Sentiment Analysis by Emoticons and Unsupervised Comment Summarization in Greek e-Government data

Author: Stefanos Tziortzis

The purpose of this paper is to provide some tools for document mining in the Greek language, especially for policy makers who need a summary of the public’s opinions regarding specific or even ministry wide topics.

Keywords: rapidminer, text analytics, greek stemmer, opinion mining, text data mining, sentiment analysis, comment clustering, emoticons detection, comment summarization

University of the Aegean
Information and Communication Systems Department

Scientific Supervisor: Dr. Manolis Maragoudakis

Samos, November 2013
# Table of Contents

Acknowledgements ........................................................................................................... 2

Chapter 1. Abstract ........................................................................................................ 3

Chapter 2. Introduction .................................................................................................. 4

Chapter 3. Related Work ............................................................................................... 6

Chapter 4. Development Tools ..................................................................................... 13
  4.1 RapidMiner Basics ................................................................................................. 15

Chapter 5. Finding Word Similarities .......................................................................... 17
  5.1 Greek Stemmer ...................................................................................................... 23

Chapter 6. Sentiment Classification ............................................................................. 25
  6.1 Emoticons Marking Tool ...................................................................................... 25
  6.2 Gathering Data ...................................................................................................... 28
  6.3 Importing Data ...................................................................................................... 29
  6.4 Classification Process ......................................................................................... 31
  6.5 Classification Results ......................................................................................... 37
  6.6 Comparison with Control Dataset ....................................................................... 38

Chapter 7. Comment Summarization .......................................................................... 40
  7.1 Gathering Data from OpenGov.gr ...................................................................... 40
  7.2 Importing Data ...................................................................................................... 42
  7.3 Clustering Process ............................................................................................... 43
  7.4 Creating Word Clouds ......................................................................................... 47
  7.5 Clustering Results ............................................................................................... 49

Chapter 8. Automatic Comment Summarization Tool .............................................. 54
  8.1 Underlying processes ........................................................................................... 56

Chapter 9. Summary .................................................................................................... 60

Bibliography .................................................................................................................... 61
Acknowledgements

A big thank you goes to my sweetheart Loukia Keliri and her family for keeping my confidence high into completing this project, and for all their patience throughout this time. I know it has been an intense period, but I’m glad that I had you by my side... 😊

I would also like to thank my parents and brother for their support during my university years, my godmother and cousins for their pep talks in keeping me up to the task, my supervisor Dr. Manolis Maragoudakis for his guidance and aid especially during the final stages of the project, and Dr. Evripidis Loukis who directed me in choosing this topic.

Finally, I would like to thank all friends for being supportive, as well as all people who make an effort in making the world a better place...

Thank you all, you are awesome! ;
Chapter 1. Abstract

Policy makers need to know the public’s opinions regarding proposed new legislations, legislation amendments, and for various other topics of interest.

A platform for collecting the public’s comments already exists in the form of the Δι@ύγεια project aiming at a Greek Open Government and hosted at www.opengov.gr, but the tools for processing the Greek language are still quite limited and so there is a need for new tools that are able to provide summaries of the public’s opinions.

In this paper we will describe the following specifically developed tools:

- An Emoticons and Shouting Marking Tool, aimed at identifying emoticons and expressions of anger and shouting in comments
- A Greek Stemmer, aimed at finding the stems of Greek words based on word comparisons
- A Sentiment Classification process, aimed at preparing a model that is based on human pre-training for providing sentiment summaries regarding a topic
- A Comment Summarization process using Clustering
- An Automatic Comment Summarization tool based on the above Clustering process, which offers a user-friendly interface targeted mainly towards end-users.
Chapter 2. Introduction

**Data mining** is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of data mining is to extract information from a data set and transform it into an understandable structure for further use.

**Classification** is considered an instance of supervised learning which is the task of inferring a function from labeled training data, i.e. learning through a training set of correctly identified observations. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. Therefore classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. The individual observations are analyzed into a set of quantifiable properties, known as variables, features, etc. These properties may be categorical (e.g. "A", "B", "AB" or "O", for blood type), ordinal (e.g. "large", "medium" or "small"), integer-valued (e.g. the number of occurrences of a word in an email) or real-valued (e.g. a measurement of blood pressure). A classification example could be assigning a given email into "spam" or "non-spam" classes, or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). An algorithm that implements a classification process is known as a classifier.

**Clustering** (or cluster analysis) is the corresponding unsupervised procedure, and involves grouping data into categories based on some measure of inherent similarity (e.g. the distance between instances, considered as vectors in a multi-dimensional vector space). In machine learning, the problem of unsupervised learning is that of trying to find hidden structure in unlabeled data, i.e. grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. And since the examples given to the learner are unlabeled, there are no error or reward signals to evaluate a potential solution.
Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. The attitude may be the author’s judgment or evaluation, the emotional state when writing, or the emotional effect he wishes to have on the reader.

A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature level, i.e. whether the expressed opinion in a document, a sentence or an entity feature is positive, negative, or neutral. Advanced, "beyond polarity", sentiment classification, looks for instance at emotional states such as "angry," "sad," and "happy."

The accuracy of a sentiment analysis system is, in principle, how well it agrees with human judgments. This is usually measured by precision and recall. However, according to research human raters typically agree 79% of the time, thus a 70% accurate program is doing nearly as well as humans, even though such accuracy may not sound impressive. If a program were "right" 100% of the time, humans would still disagree with it about 20% of the time, since they disagree that much about any answer.

The problem with most sentiment analysis algorithms is that they use simple terms to express sentiment about a product or service. However, cultural factors, linguistic nuances and differing contexts make it extremely difficult to turn a string of written text into a simple pro or con sentiment. To make matters worse, the fact that humans often disagree on the sentiment of a text illustrates how big a task it is for computers to get this right. The shorter the string of text, the harder it becomes.

Emoticons are pictorial representations of a facial expression which in the absence of body language serve to change or improve the interpretation of the sender’s verbal communication. The term emoticon is a combination of the words emotion and icon. Emoticons are usually expressed by means of punctuation marks, but a person’s feelings or mood can include numbers and letters as well. In the most recent years as the epidemic of social media and texting is at an all-time high, emoticons play a significant role in communication through technology. These emoticons offer another range of "tone" and feeling, through texting that portrays specific emotions through facial gestures while in the midst of cyber communication.

Western style emoticons usually have the eyes on the left, followed by a nose and the mouth. The two character version :) which omits the nose is also very popular.

Eastern style emoticons are a type of emoticons that can be understood without tilting one's head to the left, which were popularized in Japan. These emoticons are usually found in a format similar to (*_*) where the asterisks indicate the eyes, the central character which is usually an underscore represents the mouth, and the parentheses represent the outline of the face.
Chapter 3. Related Work

- “Sentiment Analysis and Opinion Mining”, by Bing Liu

In his book, Bing Liu introduces the field of sentiment analysis and opinion mining and surveys the current state-of-the-art. Due to many challenging research problems and a wide variety of practical applications, the research in the field has been very active in recent years and has spread from computer science to management science, as opinions about products are closely related to profits. The book first defines the sentiment analysis problem, which provides a common framework to unify different research directions in the field. It then discusses the widely studied topic of document-level sentiment classification which aims to determine whether an opinion document, for example a review, expresses a positive or negative sentiment. This is followed by the sentence-level subjectivity and sentiment classification, which determines whether a sentence is opinionated and, if so, whether it carries a positive or negative opinion. The book then describes aspect-based sentiment analysis which explores the full power of the problem definition and shows that sentiment analysis is a multi-faceted problem with many challenging sub-problems, and the existing techniques for dealing with them are discussed. After that, the book discusses the problem of sentiment lexicon generation for which two dominant approaches are covered. This is followed by a chapter on opinion summarization, which is a special form of multi-document summarization, however it is also very different from the traditional multi-document summarization because opinion summarization can be done in a structured manner which facilitates both qualitative and quantitative analysis and visualization of opinions. Chapter 8 discusses the problem of analyzing comparative and superlative sentences, where such sentences represent a different type of evaluation from regular opinions which have been the focus of the current research. In Chapter 9 the topic of opinion search or retrieval is introduced, and last but not least in Chapter 10 opinion spam detection is discussed and in Chapter 11 the quality of reviews is assessed where opinion spamming by writing fake reviews and posting bogus comments are increasingly becoming an important issue as more and more people are relying on the opinions on the Web for decision making. To ensure the trustworthiness of such opinions, combating opinion spamming is an urgent and critical task.

In Chapter 2 regarding the Problem of Sentiment Analysis, Liu defines at first the concept of opinion in the context of sentiment analysis, then the main tasks of sentiment analysis, and lastly the framework of opinion summarization. Along with them, he introduces the relevant and important concepts of subjectivity and emotion, which are highly related but not equivalent to opinion. However he explains that these concepts and their definitions are rather fuzzy and subjective, and to support this he states that there is still no set of emotions that all researchers agree upon and that opinion itself is also a broad concept. Sentiment analysis mainly deals with the evaluation type of opinions or with opinions which imply positive or negative sentiments.

In Chapter 3 regarding Document Sentiment Classification, Liu states that sentiment classification at the document level provides an overall opinion on an entity, topic or event,
and although this has been studied by a large number of researches, however this level of classification has some shortcomings regarding applications where the user needs to know additional details as for example what aspects of entities are liked and disliked by consumers. Such details are provided in typical opinion documents, but document sentiment classification doesn’t extract them for the user. Another shortcoming is that document sentiment classification is not easily applicable to non-reviews such as forum discussions, blogs and news articles, because many such postings can evaluate multiple entities and compare them. He argues that in many cases it is hard to determine whether a posting actually evaluates the entities that the user is interested in, and also whether the posting expresses any opinion at all, let alone to determine the sentiment about them. He states that document-level sentiment classification doesn’t perform such fine-grained tasks which require in-depth natural language processing and that in fact online reviews don’t need sentiment classification because almost all reviews already have user-assigned star ratings. He finds that in practice it’s the forum discussions and blogs that need sentiment classification to determine people’s opinions about different topics and entities as for example products and services.

In Chapter 4 regarding Sentence Subjectivity and Sentiment Classification, Liu states that sentence-level subjectivity classification and sentiment classification goes further than document-level sentiment classification as it moves closer to opinion targets and sentiments on the targets. He says that this can be regarded as an intermediate step in the overall sentiment analysis task, although it still has several shortcomings for many real-life applications such as the fact that in most applications the user needs to know additional details like what entities or aspects of entities are liked and disliked, details which the sentence-level analysis still doesn’t provide. Another shortcoming is that although one may say that if we know the opinion targets (like for example the entities and aspects, or topics), we can assign the sentiment orientation of a sentence to the targets in the sentence, however this is insufficient for three different reasons. First because many complex sentences have different sentiments on different targets, like for example “Trying out Chrome because Firefox keeps crashing” and “Apple is doing very well in this lousy economy.” In this latter sentence, even the clause-level classification is insufficient and we need to go to the opinion target or the aspect level. Secondly, although a sentence may have an overall positive or negative tone, some of its components may express opposite opinions like for example the sentence “Despite the high unemployment rate, the economy is doing well.” which some researchers regard as positive and it’s true that the overall tone of this sentence is positive or at least the author is trying to emphasize on the positive side, but it still does contain a negative sentiment on the unemployment rate, which we must not ignore. If one goes to the aspect-level sentiment analysis, the problem is solved in the way that the sentence will be positive about the overall economy but negative about the unemployment rate. The third reason is that sentence level sentiment classification cannot deal with opinions in comparative sentences, like for example “Coke tastes better than Pepsi.” In this case we need different methods to extract and to analyze comparative opinions because they have quite different meanings from regular opinions. Although this
last sentence clearly expresses an opinion, one cannot simply classify the sentence as being positive, negative or neutral.

In Chapter 5 regarding Aspect-based Sentiment Analysis, Liu points that aspect-level sentiment analysis usually is the level of details required for practical applications, and that most industrial systems are based on this. But although a great deal of work has been done in the research community and the fact that many systems have been built based on aspect-level sentiment analysis, the problem is still far from being solved and every sub-problem remains to be highly challenging. The two most outstanding problems are aspect extraction and aspect sentiment classifications, where for both these problems the accuracies are not high because existing algorithms are still unable to handle complex sentences that require sentiment words and simple parsing, and are also unable to handle factual sentences that imply opinions. On the whole, it seems to be a long tail problem. While sentiment words can handle about 60% of the cases (depending on the domains), the rest are highly diverse, numerous and infrequent, which makes it hard for statistical learning algorithms to learn patterns because there simply aren’t enough training data for them. In fact there seems to be an unlimited number of ways that people can use to express positive or negative opinions, and every domain appears to have something special. So far the research community has mainly focused on opinions about electronics products, hotels and restaurants, but these domains are easier (although not easy) and reasonably good accuracies can be achieved if one can focus on each domain and take care of its special cases. When one moves to other domains though, the situations get considerably harder because in these domains many factual statements imply opinions. Politics is an exceptionally difficult case, where the current aspect extraction algorithms only had limited success because few political issues (aspects) can be described with one or two words. Political sentiments are also harder to determine due to a complex mixture of factual reporting and subjective opinions, and due to their heavy use of sarcastic sentences. In terms of social media type, researchers working on aspect-based sentiment analysis have focused mainly on product/service reviews and tweets from Twitter. These forms of data are also easier (but not easy) to handle, because reviews are opinion-rich and have little irrelevant information while tweets are very short and often straight to the point. However, other forms of opinion text such as forum discussions and commentaries are much harder to deal with because they are mixed with all kinds of non-opinion contents and often talk about multiple entities and involve user interactions. This leads to another major issue which is the data noise, for which Liu states that there is limited research. Almost all forms of social media are very noisy (except reviews) and full of all kinds of spelling, grammatical, and punctuation errors, but most Natural Language Processing tools such as Part-Of-Speech taggers and parsers need clean data to perform accurately, leading to the need for a significant amount of pre-processing before analysis.

In Chapter 6 regarding Sentiment Lexicon Generation, Liu states several general-purpose subjectivity, sentiment and emotion lexicons that have been constructed from contributions of many researchers, some of which are publicly available such as the General Inquirer lexicon, the Sentiment lexicon, the MPQA subjectivity lexicon, the
SentiWordNet, and the Emotion lexicon. However he notes that domain and context dependent sentiments remain to be highly challenging even with so much research. He also points that recent work has also used word vector and matrix to capture the contextual information of sentiment words, while on the other hand factual words and expressions implying opinions have barely been studied but are very important for many domains. As a final note he states that having a sentiment lexicon (even with domain specific orientations) doesn’t mean that a word in the lexicon always expresses an opinion/sentiment in a specific sentence, like for example in “I am looking for a good car to buy” where “good” doesn’t express either a positive or negative opinion on any particular car.

In Chapter 7 regarding Opinion Summarization, Liu states that opinion summarization is still an active research area and that most opinion summarization methods which produce a short text summary have not focused on the quantitative side. He also notes that future research on opinion summarization depends critically on results and techniques from other areas of research in sentiment analysis, such as aspect or topic extraction and sentiment classification, and that all these research directions will need to go hand-in-hand.

In Chapter 8 regarding Analysis of Comparative Opinions, Liu mentions that although there have been some exciting works, comparative sentences have not been studied as extensively as many other topics of sentiment analysis and that further research is still needed. One of the difficult problems is how to identify many types of non-standard or implicit comparative sentences, like for example “I am very happy that my iPhone is nothing like my old ugly Droid.” Without identifying the comparative sentences, further sentiment analysis is hard to perform. Apart from identifying comparative sentences and their types, several researchers have also studied the extraction of compared entities, compared aspects, and comparative words, however their work is limited in the sense that it only works with simple comparative questions.

In Chapter 9 regarding Opinion Search and Retrieval, Liu remarks that it will be really useful if a Web search engine such as Google or Microsoft Bing can provide a general opinion search service. Although both Google and Microsoft Bing already provide opinion summarization services for reviews of some products, their coverage is still very limited. It is not easy to find opinions for those not covered entities and topics, because their opinions are scattered all over the Internet. Although there are also some large and well known review hosting sites such as Amazon.com and Yelp.com, however they do not cover all entities and topics either. Finding opinions about those not covered entities or topics remains a formidable task because of the proliferation of diverse sites and the difficulty of identifying relevant opinions, but a lot of research is still needed before a breakthrough can be achieved.

In Chapter 10 regarding Opinion Spam Detection, Liu notes that as social media is increasingly used for critical decision making by organizations and individuals, opinion spamming is also becoming more and more widespread. For many businesses, posting fake opinions themselves or employing others to do it for them has become a cheap way of marketing and brand promotion. Although current research on opinion spam detection is
still in its early stage, several effective algorithms have already been proposed and used in practice. Spammers however are also getting more sophisticated and careful in writing and posting fake opinions to avoid detection. In fact there is already an arms race between detection algorithms and spammers. Liu also notes that opinion spamming occurs not only in reviews, but also in other forms of social media such as blogs, forum discussions, commentaries, and Twitter postings, but so far little research has been done in these contexts.

In Chapter 11 regarding the Quality of Reviews, Liu says that in summary the task of determining review helpfulness is an important research topic that is especially useful for products and services that have a large number of reviews. Liu supports the opinion that both quality and distribution (in terms of positive and negative viewpoints) are important. Liu also notes that readers tend to determine whether a review is helpful or not based on whether the review expresses opinions on many aspects of the product and appears to be genuine, but spammers can satisfy this requirement by carefully crafting reviews that are just like normal helpful reviews. With that in mind, using the number of helpfulness feedbacks to define review quality, or even considering that alone as the ground truth, can be problematic. Furthermore, the user feedbacks can also be spammed by a robot or human spammer who clicks on the helpfulness feedback button to increase the helpfulness of a review.

In the Concluding Remarks of his book, Liu concludes that sentiment analysis is technically very challenging. Although the research community has attempted so many sub-problems from many different angles and a large number of research papers have also been published, none of the sub-problems has been solved satisfactorily. The understanding and knowledge about the whole problem and its solution are still very limited, and the main reason is that this is a Natural Language Processing task, and Natural Language Processing has no easy problems. Another reason may be due to relying too much on machine learning since some of the most effective machine learning algorithms such as Support Vector Machines, Naïve Bayes and Conditional Random Fields, produce no human understandable results such that although they may help in achieving improved accuracy, we know little about how and why apart from some superficial knowledge gained in the manual feature engineering process. However Liu recognizes that significant progresses have been made over the past decade and this is evident from the large number of start-up and established companies that offer sentiment analysis services. There is a real and huge need in the industry for such services because every business wants to know how consumers perceive their products and services and those of their competitors. The same can also be said about the consumers because whenever one wants to buy something, he wants to know the opinions of existing users. These practical needs and the technical challenges will keep the field vibrant and lively for years to come. Building on what has been done so far, it is Liu’s belief that we just need to conduct more in-depth investigations and to build integrated systems that try to deal with all the sub-problems together because their interactions can help solve each individual sub-problem, and he seems optimistic that the whole problem will be solved satisfactorily in the near future for widespread applications. Currently, a
completely automated and accurate solution is nowhere in sight, but it is possible to devise effective semi-automated solutions. The key is to fully understand the whole range of issues and pitfalls, cleverly manage them, and determine what portions can be done automatically and what portions need human assistance. And a good bet would be to work hard on a large number of diverse application domains, understand each of them, and design a general solution gradually.

- “Supervised Machine Learning: A Review of Classification Techniques” by S. B. Kotsiantis, University of Peloponnese

This paper describes various supervised machine learning classification techniques. Supervised machine learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown. Of course, the author acknowledges that a single article cannot be a complete review of all supervised machine learning classification algorithms, yet he hopes that the references cited will cover the major theoretical issues, guiding the researcher in interesting research directions.

In the conclusions of the paper, the author suggests that the application of ensemble models is suggested only if one is interested in the best possible classification accuracy, and gives three reasons for that. Despite the obvious advantage that ensemble clustering has over single classifiers, which is the utilization of the strengths of one method to complement for the weaknesses of another, the first weakness of ensemble models is increased storage requirements which depend on the size of each component classifier itself and the number of classifiers in the ensemble. The second weakness is increased computations, because in order to classify an input query, all component classifiers must be processed instead of a single classifier. The last weakness is decreased comprehensibility due to the involvement of multiple classifiers in the decision-making, making it more difficult for non-expert users to perceive the underlying reasoning process leading to a decision. The wrapper feature selection procedure which is another time-consuming attempt that tries to increase the classification accuracy without decreasing comprehensibility, has shown through practical experience that having more features doesn’t always result in more discriminating power. Finally, for the database community that deals with gigabyte databases where the requirement by most of the current learning algorithms for all data being resident in the main memory is clearly unattainable, the author suggests an orthogonal approach to partition the data, thus avoiding the need to run algorithms on very large datasets, by distributed machine learning which involves breaking the dataset into subsets, learning from these subsets concurrently and then combining the results. For this parallel execution of machine learning processes, distributed agent systems can be used where it is the responsibility of the agents to integrate the information from numerous local sources in collaboration with other agents.
In this report, the authors present an empirical study on the efficacy of machine learning techniques in classifying text messages by semantic meaning. Using movie review comments from the popular social network Digg as the dataset, they classify text by subjectivity/objectivity and negative/positive attitude. Different approaches are proposed in extracting text features such as the bag-of-words model, using large movie reviews corpus, restricting to adjectives and adverbs, handling negotiations, bounding word frequencies by a threshold, and using WordNet synonyms knowledge. The effect of the above on accuracy is evaluated for four machine learning methods, namely Naive Bayes, Decision Trees, Maximum-Entropy, and K-Means clustering. In the conclusion of this study are explanations of the observed trends in accuracy rates, and directions for future work.

The results from the sentiment analysis on social network comments using comments on articles from Digg as the text corpora, show that the simple bag-of-words model can perform relatively good, and that it can be further refined by the choice of features based on syntactic and semantic information from the text.
Chapter 4. Development Tools

There are a number of tools that you can use for data mining, both commercial and free. Here are some of the most well-known free tools, including a short description for each:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RapidMiner (Community edition)</td>
<td>is a data mining software. You can use RapidMiner as a stand-alone application for data analysis, or integrate it as a data-mining engine into your own products. Features:</td>
</tr>
<tr>
<td></td>
<td>- Data integration, analytical ETL, data analysis and reporting into a single suite</td>
</tr>
<tr>
<td></td>
<td>- Powerful yet intuitive GUI (Graphical User Interface) for the design of analytical processes</td>
</tr>
<tr>
<td></td>
<td>- Repository for process, data and metadata management</td>
</tr>
<tr>
<td></td>
<td>- The only solution with metadata transformation: Forget trial and error and inspect results already at design time</td>
</tr>
<tr>
<td></td>
<td>- The only solution that supports on-the-fly error detection and quick fixes</td>
</tr>
<tr>
<td></td>
<td>- Complete and flexible: Hundreds of methods for data integration, data transformation, modeling and visualization</td>
</tr>
</tbody>
</table>

Knime is a java open-source, cross-platform application which name means "Konstanz Information Miner". It is actually used extensively for data mining, data analysis and optimization. It can be downloaded as the core application itself (Knime Desktop), or the whole SDK which is based on Eclipse Helios. The knime software can also work with different kinds of extensions which are embedded into the "/downloads/extensions" tabs of the website.
Orange is an Open-source, cross-platform data mining and machine learning suite. It features visual programming and Python interface, Qt (C++) and python were used as the programming languages of choice. With many functionalities aboard, this software can make data management easier for novice and expert users.

Weka is a collection of machine learning algorithms for data mining tasks; with its own GUI. (The application is named after a flightless bird of New Zealand that is very inquisitive.) The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

For making our own software we chose the Java Development Toolkit (JDK) and the NetBeans IDE (Integrated Development Environment).

For making the data mining processes, our tool of choice was RapidMiner, which is also written in Java.

Finally, for creating the word clouds we used a web tool called Wordle, which also happens to be written in Java.
4.1 RapidMiner Basics

Regarding RapidMiner, which is our data mining tool of choice, we recommend that as soon as you install and run it to install some additional extensions. You can do that by going through “Help—Updates and Extensions (Marketplace)”, and then at the “Top Downloads” tab select everything there and install it. For each one of those extensions there is a description on the right if you want to know what you are installing.

You are advised to at least install the Text Mining extension, but you might also want to consider other useful extensions such as the extensions for Web Mining, Anomaly Detection, Weka, Reporting, Parallel Processing, you get the idea. Most of what is in there is useful.

If you want to search for an extension that might not be listed in the Top Downloads, you can do that by using the Search tab.
After the program is restarted with the extensions now installed, in order to begin something new we start a New Process. But first, let’s explain RapidMiner’s interface a bit.

So starting from the left, we have a list of operators which are essentially the functions that RapidMiner provides us to do various kinds of stuff. If you know the name of an operator you want to use, you can start typing it in the search area and the operators will be filtered accordingly as you type. At the top of that area you can see that there is a Repositories tab. What is a repository? Repositories are essentially RapidMiner’s save files, and each repository might include either data, or a process, or a model, etc.

In the center area is where you will essentially design your process. It’s as simple as dragging and dropping operators from the left into the center area. And it’s quite trivial to connect two operators, simply click on each operator’s ends (called pipes) to connect them.

If you want to change an operator’s parameters, first you have to select the operator that you want to tweak, and then you can see the available parameters on your right. For help on pretty much any parameter, simply leave your mouse pointer over that parameter for a few seconds and RapidMiner will bring up a helpful balloon tip with more information.

You can execute a process by pressing the play button at the top. You can also pause and stop a process, but note that RapidMiner will only stop after finishing the current operator.

When a process finishes, RapidMiner automatically asks to switch to the Results Perspective in order to display the results of the process.

To go back to the Design Perspective and make improvements on a process, there are two handy buttons right next to the process execution area, that switch between these two Views, namely Design Perspective and Results Perspective.

If any problems arise during the execution of a process, descriptions of the possible issues will appear at the bottom area.
Chapter 5. Finding Word Similarities

At this process we calculate the similarities between words, in order to use them to help us later on in finding words sharing the same stems.

Figure 4 - The word similarities extraction process

The **Retrieve** operator can retrieve whatever we have stored at a repository. Once we select the repository in which we have stored our data, we pass these data as input to the **Process Documents from Data** operator.

The **Process Documents from Data** operator is used to create word vectors from text attributes. This is a nested operator which you can tell from the blue icon at the bottom right. First let’s configure this operator’s parameters:

- **Create word vector** uses the tokens (words) of each document to generate a vector numerically representing each document.
- **Keep text** keeps the input text as a special String attribute.
- **Prune method** specifies if too frequent or too infrequent words should be ignored for building the word list, as well as the how the frequencies should be specified.

We chose a **percentual** pruning method, which ignores words according to their percentage of appearance in all documents.

We chose to prune words with frequencies below 1%, which for our classification dataset of about 230 examples means that all words that don’t appear at least twice are to be ignored.

As we don’t wish to prune words that occur above a percentage in our dataset, we leave the prune above setting at 100%.

Finally we left the **vector creation** parameter for last, in order to explain it a bit further.
Through the **vector creation** parameter we can select the schema for creating the word vector, and it has various options with the default being TF-IDF. So what is TF-IDF?

**TF-IDF** stands for term frequency–inverse document frequency. It is a numerical statistic which reflects how important a word is to a document in a collection, and it is often used as a weighting factor. The number of times a term occurs in a document is called its **term frequency**. The purpose of the **inverse document frequency** factor is to diminish the weight of terms that occur very frequently in the document set. So the TF-IDF value increases proportionally to the number of times a word appears in a document, but is offset by the frequency of the word in the collection, which helps to control for the fact that some words are generally more common than others.

Now then, since Process Documents from Data is a nested operator, let’s add a sub-process.

**Nested operators**, which are operators that feature sub-processes, are marked in their bottom right corners with the symbol of two blue overlapping windows. To go into an operator’s subprocess, simply double-click the nested operator. Once inside, you can go back through the back and up buttons on top of the subprocess, and you also have a clickable path showing where you are at.

The sub-process inside our Process Documents from Data operator is the following:

![Process Documents from Data sub-process for breaking texts into words](image)

The **Transform Cases** operator transforms the cases of all characters. In its parameters we choose to transform all characters to lower case.

The **Tokenize** operator tokenizes documents, and we select in the parameters of this operator to tokenize at non letters so that each time a non-letter is found it shall denote a new token, therefore splitting a text into words.

The **Filter Tokens (by Length)** operator, filters tokens based on their length. In its parameters we select the min chars of a token to be 2 (thus removing single letter words), and the max chars of a token to be 100 which is safe enough to say that words consisting of 100 chars are probably gibberish.

The **Filter Stopwords (Dictionary)** operator applies a stopword list from a file.

**Stopwords** are words which are filtered out prior to, or after, processing of natural language data (text). There is not one definite list of stopwords which all tools use. Any group of words can be chosen as stopwords for a given purpose. For example, some of the most common stopwords for search machines are: *the, is, at, which*, and on.
In our case we used a variety of stopword sources mostly for the English language, which are listed in the bibliography, and then we selected mostly the stopwords that were present in at least two sources, translated them to Greek, evaluated their meanings in Greek, and added any related words and synonyms that also carry mostly trivial information.

After saving the stopwords in a *.txt file, we then load it through the parameters of the Filter Stopwords (Dictionary) operator.

Lastly, it would have been ideal to be able to add a Stem operator to the sub-process, but neither the Stem (WordNet) nor the Stem (Snowball) operators support Greek, so we are stuck... Or are we? Let’s talk a bit first about what stems are, and their differences to similar terms like roots and lemmas.

A root is the primary lexical unit of a word which carries the most significant aspects of semantic content and cannot be reduced into smaller constituents.

A stem is the part of the word that is common to all its inflected variants. Thus all derivational affixes are part of the stem. For example, the stem of friendships is friendship, to which the inflectional suffix -s is attached.

An example to tell the difference between a root and a stem is this: The root of the English verb form “destabilized” is stabil-, but on the other hand the stem is destabilize, which includes the derivational affixes de- and -ize but not the inflectional past tense suffix -(e)d.

A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words. In English, for example, run, runs, ran and running are forms of the same lexeme, with run as the lemma. Lexeme, in this context, refers to the set of all the forms that have the same meaning, and lemma refers to the particular form that is chosen by convention to represent the lexeme.

Ideally then, we would want to build a Lemmatizer, but as this is far more complex, and up to a certain degree the same applies to finding the root of a word, the simpler choice is to build a Stemmer. And this is exactly our next step, for which we first need some preparation.

The last operator from our previous sub-process is the Replace Tokens operator which replaces all occurrences of all specified regular expressions within each token by the specified replacement.

In the parameters of this operator we are given the option to make a list, and we chose to add replacements for all intonated vowels. It is generally not advised to remove Greek intonations as some (mostly smaller) words might lose completely their meanings. We did however choose to remove all intonations, because in our pursue to find the word stems, a problem we are facing is that the intonations frequently change their positions in the different forms of the same word, so we remove them. Here below are the replacements:
Back to our main process, we now want to take the wordlist created by the Process Documents from Data operator and manipulate it, so we use the **WordList to Data** operator to convert the wordlist to a dataset.

**Note:** Using **Breakpoint After**, you can see the results after a certain operator in a process. Just right-click on the operator after which you want to see results, and select Breakpoint After or press the F7 key on your keyboard. Each time you press the Play button to execute the process, it will stop and display the results after the execution of that operator, and each consecutive time it will continue to the next Breakpoint until the final output of the process.

The output so far should be something similar to this. From all these attributes, the only attribute that we need for our stemmer is the one containing the words themselves, so we shall remove the rest.

Using the **Select Attributes** operator, we can select which attributes should be kept and which attributes shall be removed. From its parameters we choose that we want to keep a single attribute, the **word** attribute (simply type it in).

By executing the process so far, we discover that unfortunately the WordList to Data operator assumed that the datatype for the **word** attribute should be polynomial, so we need to change it into text in order to be able to further process the words.
The **Nominal to Text** operator changes the datatype of the selected nominal attributes into text. The defaults should be fine as we don’t have any other attributes in the dataset besides the one that we want to change into text.

As we have already explained, the **Multiply** operator does nothing more than copy its input object to all connected output ports without modifying it.

From the Multiply operator we take two copies of the input data.

Using a **Write CSV** operator, we export the first copy of the words to a *.csv file (comma separated file), which is a format that is supported by Excel. Inside this file are the words for which we will try to find their similarities.

We then direct the second copy of the words from the Multiply operator, into a second **Process Documents from Data** operator.

This time though we don’t want to use any pruning method in the parameters, because we want to process all of the words from our wordlist without filtering-out any of them.

Inside the sub-process of this operator, we only use a **Tokenizer** operator. But this time we want to tokenize the words based on a different criterion.

This time we want to tokenize the words letter by letter. One way to do this is by using regular expressions, so from the parameters of the Tokenize operator we choose the regular expression mode and the expression we use is the vertical bar | symbol.

Using the **Data to Similarity** operator, we measure the similarity of each example with all other examples. Through its parameters we set the measure types to **Numerical Measures** and then select the **Cosine Similarity** measure.

Using the **Similarity to Data** operator we can calculate an exampleset from the given similarity measure. **(Note: Although the second output of the Data to Similarity operator is an exampleset, it is only its input passing through).**

Using the **Sort** operator we then sort the exampleset based on the SIMILARITY attribute, which we want in descending order. As always, we do that through the parameters of the operator.

Finally, we write the second output, containing the word similarities, into another *.csv file.
The above are the two outputs that we wrote into the two separate *.csv files.

The first output file contains the words, and the second contains the word similarities.

We now need a tool to help us find the stems of similar words, by combining these two files. And this is exactly what we have built, once again with the help of Java and NetBeans.
5.1 Greek Stemmer

This Stemmer provides rules based on word comparisons and not based on lexical rules.

It needs two *.csv files as input, one containing the similarities of the words we want to stem and the other containing the words themselves.

Note that the similarities file might take some time to load. The reason for this is that a filtering calculation is taking place while reading this file. The Data to Similarity operator of RapidMiner calculated the similarity of let’s say words A and B, but the Similarities to Data operator also added the similarity of words B and A, which is essentially identical to the similarity of words A and B and so is redundant. So in order to reduce this redundancy, we pre-filter each row so that only rows with columns FIRST_ID < SECOND_ID are accepted.
**Minimum Similarity of words:** A minimum threshold for two words to be considered similar.

**Maximum Word ID Distance:** The maximum alphabetical distance of the two words.

Because we consider two words to be more similar if they share about the same letters, letter-based similarity matching is prone to anagrams, i.e. words that share about the same letters but in a different order.

As an example, the words «μεινουν» and «μενουν» have a high similarity because they differ by only one letter, but their differences aren’t near their ends, thus making them improbable for having the same stem. As another example, the words «αρχη» and «χαρη» share exactly the same letters, but are only anagrams with completely different meanings.

The solution to this problem is the Maximum Word ID Distance filter, which takes advantage of the fact that words are alphabetically sorted, so two words that are not alphabetically adjacent (reflected by their ID’s distance), should not be considered to have the same stem.

**Minimum Stem Length:** The minimum number of characters for each stem.

The reasons for which we want a minimum stem length are, first of all because any stem consisting of less than 2 letters cannot in most cases be considered a stem at all, and secondly because the smaller the stems are, you risk considering too general stems as valid, therefore merging words with different meanings as if they convey the same meaning.

Some examples are «δημοσ.» which can be a stem for both «δημοσιος» and «δημοσκοπηση», which are completely different words, or «κεφαλ.» which can be a stem for both «κεφαλαιο» and «κεφαλι» again being totally different words.

Moreover, stems with less than 6 letters don’t always convey reliably enough the word from which they originated, misleading human readers to assume that the stem corresponds to a word which isn’t present in the exampleset. For example the stem «προσφ.» can be the stem for the verb «προσφερει», but reading the stem «προσφ» out of context, someone can assume it’s the stem for a completely different word such as «προσφυγας».

The **Apply Restrictions** button simply applies those filters and shows the resulting stems.

The **Defaults** button resets the filters values to 0.88 for Minimum Similarity of Words, 2 for Maximum Word ID Distance, and 6 for Minimum Stem Length. These values are only empirical and were selected through a rather defensive approach.

By applying the restrictions, all rows that pass the two filters on the left are highlighted in green, and the minimum stem length filter is applied directly to the resulting stems.

The stemming algorithm is also smart enough to filter out all ending vowels from each stem, so all stems end at a consonant. The only exception to this rule are the letter combinations «αυ» and «ευ», of which the second letter is read as a consonant and therefore is accepted as a valid stem ending, e.g. the stem «δουλευ.» taken from the verb «δουλευω». 
Chapter 6. Sentiment Classification

6.1 Emoticons Marking Tool

Our first task is to identify emoticons present in people’s comments, so that later-on we can try to better classify the general sentiment of a particular comment (positive or negative) based on both the words and the emoticons that the person used in that comment.

So then, we need to build a tool that will mark the emoticons for us, and depending on what kind of emoticons it detects, separate them into some predefined categories.

Because the reason we needed the emoticons to begin with, was to help us better identify the sentiment of any given comment containing emoticons, this idea has been expanded to also include all angry and shouting occurrences. By angry we also consider any symbols that might denote cursing, like grawlixes (e.g. %^#&$ @$hole), and by shouting we consider anything written in all capital letters.

In order to make the above possible, we used NetBeans IDE, an integrated development environment that helps you develop all sorts of Java applications, but you need to install the Java Development Kit (JDK) first to be able to develop your own Java applications. In short, Java is a cross-platform (meaning regardless of Operating System) programming language.

The easiest way to design functional but also great looking apps in Java, is through the NetBeans GUI (Graphical User Interface). To help you get started, here is a NetBeans GUI introduction, a tutorial, and the basics you need to know regarding the interface.

Back to our program, we also need a way to detect character sequences in order to be able to mark those parts of a comment that are of interest to our emoticons and shouting marking tool. The easiest way to do this is by using REGEX, short for Regular Expressions. If you are interested to learn more, here is a very useful webpage containing a REGEX Quick Start and Tutorial.

Finally, we need a list of emoticons to help us identify what is of interest to our tool. In general we have two major categories of emoticons, Western style and Eastern style emoticons. The focus of our tool is mostly, but not limited to, Western style emoticons, although it is by no means complete in any category. Instead we have chosen to emphasize on those that we believe are the most frequently used emoticons.

With no further delay, let’s see the interface of our tool.
From this screenshot you should be able to see that our tool supports opening a file (plain text only, *.txt files), and allows for the file’s encoding to be selected (UTF-8 being considered as the default). Take note that UTF-8 is not the default option for saving *.txt files in Notepad, but it is necessary to choose a Unicode Transformation Format in order for Greek letters to appear correctly. You can find this option in the Save As dialogue of Notepad, right next to the Save Button.

Back to our tool then, in case you don’t happen to have a text file and you want to manually input some text, you can simply paste your text or even type some text of your own in the Text Area.

Each time you press the Mark Text button, the tool marks all emoticons, shouting and usage of foul language, and sums everything up into five categories. This function is automatically performed each time you choose to open a new file, and each time you press the enter key on your keyboard if you are writing in the text area.
Our selection of categories was indeed somewhat arbitrary, so it’s mostly up to you to choose whatever categories you consider to be the most helpful sentiment designators in your implementations.

Well then, in case you are still wondering about it, here is a snapshot of how our tool looks like in action!

![Figure 13 - The Emoticons and Shouting Marking Tool in action](image)

It does have limitations though, especially in the shouting category in regards to abbreviations. Due to the mechanism that shouting is detected, i.e. a word or phrase having all letters capital, it is hard to distinguish between a valid shouting and an abbreviation like for example EU or DNA.

Leaving that aside, our tool’s final functionality is the “Copy Text and Results in Excel Format” button. This button basically copies all the text from the text area, as well as the number of occurrences for each category in the order they appear in the application, separated by tabs. Essentially this provides an easy way to paste the data into an Excel file, in the format of a row containing six separate columns (the text plus the five categories).

Now that we have a tool to help us better identify the sentiments in a comment using some cues like emoticons, shouting and a symbol based detection of foul language, let’s see if we can train a classifier that will help us to automatically classify the emotional polarity of uncategorized comments containing any of those cues.
6.2 Gathering Data
First off, what we are going to need is data, the more the better. As we are emphasizing on Greek comments, the main sources of our data were mostly various local online newspapers, as well as online magazines and forums.

Be warned though that the task of merely collecting your data can be quite time consuming, depending on the ease of automating the task i.e. if you can gather the data using an algorithm or if you have to hand-pick them.

In our case this task was quite time consuming as our data had to be in the Greek language, and also containing any of the cues we mentioned previously in order to test the added value by our emoticons tool. Therefore the only option was to hand-pick our data.

This is how the data looks like in an Excel spreadsheet. We were able to gather around 230 different comments, 228 to be exact, so there are 229 rows in the spreadsheet (the row containing the column names and the 228 rows of data).

At this stage we do some basic preprocessing of the data, by performing a Spelling Check. That way we correct any misspelled words and fill-in any missing intonation of words, making the most out of our data.

We also evaluate each comment one by one, in order to categorize them as positive or negative in the “polarity” column. You are not limited to these values though, as it is possible to have any and as many values as you like.
6.3 Importing Data
Importing from an Excel file requires a Read Excel, and a Store operator to store the data.

Figure 15 – Importing Data from an Excel File

For the Read Excel operator, click Import Configuration Wizard from the parameters on the right.

The Data Import Wizard will help you to:

Step 1. Select your Excel file
Step 2. Select the correct sheet from your Excel file
Step 3. Check the annotations. Here our first row contains the attribute names.

Figure 16 – The Data Import Wizard’s Annotation Step

Step 4. Select the data type and role for each attribute.

Figure 17 – The Data Import Wizard’s Attribute Data Types and Roles Step
On top of each attribute, a checkbox includes or excludes that attribute from import.

Below each attribute we have to select the attribute’s data type. The wizard tries to make a prediction, but pay attention as it doesn’t always estimate correctly. In our case the ID attribute is integer and is guessed correctly, but the COMMENT attribute is guessed as polynominal (i.e. having many different values), so we fix this by choosing the text data type. Attributes HAPPY up to SHOUTING are correctly guessed as integers, and the POLARITY attribute is correctly guessed as binominal (i.e. having only two possible values).

The roles of each attribute are directly below. The default state is (regular) attribute.

First we have to fix the role of the ID attribute so that it has the role of id, and we also have to change the POLARITY attribute so that it has the role of label.

A label in RapidMiner is basically a target attribute (dependent variable). All Classification tasks require a label in order to function. A Classifier is first trained from data which were already labeled by humans, and then builds a model (set of rules) that can be used to predict the labels for new unseen data, as long as they share the same attributes.

In the orange box in Figure 8, we store the same data but without the sentiment attributes. This will be our control dataset in order to compare the performances of the two datasets.

The Multiply operator makes multiple copies of the data you provide as input, so in our case it just clones the dataset from the Read Excel operator and feeds it to the Select Attributes operator.

The Select Attributes operator selects which attributes from an exampleset should be kept. Since for the control dataset we don’t want the sentiment attributes, from the parameters we select that we want an attribute subset, and then select only the COMMENT, ID, and POLARITY attributes.

Regarding the Store operators, the only thing you need to provide as a parameter is the locations where the repositories should be saved.
6.4 Classification Process

At last we get to the classification part. Here, a classifier is trained to create a model that classifies comments into positive or negative.

We have again the retrieve operator that retrieves our data, and then in the orange box we have some more preprocessing. The Process Documents from Data operator’s parameters are configured the same way as before, but with a few additions inside the operator.

![Figure 19 - The Classification Process](image)

Again the first row contains the same operators as before, with the same configuration for each operator, but now we have two additional operators marked inside the green box.

The **Stem (Dictionary)** operator replaces terms by pattern matching rules. In the parameters of this operator we load the file containing the word stems from our Greek Stemmer that we have simply copy-pasted to a simple *.txt file. This will essentially reduce all words containing these stems to their respective stems.

The **Generate n-Grams (Terms)** operator creates additional attributes by merging up to n tokens as one, which is useful for words that usually appear together. In the parameters of this operator we set max length to 3, i.e. up to word triples.

Back to the main process, next up is the **Weight by Rule** operator, which belongs to a group of operators aimed towards Dimensionality Reduction, i.e. reducing the attributes of an example set. Some alternatives offering similar functionality are the **Weight by Information Gain**, **Weight by Information Gain Ratio**, and **Weight by Gini Index** operators.
The **Weight by Rule** operator was our preference, which calculates weights for the attributes of a given ExampleSet with respect to the label attribute, by constructing a single rule for each attribute and calculating the errors. The higher the weight of an attribute, the more relevant it is considered.

The reason we chose this operator, is because it weighs the attributes by more distinct levels rather than a more continuous range as the other operators do, and gives zero weights to more attributes in comparison to the rest of the operators. So using this operator we get a minimalistic set of attributes, which is helpful when having distinct classification classes.

The **Select by Weights** operator selects only those attributes of an ExampleSet whose weights satisfy the specified criterion. In the parameters of this operator we chose to keep attributes with greater than zero weights.

The **X-Validation** operator, which is short for cross-validation, performs a cross-validation in order to estimate the statistical performance of a learning operator, and is mainly used to estimate how accurately a model is expected to perform in practice.

In the parameters of this operator, the **number of validations** specifies the number of iterations that will take place which is also the number of subsets the ExampleSet will be divided into. Each subset has an equal number of examples, and each iteration involves training a model and testing that model. The default value is 10.

For **sampling type**, we are given 3 options to choose from: **linear, shuffled and stratified sampling**.

- **Shuffled sampling** builds random subsets of random examples from the ExampleSet, and is our preference.
- **Linear sampling** simply divides the ExampleSet into partitions without changing the order of the examples, i.e. subsets with consecutive examples are created.
- **Stratified sampling** builds random subsets and ensures that the class distribution in the subsets is the same as in the whole ExampleSet. For example in the case of a binominal classification, stratified sampling builds random subsets such that each subset contains roughly the same proportions as in the ExampleSet of the two values of class labels.

Inside the **X-Validation** operator, we have the following subprocesses:
On the left side we have the **Training subprocess**, and on the right side we have the **Testing subprocess**. The training subprocess is used for training a model. The trained model is then applied in the testing subprocess. During the testing phase, the performance of the model is also measured. The way these two subprocesses function is the following:

The input ExampleSet of the X-Validation operator is partitioned into $k$ subsets of equal size, where $k$ is the number of validations in the operator’s parameters, which as explained above is also the number of subsets. Of the $k$ subsets, $k-1$ subsets are used as a training dataset (i.e. input of the training subprocess), and a single subset is retained as the testing dataset (i.e. input of the testing subprocess). The cross-validation process is then repeated $k$ times, with each of the $k$ subsets used exactly once as the testing data. The $k$ results from the $k$ iterations are then averaged (or otherwise combined) to produce a single estimation.

The learning processes usually optimize the model they generate to make it fit the training data as well as possible. If we test this model on some independent set of data, mostly this model will not perform that well as it performed on the training data that generated it. This is called over-fitting. The X-Validation operator predicts the fit of a model to a hypothetical testing data, which can be especially useful when you don’t have separate testing data.

For training we have chosen a **Decision Tree** classifier. You can find a comparison of classifiers in the following table from the paper “Supervised Machine Learning: A Review of Classification Techniques” by S. B. Kotsiantis.
Table 4: Comparing learning algorithms, from the paper “Supervised Machine Learning: A Review of Classification Techniques” by S. B. Kotsiantis.

Regarding our dataset, one of our interest areas is the interpretability of the built model. As you can see from the table above, the classifiers with the best interpretability (explanation ability) are the Decision Trees, Naïve Bayes and the Rule Learners such as Rule Induction.

For best accuracy you should aim at Neural Networks and SVM, but these also happen to have the worst interpretability of their built models.

**Speed of learning** is the speed at which the algorithm can build a model with respect to the number of attributes and the number of examples.

**Speed of classification** is the speed at which the classifier can apply the already built model to new unseen data.

**Tolerance of missing values** concerns datasets that contain examples with missing values.

**Tolerance of irrelevant attributes** is important to our dataset, since not each and every word attribute is required for building a good text classification model.

**Tolerance to noise** refers to examples in an exampleset that are irrelevant or do not follow the same patterns as the majority of data, such as the outliers mentioned earlier.
**Overfitting** is essentially a problem of a model being built too tightly according to the training dataset, in such a way that makes it less useful with real-life testing data. Usually you want to avoid overfitting, unless your data are very representative of real-life data.

According to the above, our preference is the **Decision Trees** classifier, since it is the most balanced classifier in regard to our needs.

A **decision tree** is a tree-like graph or model. It is more like an inverted tree because it has its root at the top and it grows downwards. This representation of the data has the advantage compared with other approaches of being meaningful and easy to interpret. The goal is to create a classification model that predicts the value of a target attribute (called label in RapidMiner) based on several input attributes of the ExampleSet. Each interior node of the built tree corresponds to one of the input attributes. Outgoing edges of numerical attributes are labeled with disjoint ranges. Each leaf node represents the value of the label attribute given the input attributes values represented by the path from the root to the leaf.

Decision Trees are generated by recursive partitioning, which means repeatedly splitting on the values of attributes. In every recursion the algorithm follows the following steps:

- An attribute $A$ is selected to split on. Making a good choice of attributes to split on each stage is crucial to the generation of a useful tree. The attribute is selected based upon a selection criterion which can be selected by the criterion parameter. Our criterion of choice is **accuracy**.
  - Examples in the ExampleSet are sorted into subsets that are formed for disjoint ranges of attribute $A$’s values.
  - A tree is returned with one edge or branch for each subset. Each branch has a descendant subtree or a label value produced by applying the same algorithm recursively.

In general, the recursion stops when all the examples or instances have the same label value, i.e. the subset is pure. Or recursion may stop if most of the examples are of the same label value. This is a generalization of the first approach, with some error threshold. However there are other halting conditions such as:

- There are less than a certain number of instances or examples in the current subtree. This can be adjusted by using the minimal size for split parameter.
- No attribute reaches a certain threshold. This can be adjusted by using the minimum gain parameter.
- The maximal depth is reached. This can be adjusted by using the maximal depth parameter.

To convert an over-specific or overfitted tree to a more general form in order to enhance its predictive power on unseen datasets, there are some pruning parameters.
**Pruning** is a technique in which leaf nodes that don’t add to the discriminative power of the decision tree are removed. Pre-pruning is a type of pruning performed parallel to the tree creation process, whilst Post-pruning is done after the tree creation process is complete.

Back to our X-Validation operator, on the right sub-process where the testing occurs we use an Apply Model and a Performance operator.

![Figure 22 – X-Validation, Testing subprocess operators](image)

The **Apply Model** operator applies the already learnt (trained) model on an ExampleSet. For each of the k repetitions of X-validation, each of the k subsets of the ExampleSet is used exactly once as testing data.

*Note that it is compulsory for both the training and testing ExampleSets to have exactly the same number, order, type and role of attributes.* If these metadata properties of ExampleSets are not consistent, it may lead to serious errors.

The **Performance** operator is used for performance evaluation, and delivers a list of performance criteria values. These performance criteria are automatically determined in order to fit the learning task type.

![Figure 23 – Storing the wordlist, attribute weights and model](image)

Using the **Store** operators, we store into three separate data repositories the wordlist, weights and built model, in order to be able to apply the built model to new unseen data and automatically classify them (see **Apply Model note**).

Lastly, we repeat this entire process using pretty much the same dataset but this time not containing the added sentiment attributes, as a control dataset to compare the increase in accuracy by adding these sentiment attributes.
6.5 Classification Results

From the Decision Tree model built from our dataset with sentiment attributes, we can see that the most important decision attributes were HAPPY, which is one of our added attributes from the emoticons detection, εργασία (work), καλό (good) and ωραία (nice). If these attributes are larger than some threshold values (different thresholds for each attribute) then the example is almost always positive, whereas if none of these attributes is larger than the threshold values then the example is almost always negative.

From the Performance operator, we get various measures as seen below.

**Accuracy** is calculated by the percentage of correct predictions over the total number of examples. Correct prediction means examples where the value of the prediction attribute is equal to the value of the label attribute.

**Precision** of a class is calculated by taking the correct predictions of a label’s value over the total predictions for the same label value (correct predictions + wrong predictions).

**Recall** of a class is calculated by taking the correct predictions of a label’s value over the total of the real examples with the same label value (correct predictions + missed examples).
6.6 Comparison with Control Dataset

The decision tree of the control dataset that doesn’t contain the sentiment attributes is both much bigger, and has some decisions that are based on attributes not related to sentiments such as σχόλιο (comment).

The performance of this decision tree has a lower accuracy, and especially the recall of the POSITIVE labeled comments (which are much less compared to the NEGATIVE labeled comments) is only 25.71%.

---

**Figure 26** – Decision Tree from the control dataset without the sentiment attributes

**Figure 27** – The performance of the Decision Tree from the control dataset without the sentiment attributes
So by adding the sentiment attributes, we get an overall better model by achieving an about 20% increase in accuracy, better precision and much better recall, especially for the minority class (positive examples), as well as a better selection of attributes by the Decision Tree.
Chapter 7. Comment Summarization

7.1 Gathering Data from OpenGov.gr

OpenGov is a Greek Open Government Initiative which has been designed to serve the principles of transparency, deliberation, collaboration and accountability. Here almost every piece of draft legislation, or even policy initiative by the government, is posted in a blog like platform prior to their submission to parliament, so that citizens and organizations can post their comments, suggestions and criticisms article-by-article.

On the right as in the picture above, you are given the option to view content per ministry.

After selecting a ministry, you are taken to a page containing all public consultations regarding that ministry.
By selecting a public consultation, you are provided with the law draft, as well as the comments of the public per article of the bill. On the right there are some tools, one of which is the extraction of all comments as an Excel file.

Using some of these Excel files as our data source, we are going to try to summarize the public’s opinions by Clustering and then we will visualize the results by creating word clouds.
7.2 Importing Data

For the use of the Read Excel and Store operators, refer to the Classification process above.

The **Append** operator builds a merged ExampleSet from two or more compatible ExampleSets by adding all examples into a combined set. Note that all input ExampleSets must have the same attribute number, names and roles.

The **Remove Duplicates** operator removes duplicate examples from an ExampleSet by comparing all examples with each other on the basis of the specified attributes. Two examples are considered duplicate if the selected attributes have the same values in them.
7.3 Clustering Process

Using the Retrieve operator we read the stored data of the excel files from RapidMiner’s repository, from which we have kept only the «Κωδικός Σχολίου» (ID), and «Σχόλιο» (comment) attributes.

Inside the Process Documents from Data operator, we have exactly the same configuration as in Figure 20, but note that for each new data input we have to find the new word stems for the Stem (Dictionary) operator by using our Greek Stemmer. To find the word similarities required for our Greek Stemmer, the process in Figure 4 has to be run first, using the same input as in this process.

The Detect Outlier (LOF) operator identifies outliers in the given ExampleSet based on local outlier factors (LOF). The LOF is based on a concept of a local density, where locality is given by the k nearest neighbors, whose distance is used to estimate the density. By comparing the local density of an object to the local densities of its neighbors, one can identify regions of similar density, and points that have a substantially lower density than their neighbors. These are considered to be outliers.

An outlier is an example that is numerically distant from the rest of the examples of the ExampleSet. Outliers can therefore often (but not always) indicate faulty data.

In the parameters of this operator we set the distance function to inverted cosine distance. The only measure that seems to work well with word attributes is cosine. The reason for choosing an inverted cosine distance is that if we had chosen the regular cosine distance, the furthest an example is from the rest of the examples, signifying an outlier, then the greater will be the distance. This makes it hard to choose a distance threshold above which the examples should be considered as outliers, because the distances can go up to infinity. By choosing the inverted cosine distance, the measures are inverted, so the outliers now take the smallest values with zero being the lowest. So now it is easier to set a threshold signifying the outliers, which will be near zero.
The **Filter Examples** operator selects which examples of an ExampleSet should be kept and which examples should be removed. Examples satisfying the given condition are kept, remaining examples are removed.

Here we set the condition class to be an attribute value filter, and as a parameter string we set “outlier > 0”, without the quotes. The outlier attribute is a new attribute that the Detect Outlier operator generated. Since for the Detect Outlier operator we selected an inverted distance, the outlier>0 condition keeps all examples that are not outliers.

Next up, using a **Select Attributes** operator we want to keep all attributes besides the outlier attribute. To do that we select the single attribute filter type and select the outlier attribute, and then we check both the invert selection and include special attributes checkboxes.

The **X-Means** operator is our operator of choice for performing the clustering. There are other operators for clustering, but we chose X-Means because it improves the popular k-Means algorithm by estimating the number of clusters so the user doesn’t have to know it in advance as is the case with k-Means.

**X-Means** is a clustering algorithm which determines the correct number of centroids based on a heuristic. It begins with a minimum set of centroids and then iteratively exploits if using more centroids makes sense according to the data. If a cluster is split into two sub-clusters is determined by the Bayesian Information Criteria (BIC), balancing the trade-off between precision and model complexity. Original publication: “X-means: Extending K-means with Efficient Estimation of the Number of Clusters” by Dan Pelleg and Andrew Moore, Proceedings of the Seventeenth International Conference on Machine Learning, 2000.

For this operator we change the measure types to **Numerical Measures**, and as a Numerical Measure we choose the **Cosine Similarity**.

Again we stress that only the **Cosine** measures are capable of handling words.

The **Set Role** operator is used to change the role of one or more attributes. Using this operator we want to change the role of the cluster attribute, generated by the X-Means operator, into having a label role.

Note that although the X-Means operator seems to have an “add as label” checkbox, it doesn’t seem to be implemented correctly and the generated cluster attribute remains of cluster role instead of label role after checking that checkbox.

This time we use the **Weight by Gini Index** operator for attribute reduction, which calculates the relevance of the attributes of the given ExampleSet based on the Gini impurity index.

**Weight by Gini Index** is a rather defensive approach because the amount of irrelevant attributes it finds is quite minimal. By comparison, **Weight by Information Gain Ratio** is a...
good alternative finding an ideal proportion of irrelevant attributes, **Weight by Rule** takes a lot of time, and **Weight by Information Gain** finds a trivial amount of irrelevant attributes.

**Note** that none of these operators was evaluated for the quality of its selections, so you could omit attribute reduction if you want to be on the safe side.

Using the **Select by Weight** operator, we select to keep the attributes that have weights greater than zero.

With the second **Select Attributes** operator we now want to discard the “text” and «Κωδικός Σχολίου» (ID) attributes, so that we can aggregate the results.

The **Aggregate** operator imitates the aggregation functions of SQL. It focuses on obtaining summary information, such as averages, counts etc., by grouping examples into smaller sets and applying aggregation functions on those sets.

Using this operator we want to find the average word frequencies for each cluster.

In the parameters we first select the “*use default aggregation*” checkbox which will perform the aggregations on all selected attributes. The **default aggregation function** that we want to be performed is the average for each attribute, and we want to *group by cluster*.

Instead of using this operator, we could have used the **Extract Cluster Prototypes** operator directly after the X-Means operator, but this would make it more difficult to understand the logic behind using these results. The Extract Cluster Prototypes operator would extract the **centroids** of the X-Means operator, which are the average values of each attribute for each cluster. The centroids in general though, do not carry information regarding relationships between attributes. They are in a sense the core of each cluster, as they represent a point that is right at the center of each cluster, thus taking their name as centroids.

The centroids of each cluster represent the average frequencies of each word in each cluster, but because all of our attributes (besides the cluster attribute) are words, and because word frequencies is a measure of comparison between words by denoting a higher significance if a word occurs frequently, therefore attributes that have a greater average frequency for a given cluster are expected to be more representative for that cluster!

The idea to find the average frequency of each word in every cluster and consider the most frequent words as the most important for a cluster, happens to match with the cluster centroids, but comparisons between attributes of a centroid based on their values is meaningless for centroids in general, so this is why we preferred the Aggregate approach.

Using the **Rename by Replacing** operator, we want to restore all attributes from the now “average(attribute_name)” back to their original names. To do that we replace average\(|\) with an empty string. Here \(\backslash\) is an escape character for dealing with parentheses as symbols, and \(|\) represents the logical OR.
Then we use another **Set Role** operator to change the cluster attribute’s role into id. The reason behind this is to facilitate the transpose that follows.

The **Transpose** operator transposes the input ExampleSet, i.e. the current rows become columns and current columns become rows.

We do this to have the words as examples and the clusters as attributes.

Because the Transpose operator swapped the rows and columns, now the id column is the column containing the words. But because the transpose operator didn’t know how to name it, it simply named it as “id”. Using the **Rename** operator, we rename the “id” attribute to “word”.

The results should be similar to the following:

![Figure 35 – The Clustering Results containing the average frequencies of the words in each cluster](image)

Finally, using the **Loop Attributes** operator we now want to iterate through all attributes (the clusters), each time executing the following subprocess.

Inside the Loop Attributes operator’s subprocess we use the **Write Special Format** operator to write files in the format $i:$v[%{loop_attribute}]. $i$ means the values of the id attribute, then a ‘:’ symbol, and then $v[%{loop_attribute}]$ means the values of the attribute of the current loop iteration.

We also set the “example set file” parameter to “filename_%{loop_attribute}.txt” using a filename of our choice, where the %{loop_attribute} part will automatically be substituted by the current attribute (cluster name) in each iteration.

This will create a .txt file for each cluster, with each file containing the average word frequencies for the particular cluster in the format “word:word_frequency” for each word.

With the help of these files we can visualize the results through word clouds which we will create through the [Wordle.net](http://wordle.net) website.
7.4 Creating Word Clouds

To create the word clouds we will use a free tool provided by the Wordle.net website.

![Wordle website for creating word clouds](image)

Figure 36 – The Wordle website for creating word clouds

If you follow the link to the Create tab, you can see that you can simply paste a bunch of text and Wordle will create a word cloud based on that. However, Wordle has no capabilities (yet) for Greek stopwords filtering and stemming, and takes the entire text as a single entity.

![Wordle’s Create tab for text or URL’s](image)

Figure 37 – Wordle’s Create tab for text or URL’s
Because our results were already processed and are in the form of word:word_frequency, we will use the Advanced tab of Wordle to input our results.

Figure 38 – Wordle’s Advanced tab for creating word clouds through weighted words (and more)

In the first text area we can copy the contents of our created files and get the word clouds.

Figure 39 – A sample word cloud created by Wordle

By right-clicking on the blank area of the created word cloud, you can change the Layout into Horizontal for viewing the words more easily, and you can also change the Colors used.
7.5 Clustering Results

Since there is no straightforward way to assess the quality of the built clusters, by using the article column of the Excel data we have found the most discussed articles, created their word clouds, and then compared them with each cluster’s word clouds to see if they match.

The results for the data from the Ministry of Finances were the following:

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Comments</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster_0</td>
<td>83</td>
<td>All remaining topics</td>
</tr>
<tr>
<td>Cluster_1</td>
<td>57</td>
<td>ΠΔ 237/1986</td>
</tr>
<tr>
<td>Cluster_2</td>
<td>134</td>
<td>Pension Deposits</td>
</tr>
<tr>
<td>Cluster_3</td>
<td>102</td>
<td>Regulation of other Retirement Issues</td>
</tr>
<tr>
<td><strong>Total comments</strong></td>
<td><strong>376</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 – Clusters using data from Ministry of Finances and their references to the most discussed articles

Out of these clusters, Cluster_0 happens to be the one containing the leftover comments on various topics, while the remaining 3 clusters concentrate very specifically on the topics that gathered the most traction among the people, as can be seen by the figures below.

Cluster_0 contains keywords from comments that were referring to various topics.
In Figure 41 we see the word cloud from a widely discussed article regarding ΠΔ 237/1986, and comparing that with Figure 42 which has the keywords from Cluster_1 of our clustering process, we can see that the keywords from the two word clouds have a high resemblance!

Note that the clustering process has no input whatsoever regarding the articles. It automatically concludes from the training data that many comments make frequent use of these words and therefore should be grouped together in a cluster.
We arrive to the same conclusion comparing the word clouds in Figure 43 and Figure 44, where in the first figure we have the keywords from a widely discussed article regarding Pension Deposits, and in the second figure we have the keywords from Cluster_2 of our clustering process, where we can see again that the two of them have a high resemblance!
From our last cluster, Cluster_3, we have good clustering results again, since the keywords from this cluster in Figure 46, have a high resemblance to the keywords of a widely discussed article regarding the Regulation of other Retirement Issues in Figure 45.
By repeating this quality test for data first from the Ministry of Health and then from the Ministry of Justice, we arrive to similar conclusions.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Comments</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster_0</td>
<td>252</td>
<td>Organ Transplants</td>
</tr>
<tr>
<td>Cluster_1</td>
<td>132</td>
<td>All remaining topics</td>
</tr>
<tr>
<td>Cluster_2</td>
<td>102</td>
<td>Abolishment of «ΚΕΚΥΚΑΜΕΑ ΜΕΣΣΗΝΙΑΣ»</td>
</tr>
<tr>
<td>Cluster_3</td>
<td>128</td>
<td>Physical and Medical Restoration</td>
</tr>
<tr>
<td>Total comments</td>
<td>614</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 - Clusters using data from Ministry of Health and their references to the most discussed articles

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Comments</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster_0</td>
<td>161</td>
<td>Cannabis Cultivation</td>
</tr>
<tr>
<td>Cluster_1</td>
<td>229</td>
<td>Lawyers and Notaries</td>
</tr>
<tr>
<td>Cluster_2</td>
<td>143</td>
<td>Correctional Facilities</td>
</tr>
<tr>
<td>Cluster_3</td>
<td>81</td>
<td>Correctional Facilities</td>
</tr>
<tr>
<td>Total comments</td>
<td>614</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 - Clusters using data from Ministry of Justice and their references to the most discussed articles

In Table 4 regarding data from the Ministry of Justice, we get no clear distinction between Cluster_2 and Cluster_3. This can be attributed to the fact that the discussions for the topic of Correctional Facilities were very disproportionate (almost double the comments) compared to the next most discussed topics. Also because the remaining comments referring to other topics were too few, the clustering process was unable to assign them to a separate cluster and instead has distributed them among the already formed clusters.

In conclusion we can say that the Clustering process works pretty well in finding clusters referring to the most discussed topics, as long as the clusters are of about the same size and not very disproportionate to each other. This applies to both bigger clusters which are split (e.g. very discussed topics), and smaller clusters which are merged (e.g. remaining topics).
Chapter 8. Automatic Comment Summarization Tool

Since the above clustering process is promising but difficult to use since the user needs to be acquainted with RapidMiner first, and it also requires the user to install RapidMiner in the first place, we are now going to build a tool providing a Graphical User Interface that incorporates the clustering process into a user-friendly environment.

On the top area a user can Open an Excel file, preferably downloaded from the OpenGov.gr website which has the format seen in Table 1, or at least it has to contain the attributes «Κωδικός Σχολίου» (ID) and «Σχόλιο» (Comment) in that order.

Then the user can press the Summarize Comments button and in the Comments text area will appear the comments id’s, assigned clusters and the comments themselves where the stopwords have been removed and the words have been stemmed.

In the Comments per Cluster section on the left you get the statistics regarding the clusters that were created, and in the Average Word Frequencies per Selected Cluster section in the middle, a user can select from the combo box the cluster he wants to visualize in order to get the average word frequencies for that cluster. All words are sorted for each selection from the least frequent words on top to the most frequent words at the bottom.

In addition to the above, the user can also paste a single comment in the Single Comment Summarization text field and by pressing the Summarize Single Comment button he can get the word frequencies for the single comment at the text area on the right. Again the words are sorted from the least frequent words on top to the most frequent words at the bottom.
Lastly, by pressing the Assign Comment to Cluster button, the user can get a cluster assigned to the single comment, based on the model built at the summarize comments section.

Note that the Summarize Comments button takes a little while to finish because the clustering process needs a couple of minutes depending on the number of rows in the data.

In the example in Figure 48, we have used the comment with id 21549 as a single comment, which we can see it was correctly assigned to cluster_1.

All results containing word frequencies can be visualized using Wordle.net by going to the advanced tab to paste our results, as we have already seen in Figure 38.

For example to visualize cluster_1, we select cluster_1 at the combo box in the middle area and copy-paste the word frequencies from the middle text area to Wordle.net advanced tab.

Figure 49 shows the word cloud created by the average word frequencies in cluster_1 from the above example.
8.1 Underlying processes

In order to use libraries of RapidMiner inside a java program, you have to add into the libraries of your project the rapidminer.jar file located inside the lib directory of your RapidMiner installation directory. For obtaining the documentation of rapidminer.jar, you can right-click on it after you add it to your project’s libraries, select the Edit option and link the Javadoc section to http://rapid-i.com/api/rapidminer-5.1/overview-summary.html

To make the Automatic Comment Summarization tool, the following processes have been implemented.

During the Automatic Summarization Process Part A, we read an excel file which is given by the user during the run of the program, we select to keep only the «Κωδικός Σχόλιου» (comment id) and «Σχόλιο» (comment) attributes, we set the role of «Κωδικός Σχολίου» to id, and then using the Nominal to Text operator we convert the «Σχόλιο» attribute to text. For the Process Documents from Data operator we deselect the word vector creation checkbox and check the keep text checkbox, while applying no pruning up to this stage.

Inside the Process Documents from Data operator we tokenize the comments into words, filter-out the words with less than 2 and more than 30 letters, and filter-out the stopwords using our Greek Stopwords.txt file. For the Filter Stopwords operator it is important to change the encoding to “windows-1253” for java to read correctly the Greek characters.

Then we take the output of this process and stem the words. This time we use the Stemmer of Παπαστεργίου Χρήστος from his thesis on “Sentiment mining in e-government social means”, Chapter 5.2.4: The Stemmer class. The only change we make to this stemmer class is to change all functions into static so that we can call them without the need to create objects of the stemmer class.

After we run the stemmer and update the changes inside the exampleset, we run a second process using the stemmed exampleset as input.
During the **Automatic Summarization Process Part B**, we change the role of the text attribute into regular attribute (previously «Σχόλιο» which got renamed into “text” and was given the special role of text after the Process Documents from Data operator), and we use once again the **Process Documents from Data** operator. This time we check both the create vector and keep text checkboxes, we use TF-IDF for vector creation, and we use percentual pruning to prune the words that appear in below than 1% of the total comments.

Inside the **Process Documents from Data** operator we use a transform cases operator to transform all letters to lowercase, we tokenize the comments into words and filter-out words that have less than 2 and more than 30 letters, then replace all intonated vowels with their toneless versions as we did in Figure 6, and then generate up to word triples.

After the Process Documents from Data operator, we **Remove Duplicate** examples and then we use the **X-Means** clustering operator. Again it is of utmost importance to use Numerical Measures and the **Cosine Similarity** as this is the only measure that works well with words.

The **Performance (Cluster Distance Performance)** operator will provide some performance values for our clustering such as the average within centroid distance for each cluster.

Using the **Aggregate** operator we find the averages for all words inside each cluster by checking the use default aggregation checkbox and grouping examples by cluster, then rename the now “average({wordName})” attributes back to each attribute’s wordName, set the role of the “cluster” attribute to id to facilitate the transpose that follows, then transpose the exampleset so that now the words appear as the examples and the clusters as the attributes, and rename the “id” attribute to “word”.

The functionality of most of these operators is discussed at the Clustering Process section.

Any Weight By operators were omitted due to our lack of quality tests for their results.

**Note** that the **Detect Outlier** operator was causing the whole process to hang when used through java, so we omitted it deliberately.
The Single Comment Stopwords Filtering process works in a similar way to Figure 50, but instead of reading an excel file containing many comments, we now have a Create Document operator which creates a single document from the given input string. For processing documents instead of example sets, the Process Documents operator is used.

The single comment is then passed through the stemmer of Παπαστεργίου Χρήστος as we have done previously, and the resulting stemmed words are set in the example set.

Using the example set containing the stemmed words as input, we then run the Single Comment Word Frequencies process which functions in a similar way to Figure 51, but this time the Remove Duplicates, X-Means and Aggregate operators (as well as their dependent operators) are removed since they have no functionality for single example example sets.

Also note that for the Process Documents from Data operator we now want the vector creation to be performed using only the Term Frequency instead of TF-IDF, as the latter is again not applicable to single comment example sets.

The Single Comment Assign Cluster process has a prerequisite that the Automatic Summarization Process Part B is first ran, which among other things performs the clustering. If not, then the button for this process is disabled since you can’t assign a comment to a cluster if you don’t already have a clustering model.
The **Single Comment Assign Cluster** process takes as input the single comment after the stopwords filtering and stemming, the wordlist that was used to create the clustering model, and the clustering model itself.

The **Set Role** operator changes the role of the “text” attribute, previously the «Σχόλιο» attribute which got renamed into “text” and was given the special role of text after the Process Documents operator, into a regular attribute.

The **Process Documents from Data** operator when given a wordlist as input, loads exactly the same attributes and in the same order as in the wordlist, so that only these attributes are allowed to be measured from processing the given exampleset. This means that all new words that appear in the exampleset will be discarded. This is to ensure when applying the clustering model which was built based on a specific set of attributes, that the set of attributes remains the same; otherwise the clustering model will produce faulty results.

Again since we have only a single comment, we want the **Term Frequency** for the vector creation of the **Process Documents from Data** operator, instead of TF-IDF.

The **Apply Model** operator is then used to apply the previously built clustering model to the new exampleset, which in our case is the single comment.

The **Performance (Cluster Distance Performance)** operator is used to assess the distance of this single comment from the assigned cluster’s centroid, signifying if the comment was unquestionably or marginally assigned to that cluster.

![Diagram](image.png)

**Figure 55 – The Sort process**

Finally, the **Sort** process is used to sort the given input in ascending order, so that when used for the word frequencies it will sort the most frequent words at the bottom.
Chapter 9. Summary

In this thesis we have created:

- An **Emoticons and Shouting Marking Tool** that detects both Western (e.g. 😊) and Eastern (e.g. ^_^) style emoticons, grawlixes (e.g. %&^ @$$hole) and shouting (words written in all capital letters), and distinguishes them into Happy, Skeptical, Unhappy, Angry and Shouting emotions. This tool was mostly coded using Regular Expressions, has a weakness in detecting as shouting the acronyms written in all capitals (e.g. DNA, EU, etc.), and could be further improved by adding more sentiment categories and supported emoticons.

- A **Greek Stemmer** that uses word similarities and generates stemming rules specific to the dataset provided. This approach works through word comparisons and cannot stem words individually, but has the benefit of not needing complex language rules since it only matches words having high similarity and keeps only their shared part, and then removes all vowels from the end of the stem while preserving the «αυ» and «ευ» letter combinations where the second letter acts like a consonant. The word similarities are based on the letter compositions of the words, and are found through a RapidMiner process that needs to be run prior to the Greek Stemmer. To exclude similar words which are due to word anagrams, the ID’s of the alphabetically sorted words are taken into account so that the words that are to be considered as similar should be adjacent. This approach works well with words longer than 7 letters (>6 letter stems) but is ineffective for smaller words.

- A **Sentiment Classification process** making use of the added attributes from the Emoticons and Shouting Marking Tool. Although preliminary results look promising, the Decision Tree classifier needs more training data to build a reliable model.

- A **Comment Summarization and Clustering process** which shows very good clustering results but presumes that all clusters should be of about equal size. The clustering is based on X-Means, a k-Means variation that estimates the correct number of clusters, using the Cosine Similarity Numerical Measure which is the only one that functions well with word attributes. The summarization is based on the words having the highest appearance in each cluster.

- An **Automatic Comment Summarization Tool** which is essentially a Graphical User Interface mainly for the Comment Summarization and Clustering process, where an end-user can summarize comments provided as an Excel file having at least the columns «Κωδικός Σχολίου» (id) and «Σχόλιο» (comment) in that order. Additionally a user can get a summary for a single comment, and if an excel file was provided as training to prepare a clustering model, then the single comment can also be assigned to the cluster where it seems to fit best. The word frequencies for each cluster, or for the single comment, can easily be visualized through the Wordle.net website by copying and pasting these results to the website’s advanced tab to generate word clouds where the most frequent words appear bigger than the rest. (The stemmer used for this tool is from the thesis of Παπαστεργίου Χρήστος on “Sentiment mining in e-government social means”).
Bibliography


Sentiment Analysis by Emoticons and Unsupervised Comment Summarization in Greek e-Government data


Stemming and Lemmatization [Online] // Stanford Natural Language Processing. -

Stopwords [Online] // code.google.com. -
http://code.google.com/p/gss/source/browse/solr/conf/stopwords.txt?r=d59fbc2f0cd52506df1db4e5037e4ca27962ce.

Stopwords 2 [Online] // Github.com. -

Supervised learning [Online] // Wikipedia.org. -


The Java File Chooser [Online] // Home & Learn. -
http://www.homeandlearn.co.uk/java/opening_files.html.


Unsupervised learning [Online] // Wikipedia.org. -