CHAPTER (1)

INTRODUCTION

1.1 PREFACE

As of writing, the world is going through the second wave of COVID-19 pandemic which has started in Wuhan city in China back in December 2019 and since that time it has recorded over 45 million infections in all over the world with a death toll that has exceeded 1 million. COVID-19 had been proven to be the most infectious pandemic since Spanish flu which killed at least 50 million people between 1918 and 1920 (cdc.gov, n.d.).

COVID-19 is an is an infectious disease caused by a newly discovered coronavirus (WHO, n.d.). Governments and health ministries all around the world took strict protective measures to prevent the spread of the infection among their citizens, measures such as social distancing, borders closings and lockdowns have been taken to contain the infection. These measures, however, prevent the direct communication between healthcare authorities and people. The pandemic has spread the panic among citizens and with a lack of necessary awareness, citizens are exposed to false information as well as to be infected with the virus and contact it to healthy people around them.

1.1.1 Customer Service During Pandemic

The protective measures that governments took during COVID-19 have a downside of preventing the direct communication between healthcare

authorities and people. As the pandemic spreading panic among citizens, the lack of the necessary awareness about the disease, will eventually expose people to false information and roomers and this will have them infected with the virus and transmit it to healthy people around them.

1.1.2 Call Center Solutions

One could say that a solution for the awareness problem would be a call center with employees to answer people's questions about COVID-19, but it turns out to be no so good idea as the call center brings much people together in the same place during a pandemic. Also organizing the call center in a way that employees could work from their homes is also difficult to implement as it introduces additional costs regarding employee's internet connections and technical support fees that may hinder the feasibility of the solution specially if talking about development countries.

1.1.3 Chatbot Solution

A chatbot is a computer program that interact with users using natural languages. This technology started in the 1960's; the aim was to see if chatbot systems could fool users that they were real humans (Shawar & Atwell, 2007).

However, chatbot systems are not only built to mimic human conversation, and entertain users, chatbots had proved their high business potential, for example in a study back in 2017 chat bots were chatbots are estimated to make savings of $8 billion per annum for businesses between 2017 and 2020. The majority of savings will be made through customer service as customers can now ask chatbot rather than having to call the business, allowing businesses to save on call center staff (Doherty , 2018).

The reason for that is that the users have usually have certain motivations for using chatbots these motivations could be divided into four categories (Brandtzaeg & Følstad, 2017):

i. Productivity motivations: concerns the convenience of using chatbots (whether they are easy or fast to use).

ii. Entertainment motivations: concerns the entertainment value of using chatbots (whether they are fun to use). Some users report that they use chatbots when bored to kill time.

iii. Social/relational motivations: Concerns the use of chatbots for social or relational purposes. Typically, chatbots are seen as a personal, human means of interaction that may have social value. Some also use chatbots to strengthen social interactions with other people.

iv. Novelty/ Curiosity motivations: Concerns the use of chatbots out of curiosity or because they are a novelty. Often, the stated aim is to investigate chatbots' capabilities.

The most frequently reported motivational factor to use chatbots by the users is "productivity"; chatbots help users to obtain timely and efficient assistance or information (Brandtzaeg & Følstad, 2017).

Managing demand through existing intelligent automation and self-service is an import aspect of the era of data. AI technology, voicebots and chatbots have all made great leaps forward over the last few years and, used appropriately, can make a massive difference to your ability to handle demand (Mordue & Burton, 2020). Unfortunately there is a lack of research works in the area of Arabic chatbots (AlHumoud, et al., 2018) so deploying a solution based on another scholar's work will be hard and needs much

adaptation for the problems of interoperability of Arabic Natural Language Processing (NLP) tools (Jaafar & Bouzoubaa , 2015)

1.2 PROBLEM STATEMENT

As a result of the protective measures taken by governments, healthcare support using traditional call center solutions are no more considerable during COVID-19 pandemic because bringing a group of people into a crowded place risking their health and their family's health. It's also difficult to have all the call center employees working from homes because it introduces additional costs for maintenance and technical support. Using chatbots is also difficult for Arabic language because of there is no significant research works in this direction.

1.3 PRPOSED SOLUTION

We propose as a solution an Arabic natural-language machine-learning-based healthcare chatbot that answers the frequently asked questions (FAQ's) about COVID-19, specifically, Modern Standard Arabic (MSA).

1.4 OBJECTIVES

The objectives of this project is as follows:

i. The chatbot should interact with users using Arabic natural language interface.

ii. To compare the performance of various text classification algorithms on MSA.

iii. Study the effect of feature extraction techniques on the performance of classification algorithms.

iv. Give a recommendations for future scholars who are intended to work in Arabic text classification tasks.

1.5 METHODOLOGY

In this project a data-driven approach has been taken to implement the chatbot solution by training nine text-classifier on training sets and then aggregate their answers using hard and soft voting techniques. The steps the project has walked through are:

i. Identifying the use case.

ii. Data collection

iii. Data preprocessing

iv. Feature extraction

v. Models training

vi. Testing and evaluation

vii. Deployment and monitoring

1.6 THESIS OUTLINES

This thesis is organized as follows:

Chapter 1, the introduction, introduces the problem that the project is trying to solve and provide the reasder with the motivation behind conducting it.

Chapter 2, background and literature review, gives the reader background knowledge about the scope of the research and highlights related work that has been done before.

Chapter 3, the methodology, describes in details the steps by which the project has been conducted.

Chapter 4, the results, illustrates the results of the project.

Chapter 5, conclusions and recommendations, concludes the project achivements and provide recommendations for future projects in the same topics.

CHAPTER (2)

BACKGROUND AND LITRATURE REVIEW

2.1 BACKGROUND:

2.1.1 Machine Learning

Acquisition of knowledge is a key requirement of solutions intended to exhibit intelligent behavior. Because learning is an effective way to introduce such knowledge, most AI studies to date have employed learning techniques. The primary aim of learning from knowledge is to allow computers to learn automatically without human intervention or assistance. This process could involve any method that includes some inductive component, ranging from a simple Kalman filter to a complex convolutional neural network. No method is inherently better than any other; each is more or less well-suited to different scenarios, e.g., a softer learning curve, faster execution, or more flexible solutions. Furthermore, the performances of various methods are closely related to the quality and quantity of data: when more information is gathered, and less noise is present in the data, better solutions can be obtained. The most important families of techniques are artificial neural networks (ANNs), support vector machines (SVMs), random forest (RF), evolutionary algorithms (EAs), deep learning (DL), Naïve Bayes (NB), decision trees (DTs), and regression algorithms (RAs) (Contreras & Vehí, 2018).

Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed. Figure 3-1 shows a general machine learning process that any machine learning project should go

through. The first stage of any machine learning project is data collection, i.e. acquiring data from various sources, next to that coming data preprocessing operations which apply numerous techniques that aim to remove as much noise as possible from data and enhance its quality in order to be fed to machine learning training algorithms. Machine learning training algorithms are generally optimizing an objective function that maximizes/minimizes a model's performance measure. After the machine learning models is trained, its measured on a performance metric to assess its optimality and if it was found to be optimal, it will be deployed in production or else, the training process will be repeated in order with another set of conditions than the previous training iteration.

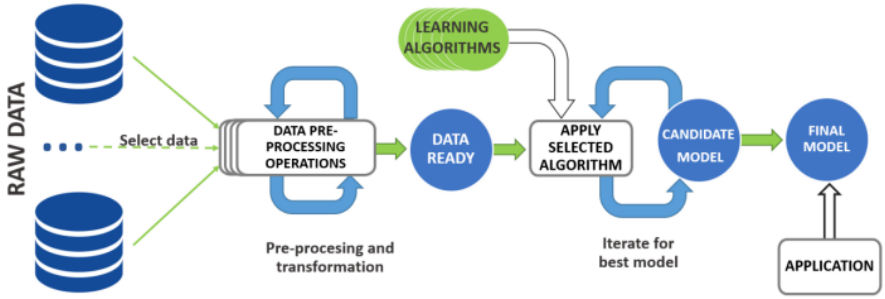

Figure 2 - 1 General process of machine learning projects (Contreras & Vehí, 2018)

Machine Learning systems can be classified according to the amount and type of supervision they get during training to three types (Géron, 2019):

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Figure 3-2 shows the three types and their applications

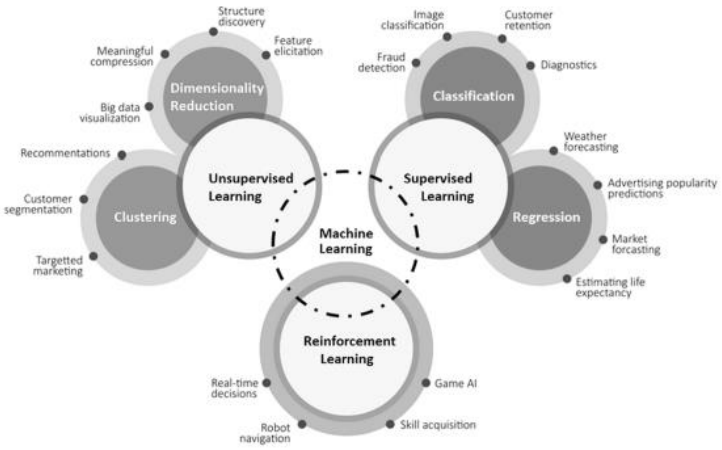

Figure 2 - 2 types of machine learning tasks and their applications (Agarwal, et al., 2020)

In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels. In unsupervised learning, as you might guess, the training data is unlabeled. In reinforcement Learning The learning system, called an agent, can observe the environment, select and perform actions, and get rewards in return or penalties in the form of negative rewards (Géron, 2019).

2.1.2 Text Classification

Classification is one of the main tasks of supervised learning and it is defined as the process of categorizing a given set of data into classes. Classification can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories (Waseem, 2020).

Text classification is the process of assigning tags or categories to text according to its content. It's one of the fundamental tasks in natural language

processing with broad applications such as sentiment analysis, topic labeling, spam detection, and intent detection (monkeylearn, n.d.).

In this project text classification has been used for intent classification to classify user questions into one of set of predetermined classes.

2.1.3 Classification Algorithms

2.1.3.1 Naive Bayes

Naive Bayes methods (Zhang, 2004) are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n:

$$P(y|x_1, \dots, x_n) = \frac{P(x_1, \dots, x_n|y)P(y)}{P(x_1, \dots, x_n)} \qquad (2\text{-}1)$$

Using the naive conditional independence assumption that

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = p(x_i|y) \qquad (2\text{-}2)$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} p(x_i|y) \qquad (2\text{-}3)$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} p(x_i|y) \qquad (2\text{-}4)$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $p(x_i|y)$; the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $p(x_i|y)$.

In this project three distribution assumptions have been used:

1- Gaussian distribution

2- Multinomial distribution

3- Complement distribution

Gaussian Naïve Bayes implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$p(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma_y^2}\right) \qquad (2\text{-}5)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

Multinomial Naïve Bayes implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_{y_i} = \left(\theta_{y_1}, \ldots, \theta_{y_n}\right)$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{y_i} is the probability $p(x_i|y)$ of feature i appearing in a sample belonging to class y.

The parameters $\theta\ y$ is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{y_i} = \frac{N_{y_i}+\alpha}{N_y+\alpha n} \qquad (2\text{-}6)$$

where $N_{y_i} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the set T , and $N_y = \sum_{i=1}^{n} N_{y_i}$ is the total count of all features for class y.

Complement Naïve Bayes (Rennie, et al., 2003) implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the complement of each class to compute the model's weights. The inventors of CNB show empirically that the parameter estimates for CNB are more stable than those for MNB. Further, CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks.

The procedure for calculating the weights is as follows:

$$\hat{\theta}_i = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}} \qquad (2\text{-}7)$$

$$w_{ci} = \frac{w_{ci}}{\sum_j w_{cj}} \qquad (2\text{-}8)$$

where the summations are over all documents j not in class c, d_{ij} is either the count or tf-idf value of term i in document j, α_i is a smoothing hyperparameter like that found in MNB, and $\alpha = \sum_i \alpha_i$. The second normalization addresses the tendency for longer documents to dominate parameter estimates in MNB. The classification rule is:

$$\hat{c} = \arg\min_c \sum_i t_i w_{ci} \qquad (2\text{-}9)$$

i.e., a document is assigned to the class that is the poorest complement match.

2.1.3.2 Nearest Neighbors

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply "remember" all of its training data.

To search for nearest neighbors three searching algorithms has been used to train the nearest neighbor classifier using grid search and cross validation:

1. Brute Force
2. K-D Tree
3. Ball Tree

The most naive neighbor search implementation involves the brute-force computation of distances between all pairs of points in the dataset: for N samples in D dimensions, this approach scales as $O[DN^2]$.

To address the computational inefficiencies of the brute-force approach, a variety of tree-based data structures have been invented. In general, these structures attempt to reduce the required number of distance calculations by efficiently encoding aggregate distance information for the sample. The basic idea is that if point A is very distant from point B, and point B is very close to point C, then we know that points A and C are very distant,

without having to explicitly calculate their distance. In this way, the computational cost of a nearest neighbors search can be reduced to $O[DN\ log(N)]$ or better. This is a significant improvement over brute-force for large N.

An early approach to taking advantage of this aggregate information was the KD tree data structure (short for K-dimensional tree), which generalizes two-dimensional Quad-trees and 3-dimensional Oct-trees to an arbitrary number of dimensions (Bentley, 1975). The KD tree is a binary tree structure which recursively partitions the parameter space along the data axes, dividing it into nested orthotropic regions into which data points are filed.

The construction of a KD tree is very fast: because partitioning is performed only along the data axes, no D -dimensional distances need to be computed. Once constructed, the nearest neighbor of a query point can be determined with only $O[\log(N)]$ distance computations. Though the KD tree approach is very fast for low-dimensional ($D<$ 20) neighbors searches, it becomes inefficient as D grows very large: this is one manifestation of the so-called "curse of dimensionality".

To address the inefficiencies of KD Trees in higher dimensions, the ball tree data structure was developed (Omohundro, 1989). Where KD trees partition data along Cartesian axes, ball trees partition data in a series of nesting hyper-spheres. This makes tree construction more costly than that of the KD tree, but results in a data structure which can be very efficient on highly structured data, even in very high dimensions.

A ball tree recursively divides the data into nodes defined by a centroid C and radius r, such that each point in the node lies within the hyper-sphere

defined by r and C . The number of candidate points for a neighbor search is reduced through use of the triangle inequality:

$$|x + y| \leq |x| + |y| \qquad (2\text{-}10)$$

With this setup, a single distance calculation between a test point and the centroid is sufficient to determine a lower and upper bound on the distance to all points within the node. Because of the spherical geometry of the ball tree nodes, it can out-perform a KD-tree in high dimensions, though the actual performance is highly dependent on the structure of the training data.

2.1.3.3 Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection (Guyon, et al., 1993; Cortes & Vapnik, 1995).

A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Given training vectors $x_i \in \mathbb{R}^n, i = 1, \ldots, n$, in two classes, and a vector $y \in \{-1, 1\}^n$, SVC solves the following primal problem:

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \qquad (2\text{-}11)$$

Subject to:

$$y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i \qquad (2\text{-}12)$$

$$\xi_i \geq, i = 1, \ldots, n \qquad (2\text{-}13)$$

Its dual is

$$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \qquad (2\text{-}14)$$

Subject to:

$$y^T\alpha = 0 \qquad (2\text{-}15)$$

$$0 \leq \alpha_i \leq C, i = 1, \ldots, n \qquad (2\text{-}16)$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix,$Q_{ij} \equiv y_i y_j K(y_i, x_j)$, where $K(y_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The kernel function used are the following:

1. linear: $\langle x, x' \rangle$. (2-17)
2. polynomial: $(\gamma\langle x, x' \rangle + r)\, d$. (2-18)
3. rbf: $\exp(-\gamma\|x - x'\|\, 2)$. (2-19)
4. sigmoid $(\tanh(\gamma\langle x, x' \rangle + r))$. (2-20)

2.1.3.4 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions (Zhang, n.d.).

Given a set of training examples $(x_1, x_2), \dots, (x_n, y_n)$ where $x_i \in R^m$ and $y_i \in \{-1,1\}$, our goal is to learn a linear scoring function $f(x) = w^T x + b$ with model parameters $w \in R^m$ and intercept $b \in R$. In order to make predictions, we simply look at the sign of $f(x)$. A common choice to find the model parameters is by minimizing the regularized training error given by

$$E(w, b) = \frac{1}{n}\sum_{i=1}^{n} L\big(y_i, f(x_i)\big) + \alpha R(w) \qquad (2\text{-}21)$$

where L is a loss function that measures model misfit and R is a regularization term (aka penalty) that penalizes model complexity; $\alpha > 0$ is a non-negative hyperparameter.

Different choices for *L* entail different classifiers such as:

- Hinge: (soft-margin) Support Vector Machines.
- Log: Logistic Regression.
- Least-Squares: Ridge Regression.
- Epsilon-Insensitive: (soft-margin) Support Vector Regression.

Popular choices for the regularization term *R* include:

- L2 norm: $R(w) = \frac{1}{2}\sum_{i=1}^{n} w_i^2$ (2-22)
- L1 norm: $R(w) = \sum_{i=1}^{n} |w_i|$ (2-23)

• Elastic Net: $R(w) = \frac{\rho}{2}\sum_{i=1}^{n} w_i^2 + (1-\rho)\sum_{i=1}^{n}|w_i|$ (2-24)

Stochastic gradient descent is an optimization method for unconstrained optimization problems. SGD approximates the true gradient of $E\ (w\ ,b)$ by considering a single training example at a time.

The algorithm iterates over the training examples and for each example updates the model parameters according to the update rule given by:

$$w \leftarrow w - \eta\left(\alpha\frac{\partial R(w)}{\partial w} + \frac{\partial\left(w^T x_i + b, y_i\right)}{\partial w}\right) \quad (2\text{-}25)$$

where η is the learning rate which controls the step-size in the parameter space. The intercept b is updated similarly but without regularization.

2.1.3.5 Voting

Voting is an ensemble learning technique used to aggregate the predications of multiple predictors (in most cases classifiers) in order to reduce bias of individual classifiers with a cost of increased variance. Voting done in two ways: hard voting and soft voting.

Hard voting is the same as voting in its traditional sense, i.e., treating predictors (in our case classifiers) as voters and counting the number of classifiers that predict each class and return the class that have the majority of classifiers predicting it.

Soft voting, however, is another story, it averages the probability estimates of each class that was estimated by each classifier and return the class with highest average probability estimate.

Soft voting sometimes be more powerful than hard voting because it gives a balance by reducing the effect of low confidence (low probability estimate) classes that can pass through hard voting.

2.1.4 Performance Metrics

Performance metrics are used in machine learning to assess the optimality of a given model or to make comparison between multiple models. Three performance metrics has been used in this project:

- Precision score
- Recall score
- F1 score

2.1.4.1 Precision Score

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances. Precision score is given by the following equation:

$$precison = \frac{\#True\ Posatives}{\#True\ Posatives + \#False\ Posatives} \qquad (2\text{-}26)$$

2.1.4.2 Recall Score

Recall (also known as sensitivity) is the fraction of the total amount of relevant instances that were actually retrieved. Recall score is given by the following equation:

$$Recall = \frac{\#True\ Posatives}{\#True\ Posatives + \#False\ Negatives} \qquad (2\text{-}27)$$

2.1.4.3 F1 Score

The F1 score (also F-score or F-measure) is a measure of a test's accuracy. It is calculated from the precision and recall of the test, where the precision is the number of correctly identified positive results divided by the number of all positive results, including those not identified correctly, and the recall is the number of correctly identified positive results divided by the number of all samples that should have been identified as positive. F1 score is given by the following equation:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \qquad (2\text{-}28)$$

2.2 LITERATURE REVIEW

Related work that has been done in the area of Arabic chatbots is divided into two parts:

- AIML approach
- Pattern matching approach

2.1.1 AIML Approach

Among the earliest research studies on Arabic chatbot application in the collected related work is Quran chatbot (Shawar & Atwell, 2004). It is a chatbot on the Quran Islamic holy book. The format of the user inputs are Arabic words with diacritics, that is used as phonetic guides. The chatbot

replies by finding the “Ayahs” from the Quran that contain the user‟s input. The nature of Quran text is non-conversational, a Java program is developed to adopt a learning process. The learning process was based on the most significant word of the “Ayah” that represents the category in the AIML file and template is the Ayah. Arabic AIML file is generated by the Java program. The conversation length is short since the chatbot responses with a single response to a single user input. The domain of the conversation is limited by the content of the Quran. From that, the sources of the chatbot dataset are retrieval-based. The interaction and response of the chatbot are limited by the pool of the most significant word of the “Ayaha” that are extracted by the java program.

The Quran14-114 chatbot by (Shawar, 2011) is a version of the ALICE chatbot (Wallace, 2003) added to the Java program to interact as the Quran chatbot. The conversation in the chatbot is short. The user inputs a question or a statement in English, and the chatbot responses with one or more appropriate „Surahs‟ and „Ayahs‟ from the Quran in both English and Arabic. The Java program reads the Quran text from a corpus and converts it to the AIML format to be used by ALICE chatbot. The domain of the conversation is closed based on the content of the Quran in both the English and Arabic languages. Also, this is a retrieval-based model as the dataset is limited by the content of the Quran. The challenge in this work is to show how ALICE chatbot adapted to learn from non-conversational text.

The Arabic Web Question Answering (QA) chatbot (Shawar, 2011; Brini, et al., 2009), is a web interface chatbot based on an Arabic QA corpus, that was built from five different web pages with 412 Arabic question and answer. Those web pages cover topics such as motherhood and pregnancy,

dental care, fasting and health, blood disease such as cholesterol and diabetes, and blood charity. The chatbot supports more than one closed domain, and thus regarded as a closed conversation domain. The chatbot conversation is short, where the user inputs a textual question in MSA about one of the supported domains and, the chatbot responses with the answer without using sophisticated NLP. Also, a Java program was developed to convert the text corpus to create two AIML files atomic and default. The atomic file contains the questions and answers that appear in the corpus. The default file is used to guarantee that the user question is mapped to the appropriate question stored in the knowledge base. Moreover, the file is built using the first word and the most significant word approach. The first word acts as a classifier to the question and the most significant word is the least frequent in the question.

The latter is done by building questions‟ frequency list after applying a tokenization process to the question. The generated list contains the question‟s words along with their frequencies. Then, the approach extracts the two most significant words in the list, those are the two least frequent words, used as keywords to map the question to an answer. The purpose of using the most significant word approach was to increase the rate of the expected output. The chatbot was tested by entering fifteen questions and the result was 93% correct answers. The main drawback in this model appears when the structure of the question is changed or altered from the stored in the knowledge base, then the chatbot responses with wrong answers. That happens because the chatbot does not use a heuristic to select a proper response and it is based on a direct retrieval model. Also, a success

rate of to be 93% is not justifiable having a dataset of fifteen questions only.

BOTTA chatbot by (Ali & Habash, 2016) is a female chatbot, that simulates friendly conversations with users. The chatbot supports Egyptian Arabic dialect for both input and output. BOTTA is available to the public. It simulates the English chatbot Rosie (Pandorabots, 2018). The knowledge base is made up of Rosie's AIML files set. Some of Rosie's AIML files are translated directly to Arabic, and the others are modified according to the use of the Arabic dialects. Also, for each conversation, BOTTA chatbot temporally stores the basic information about the user such as age, gender, and nationality by asking questions yielding a conversation that is open since the chatbot can response to different topics domain. The length of the conversation is long where chatbot can response to the user based on previous information in conversation. However, it does not update the knowledge base and add new responses, so it is based on retrieval-based model. It depends on a pool of predefined responses using heuristics to response with an appropriate output. Also, the chatbot does not perform the text normalization on the user input to get the suitable response. It performs orthographic transformations, that includes correcting common spelling mistakes of the user input. With this method, BOTTA was able to resolve 85.1% of the common spelling mistakes in Arabic typing.

2.1.2 Pattern Matching Approach

(Hijjawi, 2011) implemented the ArabChat, which is a conversational agent web interface. The chatbot conversation domain is closed, designed to serve the students of the Applied Science University in Jordon. The interaction between the user and the chatbot is through textual

Arabic MSA language. The conversation remains ongoing until one of the conversation‟s parties terminates it. The ArabChat reuses the previously exchanged information during the conversation as a response to the user input, creating long conversations. The core components of the ArabChat chatbot are the scripting engine and a scripting language. The scripting engine is divided into subcomponents, that allows handling topics of the conversations. ArabChat knowledge base contains 1218 utterances, that are classified into contexts, each context contains rules. The rules consist of patterns and associated textual responses. ArabChat was tested over 174 users, the average input for each user was seven inputs per user. The result shows that 73.56% of the inputs matched the expected output.

Enhanced ArabChat (Hijjawi, et al., 2016) is an updated version of ArabChat. This version uses extra features including Utterance Classification and Hybrid Rule. These improvements were at the engine level while some additional improvements need to be added to the scripting language and knowledge base to meet the changes needs. Utterance classification feature aims to distinguish between a question and non-question utterances. It works by adding extra keywords to the pattern of the question-based rule, to deal with keyword matching. Hybrid Rule is the second feature and it focuses on how to reply and deal with an utterance that request many topics. Although ArabChat gave a better result of Ratio of Matched Utterances to the Total (RMUT) than enhanced one due to unserious users, the manual checking gives more accurate results and showed improvement in performance. By analyzing logs manually, Enhanced ArabChat deals successfully with 82% of utterances with two topics and this ratio is decreasing when the number of topics is increased in

the utterance. Using manual checking, classifying utterance shows a high percentage of question-based utterances due to three factors: the selected domain, the users' needs, that implies that they are more likely to ask rather than discuss, and difficulties to script a large number of rules.

ArabChat with classification methodology (Hijjawi, et al., 2013) is another ArabChat update by Hijjawi, Bandar and Crockett. Using a new classification methodology for Arabic utterances. This new approach classifies the sentences into questions and non-questions including assertions and instructions. The benefit of applying this approach is that the number of patterns required per rule will decrease and hence increase the performance by firing the suitable rule, depending on the utterance type being a question or non-question. Different topics and list of function words have been used from domains such as politics, religion, sports, education, business and adding some synthetic non-question sentences and indirect questions. This classification is done by pre-processing the Arabic sentence into equivalent numeric tokens and then importing the tokens into a machine learning toolkit in WEKA. In WEKA, a Decision Tree, which achieve the highest accurate classifier to be applied on the tokenized numeric dataset, is generated and then is converted into a standard IF-THEN classification rule to classify utterances.

Mobile ArabChat (Hijjawi, et al., 2015) is based on the original (Hijjawi et al., 2011) and it is a mobile-based conversational agent and it is also used to work as an advisor for students in Applied Science University in Amman. It is a light version of ArabChat implemented in Android. Although there are some challenges facing users in the Arab Countries such as slow and unstable internet connection and limited

bandwidth, this application works even with limited Internet bandwidth. Mobile ArabChat implemented pattern matching approach based on the text. This framework consists of the same component as in ArabChat: scripting engine, scripting language and a knowledge base. Based on a subjective approach, 96% of users agree that using Mobile ArabChat via mobile is better than using the same system via desktop. However, Mobile ArabChat needs an internet connection to work.

Abdullah (Alobaidi, et al., 2013) is an Arabic Conversational Intelligent Tutoring System (CITS) that teaches children aged 10 to 12 years old essential topics about Islam. This online system can engage with students using MSA. That asking a series questions to the students, and discuss with them their answers, using Classical Arabic to give evidences from the Quran and Hadith, which is the sayings and traditions of the Prophet of Islam Muhammed. The system is using images and sound effects to interact with students and can determine the student's knowledge level and hence direct the conversation. Abdullah CITS can distinguish between the user's questions and answers. The framework is based on a Pattern Matching approach, it consists of knowledge base having subject topics, the Conversational Agent scripting language to deliver the tutorial conversation to the learners, and The Tutorial Knowledge Base to determine the level of individual student knowledge and the subject.

LANA ((Aljameel, et al., 2017) is another CITS and it was developed for children with Autism Spectrum Disorder (ASD) that are 10 to 16 years old who have reached a basic competency with the mechanics of Arabic writing to teach them topics on science using MSA. Children with ASD have difficulties in traditional learning because the teacher can't meet

the need of every individual student. LANA engages children with a science tutorial delivered in MSA. It is similar to Abdullah CITS, but it offers different learning style models such as visual, auditory and kinesthetic, enabling children to practice learning skills independently based on their needs using pattern matching and short text similarity algorithm. This system also interacts with children using materials such as picture, audio, or instructions according to the user‟s learning style.

CHAPTER (3)

METHODOLOGY

3.1 INTRODUCTION

In this chapter the methodology used in the project has been detailed including the overall project plan, data collection methods, data preprocessing steps, feature selection and extraction, machine learning models training, testing metrics and deployment model proposed.

In this project a data driven approach has been adopted by training and testing classification models on conversational data. Also a retrieval-based chatbot model has been adopted. In retrieval models the chatbot retrieve responses from a set of prelisted responses stored in chatbot's Knowledge Base (NB).

3.2 PROJECT PLAN

As can be seen from Figure 3.1 a flow chart that describes Project plan which describe how project progressed

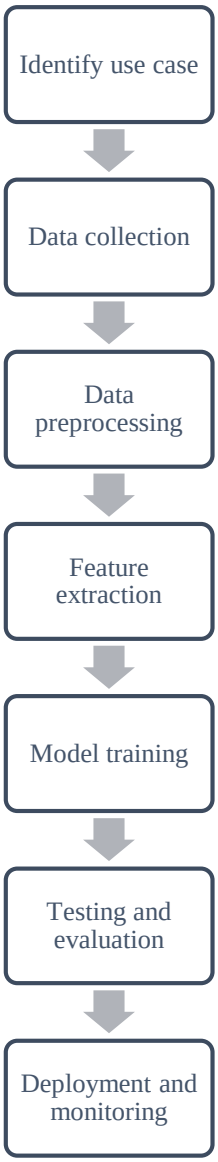

Figure 3- 1 The overall project plan

Use case: for the system to be created a use case should be identified. By the time of writing the world is living during COVID-19 pandemic and there is much fear, ignorance and lack of awareness about the disease so in this project the use case selected was COVID-19 FAQ's answering chatbot system. section 3.4 illustrates the use case in details as well as the target classes that defines the domain of the system.

Data collection: after defining the use case comes data collection step and it is an important stage of any machine learning project as the data is located at the heart of machine learning field. section 3.5 demonstrates tools and methods by which the data has been collected.

Data preprocessing: raw data collected from various sources tends to be noisy and incomplete and need further processing (preprocessing) before a useful information (features) can be extracted from it. Data preprocessing is defined as a data mining technique that involves transforming raw data into an understandable format (Techopedia.com, 2020), section 3.6 details the preprocessing steps used to clean the collected data.

Feature extraction: after the preprocessing has done data is ready for extracting useful information out of it. This process is known as feature extraction. In machine learning training examples are represented as a set of data points and each data point is represented as a set of features called the feature vector. In machine learning context a feature is defined as an individual measurable property or characteristic of a phenomenon being observed (Bishop, 2006). Two processes are involved in extracting useful features: feature selection and feature extraction. Feature extraction is defined

as the process of Getting useful features from existing data whereas feature selection is the process of Choosing a subset of the original pool of features (Quantdare.com, 2019). Section 3.7 provides a detailed description of feature selection and extraction processes used to extract information from the cleaned text examples

Models training: after features are selected they fed to classification models to train them on the data. Model training is a procedure in supervised machine learning that estimates parameters for a designed model from data set with known classes (Igi-global.com, 2020). A machine learning model can be a mathematical representation of a real-world process (bhattacharjee, 2017). section 3.8 describes in depth the whole process of training and validation of classification models,

Testing and evaluation: in order to verify performance of models; suitable evaluation metrics should be defined and used. Details on testing and evaluation metrics that have been used to assess model performance are given in section 3.9.

To make the system usable by the end users it should be encapsulated within a deployment model the model selected for deployment is a client-server model as it will be illustrated. Sections 3.10 and 3.11 provide detailed description about the structure and functions of deployment model and how components interact with each other.

3.3 SOFTWARE LIBRARIES

The project has built entirely using python programming language and the following python frameworks has been used to develop the system:

1. Numpy
2. Scipy
3. Pandas
4. Scikit –learn
5. PyQt5

3.3.1 Numpy

NumPy is a free and open source python library, adding support for large, multi-dimensional arrays and matrices for python programming language, along with a large collection of high-level mathematical functions to operate on these arrays (Oliphant, 2006; Walt, et al., 2011).

3.3.2 Scipy

The SciPy library, a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, and much more. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering (Virtanen, et al., 2020).

3.3.3 Pandas

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. In particular, it offers data structures and operations for manipulating numerical tables and time series (McKinney, 2020).

3.3.4 Scikit-learn

Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings (Pedregosa, et al., 2011).

3.3.5 PyQt5

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android (PyQt, 2012).

3.4 USE CASE

The chatbot will be designed to answer FAQ's related to COVID-19 pandemic. The FAQ's used are those in United Nations' website (United Nations, 2020). Those FAQ's has been reduced from 38 FAQ's as listed in the website down to just 15 FAQ's by:

Combining different related questions into one question.

Discard ones those are rarely asked.

The reduction was necessary to reduce the amount of data needed to train classification models Theses 15 questions we will here refer to as intents or classes, depending on the context. Table 3-1 illustrate these classes and their indices as they used in various project stages as well as the description of each class and an illustrative example.

Table 3- 1 The text classes, their indices, description and an illustrative example.

Class index	Class name	Description	Examples
0	`animals_infection`	Questions about the ability of corona virus to transmit between humans and animals	هل ينتقل المرض من الإنسان إلى القطط ؟ هل الخفاش هو المصدر الأساسي للمرض؟
1	`asymptotic_infection`	Questions about infection in Incubation period	هل يمكن أن أصاب بالعدوى من

			أشخاص لا تظهر عليهم أعراض؟
2	end_conversation	Statements that intends to end conversation	شكرا وداعاً إلى اللقاء
3	life_time	Questions about period the virus stay alive in different conditions	كم من الوقت يعيش الفيروس على مقابض الأبواب ؟
4	mailing_packages	Questions about interchanging mail and cargos with areas in which disease has wide spread	هل يمكنني شحن البضائع من دول ينتشر فيها الوباء ؟
5	mask_usage	Question about masks and their usage	هل الكمامات تقي من المرض ؟
6	period_before_symptoms	Questions about incubation period	كم تبلغ فترة حضانة الفيروس ؟
7	Sanitizers	Questions about sanitizers	كيف أستخدم المعقم ؟
8	second_infection	Questions about infection with the	هل يمكن أن أصاب بالمرض أكثر من مرة ؟

		disease more than once	
9	self_protection	Questions about ways to prevent infection	كيف أحمي نفسي ؟ ما هي إجراءات الوقاية ؟
10	start_conversation	Statements intended to start conversation with chatbot	السلام عليكم أهلاً مرحباً
11	Symptoms	Questions about the symptoms of COVID-19	ما هي أعراض المرض ؟ كيف أعرف أنني أصبت بكورونا ؟
12	Temprature	Questions about the effect of environmental conditions on the virus	هل يموت الفيروس في الجو الحار؟ هل يعيش الفيروس الجو الرطب؟
13	Test	Questions about COVID-19 testing	متى يجب علي أن أفحص ؟
14	ways_of_infection	Questions about the ways disease transmit from one person to another	هل ينتقل المرض بالمصافحة ؟ هل العطس ينقل المرض ؟

3.5 DATA COLLECTION

At the beginning a small dataset of 250 examples has been developed manually we will refer to here as `init_set.` The following Nine classifiers has been trained on this dataset:

- Gaussian naive Bayes.
- Complement naive Bayes.
- Multinomial naïve Bayes.
- K-nearest neighbors.
- logistic regression with L2 regularization.
- Support Vector Machines with polynomial kernel.
- Support Vector Machines with RPF kernel.
- Support Vector Machines with sigmoid kernel.
- Stochastic gradient decent.

A tester has been developed based on these nine classifiers using soft and hard voting among them. The tester has distributed among class students and used as data collection tool. Data collected by the tester consisted of 501 examples. The combination dataset of collected examples and `init_set` will be referred to here as `col_set`. Table 3-2 illustrates number of examples in `init_set` and collected data for each class.

Table 3- 2 Number of examples in `init_set` and `col_set`

Class name	# Examples in `init_set`	# Examples in collected data
`animals_infection`	18	37
`asymptotic_infection`	14	29
`end_conversation`	15	34
`life_time`	18	38
`mailing_packages`	16	26
`mask_usage`	19	37
`period_before_symptoms`	16	20
`Sanitizers`	17	40
`second_infection`	16	24
`self_protection`	17	47
`start_conversation`	20	30
`Symptoms`	17	42
`Temprature`	14	20
`Test`	17	32
`ways_of_infection`	16	45
Total	250	501

`col_set` has been used to train the final text classifiers. Performance achieved on collected data by the classifiers trained using `init_set` is detailed in results chapter.

3.6 DATA PREPROCESSING

To clean raw text data a preprocessing pipeline has been designed in order to remove unnecessary information as well as noisy aspects out of questions text. The preprocessing pipeline consists of two preprocessing steps:

- Text cleaning
- Stop words removal

We should note here that the two preprocessing steps should be performed in the above order (text cleaning step followed by stop words removal step). This pipeline has been performed on training set and testing set as well as on new inputs from users. The output of the of preprocessing pipeline is a clean data ready to extract useful features out of it. Figure 3.2 illustrates preprocessing pipeline.

Figure 3- 2 The preprocessing pipeline

The final output of preprocessing pipeline is a clean text ready to feature extraction process

3.3.1 Text Cleaning

Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or

irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data (Wu, 2013).

Text cleaning process consists of five processes:

- Elongation removal
- Tashkil removal
- Punctuation removal
- Extra spaces removal
- Similar characters unifying

We should note that the order of these processes is not important as long as they are all performed on the textual data. Table 3-3 illustrates text cleaning process with examples (the underline marks the position where process takes place).

Table 3- 3 text cleaning sub-processes (the underline marks the position where process takes place)

Process	Text before the process	Text after the process
Elongation removal	الســـــلام عليكم	السلام عليكم
Tashkil removal	اليَومَ تَعلَمُ	اليوم تعلم
Punctuation removal	قال محمد: السلام عليكم	قال محمد السلام عليكم
Extra spaces removal	السلام___عليكم	السلام عليكم
Similar characters unifying	أنا أقول دوماً: علي الحق أتوكل	انا اقول دوما علي الحق اتوكل

3.3.2 Stop Words Removal

A stop word may be identified as a word that has the same likelihood of occurring in those documents not relevant to a query as in those documents relevant to the query (Wilbur & Sirotkin, 1992). Stop words are words that occur very frequent and do not contribute to the meaning of the sentence such as (حروف الجر) and (حروف العطف). Since stop words has no effect on the meaning much when they occur in a sentence they should be removed so that classifiers do not get biased by their occurrences.

A list of 297 stop words has been compiled. If any of these words existed in an input question it will be removed.

Example:

(ذهب محمد من المدرسة إلى البيت) Will be (ذهب محمد المدرسة البيت) after stop words removal.

3.7 FEATURE EXTRACTION

Feature extraction process takes place after preprocessing is done on the raw text data and done on the words resulting from preprocessing. Feature extraction has been implemented through a pipeline consists of the following processes:

- Term Frequency – Inverse Document Frequency (TF-IDF) vectorization
- Cosine similarity
- Feature normalization

As in preprocessing; feature extraction steps should done in order and the output of one step will be the input of the preceding step and they should

be performed on training set, testing set and new questions when arrived from users . Figure 3.3 illustrates feature extraction process.

Figure 3- 3 Feature extraction pipeline

3.7.1 TF-IDF Vectorization

TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. TF-IDF has been calculated for every word in the cleaned dataset.

The term frequency $tf(t,d)$ is the number of times a term occurs in a given document. If the word existed in a document its component will be filled by its TF-IDF value which is computed by the following equations:

$$tfidf(t,d) = tf(t,d) \times idf(t) \qquad (3\text{-}1)$$

Where $idf(t)$ is the inverse document-frequency and is given by:

$$idf(t) = \log\frac{1+m}{1+df(t)} \qquad (3\text{-}2)$$

where m is the total number of documents in the dataset, and $df(t)$ is the number of documents in the document set that contain term t. The resulting $tfidf$ vectors are then normalized by the Euclidean norm:

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2+v_2^2+\cdots+v_m^2}} \qquad (3\text{-}3)$$

Every sentence in the corpus then has been represented by a sparse vector of shape $1 \times n$ where n is the total number of words in the corpus. Every component of that vector represent a one of the words in the corpus.

3.7.2 Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. Cosine similarity computes the L2-normalized dot product of vectors. That is, if x and y are row vectors, their cosine similarity is defined as:

$$similarity = \cos(\theta) = \frac{x.y^T}{\|x\| \|y\|} \quad (3\text{-}4)$$

Each document will be transformed to a vector consists of all cosine similarities between the document and the other documents leaving us with $1 \times k$ dense vector where k is number of documents used as references to measure similarity. After cosine similarity is performed on the dataset the feature matrix will be of shape $m \times k$ where m is the number of training examples.

3.7.3 Features Normalization

Finally the $m \times k$ feature matrix has been normalized. Normalization is the process of scaling individual samples to have unit norm. this process is important to minimize the model convergence time.

3.7.4 Cosine Similarity Extraction Techniques

Cosine similarity vector has k dimensions where k is the number of examples used as a reference points to measure similarity and each component of the vector represent a similarity value between the example and one of the

reference examples. When cosine similarity has been performed on `init_set` dataset cosine similarity vector had m dimensions (i.e. $k = m$) where m is the size of the training set. However when more examples has been collected using the tester it was an opportunity to study the effect of similarity vector length on the performance of the classifiers. For this purpose, cosine similarity extraction done in two ways:

1. Similarity with 250 examples
2. Similarity with all examples

3.7.4.1 Similarity With 250 Examples

Here a value of $k = 250$ has been selected instead of $k = m$. This done by following steps: stratified sampling has been performed 1000 times, and each time the sampler function samples a sample 250 examples, preforms preprocessing steps illustrated earlier and count the number of words in the sample. After performing these calculations on all samples the sampler function selects the sample with the largest number of words to be used as reference examples for all examples in the `col_set` dataset (including those in the sample). This has resulted in a feature matrix of shape 751×250 named `vec_250`. Figure 3.5 illustrate the whole process.

3.7.4.2 Similarity With All Examples

Here $k = m$ has been selected as in the case of `init_set` dataset this resulted in as features matrix of shape 751×751 named `vec_all`.

3.8 MODELS TRAINING

3.8.1 Technique Used for Training

In machine learning tasks there are two types of parameters: one belongs to the machine learning model and they called just “parameters”, these parameters . The other type belongs to the algorithm that optimizes the model parameters and called hyperparameters, these parameters controls the behavior of the algorithm. To achieve high performance in the machine learning problem in hand, both types of parameters have to be optimized. These two types of parameters and the optimization processes related to them have been considered in the training technique used as it is a mix of two processes:

- Parameter optimization using k-fold cross validation
- Hyperparameter optimization using gradient search

This technique is called grid search cross validation or GridSearchCV for short.

Cross validation is the process of splitting training set iteratively to different training and validation set for further parameter optimization (more the just optimization done by training algorithm).

Each iteration the algorithm train a model on a subset of training set and test the accuracy of the model of the remaining part of the set.

Grid search is a method used to optimize algorithm hyperparameters by searching a space of generated from grid of candidate values.

In GridSearchCV, $\#folds \times \# \prod_{i=1}^{n} N_i$ models are trained where N_i is the number of candidate values of hyperparameter i. After training all models, the model with better performance will be picked.

3.8.2 The Training Process

Hence `init_set` has been created manually by the project team it was a small in size and lacked diversity, so the main intention behind its creation was to serve as a training set for the models that used to build the tester in order to make testing process interactive. These limitations imply that `init_set` cannot be divided into training and test sets and the only way to assess the performance of models built using it, is to wait for the data to be collected from testers.

Nine classification models has been trained on `init_set` these models are divided into four categories:

- Probabilistic models:
    - Gaussian naive Bayes
    - Complement naive Bayes
    - Multinomial naïve Bayes
- Non-generalizable models:
    - K-nearest neighbors
- Linear models:
    - Logistic regression with L2 regularization
    - Stochastic gradient decent
- Support Vector Machines (SVM) models:

- Support Vector Machines with polynomial kernel
- Support Vector Machines with RPF kernel
- Support Vector Machines with sigmoid kernel

3.8.3 Implementation of Voters

Voter classes offered in scikit-learn library has a fundamental limitation, it should all models contained within the voter should be of the same type and because this project has intention to use models with multiple types, it was mandatory to develop a voter class that provide this functionality and at the same time consistent with scikit-learn API specifications.

For the above reasons, a class called `Voter` has been implemented by inheriting BaseEstimator base class which provide the consistency with scikit-learn API and overwrite three methods: fit which controls the training process, predict which controls the way the voter makes its predictions and predict_proba which controls the way the model makes its probability estimates. The overridden of these three methods provide the required functionality that the original voter class lacks.

3.9 TESTING AND EVALUATION

All classifiers trained on the three training sets are tested using three performance metrics:

- Precision score
- Recall score
- F1 score

Four aspects of performance have been studied:

- The effect of training set size overall performance of each classifier.
- The effect of vectorization technique on the overall performance of each classifier.
- Differences in performance among classifiers trained on the same training set.
- Performance of all classifiers trained on all training sets on individual classes.

For more details about the results of testing please refer to chapter 4.

3.10 CLIENT SOFTWARE

The client software is not intended to perform any type of computations on the user input. Client software basicly responsible of performing two functions:

- Prvide a user-friendly interface for the user to interact with.
- Send and recive user input through the network.

The client software has been implemented to include two functional modules:

- User interface module
- Network interface module

3.10.1 User Interface Module

User interface module consists of the functions that displays and control various elements of the user interface as well as accepting input from user,

passing it to the network interface module, recive the answers from it and display the them to the user. The user interface elements has neen implemented with PyQt5 Library and and is consist of five main parts:

- A containing frame that holds all other elements
- An avatar of a girl to symbolize the chatbot in a humanized character.
- A line edit wedgit for the user to write questions in.
- A push button wedgit to send the message to the server after writing it.
- A text edit wedgit to display messages sent by the user.

Figure 3-4 shows the user interface of the client software

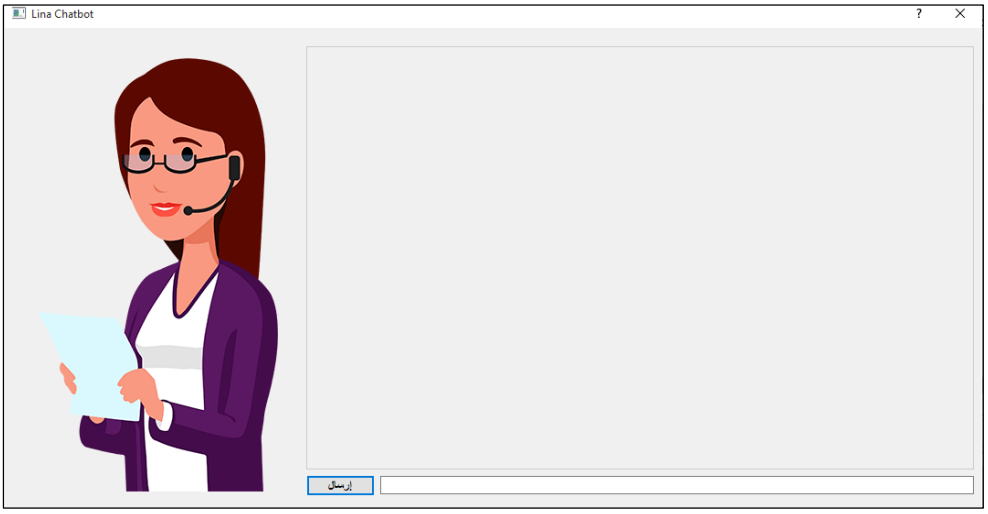

Figure 3- 4 User interface of the client software

3.10.2 Network Interface Module

Network interface module is responisble for handling communication between the client and the remote server. It has been implemented with socket API using stream communicaton paradigm in which the user inptut is converted to asequence of bytes before these bytes are sent through the network. It works by first prepare the user input rescived from the user

interface module by encode it with utf-8 and then adds information abouth the length of the message to be used by the server to buffer the message. When a message arrive from the server the network interface module first uses the length information to buffer it and then decodes it back to plain text and passes it to user interface tobe displayed to the user.

Figure 3-5 shows the block diagram of the client software.

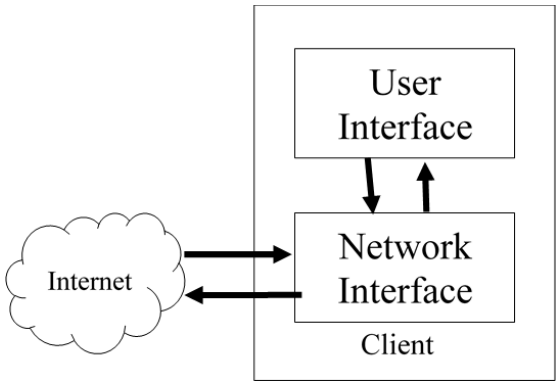

Figure 3- 5 block diagram of the client software

3.11 SERVER SOFTWARE

Server software consists of four modules:

- Network interface module
- Intent classifier module
- Dialog manager module
- Database interface module

Figure 3-6 show the block diagram of the server software.

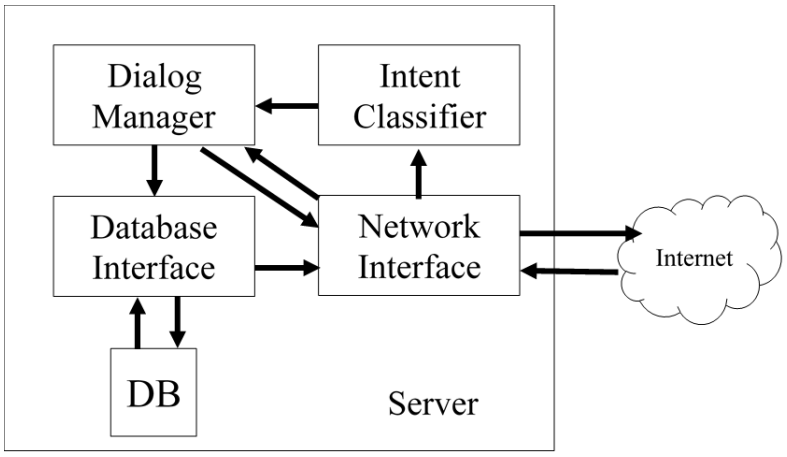

Figure 3- 6 block diagram of the server software

3.11.1 Network Interface Module:

As in the client software, network interface module in the server software is responsible for handling communication between the server and the remote client. It has also been implemented with the socket API using stream communicaton paradigm and works exactly the same way as network interface module in the client software. Each client will have two minutes to send questions before the network interface moduke ends its session and each time a clien sends a message its remaining time reset back to two minutes. It takes its input form the netwok and pass the output to both intent classifier and dialog manager.

3.11.2 Intent Classifier Module:

This module contains feature preprocessing pipeline, feature extraction pipeline and classification models. It is responsible for classifying user input text into one of the fifteen classes described in section 3.4 and then passing as an output the four most-voted classes to the dialog manager. Figure 3.7

illustrates the architecture of the intent classifier module, when an input arrives from a client it first gets preprocessed and then features are extracted using Feature extraction pipeline as described in section 3.7 and then the extracted features are fed to classification models to be classified.

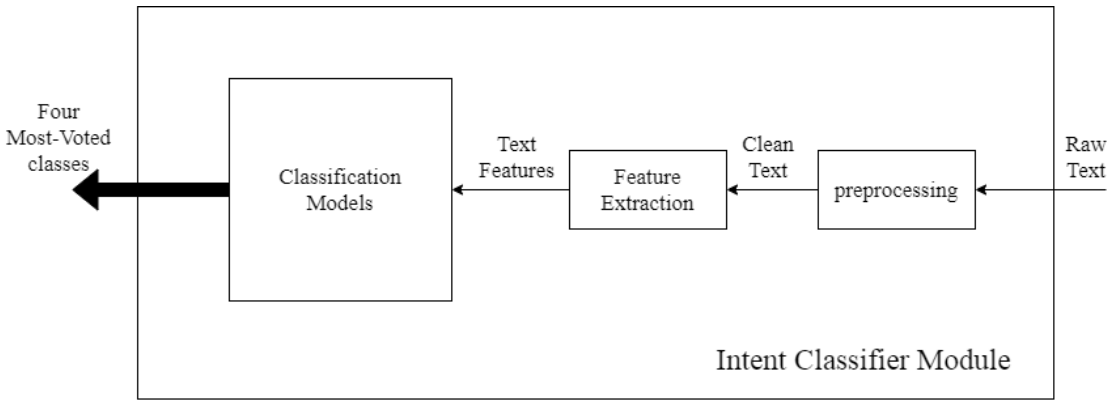

Figure 3- 7 the architecture of intent classifier module

Figure 3.8 illustrates the intent classification process. The extracted features first get fed to the nine classification models that were trained on `vec_all` dataset and have each model predict the class of the input, then these predictions are fed to a soft voter to make a soft-voted prediction based on the predictions of the individual models. Finally the individual predictions along with the soft-voted predictions are fed to a hard voter which outputs the for most-had-voted classes which will be used with the user input text as an input the dialog manager module.

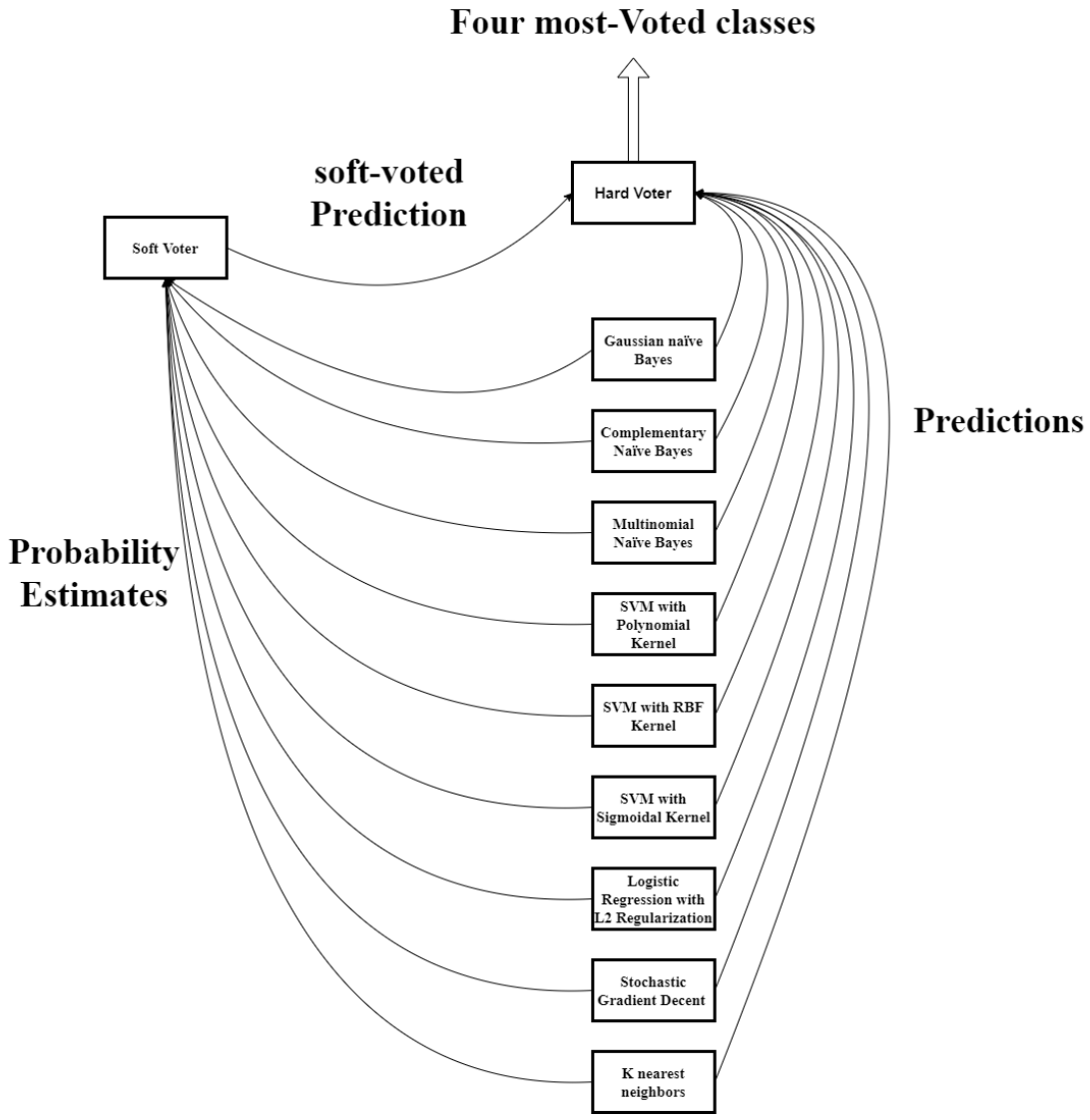

Figure 3- 8 block diagram representing the operation of intent classification process

3.11.3 Dialog Manager Module:

The dialog manager module represents the fabric that sewing the chatbot system as it incorporates all other modules to realize the goals of the chatbot system. It is responsible of providing the chatbot users with the information they need while taking them through a smooth dialog flow.

Numerous ways of dialog management are available in the literature. Here a dialog management approach based on clarification has picked. In this approach the chatbot asks the user for more information about his/her intent before providing the intended information. Instead of giving an answers directly, the chatbot provides the topics of answers for a user to choose from. These topics give this user a chance to assess how his request was understood.

The dialog manager module takes the four (or less) most-voted classes from intent classifier module and the user input text from the network interface module. The dialog between the user and the chatbot is assumed to have three states:

- Q state: in this state the chatbot is expecting the user to send a question.
- R1 state: in this state the chatbot is expecting the user to send an answer to a clarification question.
- R2 state: in this state the chatbot expecting the user to send an answer to a second clarification question.

Based on the candidate classes, the user input text and the dialog state, the chatbot responses are generated. The dialog manager receives four or less candidate classes from the intent classifier and divide the clarification about

them in two turns depending on the number of received candidates. Table 3-4 shows how the candidates are divided between the two turns.

Table 3- 4 number of candidate in clarification question in each turn

# candidate classes	Turn 1	Turn 2
1	1	0
2	2	0
3	1	2
4	2	2

Figure 3-9 shows the dialog management process in a form of a state diagram

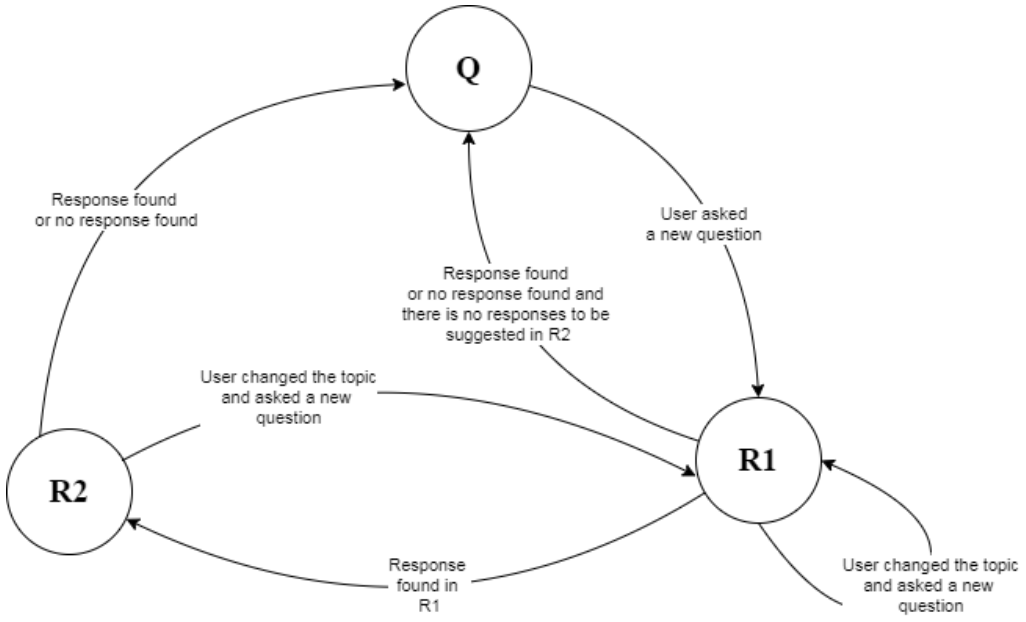

Figure 3- 9 a state diagram representing dialog management process

3.11.4 Database Interface Module

Database module is responsible of performing the following tasks:

- Load desired responses from the database and pass them to the network interface module.
- Store user queries received from network interface along with their corresponding predictions to the database.

CHAPTER (4)

RESULTS

4.1 INTRODUCTION

This chapter presents the results achieved on the text classification task as it represents the heart of the chatbot system.

The results have been divided into two parts:

- The overall performance of the classifiers.
- The performance of classifiers on various classes

4.2 OVERALL PERFORMANCE OF THE CLASSIFIERS

The overall performance of the classifiers has been measured using the macro average of each of the performance metrics mentioned the methodology chapter. To illustrate the overall performance in an understandable way, each classifier has been trained on the three datasets resulting in three classifiers of the same type. Then the performance of these three classifiers measured using metrics has plotted in a bar chart with the percentage values of metrics.

This section is divided into two subsections; the aim of the first subsection (section 4.2.1) is to illustrate the impact of the number of reference vectors used to calculate cosine similarity and the size of training vocabulary on the overall performance of the classifiers. The second subsection (section 4.2.2) compares the overall performance the classifiers on Arabic text classification task.

4.2.1 Dataset Size and Vectorization Method

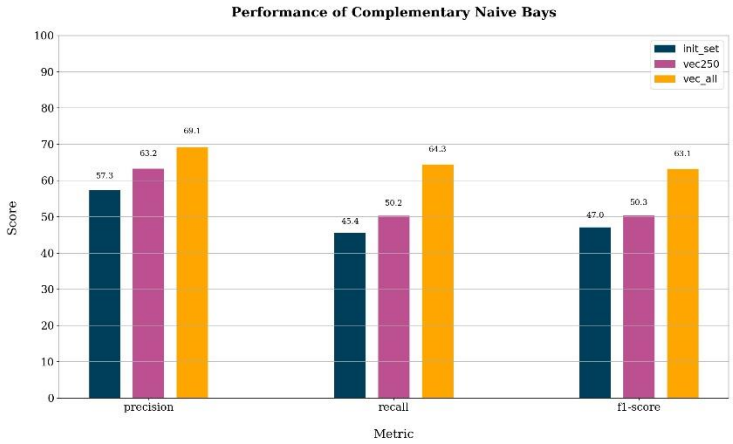

Figure 4 - 1 performance of complementary naive Bays trained with the three datasets

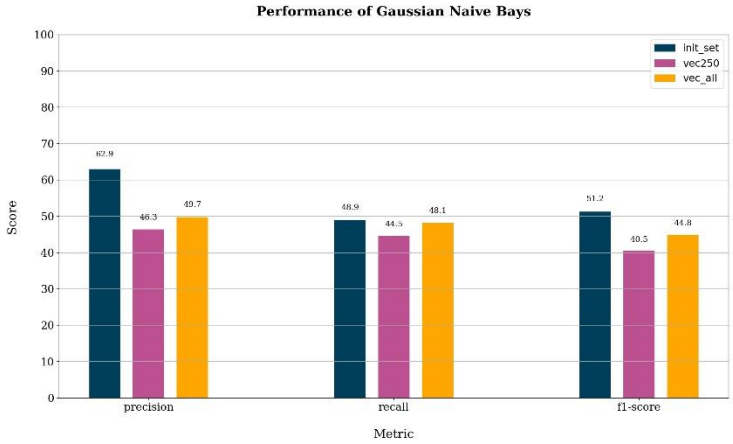

Figure 4 - 2 performance of Gaussian naive Bays trained with the three datasets

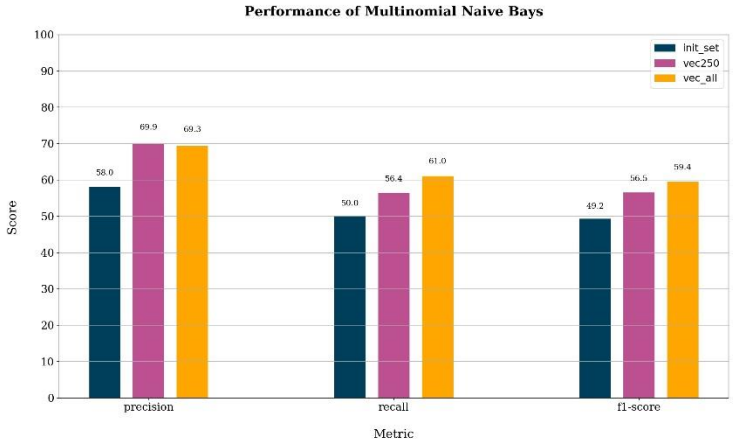

Figure 4 - 3 performance of multinomial naive Bays trained with the three datasets

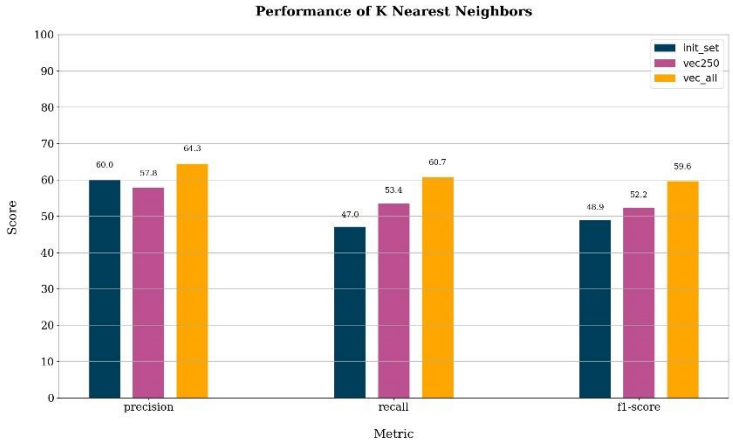

Figure 4 - 4 performance of K nearest neighbors trained with the three datasets

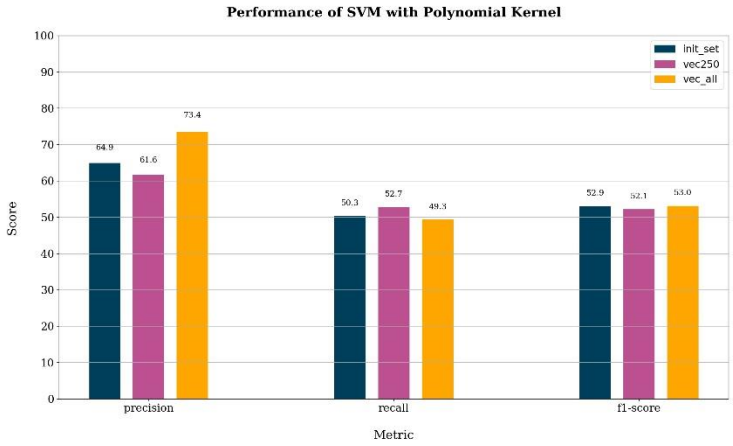

Figure 4 - 5 performance of SVM classifier with polynomial kernel trained with the three datasets

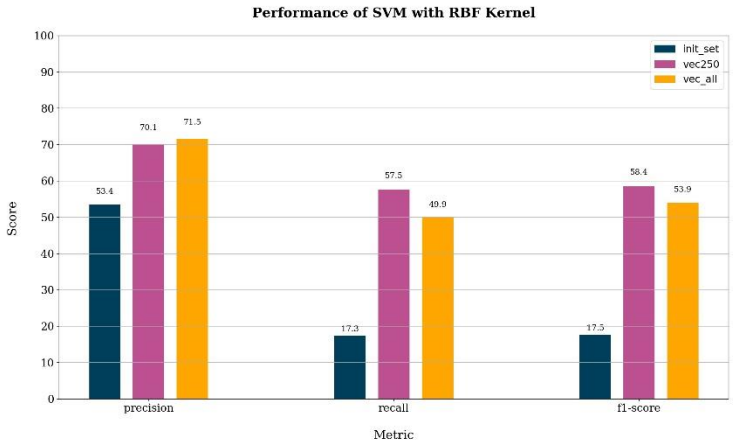

Figure 4 - 6 performance of SVM classifier with RBF kernel trained with the three datasets

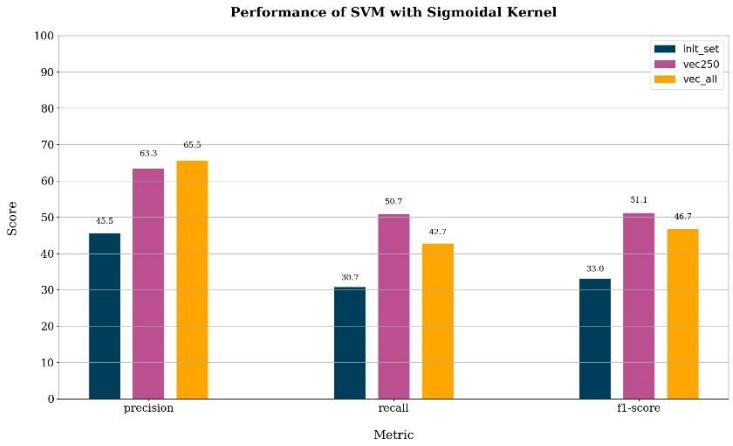

Figure 4 - 7 performance of SVM classifier with sigmoidal kernel on the three datasets

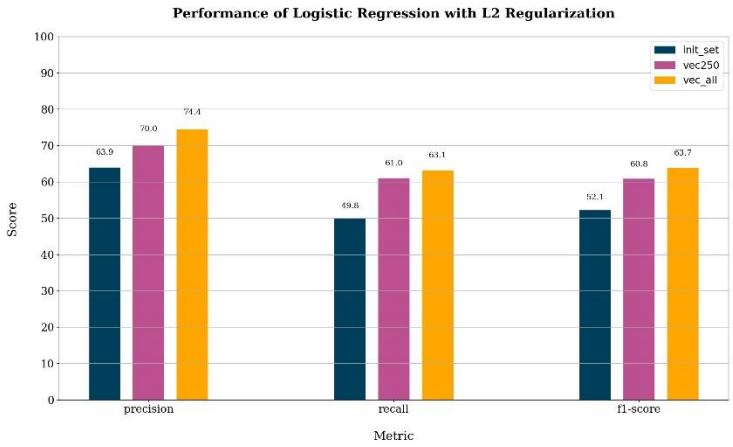

Figure 4 - 8 performance of logistic regression classifier with L2 regularization trained with the three datasets

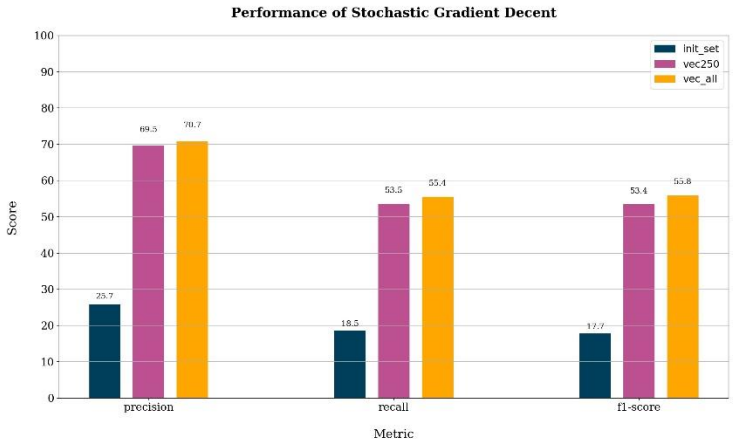

Figure 4 - 9 performance of stochastic gradient decent classifier trained with the three datasets

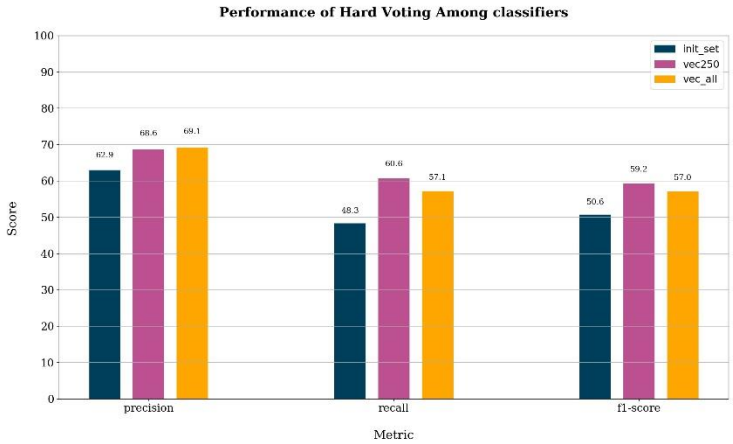

Figure 4 - 10 performance of hard voter with classifiers trained with the three datasets

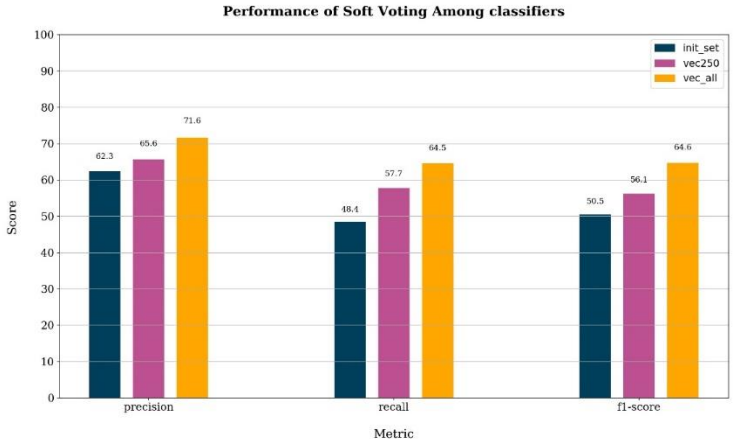

Figure 4 - 11 performance of soft voter with classifiers trained with the three datasets

4.2.2 The Comparison between Classifiers Performance:

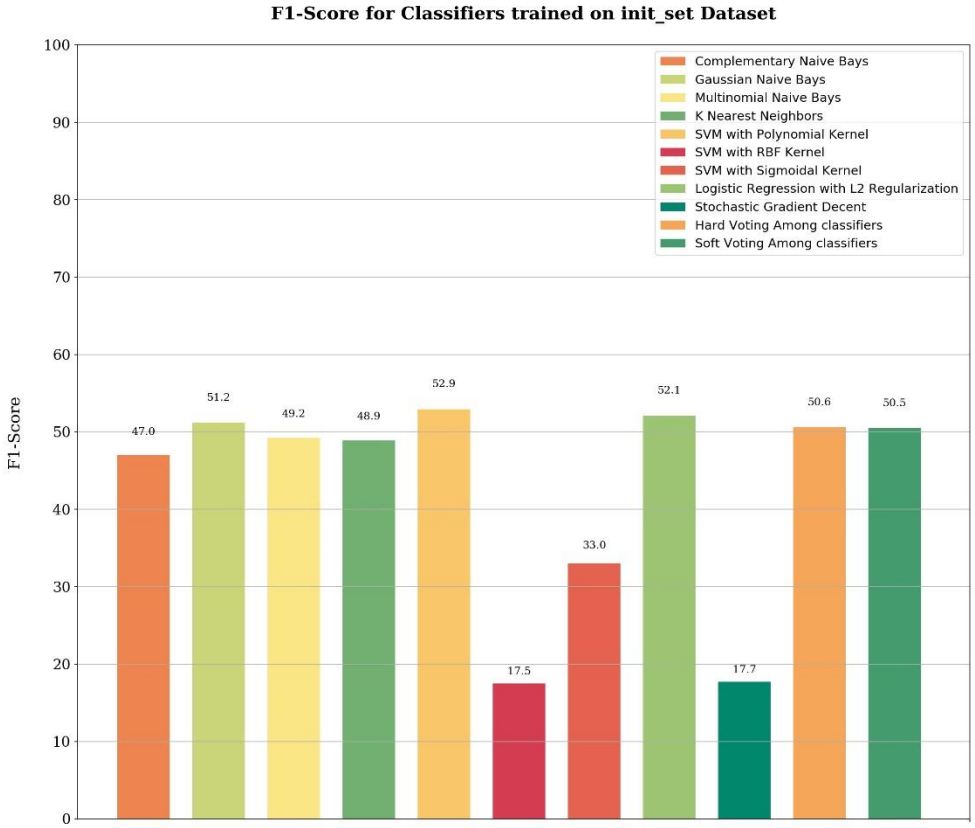

Figure 4 - 12 performance of various classifiers traind on `init_set` dataset

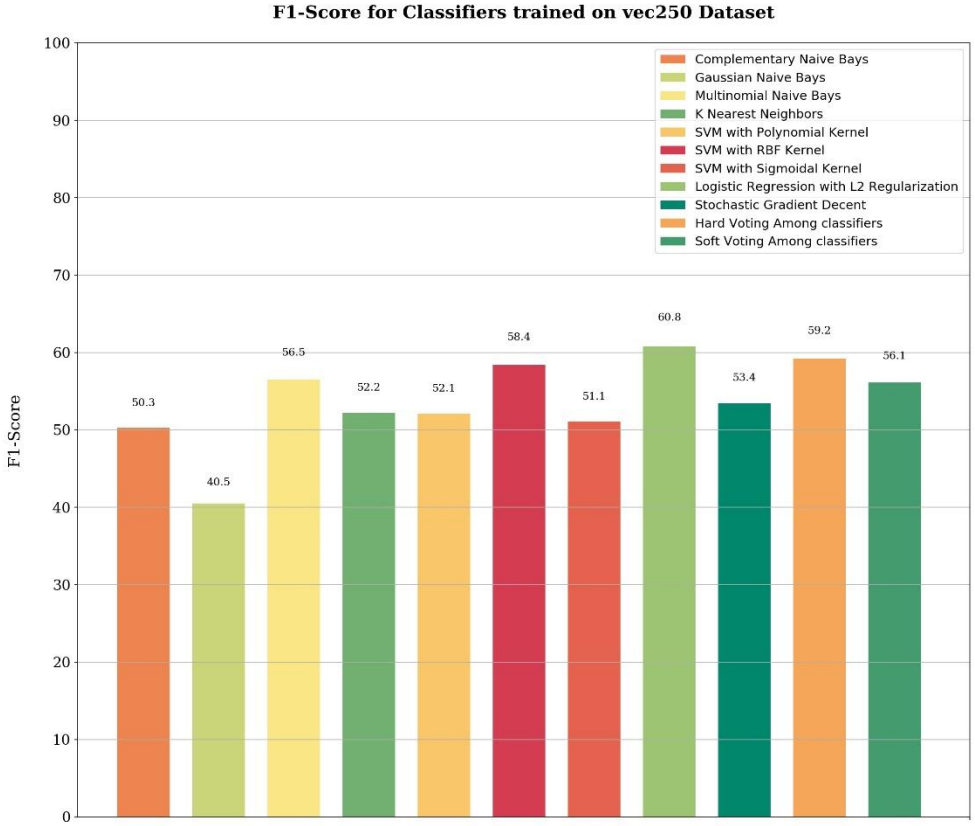

Figure 4 - 13 performance of various classifiers traind on `vec250` dataset

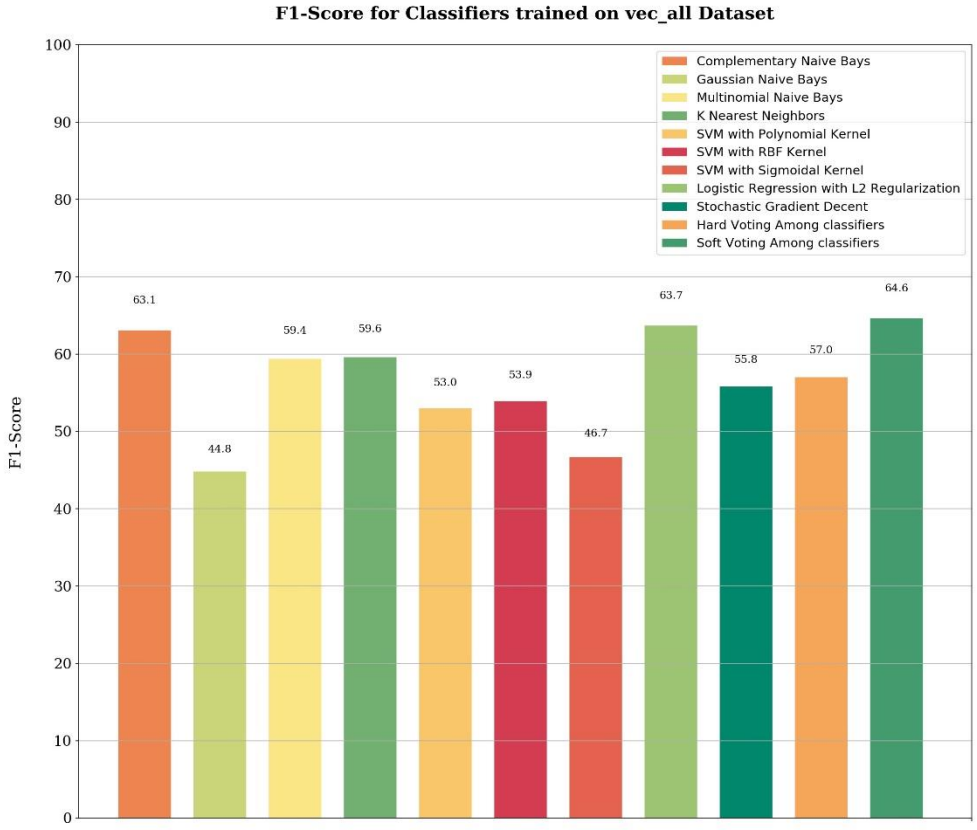

Figure 4 - 14 performance of various classifiers traind on `vec_all` dataset

4.3 PERFORMANCE ON INDIVIDUAL CLASSES

The performance of the classifiers on individual is presented in a tabular form

Table 4 - 1 precision scores of classifiers trained on `init_set` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	19.4	75	23.1	75	57.1	77.8	100	0	50	75	77.8
asymptotic_infection	100	100	100	100	100	100	100	0	40	66.7	100
end_conversation	75	75	75	75	75	75	100	0	66.7	75	75
life_time	60	60	60	60	60	60	0	66.7	50	60	60
mailing_packages	85.7	85.7	85.7	85.7	85.7	85.7	100	0	50	85.7	85.7
mask_usage	92.3	84.6	92.3	92.3	85.7	92.3	100	76.9	71.4	92.3	92.3
period_before_symptoms	66.7	66.7	75	75	66.7	75	100	85.7	60	75	75
sanitizers	69.2	64.3	69.2	75	69.2	75	100	57.1	50	69.2	69.2
second_infection	66.7	71.4	85.7	85.7	66.7	85.7	100	75	66.7	85.7	85.7
self_protection	62.5	55.6	70	25	62.5	25	100	0	17.1	22.9	70
start_conversation	100	100	100	100	39.4	100	14	0	88.9	100	100
symptoms	50	66.7	55.6	55.6	50	55.6	75	0	0	55.6	55.6
temprature	66.7	16.1	66.7	66.7	66.7	66.7	100	6.2	75	66.7	16.1
test	85.7	85.7	100	100	100	100	100	0	66.7	100	100
ways_of_infection	38.1	42.1	57.1	40	40	42.1	100	100	40	42.1	44.4

Table 4 - 2 recall scores of classifiers trained on `init_set` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	66.7	66.7	100	66.7	44.4	77.8	33.3	0	55.6	66.7	77.8
asymptotic_infection	33.3	66.7	33.3	33.3	33.3	33.3	33.3	0	66.7	33.3	50
end_conversation	50	50	50	50	50	50	50	0	33.3	50	50
life_time	37.5	37.5	37.5	37.5	37.5	37.5	0	50	25	37.5	37.5
mailing_packages	85.7	85.7	85.7	85.7	85.7	85.7	57.1	0	28.6	85.7	85.7
mask_usage	80	73.3	80	80	80	80	33.3	66.7	66.7	80	80
period_before_symptoms	75	75	75	75	75	75	62.5	75	37.5	75	75
sanitizers	81.8	81.8	81.8	81.8	81.8	81.8	45.5	36.4	63.6	81.8	81.8
second_infection	85.7	71.4	85.7	85.7	85.7	85.7	57.1	42.9	57.1	85.7	85.7
self_protection	27.8	27.8	38.9	50	27.8	50	27.8	0	33.3	44.4	38.9
start_conversation	57.1	57.1	57.1	57.1	92.9	57.1	100	0	57.1	57.1	57.1
symptoms	40	40	50	50	40	50	30	0	0	50	50
temprature	36.4	45.5	36.4	36.4	36.4	36.4	18.2	63.6	27.3	36.4	45.5
test	66.7	66.7	66.7	66.7	66.7	66.7	55.6	0	44.4	66.7	66.7
ways_of_infection	66.7	66.7	66.7	66.7	66.7	66.7	33.3	16.7	33.3	66.7	66.7

Table 4 - 3 F1 scores of classifiers trained on `init_set` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	30	70.6	37.5	70.6	50	77.8	50	0	52.6	70.6	77.8
asymptotic_infection	50	80	50	50	50	50	50	0	50	44.4	66.7
end_conversation	60	60	60	60	60	60	66.7	0	44.4	60	60
life_time	46.2	46.2	46.2	46.2	46.2	46.2	0	57.1	33.3	46.2	46.2
mailing_packages	85.7	85.7	85.7	85.7	85.7	85.7	72.7	0	36.4	85.7	85.7
mask_usage	85.7	78.6	85.7	85.7	82.8	85.7	50	71.4	69	85.7	85.7
period_before_symptoms	70.6	70.6	75	75	70.6	75	76.9	80	46.2	75	75
sanitizers	75	72	75	78.3	75	78.3	62.5	44.4	56	75	75
second_infection	75	71.4	85.7	85.7	75	85.7	72.7	54.5	61.5	85.7	85.7
self_protection	38.5	37	50	33.3	38.5	33.3	43.5	0	22.6	30.2	50
start_conversation	72.7	72.7	72.7	72.7	55.3	72.7	24.6	0	69.6	72.7	72.7
symptoms	44.4	50	52.6	52.6	44.4	52.6	42.9	0	0	52.6	52.6
temprature	47.1	23.8	47.1	47.1	47.1	47.1	30.8	11.4	40	47.1	23.8
test	75	75	80	80	80	80	71.4	0	53.3	80	80
ways_of_infection	48.5	51.6	61.5	50	50	51.6	50	28.6	36.4	51.6	53.3

Table 4 - 4 precision scores of classifiers trained on `vec250` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	12.8	37.5	50	63.6	60	75	60	60	71.4	50	53.8
asymptotic_infection	50	75	100	100	100	100	100	75	100	55.6	100
end_conversation	75	10.8	11.8	50	75	27.3	75	7.8	30	75	11.8
life_time	42.9	40	75	66.7	42.9	66.7	66.7	50	66.7	75	75
mailing_packages	80	28.6	62.5	83.3	80	83.3	80	66.7	80	80	62.5
mask_usage	77.8	85.7	70	75	77.8	85.7	77.8	87.5	87.5	77.8	75
period_before_symptoms	100	50	75	100	100	0	100	100	0	100	80
sanitizers	55.6	50	72.7	66.7	66.7	63.6	70	75	63.6	70	70
second_infection	54.5	45.5	54.5	54.5	50	55.6	45.5	40	36.4	50	58.3
self_protection	54.5	44.4	54.5	64.3	54.5	50	42.9	50	42.9	46.2	42.9
start_conversation	100	33.3	38.5	37.8	100	28.6	38.9	100	26.4	38.9	71.4
symptoms	50	60	40	50	18.4	42.9	50	83.3	75	62.5	50
temprature	100	66.7	0	100	100	100	100	100	100	100	100
test	53.8	66.7	100	87.5	77.8	85.7	100	71.4	100	87.5	77.8
ways_of_infection	41.7	0	62.5	50	46.2	60	45.5	75	70	60	55.6

Table 4 - 5 recall scores of classifiers trained on `vec250` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	55.6	66.7	77.8	77.8	66.7	66.7	66.7	66.7	55.6	77.8	77.8
asymptotic_infection	50	100	50	83.3	66.7	83.3	66.7	100	83.3	83.3	83.3
end_conversation	50	66.7	66.7	50	50	50	50	66.7	50	50	66.7
life_time	75	25	75	50	75	50	75	75	50	75	75
mailing_packages	57.1	57.1	71.4	71.4	57.1	71.4	57.1	57.1	57.1	57.1	71.4
mask_usage	46.7	40	46.7	40	46.7	40	46.7	46.7	46.7	46.7	40
period_before_symptoms	50	50	37.5	37.5	50	0	37.5	50	0	50	50
sanitizers	45.5	18.2	72.7	72.7	72.7	63.6	63.6	54.5	63.6	63.6	63.6
second_infection	85.7	71.4	85.7	85.7	85.7	71.4	71.4	57.1	57.1	85.7	100
self_protection	33.3	22.2	33.3	50	33.3	38.9	50	11.1	33.3	33.3	33.3
start_conversation	35.7	35.7	35.7	100	35.7	100	100	28.6	100	100	35.7
symptoms	40	30	40	50	70	30	50	50	30	50	40
temprature	9.1	18.2	0	18.2	9.1	9.1	9.1	9.1	9.1	9.1	9.1
test	77.8	66.7	66.7	77.8	77.8	66.7	77.8	55.6	66.7	77.8	77.8
ways_of_infection	41.7	0	41.7	50	50	50	41.7	75	58.3	50	41.7

Table 4 - 6 F1scores of classifiers trained on `vec250` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	20.8	48	60.9	70	63.2	70.6	63.2	63.2	62.5	60.9	63.6
asymptotic_infection	50	85.7	66.7	90.9	80	90.9	80	85.7	90.9	66.7	90.9
end_conversation	60	18.6	20	50	60	35.3	60	14	37.5	60	20
life_time	54.5	30.8	75	57.1	54.5	57.1	70.6	60	57.1	75	75
mailing_packages	66.7	38.1	66.7	76.9	66.7	76.9	66.7	61.5	66.7	66.7	66.7
mask_usage	58.3	54.5	56	52.2	58.3	54.5	58.3	60.9	60.9	58.3	52.2
period_before_symptoms	66.7	50	50	54.5	66.7	0	54.5	66.7	0	66.7	61.5
sanitizers	50	26.7	72.7	69.6	69.6	63.6	66.7	63.2	63.6	66.7	66.7
second_infection	66.7	55.6	66.7	66.7	63.2	62.5	55.6	47.1	44.4	63.2	73.7
self_protection	41.4	29.6	41.4	56.2	41.4	43.7	46.2	18.2	37.5	38.7	37.5
start_conversation	52.6	34.5	37	54.9	52.6	44.4	56	44.4	41.8	56	47.6
symptoms	44.4	40	40	50	29.2	35.3	50	62.5	42.9	55.6	44.4
temprature	16.7	28.6	0	30.8	16.7	16.7	16.7	16.7	16.7	16.7	16.7
test	63.6	66.7	80	82.4	77.8	75	87.5	62.5	80	82.4	77.8
ways_of_infection	41.7	0	50	50	48	54.5	43.5	75	63.6	54.5	47.6

Table 4 - 7 precison scores of classifiers trained on `vec_all` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	30.4	36.8	53.8	100	63.6	100	100	55.6	100	30	60
asymptotic_infection	100	80	100	100	100	100	100	100	100	37.5	100
end_conversation	50	20	37.5	18.7	50	9.2	9.4	8.1	7.9	14.8	16
life_time	54.5	25	71.4	66.7	40	66.7	66.7	54.5	66.7	57.1	57.1
mailing_packages	85.7	27.8	54.5	85.7	85.7	80	80	71.4	75	85.7	85.7
mask_usage	90.9	66.7	71.4	91.7	90	100	100	87.5	100	90	84.6
period_before_symptoms	83.3	44.4	100	100	100	0	0	100	0	100	80
sanitizers	57.1	62.5	53.3	61.5	57.1	100	100	50	87.5	77.8	61.5
second_infection	46.7	50	45.5	75	46.7	50	50	41.7	66.7	46.2	50
self_protection	70	27.3	57.1	71.4	63.6	62.5	66.7	70	62.5	66.7	66.7
start_conversation	87.5	70	65	100	87.5	100	100	83.3	100	100	77.8
symptoms	75	60	60	55.6	28.6	66.7	66.7	100	60	71.4	75
temprature	75	41.7	50	80	100	100	66.7	100	0	100	100
test	88.9	83.3	100	60	87.5	100	100	77.8	100	100	100
ways_of_infection	41.2	50	44.4	50	38.9	66.7	66.7	60	55.6	60	60

Table 4 - 8 recall scores of classifiers trained on `vec_all` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	77.8	77.8	77.8	66.7	77.8	44.4	44.4	55.6	44.4	66.7	66.7
asymptotic_infection	66.7	66.7	50	50	66.7	50	50	83.3	50	50	83.3
end_conversation	33.3	66.7	50	100	33.3	100	100	50	100	66.7	66.7
life_time	75	12.5	62.5	50	75	50	50	75	50	50	50
mailing_packages	85.7	71.4	85.7	85.7	85.7	57.1	57.1	71.4	42.9	85.7	85.7
mask_usage	66.7	53.3	66.7	73.3	60	60	60	46.7	40	60	73.3
period_before_symptoms	62.5	50	37.5	37.5	50	0	0	37.5	0	37.5	50
sanitizers	72.7	45.5	72.7	72.7	72.7	72.7	72.7	63.6	63.6	63.6	72.7
second_infection	100	71.4	71.4	85.7	100	28.6	28.6	71.4	28.6	85.7	100
self_protection	38.9	16.7	44.4	55.6	38.9	55.6	55.6	38.9	55.6	55.6	55.6
start_conversation	50	50	92.9	50	50	50	50	35.7	50	50	50
symptoms	60	30	60	50	60	40	40	40	30	50	60
temprature	27.3	45.5	27.3	36.4	9.1	9.1	18.2	9.1	0	18.2	36.4
test	88.9	55.6	77.8	66.7	77.8	55.6	55.6	77.8	44.4	66.7	66.7
ways_of_infection	58.3	8.3	33.3	66.7	58.3	66.7	66.7	75	41.7	50	50

Table 4 - 9 F1 scores of classifiers trained on `vec_all` dataset

class	Complementary Naive Bays	Gaussian Naive Bays	K Nearest Neighbors	Logistic Regression with L2 Regularization	Multinomial Naive Bays	SVM with Polynomial Kernel	SVM with RBF Kernel	Stochastic Gradient Decent	SVM with Sigmoidal Kernel	Hard Voting Among classifiers	Soft Voting Among classifiers
animals_infection	43.7	50	63.6	80	70	61.5	61.5	55.6	61.5	41.4	63.2
asymptotic_infection	80	72.7	66.7	66.7	80	66.7	66.7	90.9	66.7	42.9	90.9
end_conversation	40	30.8	42.9	31.6	40	16.9	17.1	14	14.6	24.2	25.8
life_time	63.2	16.7	66.7	57.1	52.2	57.1	57.1	63.2	57.1	53.3	53.3
mailing_packages	85.7	40	66.7	85.7	85.7	66.7	66.7	71.4	54.5	85.7	85.7
mask_usage	76.9	59.3	69	81.5	72	75	75	60.9	57.1	72	78.6
period_before_symptoms	71.4	47.1	54.5	54.5	66.7	0	0	54.5	0	54.5	61.5
sanitizers	64	52.6	61.5	66.7	64	84.2	84.2	56	73.7	70	66.7
second_infection	63.6	58.8	55.6	80	63.6	36.4	36.4	52.6	40	60	66.7
self_protection	50	20.7	50	62.5	48.3	58.8	60.6	50	58.8	60.6	60.6
start_conversation	63.6	58.3	76.5	66.7	63.6	66.7	66.7	50	66.7	66.7	60.9
symptoms	66.7	40	60	52.6	38.7	50	50	57.1	40	58.8	66.7
temprature	40	43.5	35.3	50	16.7	16.7	28.6	16.7	0	30.8	53.3
test	88.9	66.7	87.5	63.2	82.4	71.4	71.4	77.8	61.5	80	80
ways_of_infection	48.3	14.3	38.1	57.1	46.7	66.7	66.7	66.7	47.6	54.5	54.5

In terms of overall performance of the classifiers on Arabic text classification task, results showed that Gaussian naïve Bayes, logistic regression with L2 regularization and soft voting techniques has better performance than others.

Coming to the effect of training set size on classifier's performance, stochastic gradient decent and SVM techniques in general showed significant improvements in performance as the size of the training set increases, whereas other classifiers showed smaller changes in performance.

The effect of vectorization method has been noticed obviously in the results, as the performance of all classifiers improves with the number of vectors used to measure similarity.

In general, more samples need to be collected for all classes, but some classes on which classifiers have very poor performance comparing to others and need more samples to be collected, these classes are:

- `period_before_symptoms`
- `self_protection`
- `start_conversation`
- `Temprature`

CHAPTER (5)

CONCLUSION AND RECOMMANDATIONS

5.1 CONCLUSION

In this project an Arabic customer-service chatbot has been designed and implemented to serve as an automated answering system for the frequently asked questions concerning COVID-19 pandemic. The chatbot has been deployed as client-server system and the system has been written entirely in python programming language.

The intent classifier module which resides in the server side has been implemented by training the following nine classifiers:

- Gaussian naive Bayes classifier
- Complement naive Bayes classifier
- Multinomial naïve Bayes classifier
- K-nearest-neighbors classifier
- Logistic regression classifier with L2 regularization
- Stochastic gradient decent classifier
- Support Vector Machines classifier with polynomial kernel
- Support Vector Machines classifier with RPF kernel
- Support Vector Machines classifier with sigmoid kernel

The training has been done on the following three datasets:

- `init_set` dataset with 250 examples and vectorized with 250 vectors

- `vec250` dataset with 600 examples and vecorized with 250 vectors
- `vec_all` dataset with 600 examples and vectorized with 600 vectors

and their results has been aggregated in two ways:

- Hard voting
- Soft voting

When it is provided with an example each one of the classifiers classifies the examples in into one of the following fifteen target classes:

- `animals_infection`
- `asymptotic_infection`
- `end_conversation`
- `life_time`
- `mailing_packages`
- `mask_usage`
- `period_before_symptoms`
- `Sanitizers`
- `second_infection`
- `self_protection`
- `start_conversation`
- `Symptoms`
- `Temprature`

- `Test`
- `ways_of_infection`

The dialog manager has been designed with clarification-based method which relies on asking user to clarify its intent and based on his/her answers the dialog will flow.

The server is also provided with a database in order to store user questions and models predictions for purposes of testing and maintenance. The system as whole has been built to facilitate system upgrade by replacing old components such as preprocessing and feature extraction pipelines with new ones without needing to rewrite a single line of code given that the target classes are the same.

A comparative study has been conducted on the performance of each classifier under different feature extraction techniques and dataset sizes and the results has concluded that performance increases with the size of the training set and the number of vectors used for vectorization.

5.2 RECOMMENDATIONS

From the work that has done on this project the following recommendations are suggested for future researchers who has intention to work in Arabic chatbots or any other projects involving Arabic text classification:

- There is a scarcity in Arabic textual data, so it is important to develop powerful techniques to collect data that would enable researchers to develop their models.

- It is also on demand to develop or adopt developed preprocessing techniques, feature extraction tools and performance metrics for Arabic language and have them implemented as a one library that is able to work with python programming language as it is the defacto language of machine learning.
- Till now, no research work has been done in the area of Arabic generative-based chatbots. If in future there was abundance of Arabic textual data it is better use recurrent neural networks to develop generative-based chatbot for Arabic language.
- As the results have showed, when working on Arabic text classification, it is highly recommended to use support vector machines, logistic regression with L2 regularization or an ensemble method on these two techniques first before moving to other classifiers.
- Also from the concluded results, calculating cosine similarity between each document and the entire training set and not only on a part of the training set is preferable as it magnifies the performance gains.
- In this project, no information regarding individual words has been used to extract features from text. We recommend, however, to make use of word embbedings, which better represent the semantics of words.
- On this project only the user intents has been used in the chatbot operation, we recommend for future work to incorporate both the intents and entities in the user input for more accurate results.
- In this project no means of spelling correction is provided when users make typos it will affect the quality of predictions, so it is

recommended for future project to incorporate a spelling correction mechanism to avoid false predictions.

References

1- Agarwal, S. et al., 2020. Unleashing the power of disruptive and emerging technologies amid COVID-19: A detailed review. *Journal of Applied soft Computing.*

2- AlHumoud, S., Al Wazrah, A. & Aldamegh, W., 2018. Arabic Chatbots: A Survey. *International Journal of Advanced Computer Science and Applications*, 9(8), pp. 535-541.

3- Ali, D. & Habash, N., 2016. Botta: An arabic dialect chatbot. *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations* , pp. 208-212.

4- Aljameel, S. et al., 2017. Development of an Arabic conversational intelligent tutoring system for education of children with ASD. *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)* , pp. 24-29.

5- Alobaidi, O., Crockett, K., O'Shea, J. & Jarad, T., 2013. Abdullah: An intelligent arabic conversational tutoring system for modern islamic education. *Proceedings of the World Congress on Engineering* , Volume 2.

6- Bentley, J., 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM.*

7- bhattacharjee, j., 2017. *Some Key Machine Learning Definitions.* [Online]
Available at: https://medium.com/technology-nineleaps/some-key-machine-learning-definitions-b524eb6cb48
[Accessed 18 September 2020].

8- Bishop, C., 2006. *Pattern recognition and machine learning.* Berlin: Springer.

9- Brandtzaeg, P. & Følstad, A., 2017. *Why People Use Chatbots.* Cham, Springer.

10- Brini, W., Ellouze, M., Mesfar, S. & Belguith, L., 2009. An Arabic Question-Answering system for factoid questions. *International Conference on Natural Language Processing and Knowledge Engineering,* pp. 1-7.

11- cdc.gov, n.d. *1918 Pandemic (H1N1 virus).* [Online]
Available at: https://www.cdc.gov/flu/pandemic-resources/1918-pandemic-h1n1.html
[Accessed 30 June 2020].

12- Contreras, I. & Vehí, J., 2018. Artificial Intelligence for Diabetes Management and Decision Support: Literature Review. *Journal of Medical Internet Research.*

13- Cortes, C. & Vapnik, V., 1995. Support-vector networks. *Machine Learning,* Volume 20, pp. 273-297.

14- Doherty , D., 2018. *Task-based Interaction Chatbot*, s.l.: s.n.

15- Géron, A., 2019. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems.

16- Guyon, I., Boser, B. & Vapnik, V., 1993. Automatic Capacity Tuning of Very Large VC-dimension Classifiers. *Advances in neural information processing.*

17- Hijjawi, M., 2011. ArabChat: an Arabic conversational agent. *Doctoral dissertation, The Manchester Metropolitan University.*

18- Hijjawi, M., Bandar, Z. & Crockett, K., 2013. User's utterance classification using machine learning for Arabic Conversational Agents. *The 5th International Conference on Computer Science and Information Technology*, pp. 223-232.

19- Hijjawi, M., Bandar, Z. & Crockett, K., 2016. The Enhanced Arabchat: An Arabic Conversational Agent.. *International Journal of Advanced Computer Science and Applications,* Volume 7.

20- Hijjawi, M., Qattous, H. & Alsheiksalem, 2015. Mobile Arabchat: An Arabic Mobile-Based Conversational Agent. *Int. J. Adv. Comput. Sci. Appl. IJACSA,* 6(10).

21- Igi-global.com, 2020. *What Is Model Training | IGI Global. [online] Available.* [Online]
Available at: https://www.igi-global.com/dictionary/model-training/19072
[Accessed 18 September 2020].

22- Jaafar, Y. & Bouzoubaa , K., 2015. *Arabic Natural Language Processing from Software Engineering to Complex Pipelines.* s.l., inetrnational Conference on Arabic Computational Linguistics.

23- McKinney, W., 2020. Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference,* Volume 445, pp. 51-56.

24- monkeylearn, n.d. *Guide to textclassification.* [Online]
Available at: https://monkeylearn.com/text-classification/

25- Mordue, P. & Burton, T., 2020. *Delivering customer service during COVID-19.* [Online]
Available at: https://www2.deloitte.com/uk/en/pages/digital-transformation/articles/delivering-customer-service-during-COVID-19.html
[Accessed 2 November 2020].

26- Oliphant, T., 2006. A guide to NumPy. Volume 1, p. 85.

27- Omohundro, S., 1989. Five balltree construction algorithms. *International Computer Science Institute Technical Report.*

28- Pandorabots, 2018. Rosie: Base content for AIML 2.0 chatbot. *Pandorabots.*

29- Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python.. *The Journal of machine Learning research,* Volume 12, pp. 2825-2830.

30- PyQt, 2012. PyQt Reference Guide.

31- Quantdare.com, 2019. *What Is The Difference Between Feature Extraction And Feature Selection?.* [Online]
Available at: https://quantdare.com/what-is-the-difference-between-feature-extraction-and-feature-selection
[Accessed 18 September 2020].

32- Rennie, J. D., Shih, L., Teevan, J. & Karger, D. R., 2003. Tackling the poor assumptions of naive bayes text classifiers. *ICML* , Volume 3, pp. 616-623.

33- Shawar, A. & Atwell, E., 2004. Arabic chatbot giving answers from the Qur'an. *Proceedings of TALN04: XI Conference sur le Traitement Automatique des Langues Naturelles,* Volume 2, pp. 197-202.

34- Shawar, B., 2011. A Chatbot as a natural web Interface to Arabic web QA. *International Journal of Emerging Technologies in Learning (iJET),* 6(1), pp. 37-43.

35- Shawar, B. & Atwell, E., 2007. Chatbots: Are they Really Useful?. *LDV Forum,* January, Volume 22, pp. 29-49.

36- Techopedia.com, 2020. *What is Data Preprocessing? - Definition from Techopedia.* [Online]
Available at: https://www.techopedia.com/definition/14650/data-preprocessing
[Accessed 18 September 2020].

37- United Nations, 2020. *أسئلة شائعة عن مرض فيروس كورونا / الأمم المتحدة.* [Online]
Available at: https://www.un.org/ar/coronavirus/covid-19-faqs
[Accessed 5 June 2020].

38- Virtanen, P. et al., 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods.*

39- Wallace, R., 2003. *The elements of AIML style,* s.l.: Alice AI Foundation.

40- Walt, S., Colbert, S. & Varoquaux, G., 2011. The NumPy array: a structure for efficient numerical computation. Computing in science & engineering. 13(12), pp. 22-30.

41- Waseem, M., 2020. *How To Implement Classification In Machine Learning?.* [Online]
Available at: https://www.edureka.co/blog/classification-in-machine-learning/#:~:text=MNIST%20Digit%20Classification-,What%20is%20Classification%20In%20Machine%20Learning,as%20targe

t%2C%20label%20or%20categories.
[Accessed 8 November 2020].

42- WHO, n.d. *Coronavirus.* [Online]
Available at: https://www.who.int/health-topics/coronavirus
[Accessed 30 6 2020].

43- Wilbur, W. & Sirotkin, K., 1992. The automatic identification of stop words. *Journal of information science,* 18(1), pp. 45-55.

44- Wu, S., 2013. A review on coarse warranty data and analysis. *Reliability Engineering and System,* Volume 114, pp. 1-11.

45- Zhang, H., 2004. The optimality of Naive Bayes.

46- Zhang, T., n.d. Solving large scale linear prediction problems using stochastic gradient descent algorithms. *Proceedings of ICML '04.*

APPENDIX A

CLIENT SOFTWARE SOURCE CODE

`main.py` Module

```
import sys
import datetime
import PyQt5
from PyQt5.QtWidgets import QApplication, QDialog, QMessageBox
from PyQt5.QtGui import QFont, QColor
from PyQt5.QtGui import QIcon, QPixmap
from network_interface import *
from skin import *


class Client(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.setFixedSize(self.size())
        self.date = None

        pixmap = QPixmap('avatar.png')
        self.ui.labelAvatar.setPixmap(pixmap)
        self.plain_font = QFont('Times New Roman', pointSize=11)
        self.bold_font = QFont('Times New Roman', pointSize=11)
        self.bold_font.setBold(True)
        self.ui.textEditChat.setEnabled(False)
        self.ui.pushButtonSend.clicked.connect(self.send)
        self.setFixedSize(self.size())
        connect()

    def send(self):
        user_question = self.ui.lineEditAsk.text()

        condition = user_question == '' or user_question.isspace() or len(user_question) < 3
        if condition : return
        self.ui.lineEditAsk.clear()
        dt = datetime.datetime.now()
        date = dt.date()
        date_str = date.strftime('[ %d-%m-%y ]')
        time_str = dt.time().strftime('%H:%M')
```

```
            self.ui.textEditChat.setCurrentFont(self.plain_font)
            if self.date != date:
                self.date = date
                date_str = self.date.strftime('[ %d-%m-%y ]')
                self.ui.textEditChat.append(date_str)
                self.ui.textEditChat.setAlignment(PyQt5.QtCore.Qt.AlignCenter)
            send(user_question)
            responce = recv()
            self.ui.textEditChat.append('')
            self.ui.textEditChat.append(time_str)
            self.ui.textEditChat.setAlignment(PyQt5.QtCore.Qt.AlignRight)
            self.ui.textEditChat.setCurrentFont(self.bold_font)
            self.ui.textEditChat.append('أنت: '+user_question)
            self.ui.textEditChat.setAlignment(PyQt5.QtCore.Qt.AlignRight)
            self.ui.textEditChat.setCurrentFont(self.plain_font)
            self.ui.textEditChat.setTextColor(QColor(255, 0, 0))
            self.ui.textEditChat.append(time_str)
            self.ui.textEditChat.setAlignment(PyQt5.QtCore.Qt.AlignLeft)
            self.ui.textEditChat.setCurrentFont(self.bold_font)
            self.ui.textEditChat.append('لينه: '+responce)
            self.ui.textEditChat.setAlignment(PyQt5.QtCore.Qt.AlignLeft)
            self.ui.textEditChat.setTextColor(QColor(0, 0, 0))
            self.ui.textEditChat.setEnabled(False)
            return


if __name__=="__main__":
    app = QApplication(sys.argv)
    w = Client()
    w.show()
    sys.exit(app.exec_())
```

network_interface.py Module

```
import socket
IP = socket.gethostbyname(socket.gethostname())
PORT = 55555
ADDR = (IP, PORT)
HEADER = 64
CODE = 'utf-8'

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
def connect():
    try:client.connect(ADDR)
    except: return None
    return True
def send(msg):
    encoded_msg = msg.encode(CODE)
    len_msg = len(encoded_msg)
    send_msg = str(len_msg).encode(CODE)
    send_msg += b' '*(HEADER - len(send_msg))
    client.send(send_msg)
    client.send(encoded_msg)
def recv():
    header = client.recv(HEADER).decode(CODE)
    len_msg = int(header)
    msg = client.recv(len_msg).decode(CODE)
    return msg
```

`skin.py` Module:

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

# Form implementation generated from reading ui file 'skin.ui'
#
# Created by: PyQt5 UI code generator 5.14.2
#
# WARNING! All changes made in this file will be lost!


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(1069, 524)
        Dialog.setSizeGripEnabled(False)
        self.labelAvatar = QtWidgets.QLabel(Dialog)
        self.labelAvatar.setGeometry(QtCore.QRect(20, 20, 300, 486))
        self.labelAvatar.setText("")
        self.labelAvatar.setObjectName("labelAvatar")
        self.widget = QtWidgets.QWidget(Dialog)
        self.widget.setGeometry(QtCore.QRect(330, 20, 731, 491))
        self.widget.setObjectName("widget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.widget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
```

```
        self.verticalLayout.setObjectName("verticalLayout")
        self.textEditChat = QtWidgets.QTextEdit(self.widget)
        self.textEditChat.setReadOnly(True)
        self.textEditChat.setObjectName("textEditChat")
        self.verticalLayout.addWidget(self.textEditChat)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.pushButtonSend = QtWidgets.QPushButton(self.widget)
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(10)
        font.setBold(True)
        font.setWeight(75)
        self.pushButtonSend.setFont(font)
        self.pushButtonSend.setObjectName("pushButtonSend")
        self.horizontalLayout.addWidget(self.pushButtonSend)
        self.lineEditAsk = QtWidgets.QLineEdit(self.widget)
        self.lineEditAsk.setObjectName("lineEditAsk")
        self.horizontalLayout.addWidget(self.lineEditAsk)
        self.verticalLayout.addLayout(self.horizontalLayout)

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Lina Chatbot"))
        self.textEditChat.setHtml(_translate("Dialog", "<!DOCTYPE HTML PUBLIC \"-
//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:\'MS Shell Dlg 2\'; font-size:8.25pt; font-
weight:400; font-style:normal;\">\n"
"<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;\"><br /></p></body></html>"))
        self.pushButtonSend.setText(_translate("Dialog", "ارسال"))

if __name__ == "__main__":
    pass
```

APPENDIX B

SERVER SOFTWARE SOURCE CODE

`main.py` Module:

```
from dialog_manager import Conversation
from intent_classifier import load_models, make_prediction, get_candidate_classes
from preprocessing import TextCleaner
from network_interface import create_server, start_server
from database_interface import add_entry

models = load_models('all')
cleaner = TextCleaner()

def chatbot(conv, text):
    cand = get_candidate_classes(text, models, 'all')
    pred, prob = make_prediction(text, models, 'all')
    text = cleaner.transform([text])[0][0]

    conv.set_candidates(indices=cand)
    conv.set_user_utrance(text)
    responce, state = conv.generate_responce()
    if state == 'first responce': add_entry(text, pred)
    return responce, conv


server_socket = create_server()
start_server(server_socket, Conversation, chatbot)
```

`network_interface.py` Module

```
import socket
import threading
import select

HEADER = 64
CODE = 'utf-8'

def create_server():
    PORT = 55555
    IP = socket.gethostbyname(socket.gethostname())
    ADDR = (IP, PORT)
```

```
    server_socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_STREAM)
    server_socket.bind(ADDR)
    return server_socket

def handle_client(conn, addr, conv, func):
    global HEADER
    global CODE
    import time
    now = time.time()
    TIME_OUT = 120
    conv = conv()
    while time.time() <= now + TIME_OUT:
        msg_len = conn.recv(HEADER).decode(CODE)
        if msg_len:
            msg = conn.recv(int(msg_len)).decode(CODE)
            responce, conv = func(conv, msg)
            responce = responce.encode(CODE)
            resp_len = len(responce)
            send_len = str(resp_len).encode(CODE)
            send_len += b' ' * (HEADER - len(send_len))
            conn.send(send_len)
            conn.send(responce)
            now = time.time()
    conn.close()
    active_connections = threading.active_count() - 2
    print(f'[CONNECTIN {addr} TERMINATED, ACTIVE CONNECTIONS: {active_connections}]')

def start_server(server, conv, func):
    server.listen()
    print('[SERVER STARTED SUCCESSFULY]')
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr, conv, func))
        thread.start()
        active_connections = threading.active_count() - 1
        print(f'[NEW CONNECTION: {addr} CONNECTED , ACTIVE CONNECTIONS: {active_connections}]')


if __name__== '__main__': pass
```

`intent_classifier.py` Module:

```
import numpy as np
from sklearn.externals import joblib
from helper import resource_path, generate_paths
```

```
from feature_extraction import make_feature_extraction_pipeline
from preprocessing import make_preprocessing_pipline
from sklearn.base import BaseEstimator

class Voter(BaseEstimator):
        def __init__(self, voting='hard', models=None):
                self.voting = voting
                self.models = models
        def fit(self, X, y):
                return self
        def predict(self, X):
                predictions = []
                probas = []

                if self.voting == 'soft': return self.__soft_voting(X)
                elif self.voting == 'hard':
                        print('hard_voting')
                        for _, model in self.models.items():
                                if _ != 'hard_voter':
                                    predictions.append(model.predict(X))
                        return self.__mode_finder(predictions, X)

        def predict_proba(self, X):
                probas = []
                for _, model in self.models.items():
                    if _ != 'soft_voter' and _ != 'hard_voter':
                        probas.append(model.predict_proba(X))
                return np.average(probas, axis=0)

        def __soft_voting(self, X, index=None):
                probas = []
                for _, model in self.models.items():
                    if _ != 'soft_voter' and _ != 'hard_voter':
                                probas.append(model.predict_proba(X))
                if not index: return [np.argmax(np.average(probas, axis=0))]
                else: return [np.argmax(np.average(probas[index], axis=0))]

        def __mode_finder(self, preds, X):
                from collections import Counter
                preds = [p[0] for p in preds]
                counts = Counter(preds).items()
                counts = list(counts)
                counts.sort(key=lambda x: x[1], reverse=True)
                self.counts = counts if len(counts) <= 4 else counts[:4]
                max_count = counts[0][1]
```

```
                index=[counts[0][0]]
                for value, count in counts[1:]:
                        if count == max_count:index.append(value)
                if len(index) > 1:
                    return self.__soft_voting(X, index=index)

                else: return [index[0]]

        def get_most_voted(self, X, n):
            predictions = []
            probas = []

            if self.voting == 'soft': return self.__soft_voting(X)
            elif self.voting == 'hard':
                    for _, model in self.models.items():
                            if _ != 'hard_voter':
                                predictions.append(model.predict(X))
            from collections import Counter
            predictions = [p[0] for p in predictions]
            counts = Counter(predictions).items()
            counts = list(counts)
            counts.sort(key=lambda x: x[1], reverse=True)
            counts = counts if len(counts) <= n else counts[:n]
            return counts


def load_models(vec, voting=['soft', 'hard']):
    relative_paths = generate_paths(obj='models', vec=vec)
    paths = [resource_path(path) for path in relative_paths]
    keys = [path.split('\\')[-1] for path in paths]
    models = {}
    for k, p in zip(keys, paths):
        models[k[:-4]] = joblib.load(p)
    for v in voting:
        models[f'{v}_voter'] = Voter(voting=v, models=models)
    return models

def prepair_input(text, vec):
    preprocessing_pipeline = make_preprocessing_pipline()
    feature_extraction_pipeline = make_feature_extraction_pipeline(vec=vec)
    clean_text = preprocessing_pipeline.fit_transform(text)
    features = feature_extraction_pipeline.fit_transform(clean_text)
    return features
```

```
def get_candidate_classes(text, models, vec):
    X = prepair_input([text], vec)
    most_voted = models['hard_voter'].get_most_voted(X, 4)
    most_voted = [m[0] for m in most_voted]
    return most_voted



def make_prediction(text, models, vec):
    X = prepair_input([text], vec)
    predictions = {}
    propabilities = {}
    for name, model in models.items():
        prediction = model.predict(X)
        predictions[name] = prediction[0]
        try: class_props = model.predict_proba(X)
        except: class_props = None
        propabilities[name] = class_props
    return predictions, propabilities

if __name__ == '__main__': pass
```

`dialog_manager.py` Module:

```
import numpy as np
from preprocessing import make_preprocessing_pipline
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.externals import joblib
from helper import resource_path
from database_interface import load_responce
next_utrance_types = {'r': 'first responce', 'rr': 'second responce', 'q': 'queery'}
yes_responces = ['نعم', 'اجل', 'بلى', 'فعلا', 'هى', 'هو']
no_responces = ['لم', 'لن', 'لا', 'ما', 'ليس', 'ليست']
candidiates_turns = {4: (2, 2), 3: (1, 2), 2: (2, 0), 1:(1, 0)}
classes = {0: 'العدوى من الحيوانات',
           1: 'العدوى من أشخاص بدون أعراض',
           2: 'إنهاء المحادثة',
           3: 'عمر الفيروس',
           4: 'شحن البضائع من و الى مناطق الوباء',
           5: 'استخدام الكمامة',
           6: 'فترة الحضانة',
           7: 'استعمال المعقمات',
           8: 'الإصابة أكثر من مرة بالمرض',
           9: 'الوقاية',
           10: 'بدء المحادثة',
```

```
            11: 'أعراض المرض',
            12: 'تأثير العوامل المناخية على الفيروس',
            13: 'فحص وجود الفيروس',
            14: 'طرق انتقال الوباء'}
utrance_vectors = joblib.load(resource_path('utrances_vectors\\utrances_vectors.pkl'))
utrance_vectorizer = joblib.load(resource_path('utrance_vectorizer\\utrance_vectorizer.pkl'))
SIM_THRISH = 0.8
special_classes_indices = [2, 10]
ERROR_MSG = 'إن كان ما تسأل عنه متعلقا بفيروس كورونا الرجاء أعد كتابته بصيغة مختلفة .. !!لم أستطع فهمك'
OR = ' أم'
CLARIFICATION = ' هل تريد'
HELLO_MSG = 'روبوت دردشة آلية صنع بجامعة السودان للعلوم والتكنولوجيا للإجابة عن اسئلة كورونا الشائعة .. مرحباً اسمي لينه'
BYE_MSG = 'شكراً لتواصلك'
INFORMATION = ' معلومات عن'
class Conversation:
    def __init__(self):
        self.next_user_utrance_type = next_utrance_types['q']
        self.candidates = None
        self.user_utrance = None
        self.chatbot_utrance = None
        self.turn1 = None
        self.turn2 = None
        self.returnd_classes = None
        self.found = False

    def set_candidates(self, indices):
        indices.reverse()
        self.candidates = indices

    def set_user_utrance(self, utrance):
        self.user_utrance = utrance

    def __check_special(self):
        if self.candidates[-1] in special_classes_indices:
            return True, self.candidates[-1]
        else:
            return False, False

    def __get_similar(self, utrance, vectors, vectorizer):
        utrance_vector = vectorizer.transform(utrance)
        similarties = cosine_similarity(utrance_vector, vectors[self.returnd_classes])
        return self.returnd_classes[np.argmax(similarties)], similarties[np.argmax(similarties)]

    def __identify_class(self):
        if self.next_user_utrance_type == next_utrance_types['q']:
```

```
        print('q')
        is_special, clas = self.__check_special()
        if is_special: return [clas]
        self.turn1, self.turn2 = candidiates_turns[len(self.candidates)]
        self.returnd_classes = []
        for i in range(self.turn1):
            self.returnd_classes.append(self.candidates.pop())
        self.next_user_utrance_type = next_utrance_types['r']
        return self.returnd_classes

    elif self.next_user_utrance_type == next_utrance_types['r']:
        print('r')
        if self.turn1 == 1:
            for r in yes_responces:
                if r in self.user_utrance.split():
                    self.found = True
                    self.next_user_utrance_type = next_utrance_types['q']
                    return self.returnd_classes

            for r in no_responces:
                if r in self.user_utrance.split():
                    if self.turn2 > 0:
                        self.returnd_classes = []
                        for i in range(self.turn2):

                            self.returnd_classes.append(self.candidates.pop())
                            self.next_user_utrance_type = next_utrance_types['rr']
                            return self.returnd_classes

                    else:
                        self.turn1, self.turn2 = candidiates_turns[len(self.candidates)]
                        self.returnd_classes = []
                        for i in range(self.turn1):
                            self.returnd_classes.append(self.candidates.pop())
                        self.next_user_utrance_type = next_utrance_types['r']
                        return self.returnd_classes

        else:
            for r in no_responces:
                if r in self.user_utrance.split():
                    if self.turn2 > 0:
                        self.returnd_classes = []

                        for i in range(self.turn2):
                            self.returnd_classes.append(self.candidates.pop())
```

```
                            self.next_user_utrance_type = user_utrance_types['rr']
                            return self.returnd_classes

                    else:
                        self.next_user_utrance_type = user_utrance_types['q']
                        return [-1]

                utrance, sim = self.__get_similar(self.user_utrance, utrance_vectors, utrance_vect
orizer)
                if sim >= SIM_THRISH:
                    self.found = True
                    return [utrance]
                else:
                    self.turn1, self.turn2 = candidiates_turns[len(self.candidates)]
                    self.returnd_classes = []
                    for i in range(self.turn1):
                        self.returnd_classes.append(self.candidates.pop())
                    self.next_user_utrance_type = user_utrance_types['r']
                    return self.returnd_classes

        elif self.next_user_utrance_type == user_utrance_types['rr']:
            print('rr')

            for r in no_responces:
                if r in self.user_utrance.split():
                    self.next_user_utrance_type = user_utrance_types['q']
                    return [-1]

            utrance, sim = self.__get_similar(self.user_utrance, utrance_vectors, utrance_vectoriz
er)
            if sim >= SIM_THRISH:
                self.found = True
                return [utrance]
            else:
                self.turn1, self.turn2 = candidiates_turns[len(self.candidates)]
                self.returnd_classes = []
                for i in range(self.turn1):
                    self.returnd_classes.append(self.candidates.pop())
                self.next_user_utrance_type = user_utrance_types['r']
                return self.returnd_classes

    def generate_responce(self):
        clas = self.__identify_class()
        if len(clas) == 1:
            if clas[0] == -1:
```

```
                return ERROR_MSG, self.next_user_utrance_type

            elif clas[0] == 2:
                return BYE_MSG, self.next_user_utrance_type

            elif clas[0] == 10:
                return HELLO_MSG, self.next_user_utrance_type

            elif clas[0] in list(range(15)) and self.found == True:
                self.found = False
                return load_responce(clas[0]), self.next_user_utrance_type


            else:
                return CLARIFICATION + INFORMATION + classes[clas[0]], self.next_user_utrance_type
        else:
            return CLARIFICATION + INFORMATION + classes[clas[0]] + OR + INFORMATION + classes[cla
s[1]], self.next_user_utrance_type



if __name__ == '__main__': pass
```

database_interface Module

```
import sqlite3
from helper import resource_path


def create_connection(name):
    db = sqlite3.connect(resource_path('databases\data.sqlite'))

def load_responce(index):
    db = sqlite3.connect(resource_path('databases\\responces.sqlite'))
    c = db.cursor()
    c.execute(F"SELECT * FROM answers_data;")

    x = filter(lambda x: x[1] == str(index), c.fetchall())
    for i in x:
        responce = i[0]
    return responce
```

`preprocessing.py` Module

```
import numpy as np
import pandas as pd
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.externals import joblib
from sklearn.pipeline import Pipeline
from helper import resource_path

def clean_text(text):
    import re
    search = ["أ","إ","آ","ة","_","-
","/",".","،"," و"," يا",'"',"-","'","ى","\\",'\n', '\t','&quot;','?','؟','!']
    replace = ["ا","ا","ا","ه"," "," ","","","","" و"," يا","","","","ي","",' ', ' ',' ',' ? ',' ؟','
! ']
    #remove tashkeel
    p_tashkeel = re.compile(r'[\u0617-\u061A\u064B-\u0652]')
    text = re.sub(p_tashkeel,"", text)

    #remove longation
    p_longation = re.compile(r'(.)\1+')
    subst = r"\1\1"
    text = re.sub(p_longation, subst, text)

    text = text.replace('وو', 'و')
    text = text.replace('يي', 'ي')
    text = text.replace('اا', 'ا')
    text = text.replace('؟', '')
    for i in range(0, len(search)):
        text = text.replace(search[i], replace[i])

    #trim
    text = text.strip()

    return text

def stop_words_removal(sentance, word_list):
    sentance = clean_text(sentance)
    for word in sentance.split():
        if word in word_list:
            sentance = sentance.replace(word, '')
    sentance = clean_text(sentance)
    return sentance


class TextCleaner(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
```

```
    def fit(self, X):
        return self

    def transform(self, X):
        temp = np.array([])
        for text in X:
            t = clean_text(text)
            temp = np.append(temp, t)
        X = temp.reshape(-1, 1)
        return X


class StopWordsRemoval(BaseEstimator, TransformerMixin):
    def __init__(self, stop_words_path):
        self.words_list = joblib.load(stop_words_path)

    def fit(self, X):
        return self

    def transform(self, X):
        temp = np.array([])
        for text in X:
            t = stop_words_removal(text[0], word_list=self.words_list)
            temp = np.append(temp, t)
        X = pd.Series(temp)
        return X

def make_preprocessing_pipline():
    stop_words_path = resource_path('preprocessing\\stop_words.pkl')
    pipeline = Pipeline([('cleaner', TextCleaner()),
                         ('removal', StopWordsRemoval(stop_words_path))])
    return pipeline

if __name__ == "__main__": pass
```

`feature_extraction.py` Module

```
import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.externals import joblib
from sklearn.pipeline import Pipeline
from helper import resource_path
class Vectorizer(BaseEstimator, TransformerMixin):
    def __init__(self, vectorizer_path):
```

```
        self.vectorizer = joblib.load(vectorizer_path)

    def fit(self, X):
        return self

    def transform(self, X):
        X = self.vectorizer.transform(X)
        return X

class SimilarityExtractor(BaseEstimator, TransformerMixin):
    def __init__(self, vectors_path):
        from sklearn.metrics.pairwise import cosine_similarity
        self.vectors = joblib.load(vectors_path)
        self.cos_sim = cosine_similarity
    def fit(self, X):
        return self

    def transform(self, X):
        print(X)
        X = self.cos_sim(X, self.vectors)
        return X



def make_feature_extraction_pipeline(vec):

    vectorizer_path = resource_path(f'feature_extraction\\{vec}\\vec_{vec}.pkl')
    vectors_path = resource_path(f'feature_extraction\\{vec}\\question_vectors_{vec}.pkl')
    duplications_path = resource_path(f'feature_extraction\\{vec}\\duplicated_{vec}.pkl')
    pipeline = Pipeline([('vectorizer', Vectorizer(vectorizer_path)),
                         ('similarity', SimilarityExtractor(vectors_path))])
    return pipeline

if __name__ == '__main__': pass
```

`helper.py` Module

```
import os
def generate_paths(obj, vec='all'):
    folder = f'{obj}\\{vec}'
    names = os.listdir(folder)
    paths = [os.path.join(folder, name) for name in names]
    return paths

def resource_path(relative_path):
```

```
    try:
        base_path = sys._MEIPASS
    except Exception:
        base_path = os.path.abspath(".")

    return os.path.join(base_path, relative_path)


if __name__ =='__main__': pass
```