	5160.0
5	Character space	2733.1081...	6437.0
6	Unique words	310.625	554.0
7	Complexity	0.5114075...	0.5251184...
8	Average syllables per word	1.5322181...	1.5270142...
9	Flesch readability	107.23666...	54.370792...

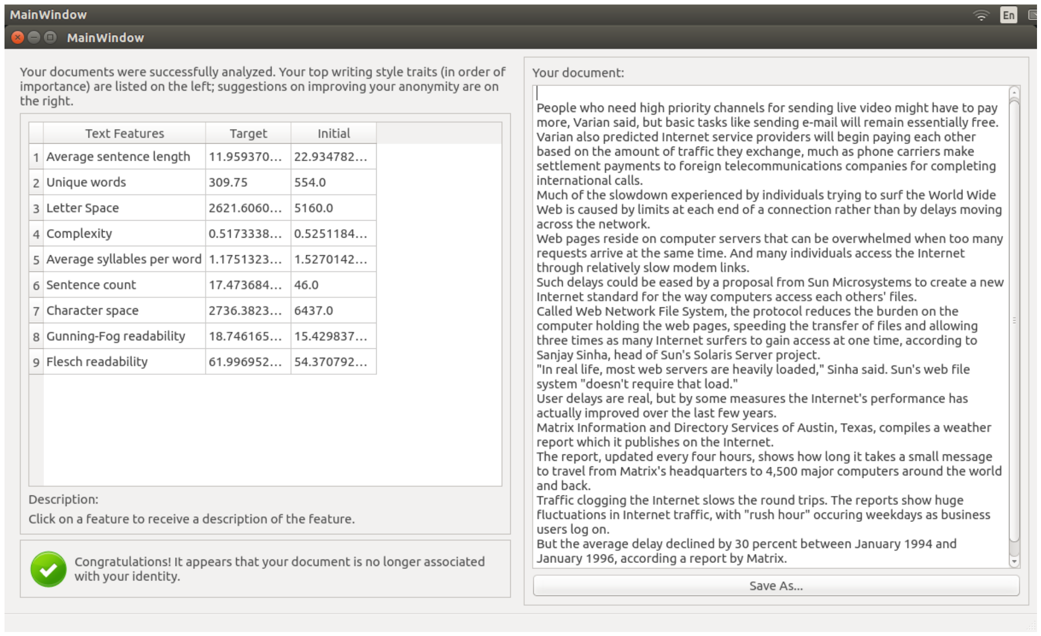
Description:
Click on a feature to receive a description of the feature.

Although you are not the most likely author, there is a statistically significant chance that you wrote the document. Continue changing your document.

Your document:

Much of the slowdown experienced by individuals trying to surf the World Wide Web is caused by limits at each end of a connection rather than by delays moving across the network.
Web pages reside on computer servers that can be overwhelmed when too many requests arrive at the same time. And many individuals access the Internet through relatively slow modem links.
Such delays could be eased by a proposal from Sun Microsystems to create a new Internet standard for the way computers access each others' files.
Called Web Network File System, the protocol reduces the burden on the computer holding the web pages, speeding the transfer of files and allowing three times as many Internet surfers to gain access at one time, according to Sanjay Sinha, head of Sun's Solaris Server project.
"In real life, most web servers are heavily loaded," Sinha said. Sun's web file system "doesn't require that load."
User delays are real, but by some measures the Internet's performance has actually improved over the last few years.
Matrix Information and Directory Services of Austin, Texas, compiles a weather report which it publishes on the Internet.
The report, updated every four hours, shows how long it takes a small message to travel from Matrix's headquarters to 4,500 major computers around the world and back.
Traffic clogging the Internet slows the round trips. The reports show huge fluctuations in Internet traffic, with "rush hour" occuring weekdays as business users log on.
But the average delay declined by 30 percent between January 1994 and January 1996, according a report by Matrix.

Figure 4.4: The user is 'semi-anonymized' as described in Chapter 3. A warning is displayed.


MainWindow

Your documents were successfully analyzed. Your top writing style traits (in order of importance) are listed on the left; suggestions on improving your anonymity are on the right.

	Text Features	Target	Initial
1	Average sentence length	11.959370...	22.934782...
2	Unique words	309.75	554.0
3	Letter Space	2621.6060...	5160.0
4	Complexity	0.5173338...	0.5251184...
5	Average syllables per word	1.1751323...	1.5270142...
6	Sentence count	17.473684...	46.0
7	Character space	2736.3823...	6437.0
8	Gunning-Fog readability	18.746165...	15.429837...
9	Flesch readability	61.996952...	54.370792...

Description:
Click on a feature to receive a description of the feature.

Congratulations! It appears that your document is no longer associated with your identity.

Your document:

People who need high priority channels for sending live video might have to pay more, Varian said, but basic tasks like sending e-mail will remain essentially free.
Varian also predicted Internet service providers will begin paying each other based on the amount of traffic they exchange, much as phone carriers make settlement payments to foreign telecommunications companies for completing international calls.
Much of the slowdown experienced by individuals trying to surf the World Wide Web is caused by limits at each end of a connection rather than by delays moving across the network.
Web pages reside on computer servers that can be overwhelmed when too many requests arrive at the same time. And many individuals access the Internet through relatively slow modem links.
Such delays could be eased by a proposal from Sun Microsystems to create a new Internet standard for the way computers access each others' files.
Called Web Network File System, the protocol reduces the burden on the computer holding the web pages, speeding the transfer of files and allowing three times as many Internet surfers to gain access at one time, according to Sanjay Sinha, head of Sun's Solaris Server project.
"In real life, most web servers are heavily loaded," Sinha said. Sun's web file system "doesn't require that load."
User delays are real, but by some measures the Internet's performance has actually improved over the last few years.
Matrix Information and Directory Services of Austin, Texas, compiles a weather report which it publishes on the Internet.
The report, updated every four hours, shows how long it takes a small message to travel from Matrix's headquarters to 4,500 major computers around the world and back.
Traffic clogging the Internet slows the round trips. The reports show huge fluctuations in Internet traffic, with "rush hour" occuring weekdays as business users log on.
But the average delay declined by 30 percent between January 1994 and January 1996, according a report by Matrix.

Figure 4.5: The user has successfully anonymized his or her document against the feature set; a congratulatory message is shown.

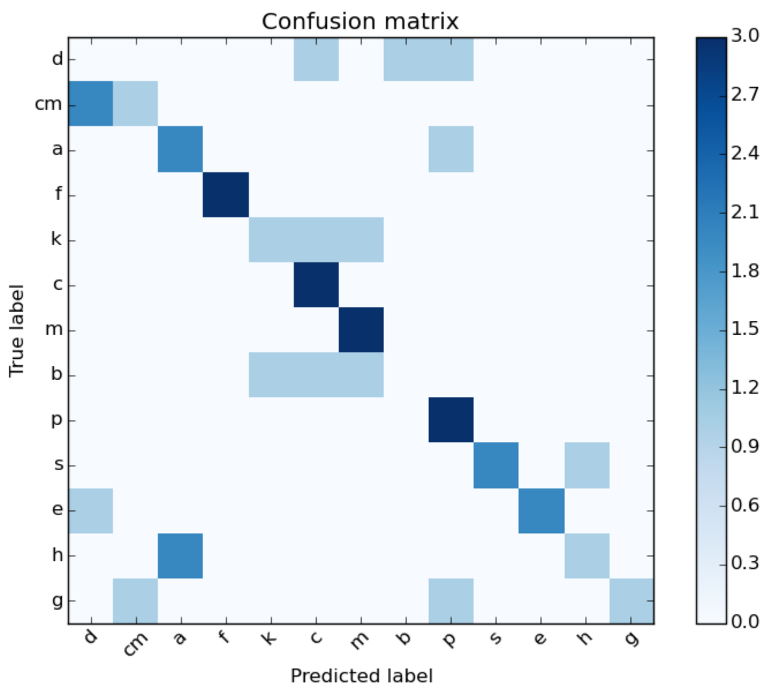


Figure 4.6: A confusion matrix depicting the classifier's performance on 13 authors in the Drexel dataset. Both the training and test sets had three documents per author.

In the final test, we perform 10-fold cross-validation on 13 authors in the C50 dataset. This yielded 36% accuracy, considerably lower than on the Adversarial Corpus. There could be a number of factors confounding stylometric analysis on this dataset. Perhaps many of the writing samples were modified by the same editor, as they are all from the same news source. More likely, the Basic-9 feature set simply does not do a great job of capturing stylistic information in this genre of writing. A "deeper" feature set, such as Writeprints, would likely perform better. See Figure 4.7 for a confusion matrix associated with this test. See Figure 4.8 for a table recording precision, recall, and F1-score for each label [15].

4.3 Evaluation of cosine similarity threshold

Before the addition of the cosine similarity measure, Unstyle was unfairly biased towards the user. For instance, rewriting the document to include nothing but the word "foobar" still resulted in the classifier labeling the document as written by the user. It quickly became apparent that this was due to the "closed world" approach to the problem; Unstyle would mislabel any writing style it did not recognize.

Implementing the cosine similarity threshold solves this problem, as Unstyle now behaves properly if major changes are made to the document. Furthermore, the relatively liberal threshold choice of t ensures that classification accuracy is not reduced.

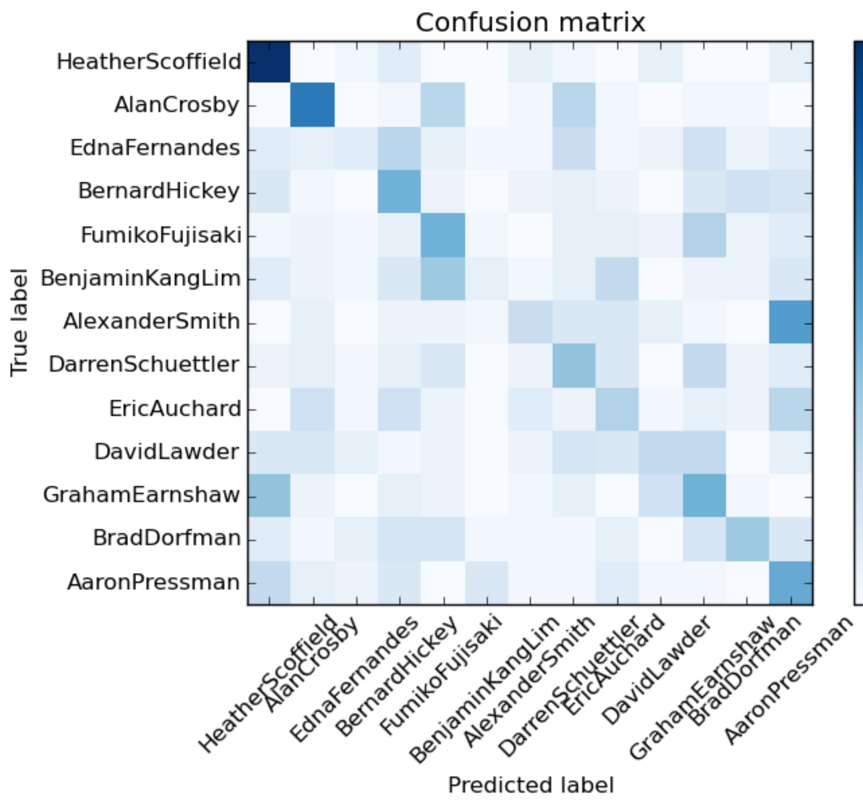


Figure 4.7: A confusion matrix depicting the classifier's performance on 13 authors in the Reuters dataset.

Label	Precision	Recall	F1-score
0	0.42	0.70	0.53
1	0.44	0.50	0.47
2	0.24	0.08	0.12
3	0.25	0.34	0.29
4	0.27	0.34	0.30
5	0.25	0.06	0.10
6	0.30	0.16	0.21
7	0.23	0.28	0.25
8	0.22	0.22	0.22
9	0.32	0.18	0.23
10	0.24	0.34	0.28
11	0.41	0.26	0.32
12	0.22	0.36	0.27
avg / total	0.29	0.29	0.28

Figure 4.8: Performance with 13 authors in the C50 dataset.

At this time, the verification step is being underutilized, as the Classify-Verify algorithm is not fully implemented. More than anything else, the current incarnation of the verification step serves as a sanity check for our classifier. A complete implementation of the Classify-Verify algorithm could potentially increase the accuracy of the classifier, thereby resulting in better anonymity for the end-user.

4.4 Evaluation of target features

Target feature selection performs poorly. Although target values are guaranteed to confound the classifier if adopted, the target values are often times not realistic for the document to adopt. Much of the time, features are too far away.

Close tracing of Unstyle's clustering process has revealed that performance of the ak-means algorithm is often times less than optimal. All too frequently, the ak-means algorithm outputs a single cluster. Even when ak-means works properly, it takes at least five recursive iterations before a cluster contains at least three documents. Furthermore, the clustering process is the slowest part of the analysis pipeline, taking up the vast majority of computation time and causing the application to hang for a couple tens of seconds.

4.5 Summary of results

We designed and implemented a simple-to-use, extendable, and accurate adversarial stylometry tool. We give a short overview of the interface, and provide screenshots of the resulting application.

We have shown that, when using the same feature sets and classifiers, Unstyle performs as well as Anonymouth on the Brennan-Greenstadt Adversarial Corpus.

We then ran a similar test on the Reuters C50 corpus, and received less impressive results. This further highlights the importance of extending Unstyle to support the Writeprints feature set.

We then critically analyze several internal features of Unstyle, some of which need considerable adjustment before they can be useful to the end user.

4.6 Threats to validity

The Anonymouth team evaluates their classifier's accuracy using 10-fold cross validation. Upon attempting to reproduce this step, it became apparent that it is not possible to do 10-fold cross validation on the Drexel feature set, due to the fact that one author only has 7 sample documents, thus making 7-fold cross-validation the highest degree of validation possible. It is not clear whether this is due to an oversight of the Anonymouth team or an experimental design issue on our part; nonetheless, results from cross-validation in our experiments were quite similar to those provided by the Anonymouth team. We will contact the developers of Anonymouth in order to resolve these methodoligical discrepancies.

When evaluating the classification methodology, we do not use Unstyle's functions to compute accuracy scores, but a separate script called `classifyTest.py` in which the same methodology is followed. The benefit of this approach is that it is very easy to experiment with methodologies; the drawback is that there could be some lurking bug negatively affecting the accuracy of Unstyle; conversely, there could be problems with `classifyTest.py` distorting the accuracy. Fortunately, extensive testing of Unstyle suggests that the application performs as intended.

Chapter 5

Conclusion

5.1 Future work

5.1.1 Implementing a better feature set

The Basic-9 feature set was primarily chosen for its simplicity. Although it performs quite well with small sets of authors, there are feature sets that scale much better and have higher accuracy. The Writeprints feature set represents the current state of the art in stylometry; as such, Unstyle should be made to use this feature set, or at least some subset of the feature set, in order to be useful against real stylometric attacks [3].

While implementing the Writeprints feature set and analyzer is not a trivial task, it is quite feasible that Unstyle could be extended to make use of the 23 feature extractors in the Writeprint featureset. Several of these features are already included in NLTK or Scikit-Learn[1].

5.1.2 Computing better target features

Preliminary results suggest that target feature values suggested by Unstyle are not very useful. Target feature values are often unrealistically far away from the current state of the document, and reaching them would require significantly shrinking the document. While the target features "beat" the classifier, this is clearly not particularly useful if reaching the target features is impossible or extremely tiresome.

5.1.3 Partial automation of obfuscation of style

See the Related Work chapter for reflections on the feasibility of automatic paraphrasing and its potential value for adversarial stylometry.

[1]NLTK is the Natural Language Toolkit, a Python framework for performing certain natural language programming tasks, including feature extraction. Scikit-Learn is a Python framework for performing machine learning and feature extraction tasks.

5.2 Conclusion

The combination of the development of advanced methods of stylometric analysis and the ever-increasing glut of easily collected online writing samples—public or otherwise—poses great risk to those whose safety or way of life is conditional on online anonymity. We have also suggested that the number of people who may be affected by stylometry is increasing as time passes. As a result, it is critical that a high-quality tool for obfuscating stylistic features becomes available to the general public.

By building on existing work in adversarial stylometry and releasing a high quality, thoroughly documented open-source tool with a permissive license, we have made considerable progress in making such a tool a reality. It is our hope that the fully open-source, accessible nature of this tool will help foster a community of dedicated developers and, eventually, an equally dedicated community of users.

Appendix A

Explanation of metrics

A.1 Understanding k-fold cross-validation

k-fold cross-validation is a statistical method for evaluating the performance of a predictive model. We use it to test the accuracy of the SVM SMO.

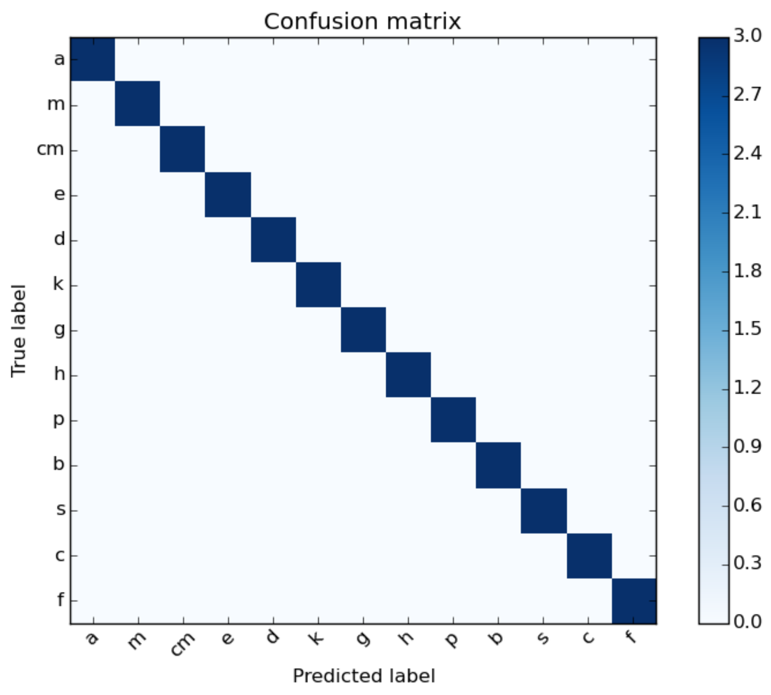
In k-fold cross-validation, we divide our set of feature vectors and their corresponding labels into k subsets. This results the creation of many different training and test datasets, all of which are tested against one another and averaged together, resulting in a realistic accuracy score with minimal variance.

For most applications, 10 is a reasonable value for k. Since one of our datasets is small, however, we must use a smaller value for k. More specifically, since the Brennan-Greenstadt Corpus has an author with only 7 document samples, we cannot choose any value k greater than 7. Intuitively, this makes sense – one cannot divide a set of 7 objects into 10 partitions. We could simply split the 7 documents into 10 documents of shorter length, but that would mean we have modified our dataset, which is unacceptable. Instead, we use 7-fold cross-validation.

A.2 Reading confusion matrices

Confusion matrices are an easy way of visualizing the performance of a classification algorithm. Labels along the x-axis are the predicted label, while labels along the y-axis are the true label. As such, higher values along the main diagonal (the line drawn between the upper left corner and the lower right corner of the matrix) indicate that a classifier is more accurate, while a confusion matrix with many misclassifications will have low values in the main diagonal and higher values everywhere else.

If a classification algorithm has an accuracy of 100%, this is what the confusion matrix will look like:


Confusion matrix
True label
Predicted label
a
m
cm
e
d
k
g
h
p
b
s
c
f
3.0
2.7
2.4
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0.0

Appendix B

Extending Unstyle

Adversarial stylometry is a young discipline; ten years ago, the field did not exist. As with any new field of study, it is one that is prone to rapid change and growth. In order for Unstyle to continue to be useful to users and researchers, Unstyle must be robust enough to keep up with advances in stylometry. To this end, we have carefully separated the implementation of the interface and the backend using the Model-View-Controller architecture. In addition, Unstyle uses a simple feature set framework that is easily extended by any researcher with a modest understanding of Python.

```
from stylproj.featuresets.feature_extract import FeatureSetExtractor
from stylproj.features import characterfeatures
from stylproj.features import lexicalfeatures

class Basic9Extractor(FeatureSetExtractor):
    def __init__(self):
        self.features = [
            "letterSpace",
            "gunningFog",
            "avgSyllablesPerWord",
            "unique_words",
            "sentenceCount",
            "characterSpace",
            "avgSentenceLength",
            "complexity",
            "fleschReadingEase"
        ]
```

Figure B.1: The implementation of the Basic-9 Feature Set (documentation and citation omitted)

Feature sets are defined as lists of feature extractor names, all defined in the `:/stylproj/featuresets` directory. As seen in Figure B.1, composing a feature set from existing feature extractors is trivial. Even researchers with only the most basic understanding of Python can create their own feature set by using basic9.py as a template.

Implementing custom feature extractors, however, is necessarily more complex, though we take certain steps to make the process less painful. Every feature extractor is expected to do the following:

1. Return the value of a feature in a given text document.
2. Normalize, if desired.
3. Register itself as a feature using the `@register_feat` decorator.

Normalization refers to the process of adjusting the value of a feature so it is comparable to features across other documents. For instance, researchers often normalize features for length by dividing their values by the length of the document. This allows researchers to compare features across documents of varied lengths without distorting the dataset.

"Registering" a feature extractor, as seen in Figure B.2 simply means adding `@register_feat` on the line before the definition of the feature extraction function. This tells Unstyle to make the function available in feature set definitions.

```
import numpy
import nltk
import collections

from stylproj.features.featregister import register_feat
from stylproj import adversarial
from stylproj import langtools

@register_feat
def gunningFog(text):
    """Return the Gunning-Fog readability measure."""
    tokens = langtools.tokenize(text)
    complexWords = 0;
    for word in tokens:
        syllables = langtools.syllableCount(word)
        if syllables is not None:
            if syllables >= 3:
                complexWords += 1
    totalWords = len(tokens)
    totalSentences = sentenceCount(text)
    return 0.4*((totalWords/totalSentences) + 100*(complexWords/
       totalWords))

@register_feat
def fleschReadingEase(text):
    """Return the Flesch reading ease score."""
    tokens = langtools.tokenize(text)
    totalWords = len(tokens)
    totalSentences = sentenceCount(text)
    totalSyllables = 0
    for word in tokens:
        syllablesInWord = langtools.syllableCount(word)
        if syllablesInWord:
            totalSyllables += langtools.syllableCount(word)

    return (206.835 - ((1.015*totalWords) / totalSentences) - 84.6*(
       totalSyllables /
    totalWords))
```

Figure B.2: An example module containing lexical feature extractors.

Appendix C

Downloading and contributing to Unstyle

The full source code can be found at `https://github.com/pagea/unstyle`.

Pull requests are both welcome and encouraged.

Appendix D

Source code

D.1 Unstyle core and support scripts

main.py:

```
if __name__ == '__main__':
    import sys
    import stylproj.controller
    from stylproj.controller import startGUI
    startGUI()
```

styl_math.py:

```
"""Various mathematical functions that are useful for performing
   stylometry.
"""
```

```
from numpy.linalg import norm


def cosine_distance(m, f):
    """Compute the cosine distance between a "model" M from an author's
       set of
    documents and a featureset F extracted from D (the document to
       anonymize.)
    Note that m and f must be of equal length for this computation to be
    meaningful.

    :param m: A set of averages of every author's feature.
    :param f: The featureset extracted from D.
    :rtype: A distance such that, given cosine_distance(x, y) <
    cosine_distance(x, z), we say that x is "closer to" y than z.
    """
    return (1 - ((m * f).sum() / (norm(m) * norm(f))))
```

features/featregister.py:

```
"""The feature extractor register. This keeps track of all feature
   extractors.
We compose feature sets (subsets of stylproj's set of all feature
   extractors) by accessing this dictionary.

** The feature register should only be used by DocumentHandlers. **

Note that the featregistry is ordered.

Use the @register_feat decorator to register a feature extractor with
   stylproj.
"""

from collections import OrderedDict

featregistry = OrderedDict()


def register_feat(func):
    """Adds decorated function to a dictionary in the form {'name of
    function':function}. For instance, we add the characterSpace feature
        to our
    registry by adding @register_feat before the definition:

    @register_feat
    def characterSpace(text):
        ...

    Then the state of our dictionary becomes:
    {'characterSpace' : <function characterfeatures.characterSpace>}

    This makes it easy to make a featureset composed of whatever
       functions we
    would like to experiment with.
    """

    name = func.__name__
    featregistry[name] = func

    return func
```

features/characterfeatures.py:

```
"""Module containing character feature extractors."""

import string
from stylproj.features.featregister import register_feat


@register_feat
def characterSpace(text):
    """Return the total number of characters."""
    return len(text)


@register_feat
def letterSpace(text):
    """Return the total number of letters (excludes spaces and
       punctuation)"""

    count = 0
    alphabet = string.ascii_lowercase + string.ascii_uppercase
    for char in text:
        if char in alphabet:
            count += 1
    return count
```

features/lexicalfeatures.py:

```
"""Module containing lexical feature extractors."""

import numpy
import nltk
import collections

from stylproj.features.featregister import register_feat
from stylproj import adversarial
from stylproj import langtools


@register_feat
def unique_words(text):
    """Return the number of unique words."""
    tokens = langtools.tokenize(text)
    wordSet = set()
    for token in tokens:
        wordSet.add(token)
    return len(wordSet)


@register_feat
def complexity(text):
    """Return the ratio of unique words to total number of words in the
    document.
    """
    tokens = langtools.tokenize(text)
    unique = unique_words(text)
    wordCount = len(tokens)
    complexityRatio = unique / wordCount
    return complexityRatio


@register_feat
def sentenceCount(text):
    """Get the number of sentences using NLTK's sentence tokenizer.
    """
    return len(nltk.tokenize.sent_tokenize(text))


@register_feat
def avgSentenceLength(text):
    """Return the average length of a sentence."""
    tokens = langtools.tokenize(text)
    return len(tokens) / sentenceCount(text)


@register_feat
def avgSyllablesPerWord(text):
    """Return the average number of syllables per word."""
    tokens = langtools.tokenize(text)
    totalWords = len(tokens)
    totalSyllables = 0
    for word in tokens:
        syllablesInWord = langtools.syllableCount(word)
        # Ignore words not found in our dictinoary.
        if syllablesInWord is not None:
            totalSyllables += syllablesInWord
    return totalSyllables / totalWords


@register_feat
def gunningFog(text):
    """Return the Gunning-Fog readability measure."""
    tokens = langtools.tokenize(text)
    # Complex words are words with 3 or more syllables.
    # print(str(tokens))
    complexWords = 0
    for word in tokens:
        syllables = langtools.syllableCount(word)
        if syllables is not None:
            if syllables >= 3:
                complexWords += 1
    totalWords = len(tokens)
    totalSentences = sentenceCount(text)
    return 0.4 * \
        ((totalWords / totalSentences) + 100 * (complexWords /
           totalWords))


@register_feat
def fleschReadingEase(text):
    """Return the Flesch reading ease score."""
    tokens = langtools.tokenize(text)
    totalWords = len(tokens)
    totalSentences = sentenceCount(text)
    totalSyllables = 0
    for word in tokens:
        syllablesInWord = langtools.syllableCount(word)
        if syllablesInWord:
            totalSyllables += langtools.syllableCount(word)

    return (206.835 - ((1.015 * totalWords) / totalSentences) -
            84.6 * (totalSyllables / totalWords))
```

featuresets/feature_extract:

```
from stylproj.features.featregister import featregistry


class FeatureSetExtractor:

    """Superclass that all feature sets should extend."""

    def __init__(self):
        self.features = []

    def extract(self, text):
        """The procedure to extract all of the features from a body of
           text.
        Returns a list of extracted features.
        """
        self.extractor = None
        self.extractedFeatures = []
        #print("\nNext feature extract cycle: ")
        for feature in self.features:
            self.extractor = featregistry[feature]
            self.extractedFeatures.append(self.extractor(text))
            #print("EXTRACTED feature: " + str(featregistry[feature]))

        return self.extractedFeatures
```

featuresets/basic9.py:

```
from stylproj.featuresets.feature_extract import FeatureSetExtractor
from stylproj.features import characterfeatures
from stylproj.features import lexicalfeatures


class Basic9Extractor(FeatureSetExtractor):

    """The Basic-9 feature set extractor.[1]

    [1] Michael Brennan and Rachel Greenstadt. Practical Attacks Against
        Authorship
    Recognition Techniques in Proceedings of the Twenty-First Conference
        on
    Innovative Applications of Artificial Intelligence (IAAI), Pasadena,
        California,
    July 2009.
    """

    def __init__(self):
        self.features = [
            "letterSpace",
            "gunningFog",
            "avgSyllablesPerWord",
            "unique_words",
            "sentenceCount",
            "characterSpace",
            "avgSentenceLength",
            "complexity",
            "fleschReadingEase"
        ]
```

feat_select.py:

```
# feat_select.py
""" Tools for feature selection.
"""

from collections import OrderedDict
from numpy import sort
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import RFE
from sklearn.svm import LinearSVC
from sklearn.cluster import KMeans
from stylproj.features.featregister import featregistry

# TODO: Grid search with k-fold cross-validation on classifier in order
   to see
# if better ranking results can be achieved.


def rank_features_dt(X, y, featureset):
    """Rank features by their importance with a decision tree.
    This does not seem to agree with rank_features_rbe, although it is
    considerably faster. Using this with SVM classifiers is likely a bad
         idea.

    :param X: A training set of features.
    :param y: A target set (aka class labels for the training set)
    :param featureset: An instance of a featureset (such as
       Basic9Extractor())
    :rtype: An OrderedDict of the form {K : V}, with K being the feature
         name
    and V being its importance. This dictionary will be sorted by
       importance.
    """

    classifier = ExtraTreesClassifier(n_estimators=500, n_jobs=-1)
    classifier.fit(X, y)
    importances = classifier.feature_importances_

    feat_importance = OrderedDict()

    # Get the names of the feature columns.
    for index, func in enumerate(featureset.features):
        feat_importance[func] = importances[index]

    print(feat_importance)

    # Sort the dictionary by value and return it.
    return sorted(feat_importance.items(), key=lambda x: x[1], reverse=
       True)


def rank_features_rfe(X, y, featureset):
    """Rank features by their importance using recursive feature
       elimination.

    :param X: A training set of features.
    :param y: A target set (aka class labels for the training set)
    :param featureset: An instance of a featureset (such as
       Basic9Extractor())
    :rtype: An OrderedDict of the form {K : V}, with K being the feature
         name
    and V being its importance. This dictionary will be sorted by
       importance.
    """

    # FIXME: Use an RBF SVC to rank features. It is likely that the "
       importance"
    # rankings derived from a LinearSVC are similar as an RBF kernel SVM
         , but,
    # for safety’s sake, it is best to assume they are not.
    classifier = LinearSVC()
    classifier.fit(X, y)

    ranker = RFE(classifier, 1, step=1)
    ranker = ranker.fit(X, y)

    # Get the names of the feature columns.
    # FIXME: Duplicate code from rank_features. Make this its own
       function.
    feat_importance = OrderedDict()
    for index, func in enumerate(featureset.features):
        feat_importance[func] = ranker.ranking_[index]

    return sorted(feat_importance.items(), key=lambda x: x[1])


def ak_means_cluster(X, numAuthors):
    """Given a set of feature values, cluster them into k groups. If,
       after
    convergence, there are less than 3 points in any given cluster,
       recurse with
    ak_means_cluster(featureVec, numAuthors - 1).

    :param X: Values for a given feature across a set of authors.
    :rtype: A tuple containing (a trained k-means cluster, numAuthors)
    """
    if numAuthors < 1:
        raise ValueError("ak-means initialized with less than 1 cluster
           .")

    km = KMeans(n_clusters=numAuthors, init='k-means++', n_jobs=-1)
    km.fit(X)

    # Check number of features found in each cluster. Restart with k - 1
        if
    # there are less than 3 members of any cluster.
    # First, we count the number of members in each cluster:
    labelTable = {}
    for label in km.labels_:
        if label in labelTable:
            labelTable[label] += 1
        else:
            labelTable[label] = 1

    # Now we check if any clusters have less than three members:
    for label in labelTable.keys():
        if labelTable[label] < 3:
            print("Reinitializing k means with ", numAuthors - 1, "
               clusters.")
            return ak_means_cluster(X, numAuthors - 1)
    return (km, numAuthors)
```

adversarial.py:

```
"""Module containing functions related to defeating stylometry.
"""
from stylproj.feat_select import ak_means_cluster
import numpy as np
import random
import sys


def compute_target_vals(docfeatures, X, classifier, featureSet,
    numAuthors):
    """Given a set of feature vectors and a trained classifier,
       determine what
    each feature must be changed to change the classifier's label of the
         set of
    feature vectors.

    :param docfeatures: The feature vector of the document we are trying
         to
    anonymize.
    :param X: A complete feature set composed of the feature vectors
       from
    other_user_documents and other_author_documents.
    :param classifier: A classifier trained on X.
    :param featureSet: A feature extractor, such as Basic9Extractor().
    :param numAuthors: The number of authors in X.
    :rtype: A list of target vectors for each feature.
    """
    # TODO: Split this function up; it's very monolithic
    # Cluster every feature vector; put them all in a list.
    print("Clustering features...")
    clusterMeansList = []
    for col in range(X.shape[1]):
        # Get every sample of a given feature; cluster into <=
            numAuthors
        # groups.
        feature = np.asmatrix(X[:, col]).transpose()
        print("Clustering the following samples:")
        print(feature)
        ak = ak_means_cluster(feature, numAuthors)

        # Put all of the clusters from this feature into a table of the
            form
        # {cluster : members_of_cluster}
        clusters = {k: [] for k in range(ak[1])}
        print("Cluster labels: ", ak[0].labels_)
        for idx, label in enumerate(ak[0].labels_):
            clusters[label].append(feature[idx])

        # Compute the mean of every cluster
        clusterMeans = {k: [] for k in range(ak[1])}
        for cluster in clusters:
            total = 0
            print(clusters[cluster])
            for num in clusters[cluster]:
                total += num
            mean = total / len(clusters[cluster])
            clusterMeans[cluster].append(mean)
        clusterMeansList.append(clusterMeans)

    # Find a target cluster that confuses the classifier.
    iterations = 0
    configuration = generate_ran_target_cluster(clusterMeansList)
    print("Initial target configuration: ", configuration)
    # Keep generating target clusters until we find one that works.
    while ((classifier.predict_proba(configuration)[0] is 'user') or (
            authorship_below_random_chance(configuration, classifier,
               numAuthors) is False)):
        configuration = generate_ran_target_cluster(clusterMeansList)
        iterations += 1
        # We have found a configuration that confounds the classifier.
        print("Found target cluster in ", iterations, " iterations.")
        return configuration
    return configuration


def generate_ran_target_cluster(clusterMeansList):
    # FIXME: Right now, we try random targets from list of potential
       clusters
    # until we find a working configuration. That's dumb. Implement a
       less naive
    # search for a target cluster. Trying to "fuzz" the author's most
       important
    # features incrementally is likely the best way to go.
    configuration = []
    print("generate_target_clusters on: ", clusterMeansList,
          " of len ", len(clusterMeansList))
    for feature in clusterMeansList:
        configuration.append(random.choice(list(feature.values())))
    print("Returning target configuration of len: ", len(configuration))

    # The configuration is populated with lists of lists of matrix
       objects of
    # [[numbers]]. We don't want that. This ugly hack turns the matrices
        into regular
    # floats.
    # FIXME: Find root cause of the matrix issue.
    fixed = []
    for matrix in configuration:
        fixed.append(matrix[0][0].item(0))
    return fixed


def generate_sane_target_cluster(clusterMeansList, docFeatures):
    """Find the closest possible feature configuration that will fool
       the
    classifier.
    :param clusterMeansList: A list of potential target values.
    :param docFeatures: The document_to_anonymize's features.
    :configuration: A non-working
    """
    pass


def authorship_below_random_chance(X, classifier, numAuthors):
    """See if the user's document features fool the classifier
    """
    randomChance = 1 / numAuthors
    authorProb = classifier.predict_proba(X)
    print("User probability: ", authorProb[0][0])
    print("numAuthors: ", numAuthors)
    print("Highest probability in array: ", np.amax(authorProb))
    if authorProb[0][0] <= randomChance:
        return True
    else:
        return False
```

dochandler.py:

```
import numpy as np
import glob
import os
# Import feature extractors:
# Import classifiers:


class DocumentExtractor:

    """A class that performs feature extraction on a given set of
       documents.

    :param feat_extractor: An instance of a feature extractor (e.g. the
       Basic-9
    feature set, or writeprints. Any class implementing
       FeatureSetExtractor
    should work.)
    :param docs: An arbitrary number of documents (strings of text).

    Example:
    >>>from stylproj.featuresets.basic9 import Basic9Extractor
    >>>d = DocumentExtractor(Basic9Extractor(), ["text of document 1", "
       text of
    document 2"])
    >>>d.docExtract()
    """

    def __init__(self, feat_extractor, docs):
        self.documents = docs
        self.featureSet = feat_extractor

        if self.documents is None:
            raise TypeError

    def docExtract(self):
        """Extract features from each document. Return it as a matrix of
            (number of
        docs) by (number of features)."""

        self.fv = []
        for doc in self.documents:
            self.fv.append(self.featureSet.extract(doc))

        # Convert to a numpy matrix.
        return np.array(np.asmatrix(self.fv))
        # return self.fv
```

langtools.py:

```
# Miscellaneous natural language programming functions.

import numpy as np
import re

from curses.ascii import isdigit
from nltk.corpus import cmudict

d = cmudict.dict()


def tokenize(text):
    """Tokenizes a given block of text. We use this rather than NLTK's
    tokenizers for more control over the tokenization process. See
    /doc/tokenization for details.
    """
    # 1. Replace hyphens and newlines with spaces

    noNewls = text.replace('\n', ' ')
    noHyphens = noNewls.replace('-', ' ')

    # 2. Split all words by spaces.
    noSpaces = noHyphens.split(' ')

    # 3. Remove all punctuation.
    # TODO: Deal with apostrophes better. We don't want to strip them
       from
    # contractions.
    punctuation = "[\.,!?;,:\"\'()\[\]\{\}   ]"
    noPunc = []
    for word in noSpaces:
        noPunc.append(re.sub(punctuation, '', word))
    return noPunc


def syllableCount(word):
    """Return the ESTIMATED number of syllables in a given word. Returns
        none if
    the word is not inthe dictionary. Be warned that there is no
       deterministic
    way to count syllables, and that some words with multiple
       pronunciations
    have ambiguous numbers of syllables. We cope with this by returning
       the
    first syllable count that we find.
    """
    if word.lower() not in d:
        # We couldn't find the word in the dictionary. Use a naive
           syllable
        # approximation algorithm.
        # TODO: Naive syllable approximation
        return None
    else:
        return nsyl(word.lower())


def nsyl(word):
    """Return the minimum syllable count."""
    # TODO: Near-indecipherable one-liner from stackoverflow; should
       probably
    # replace this.
    syllables = [len(list(y for y in x if y[-1].isdigit()))
                 for x in d[word.lower()]]
    return syllables[0]
```

controller.py:

```
"""This module is the link between the frontend and the backend of
   stylproj.
"""

from sklearn import svm, preprocessing
from scipy.spatial import distance
from itertools import combinations
# gui imports
from PyQt5.QtWidgets import QApplication, QMainWindow, QStyleFactory
from stylproj.gui.stylproj_frontend import StylProj

# backend imports
from stylproj.dochandler import DocumentExtractor
from stylproj.featuresets.basic9 import Basic9Extractor
from stylproj.feat_select import rank_features_rfe
from stylproj.adversarial import compute_target_vals

import codecs
import glob
import logging
import numpy as np
import os
import random
import stylproj
import sys
import timeit

# Instance of the frontend
window = None
# Target feature values
targets = None
# Feature set
featset = None

# Cosine similarity threshold
t = None
# Cosine similarity between document and mean
docCosSim = None
# User documents means:
userDocsMeans = None
# Boolean determining whether cosine similarity is below threshold
similar = True

# TODO: move to dochandler


def load_document(docpath):
    # TODO: Support docx and odf. Should be easy; python has internal
       libraries
    # for this.
    """Load a document into a string.
    :param docpath: The path of the document to load.
    :rtype: A string.
    """
    document = ""
    with codecs.open(docpath, "r", encoding='utf8', errors='replace') as
        docString:
        document = docString.read()
    return document

# TODO: move to dochandler


def _get_random_doc_paths(directory, numAuthors):
    """Get some random documents (and their labels) from our pre-
       supplied corpus.
    :rtype: A tuple of (document paths, labels)
    """
    paths = []
    labels = []
    authorsToLoad = random.sample(os.listdir(directory), numAuthors)
    for auth in authorsToLoad:
        # oh my glob
        docpaths = glob.glob(os.path.join(directory, auth, "*"))
        print(docpaths)
        paths.extend(docpaths)
        for _ in range(len(docpaths)):
            labels.append(auth)
            print(auth)

    print(list(zip(paths, labels)))
    return list(zip(paths, labels))


def train_on_docs(pathToAnonymize, otherUserDocPaths,
   otherAuthorDocPaths):
    """Load and classify all documents referenced by the given paths.
    :rtype: A classifier trained on the user's documents and a random
       subset
    of our corpus.
    """
    document_to_anonymize = load_document(pathToAnonymize)
    other_user_docs = []
    other_author_docs = []
    # Load other documents by the user.
    for path in otherUserDocPaths:
        other_user_docs.append(load_document(path))

    # Load documents by some other authors
    for path in otherAuthorDocPaths:
        other_author_docs.append(load_document(path[0]))

    # Load labels by the randomly selected authors:
    otherAuthorLabels = []
    for pair in other_author_paths:
        otherAuthorLabels.append(pair[1])
    # Extract features from all documents using the Basic-9 Feature Set.
    stylproj.controller.featset = Basic9Extractor()

    stylproj.controller.featlabels = []
    for index, func in enumerate(featset.features):
        stylproj.controller.featlabels.append(func)

    # TODO: Make DocumentExtractor work properly with one document
    docToList = []
    docToList.append(document_to_anonymize)
    userDocFeatures = DocumentExtractor(featset, docToList).docExtract()
    print("User doc features: ", userDocFeatures)
    stylproj.controller.to_anonymize_features = userDocFeatures
    # Features from other documents by the user (excludes
       documentToAnonymize)
    userOtherFeatures = DocumentExtractor(
        featset, other_user_docs).docExtract()
    print("User other features: ", userOtherFeatures)
    # Features from documents by other authors.
    otherAuthorFeatures = DocumentExtractor(
        featset, other_author_docs).docExtract()
    print("Other author features: ", otherAuthorFeatures)
    # Features from documents by other authors AND the user.
    userAndOtherFeatures = np.vstack((userOtherFeatures,
       otherAuthorFeatures))

    # Label all of our user’s other documents as ’user’.
    userLabels = []
    for _ in otherUserDocPaths:
        userLabels.append(’user’)

    print("User other features type: ", type(userOtherFeatures))
    print("User other features contents: ", userOtherFeatures)
    # Compute cosine similarity for every user sample document
    delta_array = np.empty(0)
    for x, y in combinations(userOtherFeatures, 2):
        delta_array = np.hstack((delta_array, distance.cosine(x, y)))

    # Compute cosine similarity threshold
    stylproj.controller.t = delta_array.mean() + delta_array.std()

    # Set threshold for verifying authorship via cosine similarity.
    stylproj.controller.t = delta_array.mean() + delta_array.std()
    initCosineSim = distance.cosine(
        np.asmatrix(
            userOtherFeatures.mean(
                axis=0)),
        np.array(userDocFeatures))
    stylproj.controller.userDocsMeans = np.asmatrix(
        userOtherFeatures.mean(axis=0))
    print("Delta array: ", delta_array)
    print("Delta array threshold: ", stylproj.controller.t)
    print("userOtherFeatures.mean(): ", np.asmatrix(
        userOtherFeatures.mean(axis=0)))
    print("np.array(userDocFeatures):", np.array(userDocFeatures))
    print("Initial cosine similarity between doc and means: ",
       initCosineSim)
    # Basic sanity check to make sure cosine threshold correctly
       identifies
    # authorship of user’s document.
    print("Cosine similarity below threshold? ", str(
        initCosineSim < stylproj.controller.t))

    # Combine documents and labels. This creates the training set.
    X = np.vstack((userOtherFeatures, otherAuthorFeatures))
    y = []
    y.extend(userLabels)
    y.extend(otherAuthorLabels)
```

```
    print("Training labels: ", y)

    # Instantiate classifier; train and predict on scaled data.
    scaler = preprocessing.StandardScaler().fit(X)
    clf = svm.SVC(probability=True, kernel='linear', C=1.0, class_weight
       ='auto')
    clf.fit(scaler.transform(X), y)
    print("Predicted author of doc: " +
          str(clf.predict(scaler.transform(userDocFeatures))))
    print("Certainty: ", clf.predict_proba(scaler.transform(
       userDocFeatures)))
    print("Classifier internal label rep: ", clf.classes_)

    # Get feature ranks
    stylproj.controller.feature_ranks = rank_features_rfe(
        scaler.transform(X), y, featset)
    print(str(feature_ranks))

    # Get target values for features.
    authors = stylproj.controller.numAuthors
    stylproj.controller.targets = stylproj.adversarial.
       compute_target_vals(
        userDocFeatures,
        X,
        clf,
        featset,
        numAuthors + 1
    )

    # Tell the frontend we're done computing on the input it gave us.
    window.update_stats()
    return (clf, scaler)


def validateInput():
    # Make sure the user didn't accidentally put document_to_anonymize
       in the
    # training set
    for doc in other_user_documents_paths:
        if document_to_anonymize_path == doc:
            return False
        else:
            return True


def readyToClassify():
    """ The frontend calls this after it has given the controller all of
    the requisite input documents.
    """
    stylproj.controller.trained_classifier = train_on_docs(
        document_to_anonymize_path,
        other_user_documents_paths,
        other_author_paths)


def checkAnonymity(text):
    """Check if the user has properly anonymized their document.
    :param: The current state of the user's document.
    :rtype: 0 if the classifier identifies the user as the most likely
       author;
    1 if the user is not the most likely author but there is an above
       random
    chance that he or she IS the author; 2 if the author is anonymous.
    """
    randomChance = 1 / (numAuthors + 1)
    # Extract features from the text
    docToList = []
    docToList.append(text)
    extractor = DocumentExtractor(featset, docToList)
    extr = extractor.docExtract()
    print("Current doc features: ", extr)
    print("Scaled doc features: ", trained_classifier[1].transform(extr)
       )
    # Get the probabilities for every label
    probas = trained_classifier[0].predict_proba(
        trained_classifier[1].transform(extr))[0]
    # Get the probability of the user label
    index = (trained_classifier[0].classes_).tolist().index('user')
    proba = probas[index]
    prediction = trained_classifier[0].predict(
        trained_classifier[1].transform(extr))[0]
    print("Probability: ", proba)
    print("Random chance: ", randomChance)
    print(trained_classifier[0].predict_proba(
        trained_classifier[1].transform(extr)))
    print("Current prediction: ", prediction)
    print("Probabilities:", probas)
    print("Highset prob: ", max(probas))
    print("Prediction type: ", type(prediction))
    print("Labels: ", trained_classifier[0].classes_)

    # Compute updated cosine similarity
    sim = distance.cosine(stylproj.controller.userDocsMeans, extr)
    stylproj.controller.similar = sim < stylproj.controller.t
    print("New cosine similarity: ", stylproj.controller.similar)
    print("New similarity below threshold? ", str(stylproj.controller.
       similar))

    if (np.isclose(probas[index], max(probas)) and
            (stylproj.controller.similar)):
        print("Predicted USER")
        return 0
    if (proba > randomChance) and stylproj.controller.similar:
        return 1
    else:
        return 2


def startGUI():
    app = QApplication(sys.argv)
    stylproj.controller.window = StylProj()
    stylproj.controller.window.show()
    sys.exit(app.exec_())

# File paths
document_to_anonymize_path = ''
to_anonymize_features = []
other_user_documents_paths = []

# Get the paths of documents from a set of random authors.
numAuthors = 13
drexel_dataset_path = os.path.join('datasets', 'drexel_1')
other_author_paths = _get_random_doc_paths(drexel_dataset_path,
   numAuthors)

# Training data
document_to_anonymize = ''
trained_classifier = None
feature_ranks = []
feat_labels = []
```

langtools.py:

```
# Miscellaneous natural language programming functions.

import numpy as np
import re

from curses.ascii import isdigit
from nltk.corpus import cmudict

d = cmudict.dict()


def tokenize(text):
    """Tokenizes a given block of text. We use this rather than NLTK's
    tokenizers for more control over the tokenization process. See
    /doc/tokenization for details.
    """
    # 1. Replace hyphens and newlines with spaces

    noNewls = text.replace('\n', ' ')
    noHyphens = noNewls.replace('-', ' ')

    # 2. Split all words by spaces.
    noSpaces = noHyphens.split(' ')

    # 3. Remove all punctuation.
    # TODO: Deal with apostrophes better. We don’t want to strip them
       from
    # contractions.
    punctuation = "[\.,!?;,:\"\’()\[\]\{\}   ]"
    noPunc = []
    for word in noSpaces:
        noPunc.append(re.sub(punctuation, ’’, word))
    return noPunc


def syllableCount(word):
    """Return the ESTIMATED number of syllables in a given word. Returns
        none if
    the word is not inthe dictionary. Be warned that there is no
       deterministic
    way to count syllables, and that some words with multiple
       pronunciations
    have ambiguous numbers of syllables. We cope with this by returning
       the
    first syllable count that we find.
    """
    if word.lower() not in d:
        # We couldn’t find the word in the dictionary. Use a naive
           syllable
        # approximation algorithm.
        # TODO: Naive syllable approximation
        return None
    else:
        return nsyl(word.lower())


def nsyl(word):
    """Return the minimum syllable count."""
    # TODO: Near-indecipherable one-liner from stackoverflow; should
       probably
    # replace this.
    syllables = [len(list(y for y in x if y[-1].isdigit()))
                 for x in d[word.lower()]]
    return syllables[0]
```

gui/stylproj_frontend.py:

```
from PyQt5.QtCore import pyqtSlot, pyqtSignal
from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog
from PyQt5.QtWidgets import QTableWidgetItem, QHeaderView
from PyQt5 import QtGui
from stylproj.gui.stylproj_auto import Ui_Unstyle
import stylproj.controller


class StylProj(QMainWindow):
    def __init__(self, parent=None):
        # Initialized the generated interface code.
        super(StylProj, self).__init__(parent)
        self.ui = Ui_Unstyle()
        self.ui.setupUi(self)
        self.featureRows = {}
        self.setWindowTitle("Unstyle")
        # Signal connections
        self.ui.stackedNext.clicked.connect(self.stackNext_clicked)
        self.ui.browseYourDoc.clicked.connect(self.browseYourDoc_clicked
           )
        self.ui.browseYourDocs.clicked.connect(self.
           browseYourDocs_clicked)
        self.ui.deleteYourDocs.clicked.connect(self.
           deleteYourDocs_clicked)
        self.ui.textEdit.textChanged.connect(self.refreshAnonymity)
        self.ui.rankTable.selectionModel().selectionChanged.connect(
            self.row_highlighted)
        self.ui.saveDoc.clicked.connect(self.saveDoc_clicked)

    def getFeatureDesc(self, functionName):
        """Translate feature extractor names into something that the end
            user
        can understand.
        :param functionName: A feature extracting function.
        :returns: A typle containing ("Feature Name", "Description of
           feature").
        """
        names = {
            "letterSpace": (
                "Letter Space",
                ("The total number of letters appearing in your "
                 "document.")),
            "gunningFog": (
                "Gunning-Fog readability",
                ("A function related to "
                 "the ratio of words/sentences and complex word/total
                    words.")),
            "avgSyllablesPerWord": (
                "Average syllables per word",
                ("The total "
                 "number of syllables/the total number of words.")),
            "unique_words": (
                "Unique words",
                ("The number of words that appear "
                 "only once in your document.")),
            "sentenceCount": (
                "Sentence count",
                ("The number of sentences in your document.")),
            "characterSpace": (
                "Character space",
                ("The total number of "
                "characters (letters and numbers) appearing in your
                    document.")),
            "avgSentenceLength": (
                "Average sentence length",
                ("The average "
                 "length of sentences in your document.")),
            "complexity": (
                "Complexity",
                ("The ratio of unique words to total"
                 "words in your document.")),
            "fleschReadingEase": (
                "Flesch readability",
                ("A function related to"
                 " the ratio of words/sentences and syllables/words."))}
        return names[functionName]

    # stackedWidget buttons
    def stackNext_clicked(self):
        # Go to the next screen.
        self.ui.stackedWidget.setCurrentIndex(1)
        # Tell the controller to train its classifier.
        stylproj.controller.readyToClassify()

    def browseYourDoc_clicked(self):
        filename = QFileDialog.getOpenFileName()
        stylproj.controller.document_to_anonymize_path = filename[0]
        self.ui.yourdoc.setText(filename[0])
        stylproj.controller.document_to_anonymize = stylproj.controller.
            load_document(
             filename[0])
        # Show the text of the document in the text editor and enable it
            .
        self.ui.textEdit.setText(stylproj.controller.
            document_to_anonymize)
        self.ui.textEdit.setEnabled(True)

    def browseYourDocs_clicked(self):
        filenames = QFileDialog.getOpenFileNames()
        if filenames is not '':
            for path in filenames[0]:
                stylproj.controller.other_user_documents_paths.append(
                    path)
                self.ui.otherdocslist.addItem(path)

    def deleteYourDocs_clicked(self):
        selected = self.ui.otherdocslist.currentItem()
        # Make sure the user selected a document before trying to delete
        # anything
        if selected is not None:
            row = self.ui.otherdocslist.currentRow()
            stylproj.controller.other_user_documents_paths.remove(
                selected.text())
            self.ui.otherdocslist.takeItem(row)
        else:
            pass

    def saveDoc_clicked(self):
        """Save the current state of the text editor to a file defined
           by the
        user.
        """
        # Open a save dialog
        filename = QFileDialog.getSaveFileName()
        if filename is not None:
            with open(filename, 'w+') as file:
                file.write(str(textEdit.toPlainText()))

    # TODO: Rather than check anonymity every time the user changes the
       text,
    # have a separate thread check every 5 or 10 seconds. Otherwise, we'
       re going
    # to be constantly locking up the interface when we use large
       featuresets.
    def refreshAnonymity(self):
        """Called whenever the user changes the text editor.
        """
        # Make sure we've trained the classifier before trying to do any
        # predictions.
        if stylproj.controller.trained_classifier is None:
            return 0

        anonymity = stylproj.controller.checkAnonymity(
            self.ui.textEdit.toPlainText())
        if anonymity is 0:
            self.ui.anonIcon.setPixmap(QtGui.QPixmap(":/icons/img/x.png"
                ))
            self.ui.anonStatus.setText(
                ("It is still possible to identify you as the "
                 "author. Continue changing your document."))
        if anonymity is 1:
            self.ui.anonIcon.setPixmap(QtGui.QPixmap(":/icons/img/w.png"
                ))
            self.ui.anonStatus.setText(
                ("Although you are not the most likely author,"
                 " there is a statistically significant chance"
                 " that you wrote the document. Continue"
                 " changing your document."))
        if anonymity is 2:
            self.ui.anonIcon.setPixmap(QtGui.QPixmap(":/icons/img/check.
                png"))
            self.ui.anonStatus.setText(
                ("Congratulations! It appears that your"
                 " document is no longer associated with your"
                 " identity."))
    def row_highlighted(self, _, __):
        """Every time someone selects a row from the table, we update
            our
        description box with the description of the feature.
        """
        selected = self.ui.rankTable.selectionModel().selectedRows()[0].
            row()
        featureHighlighted = self.featureRows[selected]
        # Display the description of the highlighted feature
        self.ui.featureDescription.setText(
            self.getFeatureDesc(featureHighlighted)[1])

    # Controller messages
    def update_stats(self):
        self.refreshAnonymity()
        # Set up rank table dimensions
        self.ui.rankTable.setRowCount(len(stylproj.controller.
            feature_ranks))
        # Name the headers of the table
        headers = "Text Features", "Target", "Initial"
        self.ui.rankTable.setHorizontalHeaderLabels(headers)
        headerObj = self.ui.rankTable.horizontalHeader()
        headerObj.setSectionResizeMode(0, QHeaderView.ResizeToContents)

        tableHeight = (len(stylproj.controller.feature_ranks))
        # XXX: Sorting should be handled in the table, not in the
        # rank_features methods. This will allow us to fix this
            embarrassingly
        # overcomplicated code.

        # Fill in the feature column
        for idx, pair in enumerate(stylproj.controller.feature_ranks):
            currItem = self.ui.rankTable.item(idx, 0)
            # If we are setting up the table for the first time,
                currItem will
            # not exist.
            if currItem is None:
                currItem = QTableWidgetItem(1)
                currItem.setText(self.getFeatureDesc(pair[0])[0])
                self.ui.rankTable.setItem(idx, 0, currItem)
            else:
                currItem.setText(
                    self.getFeatureDesc(feature_ranks[pair[0]])[0])

        # Initialize target and initial columns
        for idx, target in enumerate(stylproj.controller.targets):
            currItem = self.ui.rankTable.item(idx, 1)
            if currItem is None:
                currItem = QTableWidgetItem(1)
                currItem.setText(str(target))
                self.ui.rankTable.setItem(idx, 1, currItem)
                currItem2 = QTableWidgetItem(1)
                self.ui.rankTable.setItem(idx, 2, currItem2)

        # Populate target and current val columns
        # Track feature table locations
        labelsBeforeSorting = stylproj.controller.featlabels
        for idx, label in enumerate(labelsBeforeSorting):
            for idx2, item in enumerate(range(tableHeight)):
                currItem = self.ui.rankTable.item(item, 0)
                if self.getFeatureDesc(label)[0] == currItem.text():
                    self.featureRows[idx2] = label
                    print(label, " ", currItem.text(), " ", item)
                    currItem = self.ui.rankTable.item(item, 1)
                    currItem.setText(str(stylproj.controller.targets[idx
                        ]))
                    currItem = self.ui.rankTable.item(item, 2)
                    currItem.setText(
                        str(stylproj.controller.to_anonymize_features
                            [0][idx]))
```

scripts/filelistfromdir.py:

```
# Output a text file containing a list of all of the textfile paths in a
    given
# directory followed by their class label, delimited by a colon.
#
# Example output:
# ./datasets/C50train/AaronPressman/106247newsML.txt:AaronPressman
#
# USAGE: python filelistfromdir [WRITEOUTFILE] [DIRECTORY1] [DIRECTORY
    2] ...

import os
import sys

files = []
for directory in sys.argv[2:]:
    for i in os.listdir(directory):
        if i.endswith(".txt"):
            files.append(directory + "/" + i + ":" +
                         os.path.split(os.path.dirname(directory))[1] +
                            "\n")

with open(sys.argv[1], "w") as filelist:
    filelist.writelines(files)

# for path in files:
#    print(repr(path))
```

scripts/testcorpus.py:

```
# Run classifyTest.py on every author pair in the Reuters dataset.

import os
import subprocess
import sys

testDir = ("./datasets/C50test/")
authorPairs = set()
iterations = 0
for author1 in os.listdir("./datasets/C50train/"):
    iterations += 1
    for author2 in os.listdir("./datasets/C50train/"):
        # track our author pairs so we don't benchmark them twice
        authorPairs.add((author2, author1))

        first = "./datasets/C50train/" + author1
        second = "./datasets/C50train/" + author2

        firsttest = "./datasets/C50test/" + author1
        secondtest = "./datasets/C50test/" + author2

        if (first != second) and (author1, author2) not in authorPairs:
            subprocess.call(['python3', './scripts/filelistfromdir.py',
                             'trainingDocs.txt', first + '/', second + '
                                 /'])
            subprocess.call(['python3',
                             './scripts/filelistfromdir.py',
                             'testDocs.txt',
                             firsttest + '/',
                             secondtest + '/'])
            print(first)
            print(second)
            subprocess.call(['python3', 'classifyTest.py'])
```

scripts/parseaccuracy.py:

```
# Load a testcorpus file. Print out some statistics.

import sys

lines = []
with open(sys.argv[1], "r") as datafile:
    lines = datafile.readlines()

# The sum of all accuracy measurements
accuracySum = 0
# The accuracy of the classifier across the entire dataset.
avgAccuracy = 0
# Lowest percentage found in document
lowest = 100
# Highest percentage found in document
highest = 0

# list of all accuacy measurements
accuracyList = []
for line in lines:
    if '%' in line:
        # Parse the accuracy percentage
        accuracy = float(line[:2])
        accuracyList.append(accuracy)

for num in accuracyList:
    accuracySum += num

for num in accuracyList:
    if num < lowest:
        lowest = num
    if highest < num:
        highest = num

avgaccuracy = accuracySum / len(accuracyList)

print("Average accuracy: " + str(avgaccuracy))
print("Highest accuracy: " + str(highest))
print("Lowest accuracy: " + str(lowest))
print("Number of samples: " + str(len(accuracyList)))
```

tests/classifyTest.py:

```
# Load, train, and classify documents.
# This serves as a prototype for experimenting with document
   classification. This
# module should NOT be referred to by any other classes.
from stylproj.dochandler import DocumentExtractor
from stylproj.featuresets.basic9 import Basic9Extractor
from sklearn import svm
from sklearn import preprocessing
from sklearn import cross_validation
from sklearn.grid_search import GridSearchCV
from sklearn.feature_selection import chi2, SelectKBest
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.cross_validation import train_test_split

import codecs
import numpy as np
import matplotlib.pyplot as plt


# load our list of training document paths (original author)
labels = []
paths = []
with open("trainingDocs.txt", "r") as trainingDocs:
    for line in trainingDocs.readlines():
        paths.append(line.split(':')[0])
        labels.append(line.split(':')[1].replace('\n', ''))
        #print("Training set with label: " + line)

# load our list of test document paths
```

```
testlabels = []
testpaths = []
with open("testDocs.txt", "r") as testDocs:
    for line in testDocs.readlines():
        testpaths.append(line.split(':')[0])
        testlabels.append(line.split(':')[1].replace('\n', ''))

# load each file in our list of paths
trainingStrings = []
for path in paths:
    with codecs.open(path, "r", encoding='utf8', errors='replace') as
       docString:
        document = docString.read()
        trainingStrings.append(document)

# load each of our testfile path list
testStrings = []
for path in testpaths:
    with codecs.open(path, "r", encoding='utf8', errors='replace') as
       docString:
        document = docString.read()
        testStrings.append(document)

# extract features from each document
print("Extracting features...")
extractedFeats = DocumentExtractor(Basic9Extractor(), trainingStrings)
extracted = extractedFeats.docExtract()

# extract features from our test documents:
testFeats = DocumentExtractor(Basic9Extractor(), testStrings)
testvecs = testFeats.docExtract()

# scale our feature vectors to make them suitable for SVM input
print("Scaling feature vectors...")
extracted = preprocessing.scale(extracted)
# instantiate classifier and train it
print("Instantiating classifier...")
clf = svm.SVC(probability=True, kernel='linear', class_weight='auto')
print("Fitting dataset to classifier...")
clf.fit(extracted, labels)
# do some predictions, again with test vectors scaled
print("Computing predictions...")
normalizedpredic = clf.predict(preprocessing.scale(testvecs))

# compute number of authors
authorSet = set()
for label in labels:
    authorSet.add(label)

print("Comparing authors:")
for n in authorSet:
    print(n)
```

```
#print(str(clf.score(preprocessing.scale(testvecs), testlabels) * 100) +
     "% score on this dataset.")

testvecsscaled = preprocessing.scale(testvecs)
# Cross-validation
print("Computing cross validation...")
cvIterations = 7
scores = cross_validation.cross_val_score(clf, extracted, labels,
                                          cv=cvIterations)
print("Accuracy by " + str(cvIterations) + "-fold CV: %0.2f (+/- %0.2f)"
     %
       (scores.mean(), scores.std() * 2))


def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.
    Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(authorSet))
    plt.xticks(tick_marks, list(authorSet), rotation=45)
    plt.yticks(tick_marks, list(authorSet))
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

y_pred = clf.predict(testvecsscaled)
print("Predictions: ", y_pred)
cm = confusion_matrix(labels, y_pred)
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(cm)
print(cm)

report = print(classification_report(
    labels, y_pred, target_names=[str(x) for x in range(13)]))
plt.show()
```

D.2 Generated code

Code in this section is generated from PyQt or Qt Designer. The resource file `resources_rc.py` has been ommitted, as it contains binary image data.

`gui/stylproj_auto.py`:

```
from PyQt5.QtCore import pyqtSlot, pyqtSignal
from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog
from PyQt5.QtWidgets import QTableWidgetItem, QHeaderView
from PyQt5 import QtGui
from stylproj.gui.stylproj_auto import Ui_Unstyle
```

```
import stylproj.controller


class StylProj(QMainWindow):

    def __init__(self, parent=None):
        # Initialized the generated interface code.
        super(StylProj, self).__init__(parent)
        self.ui = Ui_Unstyle()
        self.ui.setupUi(self)
        self.featureRows = {}
        self.setWindowTitle("Unstyle")
        # Signal connections
        self.ui.stackedNext.clicked.connect(self.stackNext_clicked)
        self.ui.browseYourDoc.clicked.connect(self.browseYourDoc_clicked
            )
        self.ui.browseYourDocs.clicked.connect(self.
            browseYourDocs_clicked)
        self.ui.deleteYourDocs.clicked.connect(self.
            deleteYourDocs_clicked)
        self.ui.textEdit.textChanged.connect(self.refreshAnonymity)
        self.ui.rankTable.selectionModel().selectionChanged.connect(
             self.row_highlighted)
        self.ui.saveDoc.clicked.connect(self.saveDoc_clicked)

    def getFeatureDesc(self, functionName):
        """Translate feature extractor names into something that the end
             user
        can understand.
        :param functionName: A feature extracting function.
        :returns: A typle containing ("Feature Name", "Description of
            feature").
        """
        names = {
            "letterSpace": (
                "Letter Space",
                ("The total number of letters appearing in your "
                 "document.")),
            "gunningFog": (
                "Gunning-Fog readability",
                ("A function related to "
                 "the ratio of words/sentences and complex word/total
                     words.")),
            "avgSyllablesPerWord": (
                "Average syllables per word",
                ("The total "
                 "number of syllables/the total number of words.")),
            "unique_words": (
                "Unique words",
                ("The number of words that appear "
                 "only once in your document.")),
            "sentenceCount": (
                "Sentence count",
                ("The number of sentences in your document.")),
            "characterSpace": (
                "Character space",
                ("The total number of "
                 "characters (letters and numbers) appearing in your
                    document.")),
            "avgSentenceLength": (
                "Average sentence length",
                ("The average "
                 "length of sentences in your document.")),
            "complexity": (
                "Complexity",
                ("The ratio of unique words to total"
                 "words in your document.")),
            "fleschReadingEase": (
                "Flesch readability",
                ("A function related to"
                 " the ratio of words/sentences and syllables/words."))}
        return names[functionName]

    # stackedWidget buttons
    def stackNext_clicked(self):
        # Go to the next screen.
        self.ui.stackedWidget.setCurrentIndex(1)
        # Tell the controller to train its classifier.
        stylproj.controller.readyToClassify()

    def browseYourDoc_clicked(self):
        filename = QFileDialog.getOpenFileName()
        stylproj.controller.document_to_anonymize_path = filename[0]
        self.ui.yourdoc.setText(filename[0])
        stylproj.controller.document_to_anonymize = stylproj.controller.
            load_document(
             filename[0])
        # Show the text of the document in the text editor and enable it
            .
        self.ui.textEdit.setText(stylproj.controller.
            document_to_anonymize)
        self.ui.textEdit.setEnabled(True)

    def browseYourDocs_clicked(self):
        filenames = QFileDialog.getOpenFileNames()
        if filenames is not '':
            for path in filenames[0]:
                stylproj.controller.other_user_documents_paths.append(
                    path)
                self.ui.otherdocslist.addItem(path)

    def deleteYourDocs_clicked(self):
        selected = self.ui.otherdocslist.currentItem()
        # Make sure the user selected a document before trying to delete
        # anything
        if selected is not None:
            row = self.ui.otherdocslist.currentRow()
            stylproj.controller.other_user_documents_paths.remove(
                selected.text())
            self.ui.otherdocslist.takeItem(row)
        else:
            pass

    def saveDoc_clicked(self):
        """Save the current state of the text editor to a file defined
           by the
        user.
        """
        # Open a save dialog
        filename = QFileDialog.getSaveFileName()
        if filename is not None:
            with open(filename, 'w+') as file:
                file.write(str(textEdit.toPlainText()))

    # TODO: Rather than check anonymity every time the user changes the
       text,
    # have a separate thread check every 5 or 10 seconds. Otherwise, we'
       re going
    # to be constantly locking up the interface when we use large
       featuresets.
    def refreshAnonymity(self):
        """Called whenever the user changes the text editor.
        """
        # Make sure we've trained the classifier before trying to do any
        # predictions.
        if stylproj.controller.trained_classifier is None:
            return 0

        anonymity = stylproj.controller.checkAnonymity(
            self.ui.textEdit.toPlainText())
        if anonymity is 0:
            self.ui.anonIcon.setPixmap(QtGui.QPixmap(":/icons/img/x.png"
                ))
            self.ui.anonStatus.setText(
                ("It is still possible to identify you as the "
                 "author. Continue changing your document."))
        if anonymity is 1:
            self.ui.anonIcon.setPixmap(QtGui.QPixmap(":/icons/img/w.png"
                ))
            self.ui.anonStatus.setText(
                ("Although you are not the most likely author,"
                 " there is a statistically significant chance"
                 " that you wrote the document. Continue"
                 " changing your document."))
        if anonymity is 2:
            self.ui.anonIcon.setPixmap(QtGui.QPixmap(":/icons/img/check.
                png"))
            self.ui.anonStatus.setText(
                ("Congratulations! It appears that your"
                 " document is no longer associated with your"
                 " identity."))

    def row_highlighted(self, _, __):
        """Every time someone selects a row from the table, we update
           our
        description box with the description of the feature.
        """
        selected = self.ui.rankTable.selectionModel().selectedRows()[0].
           row()
        featureHighlighted = self.featureRows[selected]
        # Display the description of the highlighted feature
        self.ui.featureDescription.setText(
            self.getFeatureDesc(featureHighlighted)[1])

    # Controller messages
    def update_stats(self):
        self.refreshAnonymity()
        # Set up rank table dimensions
        self.ui.rankTable.setRowCount(len(stylproj.controller.
           feature_ranks))
        # Name the headers of the table
        headers = "Text Features", "Target", "Initial"
        self.ui.rankTable.setHorizontalHeaderLabels(headers)
        headerObj = self.ui.rankTable.horizontalHeader()
        headerObj.setSectionResizeMode(0, QHeaderView.ResizeToContents)

        tableHeight = (len(stylproj.controller.feature_ranks))
        # XXX: Sorting should be handled in the table, not in the
        # rank_features methods. This will allow us to fix this
           embarrassingly
        # overcomplicated code.

        # Fill in the feature column
        for idx, pair in enumerate(stylproj.controller.feature_ranks):
            currItem = self.ui.rankTable.item(idx, 0)
            # If we are setting up the table for the first time,
               currItem will
            # not exist.
            if currItem is None:
                currItem = QTableWidgetItem(1)
                currItem.setText(self.getFeatureDesc(pair[0])[0])
                self.ui.rankTable.setItem(idx, 0, currItem)
            else:
                currItem.setText(
                    self.getFeatureDesc(feature_ranks[pair[0]])[0])

        # Initialize target and initial columns
        for idx, target in enumerate(stylproj.controller.targets):
            currItem = self.ui.rankTable.item(idx, 1)
            if currItem is None:
                currItem = QTableWidgetItem(1)
                currItem.setText(str(target))
                self.ui.rankTable.setItem(idx, 1, currItem)
                currItem2 = QTableWidgetItem(1)
                self.ui.rankTable.setItem(idx, 2, currItem2)

        # Populate target and current val columns
        # Track feature table locations
        labelsBeforeSorting = stylproj.controller.featlabels
        for idx, label in enumerate(labelsBeforeSorting):
            for idx2, item in enumerate(range(tableHeight)):
                currItem = self.ui.rankTable.item(item, 0)
                if self.getFeatureDesc(label)[0] == currItem.text():
                    self.featureRows[idx2] = label
                    print(label, " ", currItem.text(), " ", item)
                    currItem = self.ui.rankTable.item(item, 1)
                    currItem.setText(str(stylproj.controller.targets[idx
                        ]))
                    currItem = self.ui.rankTable.item(item, 2)
                    currItem.setText(
                        str(stylproj.controller.to_anonymize_features
                            [0][idx]))
```

gui/stylproj.ui:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Unstyle</class>
 <widget class="QMainWindow" name="Unstyle">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1209</width>
    <height>682</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="1">
     <widget class="QFrame" name="frame_2">
      <property name="frameShape">
       <enum>QFrame::StyledPanel</enum>
      </property>
      <property name="frameShadow">
       <enum>QFrame::Raised</enum>
      </property>
      <layout class="QGridLayout" name="gridLayout_6">
```

```
        <item row="0" column="0">
         <widget class="QLabel" name="label_5">
          <property name="text">
           <string>Your document:</string>
          </property>
         </widget>
        </item>
        <item row="1" column="0">
         <widget class="QTextEdit" name="textEdit">
          <property name="enabled">
           <bool>false</bool>
          </property>
          <property name="html">
           <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//
               EN&quot; &quot;http://www.w3.org/TR/REC-html40/strict.dtd&
               quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&
    quot;1&quot; /&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'Ubuntu';
    font-size:11pt; font-weight:400; font-style:normal;&quot;&gt;
&lt;p style=&quot; margin-top:0px; margin-bottom:0px; margin-left:0px;
    margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;
    span style=&quot; font-style:italic;&quot;&gt;Select a document to
    anonymize.&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
          </property>
         </widget>
        </item>
        <item row="2" column="0">
         <widget class="QPushButton" name="saveDoc">
          <property name="text">
           <string>Save As...</string>
          </property>
         </widget>
        </item>
       </layout>
      </widget>
     </item>
     <item row="0" column="0">
      <widget class="QStackedWidget" name="stackedWidget">
       <property name="currentIndex">
        <number>0</number>
       </property>
       <widget class="QWidget" name="stackedWidgetPage1">
        <layout class="QVBoxLayout" name="verticalLayout">
         <item>
          <widget class="QFrame" name="frame">
           <property name="frameShape">
            <enum>QFrame::StyledPanel</enum>
           </property>
           <property name="frameShadow">
            <enum>QFrame::Raised</enum>
           </property>
           <layout class="QGridLayout" name="gridLayout_5">
            <item row="0" column="0">
             <layout class="QGridLayout" name="gridLayout_3">
              <item row="0" column="0">
               <widget class="QLabel" name="label">
                <property name="text">
                 <string>Document to anonymize:</string>
                </property>
               </widget>
              </item>
              <item row="1" column="0">
               <layout class="QGridLayout" name="gridLayout_2">
                <item row="0" column="0">
                 <widget class="QLineEdit" name="yourdoc">
                  <property name="enabled">
                   <bool>true</bool>
                  </property>
                  <property name="toolTip">
                   <string>Click the 'Browse' button to the right to
                      select the document you would like to anonymize.</
                      string>
                  </property>
                  <property name="text">
                   <string/>
                  </property>
                  <property name="readOnly">
                   <bool>true</bool>
                  </property>
                 </widget>
                </item>
                <item row="0" column="1">
                 <widget class="QPushButton" name="browseYourDoc">
                  <property name="text">
                   <string>Browse</string>
                  </property>
                 </widget>
                </item>
               </layout>
              </item>
             </layout>
            </item>
           </layout>
          </widget>
         </item>
         <item>
          <widget class="QFrame" name="frame_4">
           <property name="enabled">
            <bool>true</bool>
           </property>
           <property name="frameShape">
            <enum>QFrame::StyledPanel</enum>
```

```
          </property>
          <property name="frameShadow">
           <enum>QFrame::Raised</enum>
          </property>
          <layout class="QGridLayout" name="gridLayout_4">
           <item row="0" column="0">
            <widget class="QSplitter" name="splitter">
             <property name="orientation">
              <enum>Qt::Vertical</enum>
             </property>
             <widget class="QLabel" name="label_6">
              <property name="text">
               <string>Other documents written by you:</string>
              </property>
             </widget>
             <widget class="QWidget" name="layoutWidget">
              <layout class="QHBoxLayout" name="horizontalLayout">
               <item>
                <widget class="QListWidget" name="otherdocslist">
                 <property name="toolTip">
                  <string>Click the 'browse' button and select other
                      documents you have written. Hint: Holding the Ctrl
                       button allows you to select multiple documents.&
                      lt;br&gt;&lt;br&gt;It is strongly recommended that
                       the combined word count of your documents is at
                      least 6500.</string>
                 </property>
                 <property name="dragDropMode">
                  <enum>QAbstractItemView::DragOnly</enum>
                 </property>
                 <property name="resizeMode">
                  <enum>QListView::Fixed</enum>
                 </property>
                 <property name="layoutMode">
                  <enum>QListView::SinglePass</enum>
                 </property>
                </widget>
               </item>
               <item>
                <layout class="QVBoxLayout" name="verticalLayout_5">
                 <item>
                  <widget class="QPushButton" name="browseYourDocs">
                   <property name="text">
                    <string>Browse</string>
                   </property>
                  </widget>
                 </item>
                 <item>
                  <widget class="QPushButton" name="deleteYourDocs">
                   <property name="text">
                    <string>Delete</string>
                   </property>
                  </widget>
                 </item>
                 <item>
                  <spacer name="verticalSpacer">
                   <property name="orientation">
                    <enum>Qt::Vertical</enum>
                   </property>
                   <property name="sizeHint" stdset="0">
                    <size>
                     <width>20</width>
                     <height>40</height>
                    </size>
                   </property>
                  </spacer>
                 </item>
                </layout>
               </item>
              </layout>
             </widget>
            </widget>
           </item>
          </layout>
         </widget>
        </item>
        <item>
         <widget class="QPushButton" name="stackedNext">
          <property name="font">
           <font>
            <weight>75</weight>
            <bold>true</bold>
           </font>
          </property>
          <property name="text">
           <string>Next</string>
          </property>
         </widget>
        </item>
       </layout>
      </widget>
      <widget class="QWidget" name="stackedWidgetPage2">
       <layout class="QGridLayout" name="gridLayout_7">
        <item row="0" column="0">
         <widget class="QLabel" name="label_3">
          <property name="text">
           <string>Your documents were successfully analyzed. Your top
              writing style traits (in order of importance) are listed
              on the left. Changing document's feature values (listed
              in the 'initial') to target values (listed in the 'target
              ') will result in an anonymous document.</string>
          </property>
          <property name="textFormat">
           <enum>Qt::AutoText</enum>
          </property>
          <property name="wordWrap">
           <bool>true</bool>
          </property>
         </widget>
        </item>
        <item row="2" column="0">
         <layout class="QHBoxLayout" name="horizontalLayout_3"/>
        </item>
        <item row="3" column="0">
         <widget class="QFrame" name="frame_3">
          <property name="frameShape">
           <enum>QFrame::StyledPanel</enum>
          </property>
          <property name="frameShadow">
           <enum>QFrame::Raised</enum>
          </property>
          <layout class="QFormLayout" name="formLayout">
           <item row="0" column="0">
            <widget class="QLabel" name="anonIcon">
             <property name="text">
              <string/>
             </property>
             <property name="pixmap">
              <pixmap resource="resources.qrc">:/icons/img/x.png</pixmap
                 >
             </property>
            </widget>
           </item>
           <item row="0" column="1">
            <widget class="QLabel" name="anonStatus">
             <property name="text">
              <string>It is still possible to identify you as the author
                 . Continue changing your document.</string>
             </property>
             <property name="scaledContents">
              <bool>true</bool>
             </property>
             <property name="wordWrap">
              <bool>true</bool>
             </property>
            </widget>
           </item>
          </layout>
         </widget>
        </item>
        <item row="1" column="0">
         <widget class="QFrame" name="frame_5">
          <property name="frameShape">
           <enum>QFrame::StyledPanel</enum>
          </property>
          <property name="frameShadow">
```

```
           <enum>QFrame::Raised</enum>
          </property>
          <layout class="QVBoxLayout" name="verticalLayout_2">
           <item>
            <widget class="QTableWidget" name="rankTable">
             <property name="selectionMode">
              <enum>QAbstractItemView::SingleSelection</enum>
             </property>
             <property name="selectionBehavior">
              <enum>QAbstractItemView::SelectRows</enum>
             </property>
             <property name="rowCount">
              <number>9</number>
             </property>
             <property name="columnCount">
              <number>3</number>
             </property>
             <attribute name="horizontalHeaderShowSortIndicator" stdset
                ="0">
              <bool>false</bool>
             </attribute>
             <row/>
             <row/>
             <row/>
             <row/>
             <row/>
             <row/>
             <row/>
             <row/>
             <row/>
             <column/>
             <column/>
             <column/>
            </widget>
           </item>
           <item>
            <widget class="QLabel" name="label_2">
             <property name="text">
              <string>Description:</string>
             </property>
            </widget>
           </item>
           <item>
            <widget class="QLabel" name="featureDescription">
             <property name="text">
              <string>Click on a feature to receive a description of the
                  feature.</string>
             </property>
             <property name="wordWrap">
              <bool>true</bool>
             </property>
            </widget>
```

```
         </item>
        </layout>
       </widget>
      </item>
     </layout>
    </widget>
   </widget>
  </item>
 </layout>
</widget>
<widget class="QStatusBar" name="statusbar"/>
<widget class="QMenuBar" name="menuBar">
 <property name="enabled">
  <bool>false</bool>
 </property>
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>1209</width>
   <height>25</height>
  </rect>
 </property>
 <widget class="QMenu" name="menuFile">
  <property name="title">
   <string>File</string>
  </property>
  <addaction name="actionNew_session"/>
  <addaction name="actionLoad_session"/>
  <addaction name="actionSave_session"/>
 </widget>
 <widget class="QMenu" name="menuHelp">
  <property name="title">
   <string>Help</string>
  </property>
  <addaction name="actionAbout"/>
  <addaction name="actionOpen_documentation_in_browser"/>
 </widget>
 <addaction name="menuFile"/>
 <addaction name="menuHelp"/>
</widget>
<action name="actionSave_session">
 <property name="text">
  <string>Save session</string>
 </property>
</action>
<action name="actionLoad_session">
 <property name="text">
  <string>Load session</string>
 </property>
</action>
<action name="actionNew_session">
```

```
   <property name="text">
    <string>New session</string>
   </property>
  </action>
  <action name="actionAbout">
   <property name="text">
    <string>About</string>
   </property>
  </action>
  <action name="actionOpen_documentation_in_browser">
   <property name="text">
    <string>Open documentation in browser</string>
   </property>
  </action>
 </widget>
 <resources>
  <include location="resources.qrc"/>
 </resources>
 <connections/>
</ui>
```

gui/resources.qrc:

```
<RCC>
  <qresource prefix="icons">
    <file>img/check.png</file>
    <file>img/w.png</file>
    <file>img/x.png</file>
  </qresource>
</RCC>
```

Bibliography

[1] How a computer program helped reveal j.k. rowling as author of a cuckoo's calling. URL `http://www.scientificamerican.com/article/how-a-computer-program-helped-show-jk-rowling-write-a-cuckoos-call` 2013.

[2] The Anonymous SEC Whistleblower Award of $14M Is A Game Changer. URL `http://www.forbes.com/sites/walterpavlo/2013/10/03/the-anonymous-sec-whistleblower-award-of-14m-is-a-game-changer/`, 2013.

[3] Ahmed Abbasi and Hsinchun Chen. Writeprints: A stylometric approach to identity-level identification and similarity detection in cyberspace. *ACM Transactions on Information Systems (TOIS)*, 26(2):7, 2008.

[4] Scott Ambler. User interface design tips, techniques, and principles. `http://www.ambysoft.com/essays/userInterfaceDesign.html`.

[5] Shlomo Argamon and Shlomo Levitan. Measuring the usefulness of function words for authorship attribution. In *In Proceedings of the 2005 ACH/ALLC Conference*, 2005.

[6] Regina Barzilay and Lillian Lee. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 16–23, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[7] Mudit Bhargava, Pulkit Mehndiratta, and Krishna Asawa. Stylometric analysis for authorship attribution on twitter. In Vasudha Bhatnagar and Srinath Srinivasa, editors, *Big Data Analytics*, volume 8302 of *Lecture Notes in Computer Science*, pages 37–47. Springer International Publishing, 2013.

[8] Michael Brennan, Sadia Afroz, and Rachel Greenstadt. Adversarial stylometry: Circumventing authorship recognition to preserve privacy and anonymity. *ACM Trans. Inf. Syst. Secur.*, 15(3):12:1–12:22, November 2012.

[9] Michael Robert Brennan and Rachel Greenstadt. Practical attacks against authorship recognition techniques. In *IAAI*, 2009.

[10] Tanmoy Chakraborty. Authorship identification in bengali literature: A comparative analysis. *arXiv preprint arXiv:1208.6268*, 2012.

[11] Carole E. Chaski. Empirical evaluations of language-based author identification techniques. *Forensic Linguistics*, 8, 2001.

[12] Carole E. Chaski. Who's at the keyboard? authorship attribution in digital evidence investigations. *IJDE*, 2005.

[13] Larry L. Constantine and Lucy A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-centered Design*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[14] Frederick Douglass and Edward Snowden. Nesara-republic now-galactic news.

[15] Cyril Goutte and Eric Gaussier. A probabalistic interpretation of precision, recall, and f-score, with implicatino for evaluation. In *Advances in information retrieval*, pages 345–359. Springer, 2005.

[16] Cercone N. Thomas C. Keselj V., Peng F. N-gram-based author profiles for authorship attribution. In *In Proceedings of the Pacific Association for Computational Linguistics*, 2003.

[17] Adrián Pastor López-Monroy, Manuel Montes-y Gómez, Luis Villaseñor-Pineda, Jesús Ariel Carrasco-Ochoa, and José Fco Martínez-Trinidad. A new document author representation for authorship attribution. In *Pattern Recognition*, pages 283–292. Springer, 2012.

[18] AndrewW.E. McDonald, Sadia Afroz, Aylin Caliskan, Ariel Stolerman, and Rachel Greenstadt. Use fewer instances of the letter "i": Toward writing style anonymization. In Simone Fischer-Hubner and Matthew Wright, editors, *Privacy Enhancing Technologies*, volume 7384 of *Lecture Notes in Computer Science*, pages 299–318. Springer Berlin Heidelberg, 2012.

[19] Arvind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, John Bethencourt, Eui Chul, Richard Shin, and Dawn Song. On the feasibility of internet-scale author identification. In *In Proceedings of the 33rd conference on IEEE Sympsoium on Security and Privacy. IEEE*, 2012.

[20] Jakob Nielsen. How many users in a usability study? `http://www.nngroup.com/articles/how-many-test-users/`.

[21] Jakob Nielsen. Usability 101: Introduction to usability. `http://www.nngroup.com/articles/usability-101-introduction-to-usability/`.

[22] U.S. Department of Health and Human Services. Recruiting usability test participants. `http://www.usability.gov/how-to-and-tools/methods/recruiting-usability-test-participants.html`.

[23] U.S. Department of Health and Human Services. Running a usability test. `http://www.usability.gov/how-to-and-tools/methods/running-usability-tests.html`.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] Dawn L Rothe and Kevin F Steinmetz. The case of bradley manning: state victimization, realpolitik and wikileaks. *Contemporary Justice Review*, 16(2):280–292, 2013.

[26] Efstathios Stamatatos. Survey of modern authorship attribution methods. In *Journal of the American Society for Information Society and Technology*, pages 539–556. American Society for Information Science and Technology, 2008.

[27] Ariel Stolerman, Rebekah Overdorf, Sadia Afroz, and Rachel Greenstadt. Classify, but verify: Breaking the closed-world assumption in stylometric authorship attribution. In *IFIP Working Group*, volume 11, 2013.

[28] H. Turner, J. White, J. Reed, J. Galindo, A. Porter, M. Marathe, A. Vullikanti, and A. Gokhale. Building a cloud-based mobile application testbed. *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, Nov 2012.

[29] G. S. Watson. Inference and disputed authorship: The federalist. *The Annals of Mathematical Statistics*, 37(1):pp. 308–312, 1966.

[30] Zhao Y. and Zobel J. Effective and scalable authorship attribution using function words. In *In Proceedings of the 2nd Asia Information Retrieval Symposium*, 2005.