Project Title:	**Sensing and predictive treatment of frailty and associated co-morbidities using advanced personalized models and advanced interventions**
Contract No:	690140
Instrument:	Collaborative Project
Call identifier:	H2020-PHC-2014-2015
Topic:	PHC-21-2015: Advancing active and healthy ageing with ICT: Early risk detection and intervention
Start of project:	1 January 2016
Duration:	36 months

Deliverable No: D4.12
LingTester Test Results – Passive (off-line) mode (vers a)

Due date of deliverable:	M18 (30th June 2017)
Actual submission date:	30th June 2017
Version:	1.1
Date:	30th June, 2017

Lead Author(s):	C. Tsimpouris, N. Fazakis, K. Sgarbas (UoP)
Lead partners:	UoP

Horizon 2020
European Union funding
for Research & Innovation

CHANGE HISTORY

Ver.	Date	Status	Author (Beneficiary)	Description
0.1	01/05/2017	draft	C. Tsimpouris (UoP), N. Fazakis (UoP), K. Sgarbas (UoP)	Initial draft
0.2	07/06/2017	draft	C. Tsimpouris (UoP), N. Fazakis (UoP), K. Sgarbas (UoP)	First draft deliverable report, sent for internal review
0.3	22/06/2017	draft	C. Tsimpouris (UoP), N. Fazakis (UoP), K. Sgarbas (UoP)	Second draft deliverable report, Text updates on introduction & chapters 2,4,5,6
0.4	23/06/2017	draft	L. Bianconi (SIGLA), M. Toma (SIGLA) K. Petridis (HYPERTECH)	Revision of the document
1.0	29/06/2017	final	C. Tsimpouris (UoP), N. Fazakis (UoP), K. Sgarbas (UoP)	Final version circulated to partners
1.1	30/06/2017	final	C. Tsimpouris (UoP), N. Fazakis (UoP), K. Sgarbas (UoP)	Minor corrections, accompanying files included

EXECUTIVE SUMMARY

The LingTester Test Results (passive - offline mode) deliverable is the second deliverable of the Task 4.5 Processing social media which is part of the Work Package 4. In this deliverable the focus shifts from technical aspects of the LingTester tool, which prototype was presented in D4.10, to a more scientific approach regarding the test evaluation and results of the tool. This is achieved by concentrating on the essential tasks of the classification process. All the involving tasks, even if some of them seem to be trivial, in reality are equally important. The tasks are related to methodologies of collected data analysis, feature selection, classification and evaluation.

In a more general view, LingTester is the FrailSafe language analysis tool that aims to process the user's typed text and detect abnormal behaviour. At this point, the deliverable is in a preliminary version, but still it is able to perform classification according to levels of frailty.

The main objective of this Work Package is to handle the collection, management and analysis of frailty older people data streamed through their social, behavioural and cognitive activities. Both offline and online methods will be developed. Moreover, the above methods will be applied in order to manage and analyze new data and also generate the FrailSafe patient models.

Reader is strongly advised to read deliverable 4.10 in order to fully understand this report, as it is a follow up on how the prediction model has been updated.

DOCUMENT INFORMATION

Contract Number:	H2020-PHC–690140	**Acronym:**	FRAILSAFE
Full title	Sensing and predictive treatment of frailty and associated co-morbidities using advanced personalized models and advanced interventions		
Project URL	http://frailsafe-project.eu/		
EU Project officer	Mr. Jan Komarek		

Deliverable number:	4.12	**Title:**	LingTester Test Results – Passive (off-line) mode (vers a)
Work package number:	4	**Title:**	Data Management and Analytics

Date of delivery	**Contractual**	**30/06/2017 (M18)**	**Actual**	30/6/2017
Status	Draft □		Final ⊠	
Nature	Report □	Demonstrator ⊠ Other □		
Dissemination Level	Public ⊠	Consortium □		
Abstract (for dissemination)	This deliverable reports on the choices made in the design of the prediction model of the LingTester tool and its regarding Test results. The main topics discussed are feature extraction/selection techniques, classification methods and Evaluation metrics. Firstly is given an overall introduction to the classification concepts and features selected; secondly a detailed test & evaluation of the results. Also, suicidal model tendency is discussed and analysed.			
Keywords	frailty, frailty classification, natural language processing, suicidal tendency			

Contributing authors (beneficiaries)	Tsimpouris Charalampos(UoP) Fazakis Nikos (UoP) Sgarbas Kyriakos (UoP) Megalooikonomou Vasileios (UoP)			
Responsible author(s)	Sgarbas Kyriakos		**Email**	sgarbas@upatras.gr
	Beneficiary	UoP	**Phone**	+30 2610 996470

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ANNEXES

1. INTRODUCTION

As this is a preliminary report, the main focus of this deliverable is to finalize the first well performing predictive model taking in account the new populated participants data. The evaluation of the model is also an important aspect of the report, for this reason a satisfactory number of algorithms has been tested and compared. The evaluation of the test results is also presented and discussed on this text.

The report is organized in six main chapters (excluding the auxiliary sections). The first main chapters attempts to briefly describe frailty and the collected project data related to the task. The next chapter studies the feature extraction techniques and methodologies used on LingTester tool. Following feature extraction, the next chapter focuses on the predictive model creation process. On chapter five, an introduction is given for the subject of suicidal tendencies detection and the methodologies followed to build the relevant model. Test and evaluation results are the subject of the next chapter which is numbered as section six. Finally, a discussion on the ethics and safety, related to the task, is made.

2. FRAILTY

2.1 General frailty description

Frailty is a common clinical syndrome in older adults that carries an increased risk for poor health outcomes including falls, incident disability, hospitalization, and mortality [7, 8]. Frailty is theoretically defined as a clinically recognizable state of increased vulnerability resulting from aging-associated decline in reserve and function across multiple physiologic systems such that the ability to cope with everyday or acute stressors is comprised. In the absence of a gold standard, frailty has been operationally defined by Fried et al. as meeting three out of five phenotypic criteria indicating compromised energetics: low grip strength, low energy, slowed walking speed, low physical activity, and/or unintentional weight loss [9]. A pre-frail stage, in which one or two criteria are present, identifies a subset at high risk of progressing to frailty. Various adaptations of Fried's clinical phenotype have emerged in the literature, which were often motivated by available measures in specific studies rather than meaningful conceptual differences.

2.2 Collected data

Utilising eCRF API, we were able to retrieve all available raw data, stored by each medical team, containing detailed answers to the questionnaire, along with uploaded files of present and past text. For the purposes of our research, we retrieved only submissions with at least one file uploaded, and proceed to verify the uploaded content. Results were the following:

- Greece (UoP): 126 participants
    - 90 participants from the Start group

    - 36 participants from the Main group
- Cyprus (Materia): 96 participants
    - 55 participants from the Start group
    - 41 participants from the Main group
- France (INSERM): 121 participants
    - 75 participants from the Start group
    - 44 participants from the Main group

For the aforementioned submissions, however, and after manual validation, we imported to our internal database (for details please read D4.10, chapter 4) only the ones that text was digitally available, discarding during this phase all images or PDF files containing scanned images. As a result, submissions imported for the next phase were the following:

- Greece (UoP): 103
- Cyprus (Materia): 52 patients

3. FEATURE EXTRACTION

The feature extraction task is an essential pre-classification task. The FrailSafe project collects a wide range of participants' data, these data to be utilized by the candidate model must first be appropriately processed. Turning raw data into objects that patterns can be derived from is the process of creating features. A feature is simply an individual measurable property of a phenomenon being observed. Depending on the data source there are potentially dozens of different features that can be created from a single block of data.

The feature extraction task starts from an initial set of measured data and builds derived values (features) intended to be informative and nonredundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction involves reducing the amount of resources required to describe a large set of data. When performing analysis of complex data one of the major problems stems from the number of variables involved. Analysis with a large number of variables generally requires a large amount of memory and computation power, also it may cause a classification algorithm to overfit to training samples and generalize poorly to new samples. Feature extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy.

3.1 Primitive features

Apart from existing features as stated in D4.10 chapter 5.1, extra ones were extracted programmatically for all patients with digital text from eCRF:

1. Year of birth

2. Profession
3. Habitation zone
4. How many people do you follow on Twitter?
5. How many followers you have on twitter?
6. Family status
7. How many friends do you have on Facebook?
8. Do you consider yourself a familiar user of social media?
9. Do you use Facebook?
10. How often do you connect to the internet per week?
11. Have you changed your security settings in social media in order to protect your personal data?

We decided to retrieve only patient attributes that could be available directly or indirectly in the future through social media networks or other means to identify users' electronic footprint. For some of the aforementioned attributes accessed through the questionnaire, we went through a primary analysis of the structured data using the statistical tool SPSS[1], a software package used for logical batched and non-batched statistical analysis. Not all attributes were selected, as it is shown also in the following paragraphs, due to the fact that missing values were our primary obstacle to extract valuable outcomes.

Our first priority was to conduct an appropriate analysis which depends on the gathered data and the intended goal. Our steps:

[1] https://www.ibm.com/analytics/us/en/technology/spss/

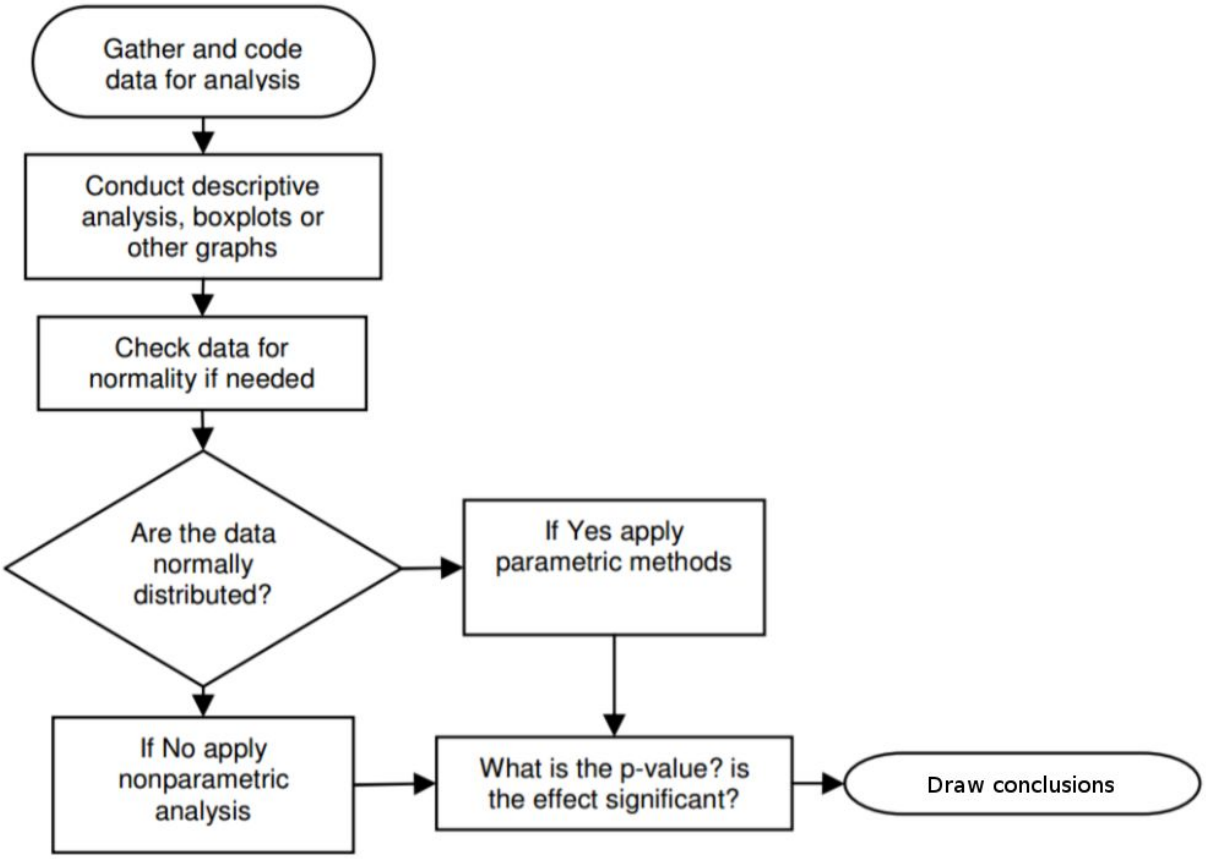

Figure 1: Steps of initial analysis

Q: How often do you connect to internet per week

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
How often do you connect to internet per week	31	1	7	5.06	2.394
Valid N (listwise)	31				

31 people answered (N=31). Minimum access to the internet is 1 per week while maximum is 7 per week. The average is Mean=5.06 and the spread out of the data is Std=2.394. While this is something answered by the patient, in theory it could be derived through API by each social media network.

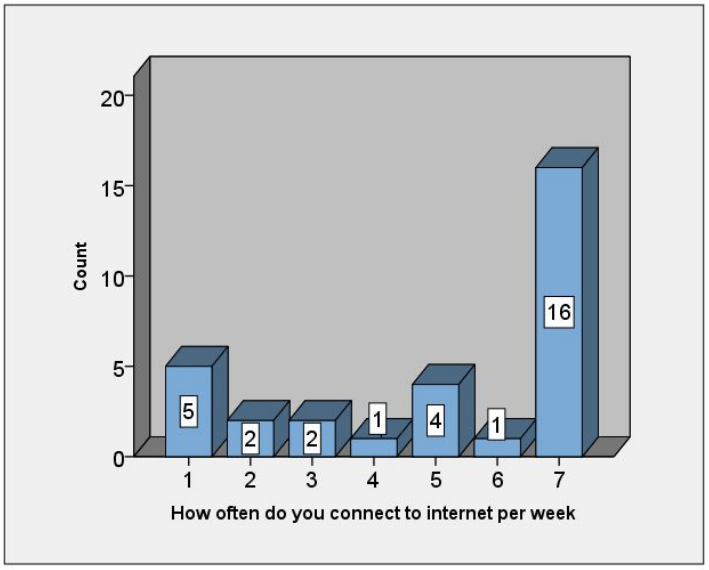

Figure 2: Distribution of answers

Q: How many contacts do you have on facebook

7 people answered (N=7). Minimum number of contacts is 20 while maximum is 9999. The average is Mean=244.14 and the spread out of the data is Std=361.58 .

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
How many contacts do you have on facebook	7	20	999	244.14	361.585
Valid N (listwise)	7				

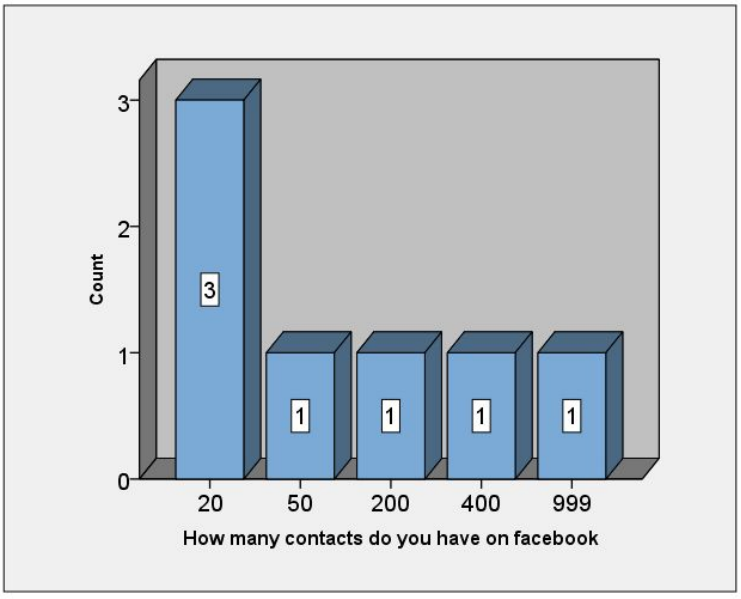

Figure 3: Distribution of answers

Q: Does the number of internet connection depends on how healthy they feel (nonfrail, prefrail, frail). The answer at first is that there is no correlation. Though, the results are not dependable due to many missing values.

Case Processing Summary

		Cases					
		Valid		Missing		Total	
	Tag	N	Percent	N	Percent	N	Percent
How often do you connect to internet per week	frail	2	3.4%	56	96.6%	58	100.0%
	missing	2	8.3%	22	91.7%	24	100.0%
	nonfrail	18	40.9%	26	59.1%	44	100.0%
	prefrail	9	12.3%	64	87.7%	73	100.0%

Descriptives

	Tag		Statistic	Std. Error
How often do you connect to internet per week	frail	Mean	4.50	2.500
		Median	4.50	
		Std. Deviation	3.536	
		Minimum	2	
		Maximum	7	
		Range	5	
	missing	Mean	5.00	2.000
		Median	5.00	
		Std. Deviation	2.828	
		Minimum	3	
		Maximum	7	
		Range	4	
	nonfrail	Mean	5.11	.577
		Median	7.00	
		Std. Deviation	2.447	
		Minimum	1	
		Maximum	7	
		Range	6	
	prefrail	Mean	5.11	.824
		Median	6.00	
		Std. Deviation	2.472	
		Minimum	1	

		Maximum	7	
		Range	6	

Table 1: Distribution of question “How often do you connect to internet per week” against frailty tag

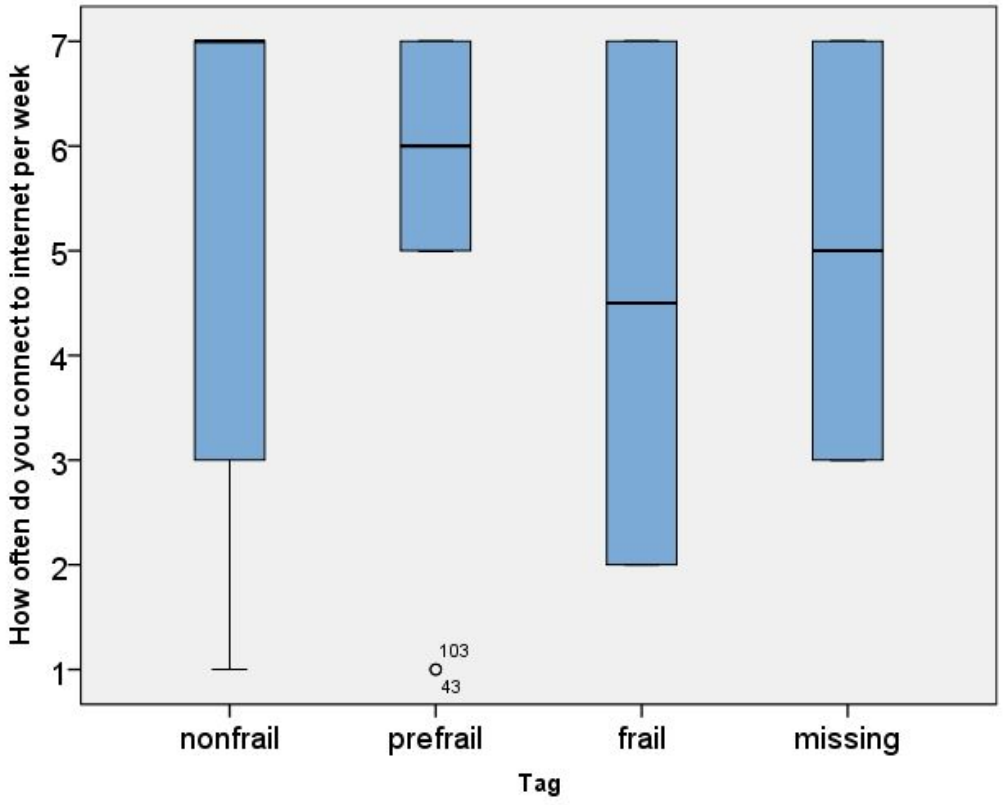

Figure 4: Distribution of answers for the question “How often do you connect to internet per week” per frailty tag

Q: Women and men have similarities concerning nonfrail and prefrail situations but female tend to be more frail at 23.6% than men 9.8%.

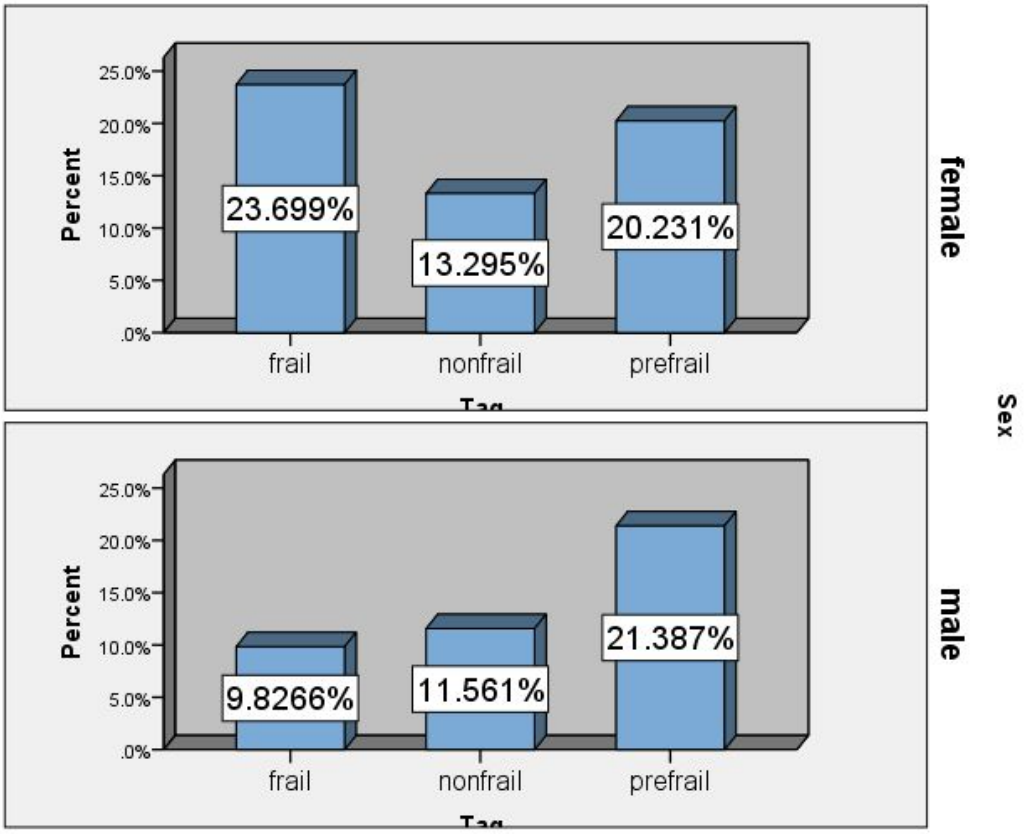

Figure 5: Distribution of participants' sex against frailty tag

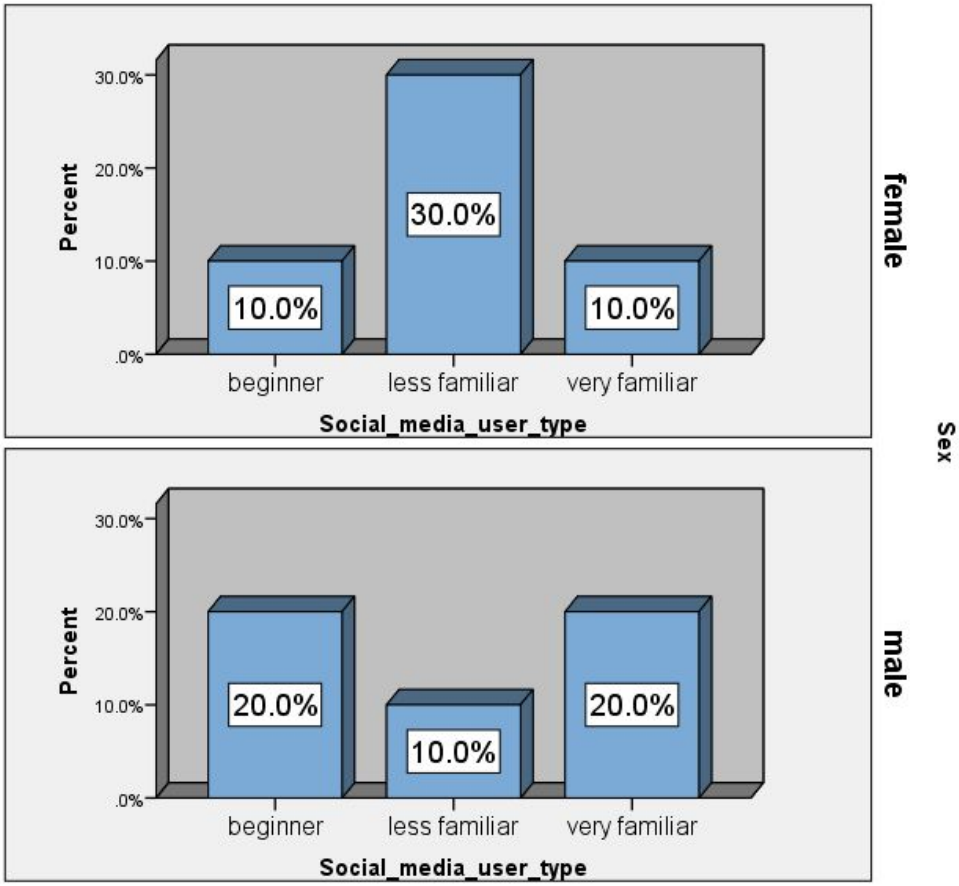

Figure 6: Distribution of social media user type against participants' sex

Due to missing values, as stated in the beginning of this chapter, we were unable to extract more results from these questions.

3.2 Derived Features

3.2.1 Misspellings

This feature is derived based on the percentage of misspellings found within a text, divided by all words. In order to achieve high accuracy, a known dictionary is used per language. The same dictionary is used by thousands of people that utilise LibreOffice[2], an open office suite of applications. LibreOffice is community-driven and developed software, and is a project of the not-for-profit organization, The Document Foundation. LibreOffice is free and open source software, originally based on OpenOffice.org (commonly known as OpenOffice), and is the most actively developed OpenOffice.org successor project.

3.2.2 Term Frequencies

Proceeding to more NLP specific techniques the term frequency–inverse document frequency (tf-idf) is used. Tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a corpus. It is used as a weighting factor in text mining. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general (Salton et al, 1983). However, it became apparent that the use of the full list of the tf-idf features, which has decreased due to the stemming process, produces substantial error to the prediction model, and thus was removed.

In order to overcome this issue, information gain (IG), also known as Mutual Information, has been used to extract only the terms (or small phrases) that indeed can help the prediction model improve in the overall accuracy. In general terms, the expected information gain is the change in information entropy ***H*** from a prior state to a state that takes some information as given:

$$IG(T, a) = H(T) - H(T|a)$$

Information gain for classification is a measure of how common a feature is in a particular class compared to how common it is in all other classes. A word that occurs primarily in non-frail patients and rarely in frail ones is high information. That makes sense because the point is to use only the most informative features and ignore the rest.

One of the best metrics for information gain is chi square [1, 3]. NLTK package, already used for other preprocessing tasks in more than one work packages, includes this in the *BigramAssocMeasures* class in the metrics package. To use it, first we need to calculate a few frequencies for each word: its overall frequency and its frequency within each class. This

[2] https://www.libreoffice.org/

is done with a *FreqDist class* for overall frequency of words, and a *ConditionalFreqDist class* where the conditions are the class labels. Once we have those numbers, we can score words with the *BigramAssocMeasures.chi_sq* function, then sort the words by score and take the top X. We then put these words into a set, and use a set membership test in our feature selection function to select only those words that appear in the set. Now each text from a patient is classified based on the presence of these high information words. This process, has been constructed within the `compute_ig` python function, available also for other tasks. An example, of this task is shown below. It can be seen, that information gain is different per feature (word n our case), and thus is more than apparent that taking a small specific percent of this feature set, minimises the noise included by the rest of the words.

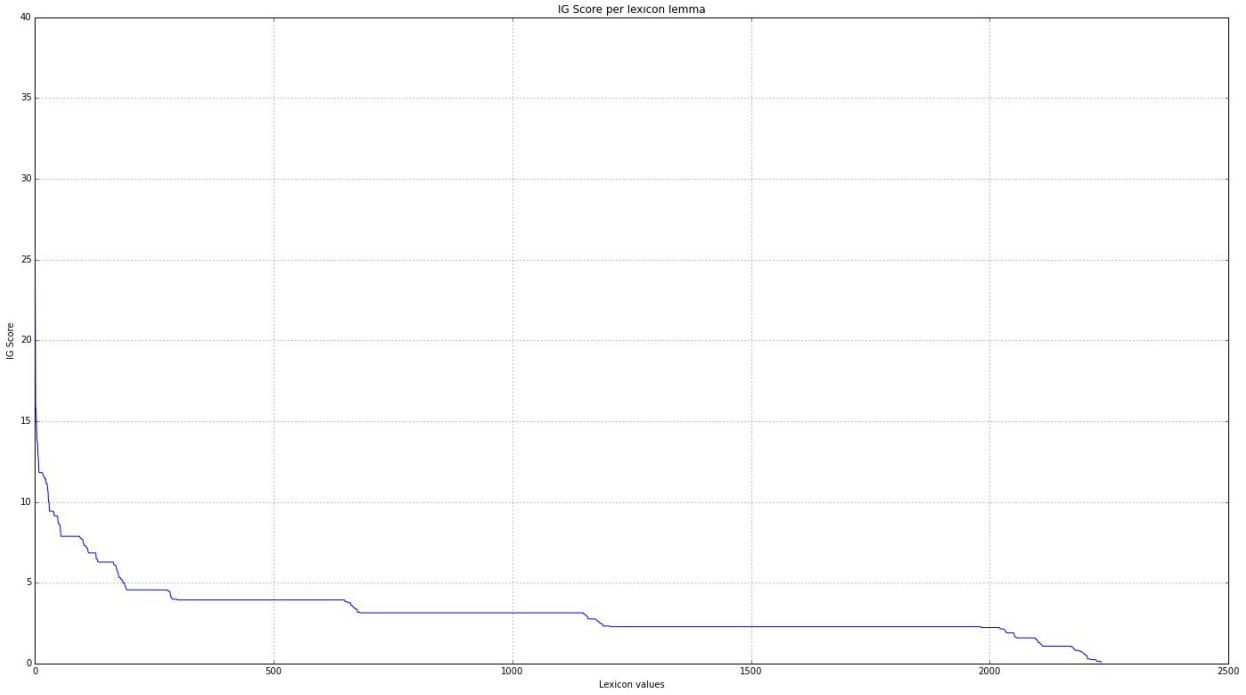

Figure 7: Distribution of information gain per available token

3.2.3 Sentiment analysis

Sentiment analysis is based on a overall rating sentiment of the patient's text, trying to detect the polarity of the text. This feature is extracted using a dictionary and a basic algorithm which calculates the polarity of the text based on the sum of all polarities of words within the same text. The dictionary used can be found within the `pattern` python module and is available for the English language. Due to missing sentiment vocabulary for other languages, text is automatically translated in English for this step.

3.2.3 Readability score

We decided that readability score may improve our prediction model, and we proceed to implement a series of various models that calculate readability, and see which one makes

the difference. All methods measure textual difficulty, which indicates how easy a text is to read. The following readability scores have been implemented.

Flesch Reading Ease

The Flesch Reading Ease Scale measures readability as follows:

- 100: Very easy to read. Average sentence length is 12 words or fewer. No words of more than two syllables.
- 65: Plain English. Average sentence is 15 to 20 words long. Average word has two syllables.
- 30: A little hard to read. Sentences will have mostly 25 words. Two syllables usually.
- 0: Very hard to read. Average sentence is 37 words long. Average word has more than two syllables.

The higher the rating, the easier the text is to understand. By the very nature of technical subject matter, the Flesch score is usually relatively low for technical documentation. If the Flesch test is used regularly, one may develop a sense of what a reasonable score is for the type of documentation one is working on and aim to align with this score. The approach to calculating the Flesch score is as follows:

1. Calculate the average sentence length, L.
2. Calculate the average number of syllables per word, N.
3. Calculate score (between 0-100%).

SMOG Index

The SMOG grade (Simple Measure of Gobbledygook) is a measure of readability that estimates the years of education needed to understand a piece of writing. The formula for calculating the SMOG grade was developed by G. Harry McLaughlin as a more accurate and more easily calculated substitute for the Gunning fog index and published in 1969. To make calculating a text's readability as simple as possible an approximate formula was also given — count the words of three or more syllables in three 10-sentence samples, estimate the count's square root (from the nearest perfect square), and add 3.

To calculate SMOG:

1. Count a number of sentences (at least 30)
2. In those sentences, count the polysyllables (words of 3 or more syllables).
3. Calculate using

$$\text{grade} = 1.0430\sqrt{\text{number of polysyllables} \times \frac{30}{\text{number of sentences}}} + 3.1291$$

After numerous tests, this feature seem not affect the prediction model, as it was made apparent that Greek language is not fully supported by the underlying NLTK functions.

Flesch–Kincaid Grade Level

Although this method uses the same core measures (word length and sentence length) like the Flesch Reading Ease, they have different weighting factors. The results of the two tests

correlate approximately inversely: a text with a comparatively high score on the Reading Ease test should have a lower score on the Grade-Level test. These readability tests are used extensively in the field of education. The "Flesch–Kincaid Grade Level Formula" instead presents a score as a U.S. grade level, making it easier for teachers, parents, librarians, and others to judge the readability level of various books and texts. It can also mean the number of years of education generally required to understand this text, relevant when the formula results in a number greater than 10. The grade level is calculated with the following formula:

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right)+11.8\left(\frac{\text{total syllables}}{\text{total words}}\right)-15.59$$

Coleman–Liau index

The Coleman–Liau index is a readability test designed by Meri Coleman and T. L. Liau to gauge the understandability of a text. Like the Flesch–Kincaid Grade Level, Gunning fog index, SMOG index, and Automated Readability Index, its output approximates the U.S. grade level thought necessary to comprehend the text.

The Coleman–Liau index was designed to be easily calculated mechanically from samples of hard-copy text. Unlike syllable-based readability indices, it does not require that the character content of words be analyzed, only their length in characters. Therefore, it could be used in conjunction with theoretically simple mechanical scanners that would only need to recognize character, word, and sentence boundaries, removing the need for full optical character recognition or manual keypunching. The Coleman–Liau index is calculated with the following formula:

$$CLI=0.0588L-0.296S-15.8$$

L is the average number of letters per 100 words and S is the average number of sentences per 100 words.

Automated readability index

The formula for calculating the automated readability index is given below:

$$4.71\left(\frac{\text{characters}}{\text{words}}\right)+0.5\left(\frac{\text{words}}{\text{sentences}}\right)-21.43$$

where characters is the number of letters and numbers, words is the number of spaces, and sentences is the number of sentences, which were counted manually by the typist when the above formula was developed. Non-integer scores are always rounded up to the nearest whole number, so a score of 10.1 or 10.6 would be converted to 11.

Dale–Chall readability formula

The Dale–Chall readability formula is a readability test that provides a numeric gauge of the comprehension difficulty that readers come upon when reading a text. It uses a list of 3000 words that groups of fourth-grade American students could reliably understand, considering any word not on that list to be difficult. The formula for calculating the raw score of the Dale–Chall readability score is given below:

$$0.1579\left(\frac{\text{difficult words}}{\text{words}} \times 100\right) + 0.0496\left(\frac{\text{words}}{\text{sentences}}\right)$$

Linsear Write

Linsear Write is a readability metric for English text, purportedly developed for the United States Air Force to help them calculate the readability of their technical manuals.It was specifically designed to calculate the United States grade level of a text sample based on sentence length and the number of words used that have three or more syllables. The algorithm to calculate Linsear Write score includes various steps of preprocessing and statistical analysis of the text.

Gunning fog index

The Gunning fog index is calculated with the following algorithm:

1. Determine the average sentence length. (Divide the number of words by the number of sentences.)
2. Count the "complex" words: those with three or more syllables. Do not include proper nouns, familiar jargon, or compound words. Do not include common suffixes (such as -es, -ed, or -ing) as a syllable
3. Add the average sentence length and the percentage of complex words; and
4. Multiply the result by 0.4.

The complete formula is:

$$0.4\left[\left(\frac{\text{words}}{\text{sentences}}\right) + 100\left(\frac{\text{complex words}}{\text{words}}\right)\right]$$

Discussion about readability scores

All aforementioned scores are heavily used in English by teachers, writers and technicians to identify the difficulty of a text, manual and try to simplify it. Within the current work package, however, we tried to use them towards a different direction and identify how frailty is connected to the readability of the expressed text, provided by a patient. This is a hypothesis that hasn’t been discarded yet and needs to be thoroughly tested the following months, after improving the aforementioned readability scores to support better the Greek language.

4. PREDICTION MODEL

4.1 Introduction

In this section of the report, will be presented a brief introduction of the major artificial intelligence techniques and methodologies that are used to produce predictive models. These methodologies can be found in the literature grouped under the term machine learning[10].

4.2 Machine Learning

4.2.1 History

The field inherits its methodologies from mathematics and statistics. The first real form of machine learning is discovered around 1950s with the very well known ‘’Turing Test’’[11], some of the most basic algorithms like ‘’Nearest neighbours’’, ‘’Back propagation’’ and ‘’Support vector machines’’ are getting discovered from 1970s through 1990s. Near this time the machine learning approach shifts from knowledge-driven to a more data-driven approach. After 2000s, technological advancements in computer chips give the ability to machine learning algorithms to utilize parallel processing and big data. Neural networks can now be run in a big number of cpu cores and this new methodology advances to become the ‘’Deep learning’’[12] approach. In the following years machine learning models are widely adopted in software programs, giving great solutions to problems in almost any real world sector.

4.2.2 The process

The construction of a prediction model involves a number of tasks, basically the figure below summarizes the whole process.

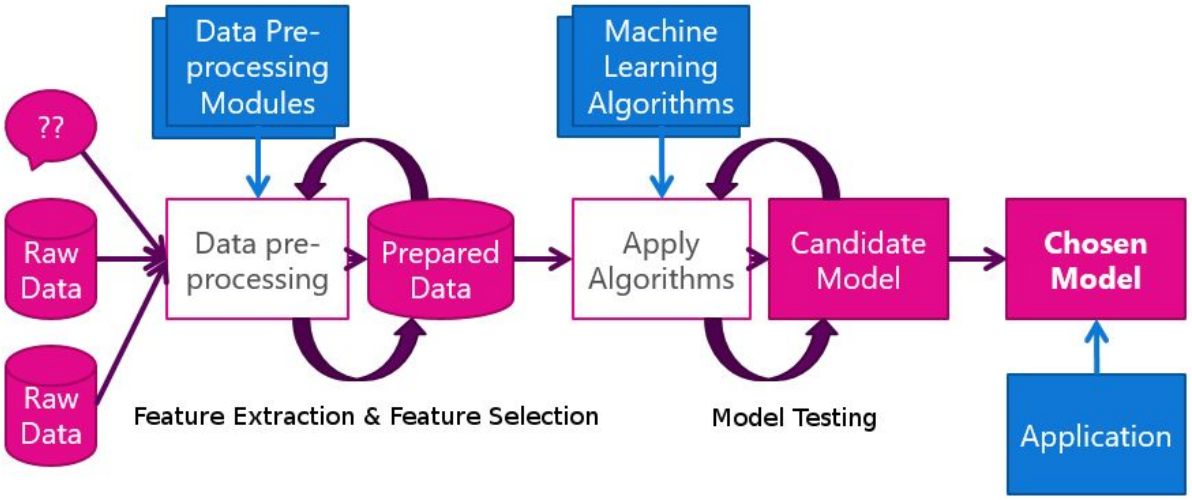

Figure 8: Process for the construction of a prediction model

The first phase to the construction of a model is the raw data collection. FrailSafe project involves several number of teams under the umbrellas of different work packages in order to collect and parse all the related project data including clinical trials, hardware sensors data and higher level (analyzed) data. All the collected is potential input to the LingTester predictive model thus it has to be analyzed by the tasks of Feature extraction and Feature Selection which will be presented separately in the following sections. The next step of the process is the classification task, in this task a big number of available classification algorithms is applied and tested in order to finalize the candidate model giving the best predictive abilities. The finalized model is afterwards ready to be applied on real world instances.

4.3 WEKA Software Package

4.2.2 Description & features

WEKA[13] Software is one of the most known packages used researchers and developers, it provides implementations of learning algorithms that you be easily applied to any dataset. It also includes a variety of tools for transforming datasets, such as the algorithms for discretization and sampling. A standard use casio scenario includes the preprocess of a dataset, the feed into a learning scheme, and the analysis of the resulting classifier and its performance. It was worth noting that every use flow the user wants to implement can be achieved by simply utilizing the GUI tools that accompany weka or in more complex scenarios, the user can access or alter directly the implemented source code through its own applications. Summarizing the most important features of the software:

- It is Freely available under the GNU General Public License.
- It is Portable, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform.
- It includes a comprehensive collection of data preprocessing and modeling techniques.
- It is Easy to use or alter due to its graphical user interfaces and extensible libraries.

4.2.2 The Explorer package

The explorer package is weka's main graphical user interface, it gives access to all its facilities using menu selection and form filling. It is illustrated in figure 9. To begin, there are six different panels, selected by the tabs at the top, corresponding to the various data mining tasks that weka supports. Further panels can become available by installing appropriate packages.

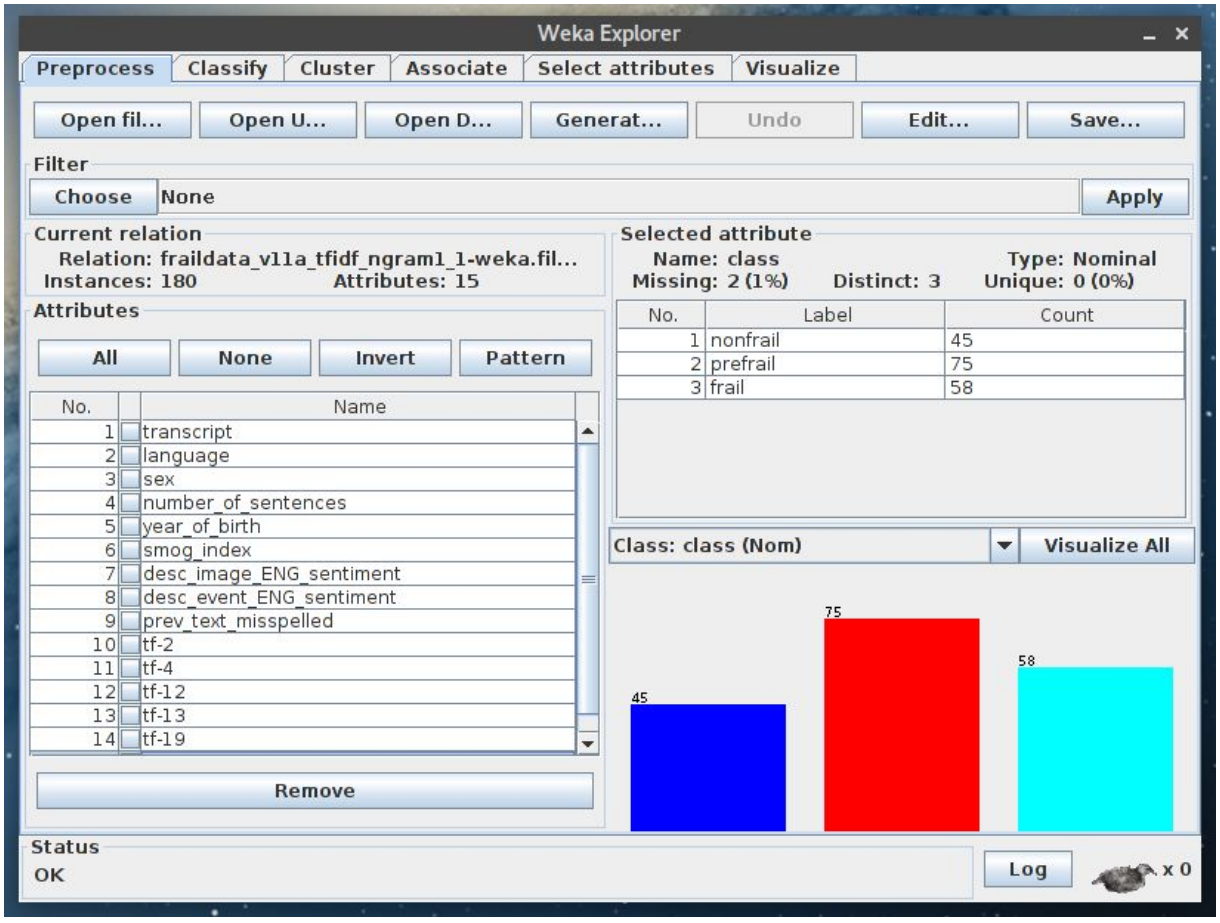

Figure 9: WEKA explorer package

All the classification task analysis done in this and the related reports was conducted using mainly using the explorer package. The tab *Preprocess* was used to alter accordingly the training dataset. For the evaluation of the available attributes the selection tab was used. The learner algorithms that were tested, were found on the Classify tab where the evaluation results are also found.

4.4 Feature extraction

For the essential task of feature extraction, a python algorithm was developed. This script is highly complex and employs all the techniques that have already been discussed in deliverable 4.10. For better understanding and presentation, available features can be categorized by the extraction method that was used for their creation. The table below attempts to summarize this information.

Feature Names	Type - Extraction Method

• transcript
  ○ yes
  ○ no
• language
	Primitive
	Rules & filters on eCRF API data

○ greek ○ greek-cypriot ○ french ● class ○ nonfrail ○ prefrail ○ frail ● sex ○ male ○ female ● do_you_consider_yourself_a_familiar_user_of_social_media ○ beginner ○ less-familiar ○ very-familiar ● family_status ○ married-or-in-a-relationship ○ single ○ divorced, ○ widow ● habitation_zone ○ urban ○ semi-urban ○ rural ● have_you_changed_your_security_settings_in_social_media_in_order_to_protect_your_personal_data ○ yes ○ no ● year_of_birth ● con_per_week *connections per week* ● twitter_follows *number if people user is following on Twitter* ● twitter_followers *number of followers on Twitter* ● fb_friends *number of friends on FB*	
● text_length ● number_of_sentences ● number_of_words ● number_of_words_per_sentence ● text_entropy	**Derived** Statistical Measures
● desc_image_ENG_sentiment ● desc_event_ENG_sentiment ● prev_text_ENG_sentiment	**Derived** Sentiment Analysis
● desc_image_misspelled ● desc_event_misspelled ● prev_text_misspelled	**Derived** Percent of misspelled words based on known vocabulary

● tf-0 ● tf-1 ● ...	**Derived** Term frequency – Inverse document frequency, after feature selection based on information gain
● flesch_reading_ease ● smog_index ● flesch_kincaid_grade ● coleman_liau_index ● automated_readability_index ● dale_chall_readability_score ● difficult_words ● linsear_write_formula ● gunning_fog	**Derived** Readability score

Table 2: List of features

4.5 Feature selection

Feature selection along with feature extraction are, as stated today in the majority of literature, the most essential tasks for creating a well performing predictive model. The prediction accuracy of a trained learner is directly depended on how much informative the selected model features are.

The feature selection process, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. The central premise when using a feature selection technique is that the data contains many features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information [14]. Redundant or irrelevant features are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated. A number of techniques have been proposed in the literature using algorithms and even classifiers for automating the process of feature selection. The most common algorithms are the exhaustive, best first [15], simulated annealing [16] and the genetic algorithm [17]. In practice, the task of feature selection is a highly empirical process where algorithms and human intelligence are combined in order to find the optimal subset of features, thus constructing the final feature set that will be used in the classification task.

As a first phase of the feature selection process we attempt to `Rank' the available extracted features. This is achieved through the use of the weka algorithm called `OneR Attribute Evaluator'. The algorithm utilizes the `One R' [18] machine learning classifier. In general, the classifier attempts to find the best feature available on the feature space, the exact algorithm follows next.

One R Classifier

Input:
Load the complete set of features (C)
Count the number of all features (N)

Loop for N
For each feature-class pair (P),
For each value of that P, make a rule:
Count how often each value of target (class) appears
Find the most frequent class
Make the rule assign that class to this value of the P
Calculate the total error of the rules of each P

Output:
Choose the P with the smallest total error.

The Evaluator algorithm uses a small variation of the classifier taking in account other evaluation metrics than the total error, like ReliefF, GainRatio, Entropy e.t.c., though the use of a Ranker package. The Attribute Evaluator algorithm was run and the results obtained are summarized in the next table.

Attribute selection output:

```
=== Attribute Selection on all input data ===

Search Method:
         Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 54 class):
         OneR feature evaluator.

         Using training data for evaluation of attributes.
         Minimum bucket size for OneR: 6

Ranked attributes:

58.427   14 year_of_birth
53.371   32 desc_event_misspelled
52.809    1 transcript
52.247    9 text_length
52.247   23 automated_readability_index
51.685   11 number_of_words
51.124   10 number_of_sentences
50.562   13 text_entropy
50       19 flesch_reading_ease
50       21 flesch_kincaid_grade
50       12 number_of_words_per_sentence
50       31 desc_image_misspelled
49.438   27 gunning_fog
48.876   24 dale_chall_readability_score
48.876   29 desc_event_ENG_sentiment
48.315   22 coleman_liau_index
47.191   33 prev_text_misspelled
47.191   15 con_per_week
46.629   26 linsear_write_formula
46.629    7 habitation_zone
```

```
45.506    5 sex
45.506    8 have_you_changed_your_security_settings_in_social_media_in_order_to_
               protect_your_personal_data
44.944   28 desc_image_ENG_sentiment
44.944   53 tf-19
44.944   46 tf-12
44.944    3 do_you_consider_yourself_a_familiar_user_of_social_media
44.382   18 fb_friends
44.382   36 tf-2
43.82    45 tf-11
43.82    51 tf-17
43.82    40 tf-6
43.82    34 tf-0
43.82    37 tf-3
43.258   50 tf-16
43.258    6 family_status
43.258   35 tf-1
43.258    4 language
43.258   42 tf-8
43.258   52 tf-18
42.697   25 difficult_words
42.697   39 tf-5
42.697   16 twitter_follows
42.697   41 tf-7
42.697   30 prev_text_ENG_sentiment
42.697   17 twitter_followers
42.697   43 tf-9
42.135   38 tf-4
42.135    2 source
42.135   49 tf-15
42.135   44 tf-10
42.135   48 tf-14
42.135   47 tf-13
42.135   20 smog_index


Selected attributes:
14,32,1,9,23,11,10,13,19,21,12,31,27,24,29,22,33,15,26,7,5,8,28,53,46,3,18,36,45,51,40,34,37,50,6,35,4,42,52,
25,39,16,41,30,17,43,38,2,49,44,48,47,20 : 53
```

To further enhance the feature selection task, a simple yet effective process has been followed. The first steps of the process involve an iteration of classifications where each individual feature was examined for its contribution to the accuracy of the temporary model, using the cross validation method [19]. After a sufficient number of iterations, the resulting decision tree was visualized and examined by hand in order to further optimize the resulting model.

Final Feature Selection Algorithm

Input:
Load the complete set of features (C)

Count the number of all features (N)
Classify with C and store the accuracy (A)
Initialize pointer as zero (P)
Loop for N
    Remove C[P]
    Classify with C (Ac)
    If Ac < A

Restore C[P]

Validate features by tree visualization

Output:
Subset of features (S)

After the successful execution of the two algorithms by combining the strongest results of the algorithms we ended up with the following selected attributes to use for the final classification task.

No.		Name
1	☐	transcript
2	☐	language
3	☐	sex
4	☐	number_of_sentences
5	☐	year_of_birth
6	☐	smog_index
7	☐	desc_image_ENG_sentiment
8	☐	desc_event_ENG_sentiment
9	☐	prev_text_misspelled
10	☐	tf-2
11	☐	tf-4
12	☐	tf-12
13	☐	tf-13
14	☐	tf-19
15	☐	class

Figure 10: Selected attributes

4.5 Classification

This section is an attempt to describe briefly the classifier models that will be used to obtain the test results in chapter 6. It will also help to understand why a classifier is better suitable than another classifier for the given problem of frailty classification.

The first big category of classifiers worth mentioning is the probabilistic or bayes classifiers. The classifiers aparting this category are based on the application of Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes[20], the most known classifier belonging in this category, has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s and remains a popular method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis. Abstractly, naive Bayes is a conditional

probability model: given a problem instance to be classified, represented by a vector **x** representing some `n' features (independent variables), it assigns to this instance probabilities calculated by the formula:

$$p(C_k \mid x_1, \ldots, x_n)$$

for each of k possible outcomes or classes C_k.

A second big category of classifiers is based on functions. The classifier family that is worth mentioning in this category, as it suits the problem, is Support Vector Machines[21]. SVMs are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. In this category of classifiers the Multilayer perceptron is also considered a strong model. MLP is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (a processing element) with a nonlinear activation function. MLP utilizes a learning technique called backpropagation for training the network.

The third category that will be mentioned is based on tree structures. Better known as Decision Trees[22], this set of classifiers mainly utilize a decision tree as a predictive model to go from observations about an item, represented in the nodes, to conclusions about the item's target value represented in the leaves. A tree can be trained by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data. An example trained tree follows in the next figure, representing the classic loan applicant acceptance decision tree.

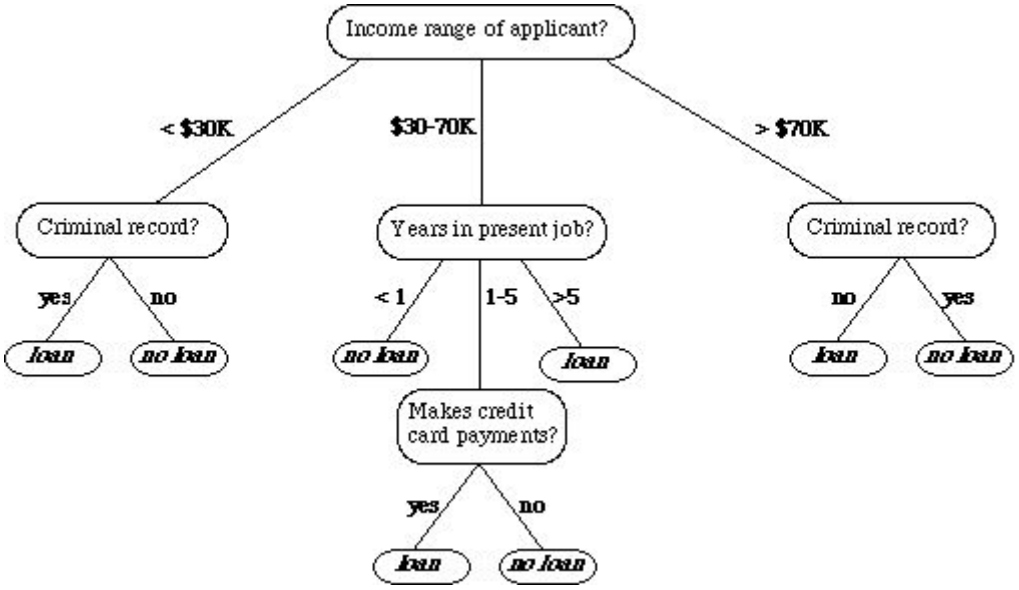

Figure 11: Example process of top-down induction of decision trees

A number of other categories exists based on specific logical rules, e.g. K-Nearest-Neighbours[23], KStar[24] and Locally weighted learning (LWL), or based on more complex combinations of the primitive models e.g. RotationForest[25], LogitBoost and utilization of techniques like bagging, voting and stacking. The number of supervised classifiers available nowadays has increased a lot, despite this section has shown the most common models, it does not intend to fully map the subject, instead it gives a brief understanding of the models deployed on the next chapters.

5. SUICIDAL TENDENCIES DETECTION

5.1 Introduction

The study of suicide risk is complex. Suicide is a medical disaster, but occurs at a relatively low frequency. The extensive body of suicide and self harm risk research has identified many risk factors, among them: diagnosed mental disorders, substance abuse, and a history of prior suicide attempts. However, much of this research has been based on retrospective study of suicide attempters and completers. The rarity of suicidal behavior limits the specificity of predicting future suicidal behavior based on relatively common risk factors. A recent survey of literature on screening tools to predict suicide attempts and death by suicide concluded that current evidence is insufficient to support reliance on screening tools based on presence or absence of risk factors in clinical practice [2], and so, the current standard of care for the at-risk patient remains in the domain of a skilled clinical assessment.

5.2 State of the art

Fine-grained automatic emotion detection can benefit from from classifier optimization and a combined lexico-semantic feature representation can achieve scores up to 68.86% F-score [4]. Corpus for model construction and prediction has been a note corpus of positive only data, annotated with fine-grained emotions, released in the framework of the 2011 i2b2 NLP Challenge on emotion classification in suicide notes [5], allowing research on which emotions might be indicative of suicidal behavior, and how they can be found automatically. However, this dataset is no longer available. Also, vocabulary based methodology, manually annotated on Twitter posts and then classification using various classifier could not produce more than 64% accuracy [6].

5.3 Database construction

According to the predefined work packages, no information could be available to identify suicidal patients, so a different approach was selected to create our dataset. We identified the Goodreads[3] web page as a potential source for our scope. Among other things, this web page provides famous quotes from known books, which have been manually saved by the Goodreads community. This manual process also forces each user to tag the quote with at least one keyword, and therefore various categories have appeared. A crawler for this purpose was constructed to identify and download all quotes set by the researchers, split in two main groups:

- suicide: quotes identified by the following keywords
    - suicide,
    - suicide-note,
    - suicide-attempt and
    - suicidal-thoughts
- non-suicide: quotes identified by the following keywords
    - humor and
    - happy

To avoid an unbalanced dataset, the second category was filled randomly till we had the same amount of quotes in both categories. After the data collection, each group has 3184 instances.

5.4 Feature extraction

Feature extraction was based on the same methodology as stated before, for the frailty prediction model. The derived feature space used for suicidal tendencies detection was intentionally kept familiar with the frailty feature space for consistency reasons. In contrast with the frailty detection process, the current task lacks the eCRF API data. Below, follows a brief description of the feature space in table 3.

[3] https://www.goodreads.com/

Feature Names	Feature Description
● class ○ suicide ○ non-suicide	goodreads crawler labels (see Annex 1 for details)
● text_length ● number_of_sentences ● number_of_words ● number_of_words_per_sentence ● text_entropy	Statistical Measures
● sentiment	Sentiment Analysis of text data
● misspelled	Percent of misspelled words based on known vocabulary
● tf-0 ● tf-1 ● ...	Term frequency – Inverse document frequency, after feature selection based on information gain
● flesch_reading_ease ● smog_index ● flesch_kincaid_grade ● coleman_liau_index ● automated_readability_index ● dale_chall_readability_score ● difficult_words ● linsear_write_formula ● gunning_fog	Readability score

Table 3: List of features

5.5 Information gain and Feature selection

Also, feature selection and information gain was implemented to lower the dimensionality of features, while also removing excessive noise from the Tf-IDF feature extraction.

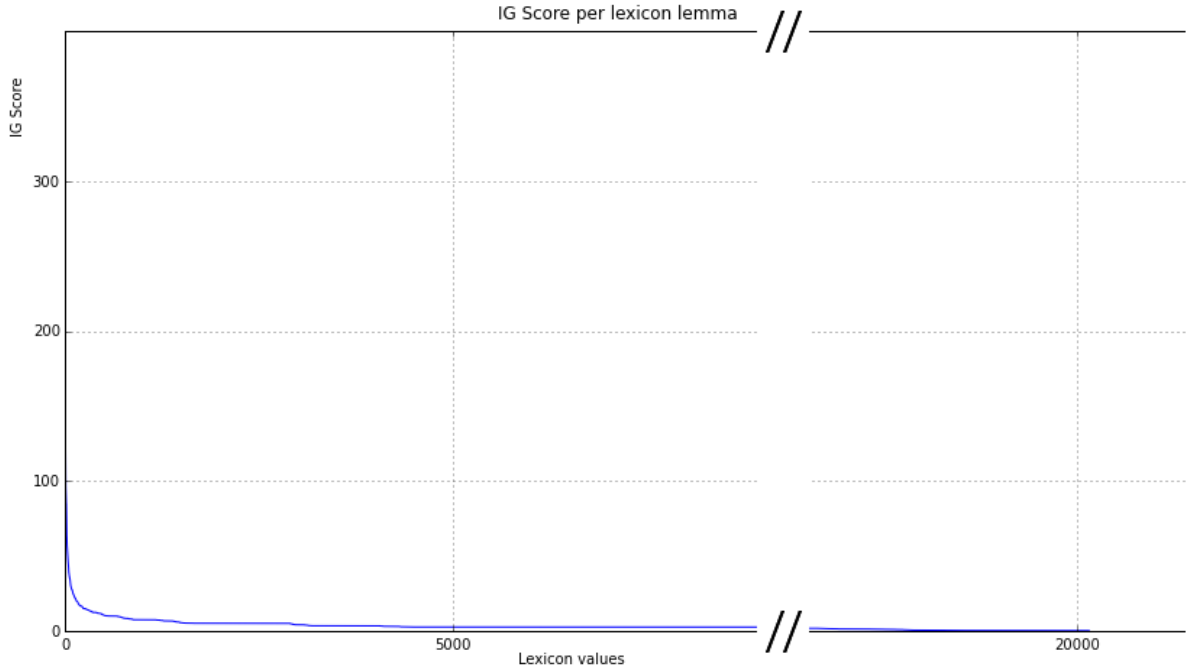

Figure 12: Distribution of information gain per available feature for the suicidal prediction model, based on Goodreads manually saved quotes.

5.6 Suicide tendency prediction

In order to detect suicidal signs in written text, a series of tests on classification models had to be run. This section, has to be mentioned, that is a work in progress and will be finalized in the final report.

The classifiers deployed where the very common in text classification `Naive Bayes', the classic `Decision Trees' implementation (J48) and the more complex trees model known as `Rotation Forest'. The tests ran on the extracted feature space mentioned in section 5.4 using 10 - cross validation on the training dataset constructed as explained in the previous sections.

The results of the experiment are presented in detail below. The following table summarizes the accuracies of the models along with the figure, giving a better visual comparison. Next, the detailed statistical measures of best performing model are shown.

Classifier	Accuracy %
NaiveBayes	68.35
J48	69.26
RotationForest	**71.04**

Table 4: Suicide tendency prediction accuracy per classifier

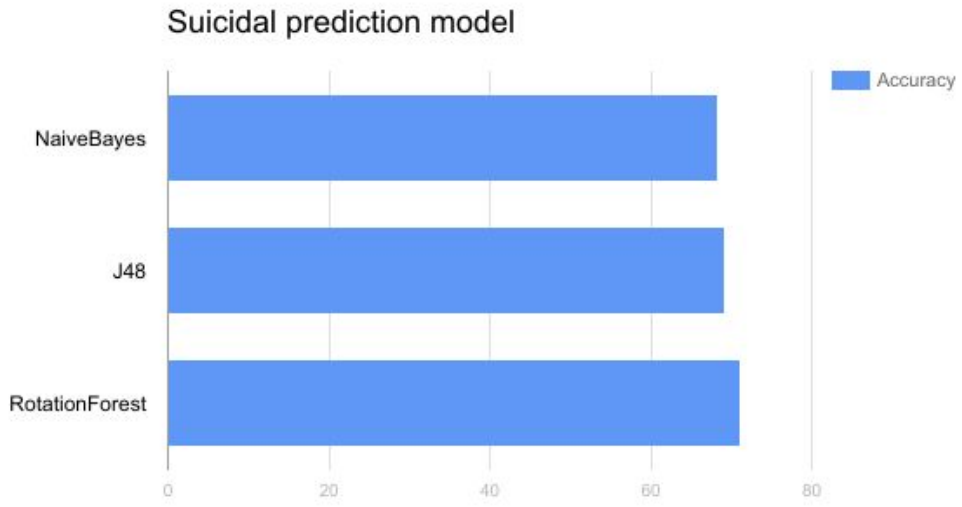

Figure 13: Suicide tendency prediction accuracy per classifier

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        4526               71.0407 %
Incorrectly Classified Instances      1845               28.9593 %
Kappa statistic                          0.4208
Mean absolute error                      0.3698
Root mean squared error                  0.4306
Relative absolute error                 73.9596 %
Root relative squared error             86.1218 %
Total Number of Instances             6371

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.61      0.19       0.763     0.61      0.678      0.784    suicide
                 0.81      0.39       0.675     0.81      0.737      0.784    non-suicide
Weighted Avg.    0.71      0.29       0.719     0.71      0.707      0.784

=== Confusion Matrix ===

    a    b   <-- classified as
 1944 1241 |    a = suicide
  604 2582 |    b = non-suicide
```

Figure 14: detailed statistical measures of Rotation Forest performing model

The Rotation Forest model seems to be superior according to the accuracy comparison of the models. Thus, the relating subject is very sensitive and important, decision to alert the involved people can not be made only by judging the accuracy metric. As this is a preliminary report, more work will be done in this direction to ensure the most appropriate metrics are used to give a more clear picture of the situation. Some of them involve, the Distribution of probabilities for the classified instance, false positives, false negatives and generally precision and recall analysis.

6. TEST RESULTS

6.1 Introduction

The major subject of this report is the test results of the predictive model. In reality, everything done in the task is heavily tied with the performance increasement of the final model. As has been emphasized in chapter 4, the performance of a model is equally influenced by all the previous tasks of the classification process. The parts of feature extraction and feature selection are considered as highly important factors of the final predictive accuracy and require fairly more expensive resources like human expertise. As the above factors are not a hundred percent controlled by this task, a very strong and extensive testing process has to be done in order to select the final classification model and ensure the best possible prediction accuracy.

This chapter presents the evaluation process followed to obtain the test results, the comparison of the various models, the parameters of the selected model and the statistics of the final model. Finally, this chapter ends with a brief discussion of the test results, as this is a preliminary report and the model is expected to change, and also a few words about the future steps of development.

6.2 Evaluation

Machine learning offers a wide range of algorithms that can fit the data and build a predictive model. In order to test the possible model performances and select the most suitable for the case, an extensive experiment was organized. Totally, seventeen of the most widely used classifiers were selected from six different algorithm families. The measured metric was that of classification accuracy.

The evaluation method that was used is the classic method of Cross Validation[26]. In general, cross validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds. One of the main reasons for using cross-validation instead of using the conventional validation (e.g. partitioning the data set into two sets of 70% for training and 30% for test) is that there is not enough data available to partition it into separate training and test sets without losing significant modelling or testing capability. In summary, cross-validation

combines (averages) measures of fit (prediction error) to derive a more accurate estimate of model prediction performance.

The table that follows presents the test results, measured in accuracy percent, for the extensive tested scenario, using the 10 fold cross validation method.

Classifier	Accuracy %
NaiveBayes	54.49
SVM	52.24
Logistic Regression	60.67
MultilayerPerceptron	59.55
SMO	57.86
IBk	56.17
KStar	52.80
LWL	57.86
RotationForest	58.98
OneR	52.24
ZeroR	42.13
DecisionStump	51.12
J48	**64.04**
LMT	60.11
RandomForest	56.74
RandomTree	51.12
REPTree	47.19

Table 5: Frailty prediction results per prediction model

Most of the tested models seem to be performing moderately with an average accuracy of 55.01% with a standard deviation of 5.44%. The outcome was very logical as frailty prediction is a very difficult and sensitive problem and is highly correlated to the constructed dataset.

The exception that one can immediately spot on the table is the J48 model which is the implementation of the classic Decision Trees classifier. This phenomenally simple algorithm achieved to perform almost 10% above the average accuracy scoring a 64.04%. Below are

presented the figures of accuracy distribution and the accuracy visualization for better understanding of the results.

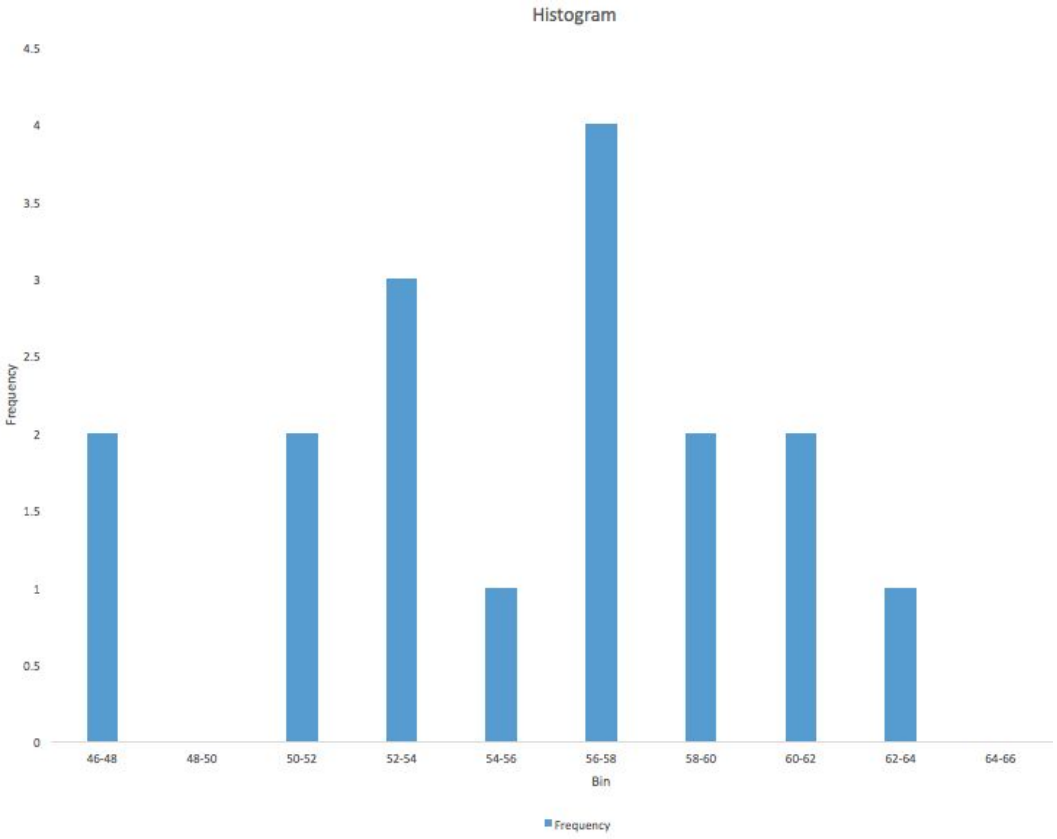

Figure 15: Accuracy distribution

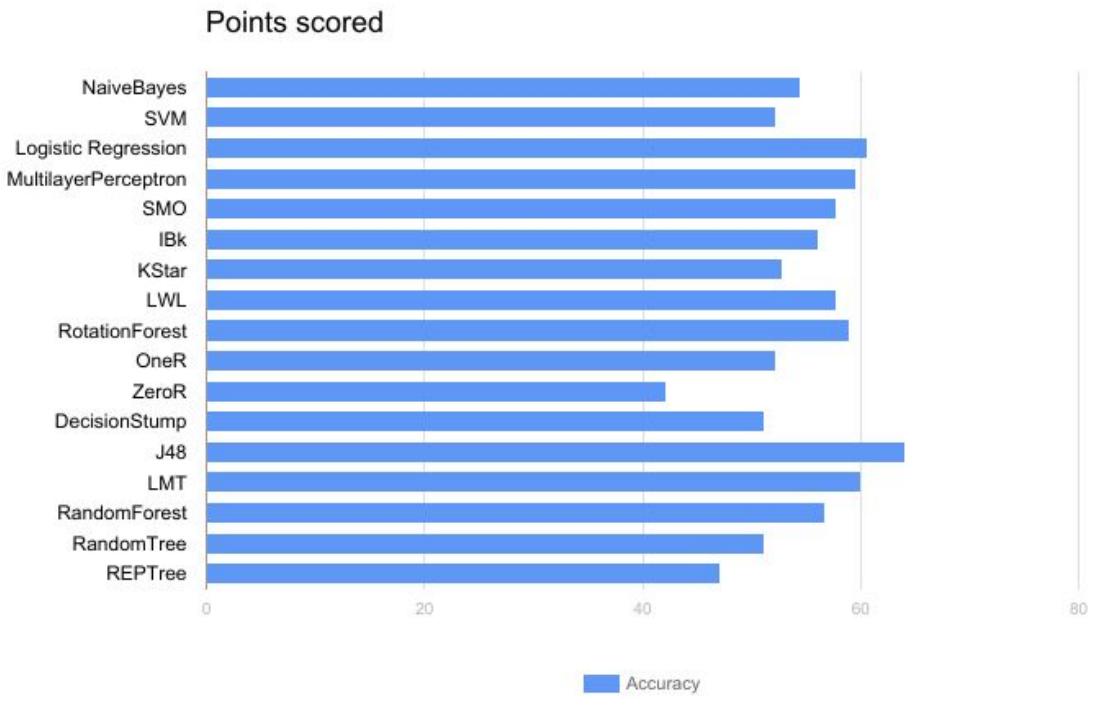

Figure 16: Frailty prediction results per prediction model

6.4 Final model parameters

The final classifier that was selected to be used in the LingTester tool is the very well known Decision Tree algorithm implementation C4.5, first proposed by Ross Quinlan as an extension of the ID3 algorithm. In the tool, an open source java version of this algorithm has been used called J48. This version is capable of being tuned by a series of input parameters, mainly giving control of the final structure of the produced tree (nodes, leaves, height). The available parameters have been gathered along with their description in the next table.

Parameter Name	**Description**
Binary splits	Whether to use binary splits on nominal attributes when building the trees.
Confidence factor	The confidence factor used for pruning (smaller values incur more pruning).
MinNumObj	The minimum number of instances per leaf.
Reduced Error Pruning	Whether reduced-error pruning is used instead of C.4.5 pruning.

Unpruned	Whether pruning is performed.
Use Laplace	Whether counts at leaves are smoothed based on Laplace.

Table 6: Explanation of parameters used

The exact values that were used to produce the LingTester tool predictive tree are presented on the table below. The process of model parameter optimization is a highly empirical process, although there have been some efforts in the field, for example Auto-Weka. As this is still a preliminary version, the empirical approach of trial and error was used to select the best performing parameter values. A small performance gain may still be possible by utilizing automated parameter exploration techniques and is thus left for future examination.

Parameter Name	**Parameter Value**
Binary splits	False
Confidence factor	0.25
MinNumObj	2
Reduced Error Pruning	False
Unpruned	True
Use Laplace	False

Table 7: Actual parameters used for the prediction model

6.5 Final model results

6.5.1 Three class classification results

The final Decision Tree model that is embedded to the LingTester tool is capable of predicting the three possible frailty conditions (nonfrail, prefrail, frail) by an average prediction accuracy of 64.04% for out of sample instances. A statistical analysis of the model has been conducted and the most important metrics are presented below.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         114               64.0449 %
Incorrectly Classified Instances        64               35.9551 %
Kappa statistic                          0.4465
Mean absolute error                      0.3041
Root mean squared error                  0.4257
Relative absolute error                 69.881  %
Root relative squared error             91.2654 %
Total Number of Instances              178
Ignored Class Unknown Instances                 2

=== Detailed Accuracy By Class ===
```

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.689     0.083      0.738     0.689      0.713      0.779    nonfrail
                 0.64      0.291      0.615     0.64       0.627      0.719    prefrail
                 0.603     0.192      0.603     0.603      0.603      0.735    frail
Weighted Avg.    0.64      0.206      0.643     0.64       0.641      0.739

=== Confusion Matrix ===

  a  b  c   <-- classified as
 31 10  4 |  a = nonfrail
  8 48 19 |  b = prefrail
  3 20 35 |  c = frail
```

Figure 17: Decision tree model results

At the end of the results above, the confusion matrix for the corresponding model is also shown. From the confusion matrix one can extract useful evaluation metrics like false positives, negatives and generally calculate indicators known as precision & recall. Further study of these metrics is left for future study on the final version of the report.

The final extracted tree structure of the trained model is also presented in the next figure.

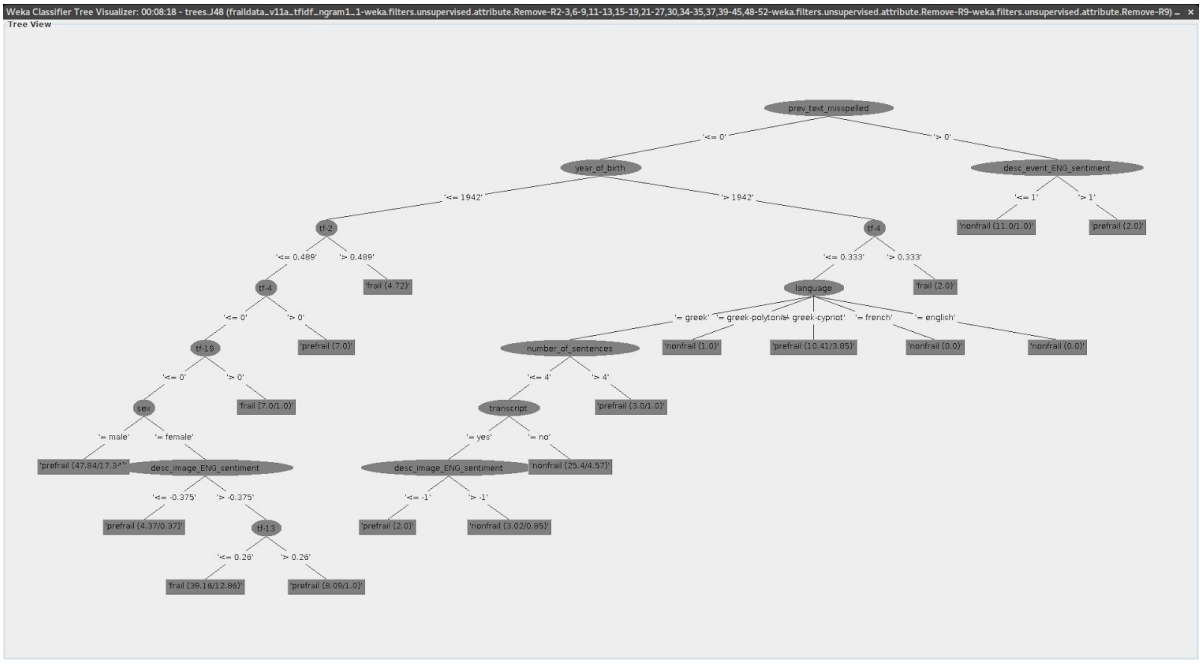

Figure 18: Decision tree visualization

6.5.2 A simplification: Binary classification approach

As the main model involves the three frailty classes, was expected to have a higher decision complexity and proportionally lower accuracy. Apart from further future dataset population and attempt was made to decrease the decision complexity of the frailty prediction problem. A common technique employed is the reduction of the number of classes by combining or removing similar classes. In this section and clearly for testing and evaluation purposes, a

similar strategy was employed for the dataset. In more detail, the Prefrail class was combined with that of Frail class, thus the instances that were labeled as prefrail were renamed as frail and the training and evaluation process was again deployed. The results were stunning, the prediction accuracy was significantly increased to 84.83%. The difference was almost 20% more than the three class classification problem. The results were expected as the decision complexity regarding the frailty status of a participant in evaluation was significantly lower. Below are shown the exact statistical results and the corresponding confusion matrix.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         151               84.8315 %
Incorrectly Classified Instances        27               15.1685 %
Kappa statistic                          0.57
Mean absolute error                      0.2242
Root mean squared error                  0.3701
Relative absolute error                 59.0842 %
Root relative squared error             85.1245 %
Total Number of Instances              178
Ignored Class Unknown Instances                 2

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.6       0.068      0.75      0.6       0.667      0.758    nonfrail
                 0.932     0.4        0.873     0.932     0.902      0.745    frail
Weighted Avg.    0.848     0.316      0.842     0.848     0.842      0.749

=== Confusion Matrix ===

   a   b   <-- classified as
  27  18 |   a = nonfrail
   9 124 |   b = frail
```

Figure 19: Results of the final prediction model

It is worth to mention that even when the experiment was reconducted and the class reduction was now done by simply removing the instances labeled as prefrail, the prediction accuracy still remained at the same levels (~81.55%).

6.6 Discussion of the results

This chapter was devoted to the evaluation process and the extraction of the according evaluation metrics. A large experiment was organized and the seventeen of the most known models were trained and evaluated.

The final model that was selected to be embedded to the LingTester tool, was a Decision Tree implementation (C4.5) due its accuracy superiority, in comparison with the other evaluated algorithms, but also due its simplicity and its fundamental property of tree structure. The last property is considered highly important for a medical application such frailty prediction. This is because the exact decision logic can be easily visualized giving the human evaluator(doctors-researchers) to observe and judge its behavior and even extract useful feature relations between the instance feature values.

The accuracy for the selected classifier was achieved after the parameter optimization process and was around 64% for three class classification problem. Further increase of the prediction performance is possible to be achieved either by the future population of participants data or by further investigation of the model parameters and the deployment of ensemble models.

Binary classification is also an approach that was evaluated and seems to perform relatively good mainly to the reduction of decision complexity. Further exploration of the approach may reveal useful relations between the dataset features and parameters.

6.7 Next steps

On weeks that follow, it has been decided to keep working on the following issues, with in mind to improve accuracy of the prediction model, and also make it more rigid.

Language per text, currently, is not verified but assumed to be greek, greek-cypriot or french through the extraction process. For example, it is taken for granted that patients from UoP partner speak and write in Greek while patients from INSERM (France) speak and write in French. We focus on an automatic detection system, which can successfully identify given text among a predefined set of languages. This step, while may impose significant errors, will make the model prediction model more rigid for cases where we are unable to identify given text, or in case language should not be taken for granted, for example a greek patient may say something in English while posting on Facebook.

Another step to improve the accuracy of the prediction model, will be firstly to improve the methodology of feature extraction, which among other steps, contains the calculation of the readability score. Throughout the testing phase, we identified a series of steps that can improve readability scores. In specific, we can improve the syllabification rules of the Greek language implemented within the NLTK module, that is widely used for our feature extraction step.

Furthermore, results were not satisfactory using POS information, and thus were removed as accuracy dropped when including these features. However, it should be stressed that extra steps will be carried out to identify a way to circumvent these problems.

7. ETHICS AND SAFETY

Throughout the construction of the offline Lingtester tool, legal issues were kept in mind so as to protect sensitive information.

First of all, as described before data anonymization is used, which confirms that the participants of the clinical trials can not be linked to their own data. The database where the collected data is stored, is always kept of offline, this fact ensures the impossibility of unauthorised access of the data. Furthermore, the LingTester tool in order to make its predictions uses an exported and pre-computed(at training phase) model thus securing the independence of participants' training data and the tool which can be deployed on less secure environments.

Moreover, the data persistence and analysis will comply with the data protection guidelines reported in deliverable "D9.9: Ethics, Safety and Health Barriers" (Section 6) with the aim of, at same time, keeping the maximum level of security and privacy of the data and allowing the successful performance of the other tasks of the project. Moreover, data will be obtained in accordance to the local ethics requirements. Any information regarding the participants will be treated as sensitive personal data (as defined in deliverable D9.9) and kept strictly private. Future provided data will be thoroughly checked by semi-automatic algorithms in order to anonymize any personal identifiers like full names, dates, emails, communication cellphone or landline numbers – hence falling outside the scope of legislation concerning personal data.

Finally, the participants are already, fully informed and agree to share their clinical data with the FrailSafe program. In any time during the program they retain the right to quit the data collection process according to their will.

8. REFERENCES

[1] M. Pacharne and V. Nayak. "Feature Selection Using Various Hybrid Algorithms for Speech Recognition," in Computational Intelligence and Information Technology, 1st ed., V. Das and N. Thankachan, Ed. Berlin: Springer Berlin Heidelberg, 2011, pp. 652-656.

[2] Haney EM, Carson S, Low A, et al.: Suicide Risk Factors and Risk Assessment Tools: A Systematic Review. U.S. Department of Veterans Affairs, Washington, D.C., 2012

[3] C. Jin, T. Ma and R. Hou, Chi-square statistics feature selection based on term frequency and distribution for text categorization, IETE J. Res. 61 (4) (2015) 351–362.

[4] B. Desmet, V. Hoste, Emotion detection in suicide notes, Expert Systems with Applications 40 (2013) 6351–6358

[5] Pestian J, Nasrallah H, Matykiewicz P, Bennett A, Leenaars A. Suicide Note Classification Using Natural Language Processing: A Content Analysis. Biomedical Informatics Insights. 2010: 19-28. PMID 21643548

[6] A. Abboute, Y. Boudjeriou, G. Entringer, J. Azé, S. Bringay, P. Poncelet Mining twitter for suicide prevention Proceedings of the Natural Language Processing and Information Systems, Springer (2014), pp. 250–253

[7] Bandeen-Roche K, Xue QL, Ferrucci L, et al. Phenotype of frailty: Characterization in the women's health and aging studies. Journals of Gerontology Series A-Biological Sciences and Medical Sciences. 2006;61(3):262–266.

[8] Ensrud KE, Ewing SK, Cawthon PM, et al. A comparison of frailty indexes for the prediction of falls, disability, fractures, and mortality in older men. Journal of the American Geriatrics Society. 2009 Mar;57(3):492–498.

[9] Fried LP, Tangen CM, Walston J, et al. Frailty in older adults: evidence for a phenotype. JGerontolA BiolSciMedSci. 2001;56(3):M146–M156.

[10] Michalski, Ryszard S., Jaime G. Carbonell, and Tom M. Mitchell, eds. Machine learning: An artificial intelligence approach. Springer Science & Business Media, 2013.

[11] "The Turing Test, 1950". turing.org.uk. The Alan Turing Internet Scrapbook.

[12] N., Aizenberg, Naum; Joos., Vandewalle, (2000). Multi-Valued and Universal Binary Neurons : Theory, Learning and Applications. Springer US. ISBN 9781475731156. OCLC 851783812

[13] Hall, Mark, et al. "The WEKA data mining software: an update." ACM SIGKDD explorations newsletter 11.1 (2009): 10-18.

[14] Bermingham, Mairead L.; Pong-Wong, Ricardo; Spiliopoulou, Athina; Hayward, Caroline;

[15] Pearl, J. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.

[16] Khachaturyan, A.; Semenovskaya, S.; Vainshtein, B. (1979). "Statistical-Thermodynamic Approach to Determination of Structure Amplitude Phases". Sov.Phys. Crystallography.

[17] Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press.

[18] R.C. Holte (1993). Very simple classification rules perform well on most commonly used datasets. Machine Learning, Vol. 11, pp. 63-91.

[19] Geisser, Seymour (1993). Predictive Inference. New York, NY: Chapman and Hall.

[20] Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall. ISBN 978-0137903955.

[21] Cortes, C.; Vapnik, V. (1995). "Support-vector networks". Machine Learning. 20 (3): 273–297. doi:10.1007/BF00994018.

[22] Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

[23] Sutton, Oliver. "Introduction to k nearest neighbour classification and condensed nearest neighbour data reduction." University lectures, University of Leicester (2012).

[24] John G. Cleary, Leonard E. Trigg: K*: An Instance-based Learner Using an Entropic Distance Measure. In: 12th International Conference on Machine Learning, 108-114, 1995.

[25] Juan J. Rodriguez, Ludmila I. Kuncheva, Carlos J. Alonso (2006). Rotation Forest: A new classifier ensemble method. IEEE Transactions on Pattern Analysis and Machine Intelligence. 28(10):1619-1630.

[26] Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." Ijcai. Vol. 14. No. 2. 1995.

9. FILE STRUCTURE

- Folder: Deliverable
    - File: fraildata_v11a_tfidf_ngram1_1.arff
        - Extracted dataset
    - File: fraildata_feature_selected.arff
        - Feature selected dataset
    - File: fraildata_feature_selected_binary.arff
        - Binary feature selected dataset
    - File: fraildata_feature_selected_binary_removed_instances.arff
        - Binary via instance removement feature selected dataset
    - File: suicide_v1c_withtfidf_30_ngram1.arff
        - Suicide constructed dataset
    - File: final_model.model
        - Built frailty predictive model
    - File: suicide_model.model
        - Built suicide predictive model
    - File: offline-parser.py
        - Python script for feature extraction - creation & dataset creation
    - File: goodreads-downloader.py
        - Python crawler script for the suicidal dataset

10. ANNEXES

10.1 Goodreads crawler

File: goodreads-downloader.py

```
# -*- coding: utf-8 -*-
"""
@author: Charalampos
"""

import math, numpy, pickle, nltk
import matplotlib.pyplot as plt

import urllib2
import pickle
import re
op = __import__("offline-parser")
from textstat.textstat import textstat

main_page                                   =
'http://www.goodreads.com/quotes'
history_file                                =
'goodreads-downloader.history.pickle'
quotes_file                                 =
'goodreads-downloader.quotes.csv'

default_suicide_groups    =    ['suicide',
'sadness',                  'suicide-note',
'suicide-attempt', 'suicidal-thoughts']
default_non_suicide_groups   =   ['humor',
'happy']

def  sortedDictValues(adict,  reverse_order
= True):
    """
      Taksinomisi leksikou, simfwna me to
"value"

http://wiki.python.org/moin/HowTo/Sorting/
    """
    ret = []
    for k in adict:
        ret.append( (k, adict[k]) )

      ret  =  sorted(ret,  key=lambda  tdf:
tdf[1], reverse = reverse_order)

    return [page[0] for page in ret]

def historySave(history_to_save):
    f = open(history_file, 'w')
    pickle.dump(history_to_save, f)
    f.close()

def historyLoad():
    history_to_load = {}
    try:
        f = open(history_file, 'r')
        history_to_load = pickle.load(f)
        f.close()
    except:
        pass
    return history_to_load

def quotesSave(quotes_to_save):
    f = open(quotes_file, 'w')
    for k in quotes_to_save:
                    f.write("%s\t%s\n"   %
('|'.join(quotes_to_save[k]['t']), k))
    f.close()

def quotesLoad():
    quotes_to_load = {}
    try:
        f = open(quotes_file, 'r')
        lines = f.readlines()
        f.close()
        for k in lines:
            k = k.strip()
            qtags, qtext = k.split("\t")
            qtags = qtags.strip()
            qtext = qtext.strip()
            if qtags == qtext:
                continue

             quotes_to_load[qtext] = {'q':
qtext, 't': qtags.split('|')}
    except:
        pass

    return quotes_to_load

def quotesByTag():
    print 'Loading..',
    quotes = quotesLoad()
    print 'Done!'

    print 'Transforming..',
    quotes_to_return = {}
    res = len(quotes) / 10
    for q in quotes:
        res -= 1
        if res < 0:
            res = len(quotes) / 10
            print '.',

        quote = quotes[q]
        for t in quote['t']:
            if not t in quotes_to_return:
                quotes_to_return[t] = {}


quotes_to_return[t][quote['q']] = quote
    print 'Done!'

    return quotes_to_return

def getQuotes(tag, page, history):
    url_to_fetch = main_page
    if tag != '':
```

```
        url_to_fetch += '/tag/' + tag

    if page > 1:
            url_to_fetch += '?page=' + str(page)

    if tag == '':
        print "Fetching all quotes, page %d" % page,
    else:
        print "Fetching quotes for %s, page %d" % (tag, page),

    try:
                                request = urllib2.Request(url_to_fetch)
                                contents = urllib2.urlopen(request).read()
    except:
        print '..Error!'
        return None

    contents = contents.replace('&ldquo;', '')
    contents = contents.replace('&rdquo;', '')
    contents = contents.replace('&#8213;', '')
    contents = contents.replace("\n", '')

    new_tags = 0
                        for m in re.finditer('\/quotes\/tag\/([a-z\-]+)', contents):
        tag = m.group(1)
        if not tag in history:
            history[tag] = 0
            new_tags += 1

    if new_tags > 0:
        print '.. %d new tags' % new_tags,
        historySave(history)

        contents = contents.replace('<div class="quote mediumText ">', "<div class='quote'>")
                        #     contents = contents.replace('quoteText">', "quoteText'>")

        text_quotes = contents.split("<div class='quote'>")
    # Useless
    del text_quotes[0]

    quotes_to_return = {}
    for q in text_quotes:
        m = re.search('quoteText">(.*?)<', q)
                                qtext = m.group(1).strip().replace("\t", '')
        if qtext == '':
            continue

        ctags = []
                            for m in re.finditer('\/quotes\/tag\/([a-z\-]+)', q):
            if m.group(1).strip() == '' or m.group(1).strip() == qtext:
                continue

            ctags.append(m.group(1))

        if len(ctags) <= 0:
            continue

        quotes_to_return[qtext] = {'q': qtext, 't': ctags}

            print '%d quotes' % len(quotes_to_return),
    print '..Done!'

    return quotes_to_return
    # print quotes_to_return
    # print contents

def getTagsByGroup(includeGroups, excludeGroups = [], quotes_by_tag = None, no_more_than = 0):
    if quotes_by_tag is None:
        quotes_by_tag = quotesByTag()

    excluded = {}
    for g in excludeGroups:
        for q in quotes_by_tag[g]:
            quote = quotes_by_tag[g][q]
            excluded[ quote['q'] ] = True

    selected = {}
    for g in includeGroups:
        for q in quotes_by_tag[g]:
            quote = quotes_by_tag[g][q]
            if quote['q'] in excluded:
                continue

            selected[ quote['q'] ] = quote
                if no_more_than > 0 and len(selected) >= no_more_than:
                break

    return selected

def getSeperateGroups(suicideGroups, nonSuicideGroups, quotes_by_tag = None):
    if quotes_by_tag is None:
        quotes_by_tag = quotesByTag()

    group1 = getTagsByGroup(suicideGroups, nonSuicideGroups, quotes_by_tag)
                            group2 = getTagsByGroup(nonSuicideGroups, suicideGroups, quotes_by_tag, len(group1))

        print "suicide group: %d" % len(group1)
        print "NON suicide group: %d" % len(group2)
    return group1, group2

def startWorking():
    quotes = quotesLoad()
    history = historyLoad()
```

```
    if not '' in history:
        history[''] = 0
        historySave(history)

     # for tag in ['suicide', 'sadness',
'suffering',               'suicide-note',
'failed-attempt',       'suicide-attempt',
'suicidal-thoughts']:
    #     if not tag in history:
    #         history[tag] = 0

      print "%d tags in my history" %
len(history)
        print "%d quotes already" %
len(quotes)

    added = True
    while added:
        added = False
        for tag in history.keys():
            if history[tag] >= 100:
                continue

              print 'Tag "%s", page %d' %
(tag, history[tag])
            lenbefore = len(quotes)
            while history[tag] < 100:
                history[tag] += 1
                                new_quotes =
getQuotes(tag, history[tag], history)
                if len(new_quotes) <= 0:
                    history[tag] = 100

                quotes.update(new_quotes)

            lenafter = len(quotes)

            if lenafter > lenbefore:
                added = True
                     print 'Saving %d new
quotes' % (lenafter - lenbefore),
                quotesSave(quotes)
                print '..Done'
            historySave(history)

    print "Update ended"

def create_arff(relation = 'suicide',
includeTFIDF = True, group_suicide = None,
group_non_suicide = None, TFIDF_thres =
20):
       """Create arff for WEKA with all
features availiable
    """
    out = []
    out.append('@RELATION %s' % relation)
    out.append('')

    other_attributes = []

other_attributes.append('get_feature_lengt
h')

other_attributes.append('get_feature_numbe
r_of_sentences')

other_attributes.append('get_feature_word_
count')

other_attributes.append('get_feature_words
_per_sentence')

other_attributes.append('get_feature_text_
shannon_entropy')

    for attr in other_attributes:
        call = getattr(op, attr)
          out.append('@ATTRIBUTE %s %s' %
(call('', 'title'), call('', 'type')))

    read_scores = []

read_scores.append('flesch_reading_ease')
    read_scores.append('smog_index')

read_scores.append('flesch_kincaid_grade')

read_scores.append('coleman_liau_index')

read_scores.append('automated_readability_
index')

read_scores.append('dale_chall_readability
_score')
    read_scores.append('difficult_words')

read_scores.append('linsear_write_formula'
)
    read_scores.append('gunning_fog')
    for attr in read_scores:
          out.append('@ATTRIBUTE %s %s' %
(attr, 'real'))

        out.append('@ATTRIBUTE %s %s' %
('sentiment', 'real'))
        out.append('@ATTRIBUTE %s %s' %
('misspelled', 'real'))

    if group_suicide is None:
                          group_suicide =
default_suicide_groups
    if group_non_suicide is None:
                      group_non_suicide =
default_non_suicide_groups
                            results =
getSeperateGroups(group_suicide,
group_non_suicide)
    group_suicide_quotes = results[0]
    group_non_suicide_quotes = results[1]

    filename = 'ARFFS/%s.arff' % relation


    texts = []
    if includeTFIDF:
                       s, score_words =
compute_ig({'suicide':
group_suicide_quotes,       'non-suicide':
group_non_suicide_quotes})

        mywords = s[:TFIDF_thres]

        for q in group_suicide_quotes:
                              temptext =
```

```
group_suicide_quotes[q]['q']
            temptext_out = []
              for word in temptext.split('
'):
                word = word.lower()
                    word = re.sub(r'([^a-z
])', '', word)
                 if word != '' and word in
mywords:

temptext_out.append(word)

                              texts.append('
'.join(temptext_out))

        for q in group_non_suicide_quotes:
                                  temptext   =
group_non_suicide_quotes[q]['q']
            temptext_out = []
              for word in temptext.split('
'):
                word = word.lower()
                    word = re.sub(r'([^a-z
])', '', word)
                 if word != '' and word in
mywords:

temptext_out.append(word)

                              texts.append('
'.join(temptext_out))

        # TF-IDF on stemmed text
                                     tf     =
op.TfidfVectorizer(analyzer='word',
ngram_range=(1,1), min_df = 0)
                           tfidf_matrix    =
tf.fit_transform(texts)
                         feature_names    =
tf.get_feature_names()
            print "TF-IDF %d features" %
len(feature_names)

        fp = open(filename + '.tf.pickle',
'w')
                  pickle.dump({'tf':  tf,
'tfidf_matrix':              tfidf_matrix,
'feature_names':  feature_names,  's':  s,
'score_words':   score_words,   'mywords':
mywords}, fp)
        fp.close()

                                for    i    in
range(len(feature_names)):
              out.append('@ATTRIBUTE tf-%d
real %% %s' % (i, feature_names[i]))

         # The following creates an "array
to big" error
        # dense = tfidf_matrix.todense()

    # Class always must go last
       out.append('@ATTRIBUTE  %s  {%s}'  %
('class',             ','.join(['suicide',
'non-suicide'])))

    out.append('')
    out.append('@DATA')
    out.append('')

    print 'Creating ARFF rows',
    f = open(filename, 'w')
    f.write("\n".join(out).encode('utf8'))
    f.write("\n")

    all_quotes = len(group_suicide_quotes)
+ len(group_non_suicide)
    rows_so_far = all_quotes / 10
    i = 0
    for tag in ['suicide', 'non-suicide']:
        if tag == 'suicide':
                                  my_quotes   =
group_suicide_quotes
        else:
                                  my_quotes   =
group_non_suicide_quotes

        for q in my_quotes:
            rows_so_far -= 1
            if rows_so_far <= 0:
                print '.',
                rows_so_far = all_quotes /
10

            clang = 'english'
            qtext = my_quotes[q]['q']
            row = []

            for attr in other_attributes:
                call = getattr(op, attr)
                row.append(str(call(qtext,
lang = clang)))

            for attr in read_scores:
                  call = getattr(textstat,
attr)

row.append(str(call(qtext)))

             # Sentiment score is based in
the english translation

row.append(str(op.get_feature_sentiment_sc
ore(qtext, lang = clang)))

            # Mispelling score

row.append(str(op.get_feature_mispelling_s
core(qtext, lang = clang)))

            if includeTFIDF:
                 # tf-idf based on stemmed
data
                # p = dense[i].tolist()[0]
                                        p   =
tfidf_matrix[i,:].toarray()[0]
                                   for   fi   in
range(len(feature_names)):
                          row.append('%.3f' %
p[fi])

            row.append(tag)
```

```
f.write(','.join(row).encode('utf8'))
            f.write("\n")

            i += 1

    f.close()
    print '..Done'

def compute_ig(quotes_per_tag):
    """
    compute_ig():
        Compute information gain for each
word
    """
    # With a little bit of help
                                        #
http://streamhacker.com/tag/information-ga
in/

              from    nltk.metrics    import
BigramAssocMeasures

    word_count_per_class = {}
    all_word_count_per_class = {}
    word_count_per_word = {}
    all_words = 0

    print "Loading files for ig..",
    for tclass in quotes_per_tag:
        word_count_per_class[tclass] = {}
        all_word_count_per_class[tclass] =
0
         i = len(quotes_per_tag[tclass]) /
10
                            for    quote    in
quotes_per_tag[tclass]:
            i -= 1
            if i <= 0:
                print '.',
                                        i   =
len(quotes_per_tag[tclass]) / 10

                                    data    =
quotes_per_tag[tclass][quote]['q'].split('
')

            for w in data:
                w = w.lower()
                   w = re.sub(r'([^a-z])',
'', w)

                if w == '':
                    continue


word_count_per_class[tclass][w]           =
word_count_per_class[tclass].get(w, 0) + 1
all_word_count_per_class[tclass] += 1
                    word_count_per_word[w] =
word_count_per_word.get(w, 0) + 1
                all_words += 1

            del data

    print "Evaluating..",
    i = int(len(word_count_per_word) / 10)
    score_per_word = {}
    for w in word_count_per_word:
        i -= 1
        if i <= 0:
            print ',',
                                        i    =
int(len(word_count_per_word) / 10)

        freq = word_count_per_word[w]
        score_per_word[w] = 0

        for c in word_count_per_class:
                        score_per_word[w]  +=
BigramAssocMeasures.chi_sq(word_count_per_
class[c].get(w,          0),          (freq,
all_word_count_per_class[c]), all_words)

              del    word_count_per_class,
all_word_count_per_class,
word_count_per_word

    print "Sorting..",
    s = sortedDictValues(score_per_word)
    print "..Done"
    # del score_per_word

    print "..Done"

    nums = []
    for w in s:
        nums.append(score_per_word[w])

    print "Creating ig histogram",
     plt.figure(figsize = (24, int(24.0 *
9.0 / 16.0)),)
                                        #
plt.hist(numpy.asarray(score_per_word.valu
es()), 5000, facecolor = 'g')
    plt.plot(nums)
    plt.xlabel('Lexicon values')
    plt.ylabel('IG Score')
        plt.title('IG  Score  per  lexicon
lemma')
    plt.grid(True)
    plt.show()
    print "..Done"
    # del s

    return s, score_per_word
```