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: D4.11

LingTester (Prototype) (vers b)

Due date of deliverable:	M24 (31th December 2017)
Actual submission date:	31st December 2017
Version:	1.6
Date:	31st December 2017

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	15/11/2016	Draft	N. Fazakis, C. Tsimpouris, K. Sgarbas (UoP)	First Draft (vers a)
1.1	22/11/2016	Draft	N. Fazakis, C. Tsimpouris	Update of introduction, list of tables, minor text corrections, addition of references (vers a)
1.2	21/12/2016	Final	N. Fazakis, C. Tsimpouris, K. Sgarbas (UoP)	Final version (vers a)
1.3	3/8/2017	Draft	C. Tsimpouris	Updates on data and images (vers b)
1.4	4/10/2017	Draft	N. Fazakis	Chapter 7 updates (vers b)
1.5	14/12/2017	Draft	N. Fazakis, C. Tsimpouris	Update of introduction, list of tables, minor text corrections, addition of references (vers b)
1.6	20/12/2017	Final	N. Fazakis, C. Tsimpouris, K. Sgarbas (UoP)	Final version (vers b)

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 development of the prototype has been completed, and the prototype 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. Ramón Sanmartín Sola		

Deliverable number:	4.11	**Title:**	LingTester (Prototype) (vers b)
Work package number:	4	**Title:**	Data management and Analysis

Date of delivery	**Contractual**	**31/12/2017 (M24)**	**Actual**	31/12/2017
Status	Draft ☐		Final ☒	
Nature	Report ☒	Demonstrator ☐ Other ☐		
Dissemination Level	Public ☐	Consortium ☒		
Abstract **(for dissemination)**	This is a confidential report 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.			
Keywords	offline data management, frailty prediction, classification			

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

Table of contents

List of Figures and Images

List of Tables

List of annexes

List of abbreviations and acronyms

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:

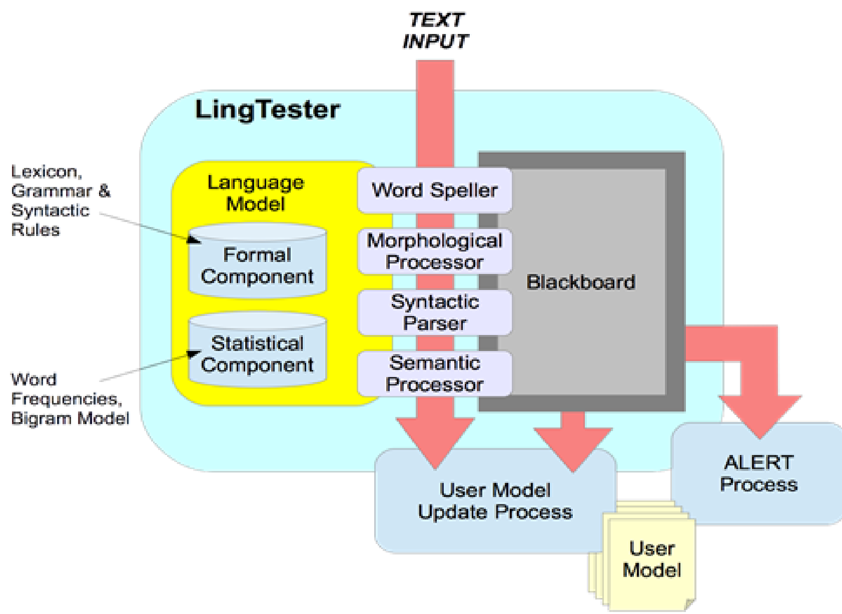


Figure 1. Initial architecture for LingTester

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).

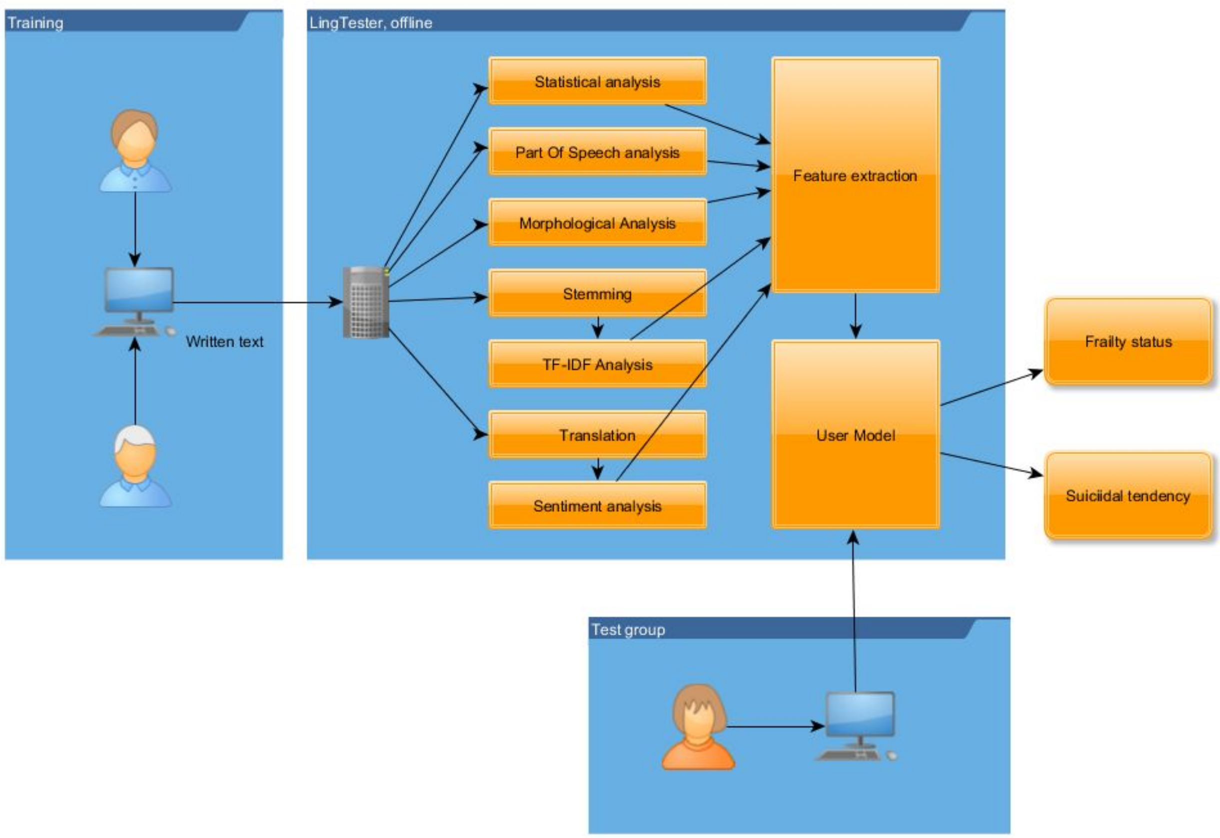


Figure 2. Current architecture for offline LingTester

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.

3.1 Data analysis

The current deliverable uses data collected until 31/10/2017 (M22). Until then we had available the following patient data:

- Data from 128 participants, UoP, Greece
- Data from 87 participants, MATERIA, Cyprus
- Data from 94 participants, MATERIA, Cyprus

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. More details about the participant's data can be found in deliverable **D4.13**.

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
-date ##/##/####

-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
- **`-date:`** The date of the written text relates to
- **`-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 *There are*	verb	--	active	plural	present
πολλά *many*	adjective	accusative	neuter	plural	-

σημαντικά *important*	adjective	accusative	neuter	plural	-
θετικά *positive*	adjective	accusative	neuter	plural	-
γεγονότα *facts*	noun	accusative	neuter	plural	-
στη *in*	article	prepositional	accusative	feminine	singular
ζωή *life*	noun	accusative	feminine	singular	-
μου *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 figure 3. 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.

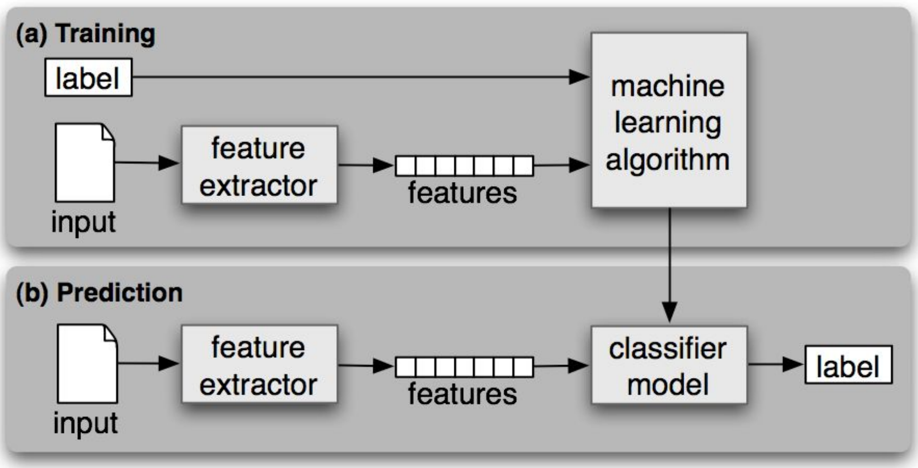


Figure 3. Training and prediction process

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, the text entropy and various readability scores.

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	Type - Extraction Method
● transcript ○ yes ○ no ● language ○ greek ○ greek-cypriot ○ french ● class ○ nonfrail ○ prefrail ○ frail ● data ○ Date of the submission for the transition study ● sex ○ male ○ female ● do_you_consider_yourself_a_familiar_user_of_social_media ○ beginner ○ less-familiar ○ very-familiar ● family_status ○ married-or-in-a-relationship ○ single	**Primitive** Rules & filters on eCRF API data

○ divorced, ○ widow ● habitation_zone ○ urban ○ semi-urban ○ rural ● have_you_changed_your_security_settings_in_social_media_in_order_to_protect_your_personal_data ○ yes ○ no ● year_of_birth ● con_per_week *connections per week* ● twitter_follows *number if people user is following on Twitter* ● twitter_followers *number of followers on Twitter* ● fb_friends *number of friends on FB*	
● text_length ● number_of_sentences ● number_of_words ● number_of_words_per_sentence ● text_entropy	**Derived** Statistical Measures
● desc_image_ENG_sentiment ● desc_event_ENG_sentiment ● prev_text_ENG_sentiment	**Derived** Sentiment Analysis
● desc_image_misspelled ● desc_event_misspelled ● prev_text_misspelled	**Derived** Percent of misspelled words based on known vocabulary
● tf-0 ● tf-1 ● ...	**Derived** Term frequency – Inverse document frequency, after feature selection based on information gain
● flesch_reading_ease ● smog_index ● flesch_kincaid_grade ● coleman_liau_index ● automated_readability_index ● dale_chall_readability_score ● difficult_words ● linsear_write_formula ● gunning_fog	**Derived** Readability score

Table 2. 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.

The first approach to feature selection, in the preliminary version of the report was the simple deployment of a custom algorithm that was based on a Decision Tree classifier and its features' information gain, as presented below.

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)

In the final version of the LingTester prototype a much stronger approach was followed, resulting in a series of multiple procedures like the ranking of features' correlation with frailty class, the deployment of a Logistic Model Tree to reduce the feature space and finally the human expertise to optimize the extracted feature space, as stated in **D4.13** in more detail. The final selected features are the following.

No.	Name
1	transcript
2	language
3	family_status
4	text_length
5	number_of_sentences
6	number_of_words
7	number_of_words_per_sentence
8	text_entropy
9	year_of_birth
10	flesch_reading_ease
11	smog_index
12	flesch_kincaid_grade
13	coleman_liau_index
14	automated_readability_index
15	dale_chall_readability_score
16	difficult_words
17	prev_text_misspelled
18	tf-5
19	tf-8
20	tf-19
21	tf-32
22	tf-33
23	tf-37
24	tf-39
25	tf-40
26	tf-45
27	tf-46
28	tf-48
29	tf-53
30	tf-58
31	tf-66
32	tf-70
33	tf-73
34	tf-74
35	tf-75
36	tf-86
37	tf-87
38	tf-91
39	tf-92
40	tf-98
41	tf-101
42	tf-102
43	tf-105
44	tf-108
45	tf-109
46	tf-110
47	tf-111
48	class

Figure 4. Selected 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.

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. In the preliminary version of the report, due to the lack of sufficient examples (only 111 labeled instances in the dataset), the common 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.

In contrast, in the final version of the LingTester prototype the frailty dataset was already bigger than double of the initial (~400 instances) and with a lot more features, thus the 10 Fold Cross Validation technique was used in order to obtain more robust results. The next table summarises the accuracies obtained by the best performing trained models.

Classifier	**Accuracy %**
Logistic	56.756
Neural Network	56.265
Simple Logistic	**61.179**
SMO	56.756
KNN	53.071
RotationForest	**59.950**
DecisionTable	51.351
HoeffdingTree	56.019

LMT	60.688
RandomForest	57.248

Table 3. Summary of classification algorithms

The final selected model used in the LingTester tool was an ensemble classifier. By utilizing a technique known as Voting meta algorithm, we combined four of the best performing models (Simple Logistic, RotationForest, LMT, RandomForest) by averaging their prediction probabilities. The final classifier will be referred as **VoteSRLR**.

Model optimization

Before embedding VoteSRLR 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 model using a series of twenty five parameters, a table with the most important parameters and their description follows below.

Parameter Name	Description	Optimum Value
Voting Algorithm		
Combination Rule	The combination to rule used.	Average of Probabilities
Simple Logistic		
Error On Probabilities	Use error on the probabilities as error measure when determining the best number of LogitBoost iterations.	False
Weight Trim Beta	Set the beta value used for weight trimming in LogitBoost.	0.1
Heuristic Stop	LogitBoost is stopped if no new error minimum has been reached in the last Heuristic Stop iterations.	50
RotationForest		
Removed Percentage	The percentage of instances to be removed.	50
Projection Filter	The filter used to project the data.	Principal Components

Classifier	The base classifier to be used	J48
LMT		
Do Not Make Split Point Actual Value	If true, the split point is not relocated to an actual data value.	False
Weight Trim Beta	Set the beta value used for weight trimming in LogitBoost.	0.0
Fast Regression	Use heuristic that avoids cross-validating the number of Logit-Boost iterations at every node.	True
RandomForest		
Calc Out Of Bag	Whether the out-of-bag error is calculated.	False
Max Depth	The maximum depth of the tree, 0 for unlimited.	0

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). In order to improve the accuracy of LingTester the Train dataset was further investigated. After the overall model optimization a `63.64%` accuracy was achieved. Figure 5 presents the VoteSRLR model statistics.

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

Correctly Classified Instances         259               63.64 %
Incorrectly Classified Instances       148               36.36 %
Kappa statistic                          0.4331
Mean absolute error                      0.3453
Root mean squared error                  0.4116
Relative absolute error                 79.4175 %
Root relative squared error             88.2764 %
Total Number of Instances              407

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.729    0.202    0.654      0.729    0.689      0.514    0.830     0.684     nonfrail
                 0.649    0.314    0.592      0.649    0.619      0.331    0.713     0.604     prefrail
                 0.485    0.062    0.716      0.485    0.578      0.490    0.796     0.643     frail
Weighted Avg.    0.636    0.214    0.644      0.636    0.633      0.433    0.773     0.641

=== Confusion Matrix ===

   a   b   c   <-- classified as
 102  34   4 |   a = nonfrail
  44 109  15 |   b = prefrail
  10  41  48 |   c = frail
```

Figure 5. Model statistics

5.3 Semi-supervised learning

As the first versions of the frailty datasets that were available were very small in size, the semi-supervised methodology was tried in order to enrich and the labeled instances with new unlabeled data. The detailed process of the task is presented in **D4.13**. In this subparagraph, as this a more technical report we will present the tools used to accomplish the task.

The first step of the procedure involved the collection and organization of the data gathered in D4.7 that were crawled from twitter. The offline-parser python script was accordingly extended to support the construction of mixed Labeled and Unlabeled datasets. The exact implementation can be found in the annexes section (13.1).

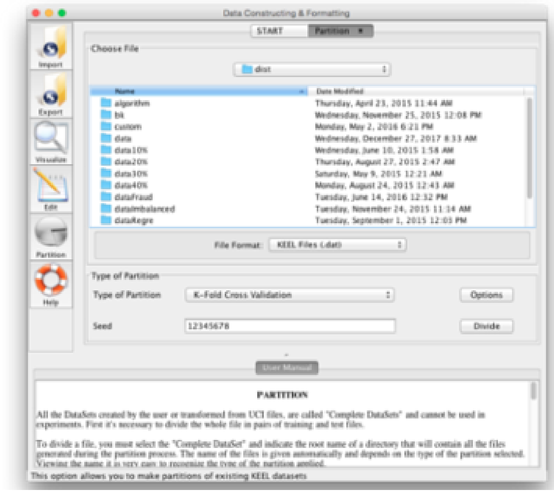
To proceed to the second step, as the WEKA tool does not support semi-supervised algorithms, we selected the research tool known as KEEL and its package named SSL for keel. All the mentioned software is open source[1].

As all our datasets and initial organization was around weka and its arff format in contrast with the standard keel format mentioned as dat, the need for development of middleware set of tools emerged. In brief the middleware programs we developed are:

- **Randomize-arff**: A php script that has as input an arff file and saves as output a new dataset with its instances in random order.
- **Weka2keel**: A java program that has as input a weka formatted dataset and produces its keel formatted equivalent.
- **Unlabelize**: A php script that reads a folder containing the keel formatted datasets, plus a user defined label ratio and creates the final keel dataset that combines labeled and unlabeled instances for use with semi-supervised algorithms.

Proceeding to the KEEL tool itself the main menu of the tools follows below in the left.

[1] http://sci2s.ugr.es/SelfLabeled



After the transformation of the weka dataset in the *keel* format, the Data Management (right image above) option of keel was used to divide the dataset in 10 Fold to be used in 10-Cross validation. The resulting datasets follow in the table below

Dataset name	Labeled ratio	Labeled instances	Unlabeled instances
semi-frailty-10	10%	178	1600
semi-frailty-20	20%	178	712
semi-frailty-30	30%	178	415
semi-frailty-40	40%	178	267

The full semi-supervised scenario experiment was built using the experiment designer (image 1). The scenario consists of:

- The four datasets, labeled ratios:
  - 10%,
  - 20%,
  - 30%,
  - 40%
- The five semi-supervised algorithms
  - SelfTraining
  - CoForest
  - CoTraining
  - TriTraining

   - RASCO
- Five calculation elements that gather and produce the final results for each algorithm

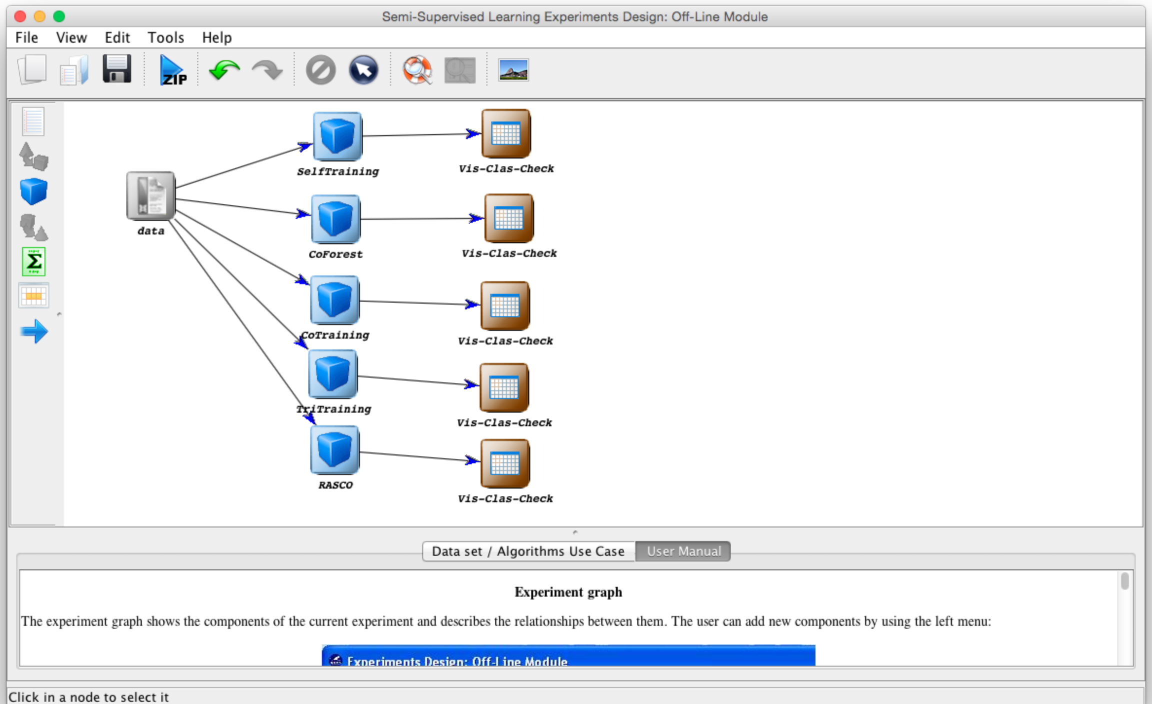


Image 1. KEEL Scenario

The execution of the scenario, as it is a relatively heavy load for a GUI program, is done through a console java program named RunKeel ([image 2](#)).

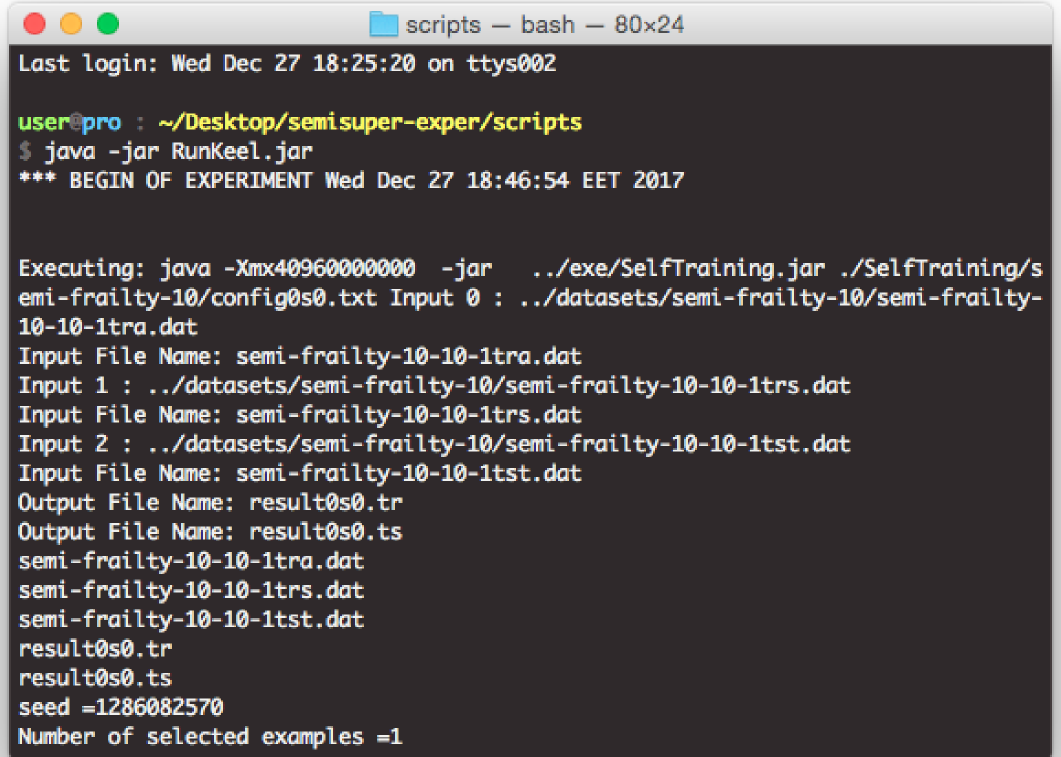


Image 2. Execution of the scenario through RunKeeL

The detailed presentation and analysis of the semi-supervised models built, are a subject of deliverable **D4.13**.

5.4 Conclusions

In this Chapter, effort has been made to present briefly and in a more technical manner, the `brain' of the offline LingTester tool. The chapter is considered complementary with chapters 4 and 5 of **D4.13**.

Concerning the final frailty prediction model, the obtained accuracy of 63.64% by the complex VoteSRLR, taking into account the difficulty of the frailty problem and its niche (in the sense of strict) medical data, is considered to be a good outcome. Under certain assumptions and simplifications as stated in **D4.13** the accuracy increased in the levels of 83.68%. Further increase in the dataset population, in the future of the FrailSafe project timeline, could result in better decision accuracies, as VoteSRLR model seems to have a strong learning capacity. The potential of the `brain' of the offline LingTester tool is strictly bound to the quality and size of the available dataset.

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[2]. 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

[2] http://nlp.cs.aueb.gr/software.html

- 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[3], 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[4]. 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.

[3] https://www.java.com/en/
[4] http://nlp.cs.aueb.gr/software.html

7.2.5 Weka

We can easily download and install WEKA by navigating to the page official[5] and following the detailed steps in that page, according to our operating system.

7.3 Uploading existing data

All aforementioned data was internally manipulated in a custom-built database consisted in text files. However, within Work Package 4 it was necessary all data provided from eCRF and extracted features were available to the DSS module. As a result, data can be automatically updated through the execution of the `updateTextToNoSQL` function in `offline-parse.py file`, found in Annex *Database management & feature extraction* of the **D4.11**.

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

[5] http://www.cs.waikato.ac.nz/ ml/weka/downloading.html

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).

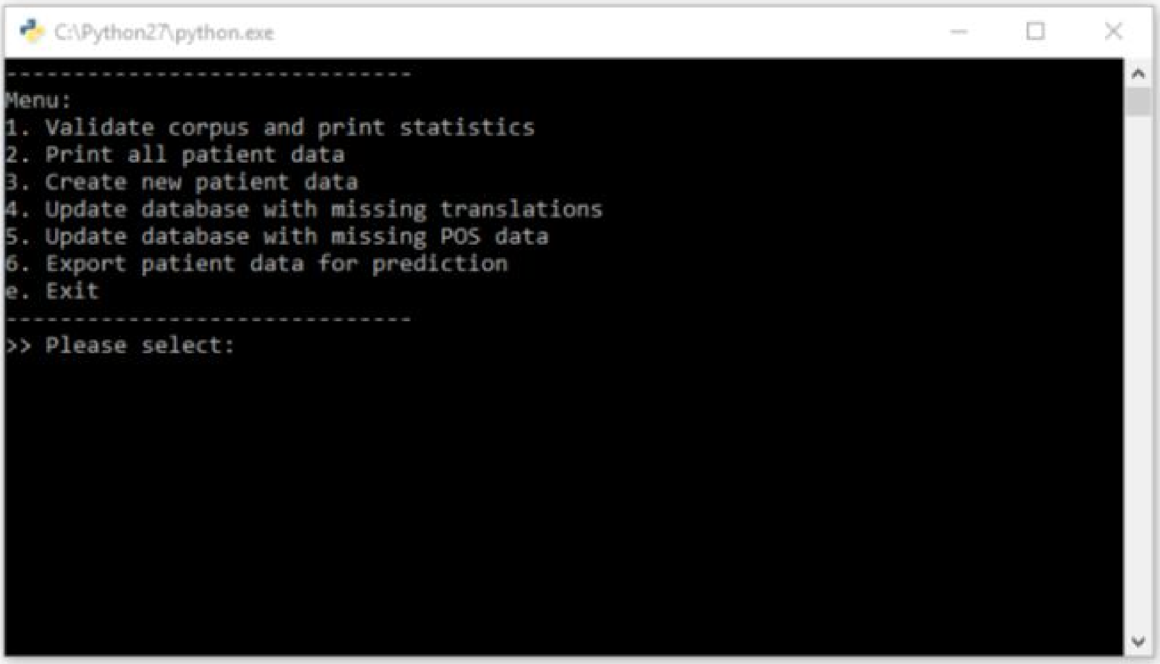


Image 3. Screenshot of real time operation

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 'fraildata'

@attribute transcript {yes,no}
@attribute language {greek,greek-polytonic,greek-cypriot,french,english}
@attribute family_status {married-or-in-a-relationship,single,divorced,widow}
@attribute text_length numeric
@attribute number_of_sentences numeric
@attribute number_of_words numeric
@attribute number_of_words_per_sentence numeric
@attribute text_entropy numeric
@attribute year_of_birth numeric
@attribute flesch_reading_ease numeric
@attribute smog_index numeric
@attribute flesch_kincaid_grade numeric
@attribute coleman_liau_index numeric
@attribute automated_readability_index numeric
@attribute dale_chall_readability_score numeric
@attribute difficult_words numeric
@attribute prev_text_misspelled numeric
@attribute tf-5 numeric
@attribute tf-8 numeric
@attribute tf-19 numeric
@attribute tf-32 numeric
@attribute tf-33 numeric
@attribute tf-37 numeric
@attribute tf-39 numeric
@attribute tf-40 numeric
@attribute tf-45 numeric
@attribute tf-46 numeric
@attribute tf-48 numeric
```

Image 4. 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.

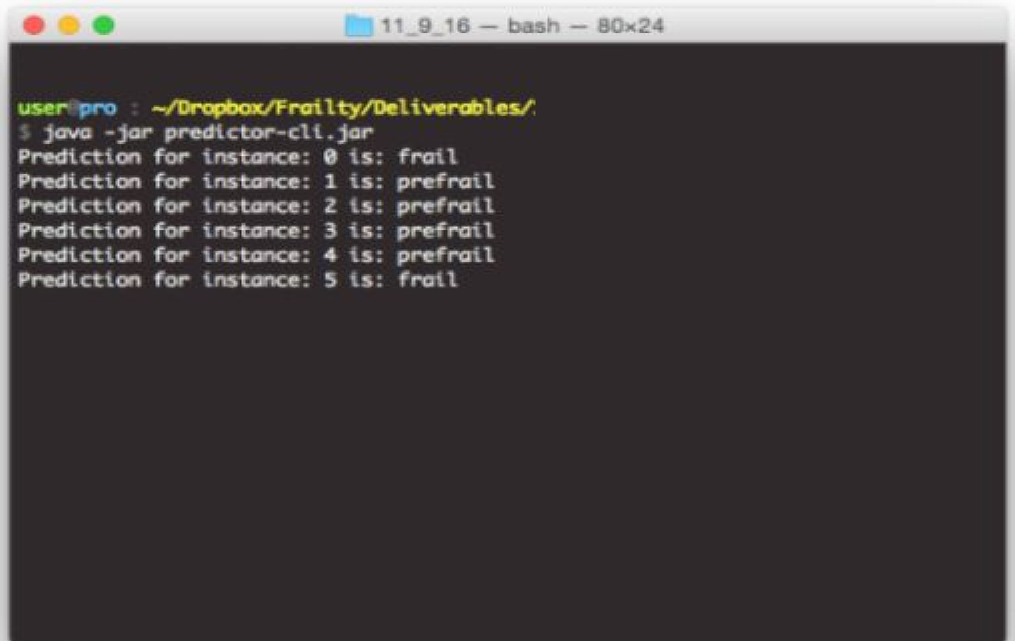


Image 5. Screenshot of prediction programme while executing

9. 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.

10. 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).

11. 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

12. Annexes

12.1 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.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# To install execute: pip install pyenchant

"""

@author: Charalampos Tsimpouris
@author: Nikolaos Fazakis

"""

import enchant

# To install execute: pip install httplib2

import httplib2
import json
import math
import urllib

# 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 re
import stemming
import shutil
import subprocess

import matplotlib.pyplot as plt
from textstat.textstat import textstat

import downloader as ecrf

# 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

nosql_api = 'http://172.16.2.50:5050'

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')
verify_tags['-sex'] = ('male', 'female')
verify_tags['-tag'] = ('nonfrail', 'prefrail', 'frail')
verify_tags['-language'] = ('greek', 'greek-polytonic',
'greek-cypriot'
                        , 'french', 'english')
verify_tags['-source'] = ('questionnaire', 'twitter-untagged')

verify_tags['-habitation_zone'] = ('urban', 'semi-urban', 'rural')
verify_tags['-family_status'] = ('married or in a relationship',
                        'single', 'divorced', 'widow')
verify_tags['-have_you_changed_your_security_settings_in_
social_media_in_order_to_protect_your_personal_data'
        ] = ('yes', 'no')
verify_tags['-do_you_consider_yourself_a_familiar_user_of_
social_media'
        ] = ('beginner', 'less familiar', 'very familiar')

# Languge convert helper dictionaries for various reasons

langs = {}
```

```
langs['english'] = 'en'
langs['greek'] = 'el'
langs['greek-polytonic'] = 'el'
langs['greek-cypriot'] = 'el-cy'
langs['french'] = 'fr'

langs_speller = {}
langs_speller['english'] = 'en'
langs_speller['greek'] = 'el_GR'
langs_speller['greek-polytonic'] = 'el_GR'
langs_speller['greek-cypriot'] = 'el_GR'
langs_speller['french'] = 'fr_FR'

# These tags are multi line
# .. 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 sortedDictValues(adict, reverse_order=True):
    """
    Taksinomisi leksikou, simfwna me to "value"
    http://wiki.python.org/moin/HowTo/Sorting/
    """

    ret = []
    for k in adict:
        ret.append((k, adict[k]))

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

    return [page[0] for page in ret]


def 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 my_sort(x, y):
    return int(x) - int(y)


# def fetch_patient_ids(only_local = False):
#     """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('.')
#
#         if only_local and (int(pid) < 1000 or int(pid) >= 4000):
#             continue
#         out_files.append(pid)
#
#     return sorted(out_files, cmp = my_sort)
#     out_files.sort()
#     return out_files

def fetch_patient_visits(only_local=False):
    """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

        parts = f.strip().split('.')
        if len(parts) == 3:
            (pre, pid, visit, suf) = (parts[0], parts[1], 1, parts[2])
        elif len(parts) == 4:
            (pre, pid, visit, suf) = (parts[0], parts[1], parts[2],
                    parts[3])
        else:
            print 'Unable to load file %s' % f
            continue

        if only_local and (int(pid) < 1000 or int(pid) >= 4000):
            continue

        if not pid in out_files:
            out_files[pid] = []

        out_files[pid].append(int(visit))

    return out_files


def export_twitter_untagged_to_files():
    t_id = 100000

    files = os.listdir(data_path)
    for f in files:
        if not f.startswith('twitter-untagged.') \
            or not f.endswith('.txt'):
            continue

        print 'Working on %s' % f
        (pre, lang, suf) = f.strip().split('.')

        fl = open(data_path + '/' + f)
        lines = fl.readlines()
        fl.close()

        for l in lines:
            if l.strip() == '':
                continue
            if l.strip().startswith('||RT'):
                continue

            t_id += 1

            # Remove the || at the end

            l = l.replace('||', '')

            ret = {}
            ret['-source'] = 'twitter-untagged'
            ret['-transcript'] = 'no'
            ret['-sex'] = 'na'
            ret['-tag'] = 'na'
            ret['-language'] = lang
            ret['-desc_event'] = l

            save_patient_data(t_id, ret)


def fetch_patient_data(cpid, visit=1):
    """Tries to load all patient data, based on the id
    """

    try:
        if visit == 1:
            f = open(data_path + '/p.%s.txt' % cpid, 'r')
        else:
            f = open(data_path + '/p.%s.%02d.txt' % (cpid, visit),
'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 == '':
            continue

        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()

    if not '-source' in ret:
        ret['-source'] = 'questionnaire'
    return ret


def save_patient_data(cpid, cpdata, visit=1):
    """Saves all patient data in a file
    """

    if int(visit) == 1:
        f = open(data_path + '/p.%s.txt' % cpid, 'w')
    else:
        f = open(data_path + '/p.%s.%02d.txt' % (cpid,
int(visit)), '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

''' % tag)
            f.write(cpdata[tag].strip())
            f.write('''

''')
            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 get_last_avail(
    vals,
    vis_a,
    tag,
    default='',
    vis_b=1,
    ):
    if vis_a in vals:
        if tag in vals[vis_a]:
            return vals[vis_a][tag]

    if vis_b in vals:
        if tag in vals[vis_b]:
            return vals[vis_b][tag]

    return default


def validate_patient_data(only_local=False):
    """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_visits(only_local)
    ecrf_tags = getECRFTags()

    stats = {}
    prev_texts = 0
    so_far = 0
    for cpid in pids:
        so_far += 1
        if so_far % 1000 == 1:
            print '.',

        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)

            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 (visit %d) is missing %s data' \
                        % (cpid, visit, t)
                    continue

                if t == 'tag' and cpdata[t] not in valid:
                    print 'Patient %s (visit %d) has invalid %s data
%s' \
                        % (cpid, visit, t, cpdata[t])
                    continue

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

            for t in ecrf_tags:
                tin = '-' + correct_title_for_arff(t)

                if not tin in cpdata or cpdata[tin] is None \
                    or cpdata[tin] == '':
                    continue

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

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

            text = ''
            for m in multi_line_tags:
                text += cpdata.get(m, ' ')

            text = text.strip()
            if text == '':
                stats['-language'][cpdata['-language'] + '-empty'] = \
                    stats['-language'].get(cpdata['-language']
                        + '-empty', 0) + 1

            if cpdata.get('-prev_text', '').strip() != '':
                prev_texts += 1

            break

    print 'Checked %d patients, and here are the statistics' %
len(pids)
    for t in stats:
        print t
        for k in stats[t]:
```

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

    print 'Number of patients that provided writen texts, from
previous times: %d' \
        % prev_texts


def clean_up_text(text, lang='english'):
    """Tries to clean up text, as much as possible
       in some ways language inndipendantly
       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',
    includeTFIDF=True,
    includePOS=True,
    only_local=False,
    TFIDF_thres=20,
    ngram_range_select=(1, 2),
    ):
    """Create arff for WEKA with all features availiable
    """

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

    out.append('%% %s: %s' % ('Differential', 'no'))
    params = locals()
    for p in params:
        if p != 'out':
            out.append('%% %s: %s' % (p, params[p]))
    out.append('')

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

    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')))

    # from eCRF

    ecrf = []
    ecrf.append('get_ecrf_year_of_birth')

ecrf.append('get_ecrf_how_often_do_you_connect_to_the_i
nternet_per_week'
            )

ecrf.append('get_ecrf_how_many_people_do_you_follow_on
_twitter')

ecrf.append('get_ecrf_how_many_follower_you_have_on_tw
itter')

ecrf.append('get_ecrf_how_many_friends_contacts_do_you
_have_on_facebook'
            )

    for attr in ecrf:
        out.append('@ATTRIBUTE %s %s' % (globals()[attr]('',
'title'),
                globals()[attr]('', 'type')))

    read_scores = []
    read_scores.append('flesch_reading_ease')
    read_scores.append('smog_index')
    read_scores.append('flesch_kincaid_grade')
    read_scores.append('coleman_liau_index')
    read_scores.append('automated_readability_index')
    read_scores.append('dale_chall_readability_score')
    read_scores.append('difficult_words')
    read_scores.append('linsear_write_formula')
    read_scores.append('gunning_fog')
    for attr in read_scores:
```

```
    out.append('@ATTRIBUTE %s %s' % (attr, 'real'))

  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_visits(only_local)
  filename = 'ARFFS/%s.arff' % relation

  labels = []
  for cpid in corpus:
    labels.append(cpid)

  text_POS = []
  if includePOS:
    for cpid in corpus:
      for visit in [3, 2, 1]:
        if not visit in corpus[cpid]:
          continue

        if get_last_avail(corpus[cpid], visit, 'tag', 'na') \
          == 'na':
          continue

        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('-',
                ''))

        # Only first avail visit

        break

    # 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()

    fp = open(filename + '.tf_POS.pickle', 'w')
    pickle.dump({'tf_POS': tf_POS,
          'tfidf_matrix_POS': tfidf_matrix_POS,
          'feature_names_POS': feature_names_POS},
fp)
    fp.close()

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

    # The following creates an "array to big" error
    # dense_POS = tfidf_matrix_POS.todense()

  texts = []
  if includeTFIDF:
    if only_local:
      texts_per_group = {'nonfrail': [], 'prefrail': [],
              'frail': []}
      for cpid in corpus:
        for visit in [3, 2, 1]:
          if not visit in corpus[cpid]:
            continue

          mytag = get_last_avail(corpus[cpid], visit, 'tag',
            'na')
          if mytag == 'na':
            continue

          # Per language
          # texts_per_group[ mytag
].append(corpus[cpid][visit]['clean_text'])

          # In english

texts_per_group[mytag].append(corpus[cpid][visit]['text_ENG
'
            ])
          break
    else:
```

```
        texts_per_group = {
            'nonfrail': [],
            'prefrail': [],
            'frail': [],
            'na': [],
            }
        for cpid in corpus:
            mytag = get_last_avail(corpus[cpid], visit, 'tag',
'na')

texts_per_group[mytag].append(corpus[cpid]['clean_text'
                ])

    (s, score_words) = compute_ig(texts_per_group,
filename
            + '.ig.png')
    mywords = s[:TFIDF_thres]

    for cpid in corpus:
        for visit in [3, 2, 1]:
            if not visit in corpus[cpid]:
                continue

            temptext = corpus[cpid][visit]['clean_text']
            temptext_out = []
            for word in temptext.split(' '):
                if word != '' and word in mywords:
                    temptext_out.append(word)

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

            break

    # TF-IDF on stemmed text

    tf = TfidfVectorizer(analyzer='word',
                    ngram_range=ngram_range_select,
min_df=0)
    tfidf_matrix = tf.fit_transform(texts)
    feature_names = tf.get_feature_names()

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

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

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

  # Panta teleftaio
        tag = 'class'
        valid = ('nonfrail', 'prefrail', 'frail')
        out.append('@ATTRIBUTE %s {%s}' % (tag, ','.join(valid)))

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

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

        rows_so_far = len(labels) / 10
        for i in range(len(labels)):
            rows_so_far -= 1
            if rows_so_far <= 0:
                print '.',
                rows_so_far = len(labels) / 10

            cpid = labels[i]
            for visit in [3, 2, 1]:
                if not visit in corpus[cpid]:
                    continue

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

                # To absorb all Greek variations

                if clang.startswith('greek'):
                    clang = 'greek'

                # Patients with inknown frailty tag are automatically
removed

                if not '-tag' in corpus[cpid][visit]['data']:
                    print '[skipping %s]' % cpid,
                    continue

                nosql_data = {}
                nosql_data_tag = '%s_%d_' % (relation, visit)
                nosql_data[nosql_data_tag + 'lang'] = clang
                nosql_data[nosql_data_tag + 'text'] = \
                    corpus[cpid][visit]['text']
                nosql_data[nosql_data_tag + 'text_eng'] = \
                    corpus[cpid][visit]['text_ENG']

                row = []
                for bt in basic_tags:
                    if bt == '-tag':
                        continue

                    if corpus[cpid][visit]['data'].get(bt, 'na') == 'na':
                        res = '?'
                    else:
                        res = corpus[cpid][visit]['data'].get(bt, '?'
                                ).replace(' ', '-')

                    row.append(res)
                    nosql_data[nosql_data_tag + bt] = res
```

```
        for attr in other_attributes:
            res = str(globals()[attr](corpus[cpid][visit]['text'],
                    lang=clang))
            if res == 'na':
                res = '?'

            nosql_data[nosql_data_tag + attr] = res
            row.append(res)

        for attr in ecrf:
            res = str(globals()[attr](corpus[cpid][visit]['data'],
                    lang=clang))
            if res == 'na':
                res = '?'

            nosql_data[nosql_data_tag + attr] = res
            row.append(res)

        for attr in read_scores:
            call = getattr(textstat, attr)
            if corpus[cpid][visit]['text'] == '':
                res = '?'
            else:
                res = str(call(corpus[cpid][visit]['text']))

            nosql_data[nosql_data_tag + attr] = res
            row.append(res)

        # Sentiment score is based in the english translation

        for tag in multi_line_tags_ENG:
            res = str(corpus[cpid][visit]['data'].get(tag
                    + '_SENTIMENT_SCORE', '0'))
            nosql_data[nosql_data_tag + tag +
'_SENTIMENT_SCORE'] = \
                res
            row.append(res)

        for tag in multi_line_tags:
            res = str(corpus[cpid][visit]['data'].get(tag
                    + '_MISSPELLING_SCORE', '0'))
            nosql_data[nosql_data_tag + tag +
'_SENTIMENT_SCORE'] = \
                res
            row.append(res)

        # tf-idf info based on POS data

        if includePOS:

            # p_POS = dense_POS[i].tolist()[0]

            p_POS = tfidf_matrix_POS[i, :].toarray()[0]
            for fi in range(len(feature_names_POS)):
                res = '%.3f' % p_POS[fi]
                nosql_data[nosql_data_tag + '_pos_'
                        + feature_names_POS[fi]] = res
                row.append(res)

        if includeTFIDF:

            # tf-idf based on stemmed data
            # p = dense[i].tolist()[0]

            p = tfidf_matrix[i, :].toarray()[0]
            for fi in range(len(feature_names)):
                res = '%.3f' % p[fi]
                nosql_data[nosql_data_tag + '_tf_'
                        + feature_names[fi]] = res
                row.append(res)

        bt = '-tag'
        if corpus[cpid][visit]['data'].get(bt, 'na') == 'na':
            res = '?'
        else:
            res = corpus[cpid][visit]['data'].get(bt, '?')

        nosql_data[nosql_data_tag + '_tag'] = res
        row.append(res)

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

        updateTextToNoSQL(cpid, visit, nosql_data)

    f.close()
    print '..Done'


def create_arff_diferential(relation='fraildata'):
    """Create arff for WEKA with all features available
       with diferential features from old texts
    """

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

    out.append('%% %s: %s' % ('Differential', 'yes'))
    params = locals()
    for p in params:
        if p != 'out':
            out.append('%% %s: %s' % (p, params[p]))
    out.append('')

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

    other_attributes = []
    other_attributes.append('get_feature_length_diff')

other_attributes.append('get_feature_number_of_sentences
_diff')
    other_attributes.append('get_feature_word_count_diff')
```

```
other_attributes.append('get_feature_words_per_sentence_
diff')

other_attributes.append('get_feature_text_shannon_entropy
_diff')

    for attr in other_attributes:
        out.append('@ATTRIBUTE %s %s' % (globals()[attr]('',
'', 'title'
                ), globals()[attr]('', '', 'type')))

    # from eCRF

    ecrf = []
    ecrf.append('get_ecrf_year_of_birth')

ecrf.append('get_ecrf_how_often_do_you_connect_to_the_i
nternet_per_week'
                )

ecrf.append('get_ecrf_how_many_people_do_you_follow_on
_twitter')

ecrf.append('get_ecrf_how_many_follower_you_have_on_tw
itter')

ecrf.append('get_ecrf_how_many_friends_contacts_do_you
_have_on_facebook'
                )

    for attr in ecrf:
        out.append('@ATTRIBUTE %s %s' % (globals()[attr]('',
'title'),
                globals()[attr]('', 'type')))

    read_scores = []
    read_scores.append('flesch_reading_ease')
    read_scores.append('smog_index')
    read_scores.append('flesch_kincaid_grade')
    read_scores.append('coleman_liau_index')
    read_scores.append('automated_readability_index')
    read_scores.append('dale_chall_readability_score')
    read_scores.append('difficult_words')
    read_scores.append('linsear_write_formula')
    read_scores.append('gunning_fog')
    for attr in read_scores:
        out.append('@ATTRIBUTE %s_diff %s' % (attr, 'real'))

    out.append('@ATTRIBUTE %s %s' % ('sentiment_diff',
'real'))
    out.append('@ATTRIBUTE %s %s' % ('misspelled_diff',
'real'))

    corpus = get_corpus_visits(True)
    filename = 'ARFFS/%s.arff' % relation

    labels = []
    for cpid in corpus:
        labels.append(cpid)

    # Panta teleftaio

    tag = 'frail_past'
    valid = ('nonfrail', 'prefrail', 'frail')
    out.append('@ATTRIBUTE %s {%s}' % (tag, ','.join(valid)))

    tag = 'class'
    valid = ('nonfrail', 'prefrail', 'frail')
    out.append('@ATTRIBUTE %s {%s}' % (tag, ','.join(valid)))

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

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

    rows_so_far = len(labels) / 10
    for i in range(len(labels)):
        rows_so_far -= 1
        if rows_so_far <= 0:
            print '.',
            rows_so_far = len(labels) / 10

        cpid = labels[i]

        text_cur = ''
        text_prev = ''
        clang = ''
        visit_cur = -1
        visit_prev = -1

        sent_cur = 0.0
        sent_prev = 0.0

        miss_cur = 0.0
        miss_prev = 0.0

        for visit in [3, 2, 1]:
            if not visit in corpus[cpid]:
                continue

            if clang != '':
                clang = corpus[cpid][visit]['data']['-language']

                # To absorb all Greek variations

                if clang.startswith('greek'):
                    clang = 'greek'

            # Patients with inknown frailty tag are automatically
removed

            mytag = get_last_avail(corpus[cpid], visit, 'tag', 'na')
            if mytag == 'na':
                print '[skipping %s/%d]' % (cpid, visit),
                continue

            if corpus[cpid][visit]['text'].strip() == '':
                continue
```

```
        # exoume keimeno

        if text_cur == '':
            text_cur = corpus[cpid][visit]['text']
            visit_cur = visit

            # Sentiment score is based in the english
translation

            for tag in multi_line_tags_ENG:
                if tag.find('prev_text') >= 0:
                    continue
                else:
                    sent_cur += float(corpus[cpid][visit]['data'
                        ].get(tag + '_SENTIMENT_SCORE', '0'))

            for tag in multi_line_tags:
                if tag.find('prev_text') >= 0:
                    continue
                else:
                    sent_cur += float(corpus[cpid][visit]['data'
                        ].get(tag + '_MISSPELLING_SCORE',
'0'))

            continue

        # Sentiment score is based in the english translation

        for tag in multi_line_tags_ENG:
            if tag.find('prev_text') >= 0:
                sent_prev += float(corpus[cpid][visit]['data'
                    ].get(tag + '_SENTIMENT_SCORE', '0'))
            else:
                continue

            for tag in multi_line_tags:
                if tag.find('prev_text') >= 0:
                    sent_prev += float(corpus[cpid][visit]['data'
                        ].get(tag + '_MISSPELLING_SCORE',
'0'))
                else:
                    continue
        text_prev = corpus[cpid][visit]['text']
        visit_prev = visit
        break

    if visit_cur == visit_prev or visit_cur < 0 or visit_prev <
0:
        continue

    row = []

    # Ta basic tags den allazoun pote

    for bt in basic_tags:
        if bt == '-tag':
            continue

        if corpus[cpid][1]['data'].get(bt, 'na') == 'na':
            row.append('?')
        else:
            row.append(corpus[cpid][1]['data'].get(bt, '?'
                ).replace(' ', '-'))

    for attr in other_attributes:
        res = str(globals()[attr](text_cur, text_prev,
lang=clang))
        if res == 'na':
            res = '?'
        row.append(res)

    for attr in ecrf:
        res = str(globals()[attr](corpus[cpid][1]['data'],
            lang=clang))
        if res == 'na':
            res = '?'
        row.append(res)

    for attr in read_scores:
        call = getattr(textstat, attr)
        if text_cur == '' or text_prev == '':
            row.append('?')
            continue

        res1 = call(text_cur)
        res2 = call(text_prev)
        row.append(str(res2 - res1))

    row.append(str(sent_cur - sent_prev))
    row.append(str(miss_cur - miss_prev))

    # Prev tag

    bt = '-tag'
    if corpus[cpid][visit_prev]['data'].get(bt, 'na') == 'na':
        row.append('?')
    else:
        row.append(corpus[cpid][visit_prev]['data'].get(bt, '?'))

    # Cur tag

    bt = '-tag'
    if corpus[cpid][visit_cur]['data'].get(bt, 'na') == 'na':
        row.append('?')
    else:
        row.append(corpus[cpid][visit_cur]['data'].get(bt, '?'))

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

  f.close()
  print '..Done'


# def get_corpus(only_local = False):

def get_corpus_visits(only_local=False):
  """Returns all corpus in a more scientific-friendly way
  """

  pids = fetch_patient_visits(only_local)
```

```
corpus = {}

for cpid in pids:
    corpus[cpid] = {}
    for visit in pids[cpid]:
        corpus[cpid][visit] = {}
        cpdata = fetch_patient_data(cpid, visit)

        corpus[cpid][visit]['data'] = cpdata
        corpus[cpid][visit]['tag'] = cpdata.get('-tag', 'na')

        corpus[cpid][visit]['text'] = ''
        corpus[cpid][visit]['text_prev'] = ''
        corpus[cpid][visit]['text_cur'] = ''
        for m in multi_line_tags:
            corpus[cpid][visit]['text'] += cpdata.get(m, ' ')
            if m.find('prev_text') >= 0:
                corpus[cpid][visit]['text_prev'] += cpdata.get(m,
                    ' ')
            else:
                corpus[cpid][visit]['text_cur'] += cpdata.get(m, ' '
                    )
        corpus[cpid][visit]['text'] = corpus[cpid][visit]['text'
            ].strip()
        corpus[cpid][visit]['text_prev'] = \
            corpus[cpid][visit]['text_prev'].strip()
        corpus[cpid][visit]['text_cur'] = \
            corpus[cpid][visit]['text_cur'].strip()

        corpus[cpid][visit]['text_POS'] = ''
        corpus[cpid][visit]['text_prev_POS'] = ''
        corpus[cpid][visit]['text_cur_POS'] = ''
        for m in multi_line_tags_POS:
            corpus[cpid][visit]['text_POS'] += '\n' +
cpdata.get(m,
                    ' ')
            if m.find('prev_text') >= 0:
                corpus[cpid][visit]['text_prev_POS'] += '\n' \
                    + cpdata.get(m, ' ')
            else:
                corpus[cpid][visit]['text_cur_POS'] += '\n' \
                    + cpdata.get(m, ' ')

        corpus[cpid][visit]['text_ENG'] = ''
        corpus[cpid][visit]['text_prev_ENG'] = ''
        corpus[cpid][visit]['text_cur_ENG'] = ''
        for m in multi_line_tags_ENG:
            corpus[cpid][visit]['text_ENG'] += '\n' +
cpdata.get(m,
                    ' ')
            if m.find('prev_text') >= 0:
                corpus[cpid][visit]['text_prev_ENG'] += '\n' \
                    + cpdata.get(m, ' ')
            else:
                corpus[cpid][visit]['text_cur_ENG'] += '\n' \
                    + cpdata.get(m, ' ')

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

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

        tutf8 = corpus[cpid][visit]['text'].decode('utf-8')
        corpus[cpid][visit]['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_feature_length_diff(
    text1,
    text2,
    meta=None,
    lang='english',
    ):
    if meta == 'title':
        return 'text_length_diff'
    if meta == 'type':
        return 'integer'

    return len(text1) - len(text2)


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_feature_number_of_sentences_diff(
    text1,
    text2,
    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_diff'
    if meta == 'type':
        return 'integer'

    return len(get_sentences(text1, lang)) -
len(get_sentences(text2,
        lang))


def get_ecrf_year_of_birth(cpdata, meta=False,
lang='english'):
    """Returns a feature with number of sentences
    TODO: This has to be more elaborate
    """

    if meta == 'title':
        return 'year_of_birth'
    if meta == 'type':
        return 'integer'

    return cpdata.get('-year_birth', '?')


def
get_ecrf_how_often_do_you_connect_to_the_internet_p
er_week(cpdata,
        meta=False, lang='english'):
    """Returns a feature with number of sentences
    TODO: This has to be more elaborate
    """

    if meta == 'title':
        return 'con_per_week'
    if meta == 'type':
        return 'integer'

    return
cpdata.get('-how_often_do_you_connect_to_the_internet_pe
r_week'
                    , '?')


def
get_ecrf_how_many_people_do_you_follow_on_twitter(
cpdata,
        meta=False, lang='english'):
    """Returns a feature with number of sentences
    TODO: This has to be more elaborate
    """
```

```
    if meta == 'title':
        return 'twitter_follows'
    if meta == 'type':
        return 'integer'

    return
cpdata.get('-how_many_people_do_you_follow_on_twitter',
'?')


def
get_ecrf_how_many_follower_you_have_on_twitter(cpda
ta, meta=False,
        lang='english'):
    """Returns a feature with number of sentences
    TODO: This has to be more elaborate
    """

    if meta == 'title':
        return 'twitter_followers'
    if meta == 'type':
        return 'integer'

    return
cpdata.get('-how_many_follower_you_have_on_twitter', '?')


def
get_ecrf_how_many_friends_contacts_do_you_have_on
_facebook(cpdata,
        meta=False, lang='english'):
    """Returns a feature with number of sentences
    TODO: This has to be more elaborate
    """

    if meta == 'title':
        return 'fb_friends'
    if meta == 'type':
        return 'integer'

    return
cpdata.get('-how_many_friends_contacts_do_you_have_on_
facebook'
                    , '?')


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_word_count_diff(
    text1,
    text2,
    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_diff'
    if meta == 'type':
        return 'integer'

    return len(get_words(text1, lang)) - len(get_words(text2,
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_words_per_sentence_diff(
    text1,
    text2,
    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_diff'
    if meta == 'type':
        return 'real'

    res1 = 0
    if get_feature_number_of_sentences(text1) > 0:
        res1 = float(get_feature_word_count(text1)) \
            / get_feature_number_of_sentences(text1)

    res2 = 0
    if get_feature_number_of_sentences(text2) > 0:
        res2 = float(get_feature_word_count(text2)) \
            / get_feature_number_of_sentences(text2)
    return '%.3f' % (res1 - res2)


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_text_shannon_entropy_diff(
    text1,
    text2,
    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_diff'
    if meta == 'type':
        return 'real'

    mmap1 = {}
    for c in text1:
        mmap1[c] = mmap1.get(c, 0) + 1

    text_len1 = get_feature_length(text1)
```

```
    result1 = 0.0

    for c in mmap1:
        freq = mmap1[c] / float(text_len1)
        result1 -= freq * (math.log(freq) / math.log(2))

    mmap2 = {}
    for c in text2:
        mmap2[c] = mmap2.get(c, 0) + 1

    text_len2 = get_feature_length(text2)
    result2 = 0.0

    for c in mmap2:
        freq = mmap2[c] / float(text_len2)
        result2 -= freq * (math.log(freq) / math.log(2))

    return '%.3f' % (result1 - result2)


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_sentiment_score_diff(
    text1,
    text2,
    meta=False,
    lang='english',
    ):
    """Returns sentiment score, works based on the english
translation
    """

    if meta == 'title':
        return 'sentiment_score_diff'
    if meta == 'type':
        return 'real'

    v1 = 0
    for w in text1.split(' '):
        w = w.strip(',.!?)(#:;"\'').lower()
        if w in sentiment:
            v1 = v1 + sentiment[w][0] - sentiment[w][1]

    v2 = 0
    for w in text2.split(' '):
        w = w.strip(',.!?)(#:;"\'').lower()
        if w in sentiment:
            v2 = v2 + sentiment[w][0] - sentiment[w][1]

    return str(v1 - v2)


def get_feature_mispelling_score(text, meta=False,
lang='english'):
    """Returns mispelling statistics
    """

    if meta == 'title':
        return 'mispelling_score'
    if meta == 'type':
        return 'real'

    # To absorb all Greek variations

    if lang.startswith('greek'):
        lang = 'greek'

    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 get_feature_mispelling_score_diff(
    text1,
    text2,
    meta=False,
    lang='english',
    ):
    """Returns mispelling statistics
    """

    if meta == 'title':
        return 'mispelling_score'
    if meta == 'type':
        return 'real'

    mis1 = float(get_feature_mispelling_score(text1))
    mis2 = float(get_feature_mispelling_score(text2))
```

```
    return '%.3f' % (mis1 - mis2)


def clean_greek_letters(text):
    text = text.replace(u'Α', u'α')
    text = text.replace(u'Β', u'β')
    text = text.replace(u'Γ', u'γ')
    text = text.replace(u'Δ', u'δ')
    text = text.replace(u'Ε', u'ε')
    text = text.replace(u'Ζ', u'ζ')
    text = text.replace(u'Η', u'η')
    text = text.replace(u'Θ', u'θ')
    text = text.replace(u'Ι', u'ι')
    text = text.replace(u'Κ', u'κ')
    text = text.replace(u'Λ', u'λ')
    text = text.replace(u'Μ', u'μ')
    text = text.replace(u'Ν', u'ν')
    text = text.replace(u'Ξ', u'ξ')
    text = text.replace(u'Ο', u'ο')
    text = text.replace(u'Π', u'π')
    text = text.replace(u'Ρ', u'ρ')
    text = text.replace(u'Σ', u'σ')
    text = text.replace(u'ς', u'σ')
    text = text.replace(u'Τ', u'τ')
    text = text.replace(u'Υ', u'υ')
    text = text.replace(u'Φ', u'φ')
    text = text.replace(u'Χ', u'χ')
    text = text.replace(u'Ψ', u'ψ')
    text = text.replace(u'Ω', u'ω')
    text = text.replace(u'Ά', u'α')
    text = text.replace(u'Έ', u'ε')
    text = text.replace(u'Ή', u'η')
    text = text.replace(u'Ί', u'ι')
    text = text.replace(u'Ϊ', u'ι')
    text = text.replace(u'Ό', u'ο')
    text = text.replace(u'Ύ', u'υ')
    text = text.replace(u'Ϋ', u'υ')
    text = text.replace(u'Ώ', u'ω')

    text = text.replace(u'ά', u'α')
    text = text.replace(u'έ', u'ε')
    text = text.replace(u'ύ', u'υ')
    text = text.replace(u'ί', u'ι')
    text = text.replace(u'ό', u'ο')
    text = text.replace(u'ή', u'η')
    text = text.replace(u'ώ', u'ω')
    text = text.replace(u'ϊ', u'ι')
    text = text.replace(u'ϋ', u'υ')
    text = text.replace(u'ΐ', u'ι')
    text = text.replace(u'ΰ', u'υ')

    text = text + u' '

    # text = text.replace(u'ν ', u' ') # p.x. to "ενοριαν" ginetai
"ενορια"

    text = text.replace(u'σ ', u'ς ')  #
    text = text.strip()

    return text


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

    return text


def upper(text):
    """Capitilizes text"""

    text = text.replace(u'α', u'Α')
    text = text.replace(u'β', u'Β')
    text = text.replace(u'γ', u'Γ')
    text = text.replace(u'δ', u'Δ')
    text = text.replace(u'ε', u'Ε')
    text = text.replace(u'ζ', u'Ζ')
    text = text.replace(u'η', u'Η')
    text = text.replace(u'θ', u'Θ')
    text = text.replace(u'ι', u'Ι')
    text = text.replace(u'κ', u'Κ')
    text = text.replace(u'λ', u'Λ')
    text = text.replace(u'μ', u'Μ')
    text = text.replace(u'ν', u'Ν')
    text = text.replace(u'ξ', u'Ξ')
    text = text.replace(u'ο', u'Ο')
    text = text.replace(u'π', u'Π')
    text = text.replace(u'ρ', u'Ρ')
    text = text.replace(u'σ', u'Σ')
    text = text.replace(u'ς', u'Σ')
    text = text.replace(u'τ', u'Τ')
    text = text.replace(u'υ', u'Υ')
    text = text.replace(u'φ', u'Φ')
    text = text.replace(u'χ', u'Χ')
    text = text.replace(u'ψ', u'Ψ')
    text = text.replace(u'ω', u'Ω')

    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)))
        try:
            result = json.loads(json_content)
        except:
            print 'Error decoding json from mymemory, aborting..
Debug: %s' \
                % debug
            return ''

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

        trans_result += result['responseData']['translatedText'] +
'\n'

    # -------------------------------------------------------

    return trans_result.encode('utf-8')


def
update_corpus_with_sentiment_scores(force_rebuild=Fal
```

```
se):
    """Get all text data from all patients
       and updates the corpus with missing entiment analysos

       The force_rebuild parameter, will force to update all
translations
    """

    pids = fetch_patient_visits()

    for cpid in pids:
        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)

            updated = False
            for mt in multi_line_tags_ENG:
                d = cpdata.get(mt, '')

                # Adeio keimeno

                if d == '' and not force_rebuild:
                    continue

                # Exw idi ipologisei POS data

                if cpdata.get(mt + '_SENTIMENT_SCORE', '') != ''
\
                    and not force_rebuild:
                    continue

                # Den to exoume, as to paroume

                ret = get_feature_sentiment_score(d)
                if ret == '' and not force_rebuild:
                    continue

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

            # Something changed, time to store it

            if updated:
                print 'Updating patient %s, visit %d' % (cpid, visit),
                save_patient_data(cpid, cpdata, visit)
                print '..Done'


def
update_corpus_with_misspelling_scores(force_rebuild=F
alse):
    """Get all text data from all patients
       and updates the corpus with missing entiment analysos

       The force_rebuild parameter, will force to update all
translations
    """

    pids = fetch_patient_visits()

    for cpid in pids:
        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)
            updated = False
            clang = cpdata['-language']

            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 + '_MISSPELLING_SCORE', '') !=
'' \
                    and not force_rebuild:
                    continue

                # Den to exoume, as to paroume

                ret = get_feature_mispelling_score(d, lang=clang)
                if ret == '' and not force_rebuild:
                    continue

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

            # Something changed, time to store it

            if updated:
                print 'Updating patient %s, visit %d' % (cpid, visit),
                save_patient_data(cpid, cpdata, visit)
                print '..Done'


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_visits()

    for cpid in pids:
        cpdata = fetch_patient_data(cpid)
        def_lang = cpdata.get('-language', '')

        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)
            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

            if not '-language' in cpdata:
                cpdata['-language'] = def_lang

            if cpdata.get('-language', '') == '':
                print 'Missing lanfuage in patient %s, visit %d' \
                    % (cpid, visit)
                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, visit %d' % (cpid, visit),
            save_patient_data(cpid, cpdata, visit)
            print '..Done'


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_visits()

    for cpid in pids:
        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)
            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, visit %d' % (cpid, visit),
                save_patient_data(cpid, cpdata, visit)
                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 getECRFTags():
    responses = []
    responses.append('habitation zone')
    responses.append('how many people do you follow on
twitter?')
    responses.append('family status')
    responses.append('how many friends/contacts do you
have on facebook?'
                )
    responses.append('do you consider yourself a familiar
user of social media?'
                )
    responses.append('which of  below social media you use?
[facebook]')
    responses.append('how often do you connect to the
internet per week?'
                )
    responses.append('have you changed your security
settings in social media in order to protect your personal
data?'
                )
```

```
    responses.append('how many follower you have on
twitter?')

    return responses


def updateFromECRF(showMissing=False):
    p = ecrf.get_latest_data_export()

    # Only local patients

    pids = fetch_patient_visits(True)

    # Update frailty tag

    for cpid in pids:
        if not cpid in p:
            if showMissing:
                print 'ID %s is missing from eCRF' % cpid
            continue

        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)
            existing_frailty_status = cpdata.get('-tag', '')

            visit = str(visit)
            if not visit in p[cpid]['responses']:
                continue

            # So nice to have such short titles

            frailty_system = p[cpid]['responses'
                ][visit].get("{if(sum( q851161, q977341,
q833301, q696310, q689142)=='5',\\non frail\\,(if(sum(
q851161, q977341, q833301, q696310,
q689142)>'7',\\frail\\,(if(sum( q851161, q977341,q833301,
q696310, q689142)>'5'and sum( q851161,
q977341,q833301, q696310, q689142)"
                                , '').strip()
            frailty_doctor = p[cpid]['responses'
                ][visit].get("fried's categorization according to
clinician's estimation"
                                , '').strip()

            if frailty_doctor == frailty_system and frailty_doctor \
                == '':
                continue

            # Doctor beats system?

            if frailty_doctor != '':
                frailty_system = frailty_doctor

            if frailty_system in ('nonfrail', 'Non frail', 'Non-fragile'
                                , 'Non frail'):
                frailty_system = 'nonfrail'
            elif frailty_system in ('prefrail', 'Pré-fragile',
                                    'Pre-frail', 'Pre-frail'):
                frailty_system = 'prefrail'
            elif frailty_system in ('frail', 'Fragile', 'Frail'):
                frailty_system = 'frail'
            else:
                print "Unable to identify frailty status '%s' for
patient id %s" \
                    % (frailty_system, cpid)
                continue

            if existing_frailty_status == frailty_system:
                continue

            cpdata['-tag'] = frailty_system
            save_patient_data(cpid, cpdata, visit)
            print 'Frailty status: Patient ID %s, visit %s has been
auto updated from %s to %s' \
                % (cpid, visit, existing_frailty_status,
frailty_system)

    # Update gender tag

    for cpid in pids:
        if not cpid in p:
            continue

        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)

            existing_gender = cpdata.get('-sex', '')
            gender_system = p[cpid].get('gender',
'').strip().lower()
            if gender_system != '':
                if gender_system in ('m', 'male'):
                    gender_system = 'male'
                elif gender_system in ('f', 'female'):
                    gender_system = 'female'
                else:
                    print "Unable to identify gender status '%s' for
patient id %s" \
                        % (gender_system, cpid)
                    continue

                if existing_gender != gender_system:
                    cpdata['-sex'] = gender_system
                    save_patient_data(cpid, cpdata, visit)
                    print 'Gender: Patient ID %s has been auto
updated from %s to %s' \
                        % (cpid, existing_gender, gender_system)

            existing_year_birth = cpdata.get('-year_birth', '')
            year_birth_system = p[cpid].get('year_birth', ''
                ).strip().lower()
            if year_birth_system != '':
                if existing_year_birth != year_birth_system:
                    cpdata['-year_birth'] = year_birth_system
                    save_patient_data(cpid, cpdata, visit)
                    print 'Year of birth: Patient ID %s has been auto
updated from %s to %s' \
                        % (cpid, existing_year_birth,
year_birth_system)

            existing_profession = cpdata.get('-profession', '')
            profession_system = p[cpid].get('profession', ''
                ).strip().lower()
```

```
        if profession_system != '':
            if existing_profession != profession_system:
                cpdata['-profession'] = profession_system
                save_patient_data(cpid, cpdata, visit)
                print 'Profession: Patient ID %s has been auto
updated from %s to %s' \
                    % (cpid, existing_profession,
profession_system)

        existing_group = cpdata.get('-group', '')
        group_system = p[cpid].get('gorup', '').strip().lower()
        if group_system != '':
            if existing_group != group_system:
                cpdata['-group'] = group_system
                save_patient_data(cpid, cpdata, visit)
                print 'Profession: Patient ID %s has been auto
updated from %s to %s' \
                    % (cpid, existing_group, group_system)

    responses = getECRFTags()

    for r in responses:
        internal_value = correct_title_for_arff(r)
        print 'Checking %s (%s)' % (r, internal_value)
        for cpid in pids:
            if not cpid in p:
                continue

            for visit in pids[cpid]:
                cpdata = fetch_patient_data(cpid, visit)

                visit = str(visit)
                if not visit in p[cpid]['responses']:
                    continue

                existing_response = cpdata.get('-%s' %
internal_value,
                    '')

                system_response = p[cpid]['responses'][visit].get(r,
''
                    ).strip().lower()
                if system_response == '' or system_response in
('na',
                    'n/a'):
                    system_response = 'na'

                if existing_response != system_response:
                    cpdata['-%s' % internal_value] =
system_response
                    save_patient_data(cpid, cpdata, visit)
                    print '\tPatient ID %s, visit %d, has been auto
updated from %s to %s' \
                    % (cpid, int(visit), existing_response,
                      system_response)


def correct_title_for_arff(t):
    t = t.lower()
    return re.sub(r'[^a-z0-9]+', '_', t).rstrip('_').lstrip('_')
```

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

    pids = fetch_patient_visits()

    per_place = {}
    for cpid in pids:
        for visit in pids[cpid]:
            cpdata = fetch_patient_data(cpid, visit)
            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 exportBasicCSV(only_local=True, dec=','):
    pids = fetch_patient_visits(only_local)

    tags_to_export = []
    for k in verify_tags:
        tags_to_export.append(k)


tags_to_export.append('-how_often_do_you_connect_to_the
_internet_per_week'
                    )

tags_to_export.append('-how_many_people_do_you_follow_
on_twitter')

tags_to_export.append('-how_many_follower_you_have_on
_twitter')

tags_to_export.append('-how_many_friends_contacts_do_y
ou_have_on_facebook'
                    )
```

```
    filename = 'csv_export.csv'
    print 'Exporting to %s..' % filename,
    f = open(filename, 'w')

    f.write(dec.join(tags_to_export) + '\n')

    for cpid in pids:
        for visit in pids:
            cpdata = fetch_patient_data(cpid, visit)
            row = []
            for k in tags_to_export:
                if cpdata.get(k, 'na') == 'na':
                    row.append('')
                else:
                    row.append(cpdata[k])

            f.write(dec.join(row) + '\n')

    f.close()
    print '..Done!'


def install_spell_greek_checker_files():
    """This must be run a root
    """

    # Linux
    # sudo apt-get install myspell-gr-el

    pass


def compute_ig(texts_per_tag, historgram_name=None):
    """
    compute_ig():
        Compute information gain for each word
    """

    # With a little bit of help
    # http://streamhacker.com/tag/information-gain/

    from nltk.metrics import BigramAssocMeasures

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

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

            data = text.split(' ')
            for w in data:
                if w == '':
                    continue

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

            del data

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

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

        for c in word_count_per_class:
            score_per_word[w] += \

BigramAssocMeasures.chi_sq(word_count_per_class[c].get(
w,
                0), (freq, all_word_count_per_class[c]),
all_words)

    del word_count_per_class
    del all_word_count_per_class
    del word_count_per_word

    print 'Sorting..',
    s = sortedDictValues(score_per_word)
    print '..Done'

    # del score_per_word

    print '..Done'

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

    print 'Creating ig histogram',
    plt.figure(figsize=(24, int(24.0 * 9.0 / 16.0)))

    # plt.hist(numpy.asarray(score_per_word.values()), 5000,
facecolor = 'g')

    plt.plot(nums)
    plt.xlabel('Lexicon values')
    plt.ylabel('IG Score')
    plt.title('IG Score per lexicon lemma')
```

```
    plt.grid(True)
    if not historgram_name is None:
        plt.savefig(historgram_name)
    else:
        plt.show()
    print '..Done'

    # del s

    return (s, score_per_word)


def updateTextToNoSQL(pid, key, data):
    data_out = {}
    for k in data:
        data_out[k.replace('-', '_').replace('__', '_')] = data[k]
    headers = {'Content-Type': 'application/json'}
    body = json.dumps(data_out)
    print 'Sending patient data %s/%s to nosql..' % (str(pid),
        str(key)),
    uri = nosql_api + '/social/update_text/%s/%s' % (str(pid),
str(key))
    try:
        (resp, json_content) =
httplib2.Http(timeout=5).request(uri,
            'POST', body=body, headers=headers)
        print '..Done'
    except:
        print 'Unable to send data for patient %s and text with
key %s' \
            % (str(pid), str(key))
        print 'Data is'
        print data_out
```

12.2 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();
        }
    }

}
```