Lab Project - Final Report
Developing an Linguistic Forensics System and Providing useful NLP Data Visualizations

Daniel Thuerck
Supervisor: Oren Halvani, MSc.

October 9, 2013

With the advent of better and better machine learning techniques and research in the natural language processing (NLP) domain, linguistic profiling has caught a lot of attention in the scientific community in recent years. The field combines the use of machine learning ideas with features and descriptors from the NLP area to profile texts regarding author traits such as gender or age, identify authors of anonymous documents or improve plagiarism detection. In this lab project, we implemented the whole linguistic profile pipeline in C# and added four different visualization techniques that help in investigating which part of the document is responsible for its classification. The final report at hand will give a short introduction to the field, present the implemented framework and execute a simple linguistic analysis using the mentioned visualizations.

1 Introduction

Applications in the linguistic forensics field usually have one of two targets: author trait identification or authorship recognition. In the first case, the researcher is interested in the traits of the possible author of a given document, e.g. gender, age or origin. In most cases, the software connects the traits to probabilities after training on large document collections ("corpora"). To decouple trait recognition from subject recognition, the corpora contain documents from multiple realms. In the second case, one deals with anonymously submitted texts, e.g. texts on a message board posted under a pseudonym and wishes to identify the author from a given collection of authors. Each author in the database is represented by his profile, i.e. a corpus assembling only texts of this particular author.

The application fields for this method called linguistic profiling are wide-spread: Apart from homeland security and criminalistic forensics it can even be used in the insurance sector for linking the customer’s letter of interest to their earnings [10]. Apart from that, profiling can help tracking authors over multiple message boards with different pseudonyms. Similar to video surveillance however, a discussion in the society is necessary due to the impact of the used techniques as they allow not only the tracking of certain people, but also permit us to circumvent
anonymizing techniques that users acquire to exercise their constitutional rights. Interestingly, there is a new upcoming field in NLP research that conquers this problem: After gathering information about a specific author’s style, one can use linguistic profiling techniques to anonymize the text by changing its style while keeping the content and semantics the same [13]. This essentially turns the whole NLP pipeline around.

From a machine learning perspective, both tasks are somewhat similar: Input data consists of raw texts, hence a feature extraction step (features come from the NLP domain) is necessary. Then, both tasks represent classification problems: In the first case, each trait is one class while with authorship attribution, each author is one class. Effectively, most algorithms create a typical style profile (depending on the feature quality) for each class and learn them using easy or more complex machine learning techniques. Consequently, one can model the whole research pipeline as depicted in Figure 1.

The first step is always feature extraction, where the application tokenizes the texts and extracts feature categories such as inter punctuation characters, words, POS-tags or functionwords and assembles them, if desired, into n-Grams. Afterwards, the feature occurrences are counted. Usually, we end up with a lot of different features, mostly in the range of thousands or even ten thousands. To reduce that amount in order to get acceptable program runtimes, feature subset selection methods are used. As wrapper approaches use up too much runtime, one usually resorts to heuristic and filter selections such as $\chi^2$ statistics or correlation. Using that input, most standard machine learning algorithms can be used for classification. Their results are usually evaluated using k-fold cross validation or similar methods and the results can be visualized for the researcher.

In this lab, the main goal was to implement the whole linguistic profiling pipeline in C# and - if possible - invent interesting visualizations that help researchers in drawing conclusions from the classification result.
2 Related work

As linguistic profiling is an interdisciplinary field between machine learning and NLP, there is active research in both areas separately, but seldom in a combined fashion. The most well-known scientific event is the yearly PAN challenge where a specific corpus and a set of classes is given and researchers compete with their programs in terms of classification accuracy [4]. From the machine learning perspective, all techniques that efficiently handle large data amounts can be used, which include the methods used in the data mining field. A broad overview can be found in [7]. Mostly, people use the simplest techniques such as kNN [4] as they deliver results that are better to interpret than with complicated methods such as neural networks. Only a few approaches try unsupervised learning algorithms to extract exemplary profiles for authors and their traits [5]. Computational optimization is widely used, with maximum likelihood and maximum entropy approaches [8].

However, most publications identify the feature categories to be crucial for the success rate, not the used classification algorithms [6]. Hence, a large number of publications care about feature selection measures [1] [2]. Naturally, researchers from the field of linguistic profiling [12] contribute their research to the machine learning community - for example by parameter studies [3] or work on the theoretical side of statistics and feature selection [9]. For starters, [11] provide an interesting survey on the current status on authorship attribution. Unlike related fields that use machine learning such as computer vision, the field currently lacks a set of standard software pipelines - see OpenCV 1. Usually, researchers use modules from NLP software from Stanford 2 and the SharpNLP 3 framework that offer things like POS tagging and sentence splitting. Feature extraction is usually done by hand while the machine learning methods can be used from the Accord.NET 4 and AForge.NET 5 frameworks. Hence, this work can solve the situation by providing a standard API pipeline that is open source. Interestingly, there is also no available set of visualizations that help with the task of linguistic profiling. In this area, we are opening new doors with our work.

3 Our framework

To establish a new standard linguistic profiling (LP) pipeline, we decided to write all components from scratch, only using some basic libraries for maths from AForge.NET. In this section, we present the technical side of the resulting framework as well as a list of its content and functions.

3.1 Runtime environment and used components

The programming language of the framework is C#. Development was done on an Ubuntu 12.04 LTS Intel/Nvidia Linux system using the Mono runtime and tested on a Windows 7 system using Microsoft .NET 4.5 runtime. On top of the Sprachprofiling library, there is a small GUI for the

1 http://opencv.org/
2 http://www-nlp.stanford.edu/links/statnlp.html
3 http://sharpnlp.codeplex.com/
4 http://code.google.com/p/accord/
5 http://www.aforgenet.com/
user’s first experiments with the library and its visualization classes.
The visualization part combines a backend in C#/Mono with a browser-based frontend in HTML,CSS
and JavaScript. The backend records all necessary information into flat files, enabling the
browser-based part to load these files and create the interactive visualizations.
To simplify development, the following libraries (only for some classifiers and visualizations,
not for the pipeline core components) were used (enlisted with their respective licence):

- AForge.NET - LGPLv3
- Accord.NET - LGPL
- SharpNLP/SharpEntropy - LGPL
- jQuery - MIT licence
- heatmap.js - MIT licence + Beerware licence
- d3.js - BSD licence
- d3 parallel coordinates - BSD licence

All mentioned licences are considered compatible to the GNU GPL (v2,v3), which implies
that there is no obstacle in publishing the created source code as a framework under the GPL
along with this report for citation.

The input data consists of a corpus of labelled documents. In our framework, we assume a
special directory structure for a corpus (see Figure 2). In the directory that holds the corpus
name, every subcorpus (e.g. a category, sorted by topics) is another folder. These folders contain
the actual documents in the format class - filename.txt, where the class is separated
from the filename by a simple dash between spaces. Each document is a simple, UTF8-coded
.txt-file with plain text, no markup. Using this structure, the user is able to select one or multiple
subcorpi in our GUI (see next section).
3.2 Framework content

The implemented framework contains all classes that are necessary to build the pipeline presented in the introduction. The principal structure of the library - and with it the C# namespaces - are presented in Figure 3. In the next section, we present each namespace with its most important classes and functions.

3.2.1 Library functions

Let us now discuss the separate functional blocks of the SprachProfiling library and API. The very basic classes are assembled in the Foundations namespace. The block contains the classes Document - a data holder for documents with a given label and I/O functions, Corpus with I/O functions for reading and managing the file structure described above and finally, a set of useful text processing functions is assembled in the class TextTools. All classes that are concerned with feature extraction, management and selection can be found in the namespaces Features and Selection. The framework offers the following feature categories (class names in brackets):

- Inter punctuation characters (Interpunctuation)
- Letters (Letters)
- POS tags at sentence beginnings (POSSentenceBeginning)
- POS tags at sentence endings (POSSentenceEnding)
- POS tags (POSTag)
- Stems (Stem)
Each class offers standardized procedures for feature extraction and of course, functionality for creating n-grams and k-of-n-grams. Additionally, the feature positions in the text can be memorized for the use in our visualizations. Most functional classes need a list of FeatureOrders as input. Such a class is essentially an order for producing a specific feature category. The programmer specifies the feature category and all necessary parameters such that the features can be extracted from given documents. The extraction process itself is handled by the class FeatureCategoryFactory. As a result, we usually deal with lots of features. To obtain acceptable program runtimes, feature selection is crucial. For that purpose, the framework contains the heuristics Average Entropy (Information Gain, AverageEntropy), Gini Index (Gini), $\chi^2$ statistics (ChiSquared), TFIDF (TFIDF), Pearson Correlation (PearsonCorrelation) and Mutual Information (MutualInformation) as well as the option to skip selection (NoRanker).

The base class for heuristics, AbstractRanker, also contains the ability to select the features with the highest occurrence.

By far the largest namespace is Classifier which contains all implemented machine learning methods. Particularly, we offer the following classifiers:

- SVM (2-class and multiclass) from Accord.NET in AccordDualSVM, AccordSVM with data converter from and to Accord.NET in AccordDataGenerator.
- TDIDT (ID3, C4.5; using relative occurrences or binary decisions; one custom implementation and one from Accord.NET) in AccordTDIDT, PTDIDT and BTDIDT.
- Maximum Entropy classification (from SharpEntropy) in MaxEnt.
- kNN (kNN), bNN (binarized nearest neighbour using median thresholding and Hamming distance) in bNN and wNN (distance weighted kNN) in wNN using distances Euclidean, Manhattan and Minkowski (Euclidean, Manhattan, Minkowski).
- Naive Bayes classifier (NaiveBayes).
- PCA-1NN (reduces dimensionality using PCA from AForge.NET, then does template matching with standard 1-NN classifier) in PCATemplateMatching.

All mentioned classifiers execute an ensemble over the feature categories, e.g. by majority voting. Where possible, we derived a confidence measure for the classifier’s decision and use confidence voting. In some cases, combining all feature categories without any form of voting improves the results. For such experiments, the framework offers three modified classifiers: CombinedkNN, CombinedwNN and CombinedNaiveBayes.

The performance of all pipeline stages can be evaluated using classes for k-fold cross validation (kCVEvaluator), stratified k-fold cross validation (StratifiedCVEvaluator) or a training/test data split (SplitEvaluator). All classes are resident in the Evaluation namespace.

Last but not least, the framework offers the Visualization namespace. For each visualization described in the next section, there is one class that handles data generation, export and formatting for that specific visualization. Unfortunately, there is yet not standardization, so each data
### Table 1: Code length statistics for the mentioned namespaces.

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Lines of Code</th>
<th>Lines of Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SprachProfiling (sum)</td>
<td>3675</td>
<td>2120</td>
</tr>
<tr>
<td>SprachProfiling.Foundations</td>
<td>249</td>
<td>244</td>
</tr>
<tr>
<td>SprachProfiling.Features</td>
<td>639</td>
<td>301</td>
</tr>
<tr>
<td>SprachProfiling.Selection</td>
<td>336</td>
<td>194</td>
</tr>
<tr>
<td>SprachProfiling.Classifier</td>
<td>1796</td>
<td>851</td>
</tr>
<tr>
<td>SprachProfiling.Evaluation</td>
<td>189</td>
<td>114</td>
</tr>
<tr>
<td>SprachProfiling.Visualization</td>
<td>368</td>
<td>205</td>
</tr>
<tr>
<td>SprachProfilingGUI</td>
<td>1812</td>
<td>376</td>
</tr>
<tr>
<td>Visualizations (HTML/CSS/JS)</td>
<td>775</td>
<td>10</td>
</tr>
</tbody>
</table>

class in hand-crafted. The visualizations themselves are written by hand in HTML, CSS and JavaScript tested in the Firefox (v23) browser. Hence, all data is written into flat files and read into the HTML page.

Some statistics about the code length for each framework section can be found in Table 1. All in all, the project (not including external libraries) extents to 6262 lines of program code and 2506 lines of comments.

#### 3.2.2 GUI prototype

All functionality described in the last section is contained in a library for use in C#.NET/MONO programs. However, for first experiments with the framework and for the evaluation in the fifth section, the framework comes with a nice Gtk# GUI in the `SprachProfilingGUI` namespace. Essentially, the GUI models the pipeline graphically so that the user can set up all pipeline stages and then submit the task as a “job” for execution in a simple in-order pipeline.

The main GUI views are presented in Figure 4. After selecting the folder holding the corpus in the file chooser on the top, the user is confronted with a list of subcorpora that can be combined by multiple selection for the next job. Going step-by-step through the tab window, the user selects feature categories (in an Amazon-like shopping cart style), classifiers and an evaluator. In all pages, only the contextual sensible information is displayed. Having entered all information, the user can either choose to submit a classification/evaluation job or start the visualization unit with his settings. Eventually, the job is enqueued in a simple FIFO queue on the left side of the window. As the library is already parallelized, experiments resulted in a slight favour for a serialized queue. The queue has eight slots and one active slot at all times. After finishing a job, the user can view the results in a reporting window or the browser, depending on the job type. Please be aware that the GUI is only for demonstration purposes and as such not well tested.
Figure 4: Most important views of the prototypical GUI.
4 Visualizations

Apart from the framework on GUI functionality described in the first part of this document, an important area in this lab project was the visualization. Usually in our classification scenario, we only get a classifier decision and can compare accuracy, F1 and other metrics depending on the input features. To better understand what is going on behind the scenes, we need visualizations that help with feature selection and show why a document was assigned a particular class. In this section, we present the four resulting visualizations using the data exporters mentioned above.

4.1 Feature selection heuristic comparison
As described above, the framework contains different heuristics for feature selection. Each heuristic can assign a numeric score to a feature or even to a whole feature category. Afterwards, the features with the $n$ best scores are chosen for the documents’ feature vector, influencing the classification result. Ideally, such a heuristic would assign a score that is related to the degree of the feature being discriminative for classification. To understand and evaluate the different heuristics, a visualization using parallel coordinates is used (see Figure 5 (a)). Each heuristic is an y-axis and the parallel lines show the features’ scores for different heuristics. Parallelism between lines show heuristics that are very similar, e.g. Average Entropy and Gini index, which are equal except for scale. As we can see, Pearson Correlation usually gives the best distribution.

4.2 Feature frequency and score comparison
In the parallel coordinates visualization, only feature scores can be compared. By exploiting the TFIDF measure, we can also incorporate approximated frequency information. For a better connection between feature occurrence/frequency and score, the tree map visualization (see Figure 5 (b)) was created. Each feature order gets a base color; afterwards, the contained features are displayed as rectangles. Their color intensity shows their heuristic score, a tooltip can explicitly give the numerical score. For better visualization, the scores are placed on an exponential scale to improve the color differences for features with nearly equal, high score. Apart from that, the rectangle’s size models the feature’s relative occurrence, whereas all rectangles of one color sum up to 1.

Another neat feature is the animated class selection: The user can select different classes that are available in the corpus. The feature scores and frequencies are recorded for each class separately, so the user can compare the feature distribution between the classes by paying attention to the transition animation.

4.3 Corpus document structure
In nearest neighbours methods, documents are related to each other by a given distance metric. If all inter-document distances are recorded, we can create a graph that models a map of the corpus - with different colors per class, the user can even identify clusters of documents. However, a fully connected graph would need $O(n^2)$ connections, which is problematic due to the used technique. The D3 force layout initializes the document-distance graph with the given distances and treats each document as a particle - for interactive layout, a PDE-based multibody-simulation
Figure 5: The framework’s visualizations on a gender-corpus (I+II).
Figure 6: The framework’s visualizations on a gender-corpus (III+IV).
is then iteratively solved. During this phase, the user needs to be patient while the graph is converging towards a static layout. With too many connections, the computational overhead becomes huge and the framerate drops below acceptable limits. Hence, we randomly select 30% of the document distances, which should be sufficient for a approximate layout (see Figure 6 (a)).

4.4 Classification hot spots made visible

The final and most distinguished visualization is the document-classification heatmap (see Figure 6 (b)). This visualization tries to answer the key question "Why is the document classified as X - what and where are the key features?". To solve that question, first the backend part uses a Naive Bayes class to model the likelihoods (and posteriors) for each feature in each class over the whole corpus. In a second feature extraction run, the positions of the features are recorded. Combining this information, we can export (in HTML-format) annotated documents. For each feature category, we create a separate div in HTML where each feature is marked with the classes and their likelihoods in its attributes. A JavaScript then uses a transform to create a canvas overlay on the text and displays the heatmap. In the default mode, the heatmap shows normalized (the highest likelihood is scaled to 1) absolute likelihoods. By switching between the classes, we can see differences and identify important, discriminative features. However, this way is not convenient.

A better perspective if offered by class hints: In this case, the feature is only shown on the heatmap of the class with the highest corresponding likelihood, the likelihood for all other classes is set to zero. While this view reveals the striking features in the normalized mode, this mode creates the impression that there are really discriminative features and offers only a qualitative evaluation. In the non-normalized class hint mode, the user can also see the degree of distinctiveness for each feature. Using the provided three modes, the heatmaps can reveal a lot about the feature-class relation.

5 Application and Evaluation

In this section, we will use our library and GUI for a short analysis of text classification.

5.1 Classification results

With all the tools described in the last two sections, we are now able to present a short evaluation of linguistic profiling techniques and inspect a corpus for gender classification using our visualizations. For our first experiment, we used different classifiers on the "Gender_Bravo" dataset using only inter punctuation 1-gram features without any ranking heuristic and evaluated the resulting pipeline with a 5-fold stratified cross-validation. As Figure 7 (a) shows, the resulting accuracy scores differ greatly. While the weakest classifiers - MaxEnt and PTDIDT - are even worse than chance, the best classifier scores an accuracy of 0.75 and an F1 of 0.74 and is as such state-of-the-art. A notable difference can be noticed in the instance-based classification section: Using a simple Manhattan distance, we improve wNN’s accuracy by 20%. Be aware that for this experiment, we used only one feature category, so that neither majority nor confidence voting
Figure 7: Accuracy and F1 histograms for our pipeline evaluation.
could influence the result. Naturally, the score can be improved by feature subset selection and adding more feature categories. However, as later analysis shows, inter punctuation features are extremely discriminative.

This thought directly leads to the second chart (Figure 7 (b)): In this scenario, we compared the usage of different feature categories on the "Gender_News" corpus. The pipeline is constructed using a Naive Bayes classifier, Pearson Correlation selecting 30 out of the 50 most frequent features per category and evaluation by 5-fold stratified cross-validation. In the plot, we used the following feature categories: Functionwords (1), Token 2-grams (2) and Inter punctuation (3) as well as combinations. Again, the result is that inter punctuation features yield the best accuracy score. Hence, the other two features are slightly better than chance. Using two feature categories, we usually improve the accuracy over the smaller score; in this experiment, the score rises with the number of feature categories combined. However, the formula \( \text{Accuracy}(f_1, f_2, f_3) \neq \max(\text{Accuracy}(f_1), \text{Accuracy}(f_2), \text{Accuracy}(f_3)) \). In a follow-up experiment, it would be interesting to evaluate differently sized n-grams. In our experiments, we were limited to \( n = 2 \) due to runtimes.

After comparing classifiers and features, the remaining part of the pipeline ready for evaluation is the feature selection heuristics. In Figure 7 (c),(d) we see that the heuristics only make a 8%-difference in accuracy (scenario: Naive bayes classifier, feature categories 1,2 and 3, 5-fold stratified cross-validation on "Gender_News" and "Gender_Kochrezepte" using the top 20 features per category). Interestingly, there is no clear winner regarding the combination of both experiments. However, Pearson Correlation performs well in both scenarios. A fact that is also visible in our visualizations is the equivalence of Average Entropy and Gini Index (except for scale).

Be aware that the results differ strongly from corpus to corpus - for each scenario, the researcher has to manufacture a specific optimized pipeline.

5.2 The Gender_Bravo corpus visualized

In this last evaluation, we take a closer look on the features from all three mentioned categories and their performance on the "Gender_Bravo" corpus. In Figure 5 and Figure 6 (a), we already used this data for computation, so the reader can instantly draw conclusions from the charts. In this part, we mainly seek to use the provided heatmap from Figures 8 and 9. To begin with, we use the pipeline from the last experiment above. In Figure 8, we present the document m-[Adlerauge].txt that is wrongly classified as female. A quick look on the normalized class hints mode for the female (w) class explains the reasons: Regarding the Functionwords category, the text contains the words "und" and "ich" quite often, which are usually signs for female texts - thus, the heatmap is more intense at the location of these words. Additionally, the integral of the heatmaps for w and m differ strongly, with a favour for the w class. Hence, the posterior \( p(w|m-[Adlerauge].txt) \) is higher than its counterpart of the other class and the document is classified as female. In the other feature categories, there are no striking differences so that the Functionwords alone trigger the decision. In conclusion, the document strongly differs in Functionwords choice from the average male document and hence the text is misclassified.

Let us now take a look at w - [anni24061995].txt. The document is correctly classified.
by the Naive Bayes classifier. Following the logic from above, it should be closer to the average female document, so we expect intense differences between the normalized class hints for both classes. As Figure 9 (a) and (b) show, we are right: The usage of \(\ldots\) and \(\ldots\) is much more common for females, as well as the use of commas. Men on the other hand prefer brackets and seldom used characters (":" and ";"). Again, the other two feature categories are not discriminative enough, as the non-normalized class hints show. In this view, we only see slight blue, supporting the view that NLP can improve greatly with better features instead of adapting better machine learning techniques.

6 Conclusion and Outlook

In this article, we described our implemented SprachProfiling framework, its features and a GUI for demonstration and experimentation. Our main contribution however was the introduction of useful visualizations for document analysis and the hints how to exercise such an analysis in the last part. Elementary, there is still a lot of work to do: As we pointed out, the implemented features are weak. Hence, there is the need to include contextual information to create better features or try stronger k-of-n-gram combinations. Additionally, the heatmap can be improved to show multiple layers at the same time and to give interactive feedback to the user. While these improvements would add new feature, we would suggest to start with performance optimization: Using n-grams with \(n > 2\) usually increases computation time from minutes to hours. With some easy parallelization, the feature extraction process may be ported to massively-parallel hardware and runtimes may be decreased drastically. This could be the scope for a follow-up project in the next months.
Figure 9: Heatmaps for "Gender_Bravo" analysis - a correctly classified document.
References


