AUTHOR IDENTIFICATION OF NEWSPAPER COLUMNS USING STYLE AND SEMANTIC FEATURES

Ergin Doğan Yıldız
B.S., Mathematics and Computer, İSTANBUL KÜLTÜR UNIVERSITY, 2011

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Engineering

IŞIK UNIVERSITY
2016

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

AUTHOR IDENTIFICATION OF NEWSPAPER COLUMNS USING STYLE AND SEMANTIC FEATURES

Ergin Doğan Yıldız

AUTHOR IDENTIFICATION OF NEWSPAPER COLUMNS USING STYLE AND SEMANTIC FEATURES

Abstract

This study has two major purposes : to implement and compare the author classification results of different Naive Bayes Classifiers, and to investigate whether merging individual methods will increase classification success or not.
The subjects of this study were newspaper columnists. Data was collected from well known public newspapers. This study first investigates Numeric, Nominal, Multinominal NBC, and their various merged versions. We then address each method using cross-validation.
The results of the experiments show that merging multiple classification methods can increase classification success. Additionally, it depends on how well individual classification models are constructed.

KÖŞE YAZILARININ YAZARLARINI STİL VE ANLAMSAL ÖZELLİKLER KULLANARAK TANIMA

Özet

Bu çalışmanın iki amacı vardır : farklı Naive Bayes Sınıflandırma metodlarını uygulamak, karşılaştırmak ve farklı sınıflandırma metodlarının birleştirilmesinin sınıflandırma performansına olan etkisini ölçmek.
Bu çalışma gazete köşe yazarlarını konu almaktadır. Çalışma ilk olarak Sayısal, Nominal, Multinominal NBC ve olası birleşik sınıflandırma metodlarını incelemektedir. Sonrasında her bir metod, çapraz doğrulama yöntemi ile test edilmektedir.
Deney sonuçları sınıflandırma metodlarının birleştirilmesinin sınıflandırma başarısını arttırdığını göstermektedir. Bunun ile birlikte bu başarı, birleşimin parçası olan sınıflandırma metodlarının tekil başarılarına bağlıdır.

Acknowledgements

I would first like to thank my thesis advisor Prof. Dr. Ercan Solak who gave me the freedom to research on my own, and guidance to recover when I need help. I also want to thank to my older brother Doç. Dr. Olcay Taner Yıldız for always being there whenever I ran into trouble or had a question about my research or writing. I would also like to acknowledge Anya Gallardo as the second reader of this thesis, and I am thankful for her corrections of my grammatical and semantic mistakes.

Most importantly, none of this would have been possible without the support and patience of my family. I feel lucky to be a part of a well educated family. I want to thank my parents for always teaching me to ask questions and improve myself.

Finally, I want to thank Işık university for their financial support through my Master's degree.

To my Family…

Table of Contents

List of Tables

List of Figures

List of Abbreviations

NLP	**N**atural **L**anguage **P**rocessing
NBC	**N**aive **B**ayes **C**lassifier
URL	**U**niform **R**esource **L**ocator
HTML	**H**yper**T**ext **M**arkup **L**anguage
API	**A**pplication **P**rogramming **I**nterface

Chapter 1

Introduction

There is a vast amount of data on the Internet and it is growing rapidly everyday. Such a high rate of growth also brings some problems with it. We can encounter fraudulent, stolen or unidentified data online on a daily basis. These problems can be dangerous and serious in places like the government, schools or public websites. Because of these threats, and in pursuit of truth, it is important to know the author of a text.

History gives us a fascinating example in The Federalist Papers. The Federalist Papers [1] is a collection of 85 articles and essays written by Alexander Hamilton, James Madison and John Jay promoting the creation of the United States Constitution. At first authorship of the articles was a secret. After Hamilton's death two-thirds of the articles became public. The remaining one-third was identified by Douglass Adair in 1944. After 20 years, Douglass Adair's claims were corroborated by a computer analysis in 1964. Even years after, the authorship of the articles was controversial, drawing more statistical analysis.

The author identification of The Federalist Papers can be considered the most iconic example of the author identification problem. More than 2 centuries have passed since the papers were written but the problem of authorship still exists.

For our work, we want to focus on textual data, specifically newspaper passages. We want to use Turkish newspapers as a data source. We want to use familiar methodologies and algorithms to compare individual methods, as well as we want to add something new to existing solutions. We test on interesting predictors and synonyms to identify authors.

In this thesis, we attempt to create three different classifiers. Each of them use a different methodology to classify authors. We merge two of them together and later merge them all to increase the classification success. We expect that merging classifiers will increase the total performance.

1.1 Literature Review

The problem of author identification problem is an old topic in the Natural Language Processing field. Many researchers and scientists have worked on the problem before. Essays, emails and other sources have all received attention. Some of the methods used are unigrams, bigrams and Naive Bayes Classifier.

There are many researchers who have worked on the subject in the last fifty years. Brinegar [2] used world lengths to analyze documents, Morton [3] took a look at sentence lengths, Brainerd [4] counted syllables per word, Holmes [5] looked at the connection between word frequency and text length. Stamatatos-Fakotakis-Kokkinakis [6] used Multiple Regression and Discriminant Analysis to identify authors.

There have also been studies in author identification in Turkish texts. Taş and Görür [7] used 35 different features on 20 different authors. They used many classification methods such as NBC, NBC Multinominal and compared their success rates. Diri and Amasyalı [8] used 18 different authors with 20 text each. They used 22 style markers and a new proposed method to identify authors.

1.2 Organization

The rest of the thesis is organized as follows :

In chapter 2, discuss our Solution Methodology. We talk about Corpus Construction, and our Classification Methodology. We give details about how we collect our data, and what we do for data cleaning. We also explore our classification methods and its training and testing phases in detail.

In chapter 3, we review the Linguistic Features we used. We talk about Numeric, Nominal and Semantic features. We discuss each Numeric Features we use and provide each of their benefits and their extraction details. We explain what Nominal features are and their differences from Numeric Features. And finally, we describe the Semantic Feature that we use and explain their similarities and differences with Nominal Features.

In chapter 4, we discuss theoretical information about Machine Learning Techniques. We examine the Naive Bayes Classifier, as well as the Gaussian Naive Bayes and Multinominal Naive Bayes, which are different implementations of the same classification methodology.

In chapter 5, we delve into our experiments. We state our experiment parameters, expectations and results. We list our experiments on Naive Bayes Classifier, Nominal Naive Bayes Classifier and Multinominal Naive Bayes Classifier. Finally, we give results of NBC merged with Nominal NBC, and all three methods merged together.

In chapter 6, we give our results a general evaluation. We explore what we can do to expand our work and achieve better results. We conclude the chapter with a list of tasks to do in the future.

Chapter 2

Solution Methodology

In this chapter we explain our approach to the problem. We talk about corpus construction and classification methodologies in detail. We give coding level detail about data collecting, data clean-up and extra data processing. We explain classification concepts briefly and give detailed information about its training and testing phases, including the code references and formulas that we used.

2.1 Our approach to the problem

Author identification is a complex problem and we want to use machine learning techniques. We use a few probabilistic classifiers to predict the author of a given passage.

In order to implement our solution we first need to construct a corpus. In order to do this, we need to collect data from various sources. For our work, we choose to analyze newspaper columns. Newspapers usually are good source for reliable, grammatically correct data. Then we clean up our data. After completing the clean-up, we develop several probabilistic classifiers that are specialized for author identification.

We used 10 fold cross-validation to validate our classifiers. Finally we experiment on and compare our classifiers with various parameters and arguments.

2.1.1 Corpus Construction Methodology

Corpus construction contains two stages. One is collecting the data and the second is to do a clean-up on the data. Each of these stages play a huge role on the classification. Collecting data involves downloading the resources we want to classify. We want to collect diverse and meaningful data to test our classifiers. With the original data, we also create a root version of the same data, and further on we cleaned the data we collected. This helps us remove incorrectly formatted data and finally remove the classification noise.

2.1.1.1 Collecting Data

Data collection was an important part of our work. In order to test our classifiers thoroughly, we needed a big corpus. We decided that we should choose newspapers that have online data on their websites. This allows us to fetch their data easier. For each newspaper, we fetch data into a local file with a pre-defined format. Then later in the project, we create alternative and modified versions of the same data for our future needs.

We work on newspapers because they are good resource which we have easy and public access. Because we want a big corpus, we have to choose many newspapers. We select 10 the top best selling newspapers in Turkey as our data source. We are careful to choose our newspapers from different political, economical and cultural perspectives.

We prefer not to discard any writer of a newspaper for our research. With that said, in certain circumstances, we have to discard writers who have only a few passages or writers who write in areas such as sports, magazine and such.

After we choose our newspapers and writers, we need to download their passages to a local text file. Before doing any type of coding or work, we decide to take a look at the pattern of the work we will do.

Newspaper	Author Size	Passage Size
Cumhuriyet	31	3409
Hürriyet	25	1622
Milliyet	163	8218
Posta	25	4146
Radikal	42	8229
Sabah	36	8112
Sözcü	23	9271
Takvim	76	8973
YeniŞafak	65	12332
Zaman	50	8430

Table 2.1: Corpus throughput

In order to download a writer's passages, we need to reach to the newspaper's website first. This was a trivial problem due to the fact that all of the newspapers we chose had public websites. We then have to find the writers of the newspaper. Every newspaper has some kind of section or part of their website where they list their writers. With this list, we had our writers. This list contains writer's names and some sort of identification of these writers. The next step is to reach all or a portion of those writers passages through these identifiers.

With the ID of an author, we can construct the URL of an author's passages. For this, we need to solve the author-URL format of each newspaper. URL format usually has data, pagination and the identification of an author. After we solve the URL format, we can retrieve all passages of any author at any publishing time for that newspaper.

The final step is to get the title, date, and actual passage from the web page. This is common among all newspapers since all of them have it under "content" tag. Fetching the data from a web page is the easier part of our work. We use `jsoup` to download the actual content.

Jsoup [9] is a Java library for working with HTML. Jsoup parses HTML to simplified java objects. With jsoup, we can parse HTML pages of authors and eventually extract the elements of a passage. It makes our work a lot easier since we do not need to write the code to extract the data from a web page, remove, and validate tags and unwanted elements. Jsoup is very easy to use. For example, the below code can fetch the headlines in the news section.

```
Document doc = Jsoup.connect("http://en.wikipedia.org/").get();
Elements newsHeadlines = doc.select("#mp-itn b a");
```

Applications that index and fetch data from websites are called crawlers. Their purpose is to accelerate the search processes and allow real time searches. For our project, we develop a crawler-like application for each newspaper. Each application is different and unique, because they are modified for a specific newspaper. There are many differences between a traditional crawler and our application. A traditional crawler has a seed (a starting point) to start crawling and it jumps from link to link and indexes each web page it loads. Our crawler is only responsible for one newspaper and its writers, nothing more.

Each of our crawler applications has the same project structure. All of our crawlers are dependent on a base crawler project that contains required base classes. We have a fetcher class and it extends from thread class. Since the fetcher class is a thread, we can create many fetchers and fetch different writers in separate threads. This approach helps us increase the speed of the crawler and make the code easier to develop. Each fetcher fetches the data of an author that they are responsible for.

Crawling starts in the main function of the main class. The main classes are named after the newspaper. For example, for the hypothetical newspaper Tiger Daily, the main class would be `TIGER.class`. In the main function, we get the list of the writers then we start the fetchers. After the fetchers start, their run function is called. In the run function of the fetchers, we fetch the passages of the author that the fetcher is responsible for. Since each newspaper has a different URL formatting, we have to write a different URL generator for each crawler. URL generators will generate a URL that usually contains multiple passage links.

With every generated URL, we get passage links and then loop around these links to fetch the passage. Usually after we get the link to the passage page, the actual passage string is under a `content` tag. We store authors, and their passages in respective lists and eventually write their data to a text file.

We have respective classes for an author and a passage. In the author class, we have an identifier, author name, all passages link and other variables. In the passage class, we have title, passage, publish date, passage link and paragraph list variables. After creating the file, we use `log4j` java library to write passages and their meta data. We separate each passage with a "-" separator.

2.1.1.2 Data Cleanup

Data clean-up is a necessary part for our work. Even though most of the clean-up is done by our `jsoup` library, we still need to do some further work. We mainly do two types of cleaning : one is removing HTML tags from the data and the second is to remove data that failed to download.

Data clean-up is important because it helps filter the data and even discards faulty data. In the end, this helps to clear the data and helps us to test our methods more accurately.

We did tag removing clean-up by using `jsoup`. In `jsoup` we translate a web page into a list of elements. In these elements, we filter out the unnecessary data and only retrieve the type we want to process. This is done by using tags. Thanks to `jsoup's` API, we managed to remove tags effortlessly.

We also remove passages that fail to download. While downloading the passages, we might encounter with some exceptions. The most frequent exceptions we encounter are : run-time exceptions and connection reset by server. For these records, we write the type of the exception in the file and move to the next passage. We have to remove these passages after we complete the respective newspaper's crawling.

2.1.1.3 Linguistic Processing

During our classification process and various tests, we come up with features and methods that need a data that is stripped to its roots. Because of this reason, we have to process the data and create a new version of the data. We could do this processing on the fly, but that would be repetitive work and would lengthen the time we need to classify.

To further process the data, we need to load the data we have already downloaded. For this purpose, we wrote a function which loads the data. This function has several parameters ; top passage count, min passage count and author count. These parameters allow us to control the loading process.

We used Zemberek [10] to find the roots of the words. Zemberek is an open source NLP library. Zemberek has many modules including morphology, tokenization, and others. We used Zemberek because it is very fast and has a high success rate at finding roots of the words.

In order to use Zemberek to find the roots, we had to write a disambiguator package in our project. First we load all of the original data. For each newspaper, we loop through their authors and then loop through each author's passages. We split each passage into sentences. Then we send them to Zemberek's "parser" function which will return the root version of each word in the sentence. By adding each sentence to each other, we reconstructed the passage and re-wrote the stemmed version passages to a new file.

2.1.2 Classification Methodology

In this section we give a brief definition of classification and its phases. We provide a list of classifiers as well as detailed information about the classifier we used. We talk about the phases of a classifier; testing and training. For each phase, we give information regarding our models to finding results and finally returning the success result of a classifier.

2.1.2.1 Classification Concept

Classification is a problem of identifying which category a new input belongs in. An algorithm that implements a classification is called a classifier. There are many algorithms that can be used for classification. Some of them are : Linear Classification Algorithms, Support Vector Machines, Quadric Classifiers, Kernel Estimation, Decision Trees, Neural Networks [11]. For our work we implemented a Linear Classification Algorithm, Naive Bayes Classifier.

Classifiers usually have three main parts : an algorithm that will create a model for the data which can be called as modeling, a training part which entails training our model with training data, and finally a testing part. In testing part, we test our model with the data that we already has a known category or group. Success of the classifier depends on the type of the data to be classified. Thus it is very important to examine the characteristics of the data and understand the details of it. Classifiers usually need to be modified and specialized according to data.

We used a k-fold cross-validation testing method to test our classifier. K-fold cross-validation is widely used for classifiers. K-fold cross-validation has `k` iterations. On each iteration one random unit datum is selected for testing and the remaining `k-1` are used for training. This process is repeated `k` times while each randomly selected unit is used exactly once. With this method, we ensure that all data is used for both training and testing. For our work we use 10-fold cross-validation.

Identification of authors is a complex problem. We want to use machine learning practices when approach this problem. We choose to implement a classifier because of these reasons and specifically NBC because it has high scalability due to number of features/predictors it can have. When we need to try new model we can do it easily.

As mentioned above, for our classification method we used various features to predict the authors. First we had to come up with features. Features can be numeric or nominal. We search for features that will differentiate authors from one another. When we decide that we found a feature we would add it to our feature list. The feature list serves as an outline for a passage. For numeric features we wrote a `FeatureExtractable` interface that every concrete feature class we create will extend from. The `FeatureExtractable` interface has only one function : `extractFeatureResult`. `ExtractFeatureResult` takes passage as an input and returns a double value. For nominal features we had to use respective classifier's feature representation. This usually is a map of `string-double`.

At the beginning of each classifier we experimented, we transformed passages into feature vectors. We only need to do this transformation/extraction once. After we extract the features from passages, we add these features to a feature vector. Transforming the list of features to a vector makes future calculations easier. As we discussed above, we use 10-fold cross-validation. For each fold of the validation, we construct the training vectors and test vectors. We pass the training vectors to `Trainer` class. `Trainer` class will iterate over all of the training vectors and the classifier will create a model. This classifier model will be an input to the tester as well as testing vectors. The tester class will test the testing vector with the model and print a success result. The average of all folds will be the final success of our classification methodology.

2.1.2.2 Training the Classification Model

The training phase of a classifier is responsible for creating the training model. Training model loop through the training data and calculates necessary probabilities to create the model. We use a training model to classify testing data in the testing phase.

Training classes have two input arguments : features and training data. For numeric features, we have a feature list; for nominal features we have a map of `string-double` pairs. After we loop through all training data, we have a list of `string(author name)-double vector`. This list is the output of the training phase and re-calculated in every fold. Calculation details are as follows :

- **Numeric Feature's training calculation :** For this type of training we are going to calculate mean and standard deviation vectors for each author.

  For each element x_i of the feature vector, we can calculate the μ of the feature with:

  $$\mu = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{2.1}$$

  For x_i and μ_i we can calculate the σ with:

  $$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_i)^2} \tag{2.2}$$

  To calculate these vectors we have to loop through authors. On each author we retrieve training data of that author and set it to a list. Now we can loop through each feature vector's values and calculate their mean and standard deviation.

- **Nominal Feature's training calculation :** For this type of training we have to calculate the probability of each author. For T_{ct} is the counter for word t in passage c, we can calculate the conditional probability of t given c with:

  $$\text{condprob}[t][c] = \frac{T_{ct} + 1}{\sum_{t'}(T_{ct'} + 1)} \tag{2.3}$$

  To calculate the conditional probabilities, we have to loop through authors. On each author we retrieve training data of that author and set it to a list. Now we can loop through each vector and calculate the probability.

2.1.2.3 Testing the Classification Model

The testing phase of a classifier is responsible for testing the training model and finding the success rate of it. The tester class loops through testing data and predicts an author for each of them. Adding each prediction result to each other will give us the final result for a classification method.

Our testing class has two arguments : testing data and training model. We calculate the probability of all authors for every testing data, and we return the author who has the highest probability as a decision. The probability calculation details are as follows :

- **Numeric Feature's testing calculation :** For this type of testing we are calculate likelihood of each author using training model. We can calculate the probability of passage x authored by C_j with:

$$P(x|C_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{\frac{-(x-\mu_j)^2}{2\sigma_j^2}} \tag{2.4}$$

  To calculate the probability of an author for given passage, we have to use the training model. At this point we already have the mean and standard deviation of a given author, and we can use these vectors with the formula 2.4 to calculate the probability.

- **Nominal Feature's testing calculation :** For this type of testing we have to calculate the probability of each author. T begin the passage, for every word t in passage T, we can calculate passage T authored by author c with:

$$\sum_{t}^{T} \log \mathrm{condprob}[t][c] \tag{2.5}$$

  To calculate the probability of an author for any given passage, we have to use the training model. We loop through the passage vector and for every value greater than zero, we fetch the related vector value from our training model with the parameter of the author. Adding these values will give us the final probability of an author.

After iterating over all of the testing data, we have a map of successful and failed guesses. Successful guesses divided by all guesses gives us the success rate of the fold.

Chapter 3

Linguistic Features

In this chapter we explain the types of features we use. We discuss numeric, nominal and semantic features in detail. We talk about differences between the three. We give detailed information about their data structure and how to extract their feature values. We explain the benefit of using them in our research, including code references to class structure.

3.1 Numeric, Nominal and Semantic Features

A feature is an attribute of an object that can characterize it. Most objects and entities have more than one feature. In machine learning, we represent such objects as a vector of features. Features help us differentiate objects from one another and help us describe them. It is essential to select useful and distinctive features in order to achieve high classification scores. Features can be numeric, nominal and semantic. We will explain each of them in detail below.

Representing passages can be challenging. To achieve a sufficient representation, we try to find various types of features. We add different features together to achieve better representation and better classification scores. We also experiment on each type of feature in isolation as well as experimenting by merging them together gradually.

3.1.1 Numeric Features

A numeric feature is a measurement. Numeric features represent a feature of a passage with numbers. We might say word count in a passage is a numeric feature and that is true, but whether it helps to identify one passage from another is a question we will answer later.

Numeric features were the starting point for our research. This is fitting because numeric features are simpler and easier to implement. They are also good at identifying passages. With the outcome of a number, it is easier to find new useful features. Numeric features help us roughly shape our model.

We choose our numeric features carefully. We want to select features that are distinctive and easy to calculate. We must find features that are neither common nor rare. We came up with six feature that are suitable with our criteria. We want to use more than one feature because, passages and authors have multiple features and styles.

All of the features we use, their extracted result is a `double` value. For the ease of use and design, we define an abstract `Feature` class and a `Feature Extractable` interface. The `Abstract Feature` class contains variables and functions that all features will share. Abstract Feature class implements `Feature Extractable` interface. `Feature Extractable` interface has a function called `extractFeatureResult`. When we want to create a new feature, we extend the new feature from the abstract class and implement `extractFeatureResult` function. With this design we can create new features and add them to the feature list easier.

3.1.1.1 Features Used

We wanted to use length and quantity as features. We used Average Word Length as length feature. For countable features we used Punctuation Count, Sentence

Length as Word count, Vocabulary Extend, Average Paragraph Length, and Word Count. We explain these features more in detail below.

- **Average Word Length :** Average word length is a simple but effective feature. While selecting the features, we think about how we can represent an author's writing style. Some authors like to use longer and more complicated words, while some do not. We think this can be a good feature to identify an author.

- **Punctuation Count :** Punctuation count is an interesting feature. We want to create features with different aspects of writing, and punctuation is definitely one of them. We suspect some authors use commas or semicolons more extensively. This can indicate that they like to present information in list format.

  The punctuation we are working with are : ".,:;"

- **Average Sentence Length as Word count :** Average sentence length is one of the first features we first selected. It is very clear for us that some authors uses considerably longer and complicated sentences, while some uses shorter and more easily understood sentences. Both of them are style choices and we believe it will help us identify authors.

- **Vocabulary Extend :** The vocabulary extend feature is used to measure the vocabulary of an author. We think the vocabulary size of an author can be a good identifier. Some authors might like to use the same words repeatedly, while some authors might want to use different words.

  We calculate vocabulary feature by dividing the distinct word count by the total word count.

- **Average Paragraph Length :** The average paragraph length feature is used to measure the paragraph length of authors. We separate paragraphs by special characters in the data. While we are loading the data, we are loading the passage as a list of paragraphs. Each paragraph is a separate string. We think of paragraphs as sections or parts of the passage. Paragraph length is a good feature because a length of a paragraph indicates an author's style to form his/her passage.

We calculate average paragraph length by dividing total words in the passage by paragraph count.

- **Word Count :** The word count feature measures the size of the passage. We think passage size can be a good identifier. Some authors might prefer to write longer sentences, some are more succinct. We think it is a good predictor for us.

  We calculate word count by splitting sentences into words and counting them.

3.1.2 Nominal Features

In linguistics, "nominal" refers to the group nouns and adjectives. The nominal feature we use can also be called as "word feature". In numeric features, we use numerical measurements to represent a passage. For nominal features, this is done by word frequencies. In numeric features we calculate, for example, average word length. Now with nominal features, we create a histogram with the frequencies of the words.

We want to use Nominal Features because, we want a non-numeric feature to represent passages. We want to compare the classification scores using nominal features compared to numeric features. The benefit of using nominal features is to use word frequencies as a representation, thus create a bag of words for each author. This helps us to represent an author with a set of words.

We think that word frequencies can help identify passages. Most authors have different backgrounds, education, and culture. We think that since they have different backgrounds, they should have a different word set that they use often.

If we can represent their habits of using these words, this might help us identify them. Nominal features are really helpful to create these word sets.

To construct our word set, we want to create a word-frequency file that contains all words of data and their frequencies in descending order of their frequencies. Using all of this file would be unnecessary since there will be words that are used so rarely that will have no value on the identification process. Similarly, we need to avoid top used words because these words are usually `stop-words` and they also have no value on the identification. We solve this problem by using two arguments : padding and window size.

We think of our nominal feature list as a window on the file. We have to find the optimal padding from the beginning of the file (avoiding most frequent words) and also the window size (avoiding least used words). We start to experiment with window size. Since if we don't know the optimal window size we can not set the padding. We fix the padding to 0 and experiment on the value of window size. We find that a 500 word window size gives us the highest score. After finding the optimal window size, we fix the window size to 500 and experiment on padding. We find that 125 gives us the highest score. With these test, we find that we need to discard first 125 words and use the 500 after the first 125 words.

In order to find the word frequencies, we have to load all of the data and create data that is stripped to its roots. This is necessary because we want to use the core of the words to represent the passages. After we construct the root version of data, we use this data to create a `word-frequency` file. We simply go through the new data and count the frequency of each word. Then we write the `word-frequency` map to the output file.

3.1.3 Semantic Features

A numeric feature was representing features with numbers and a nominal feature was representing features with words. Semantic features represent features with sets of meanings. We use synonym sets as semantic features. We want to use semantic features because all the features we use were not directly tied to the meaning of word. We want to use the meaning of the words in our features. We want to create a wordNET of synonym set for each author as a model which can represent an author's writing topic.

To use synonyms for semantic features, we needed a wordNET. We found open source project BalkaNet [12] for our synonym set source. BalkaNet is a multi-lingual lexical database containing individual wordNETs for Balkan languages. BalkaNet data is an XML formatted file. Every line of the file defines a synset. The structure of each line is as follows :

```
<?xml version="1.0" encoding="UTF-8"?>
<SYNSET>
   <ID>ENG20-09434163-n</ID>
   <POS>n</POS>
   <SYNONYM>
      <LITERAL>
         baba
         <SENSE>1</SENSE>
      </LITERAL>
      <LITERAL>
         peder
         <SENSE>1</SENSE>
      </LITERAL>
   </SYNONYM>
   <ILR>
      ENG20-09725018-n
      <TYPE>hypernym</TYPE>
   </ILR>
   <ILR>
      ENG20-09663987-n
      <TYPE>near_antonym</TYPE>
   </ILR>
   <DEF>Bir cocuga gore kendisinin dunyaya gelmesinde
   etken olan erkek</DEF>
   <BCS>1</BCS>
</SYNSET>
```

Figure 3.1: Structure of synonym set record in BalkaNet file

We first need to process the BalkaNet data to transform it to a synset map. For this we have to go through every line of the file and find the synonym relations. We discard everything but nouns. Every line in the BalkaNet file has a unique synset id. We have to define a map with the synset id and add each new synonym we found to the value of this id. Every synset has a list of words (synonyms) that belongs to that list. For instance, in figure 3.1, `ENG20-09434163-n` is the id of the synset and literals `baba` and `peder` belong to this synset. We calculate the synset value by looking into this list and increasing the value by one when we see the word in the list.

While running the classification, we have to have a reverse dictionary for synsets. Because BalkaNet contains 10K synsets, finding which synset a new word belongs to would take too much time. To make this search faster we create a new map, mapping each word with the synset they belong to.

Chapter 4

Machine Learning Techniques

In this chapter we review the machine learning techniques that we use. We explain why we use certain methods and classifiers. We explore each method with their formulas and their classification phases.

Machine learning, is to use previous experiences to increase its future predictions and decisions. It can be applied to areas such as data mining, pattern recognition, classification and such. For our problem, we have to use machine learning practices because there was no simple or straightforward way to identify the authors. Since we can not know certainly whether a passage belongs to a given author, we can only calculate the probability with that author's previous passages.

In our solution, we want to use the Naive Bayes Classifier. The Naive Bayes Classifier is a suitable method for our problem for two reasons : one, is that it uses a statistical decision logic based on the collected data, and second, it is easy to implement and extend with new features and predictors.

4.1 Naive Bayes Classifier

The Naive Bayes Classifier is a probabilistic classifier that applies the Bayes theorem while presuming the features are independent. NBC has two parts : training and testing. In the training part, NBC creates a model with the training data using features/predictors. In the testing part, NBC tests the classification model using testing data. Some of the NBCs are Gaussian NBC, Multinominal NBC and Bernoulli NBC.

In order to classify an instance of data, NBC creates a probabilistic model. Let's say our instance is $X = [x_1, x_2, ..., x_n]$. With given class C, probability of instance X belonging to class C is :

$$P(C|x) = \frac{P(C)P(x|C)}{P(x)} \tag{4.1}$$

We can also rephrase the formula like :

$$posterior = \frac{prior \times likelihood}{evidence} \tag{4.2}$$

With an assumption of independent features, the probability of X belonging to class C_i with K being the number of total classes is :

$$P(C_i|x) = \frac{P(C_i)P(x|C_i)}{P(x)} = \frac{P(C_i)P(x|C_i)}{\sum_{j=1}^{K} P(x|C_j)P(C_j)} \tag{4.3}$$

And NBC picks the class that has the highest posterior probability :

$$C_{max} = \max_i P(C_i|x) \tag{4.4}$$

Depending on the type of the data we are going to classify, we can use different types of NBC. For continuous data, we use Gaussian NBC and for discrete data, we use Multinominal NBC.

4.1.1 Gaussian Naive Bayes

Gaussian NBC assumes that the data of class C_j is a Gaussian distribution. With Gaussian NBC, for every X_i feature and C_j class, we calculate the mean and variance. Then we can calculate the probability of X given with class C_j with

$$P(x|C_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{\frac{-(x-\mu_j)^2}{2\sigma_j^2}} \tag{4.5}$$

In the training phase of Gaussian NBC we calculate the μ_i and σ_i of the training data. Then in the testing phase, we use μ and σ vectors and formula 4.5 to calculate the probability of an instance X belonging to a class C_j.

4.1.2 Multinominal Naive Bayes

Multinominal NBC [13] is widely used in text classification. With Multinominal NBC we can represent passages as a histogram of words. We can transfer a passage d to a word frequency list, $d = [t_1, t_2, ..., t_n]$ where n_d is the number of tokens in passage d. The probability of passage d authored by author c is :

$$P(c|d) \propto P(c) \prod_{k=1}^{n_d} P(t_k|c) \tag{4.6}$$

where $P(t_k|c)$ is the conditional probability of word t_k used by author c. We use $P(c)$ (prior probability) in case the passage's words do not provide clear evidence. In equation 4.6 for each t_k in n_d probabilities are multiplied. To avoid floating point underflow we will transform the equation into logarithmic form.

$$P(c|d) \propto \log P(c) + \sum_{k=1}^{n_d} \log P(t_k|c) \tag{4.7}$$

We can calculate the conditional probability $P(t|c)$ for t being the word that author c uses :

$$P(t|c) = \frac{T_{ct}}{\sum_{t \in V}(T_{ct})} \tag{4.8}$$

where V is the vocabulary, T_{ct} is the frequency of word t in training set of author c. To avoid zeros, we use Laplace smoothing, which adds one to each frequency.

$$P(t|c) = \frac{T_{ct} + 1}{\sum_{t \in V}(T_{ct} + 1)} \tag{4.9}$$

In the training phase of Multinominal NBC, we calculate the probability of each word t written by author c with equation 4.9. In the testing phase we calculate the probability of passage d authored by c by getting the maximum value of :

$$c_{max} = \arg\max_{c \in C}[\log P(c) + \sum_{k=1}^{n_d} \log P(t_k|c)] \tag{4.10}$$

where C is all authors.

Chapter 5

Experiments

This chapter contains the experiments we perform on our different classifiers. In the first section we talk about how we select the data from the corpus and with which rules. We explain and detail the parameters we use. In the second section we discuss experiments on Gaussian NBC. We give information about numeric, and nominal feature testing as well as methodology. Then we experiment on the merged classifier of numeric features and nominal features. In the third section we talk about experimenting on synonym NBC. We perform an isolated experiment on Synonym NBC and also the merged version with all other classifiers.

In our experiments we want to fix the input of our tests to compare each method. These parameters are : min passage count, max passage count, word window size, and padding size. In the first section we do experiments to find the optimal (min passage count, max passage count) pairs that makes features meaningful. In the second section we experiment on word window size and padding size to achieve better scores with Nominal NBC. In all of our experiments we do 10 fold cross-over validation.

5.1 Experiments about data selection

Loading data to our classifiers is a routine task that we do at the beginning of our classification process. The loading function has parameters such as `minPassageCount`, `maxPassageCount`, `authorCount` and more. I want to talk about `minPassageCount` and `maxPassageCount` parameters because they are the ones that determine the data we load. `AuthorCount` is also a parameter that determines how many authors we will load but for our test we do not want to use this parameter to restrict the data.

Every author in the loading data has to have `minPassageCount` number of passages. Similarly, every author in the loading data can have `maxPassageCount` number of passages. Deciding `minPassageCount` and `maxPassageCount` was an experiment case for us.

In order to find these numbers, we create a histogram of the corpus. We find that in average every author has 157 passages with a standard deviation of 153. The reason we have such an high standard deviation is that some authors write more frequently than others. Some authors might write once a month and some might write everyday.

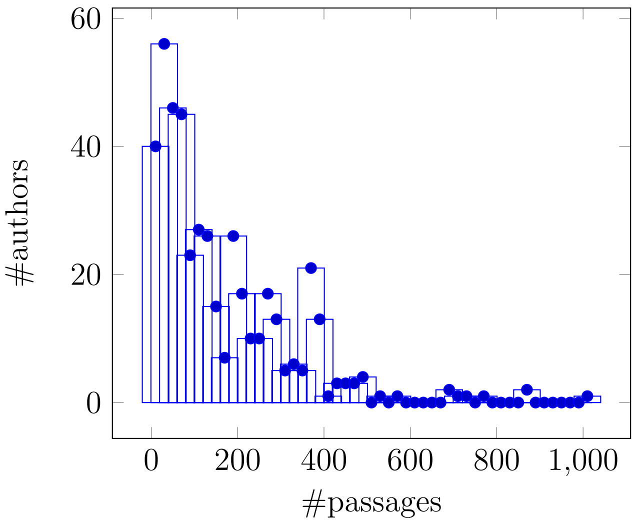

Figure 5.1: Passage count relative to authors.

We need to test the minimum passage parameter further. We choose a feature like the `AverageParagraphLength` feature. We create a histogram of this feature for every author. We calculate the respective mean and standard deviation values for each. Then we select the data between μ - σ and $\mu + \sigma$. With this method, we discover how many passages are needed to make this feature meaningful. Proportioning this number with how many passages an author has written gives us the percentage of the passages needed for a meaningful feature. Since we calculate this percentage among authors, we make this value author independent by getting the mean of all percentages. The result was 74%. 74% of an author's passages was needed to make `AverageParagraphLength` feature meaningful.

To further extend our experiment we perform the same test with all the features we have. The results are quite illuminating.

Feature	Percentage Needed
AverageParagraphLengthFeature	74.45
AverageWordLengthFeature	70.52
PunctuationCountFeature	73.69
SentenceLengthAsWordCountFeature	84.48
VocublaryExtendFeature	72.67
WordCountFeature	75.28

Table 5.1: Percentage needed for a feature to be meaningful

As we can see above, almost all features are in a range between 70% and 85%. This shows us that the feature type does not have an effect on the minimum passage count that is needed. Regardless, we calculate the mean of these percentages; our final result is 75%. Now with this finding, we can set `minPassageCount`. `MinPassageCount` is 75% of 157 (mean of the whole corpus) 117 $\approx$ 120. `MaxPassageCount` is 157 + 153 = 310.

In conclusion, we find that the minimum passages needed for our classification method is 120 and the maximum passages needed is 310. For our further tests we will use these values as their respective parameters.

5.2 Experiments with Gaussian NBC

In this section we do experiments on Gaussian Naive Bayes Classifiers. In the first subsection we perform experiments on Naive Bayes Classifier using numeric features. We give information about its testing methodology and its success rate. In the second subsection we do experiments on Nominal Naive Bayes Classifier using our bag of words feature. We discuss the methodology and success rate of the classifier. Finally, we experiment by merging the two methods together.

5.2.1 Experiments with NBC

While experimenting with Naive Bayes Classifier we use numeric features. We want to add more distinctive features to increase our success rate. We find in the experiments that when we follow the cumulative approach, our success rate increases.

In our first six experiments we wanted to use all our features in isolation. With this method we want to find the most beneficial feature in all of them. Then we try to add remaining features to the most beneficial one and try to find the most beneficial pair. This is also called forward-feature selection.

For all of our experiments on NBC with numeric features, we experiment with the following parameters; Minimum passage count : 120, Maximum passage count : 310, Total authors loaded : 215, Total passages loaded : 50620, Total passages tested per fold : 5062. For ease of use we refer to our features as shown in table 5.2.

We want to experiment on isolated features. With this experiment we will find the best isolated feature.

As we see in table 5.3, we find that `WordCountFeature` is the most successful feature. We also see that σ of correct guesses are very low. This indicates that

Feature	Referral
AverageParagraphLengthFeature	F1
AverageWordLengthFeature	F2
PunctuationCountFeature	F3
SentenceLengthAsWordCountFeature	F4
VocublaryExtendFeature	F5
WordCountFeature	F6

Table 5.2: Feature referrals

Feature	Success Rate	μ of Correct Guesses	σ Correct Guesses
F1	4.97%	251.9	14.6
F2	2.51%	127.5	9.2
F3	3.4%	172.2	12.33
F4	3.39%	172.1	10.23
F5	2.59%	131.4	8.23
F6	5.06%	256.3	8.04

Table 5.3: Isolated performances of each feature

our model has consistent predictions. We will give a brief comment on each feature and its result as follows:

`AverageParagraphLengthFeature` was one of the good features we introduced. With 4.97% success rate it is the second most successful feature.

`AverageWordLengthFeature` was not very useful like the `VocublaryExtendFeature`. While calculating the average word length, a whole passage of words might draw the average closer to the other passages. With 2.51% success rate, it is the least successful feature.

`PunctuationCountFeature` was a moderately good feature. With 3.4% success rate, it is the third most successful feature.

`SentenceLengthAsWordCountFeature` was as useful as the `Punctuation Feature`. Assuming every sentence has punctuation at the end and some in the middle, this might explain the similarity in benefit of the two. With a 3.39% success rate, it is the fourth most successful feature.

`VocublaryExtendFeature` was not very useful compared to other features. We can say that for our data, authors did not use many different words compared to one another. In other words, they keep the ratio of unique words similar to each other. With 2.59% success rate, it is the fifth most successful feature.

`WordCountFeature` was the best feature we created. We first thought newspapers might demand a minimum an column size from an author. But the upper limit of the column might have an effect on the success here. With 5.06% success rate our word count feature is the most successful feature.

Before we start forward-feature selection, we want to experiment on `AverageWord-LengthFeature`'s parameter whether or not it is beneficial to look at the roots of the word when using `AverageWordLengthFeature`. For this experiment, we will use a smaller data sample. We will use 6 authors and each author will have 310 passages.

Feature	Root Version	Success Rate	μ of Corr. Guess.	σ Corr. Guess.
All Features	false	77.95%	145	5.92
All Features	true	76.55%	142.4	6.05

Table 5.4: AverageWordLengthFeature performance with root-version parameter

As expected, using the root version when using the average word length feature, decreases the success rate. Using the roots of the words decreases the difference between words and eventually decreases the average difference between authors.

Next we will experiment using two features. We will fix `WordCountFeature` and experiment to find the best feature pair.

Feature	Success Rate	μ of Correct Guesses	σ Correct Guesses
F6&F1	9.88%	500.6	20.42
F6&F3	8.46%	428.3	9.95
F6&F5	8.14%	412.4	20.44
F6&F2	9.64%	488.1	18.08
F6&F4	9.38%	474.8	10.1

Table 5.5: Performance of each feature when paired with feature F6

As we see in table 5.5, we find using the `AverageParagraphLengthFeature` with the `WordCountFeature` gives us the best result. They are both the two most successful features. Pairing `WordCountFeature` with `AverageWordLengthFeature` also gives us a close result of 9.64% compared to 9.88%. The idea of adding the second most successful to the most successful to get better scores might not be true. To find an answer to this question, we will continue trying new features in the same fashion.

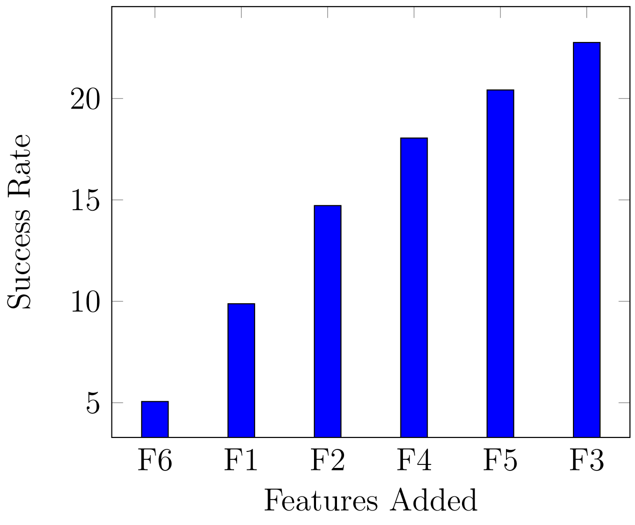

Figure 5.2: Forward feature selection success results

On tri-grouping, merging F6&F1 pair with `AverageWordLengthFeature` gives us better results compared to `PunctuationCountFeature`. This is an interesting result because the `PunctuationCountFeature` was more successful compared to `AverageWordLengthFeature` in isolated tests. But merging F6&F1 with `PunctuationCountFeature` is not more successful compared to the `AverageWord-LengthFeature`.

On tetra-grouping, merging F6&F1&F2 with `SentenceLengthAsWordCount Feature` gives us the best result. Despite its weaker contribution in getting better success in the tri-grouping test and the isolated test, `SentenceLengthAsWord-CountFeature` gives us the best tetra-grouping performance.

On penta-grouping, merging F6&F1&F2&F4 with `VocabularyExtendFeature` gives us better results compared to merging with the `PunctuationCountFeature`. Both in isolated and tetra-grouping tests, `PunctuationCountFeature` was more beneficial, but on penta-grouping, this is again different.

In our final grouping we used all of our features together. Combining all features increased our classification success from 20.42% to 22.76%.

All of our comments above have one common result. Even though isolated performances are lower, some features might score higher when merged with more appropriate features.

We find with our grouping experiments that combining our features with each other increases our success rate linearly. We do not need to do forward feature selection.

Our numeric feature usage brought us some level of success. However, we want to know whether our success will increase with using different types of classification methods or merging them.

5.2.2 Experiments with Nominal NBC

Another Naive Bayes Classifier we experimented on had nominal features. With nominal features, we need to create a model with a bag of words. To find our bag of words, we have to make two experiments as we mentioned in Chapter 2. We need to experiment on window size and padding. And finally after we find the windows size and padding parameters, we can experiment on the classifier.

5.2.2.1 Experiments with word window-padding

Before experimenting on window size and padding, we need to find the window size first. We set the padding to 0 and increase the window size gradually. After we find the optimum window size, we start to experiment on the padding size.

To experiment on padding size, we fix the window size to the optimum value and start to increase the padding from 0 gradually. Then we make an experiment with optimum window and padding sizes.

We use a smaller sample from original data to determine window size and padding. For our window size and padding tests, we loaded 38 authors with 8720 passages loaded.

- **Experiment 1 :** In this experiment we set the padding to 0. We increased the word window size from 30 to 600.

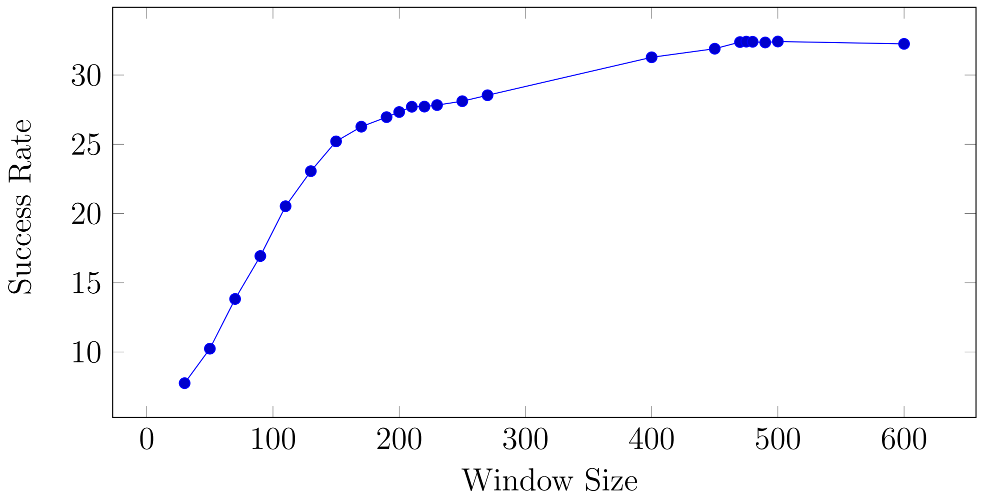

Figure 5.3: Window size relative to success rate

The nominal classifier success rate peaked at 32.42% with a 500 word window size and 0 padding. This experiment showed us that using 500 words as a bag of words size was optimal.

- **Experiment 2 :** In this experiment we set the window size to 500. We increased padding from 0 to 10000.

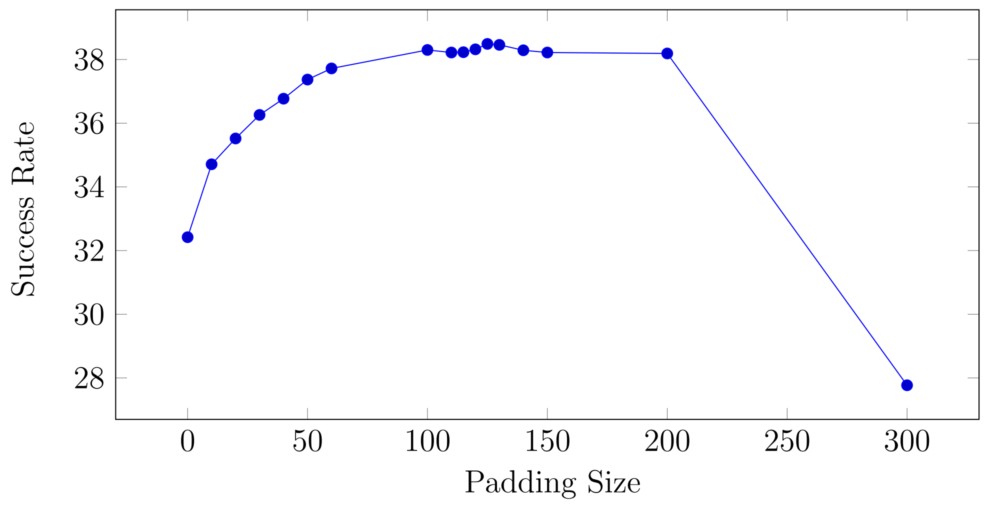

Figure 5.4: Padding size relative to success rate

The nominal classifier success rate peaked at 38.46% with a 500 window size and 125 padding. This experiment shows us that we need to discard 125 most frequent words from the word frequency file to maximize the success rate.

- **Experiment 3 :** After we found the optimal values for window size and padding size, we want to make a final test for Nominal NBC. For this experiment we set the minimum passage count to 120, the maximum passage count to 310, the window size to 500, and the padding size to 125. We used 195 authors and 45440 total passages to test our Nominal NBC.

Classification Method	Success Rate	μ of Corr. Guess.	σ Corr. Guess.
Numeric NBC	22.76%	1152 out of 5062	19.51
Nominal NBC	27.89%	1267.4 out of 4544	21.77

Table 5.6: Numeric NBC vs Nominal NBC

As we can see in table 5.6, Nominal NBC has better success rate compared to Numeric NBC. This experiment tells us that for stand alone usage, Nominal NBC is better at identifying authors than Numeric NBC. Representing passages with a bag of words compared to numeric values is a better solution.

5.2.3 Experiments with Merged Gaussian NBC

Numeric features and Nominal features represent our data and create their model differently. We want to know whether we can increase our success rate by merging these two classifiers? In a merged classifier, we model two different models and calculate their probabilities separately. To find the final probability, we multiply respective probabilities.

For the next two experiments we will use all of our numeric features for our numeric classifier as in table 5.2. We used 500 as the window size and 125 as the padding size.

- **Experiment 1 :** In this experiment we test the result of two classification methods merged together. We are expecting a better success rate relative to each separate classification method. We experiment with the following parameters; minimum passage count : 120, maximum passage count : 310, total authors : 197, total passages : 45700.

Classification Method	Success Rate	μ of Corr. Guess.	σ Corr. Guess.
Numeric&Nominal	35.23%	1610.1 out of 4570	17.22

Table 5.7: Numeric NBC and nominal NBC merged success rate

As expected, our classification success rate increases relative to each classification method. This increase was not the combination of each individual method but it was a subtle increase.

- **Experiment 2 :** We want to further experiment on the author parameter to identify how our classifier's success changes depending on small author samples and bigger samples. In each experiment shown below we fixed all parameters but author size. The author size changes between 2 to 190, and the total passage count changes between 1120 to 44100.

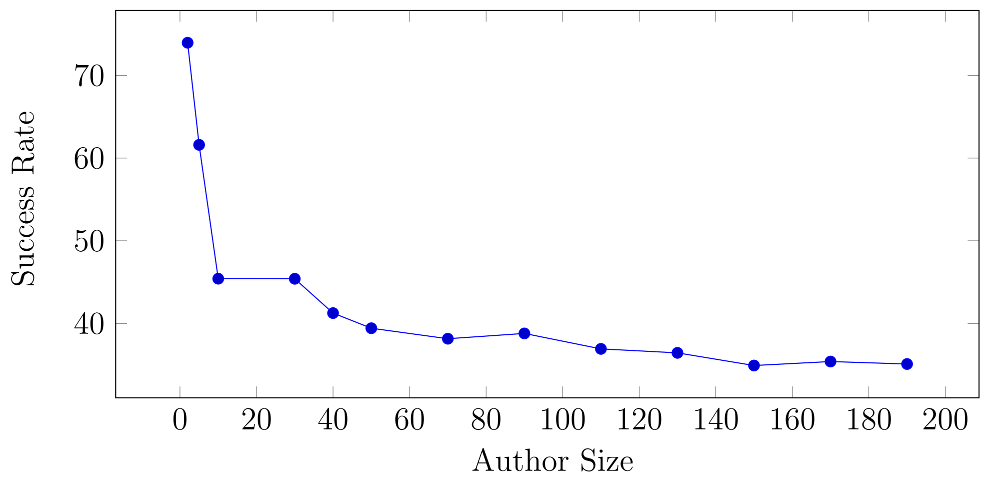

Figure 5.5: Changes on success rate with author size using merged classifier

Author identification is more successful with smaller samples. For example, with 2 authors our success rate is 73.95%. Choosing from a smaller sample and thus creating less models, inevitably increases success rate. We also determined that 70 authors, especially after 110 authors, the difference between test result decreases. This indicates that for larger author samples the success rate will not go down drastically.

5.3 Experiments with Multinominal NBC

In this section we list our experiments on Multinominal NBC. In the first sub-section we experiment on Synonym NBC. We do our experiments with various author sizes. In the second sub-section we test Synonym NBC by merging it with our previous classifiers.

5.3.1 Experiments with Synonym NBC

Synonym Naive Bayes Classifier is similar to Nominal NBC. Instead of a bag of words, it uses a bag of synonyms. We extract our synsets from BalkaNet wordNET and use them to represent our passages. We experiment on various author sizes.

We want to test isolated performance of Synonym NBC. For this experiment we used the following parameters : minimum passage count : 120 and maximum passage count : 310. We test our classifier with author size between 2 and 20 and 620 to 4460 passages.

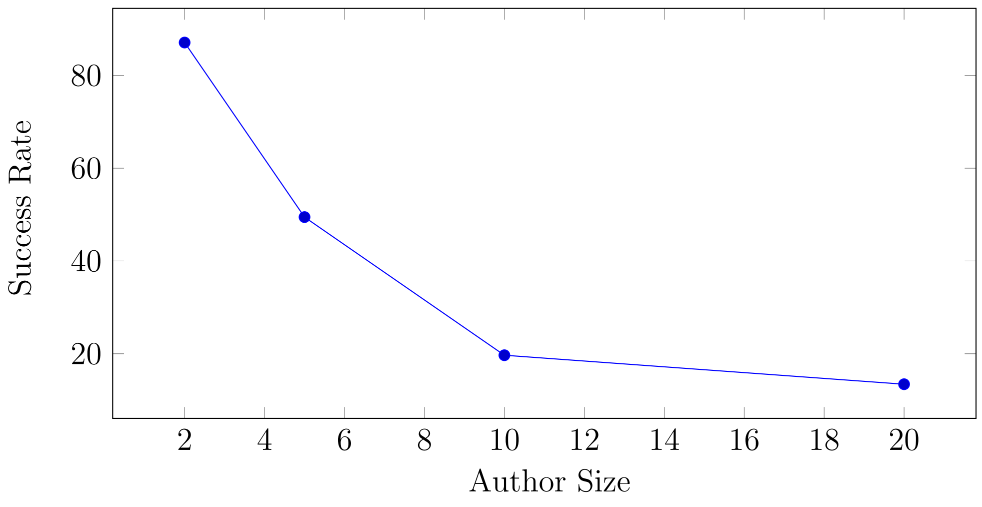

Figure 5.6: Success rate relative to author size using Synonym NBC

Synonym NBC and Nominal NBC are similar in the way we implement them. In Nominal NBC we used a bag of words as a feature vector. In Synonym NBC we are using synsets as a feature vector. We expect a similar or better result compared to Nominal NBC's relative scores. But compared to Nominal NBC, the results are considerably lower and computation time is a lot longer. The reason behind the poor results might be the synset source we are using. In our classification, we used BalkaNet's synset source. It is a general purpose synset but it might not include the necessary synsets our data needs. To test this reasoning, we count every word that we find in a relative synset. We find that for 69% of the words, we can not find their synset in BalkaNet. The 69% absence might explain why our success rates are low.

5.3.2 Experiment with all NBC Merged together

Merging NBC with numeric features and NBC with nominal features was a good step to increase classification success. We want to try to merge those two methods with multinominal NBC to increase success rate. We follow the same method as previous merging to calculate the final probability. For this experiment we use the following parameters; minimum passage count : 120, maximum passage count : 310, window size : 500, padding size : 125. We test our classifier with an author size between 2 and 20 and 620 to 4460 passages.

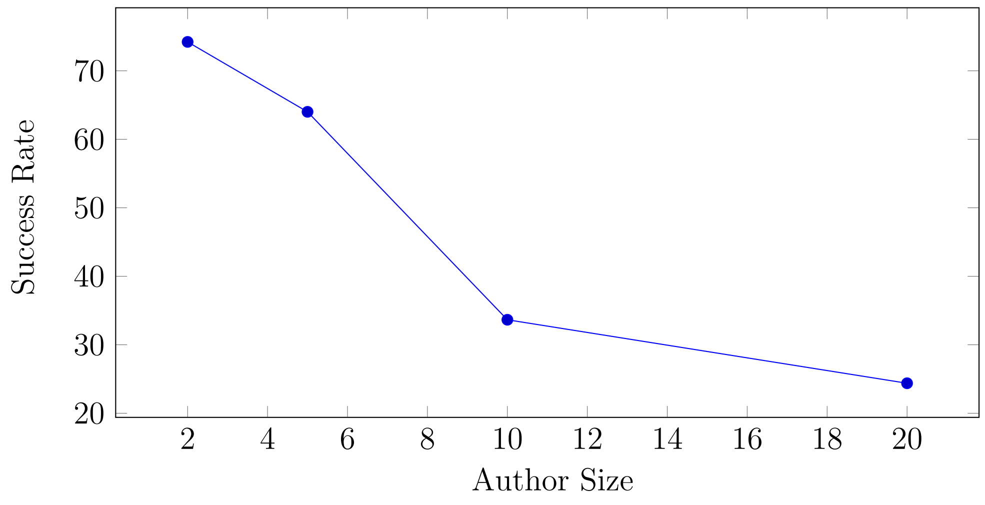

Figure 5.7: Success rate relative to author size using all classifiers merged together

Poor results from the isolated test of Synset NBC affects the merged classification results. 73.95% success rate increased to 74.22%. Synset NBC added little to no contribution to the merged success rate of Numeric NBC and Nominal NBC; for bigger author sizes, the success rate decreases rapidly. We think increasing the success rate of Synset NBC will contribute to the merged version. In order to do this, we need to increase the synset finding rate.

Chapter 6

Conclusion and Future Work

The study was set out to implement and compare Numeric NBC, Nominal NBC, and Multinominal NBC algorithms for author identification. The study has also sought to know whether merging individual classifiers together can bring better classification results. The study attempted to answer two questions:

1. How does each classification method perform? Which one is the most successful?

2. Does merging individual classification methods result with better classification results?

The experiments in Chapter 5 are listed by various classification methods. These are : Experiments with NBC in section 5.2.1, Experiments with Nominal NBC in section 5.2.2, Experiments with Merged Gaussian NBC in section 5.2.3, Experiments with Synonym NBC in section 5.3.1 and, Experiment with all NBC Merged together in section 5.3.2. We can answer the study's two research questions by synthesizing the experiment results.

1. How does each classification method perform? Which one is the most successful?

   In isolated experiments, Nominal NBC is the most successful classification method with 27.6% success rate. In our merged classification experiments, Merged Gaussian NBC is better compared with Nominal NBC with 34.95% success. Merging two Numeric NBC with Nominal NBC gives us the most successful result overall.

2. Does merging individual classification methods result is greater classification success?
   Merging Numeric NBC with Nominal NBC increased classification results compared to individual scores. But merging the two with Synonym NBC did not increase the classification success rate. Because of the low synonym vector extraction rate, as we mention in experiment 5.3.1, merging the three decreased overall classification success.

Our study shows that the use of a single method Nominal NBC is the best solution for author identification. With an exception of Synonym NBC method, merging multiple methods results in better classification rates.

The author identification problem is extensive and complicated. To increase classification success and examine an author's styles and representation, there is need for further research and work. Exploring the following as future research items can achieve this goal:

- Find a better resource for synonym feature extraction.
- Implement and experiment on merging different semantic classifiers.
- Implement and experiment with different classification algorithms other than NBC.

In support of what is often reported about the benefit of using Nominal NBC for text classification, merging Numeric NBC with Nominal NBC achieves a better classification scores.

References

[1] Wikipedia. The Federalist Papers — Wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/wiki/The_Federalist_Papers. [Online; accessed 22-April-2016].

[2] Brinegar Claude S. Mark Twain and the Quintus Curtius Snodgrass Letters: A Statistical Test of Authorship. *Journal of the American Statistical Association*, (58):85–96, 1963.

[3] Morton Andrew Q. The Authorship of Greek Prose. *Journal of the Royal Statistical Society, Series A*, (128):169–233, 1965.

[4] Brainerd Barron. Weighting Evidence in Language and Literature: A Statistical Approach. *University of Toronto Press*, 1974.

[5] Holmes David I. A Stylometric Analysis of Mormon Scripture and Related Texts, year = 1992. *Journal of the Royal Statistical Society*, (155):91–120.

[6] G. Kokkinakis N. Fakotakis, E. Stamatatos. Automatic Text Categorization in Terms of Genre and Author. *Journal Computational Linguistics*, 26(155): 471–495, 2000.

[7] Tufan Taş and Abdül Kadir Görür. Author Identification for Turkish Texts. *Journal of Arts and Sciences*, 26:151–161, 2007.

[8] Banu Diri and M. Fatih Amasyalı. Automatic Author Detection for Turkish Texts. ICANN, 2014.

[9] Jonathan Hedley. jsoup. https://github.com/jhy/jsoup.

[10] Ahmet A. Akın. Zemberek. https://github.com/ahmetaa/zemberek-nlp, 2005.

[11] Wikipedia. Statistical Classification — Wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/wiki/Statistical_classification. [Online; accessed 22-April-2016].

[12] BalkaNet. Balkanet. URL http://www.dblab.upatras.gr/balkanet/index.htm.

[13] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.