**Methodology and Implementation:**

The methodology of this project is separated into several parts

**Acquiring the dataset:**

 Acquiring the dataset was made possible by creating a data crawler using Java JSoup library using the following steps:

1. Get the establish connection to server.
2. Retrieve HTML scripts.
3. Use HTML hyperlink tags to navigate and retrieve post pages.
4. Find pages with the tags that contain the full text.
5. Get user profile pages which includes some user profile data (age and gender).
6. Find the personal data tag, split it into parts, take the age part and the gender part (which is marked by integers and the words male and female)
7. Repeat and store the data in an array list of data objects.
8. Store the dataset into CSV files that can be easily accessed later (switched to excel later because it is even more convenient).
9. Handle connection problems during the crawling process by Backing up the data.

**Subjects to consider:**

* Choosing the Drugs for this research: For the purpose of this research, drugs using for chronic diseases had to be chosen. After some consideration, drugs used for hypertension were chosen, and they are Lisinopril, Nadolol, Amlodipine, Diltiazem, Hydrochlorothiazide and Atenolol.
* Choosing the medical forums for mining: The websites chosen for this purpose are MedHelp and AskAPatient, which was chosen since most of its posting members are more committed to share their personal data, such as age and gender. The dataset acquired from both forums will be used comparing results and quality of the datasets.
* Finding the correct links and tags: JSoup establishes a connection to the server and returns the HTML script as text to a variable, and from that script tags can be chosen based on IDs or classes, given a universal search query link (example: <https://www.medhelp.org/search/expanded?cat=posts&page=2&query=Nadolol>), the web can easily be navigated through JSoup, and given the correct tags from each the given posts (example: subject\_msg), data can extracted from each page and it’s HTML script.
* Store Data: The chosen data storage is on CSV file which can be accessed using MS excel, they can also be used later using Pandas library in Python, and later saved as excel files (which proved even more convenient than CSV files).
* AksAPatient: Using crawler was neither necessary nor possible, the site was protected against crawling activities. However its data was in a table format which was much easier to simply copy and paste, a script was later made to remove all the problems in the text format. AskAPatient is more consistent comparing to MedHelp, however user data is impossible to extract other than age and gender, however all data extracted is correct and therefore could be used for comparison against MedHelp dataset.

**Data retrieval:**

Several natural language processing techniques were implemented using Python to extract the data necessary, using NLTK (Natural Language Tool Kit) library

* Tokenize data: Turn the words into separate tokens.
* Remove stop-words: Stop-words like (and, a, or) were removed to decrease the size of data.
* Stemming data: Porter stemmer was used to turn words into their roots (exhaustion, exhaustive, exhausted= exhaust), both the original and the stemmed tokens were kept into separate csv columns.

**Build Dictionary:**

A dictionary filled with concepts like ADR, Disease and Mental issues were needed to narrow down the search premise into the UMLS, using the following steps:



1. Find and store all unique tokens.
2. Identify UMLS concepts using MetaMap.
3. Parse MetaMap output into a python dictionary format.
4. Remove secondary concepts.
5. Stem the token to remove similarity.
6. Extract mentions from the Dataset using the dictionary.
7. Extract missing user profile data as features.
8. Store mentions as features and classes for classification.

**Subjects to consider:**

* Find term frequency: This in itself is not necessary for building a dictionary, but the TfidfVectorizer from Sklearn library can double as a retrieving method for all unique words in the text, term frequency will be used later, but for now all unique words are stored in two term frequency files, one for stemmed words and one without stemming.
* MetaMap batch: MetaMap as mentioned before is used to extract UMLS concepts, by sending a file to the batch system, which was necessary due to size of the dataset.
* Stemmed or Un-stemmed: The stemmed version was not effective in extraction because it removed the meaning of the words (“Acne” became “acn” which means nothing and was therefore undiscovered). Therefore the un-stemmed version was instead used for discovery, with the terms being stemmed later, which reduced the size of the dictionary by insuring that similar words are not repeated (confusion, confused = confus).
* Extract the concepts from MetaMap: MetaMap output needed is classified into three categories ([Signs and Symptoms] = ADR, [Disease or Syndrome], and [Mental or Behavioral Dysfunction]), a script was made to handle MetaMap output and extract these concepts into files containing the concept in Python Dictionary Format.
* Extract The concepts per post: The dictionary was copied and pasted into a python script (After some minimal manual revision) to be used for concept extraction from the dataset, every post had was scanned for any token that match any concept in the dictionary, and these concepts were than aligned it’s related meaning ('cramp': ['Muscle Cramp ', 'Cramping sensation quality ']) or meanings, this arrangement is made so that concepts with similar meanings don’t get repeated (Pain = Pain, Ache = Pain), secondary meanings were removed if they were similar and don’t add meaning. Age was also extracted by applying a moving window of three tokens that searches for some limited age related words couple with numbers (I am 28 years old). As well as blood pressure (150/70).

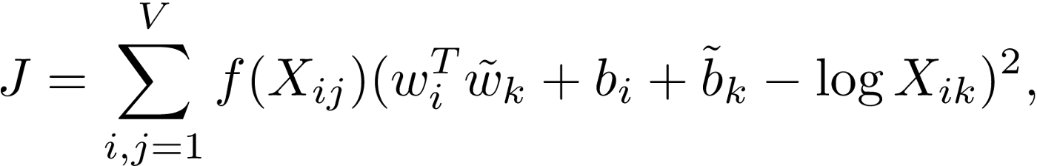
**Association with GloVe:**

**Word Embedding:** A technique for language modeling and feature learning [1], which transforms words in a document into a continuous real number [1], traditional way to find calculate it include simple term frequency, TFIDF and co-occurrence matrix [2]. Newer methods however include the use of neural networks, such methods include Word2Vec [2].

**GloVe:** Global Vectors for Word Representation [3], is a very popular library created by Stanford to be used for word embedding based on co-occurrence matrix. It is used here to make some indications which can be observed later in the machine learning phase. Word embedding shows the association between all the words in the corpus.

GloVe works as follows [4]:

* Find the co-occurrence matrix probability of each word x [3] [4].
* Calculate the context of the words together based on the distance x+= 1/distance [3] [4].
* Take the log for each value in the matrix (+1 to avoid 0 values) [3] [4].
* Weight value x in the matrix from 0 to 1 based on a function (f(x) =(x /xmax)a) [3] [4].
* Calculate the final Co-occurrence matrix using the cost function [3] [4]:



* The matrix can be calculated over many dimensions, creating many vector values that can be calculate to find the distance between two certain words using cosine or Euclidean distance [3] [4].

**Implementation Details:**

* Both stemmed and un-stemmed datasets were used to create a word vector models, the stemmed dataset turned out to be more effective as it generalized some terms instead of repeating them (confused, confusion=confus).
* The models were saved as 2D array for vectors in .txt format.
* The text file was loaded into tensorflow projector [5] [6]to represent the points of the model into an intelligible 3D (in truth 100D) plot (pictured above).
* This plot allows the observation of related concepts in a readable manner, each dot representing one of the model’s label. The closer two dots are to each other, the higher their association, and therefore the probability of co-occurrence.

Two approaches were made for the analysis:

1. Discover the relation between ADRs and Drugs.

* After the models were saved into several formats, they were trimmed to only include ADRs and Mental issues, as they are the most relevant for this analysis.
* By looking at the closest vectors to a certain drug, it was possible to find which ADRs and Mental issues have the highest chance to occur when using the drug. For example, amlodipine was found more related to hoarseness, frenzy and hallucinations than it is related to pain, nervousness and alcohol abuse. It is therefore more expected for a patient to encounter hoarseness- for example- than to encounter pain while taking the drug.

1. Discover any related concepts to the drugs and ADRs.

* The models were trimmed to include the words with a relatively high term frequency (above 20, 40, 60, or 100), in order to make it more readable by removing the least used words with little effect on the model.
* The goal is to find any randomly related concepts that might relate to the use of the drug, like for example dosage, age, a certain height or weight. It could therefore be understood from the results if there is any remote relation between the usage of this drug, or the presence of the ADR, and the presence of these other concepts.
* The dataset was slightly modified to attach several concepts to each other to become on term (200 mg = 200mg) which limits the model size and helps make it more accurate and readable.

**Association with Apriori:**

Association rule mining is extremely important as a data mining technique, it can be help for decision making. And in the context of this project, it can identify the relation between the occurrences of Drugs, Drug Family and all the ADRs, mental issues and diseases.

In association rules, the goal is to identify the following major components [7]:

* Support: Records Containing A/Total Records = the number of times one or more items appear in the records [7].
* Confidence: Confidence (A =>B) = (Records containing both (A and B)) / (Records containing A) = the likelihood that if one item appear in a record, another one also appears [7].
* Lift: Lift (A=>B) = (Confidence (A=>B)) / (Support (B)) = refers to the increase ration if item B when item A appear [7].

Apriori Algorithm is an application to the association rule mining used to decrease the run time for a large dataset [7] [8]. Generally, the standard algorithm is calculate all the rules for all the dataset. In Apriori however:

* Set a minimum requirement for any component, support as a beginning [7] [8].
* Calculate the support for all 1-itemsets (I1) [7] [8].
* Choose I1 that meet the minimum requirement as candidates (C1), and drop the rest [7] [8].
* Find I2 for C1 and calculate the rest of the components [7] [8].
* Repeat for Ix and C(x-1) until no more candidates are available [7] [8].
* Typically, the biggest length is C3 at I4 sets, in practice a limit to the length can be placed.

**Implementation Details:**

Apyori [7] library was used for this implementation on python.

* As input, concepts mentioned were put into a list of lists, each list corresponding to a record of the posts (example: ["Hypothyroidism ", "Hypertensive disease ", "Ulcer "]).
* Each list was extended to include age, gender, drug and drug family, in order to obtain the relation between them (Drug and Drug Family relations were later removed to avoid confusion.
* After running Apriori using Apyori. The result was printed in a .txt file to be reviewed later.

**Machine Learning:**

**Preparing the dataset:** After building the dictionary, it is now possible to identify the diseases, ADRs, and mental issues that were mentioned in the user posts.

* **Pandas** library was used to access and manipulate the data.
* A scanner was made to iterate on every token in every record in the stemmed version of the dataset, matching each token with an equivalent in the dictionary.
* When a token matches the dictionary, it marks the meaning of the concept as existing if another token with same meaning, such as (‘ache’: ‘pain’, ‘pain’: ‘pain’), it is ignored to limit repetition.
* The number of concepts (diseases, ADRs and mental issues), were counted for each record.
* Two types of classes are created using this process.
  1. Concept exists, where for each concept, a Boolean value is given to determine the existence of the concept in the record.
  2. Concept count, where the number of specific concepts in a particular range is drawn, meaning the number of distinct ADRs in a record for example is 5, this number is recorded and then assorted in the following ranges as a class.
     + 0 for 0 concept.
     + 1 for range [1,3].
     + 2 for anything more than 3.

In the previous example, 5 will be in the 2 class.

1. As for the features, user information harvested is used, such as age, gender, blood pressure, weight and height. Unfortunately, the MedHelp dataset has a lot of missing data, so not all user information could be used at the same time.
2. To solve this issue, the dataset was divided into three groups.
   1. Age + Gender only
   2. Age + Gender + Weight + Height
   3. Age + Gender + Blood Pressure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset Description | MedHelp Complete | Weight/Height | Blood Pressure | Ask A Patient |
| Size | 1557 | 130 | 462 | 757 |
| Maximum size | 12951 | 7868 | 6829 | 1939 |
| Minimum Size | 8 | 254 | 174 | 3 |
| Features | Age, Gender, Drug, Drug Family | Age, Gender, Drug, Drug Family, Weight and Height | Age, Gender, Drug, Drug Family, Blood Pressure | Age, Gender, Drug, Drug Family |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature Description | MedHelp Complete | Weight/Height | Blood Pressure | Ask A Patient |
| Age | 12 to 107  Blank=515 | 15 to 79  Blank=10 | 12 to 80  Blank=107 | 15 to 91  Blank=0 |
| Gender | ‘Male’: 581, ‘Female’: 553, Blank: 423, | ‘Male’: 71, ‘Female’: 41, Blank: 18 | ‘Male’: 199, ‘Female’: 167, Blank: 96 | ‘Female’: 423,  ‘Male’: 333 |
| Weight |  | 27 to 172 KG |  |  |
| Blood Pressure |  |  | Max: 295/135  Min: 102/60 |  |

* Now that missing data is in an acceptable level, the remaining missing values (Age and Gender) were imputed using the SciKit learn library, SimpleImputer. Gender was imputed based on most frequent strategy, while age was imputed based on median strategy.
* The data sizes for the three datasets respectively is: 1557, 130, 462. With dataset 2 ⊆ dataset 1 and dataset 3 ⊆ dataset 1. Each subset is stored in a separate excel file.
* The classes labels were stored in separate excel files, from which they can be extracted later and used in the classifier.
* The same procedure was applied for AskAPatient dataset, however the dataset did not include anything other than age and gender, therefore as a whole the size of the dataset is 757 with no divided parts between them.
* The dataset can thus be modified to suite any learning model, either by including all the labels (except the class) as a part of the features, or by separating them completely and only using the patient profile and drug/drug family as features.
* Drug/Drug family can also be used as features, where the goal is to find out the possibility by which a drug could cause ADRs and diseases.

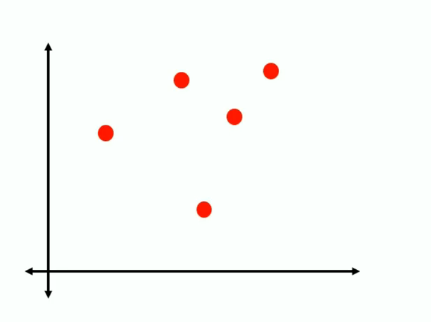
**Dataset Imbalance:** All the dataset have data label imbalance due to infrequency in the ADRs per each patient, in the complete MedHelp dataset, the highest percentage is “Pain” label, with 76% negative and 24% positive. The subsets are generally more even, but there is still too much imbalance in most labels. The “Hypertensive Disease” label and count labels are however have a better distribution, so they could be used for a more accurate measure of the quality of the dataset.

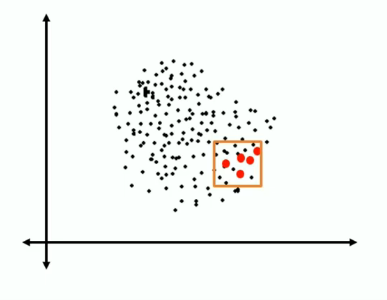
As for the remaining labels, several techniques could be applied.

1. Oversampling re-use: Several records of the minority class are repeated [9].
2. Under-sampling: Several records from the majority class are unused [9].
3. SMOTE (Synthetic Minority Over-sampling Technique): Where entirely new instances of the records are created to fill the gaps [9].

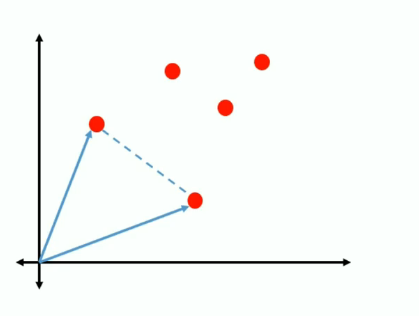
Approach number 3 was chosen for this application, using the imbalanced-learn implementation [10], using SMOTENC library (NC = Nominal Continuous).

**SMOTE** works as follows [9]:

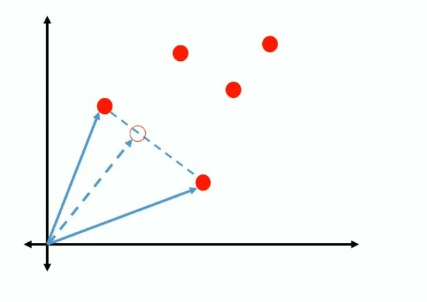
1) Isolate the minority class [11]



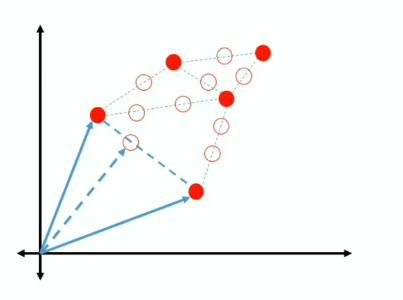
2) Find the k nearest neighbors depending on the over sampling requirement, calculate the distance between the two neighbors [11].



3) Multiply the distance with a random number between 0 and 1, placing a new record of the minority class on the new point created [11].



4) Repeat until data label imbalance is solved [11].



SMOTE however only works with continuous features, which is problematic since most the dataset is nominal/categorical, using either Boolean features or discrete categories that describe drug/drug family and counts.

This is why **SMOTENC** is used instead [9], SMOTENC looks acts the same way as SMOTE, however, between the closest neighbors, the median of the standard deviation of the nominal feature is taken into consideration for the distance [9]. With the distance calculated, continuous values are calculated based on normal SMOTE, while nominal features are given based on the majority of values within those neighbors [9].

SMOTENC implementation works exactly the same as SMOTE, by fitting the dataset to each other using SMOTENC object, only the difference is pointing out which of these features are nominal, SMOTENC is compatible with Pandas Dataframe, with the a list of new features as output to be used in the classifier.

Performance of the classifiers with and without SMOTENC will be compared together.

**Preparing the classifiers:**

**Goals:** There are several goals that could be achieved by using classifiers.

1. Predict the possibility for a user to be affected by an ADR or Disease based on extracted profile.
2. Predict which drug or drug family is the cause of the ADRs or Disease affecting the user, with the help of the user profile.
3. Predict the number of ADRs or Diseases that could be caused by a drug, based on the user profile and/or existing cases.

A separate classifier group will be made for each goal, each classifier group using a different dataset arrangement, which will be furthermore elaborated on in the following section.

**Dataset Arrangements:**

Features differ depending on the used subset in terms of user profile.

|  |  |  |  |
| --- | --- | --- | --- |
| **Arrangements** | **Labels** | | **Features** |
| 1 | Labels, Counts are categorized while running.  One Label is chosen for each prediction. | | User Profile, drug and drug family, profile differs for |
|  |  | |  |
| 2 | Labels are only limited to identifying which drug or drugfamily was responsible for any given disease. | Features include labels from the arrangement 1 and the user profile. | |
| 3 | Labels and Features are mixed, with the wanted feature being removed from the comparison dynamically. | | |

Arrangement 1 and 3 will include predictions for the Count of ADRs diseases, mental issues count will be ignored as their mentions are very low for in askapatient dataset. However the mentions themselves will be used for as features.

**The Classifiers:** Three classifiers are made as a performance comparison.

1. Random Forests
2. SVM
3. Naïve Bayes

**Random Forests (RF):** Random Forests is an ensemble classifier for classification and regression trees [12] [13].

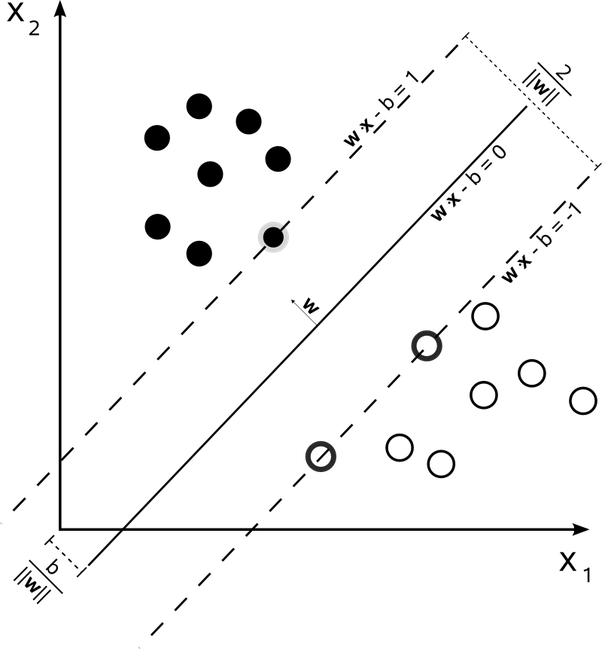
It works as follows:

1. Create a certain number of decision trees classier N [13] [14].
2. Commence bootstrapping operation:
   * Subsets of the original dataset are randomly selected to be used for each tree classifier in N, repetitions are possible.
   * For each subset, choose random features to use instead of using them all [13] [14].
3. Each resulting tree can be completely different from the others and can give different classification results.
4. Each one of these results calculated in a vote, with the highest voted result being chosen as the final result for the classification [13] [14].

**Random Forests (RF) in Scikit Learn:** The random forests implementation is provided is SciKit learn as an ensemble classifier, the main parameter given is “n\_estimators”, which is the number of bootstrapped trees that should be constructed. Estimators between 100 and 1000 were tested to ensure accuracy.

**Feature Selection:** In addition to the classification capabilities, RFs can be used for embedded feature selection [15], as each time a new tree is made with a subset, a purity metric can be measured, since not all trees see all features, there is an assured de-correlation between all features, this also makes them less prone to overfitting [15]. By calculating the purity of the tree can be used to derive the importance of the features, giving a standard on which feature is more important than the other and thus which feature could be removed [15]. Scikit Learn provides this capability view feature\_selection library, which can use an RF classifier to test the data and choose the best features based on an initial classification [15]. This method will be tested on all the classification methods, and observations will be provided for the improved performance if any.

**SVM (support vector machine):** First introduced by Vapnik, SVM is a very popular classification and regression technique [16]. It is based on the Structural Risk Minimization principle (SRM) [16], where the classifier maps input vectors to a higher dimensional space where a maximal separating hyperplane is constructed on each side of the hyperplane separating the data [16]. The goal is to maximize the hyperplane, the assumption being that when the bigger distance between two hyperplanes are maximized, the better the generalization error of the classifier [16].

The goal is to maximize 2 / | W |

The hyperplane is calculated based on several factors, the most important of which is kernel. Here are the most popular kernels [16]. RBF

* Linear
* Sigmoid
* Polynomial

RBF being the most commonly used for handling higher dimensional space better than Linear, with less parameter than Polynomial, and less numerical difficulties [16].

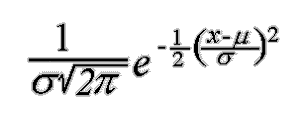
SciKit learn implementation provides all these assortment of kernels, as a parameter to the SVM classifier [16], with RBF as default kernel.

**Naïve Bayes classifier:** Naïve Bayes (NB) is one of the simplest probabilistic classifiers available, it is based on Bayes rule [17]. Where P© is the probability of a class (yes or no/Pain exist or doesn’t exist), and P(X) is the probability of all the features. P (C|X) is the probability of the class given all the features used [17]. If the probability of P(C=exist|X) > P(C=not|X), than the classifier predicts that yes, this ADR exists and vice versa [17].



Bayes rule

NB is reported to be extremely simple as it only relies only on simple mathematics.

The implementation on SciKit learn is GaussianNB, which uses the Gaussian distribution to represent continuous variables. Although there are other distribution options are available [18]. Including Bernoulli, Multinomial and Complement.

**Performance Metrics:** The performance of the classification is important to test capabilities of the trained classifier and evaluate the results without having to delve into specific cases [19]. The performance metrics used are based around the concept of the confusion matrix [19].

Each cell identifies the count of test cases that received a given result. True positives and negatives (TP and TN) indicate that the prediction was correct and matching to the actual test data. While false negative and positive (FN and FP) indicate the wrong in predictions. Usually it is better to have FP than FN because a positive can be an alert, while negative means no detection at all [19].

Based on the matrix, four metrics were chosen to measure the performance:

1. Accuracy = (TP+TN)/(FP+FN+TN+TP): Correct prediction/all predictions [19], it is good measure when most classes are nearly balanced [19].
2. Precision = TP/(TP+FP): The amount of TP in proportion to all the positives [19].
3. Recall = TP/(TP+FN): The amount of Negatives that were missed [19].
4. F1-Score= 2\*Precision\*Recall/(Precision + Recall): the harmonic mean of precision and recall, meant to be used as a simplification, instead of having to look into 2 different values, F1-Score gives an approximation that is closer to the smaller number, and thus more accurate and representable number [19].

There are other metrics that could be used, but these are the most standard (as discussed in the literature survey).

**Testing Parameters:** To calculate the performance, the dataset has to be trained using a given set of data, and tested against another set it has never seen. The dataset can be used for that by randomly splitting the data into a testing set and training set, the usual ratio for this purpose is 70% for training and 30% for testing. Values in the confusion matrix can be calculated in the testing part, and from this the metrics can be calculated.

**K-Fold Cross Validation:** Sometimes the testing of a certain part of the dataset can have different consequences from another, as in, the 30% testing if chosen from another part of the dataset can have a completely different result if it was used for training, while another part was used for testing [20]. One way to test this is to use K-Fold cross validation, where the dataset is split into K parts, while each K-1 part is used for training and only one for testing, this is repeated K times with each part [20]. The metrics are calculated and averaged together over the value of K. This usually take a longer time but it should provide a better perspective [20].

**Implementation Details:**

The implementation was originally meant to be made in python 2.7, and indeed most the text processing made Data retrieval and Dictionary Building was in 2.7. However the initial experiments with GloVe showed that 2.7 was extremely outdated. Therefore the learning process was moved to python 3.6, in which installing was much easier.

Environment: Using python required setting up an environment that could allow the use of all necessary libraries, thus Anaconda was installed, and configured for both 2.7 and 3.7 while each was in use.

This small list of libraries were necessary for the making of this project.

* Pandas: File and Dataset management
* NLTK (PorterStemmer, remove stopwords, tokenize text)
* Sklearn (classifiers, Count Vectorizer, label encoder, feature selection, K-Fold testing, imputer, TFidf Vectorizer)
* Imblearn.SMOTENC
* GloVe

IDE: Most text processing was made on anaconda’s Spyder IDE, however, while making the classifiers, the implementation was moved to Jupyter Notebook, which was more convenient as it allowed complete code separation and easier document comments. Jupyter uses separated cells which can be modified and run separately, while still maintaining the same variable and importation pool. This allowed the interfacing between several functions, where each classifier is in a different cell, each run of the classifiers is in a different cell, the dataset manipulation and preparation is independent from running the classifiers. Therefore whenever any arrangement was tested, the only change needed was to the data manipulation functions with minimal changes to the classifier.

Jupyter does not support concurrency. However for convenience, all cells could run at once in the sequence they were placed.

Cells could be changed from code to headings or simple text, they could be added, moved, stopped and repeated independently from one another.

Classifier Flow:

As shown in this flow chart:

1. The dataset is collected from the excel files.
2. The dataset is modified to suit whichever dataset arrangement in use.
3. Then there are three steps, any of which can be skipped at any point, and two of which at most are meant to compare performance whenever they are applied:
   * SMOTENC resampling.
   * Feature Filtering with random forests.
   * KFold Cross Validation (Which can be replaced with fixed percentage testing)
4. Store metrics to be used for comparison between all combinations later on.

