| | |
|---|---|
| **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: D4.10

## LingTester (Prototype) (vers a)

| | |
|---|---|
| **Due date of deliverable:** | M12 (31th December 2016) |
| **Actual submission date:** | 31th December 2016 |
| **Version:** | 1.3 |
| **Date:** | 22nd December 2016 |


**Lead Author:** Sgarbas Kyriakos (UoP)

**Lead partners:** UoP



Horizon 2020
European Union funding
for Research & Innovation

**CHANGE HISTORY**

| Ver. | Date | Status | Author (Beneficiary) | Description |
|---|---|---|---|---|
| 1.0 | 15/11/16 | Draft | N. Fazakis,<br>C. Tsimpouris,<br>K. Sgarbas (UoP) | First Draft |
| 1.1 | 22/11/16 | Draft | N. Fazakis,<br>C. Tsimpouris | Update of introduction, list of tables,<br>minor text corrections,<br>addition of references |
| 1.2 | 21/12/16 | Pre-Final | N. Fazakis,<br>C. Tsimpouris,<br>K. Sgarbas (UoP) | Accompanying files included |
| 1.3 | 22/12/16 | Final | N. Fazakis,<br>C. Tsimpouris,<br>K. Sgarbas (UoP) | Final version |

## EXECUTIVE SUMMARY

LingTester is the FrailSafe language analysis tool that aims to process the user's typed text and detect abnormal behaviour. At this point, the prototype is in early alpha stage, but still it is able to perform classification according to levels of frailty. The present deliverable describes the development of the prototype, the algorithms used, the training process and some preliminary test results.

This deliverable is part of WP4. 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, cognitive and physical 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.

LingTester will be able to detect signs of mental frailty and personality trait shifts by linguistic processing of a person's written (typed) messages. The linguistic analysis is performed in several layers (ranging from word spelling to Part of Speech -POS- analysis) utilizing the state of the art models in order to determine the mental states of the patients the input texts exhibit. The linguistic corpus obtained from D4.7 is used both for the initial training and the final passive mode (off-line) testing of the prototype.

Along with the results of the patient data analysis, two aiding software programs are delivered in order for the FrailSafe user to manage the patient database and use prediction model to obtain predictions for specific potential patients.

## 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:** | 4.10 | **Title:** | LingTester (Prototype) (vers a) |
| **Work package number:** | 4 | **Title:** | Data Management and Analytics |

| | | | | |
|---|---|---|---|---|
| **Date of delivery** | **Contractual** | **31/12/2016 (M12)** | **Actual** | 31/12/2016 |
| **Status** | Draft ☒ | | Final ☐ | |
| **Nature** | Report ☐ Demonstrator ☐ Other ☒ | | | |
| **Dissemination Level** | Public ☒ Consortium ☐ | | | |
| **Abstract** (for dissemination) | This is a public deliverable that summarizes the progress of the construction of LingTester offline prototype. The architecture of the system is described in detail, and what steps were needed to evaluate its output. Also, the structure of the internal offline database is described and how it is managed. The deliverable includes a demo and source files. | | | |
| **Keywords** | offline data management, frailty prediction, classification | | | |


| | | | | |
|---|---|---|---|---|
| **Contributing authors** (beneficiaries) | Fazakis Nikos (UoP)<br>Tsimpouris Charalampos(UoP)<br>Sgarbas Kyriakos (UoP) | | | |
| **Responsible author(s)** | Sgarbas Kyriakos | | **Email** | sgarbas@upatras.gr |
| | **Beneficiary** | UoP | **Phone** | +30 2610 996 470 |

# Table of Contents

# List of Figures

## List of Tables



## List of annexes

# List of abbreviations and acronyms (*in alphabetical order*)

| | |
|---|---|
| API | Application Programming Interface |
| ARFF | Attribute-Relation File Format |
| JVM | Java Virtual Machine |
| KNN | K-Nearest Neighbor |
| NLP | Natural Language Programming |
| LOOCV | Leave-one-out cross-validation |
| PoS | Part Of Speech |
| SVM | Support Vector Machine |
| UoP | University of Patras |
| Weka | Waikato Environment for Knowledge Analysis |

# 1. Introduction

A proper evaluation of the nature of a patient's language impairment requires consideration of patterns of breakdown in the context of an account of language comprehension which specifies the various processes and representations involved: the representation of linguistic knowledge, it's automatic and controlled access, and the mental processes which combine different types of linguistic knowledge. When we understand a written sentence, we automatically access the meanings of individual words together with their syntactic specifications and combine them on the basis of this lexically specified information. There are various types of text analysis processes; some are syntactic, such as the integration of a definite article with the following noun or adjective to form a noun phrase, and the integration of a verb and its argument into a verb phrase. Others are morphological, such as the combination of stems and affixes to form morphologically complex words, and yet others involve combinatorial processes which modify the meanings of words when they are integrated with other words. For example, an aspect of the meaning of grass is that it is green; but in the phrase dry grass, its meaning changes slightly to highlight a different aspect of grass— its brownness.

In order to implement the first version of LingTester an architectural analysis was prepared. The current architecture of LingTester led the need to categorize the research in four main areas.

The first area of research relates to the analysis and the issues of the collected subjects' data. The key aspects and features of the dataset were investigated and solutions were given to the unexpected arosen problems. The structural organization of the local database and the development of the processes for the management of the local database was another area of research. A simple but very informative and mobile structure for offline data storage along with its necessary manipulation methods was designed and implemented. In the area of the highly critical classification task, the domains of feature extraction, feature selection text classification and model optimization were deeply studied and exhaustively experimented in order to obtain the first acceptable prediction results. The implementation of the aiding software for the FrailSafe user was another key area of research, with the deployment of technologies like Python and Java a cross-platform approach was achieved and the first semi-integrated user software package has been developed successfully.

Following the analysis of the basic areas of research this deliverable includes, chapters relating with the System Development subjects of the software used and the installation notes, the basic usage instructions of the FrailSafe software package and finally the future work that seems promising and is already being planned, for the completeness of the document.

# 2. LingTester architecture

## 2.1 Initial architecture

The initial design of the Natural Language Analysis component (a.k.a. LingTester), is shown in the following diagram:

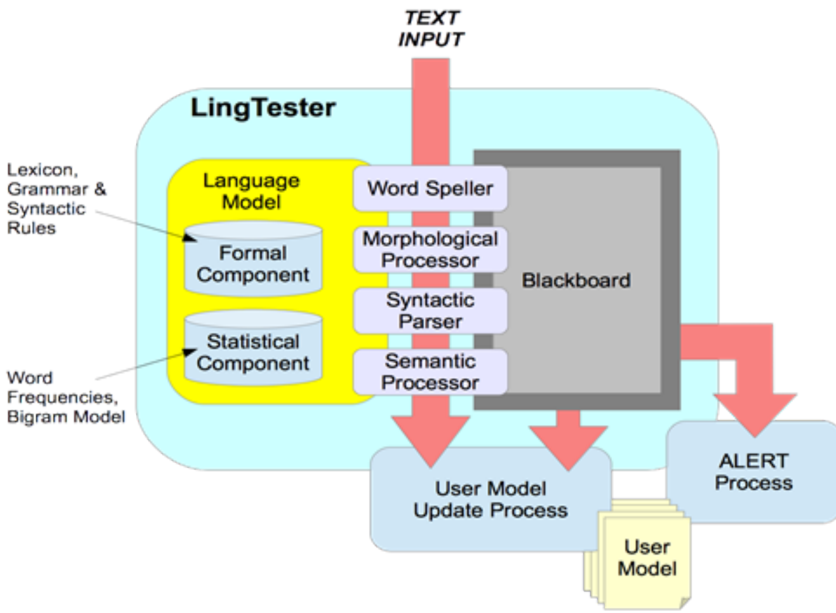


Figure 1. Initial architecture for LingTester

LingTester was initially designed to include four computational linguistic modules: a Word Speller, a Morphological Processor, a Syntactic Parser, and a Semantic Processor. A Language Model would be able to feed them with linguistic information. The Language Model would be composed by a Formal Component representing the lexicon, grammar and syntactic rules of the language, and a Statistical Component containing results of word and bigram frequencies. The whole structure was meant to be modular (in order to facilitate its use in several languages - two in the project) and would be developed over a blackboard scheme representation model that would be able to collect the output of each component and enable them to interact with each other, when necessary. The system was based on the main assumption that all texts would be written by the patients themselves and the classification module would provide rule-based results.

Thus, after a stage of adaptation/training/fine-tuning, LingTester *was expected to* detect frailty symptoms related to the use of language, and derive the patient's condition, according to the following table:

| STAGE | CONDITION | SYMPTOMS (Indicative) |
|---|---|---|
| 1 | Normal | - |
| 2 | Cognitive Decline | Misspellings, character transpositions/ eliminations. |
| 3 | Cognitive Impairment | Misuse of functional words, morphological errors. |
| 4 | Dementia | Serious syntactic and semantic errors. |

It should be stressed once more that the aforementioned design presupposes that the users will be able to type their own text messages, since key features like spelling, morphology and punctuation carry a significant portion of the information we expected to use in order to detect possible discrepancies in written text.

With this in mind, we have included some questions in the questionnaires (detailed in deliverable D2.1 Clinical Study Methodology) that serve the purpose of producing data for the LingTester (for training and test). However, during the process of questionnaire collection we realized that very few subjects were able to type. The vast majority were not, and they dictated their answers to the person responsible for carrying out the interview who then typed the answers as best as they could.

This changed the initial architecture significantly. We had to use methods of mental frailty detection that would be robust under interpretation via a third person. Specifically, it became evident to treat the process of text writing as a Hidden-Markov Process, where the person's mental state evolves over time, but we do not have direct data on this evolution, but only indirect evidence (the written text) that can reveal the (hidden) mental state. For this reason, statistical and information-based text analysis methods were prefered instead of the rule-based of the initial design. This was already documented in D2.1. The new (current) architecture is described next.

## 2.2 Current architecture

After thorough study of current practices and development, we re-evaluated the initial architecture according to the following figure. The structure has been kept mostly intact, however with some specific changes. Firstly, the syntactic parser has been replaced by a Part Of Speech tagger (see section 4.3 for more details), as the later can produce more accurate results for the Greek language. Furthermore, semantic processing module was replaced by a sentiment analysis module to try and provide a sentiment analysis of the patient through written text. Written texts are translated to English one, and then we can export polarity features to the classification process. This decision was based at the fact, that there is no state of the art available tools for sentiment analysis for this language. It has been shown that working with standard technology and existing sentiment analysis approaches is a viable approach to sentiment analysis within a multilingual framework (Denecke, 2008).

As shown in the following figure (Figure 2), written text is submitted to LingTester tool through a predetermined process and is stored within a secure database for further analysis. In order to create the training model, all patient rows are fetched from the offline database and features are extracted for the next step. Each feature utilises different resources and is based on different third-party or not tools. These tools are described thoroughly in section 4. This step is followed by the training module which extracts a model in a binary format for testing and evaluation. This methodology has been repeated multiple times so as to maximise accuracy while also optimising all parameters of the system. The final model is packaged in a way to be more programmer-friendly (see section 8 for more details).

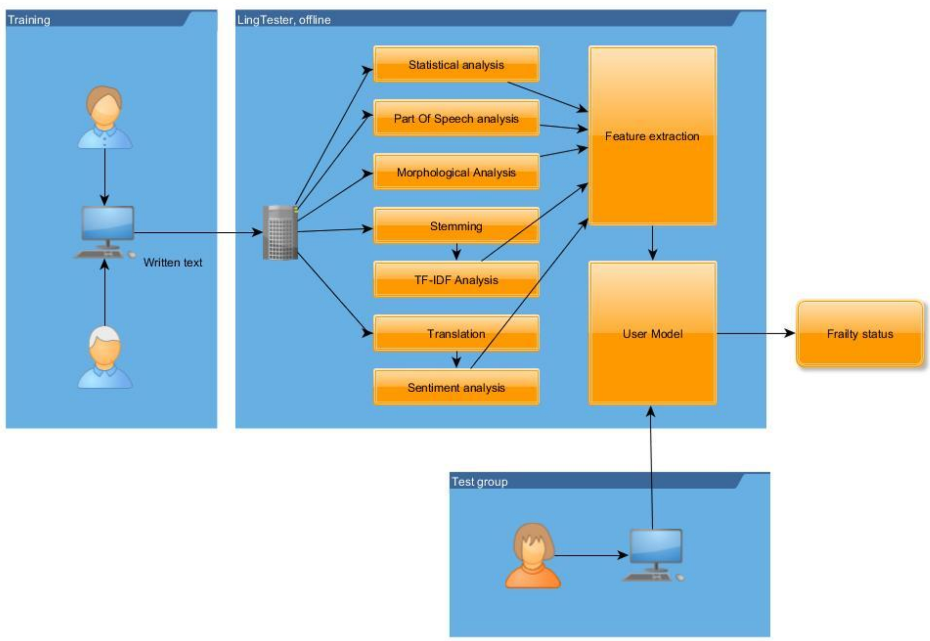


Figure 2. Current architecture for offline LingTester

Finally, it should be stressed that system should provide patient's condition in the following clusters: non-frail, pre-frail, frail.

# 3. Data collection

Although data collection at first might not seem related to the development of the prototype, it actually plays a very significant role. Even inspection of the data collected can indicate which analysis tools and methods are applicable to the task. We have already stated that the nature of the collected data forced us to reconsider the aforementioned initial design. Moreover, in the current architecture text data are needed to train and test several components of LingTester. For this first version of the prototype, the texts we used were the ones that were available until 31st October 2016. This first batch of text data is described next.

## 3.1 Data analysis

The current deliverable uses data collected until 31/10/2016. Until then we had available the following patient data:

- Data from 51 patients, UoP, Greece. One submission per patient.
- Data from 66 patients, MATERIA, Cyprus. One submission per patient.

The following patient data per recruitment centre are expected, according to the following timeline as set in detail at *D2.1 Clinical study methodology*:

- 80 patients from Start-up Group A,
- 20 patients from main Group B,
- 25 following afterwards will belong to the Evaluation Group C and
- the last 25 to the Control Group (D), totalling 150 patients.

The total data volume from all recruitment centers will include 450 patients, with multiple submissions during the duration of the project, per patient. An initial statistical analysis based to the current set of patient data returned the following results per classified feature:

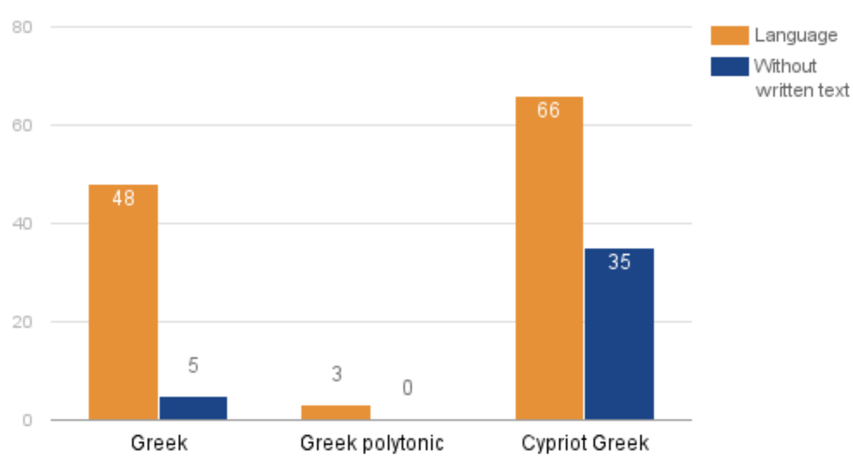


Figure 3. Patients per language

- Per language:

- Data from 48 patients were provided in Greek, but 5 of them refused or were unable to provide any written text, neither for the description of image, nor describing a personal event
- Data from 3 patients were written in Greek polytonic. While it was not in our initial scope to differentiate this information, it was recorded in the offline database for future study
- Data from 66 patients were provided in Cypriot Greek, but 35 of them refused or were unable to provide any written text, neither for the description of image, nor describing a personal event

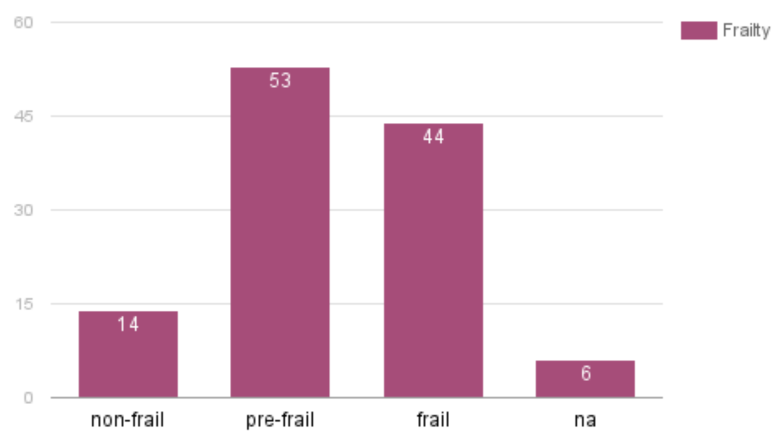


Figure 4. Patients per frailty status

- Per frailty status:
    - 14 patients were classified as non-frail
    - 53 patients were classified as pre-frail
    - and 44 patients were classified as frail
    - while for 6 patients, there was no information available for their frailty status. At this point, we should stress that all these patient data (6) were excluded from the training procedure

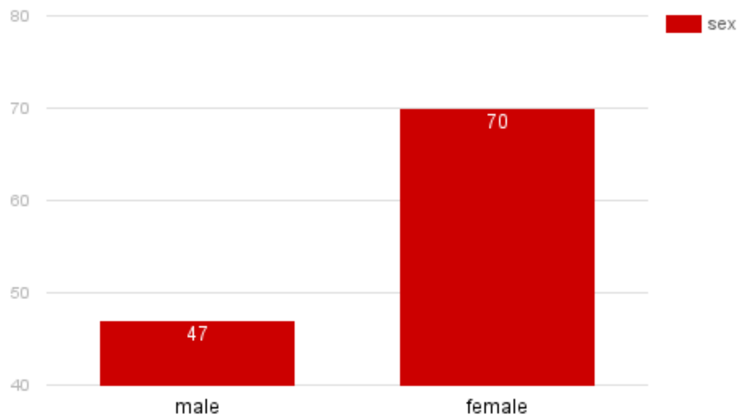


Figure 5. Distribution of patients per sex

- Per sex:
    - 47 patients were male
    - and 70 patients were female

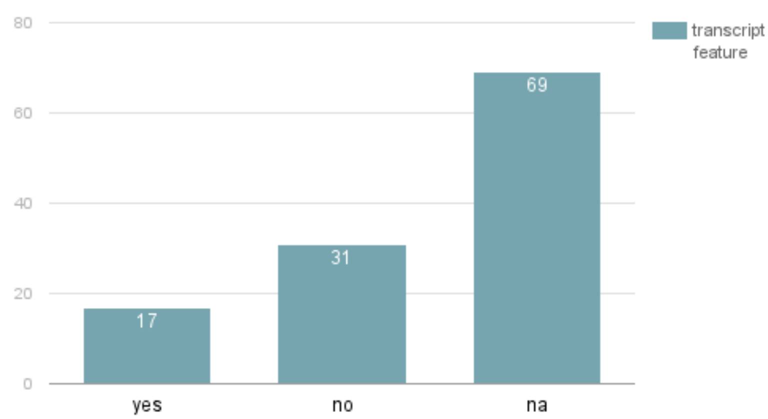


Figure 6. Patient distribution per transcript feature

- Per transcript mode:
    - 17 transcripts (column “yes”) were dictated from the patient to the doctor, and the latter typed the text

- 31 transcripts (column “no”) were handwritten by the patient, and they were afterwards digitized by the doctor
- 69 was classified as na (not available) meaning that patient refused to participate in this action (written or oral) which are 40 in total, see figure 3 column “without written text”. Furthermore, for the remaining 29 cases we were unaware of which case it was between yes/no and this is the reason we set this attribute to na, till we have a detailed report of which is the case by the providing team. We project, that this number will be lower in the revised report.

Finally, in total only two patients provided written text outside the aspect if the project. As this number is extremely small, we decided not to include this data in the following research methodology.

## 3.2 Data verification

During data collection, some discrepancies were noted concerning the slightly different patient numbering among different research groups. In the local database we took measures to verify the correct index for each patient since no version control was used during the collection of each dataset.

# 4. Local database

## 4.1 Introduction

An offline database has been created based on initial raw data, as given by the partners. While we are still under heavy development till we finalise the selection of features, we had to keep an easy-to-use, cross-platform or operating system, and loose structure database. This aforementioned decision provided numerous advantages, as discussed below:

- Each patient data is saved within a single text file, with UTF8 encoding, based on the underlying file structure system, whatever this is (NTFS/Fat32 for Windows, Ext4 for Linux, etc). This also helps to avoid any database management tools like ODBC drivers and so forth.
- File naming is based on patient id. So, it is really easy to retrieve data for a single patient and edit the file whenever necessary through user's favourite editor
- All tags (attributes) per patient are dash prefixed, so we can add or remove attributes however it suits best
- Data is retrieved and saved through generic Python functions, as described in the following section. Also, due to the use of the filesystem, we can also construct different or new methodology in a different programming language without potential connection problems with a database system.
- Create backups of all data, using a compressed data type, like zip
- Can support versioning. Using any known version control system such as GIT, we are able to keep track of the database changes at all times looking backwards

On the downside, this database structure does not provide any security firewall by itself. Security is based on the access provided by the file system, and for this reason all files are stored locally.

While it can be argued if this this structure can sustain a production-ready solution with thousands of rows, we keep in mind at all times. Upon methodology finalisation, database will also get its final form and this will be discussed again in a later report.

## 4.2 Database description

The structure of the data within the database files is described below. An example of a patient's data with id 1001 is the following:

**-patient** 1001
**-tag** prefrail
**-transcript** no
**-language** greek
**-sex** female

**-desc_event**
Υπάρχουν πολλά σημαντικά θετικά γεγονότα στη ζωή μου. Ένα από αυτά είναι η επιτυχία μου στη Φιλοσοφική Σχολή του Πανεπιστημίου Αθηνών, καθώς την ίδια περίοδο, με παράλληλες εξετάσεις και στη Νομική Αθηνών.
Πήρα μεγάλη χαρά γιατί ήταν καρπός πολύ εντατικής μελέτης- ένα ολόκληρο καλοκαίρι μόνον- με δεκαεπτά ημέρες μόνον φροντιστήριο και επί πλέον είχα μεγάλη επιθυμία να σπουδάσω, να μορφωθώ. Διάλεξα τη Φιλοσοφική και δεν έχω ούτε μια στιγμή μετανοιώσει…

**-desc_event_ENG**
There are many important positive events in my life. One of them is my success at the Philosophical Faculty of the University of Athens, as the same period, with parallel tests and Athens Law.
I took great pleasure because it was the result of very intensive meletis- an entire summer monon- with seventeen days only tutorial and moreover I had a great desire to study, get an education. I chose Philosophy and I have not a moment regret ...

**-desc_event_POS**
Υπάρχουν verb/--/active/plural/present
πολλά adjective/accusative/neuter/plural/--
σημαντικά adjective/accusative/neuter/plural/--
θετικά adjective/accusative/neuter/plural/--
.
.
.
έχω verb/--/active/singular/present
ούτε conjunction/--/--/--/--
μια numeral/--/--/--/--
στιγμή noun/accusative/feminine/singular/--
μετανοιώσει… noun/genitive/masculine/singular/--

**-desc_image**
Βρισκόμαστε μπροστά σε μια '' σουρεαλιστική'' εικόνα, σε μια σκηνή που διαδραματίζεται σε μια κουζίνα. Μια ''καλή'' νοικοκυρά ασχολείται με το πλύσιμο ή το σκούπισμα των πιάτων ενω μπροστά της η λεκάνη του νεροχύτη πλυμμηρίζει και τα νερά χύνονται στο πάτωμα.
Θαυμάζει κανείς τη μακαριότητά της, την αταραξία της μπροστά στο φαινόμενο.
Άραγε τί την απασχολεί που δεν μπορεί να αντιληφθεί ότι τα παιδιά της, στον ίδιο χώρο, πίσω από την πλάτη της…. '' κλέβουν'' το γλυκό από το επάνω ντουλάπι και το χειρότερο, ο γιός της που έχει ανέβει πάνω σε ένα ψηλό σκαμπό κοντεύει να πέσει, καθώς αυτό γέρνει στο πλάϊ ,έτοιμο να καταρρεύσει. Ω, τι κόσμος μαμά!!

**-desc_image_ENG**
We face a surreal picture, in a scene that takes place in a kitchen. A good housewife engaged in washing or wiping the dishes while in front of the sink basin plymmirizei and the water poured on the floor. One admires the blessedness, the equanimity of the front of the phenomenon.
I wonder what the concern can not perceive that children of the same place, behind the ... back. Steal sweet from the upper cabinet and the worst, the son who has climbed on a tall stool is about to fall, as it leans on the side, ready to collapse. Oh, mama world !!

```
-desc_image_POS
Βρισκόμαστε verb/--/active/plural/present
μπροστά adverb/--/--/--/--
σε preposition/--/--/--/--
μια numeral/--/--/--/--
.
.
.
τι pronoun/inflectionless/--/--/--
κόσμος noun/nominative/masculine/singular/--

-prev_text

-prev_text_ENG

-prev_text_POS
```

While, there is no styling within the plain text files, and newlines do not affect the parsing of the file structure, from the above structure we can easily retrieve all available attributes (tags) as they are all prefixed by a dash, which are:

- **`-patient`**: The patient ID. This attribute exists in all files. While the same number exists also in the filename, we put it here for consistency reasons and backwards compatibility for future updates.
- **`-transcript`**: This is identified by the following options (also described in detail in section 3.1)
    - `yes`: Text was written by the doctor, while the patient was talking
    - `no`: Text was written in hand by the patient, and it was digitized through the doctor
    - `na`: Not available, for instance for patients that refused to participate in this action
- **`-language`**: This is identified by the following options
    - `greek`: Text is in Greek
    - `greek-polytonic`: Text is in Greek Polytonic.
    - `greek-cypriot`: Text is in Greek cypriot.
    - `french`: Text is in French.
- **`-tag`**: This is identified by the following options
    - `nonfrail`: Patient is identified as non-frail
    - `prefrail`: Patient is identified as pre-frail
    - `frail`: Patient is identified as frail
    - `na`: Data is missing/not available
- **`-sex`**: This is identified by the following self-explanatory options
    - `male`
    - `female`
- **`-desc_event`**: Multiline text, the description of an event
- **`-desc_image`**: Multiline text, the description of an image

- **`-prev_text`**: Multiline text, previous text of the same patient, which is not necessarily a description of an event or an image. It can be of any context and is provided by the subject, for instance an old email, to compare extracted features between different time periods.
- **`-desc_event_POS, -desc_image_POS, -prev_text_POS:`** Part of speech information for each multiline tag, as set before
- **`-desc_event_ENG, -desc_image_ENG, -prev_text_ENG:`** English translation, based on **`-desc_event, -desc_image, -prev_text`** data.

## 4.3 Part-Of-Speech extraction

The Part of Speech tagger attempts to automatically determine the part of speech (e.g., noun, adjective, verb, etc.) of each word occurrence in Greek texts. It can also tag each word occurrence with additional information, such as the gender, number, and case of each noun, the voice, tense, and number of each verb (Koleli, 2011). The current version of AUEB's Greek POS tagger that was used is version 2 alpha and is released under the GNU General Public License.

The POS tagger can recognise the following classes of words, along with other useful information per case:

1. adjective
2. adverb
3. article
4. conjunction
5. noun
6. numeral
7. other
8. particle
9. preposition
10. pronoun
11. punctuation
12. verb

Following is an example evaluating this POS tagger in action. The sentence "*Υπάρχουν πολλά σημαντικά θετικά γεγονότα στη ζωή μου.*" (which translates to "*There are many important positive facts in my life.*") produces the following information.

| | | | | | |
|---|---|---|---|---|---|
| **Υπάρχουν**<br>*There are* | verb | -- | active | plural | present |

| | | | | | |
|---|---|---|---|---|---|
| **πολλά**<br>*many* | adjective | accusative | neuter | plural | - |
| **σημαντικά**<br>*important* | adjective | accusative | neuter | plural | - |
| **θετικά**<br>*positive* | adjective | accusative | neuter | plural | - |
| **γεγονότα**<br>*facts* | noun | accusative | neuter | plural | - |
| **στη**<br>*in* | article | prepositional | accusative | feminine | singular |
| **ζωή**<br>*life* | noun | accusative | feminine | singular | - |
| **μου**<br>*my* | noun | genitive | masculine | singular | - |
| **.** | punctuation | - | - | - | - |

Table 1. POS tagger example

In order to optimize the system, POS information is extracted once per case, and the database is automatically updated for future use. See also section 4.6 for more details.

## 4.4 English translation for sentiment analysis

To achieve better results in sentiment analysis, a significant decision was made to avoid direct sentiment analysis in Greek language or French one, but translate texts in English and then evaluate the later one. This process, also helps make the system even more language independent by utilising a unified translation system, and then shifting the sentiment analysis problem to a different level.

However, in order for this methodology to work, a third party translation service had to be used. After further investigation, we narrowed down to MyMemory service (https://mymemory.translated.net/), a **free to use** service. This translation service, uses both human and machine learning techniques for best results. MyMemory gives quick access to a large number of translations originating from professional translators, LSPs, customers and multilingual web content. It uses a powerful matching algorithm to provide the best translations available for the source text. Last but not least, we should mention that MyMemory currently contains professionally translated segments. System is constructed in a way to be modular in mind, and this is also the case for the translation submodule. In case there is any discontinuance of this service, we can easily switch to a different one, like the well known paid Google Translation API service. However, MyMerory was used for its simple API, free of charge pricing while providing translations of high quality results.

## 4.5 Database self-validation

As database management is one of the many steps to extract and create the frailty status prediction model, we had to be sure that all data stored should fulfill all the requirements for the next steps of analysis, which is data/text mining and will be performed by the free software WEKA (Eibe, Mark, Ian 20016). Having said that, it was of paramount importance for the created ARFF files (the WEKA-specific file format) to be always valid, avoid missing attributes or identify typos. So, a function was created for this purpose that reads all patient data and tries to identify discrepancies between saved data and expected classes. Finally, it also exports some basic statistical information, as were explained in detail in section 3.1

## 4.6 Database auto-update

In order to have a valid state of the database at all times, a library has been constructed in a way to always keep its internal structure useable and filled with all needed data. For this reason, specific functions have been created to check and update all patient data according to user standards. Function `update_pos_info_everywhere()` loads all patient data available from the database, and updates any missing POS data for new or updated rows. Also, function `update_corpus_with_translations()`, whenever called, will update the whole database with any missing english translations. The function has been constructed in a way to respect initial language as set in each patient data, which means that translation respects patient's language. For example Greek to English for Greek patients and French to English for patients from France and so forth.

# 5. Experiments and results

## 5.1 Preproccessing

***Feature extraction***

The classification task of the mental state of a subject requires the deployment of machine learning and pattern recognition techniques. The basic requirement for these techniques is the processing of the organized patient data with feature extraction methods before the training and prediction procedures as can be viewed in fig. 7. Feature extraction 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.

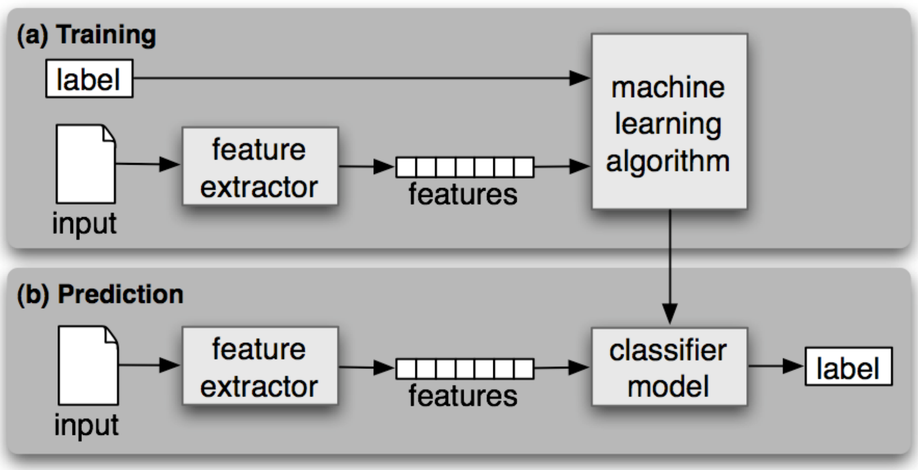


Figure 7. Training and prediction process

The implemented feature extraction algorithm for the LingTester uses several extraction methods (see table 3). The first ones involve the standardization of the basic attributes of the collected data. For the features transcript, language, class and sex, which as their name suggests describe basic information from collected data, simple rules and correction algorithms have been applied in order for the extracted data to be distinctly formalized.

Another categorization of feature extraction methods implemented uses statistical measures for the written text of the subjects. Those measures include the text length, the number of sentences, the number of words per sentence and the text entropy. The text entropy, a measure of unpredictability of information content based on characters.

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). To gain as much information as possible from this methodology, we utilised tf-idf twice. The first time is based in stemmed words, in order to avoid all suffixes. The second one, is based on POS data.This way, we could identify possible unigrams, or bigrams that are more frequent than other (for instance verb+adjective).
Written text can be broadly categorized into two types: facts and opinions. Opinions carry people's sentiments, appraisals and feelings toward the world. The module(open source) that is used for sentiment analysis (`sentiment` within `pattern.en`) bundles a lexicon of adjectives (e.g., good, bad, amazing, irritating, ...) that occur frequently in product reviews, annotated with scores for sentiment polarity (positive ↔ negative) and subjectivity (objective ↔ subjective). Using the `sentiment()` function we gain polarity and subjectivity for the given sentence, based on the adjectives it contains, where polarity is a value between -1.0 and +1.0 and subjectivity between 0.0 and 1.0.

A last preprocess step was to try and identify misspellings. In order to base our work on open source or community based tools, we used the python `pyenchant` library combined with the OpenOffice speller dictionary. This speller, is widely used by thousands of users through OpenOffice applications like OpenOffice Writer, for multiple operating systems like Linux or Microsoft Windows and is easily accessible through a Python API. For our case, we extracted the number of misspelling words against all words per case. The following table summarizes all exported features.

| Feature Names | Extraction Method |
|---|---|
| Transcript, Language, Class, Sex | Rules & Correction filters |
| Text_length, Number_of_sentences, Number_of_words, Number_of_words_per_sentence, Text_entropy | Statistical Measures |
| Desc_image_ENG_sentiment, Desc_event_sentiment, Prev_text_ENG_sentiment | Sentiment Analysis |
| Tf-XX | Term frequency – Inverse document frequency |
| Tf-pos-XX | Part of Speech analysis, using tf-idf methodology |

Table 2. Analysis of all extracted features

## 5.2 Precision & Recall

***Feature selection***

The next step before proceeding to classification task is the feature selection task. Feature selection, 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. Feature selection techniques are used for three reasons:

- simplification of models to make them easier to interpret by researchers/users,(Gareth, 2013)
- shorter training times,
- enhanced generalization by reducing overfitting(formally, reduction of variance)

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(Bermingham, 2015). 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 (Pearl, 1984), simulated annealing (Khachaturyan, 1979) and the genetic algorithm (Mitchell, 1996). 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 approach to feature selection, a simple 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(Geisser, 1993). After a sufficient number of iterations, the resulting decision tree was visualized and examined by hand in order to further optimize the resulting model.

**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)

---

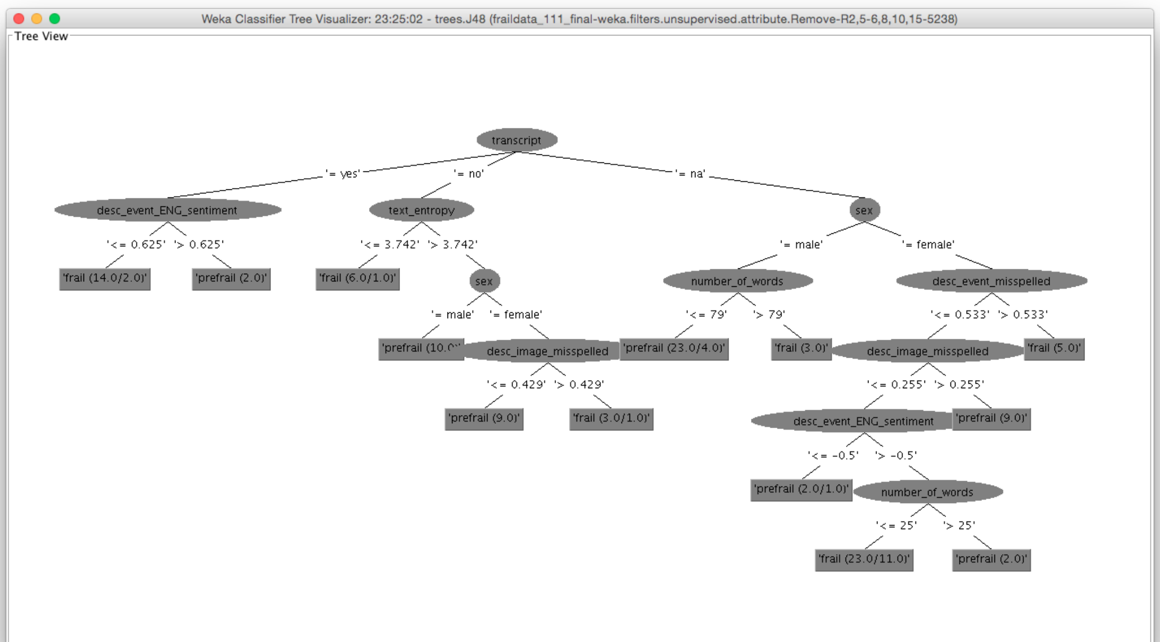


Figure 8. Decision tree based on C4.5 algorithm

The final step of tree visualization was intentionally added in order to exploit a fundamental property of the way decision trees build their structure. In more detail, these models use the information gain (Mitchell, 1997) to rank and place the best contributing features on the top of the tree. Thus, it was possible to validate the importance of its selected feature, understand its contribution and remove the remaining non important features.

***Classification process***

The automatic classification of documents into predefined categories is an important field of active research, the documents can be classified by three classes of methods:

- Unsupervised(Duda, 2001) methods, where no human intervention is required for labeling the collected data and the algorithms deployed are responsible for grouping the data to distinct categories.
- Supervised methods, usually the human expertise is used for labeling each individual instance of the dataset.
- Semi supervised methods, in this class of methods as little as possible human expertise is required to label a small initial amount of data and the algorithms exploit the existence of unlabeled data in order to enrich the training dataset.

The last few years, the task of automatic text classification has been extensively studied and rapid progress seems in this area, the machine learning approaches include the use of classifiers like Bayesian classifier (Russel, 2003), Decision Tree, K-nearest Neighbors (KNN), Support Vector Machines (SVMs) (Cortes, 1995) and Neural Networks (McCulloch, 1943).

As an essential part of the LingTester is the Frailty predictive model, the examination of the most common classifiers for text classification was conducted. The constructed dataset was used to feed the classifier using only the currently optimal features:

- transcript
- sex
- number_of_words
- text_entropy
- desc_event_eng_sentiment
- prev_text_eng_sentiment
- desc_image_mispelled
- desc_event_mispelled
- class

For the model evaluation, the well known cross validation technique was deployed. Cross validation assesses how the results of a statistical analysis will generalize to an independent dataset. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. Due to the lack of sufficient examples (only 111 labeled instances in the dataset), the common 10-fold cross validation approach was dropped and the Leave-one-out cross-validation (LOOCV) was used instead. In LOOCV is a particular case of leave-p-out cross-validation with p = 1, where a statistic on the left-out samples is computed.

The next table summarises the accuracies obtained by the trained models, as can be seen in the table, the Decision Tree model scores the best results, thus it is selected as a first classification approach for the LingTester.

| Classifier Name | Accuracy |
|---|---|
| Decision Tree (C4.5) | 62.16 % |
| NaiveBayes | 54.05 % |
| K-Nearest Neighbours | 61.26 % |
| Support Vector Machines | 31.35 % |
| Neural Networks | 61.92 % |

Table 3. Summary of classification algorithms

***Model optimization***

Before embedding the Decision Tree Classifier to the LingTester the final step of model parameter optimization was conducted using the Weka Data mining and Classification tool(Holmes, 1994). Specifically, the software enables the parameterization of the Decision Tree model using a series of eleven parameters, a table with the most important parameters and their description follows below.

| 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 4. Parameterization of the Decision Tree model

The process of model parameter optimization is a highly empirical process, although there have been some efforts in the field, for example Auto-Weka (Thornton, 2013). As this is a first approach of the classification task the simple strategy of test and recall has been followed. In order to improve the accuracy of LingTester the Train dataset was further investigated. In relation to its class (Non-frail, Pre-frail, Frail) it was judged as highly imbalanced as the Non-frail instances were only representing 12% of the class. For this reason a temporary decision was made to group the classes Non-frail and Pre-frail to restore the balance of the dataset until more data is collected by the other tasks. After the overall model optimization an astonishing 10% accuracy increase was achieved. Figure 9 and Table 5 present the model statistics and the optimized parameter values accordingly.

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

Correctly Classified Instances          80               72.0721 %
Incorrectly Classified Instances        31               27.9279 %
Kappa statistic                          0.3951
Mean absolute error                      0.3206
Root mean squared error                  0.4467
Relative absolute error                 66.3472 %
Root relative squared error             90.5213 %
Total Number of Instances              111

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.836     0.455     0.737     0.836     0.783      0.742    prefrail
                 0.545     0.164     0.686     0.545     0.608      0.742    frail
Weighted Avg.    0.721     0.339     0.717     0.721     0.714      0.742

=== Confusion Matrix ===

  a  b   <-- classified as
 56 11 |  a = prefrail
 20 24 |  b = frail
```

Figure 9. Model statistics

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

Table 5. Parameterization values of the Decision Tree model

# 6. Conclusions

The obtained accuracy of 72% by the common Decision Tree classifier seems promising and is a good starting point for the construction of more complex ensemble models. Although many of the state of the art techniques were implemented for the feature extraction process, they can not be exploited at the current phase of the project due to the small amount of collected data. As the dataset grows it is expected more features to contribute to the performance of the predictive model therefore a more capable and accurate model can be obtained and integrated to the FrailSafe user software package.

The further reduction of features and the hyper-optimization of the model showed to have slightly better test performance but in practice is an overfit (Everitt, 2002) of the current dataset and it will probably lead to worse overall results as the dataset examples increase in the future.

# 7. System development

## 7.1 Software used for training and prediction

Software development was based on various programming languages along with the use of third party tools, for both database management, creating models and predicting results. First steps of database management and export tools are based on Python (see Annex 12.1 Database management & feature extraction), while the use of the prediction model is based on a Java implementation (see Annex 12.2 Prediction model). Both programming languages are operating system independent, meaning that upon successful installation of Python console for the first, and Java Virtual machine for the later, provided programmes may start processing.

Furthermore, all third party tools within the discussed methodology should be provided. Firstly, the Greek POS tagger is a third party tool that needs Java Virtual Machine to run. It can be downloaded from the official site of Natural Language Processing Group at Department of Informatics - Athens University of Economics and Business (http://nlp.cs.aueb.gr/software.html). It is executed automatically from the Python script, whenever we request POS information for existing or new patient data.

For the classification process WEKA suite was used. Weka is a collection of machine learning algorithms for data mining tasks. 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. In the following section, we discuss steps to reproduce all installations steps.

## 7.2 Installation notes for training and prediction

Extra steps must be carried out so as for all submodules to work correctly, for the current state of LingTester. While the following submodules are platform independent, this doesn't mean that steps are also the same. Each platform may request different dependencies, but this analysis will assume Linux as the underlying operating system. These are the main parts:

- Python core and Python modules
- Java Virtual Machine
- POS Tagger
- OpenOffice dictionaries
- Weka

### 7.2.1 Python core and Python modules

Python is preinstalled in all Linux distros. However, we must install some extra modules. The following modules can be easily installed using the `pip` package, by issuing a command of type `sudo pip install {module}` for each module. These instructions are also described within the source code. The modules are:

1. **pyenchant**, for the translation service
2. **httplib2**, to request access to the MyMemory service through http protocol
3. **nltk**, to export tf-idf features
4. **pattern**, to export sentiment features in english

### 7.2.2 Java Virtual Machine (JVM)

In order to download and install the Java Virtual Machine, user must navigate to the download page[1], click Download and execute the downloaded file. Installation will automatically finalise. JVM is needed in order to execute both the ready made model and the POS tagger, in case it is needed for new data.

### 7.2.3 POS Tagger

POS tagger is available for download from the official page[2]. Assuming JVM works as expected, downloading and installing all files as stated in the readme file from the downloaded archive is all that is needed to have access to this service.

### 7.2.4 OpenOffice dictionaries

While `pyechant` python module gives access to the speller of OpenOffice, we have to install the speller for each language, in case it is not already installed. In Linux, we can install new dictionaries by issuing in the command line the command `sudo apt-get install myspell-gr-el`. In case we have a different package manager, we issue the install command for the `myspell-gr-el` package.

### 7.2.5 Weka

We can easily download and install WEKA by navigating to the page official[3] and following the detailed steps in that page, according to our operating system.

[1] https://www.java.com/en/
[2] http://nlp.cs.aueb.gr/software.html
[3] http://www.cs.waikato.ac.nz/ ml/weka/downloading.html

# 8. Frailsafe user software

## 8.1 Steps to import new data

There are two reasons to import new data within our offline database. First one is to populate new training data and reconstruct the prediction model, a process which cannot be done outside laboratory or by non technical persons. The reason for this, is that a full installation of WEKA is needed along with various parameters that make this procedure not feasible for everyday use. The second reason is to use the model that has already been trained and exported for easy use (see next chapter). For this to work, we execute the secondary file offline-parser-ui.py through the console, `python offline-parser-ui.py`, and the following menu appears:

1. *Validate corpus and print statistics:* This option goes through each patient data, and validates there are no missing or not accepted information
2. *Print all patient data:* This option requests a patient id from the user, and dumps all patient data within the console for easy access.
3. *Create new patient data:* This option will start by asking the ID of the new patient data and in case this ID already exists, then an error is shown to the user and he has to start over. This validation is needed in order to avoid overwriting existing data by accident.
4. Update database with missing translations: As the name suggests, this option will automatically update any missing translations
5. *Update database with missing POS data:* Also, as the name suggests, this will update whole database with any missing POS data for future use.
6. *Export patient data for prediction:* This option requests a patient id from the user and then creates the ARFF that is needed in order for the prediction model to work correctly (see next chapter).
7. *Exit:* This will force for the application to terminate

As this tool, is also Python based, it is cross-platform. However, this doesn't mean it can work out of the box. Extra steps need to be done in order for all submodule operate normally (See *Installation Notes*, under System development chapter).

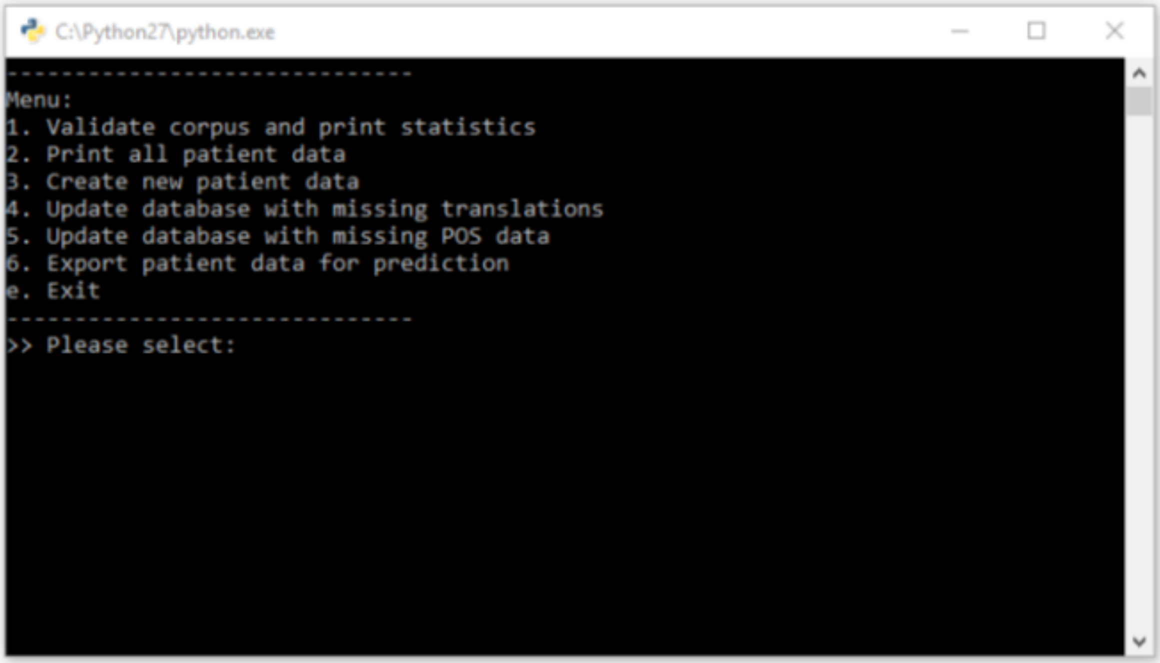


Figure 10. Screenshot of real time operation

## 8.2 Steps to export results

In order to for the FrailSafe user to obtain predictions for a number of subjects, a java software package was developed. The official name of the package is Predictor due to the task assigned to it.

The Predictor is a cross-platform command line tool that expects as input an arff file containing the pre-processed collected data of the subjects exported by the user using the Offline-parser-ui tool. Currently only the default ``in.arff'' can be used and Predictor expects it in the current working directory. An example input file structure is presented in the following figure.

```
in.arff
@relation frailsafe_111

@attribute transcript {yes,no,na}
@attribute class {prefrail,frail}
@attribute sex {male,female}
@attribute number_of_words numeric
@attribute text_entropy numeric
@attribute desc_event_ENG_sentiment numeric
@attribute prev_text_ENG_sentiment numeric
@attribute desc_image_misspelled numeric
@attribute desc_event_misspelled numeric

@data
na,?,female,48,4.023,0,0,0.429,0.571
na,?,female,43,3.869,0.625,0,0.345,0.467
na,?,male,0,0,0,0,0,0
na,?,female,33,3.805,1.25,0,0.4,0.368
na,?,male,0,0,0,0,0,0
na,?,female,0,0,0,0,0,0
```

Figure 11. Input file structure

In order for the Predictor not be computationally expensive, the training of the classifier at runtime of the tool was intentionally avoided. Instead, a pre-computed model of the classifier is delivered with the software package and is loaded by the tool.

The detailed program algorithm of the Predictor tool is presented below.

**Predictor tool algorithm**

---

**Input:**

Load the pretrained model (M)
Load the Test dataset (T)

Count the number of test instances (N)
**Loop for N**
    Predict Instance T(i)
    Print prediction for T(i)

Software tool usage instructions:

1. Open a terminal inside the deliverables root directory
2. Run the command `java -jar predictor-cli.jar`

Notes: The user's system should use java version 1.6.0 or higher.

It is recommended to use the `-Xmx2g` java parameter if the input arff file has more than a few thousand instances.

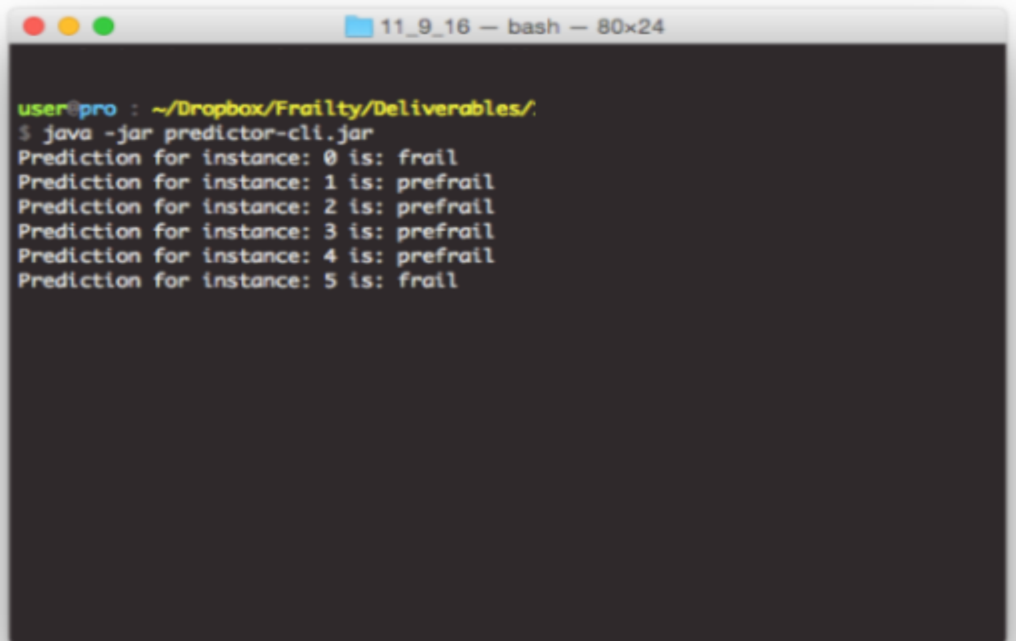


Figure 12. Screenshot of prediction programme while executing

# 9. Future improvements for the Prototype

A series of improvements is expected to improve the overall performance, add more predictive capabilities and improve the LingTester user experience. More specifically, on the subject of data collection, management and dataset exportation, a number of actions like the offline database population with new collected data, the features evaluation against a different input language (French language) and the optimization and bug fix of current features and feature extraction methodology is expected to improve the quality of the train dataset. On the subject of the classification task, the future research includes the exhaustive research on feature selection methodologies, the experimentation on supervised ensemble classifier models and the deployment of semi-supervised (Chapelle, 2006) techniques in order enhance the small available dataset with a bigger unlabeled (Ratsaby, 1995) dataset. Moreover, two new areas of research are going to be explored, the detection of suicidal statements and the patient mental state transition in time.

To benefit the end user of LingTester and simplify the prediction process, the integration of the two software tools, presented in section 8, in a complete and all inclusive software package will be implemented.

# 10. Ethics and Safety

Throughout this study's methodology, special care has been taken for ethical and safety issues. The nature of the study requires the processing, storing and analysis of a large amount of data. In all these stages, confidentiality and personal data protection will be reassured by an anonymization procedure. Each participant is traced solely by his/her ID, provided initially by the recruitment center, a number and only this, with no identifiable personal data, will be exposed to large scale data exchange like name, date of birth, place of living. Access to the database have only specific people, researchers, in order to create the prediction models.

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.

# 11. References


- A survey of text classification algorithms CC Aggarwal, CX Zhai - Mining text data, 2012 - Springer
- Gerard Salton, Edward A. Fox, and Harry Wu. 1983. Extended Boolean information retrieval. Commun. ACM 26, 11 (November 1983), 1022-1036. DOI=http://dx.doi.org/10.1145/182.358466
- Eibe, F., Mark, A. Hall, and Ian, H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.
- K. Denecke, "Using SentiWordNet for multilingual sentiment analysis," Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on, Cancun, 2008, pp. 507-512. doi: 10.1109/ICDEW.2008.4498370
- Koleli, E. A new Greek part-of-speech tagger, based on a maximum entropy classifier. Diss. Thesis, Athens University of Economics, 2011.
- Tsakalidis, A., Papadopoulos, S. and Kompatsiaris I. (2014) An Ensemble Model for Cross-Domain Polarity Classification on Twitter. Proceedings of 15th International Conference on Web Information Systems Engineering (WISE 2014), LNCS, Part II, pp. 168-177, Thessaloniki, Greece, October 12-14, 2014, Springer
- Gareth James; Daniela Witten; Trevor Hastie; Robert Tibshirani (2013). An Introduction to Statistical Learning. Springer.
- Bermingham, Mairead L.; Pong-Wong, Ricardo; Spiliopoulou, Athina; Hayward, Caroline; Rudan, Igor; Campbell, Harry; Wright, Alan F.; Wilson, James F.; Agakov, Felix; Navarro, Pau; Haley, Chris S. (2015). "Application of high-dimensional feature selection: evaluation for genomic prediction in man". Sci. Rep.
- Pearl, J. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.
- Khachaturyan, A.; Semenovskaya, S.; Vainshtein, B. (1979). "Statistical-Thermodynamic Approach to Determination of Structure Amplitude Phases". Sov.Phys. Crystallography.
- Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press.
- Geisser, Seymour (1993). Predictive Inference. New York, NY: Chapman and Hall.
- Mitchell, Tom M. (1997). Machine Learning. The Mc-Graw-Hill Companies, Inc.
- Duda, Richard O.; Hart, Peter E.; Stork, David G. (2001). "Unsupervised Learning and Clustering". Pattern classification (2nd ed.). Wiley.
- Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall.
- Cortes, C.; Vapnik, V. (1995). "Support-vector networks". Machine Learning.

- McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics.
- G. Holmes; A. Donkin; I.H. Witten (1994). «Weka: A machine learning workbench». Proc Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia.
- Chris Thornton, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown, Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In Proc. of KDD 2013
- Everitt B.S. (2002) Cambridge Dictionary of Statistics, CUP.
- Chapelle, Olivier; Schölkopf, Bernhard; Zien, Alexander (2006). Semi-supervised learning. Cambridge, Mass.: MIT Press.
- Ratsaby, J. and Venkatesh, S. Learning from a mixture of labeled and unlabeled examples with parametric side information. In Proceedings of the Eighth Annual Conference on Computational Learning Theory, pages 412-417 (1995).

# 12. Source files

These are the files that accompany this deliverable:

- Folder: demo
    - File: frailsafe.model
        - Pre-trained prediction model
    - File: in.arff
        - Test input data. This is a text file structured like figure 11. For more details, please see chapter 8.1
    - File: predictor-cli.jar
        - This java software uses the pre-trained prediction model in order to predict patients' mental state from test data found in file in.arff(6 patients' test data after feature extraction).
    - File: README.txt
        - Readme file of how to execute the java file
- Folder: source code
    - File: PredictorCLI.java
        - Source code of the demo *predictor-cli.jar* file
    - File: offline-parser.py
    - File: offline-parser-ui.py
        - Source file in Python that manage the offline database while also support the feature extraction process

# 13. Annexes

## 13.1 Database management & feature extraction

The following are the contents of the base Python library with various functions that help utilise the offline database in its full extent.

```
# -*- coding: utf-8 -*-
"""

@author: Charalampos Tsimpouris
@author: Nikolaos Fazakis


"""

# To install execute: pip install pyenchant
import enchant

# To install execute: pip install httplib2
import httplib2
import json
import math

# To install execute: pip install nltk
import nltk

import os
import pickle

# To install execute: pip install pattern
from pattern.en.wordnet import sentiment

import stemming
import shutil
import subprocess

# py-translate Cannot work
# it uses, free google services and is easilly blocked
# import translate

# Google API
# cannot be used, only with billing plans
#from google.cloud import translate

import warnings
from sklearn.feature_extraction.text import TfidfVectorizer

# This is the path where all patient data is stored
# .. one file per patient, with file name from the patient id
data_path = './Data'

frailsafe_google_api_key = '???????'

# MyMemory Translation
# .. these are basic settings for the translation service
mymemory_account_email = 'kifinas.uop@gmail.com'
mymemory_base_url = u'http://api.mymemory.translated.net/get'

# These tags must always exist
# .. along with the availiable tags
# .. and they are class tags
verify_tags = {}
verify_tags['-transcript'] = ('yes', 'no', 'na')
verify_tags['-sex'] = ('male', 'female')
verify_tags['-tag'] = ('nonfrail', 'prefrail', 'frail', 'na')
verify_tags['-language'] = ('greek', 'greek-polytonic', 'french', 'greek-cypriot')

# Languge convert helper dictionaries for various reasons
langs = {}
langs['greek'] = 'el'
langs['greek-polytonic'] = 'el'
langs['french'] = 'fr'
langs['greek-cypriot'] = 'el-cy'

langs_speller = {}
langs_speller['greek'] = 'el_GR'
langs_speller['greek-polytonic'] = 'el_GR'
langs_speller['french'] = 'fr'
langs_speller['greek-cypriot'] = 'el_GR'

# These tags are multi line
```

```python
# .. and make the multi text that we try to classify from
multi_line_tags     = ['-desc_image',     '-desc_event',     '-prev_text']

# Part of speech info, in case there is one
multi_line_tags_POS = ['-desc_image_POS', '-desc_event_POS', '-prev_text_POS']

# English translation of the initial text
multi_line_tags_ENG = ['-desc_image_ENG', '-desc_event_ENG', '-prev_text_ENG']

def split_list():
    """This function tries to split the main initial patient
        list in a simplistic way.
        Caution, detroyes existing data.
    """
    f = open(data_path + '/lists.txt', 'r')
    lines = f.readlines()
    f.close()

    patients = {}
    for line in lines:
        if not line.startswith('-patient'):
            continue

        tag, pid = line.strip().split(' ')
        if pid in patients:
            print "Patient %s already set" % pid
            print "Aborting"
            return
        patients[ pid ] = line


    cpid = None
    pdata = []

    for line in lines:
        line = line.strip()

        if line.startswith('-patient'):
            if cpid is not None:
                f = open(data_path + '/p.%s.txt' % cpid, 'w')
                f.write("\n".join(pdata).strip() + "\n")
                f.close()

            pdata = []
            tag, cpid = line.strip().split(' ')

        pdata.append(line)

    if cpid is not None:
        f = open(data_path + '/p.%s.txt' % cpid, 'w')
        f.write("\n".join(pdata).strip() + "\n")
        f.close()

def fetch_patient_ids():
    """Identifies all patient ids, as stated in the filenames
    """
    files = os.listdir(data_path)
    out_files = []
    for f in files:
        if not f.startswith('p.') or not f.endswith('.txt'):
            continue

        pre, pid, suf = f.strip().split('.')
        out_files.append(pid)

    out_files.sort()
    return out_files

def fetch_patient_data(cpid):
    """Tries to load all patient data, based on the id
    """
    try:
        f = open(data_path + '/p.%s.txt' % cpid, 'r')
        lines = f.readlines()
        f.close()
    except:
        print 'Error opening patient data file %s' % cpid
        return {}

    ret = {}
    ctag = None
    for line in lines:
        line = line.strip()
        if line.startswith('-'):
            if ctag:
                ret[ctag] = ret[ctag].strip()

            if line.find(' ') > 0:
                ctag = line[:line.find(' ')].strip()
                info = line[line.find(' ')+1:].strip()
            else:
                ctag = line
```

```
                info = ''
            ret[ctag] = ''
            ret[ctag] += info
            continue

        ret[ctag] += line + "\n"
    if ctag:
        ret[ctag] = ret[ctag].strip()

    return ret

def save_patient_data(cpid, cpdata):
    """Saves all patient data in a file
    """
    f = open(data_path + '/p.%s.txt' % cpid, 'w')

    cpdata['-patient'] = cpid

    for tag in cpdata:
        if cpdata[tag] is None:
            cpdata[tag] = ''

        if tag in multi_line_tags or tag in multi_line_tags_POS or tag in multi_line_tags_ENG:
            f.write("%s\n\n" % tag)
            f.write("%s\n\n" % cpdata[tag].strip())
            continue

        f.write("%s %s\n" % (tag, str(cpdata[tag]).strip()))
    f.close()
    return cpdata

def print_patient_data(pdata):
    """Tries to print all patient data, as beautiful as it can
    """
    for k in pdata:
        print '%s:%s' % (k, pdata[k])

def validate_patient_data():
    """Validates there are no missing tags
        in all patient files, around the required ones
        Also, it tries to print some minor statistics
    """
    pids = fetch_patient_ids()

    stats = {}
    for cpid in pids:
        cpdata = fetch_patient_data(cpid)

        for t in verify_tags:
            valid = verify_tags[t]
            if not t in cpdata or cpdata[t] is None or cpdata[t] == '':
                print 'Patient %s is missing %s data' % (cpid, t)
                continue

            if cpdata[t] not in valid:
                print 'Patient %s has invalid %s data %s' % (cpid, t, cpdata[t])
                continue

            if not t in stats:
                stats[t] = {}
            stats[t][ cpdata[t] ] = stats[t].get(cpdata[t], 0) + 1

    print 'Checked %d patients, and here are the statistics' % len(pids)
    for t in verify_tags:
        valid = verify_tags[t]

        print t
        for k in valid:
            print '  %s:%d' % (k, stats[t].get(k, 0))

def temp_change_tags():
    """Makes a change of tags, after initial estimation
        Temporary function
    """
    pids = fetch_patient_ids()

    for cpid in pids:
        cpdata = fetch_patient_data(cpid)
        if '-desc_image' not in cpdata:
            cpdata['-desc_image'] = cpdata.get('-perigrafi_eikonas', '')
            del cpdata['-perigrafi_eikonas']

        if '-desc_event' not in cpdata:
            cpdata['-desc_event'] = cpdata.get('-perigrafi_gegonotos', '')
            del cpdata['-perigrafi_gegonotos']

        save_patient_data(cpid, cpdata)

def clean_up_text(text, lang = 'english'):
    """Tries to clean up text, as much as possible
```

```
        in some ways language indipendantly
        in some ways not
    """
    new_list = []

    text = text.replace('.', '. ')
    # Minor clean up per language
    if lang.startswith('greek'):
        text = clean_greek_letters(text)
    elif lang == 'french':
        text = clean_french_letters(text)

    text = upper(text)

    words = get_words(text)
    for w in words:
        if len(w) <= 3:
            continue

        w = stemming.stem(w)

        new_list.append(w)

    return ' '.join(new_list)

def create_arff(relation = 'fraildata'):
    """Create arff for WEKA with all features availiable
    """
    pids = fetch_patient_ids()

    out = []
    out.append('@RELATION %s' % relation)
    out.append('')

    basic_tags = []
    for t in verify_tags:
        basic_tags.append(t)
        valid = verify_tags[t]
        tag = t.lstrip('-')
        if tag == 'tag':
            tag = 'class'
            valid = ('nonfrail', 'prefrail', 'frail')
        out.append('@ATTRIBUTE %s {%s}' % (tag, ','.join(valid)))

    other_attributes = []
    other_attributes.append('get_feature_length')
    other_attributes.append('get_feature_number_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:
        out.append('@ATTRIBUTE %s %s' % (globals()[attr]('', 'title'), globals()[attr]('', 'type')))

    for tag in multi_line_tags_ENG:
        out.append('@ATTRIBUTE %s %s' % (tag.lstrip('-') + '_sentiment', 'real'))

    for tag in multi_line_tags:
        out.append('@ATTRIBUTE %s %s' % (tag.lstrip('-') + '_misspelled', 'real'))

    corpus = get_corpus()

    labels = []
    texts = []
    text_POS = []
    for cpid in corpus:
        labels.append(cpid)
        texts.append(corpus[cpid]['clean_text'])

        pos_data = pos_explode_data(corpus[cpid]['text_POS'])
        temp_pos_as_text = []
        # ta panta ola
        for word, ps in pos_data:
            temp_pos_as_text.append('8'.join(ps))

        # 1:1 mono ta prwta
        for word, ps in pos_data:
            temp_pos_as_text.append(ps[0])

        # 1:1 mono ta prwta/deftera
        for word, ps in pos_data:
            temp_pos_as_text.append('8'.join(ps[0:1]))

        # 1:1 mono ta prwta/deftera/trita
        for word, ps in pos_data:
            temp_pos_as_text.append('8'.join(ps[0:2]))

        # 1:1 mono ta deftera
        for word, ps in pos_data:
            temp_pos_as_text.append(ps[0])
```

```
        # 1:1 mono ta trita
        for word, ps in pos_data:
            temp_pos_as_text.append(ps[0])

        text_POS.append(' '.join(temp_pos_as_text).replace('-', ''))

    # TF-IDF on POS data
    tf_POS = TfidfVectorizer(analyzer='word', ngram_range=(1,2), min_df = 0)
    tfidf_matrix_POS = tf_POS.fit_transform(text_POS)
    feature_names_POS = tf_POS.get_feature_names()

    for i in range(len(feature_names_POS)):
        out.append('@ATTRIBUTE tf-pos-%d real %% %s' % (i, 'POS: ' + feature_names_POS[i]))
    dense_POS = tfidf_matrix_POS.todense()

    # TF-IDF on stemmed text
    tf = TfidfVectorizer(analyzer='word', ngram_range=(1,2), min_df = 0)
    tfidf_matrix =  tf.fit_transform(texts)
    feature_names = tf.get_feature_names()

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

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

    for i in range(len(labels)):
        cpid = labels[i]

        # Patients with inknown frailty tag are automatically removed
        if not '-tag' in corpus[cpid]['data'] or corpus[cpid]['data']['-tag'] == 'na':
            print "Skipping patient with ID %s: missing frailty tag" % cpid
            continue

        row = []
        for bt in basic_tags:
            row.append( corpus[cpid]['data'].get(bt, 'na') )

        for attr in other_attributes:
            clang = corpus[cpid]['data']['-language']

            # To absorb all Greek variations
            if clang.startswith('greek'):
                clang = 'greek'

            row.append(str(globals()[attr](corpus[cpid]['text'], lang = clang)))

        # Sentiment score is based in the english translation
        for tag in multi_line_tags_ENG:
            row.append(get_feature_sentiment_score(corpus[cpid]['data'].get(tag, '')))

        for tag in multi_line_tags:
            row.append(get_feature_mispelling_score(corpus[cpid]['data'].get(tag, ''), lang = clang))

        # tf-idf info based on POS data
        p_POS = dense_POS[i].tolist()[0]
        for fi in range(len(feature_names_POS)):
            row.append('%.3f' % p_POS[fi])

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

        out.append(','.join(row))

    filename = 'ARFFS/%s.arff' % relation
    f = open(filename, 'w')
    f.write("\n".join(out).encode('utf8'))
    f.close()

    print 'Saved at %s.' % filename

def get_corpus():
    """Returns all corpus in a more scientific-friendly way
    """
    pids = fetch_patient_ids()

    corpus = {}

    for cpid in pids:
        cpdata = fetch_patient_data(cpid)
        corpus[cpid] = {}
        corpus[cpid]['data'] = cpdata
```

```
        corpus[cpid]['tag'] = cpdata['-tag']

        corpus[cpid]['text'] = ''
        for m in multi_line_tags:
            corpus[cpid]['text'] += cpdata.get(m, ' ')
        corpus[cpid]['text'] = corpus[cpid]['text'].strip()

        corpus[cpid]['text_POS'] = ''
        for m in multi_line_tags_POS:
            corpus[cpid]['text_POS'] += "\n" + cpdata.get(m, ' ')

        corpus[cpid]['text_ENG'] = ''
        for m in multi_line_tags_ENG:
            corpus[cpid]['text_ENG'] += "\n" + cpdata.get(m, ' ')

        clang = corpus[cpid]['data']['-language']

        # To absorb all Greek variations
        if clang.startswith('greek'):
            clang = 'greek'

        tutf8 = corpus[cpid]['text'].decode('utf-8')
        corpus[cpid]['clean_text'] = clean_up_text(tutf8, clang)

    return corpus

def get_feature_length(text, meta = None, lang = 'english'):
    if meta == 'title':
        return 'text_length'
    if meta == 'type':
        return 'integer'

    return len(text)

def get_sentences(text, lang = 'english'):
    """This should be language dependant to be more precise
    """
    sent_tok_file = "greek.law.utf8.70.pickle"

    f = open(sent_tok_file)
    sent_tokenizer = pickle.load(f)
    f.close()

    return sent_tokenizer.tokenize(text.decode('utf8'))

def get_feature_number_of_sentences(text, meta = False, lang = 'english'):
    """Returns a feature with number of sentences
    TODO: This has to be more elaborate
    """
    if meta == 'title':
        return 'number_of_sentences'
    if meta == 'type':
        return 'integer'

    return len(get_sentences(text, lang))

def get_words(text, lang = 'english'):
    """Splits text in words
    """
    word_tokenizer = nltk.WhitespaceTokenizer()

    # Word tokenizer, auto parses sentences
    # ..so no need to split in sentences
    return word_tokenizer.tokenize(text)

def get_feature_word_count(text, meta = False, lang = 'english'):
    """Returns a feature with number of words
    TODO: This has to be more elaborate
    """
    if meta == 'title':
        return 'number_of_words'
    if meta == 'type':
        return 'integer'

    return len(get_words(text, lang))

def get_feature_words_per_sentence(text, meta = False, lang = 'english'):
    """Returns a feature with number of words
    TODO: This has to be more elaborate
    """
    if meta == 'title':
        return 'number_of_words_per_sentence'
    if meta == 'type':
        return 'real'

    if int(get_feature_length(text)) <= 0:
        return 0

```

```
            return '%.3f' % (float(get_feature_word_count(text)) / get_feature_number_of_sentences(text))

    def get_feature_text_shannon_entropy(text, meta = False, lang = 'english'):
        """Returns bits of entropy represented in a given string, per
        http://en.wikipedia.org/wiki/Entropy_(information_theory)
        """
        if meta == 'title':
            return 'text_entropy'
        if meta == 'type':
            return 'real'

        mmap = {}
        for c in text:
            mmap[c] = mmap.get(c, 0) + 1

        text_len = get_feature_length(text)
        result = 0.0

        for c in mmap:
            freq = mmap[c] / float(text_len)
            result -= freq * (math.log(freq) / math.log(2))

        return '%.3f' % result

    def get_feature_sentiment_score(text, meta = False, lang = 'english'):
        """Returns sentiment score, works based on the english translation
        """
        if meta == 'title':
            return 'sentiment_score'
        if meta == 'type':
            return 'real'

        v = 0
        for w in text.split(" "):
            w = w.strip(",.!?)(#:;\"\'").lower()
            if w in sentiment:
                v = v + sentiment[w][0] - sentiment[w][1]
        return str(v)

    def get_feature_mispelling_score(text, meta = False, lang = 'english'):
        """Returns mispelling statistics
        """
        if meta == 'title':
            return 'mispelling_score'
        if meta == 'type':
            return 'real'

        if not lang in langs_speller:
            warnings.warn('Unknown input language: %s' % (from_lang))
            return ''
        slang = langs_speller[lang]

        word_counting = 0
        misspelled_words = 0

        d = enchant.Dict(slang)
        for w in get_words(text, lang):
            word_counting += 1
            if not d.check(w):
                misspelled_words += 1

        if word_counting <= 0:
            return '0.0'

        return '%.3f' % (float(misspelled_words) / float(word_counting))

    def clean_greek_letters(text):
        text = text.replace(u'Î‘', u'Î±')
        text = text.replace(u'Î’', u'Î²')
        text = text.replace(u'Î“', u'Î³')
        text = text.replace(u'Î”', u'Î´')
        text = text.replace(u'Î•', u'Îµ')
        text = text.replace(u'Î–', u'Î¶')
        text = text.replace(u'Î—', u'Î·')
        text = text.replace(u'Î˜', u'Î¸')
        text = text.replace(u'Î™', u'Î¹')
        text = text.replace(u'Îš', u'Îº')
        text = text.replace(u'Î›', u'Î»')
        text = text.replace(u'Îœ', u'Î¼')
        text = text.replace(u'Î�', u'Î½')
        text = text.replace(u'Îž', u'Î¾')
        text = text.replace(u'ÎŸ', u'Î¿')
        text = text.replace(u'Î ', u'Ï€')
        text = text.replace(u'Î¡', u'Ï�')
        text = text.replace(u'Î£', u'Ïƒ')
        text = text.replace(u'Ï‚', u'Ïƒ')
        text = text.replace(u'Î¤', u'Ï„')
        text = text.replace(u'Î¥', u'Ï…')
        text = text.replace(u'Î¦', u'Ï†')
        text = text.replace(u'Î§', u'Ï‡')
        text = text.replace(u'Î¨', u'Ïˆ')
        text = text.replace(u'Î©', u'Ï‰')
        text = text.replace(u'Î†', u'Î±')
        text = text.replace(u'Îˆ', u'Îµ')
        text = text.replace(u'Î‰', u'Î·')
        text = text.replace(u'ÎŠ', u'Î¹')
        text = text.replace(u'Îª', u'Î¹')
        text = text.replace(u'ÎŒ', u'Î¿')
```

```
        text = text.replace(u'ÎŽ', u'Ï…')
        text = text.replace(u'Î«', u'Ï…')
        text = text.replace(u'Î⍰', u'Ï‰')

        text = text.replace(u'Î¬', u'Î±')
        text = text.replace(u'Î-', u'Îµ')
        text = text.replace(u'Ï⍰', u'Ï…')
        text = text.replace(u'Î¯', u'Î¹')
        text = text.replace(u'ÏŒ', u'Î¿')
        text = text.replace(u'Î®', u'Î·')
        text = text.replace(u'ÏŽ', u'Ï‰')
        text = text.replace(u'ÏŠ', u'Î¹')
        text = text.replace(u'Ï‹', u'Ï…')
        text = text.replace(u'Î⍰', u'Î¹')
        text = text.replace(u'Î°', u'Ï…')

        text = text + u' '
        # text = text.replace(u'Î½ ', u' ') # p.x. to "ÎµÎ½Î¿Ï⍰Î¹Î±Î½" ginetai "ÎµÎ½Î¿Ï⍰Î¹Î±"
        text = text.replace(u'Ïƒ ', u'Ï‚ ') #
        text = text.strip()

        return text

    def clean_french_letters(text):
        """Currently not implemented"""
        return text


    def upper(text):
        """Capitilizes text"""
        text = text.replace(u'Î±', u'Î‘')
        text = text.replace(u'Î²', u'Î’')
        text = text.replace(u'Î³', u'Î“')
        text = text.replace(u'Î´', u'Î”')
        text = text.replace(u'Îµ', u'Î•')
        text = text.replace(u'Î¶', u'Î–')
        text = text.replace(u'Î·', u'Î—')
        text = text.replace(u'Î¸', u'Î˜')
        text = text.replace(u'Î¹', u'Î™')
        text = text.replace(u'Îº', u'Îš')
        text = text.replace(u'Î»', u'Î›')
        text = text.replace(u'Î¼', u'Îœ')
        text = text.replace(u'Î½', u'Î⍰')
        text = text.replace(u'Î¾', u'Îž')
        text = text.replace(u'Î¿', u'ÎŸ')
        text = text.replace(u'Ï€', u'Î ')
        text = text.replace(u'Ï⍰', u'Î¡')
        text = text.replace(u'Ïƒ', u'Î£')
        text = text.replace(u'Ï‚', u'Î£')
        text = text.replace(u'Ï„', u'Î¤')
        text = text.replace(u'Ï…', u'Î¥')
        text = text.replace(u'Ï†', u'Î¦')
        text = text.replace(u'Ï‡', u'Î§')
        text = text.replace(u'Ïˆ', u'Î¨')
```

```
        text = text.replace(u'Ï‰', u'Î©')

        return text

    def get_pos_info(text, debug = '', pos_directory = '/media/xaris/Data/PhD/POS/bin'):
        """Tries to execute POS tagger, and retrieves the results
        from the exported file"""

        if not text or text == '':
            return ''

        # Till a better solution
        # .. everything has to be done where the java files are
        # .. so we hardcode pos_directory
        # .. but keep it as a parameter

        current_directory = os.getcwd()
        os.chdir(pos_directory)

        # this is the file where data will be stored
        in_file = pos_directory + '/in.txt'

        f = open(in_file, 'w')
        f.write(text)
        f.close()

        # output file is hardcoded according to maintainer
        out_file = pos_directory + '/result.txt'

        # Keep everything clean
        f = open(out_file, 'w')
        f.write('')
        f.close()

        res = subprocess.call(['java', '-jar', 'POStagger.jar', '1', in_file])
        if res != 0:
            # This means that program terminated with error
            return ''

        if not os.path.exists(out_file):
            # I don't why this can happen
            warnings.warn('Output file from POS method was empty, debug data: %s' % (str(debug)))
            return ''

        f = open(out_file, 'r')
```

```
    ret = f.readlines()
    f.close()

    # And resoter current working directory
    os.chdir(current_directory)

    return "".join(ret)

def get_translated_data(data, from_lang, to_lang = 'en', debug = ''):
    """Tries to translate the text to english
        using Google Translate API
    """
    if not from_lang in langs:
        warnings.warn('Unknown input language: %s' % (from_lang))
        return ''
    from_lang = langs[from_lang]

    if from_lang == to_lang:
        # Apparently there is no need to call the API
        return data

    # The following is based on py-translate
    # and is blocked for overuse
    # -------------------------------------------------------
    # return translate.translator(from_lang, to_lang, data)
    # -------------------------------------------------------

    # The following is based on Google API
    # and is only on paid services
    # -------------------------------------------------------
    # translate_client = translate.Client(frailsafe_google_api_key)
    # translation = translate_client.translate(data, source_language = from_lang, target_language = to_lang)
    # print('Text: {}'.format(text))
    # print('Translation: {}'.format(translation['translatedText'].encode('utf-8')))
    # -------------------------------------------------------

    # The following is based at MyMemory service
    # -------------------------------------------------------
    lines = data.split("\n")
    trans_result = ''
    for line in lines:
        line = line.strip()
        if line == '':
            continue

        f = {}
        f['q'] = line
        f['langpair'] = '%s|%s' % (from_lang, to_lang)
        f['of'] = 'json'
        f['de'] = mymemory_account_email

        resp, json_content = httplib2.Http().request("%s?%s" % (mymemory_base_url, urllib.urlencode(f)))
        result = json.loads(json_content)

        if result['responseStatus'] != 200:
            print "Error from mymory, aborting.. Debug: %s" % debug
            return ''

        trans_result += result['responseData']['translatedText'] + "\n"
    # -------------------------------------------------------

    return trans_result

def update_corpus_with_translations(force_rebuild = False):
    """Get all text data from all patients
        and updates the corpus with missing translations
        In order to avoid overuse of the third-part service,
        we save locally the translation for future use.

        The force_rebuild parameter, will force to update all translations
    """
    pids = fetch_patient_ids()

    for cpid in pids:
        cpdata = fetch_patient_data(cpid)
        updated = False
        for mt in multi_line_tags:
            d = cpdata.get(mt, '')
            # Adeio keimeno
            if d == '' and not force_rebuild:
                continue

            # Exw idi ipologisei POS data
            if cpdata.get(mt + '_ENG', '') != '' and not force_rebuild:
```

```
                continue

            # Den to exoume, as to paroume
            ret = get_translated_data(d, cpdata['-language'], debug = cpid)
            if ret == '' and not force_rebuild:
                continue

            updated = True
            cpdata[mt + '_ENG'] = ret

        # Something changed, time to store it
        if updated:
            print 'Updating patient %s' % cpid,
            save_patient_data(cpid, cpdata)
            print '..Done'

def get_sentiment_analysis_greek(data, lang = 'greek'):
    """The following idea was based on a locally saved, sentiment analysis file
        .. but due to poor results, this idea got rejected
        .. and we shifted to the sentiment analysis of the english translation
    """
    # Load all data from sentiment corpus
    f = open('%s-sentiment-lexicon/%s_sentiment_lexicon.tsv' % (lang, lang), 'r')
    lines = f.readlines()
    f.close()

    titles = lines[0].lower().split('\t')
    del lines[0]

    sentiment_words = {}
    for line in lines:
        parts = line.split('\t')

        parts[0].rtim('-Î· -Î¿')
        term = stemming.stem(parts[0])

        sentiment_words[ term ] = {}
        for p in range(len(parts)):
            sentiment_words[ term ][ titles[p] ] = parts[p]

    for k in sentiment_words.keys():
        print k,
    print

    data_clean = clean_up_text(data, lang)
    result = {}
    result['anger'] = 0
    result['disgust'] = 0
    result['fear'] = 0
    result['happiness'] = 0
    result['sadness'] = 0
    result['surprise'] = 0

    words_identified = 0

    for word in data_clean.split(' '):
        if not word in sentiment_words:
            continue

        words_identified += 1
        for r in result:
            for k in sentiment_words[word]:
                if not k.startswith(r) or sentiment_words[word][k] == 'N/A':
                    continue

                result[r] += sentiment_words[word][k]

    if (words_identified > 0):
        for r in result:
            result[r] = float(result[r]) / float(words_identified)

    return result

def update_pos_info_everywhere(force_rebuild = False):
    """Get all text data from all patients
       and updates the corpus with Part-Of-Speech information

       All results are saved within the database
    """
    pids = fetch_patient_ids()

    for cpid in pids:
        cpdata = fetch_patient_data(cpid)
        updated = False
        for mt in multi_line_tags:
            d = cpdata.get(mt, '')
            # Text is empty
            if d == '' and not force_rebuild:
                continue

            # I already have this POS data
            if cpdata.get(mt + '_POS', '') != '' and not force_rebuild:
```

```
                continue

            # POS data is missing, let's calculate it
            ret = get_pos_info(d)
            if ret == '' and not force_rebuild:
                continue

            updated = True
            cpdata[mt + '_POS'] = ret

        # Something changed, time to store it
        if updated:
            print 'Updating patient %s' % cpid,
            save_patient_data(cpid, cpdata)
            print '..Done'

def pos_explode_data(data):
    """Explodes all POS data from a string
       .. as given by the POS tagger
       .. and returns a more programmaing-friendly object.

       In case POS tagger changes, this function must re-implemented
    """
    result = []
    lines = data.split("\n")
    for l in lines:
        if l.strip() == '' or l.find(' ') < 0:
            continue

        word, tags = l.split(' ')
        ps = tags.split('/')
        result.append((word, ps))

    return result

def print_all_possible_pos_tags():
    """Prints all POS data within our corpus
    for statistical reasons"""
    pids = fetch_patient_ids()

    per_place = {}
    for cpid in pids:
        cpdata = fetch_patient_data(cpid)
        for mt in multi_line_tags_POS:
            d = cpdata.get(mt, '')
            # Adeio keimeno
            if d == '':
                continue

            pos_data = pos_explode_data(d)
            for word, ps in pos_data:
                for pos in range(len(ps)):
                    info = ps[pos]
                    if not pos in per_place:
                        per_place[pos] = {}

                    per_place[pos][ info ] = per_place[pos].get(info, 0) + 1

    i = 0
    while i in per_place:
        print 'Position %d' % i
        keys = per_place[i].keys()
        keys.sort()
        for k in keys:
            print ' ' * 3 + '%s: %d' % (k, per_place[i][k])

        i += 1

def install_spell_greek_checker_files():
    """This must be run a root
    """
    # Linux
    # sudo apt-get install myspell-gr-el
    pass
```

# 13.2 Prediction tool

```java
package predictor;

import weka.classifiers.Classifier;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class PredictorCLI {

    public static void main(String[] args) {

        Classifier cls;
        try {
            //load model
            cls = (Classifier) weka.core.SerializationHelper.read("frailsafe.model");


            DataSource source;
            try {
                //load test data
                source = new DataSource("in.arff");
                Instances data = source.getDataSet();
                if (data.classIndex() == -1)
                   data.setClassIndex(1); //class attribute is the second attribute

                //predict & print
                for(int i=0; i<data.numInstances();i++){
                    double value=cls.classifyInstance(data.instance(i));
                    String prediction=data.classAttribute().value((int)value);
                    System.out.println("Prediction for instance: "+i+" is: "+prediction);
                }

            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```