Opinion mining from Data of Social Media by Probabilistic Logic Reasoning

by

Stefanos Zervoudakis

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

May 2020

Approved by:

Supervisor

Prof. Emmanouil Marakakis

Abstract

This thesis studies opinion mining from social media with probabilistic logic reasoning. Twitter is one of the most active social networks, with millions of tweets sent daily, where multiple users express their opinion about travelling, economic issues, political decisions etc. As such, it offers a valuable source of information for opinion mining. Our approach uses a Bayesian-based opinion mining framework exploiting Twitter Data. It is described by the following steps. First, the framework of our approach imports Tweets massively by using Twitter's API. Next, the imported Tweets are further processed automatically for constructing a set of untrained rules and random variables. Then, a Bayesian Network is derived by using the sets of untrained rules, the random variables and an evidence set. After that, the trained model can be used for the evaluation of new Tweets. Finally, the constructed model can be retrained incrementally thus becoming more robust.

As application domain for the development of our methodology we have selected tourism because it is one of the most popular topics in social media. Our system can predict with some probability users' preferences, regarding their intention to visit a place or not. We have developed algorithms which create automatically efficient rules and random variables based on the trainset. Our system uses for model training the probabilistic logic reasoning system of ProbLog. The advantages of our approach are the following. First, our system follows an incremental learning strategy. That is, the derived model can be retrained incrementally with new training sets thus becoming more robust. Second, our system can be easily adapted to opinion mining from social media on other topics. Finally, the rules of the derived model are constructed in an efficient way and automatically.

Dedication

To ….

Acknowledgements

Table of Contents

List of Programs

List of Figures

List of Tables

List of Algorithms

1 Overview and Motivation

1.1 Introduction

The exponential growth of social networks has largely changed the way people interact with each other, influencing the way they think, eventually changing their opinion on several topics. As such, analyzing user-generated content offers a great opportunity for automatically detecting opinions and trends on topics of interest.

Opinion mining is the science which performs text analysis in order to understand the drivers behind public sentiment. *Sentiment analysis* is predecessor of opinion mining, because opinion mining goes a level deeper. For example, sentiment analysis examines how people feel about a given topic (positive or negative), unlike opinion mining that examines why people feel the way they do.

In this master thesis we present a system for opinion mining based on information available on Twitter. Our system performs probabilistic logical reasoning using Bayesian networks. Our approach can be applied in many other topics such health care, tourism, political opinions, economic issues etc.

Bayesian networks, are a type of probabilistic graphical models that use Bayesian inference for probability computations. Bayesian networks aim to model conditional dependence and therefore causation, by representing conditional dependence by edges in a directed graph. The probability to reach from one state to the next state (edge), use the joint probability distribution. Mathematically, the joint probability distribution is the probability of two or more events happening at the same time (happening together). The joint probability for two events *event A* and *event B* can be written as $P(A\ and\ B)$ or as $P(A \cap B)$

We have used Twitter data, collected through API's, in order to construct and train a Bayesian model which can be applied to new Tweets identifying user preferences. For instance, each user of social media can post photos for places that he/she has been visited. So, with an appropriate framework (that use Probabilistic logic through machine learning), questions such as "The user X likes the city of Chania" can be answered by examining the photos that the user X has posted in the past. As an application scenario, we focus on travelling. Our system can predict user preferences, regarding visiting a place or not with some probability.

1.2 Problem Definition

As we have mentioned earlier, our system can predict user preferences, regarding the intention to visit a place or not with some probability. More specifically, this thesis focuses on mining the intention of Tweeter users for visiting the Greek island Crete. Initially, we collected data from Twitter that have been posted by users who have visited Crete or by users which are planning to visit Crete in the near future. So, data from twitter have been collected based on specific hashtags.

The web-page http://best-hashtags.com has been used in order to identify popular hashtags for our problem [Best- Hashtags, n.d.]. According to this web-page the most popular hashtags used by users who visited Crete are **#visitcrete**, **#travel**, **#crete**, **#creteisland**. All tweets with the aforementioned hashtags have been stored in the form of a Comma-Separated Values (CSV) file for further proceccing. Then, we developed algorithms in order to generate the evidence set and the probabilistic rules based on the tweets in the CSV file. Before we present the scientific methods(Sentiment Analysis, Entity Recognition) that use the algorithms that we have constructed it is important to define the evidence term. *Evidence set* is a set of information decribing events that are true or false. So, the scientific methods that we use in our algorithms are the following:

- *Sentiment Analysis* is the process by which we characterize the emotional tone that contains a set of words [Agarwal, et al., 2011] . That is, how positive or negative the sentiment is. That process refers to the use of Natural Language Processing (NLP), text analysis, computational linguistics and biometrics.
- *Entity Recognition* is a task of information extraction that seeks to locate and classify named entity mention in unstructured text into some pre-defined categories. Some categories that are used generaly are location, travel, medical codes, company names etc. [Ritter, et al., 2011].

The next step, is to use the ProbLog for probabilistic logic reasoning. Probabilistic Logic Programs are logic programs in which some of the facts are annotated with probabilities. ProbLog is a library of Python, and with this toolbox we can train the model based on evidence set and probabilistic rules. In addition, with ProbLog we evaluate the trained model based on new Tweets as mentioned earlier. In the train process, ProbLog uses the machine learning algorithm Expectation Maximization E.M. The general idea of E.M algorithm is to estimate the parameters of a statistical model by an iterative process. We use ProbLog in order to train our model and for the evaluation of our model as well as. ProbLog uses the machine learning algorithm Expectation Maximization (E.M.) for training a Bayesian model. The general idea of E.M algorithm is to estimate the parameters of statistical models with iterative process. A parameter in a probabilistic model is the quantity entering into the probability in random variables (i.e. $P(X=x) = 0.50$). The final step is to evaluate the trained model. Our system takes as input tweets in textual form and then based on the evidences, returns the opinion about visit Crete with some probability.

1.3 Thesis Structure

The rest of this thesis is organized in the following chapters. In Chapter 2, we present the background and related work in machine learning and probabilistic logic reasoning. In addition, the Bayes' law in presented. Also, the Bayesian method in machine learning, both inference and learning, is presented. In Chapter 3, an overview of our machine learning system is presented. In Chapter 4, the architecture of our machine learning system is discussed. In Chapter 5, the basic components of our system are presented. In Chapter 6, some sample sessions of our system are illustrated. In Chapter 7, the evaluation of our system using RME metrics is presented. In Chapter 8, the conclusions of this research and future work are discussed.

2 Background and Related Work

In this chapter we present all scientific methods that have been used for the development of our system. Initially, a short review of Machine Learning domain will be presented and how it is applied in Probabilistic Logic Reasoning. Next, the Bayesian method in the Machine Learning domain is presented. Finally, all related work of our approach is discussed.

2.1 Introduction to Machine Learning

In computer science, Machine Learning is a very popular and important scientific method[Shindle, et al, 2018]. *Machine Learning* is at the core of artificial intelligence research. Machine Learning is the learning in which machine can learn by its own without being explicitly programmed. This science, provides to an intelligent system the ability to automatically learn and improve from experience. The most popular learning algorithms belong in the following categories: *supervised learning*, *unsupervised learning* and *reinforcement learning*. On one side, *supervised learning* is a learning in which we teach or train a model using data which are well labeled, it means that some or all data are already tagged with the correct answer. On the other side, *unsupervised learning* is the training of a model using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. In the *reinforcement learning*, the system is given a sequence of examples or states and a reward after completing that sequence, it learns to predict the action to take in for an individual example or state. This can also be done through direct interaction with the intended environment.

The algorithms which perform *supervised learning* are classified into the following two categories:

- *Classification*: An algorithm performs classification when the output variable is a category, a class such as "disease" and "no disease"
- *Regression*: An algorithm performs regression when the output variable is a real value such as "weight" or "probability".

The algorithms that use *unsupervised learning* techniques can be classified into the following two categories:

- *Clustering*: An algorithm performs clustering by discovering the inherent groupings in the data, such as grouping customers by purchasing behavior, etc.
- *Rule-based*: The rule-based method is the machine learning method for discovering important relations between variables in large datasets. All significant association rules between items in the database can be discovered by algorithms which perform machine learning by the rule-based method [Agarwal, et al, 1993].

2.2 Machine Learning and Probabilistic Logic Reasoning

The term "probabilistic logic" was first used by Nils Nilsson in 1986 [Nilsson, 1986], where the truth values of clauses are probabilities. The proposed semantical generalization induces a probabilistic logical entailment, which reduces to ordinary logical entailment when the probabilities of all sentences are either 0 or 1. This generalization applies to any logical system for which the consistency of a finite set of sentences can be established. Other scientific methods they use probabilistic logic are Markov Logic Network, Bayesian Logic, Probabilistic Argumentation, etc. The most popular application areas that use Probabilistic Logic Reasoning are the following:

- Statistics
- Bioinformatics
- Game theory
- Psychology
- Real-life

As we mentioned earlier, there are many scientific methods that use Probabilistic Statistical relational learning (SRL) studies the integration of probabilistic reasoning with machine learning and first order and relational representations. Statistical Relational Learning is related to the research work in this thesis. This is discussed in Section 2.10.

2.3 The Bayes' law in our System

As it is known the Bayesian Networks use the Bayes theorem (alternatively Bayes' law or Bayes rule) in order to perform probabilistic inference. We present the Bayes' law based on the trained model of our system.

$$P_M(\theta|D) = \frac{P_M(D|\theta) * P_M(\theta)}{P_M(D)}$$

The items from the model, shown in Figure 8, correspond to the elements of the above formula as follows:

- PM(θ|D) stands for the probability of hypothesis θ when D is given (evidence). In our model θ corresponds to the item "visitLocation" (Figure 10 Trained Bayesian Network) and D corresponds to the evidence data, i.e. "locationInTweet", "negativeSentiment", "positiveSentiment" , "userLocation", "hotel","restaurant","tourism" and "vacation".
- PM(D|θ) stands for the probability of evidence D given that the model parameters θ, i.e." visitLocation", are known
- PM(D) is the total probability of generating these data under each and all possible θ (in our case we have just one rule about visiting Crete).

2.4 Bayesian Networks Inference

In this sub-section we will describe the inference from Bayesian Networks. As it is known Bayesian Networks are a type of probabilistic graphical model that uses Bayesian inference for probability computations. A Bayesian is a Directed Acyclic Graph (DAG) G representing a dependency structure over a set of random variables X = $\{X_1, X_2, X_3, \dots, X_n\}$, where each node N_i (1≤i≤n) represents a random variable and it has a direct influence on other random variables. Because each node corresponds to a random variable, i.e. $N_i = X_i$ (1≤i≤n) we will denote each node by the random variable X_i (1≤i≤n) that it represents. In addition, Bayesian Networks have conditional probability distribution (CPD) associated with each aforementioned nodes-random variables. In the Bayesian Networks, the *conditional probability distribution* of X is denoted by $cpd(X)$ and the parents of a node X are denoted by $Pa(X)$. So, for node X its cpd(X) is defined in terms of its parents of $Pa(X)$ as follows [Farasat et, al. 2015] [De Raedt, 2008].

$$cpd(X) = P(X|Pa(X))$$

Let have the following network which is almost a standard example for discussing Bayesian Networks[Farasat et, al. 2015] [De Raedt, 2008] that is illustrated in **Figure** 1**.**

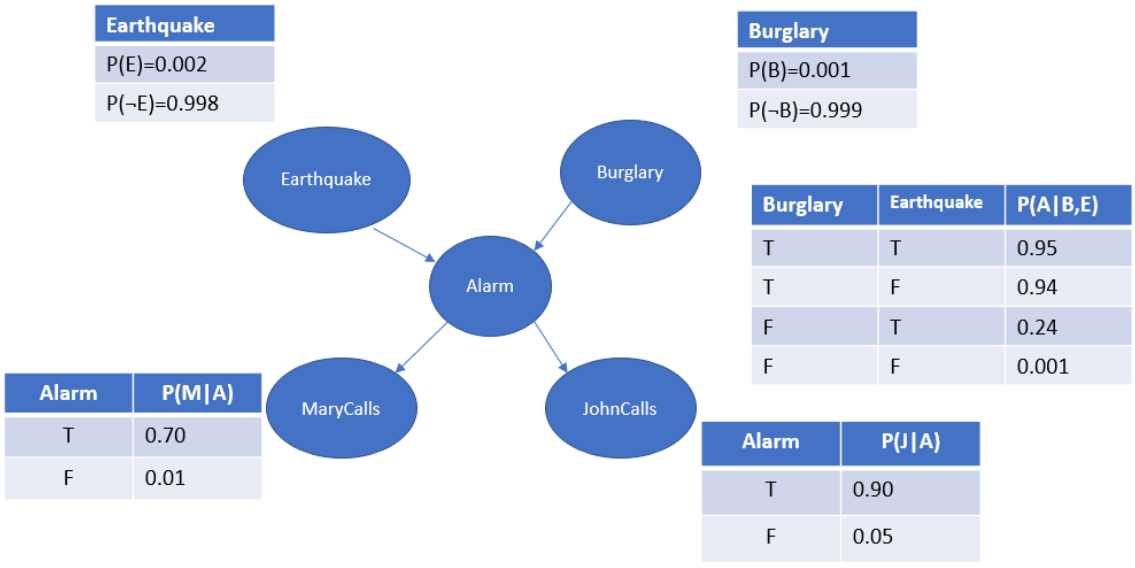

Figure 1 Bayesian Network and joint probability distributions

As it is known, *conditional dependence* is a relationship among random variables, i.e. nodes in Bayesian nets. More specifically, *conditional dependence* is a relationship, between two random variables or events that are dependent when a third occurs. The above network, Figure 1, represents 5 random variables, {Burglary, Earthquake, Alarm, JohnCalls, MaryCalls}. These random variables have a set of values such {true, false}. The edge *(Burglary, Alarm)* shows the direct influence among the random variables *Burglary -> Alarm*. So, between these two variables we have *conditional dependence*. That is, the random variable Alarm conditionally depends on the random variable Burglary. The network, also represents the *conditional independence* among the variables. For example, the variables MaryCalls and Burglary are *conditional independent* when Alarm is given. According to [De Raedt, 2008] based on local Markov assumptions, we can write the *joint probability density* as:

$$P(X_1, \dots, X_n) = \prod_{i=1}^{n} P(X_i | X_1, \dots, X_{i-1}) = \prod_{i=1}^{n} P(X_i | Parents(X_i)).$$

Let's denote random variables by their initials, i.e. JohnCalls by J, MaryCalls by M, Alarm by A, Burglary by B and Earthquake by E. Let's assume that alarm rang, Mary and John calls, and we know that there was no earthquake or burglary in the house. The aforementioned events correspond to *joint probability distribution* and is defined by the set {J, M, A, ¬B, ¬E}. According to the above formula, the *joint probability distribution* of the set {J, M, A,¬B,¬E } is calculated as follows:

$$P(\mathrm{J}, \mathrm{M}, \mathrm{A}, \neg\mathrm{B}, \neg\mathrm{E}) = \prod_{i=1}^{n} P(X_i | X_1, \dots, X_{i-1}) = \prod_{i=1}^{n} P(X_i | Parents(X_i)).$$

$$= P(J|A) * P(M|A) * P(A|\neg B, \neg E) * P(\neg B) * P(\neg E) = 0.90 * 0.70 * 0.001 * 0.999 * 0.998$$
$$= \mathbf{0.00062}$$

2.5 Likelihood function

The term parameter means the quantity entering into the probability of a random variable , e.g. P(X=x) = 0.43. In many probability distributions we do not know its parameters, and in this case, we use Likelihood function to estimate these parameters using sample data. In general, the Likelihood function gives us an idea of how well the data summarizes parameters. The *Probability Mass Function* (PMF) of a discrete random variable X gives the probability of each numerical value x that the random variable can take, i.e. $p_X(x)=P(\{X=x\})$ is the probability of the event {X=x}.

Now, we give the definition of *discrete likelihood function*. Let's we have the discrete random variable X with probability mass function p depending on the parameter θ. So, we have the following equation:

$$L(\theta|x) = p_\theta(x) = P_\theta(X = x)$$

We know that, the probability of the value x of random variable X for the parameter value θ is written as follows:

$$P(X = x|\theta)$$

The likelihood $L(\theta)$ is equal to the probability that a particular outcome (is a possible result of an experiment) x is observed, when the true value of the parameter is θ.
$L(\theta)$ is called the $likelihood\ function$ and is expressed as follows:

$$L(\theta) = P(X = x|\theta)$$

To be more accurate, we give an example. Let's have the experiment of coin flip. We have the parameter p_H which corresponds to the "fairness" of the coin. For perfectly fair coin we have the parameter $p_H = 0.5$. Now, we start the experiment, we flip the coin two times and observing the following data: *two heads in two tosses*.

We assume that each successive coin flip is identically and independently distributed. A set of random variables are *identically and independently distributed (iid)* if each random variable has the same probability distribution as the others and they are mutually independent. Let X and Y be the random variables for the 1st and the 2nd coin toss respectively.

P(Y=h|X=h) = P(Y=h ∩ X=h) / P(X=h) (1)

P(Y=h|X=h) = P(Y=h) (2) because the random variables X and Y are iid

From (1) and (2) we have P (Y=h ∩ X=h) = P(X=h) × P(Y=h) = ½ × ½ = ¼

Let X is the random variable and Y be the random variables that corresponds to the 1st and the 2nd coin toss respectively. In addition, **h** stands for head. So, according to the above formula we have the following:

$$P(Y = h \cap \mathrm{X} = \mathrm{h}) = P(X = h) * P(Y = h) = 0.5 * 0.5 = \mathbf{0.25}$$

Hence, given the observed data HH (two heads in two tosses), the likelihood function is written as follows:

$$L(\theta|x) = L(p_H = 0.5|HH) = P_\theta(X = H) * P_\theta(X = H) = 0.5 * 0.5 = \mathbf{0.25}$$

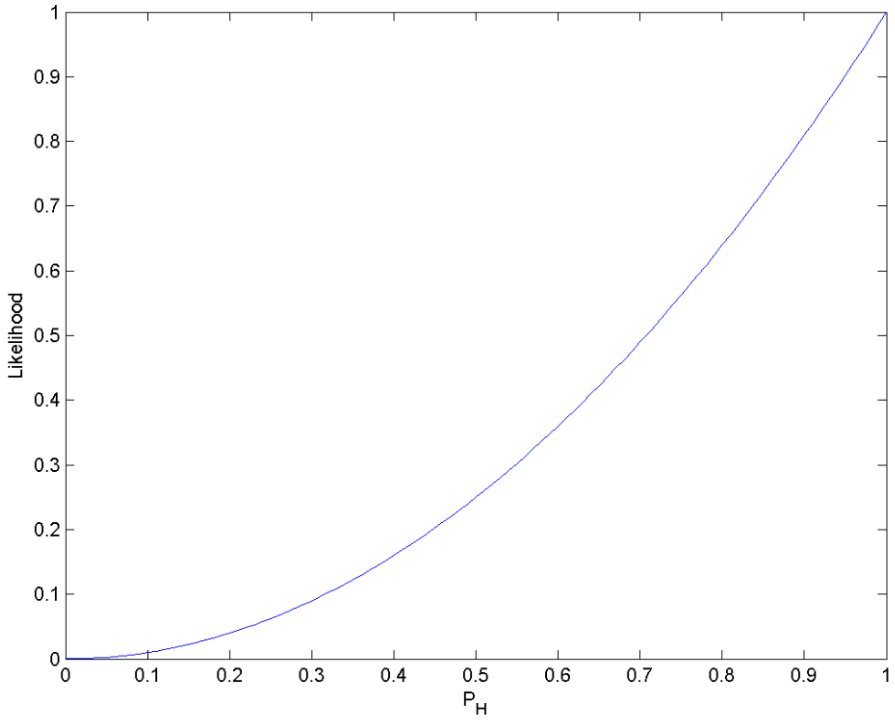

Figure 2 Likelihood function of example

As it is known, the probability is the number between 0 and 1 and it is the description of how likely an event is to occur. Also, in other words probability is the percentage that a success occurs. In summary, the likelihood function according to the aforementioned examples, is the conditional probability that an event occurs by knowing the probability of a success occurrence.

2.6 Structure Learning-Parameter Learning

In this sub-unit, we will describe two very important problems over Bayesian Networks, such structure learning and parameter learning. In a Bayesian network the DAG is called the *structure* and the values in the conditional probability distributions are called the *parameters*. The learning problem in a Bayesian Network includes the *structure learning* and the *parameter learning* tasks.

2.6.1 The parameter estimation problem.

In this subsection we will discuss the problem of parameter learning from data in Bayesian networks. In order to study this problem, we have to assume that we know the structure of the Bayesian network, i.e. the DAG. That is , we know the conditional dependencies of a set of random

variables. The problem of parameter learning involves estimates of the values of relative frequencies. The basic formalism of parameter learning is the following.

Definition

The problem of parameter learning is formalized from a set of elements that are given and a set of elements that they have to be found [De Raedt, 2008].

Given Elements

- A set of examples ***E***
- A probabilistic model ***M= (S, λ)*** with structure S and parameters λ.
- A probabilistic coverage relation ***P (e | M)*** that computes the probability of observing the example e given the model M.
- A scoring function ***score (E, M)***, that employs the probabilistic coverage relation P(e|M). This function is used to quantify the fitting of a Bayesian Network. This function it returns the maximum score that corresponds the best fit of the Bayesian Network.

Asked Elements

- After we have the above formalism, the problem is to find the parameters λ^* that maximize the scoring function, i.e.

$$\lambda^* = argmax\ score(E, (S, \lambda))$$

It is mentioned in [De Raedt, 2008] that "the problem specification shows that parameter estimation is essentially an optimization problem that depends on the scoring function and type of model employed. The standard scoring function is the probability of the model or hypothesis given the data as we present earlier in the Given Elements. This yields the *maximum a posteriori hypothesis* H_{MAP} such:

$$H_{MAP} = argmax \frac{P(E|H) * P(H)}{P(E)}$$

The maximum a posteriori hypothesis is the estimate of an unknown quantity that corresponds at the value that takes a discrete random variable where it has the maximum parameter (probability).

Example 1

Let's have the following Bayesian Network with two nodes. That is a subnetwork of the one shown in Figure 1. This Bayesian Network is illustrated in Figure 3.

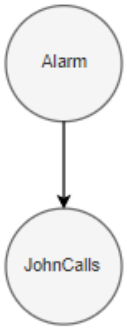

Figure 3 Probabilistic model M with structure S

Let have the full observable data such the ones shown in Table 1. That is, the set of examples E on the probabilistic model M is {e1, e2, e3, e4}. In the next discussion, the random variables of this Bayesian network are denoted by their initials which are shorter, *Alarm* is denoted by **a** and *JohnCalls* is denoted by **j**. As we mentioned earlier **λ** is used to denote the parameter of a model. We have the following parameters $P(\alpha) = \lambda_o$ and $P(\neg a) = (1 - \lambda_o)$ which are illustrated in Table 2 . The distributions of the Probabilistic Model in Figure 3 are specified over {true, false}.

In Table 3, represented the probability distribution over the model M.

Examples	a	j
e1	true	true
e2	true	false
e3	false	true
e4	false	false

Table 1 Completed dataset with examples E

P(a)
$(\boldsymbol{\lambda_o}, \mathbf{1} - \boldsymbol{\lambda_o})$

Table 2 Parameters for P(a)

a	P(j|a)
true	$(\boldsymbol{\lambda_1}, \mathbf{1} - \boldsymbol{\lambda_1})$
false	$(\boldsymbol{\lambda_2}, \mathbf{1} - \boldsymbol{\lambda_2})$

Table 3 joint distr. with parameters

In order to understand the usefulness of the likelihood function in parameter estimation problem we compute the likelihood of each one of the examples of Table 1.

The likelihood for example **e1,** satisfying specific values of random variables such **a**=true and **j**= true is the following :

$$P(a, j) = P(a) * P(j|a) = \boldsymbol{\lambda_0} * \boldsymbol{\lambda_1}$$

The likelihood for example **e2,** satisfying specific values of random variables such **a**=true and **j**= false is the following :

$$P(a, \neg j) = P(a) * P(\neg j|a) = \boldsymbol{\lambda_o}(1 - \boldsymbol{\lambda}_1).$$

The likelihood for example **e3,** satisfying specific values of random variables such **a**=false and **j**= true is the following :

$$P(\neg a, j) = P(\neg a) * P(j|\neg a) = (1 - \boldsymbol{\lambda_0}) * \boldsymbol{\lambda_2}$$

The likelihood for example **e4,** satisfying specific values of random variables such **a**=false and **j**= false is the following :

$$P(\neg a, \neg j) = P(\neg a) * P(\neg j|\neg a) = (1 - \boldsymbol{\lambda_0}) * (1 - \boldsymbol{\lambda_2})$$

As it is known, we have the model M with structure S represented in Figure 3, and we also have the set of examples E illustrated in table Table 1. In the next, we show in details the required steps in order to find the maximum likelihood estimation.

Step 1:

The first step, computes the probability of observing the set of examples E given the model M, it is denoted by the notation **P(E|M)**. So, under the i.i.d assumption, for n=4 examples we have the following equation that corresponds to the likelihood of the data of the table Table 1. Also, it is important to mention that we use |x| notation, that corresponds the number of examples of Table 1, satisfying the logical condition x.

Equation 2.1:

$$P(E|M) = \prod_{i}^{n=4} P(ei \mid M) = P(e1|M) * P(e2|M) * P(e3|M) * P(e4|M) =$$

$$\lambda_0^{|a|}(1 - \lambda_0)^{|\neg a|}\ \lambda_1^{|a \wedge j|}(1 - \lambda_1)^{|a \wedge \neg j|}\lambda_2^{|\neg a \wedge j|}(1 - \lambda_2)^{|\neg a \wedge \neg j|}$$

Step 2:

The above function can be maximized by maximizing the logarithm of the function instead, which is easier as well as justified because the logarithm is a monotonic function. The log-likelihood can be maximized by computing the derivatives, setting them to 0, and solving for the λ_i with the following steps:

Equation 2.2

$$\mathrm{L} = \log \mathrm{P(E \mid M)} = = \log (\lambda_0^{|a|} (1-\lambda_0)^{|\neg a|} \lambda_1^{|a \wedge j|} (1-\lambda_1)^{|a \wedge \neg j|} \lambda_2^{|\neg a \wedge j|} (1-\lambda_2)^{|\neg a \wedge \neg j|}) == \log (\lambda_0^{|a|}) + \log (1-\lambda_0)^{|\neg a|} + log\, \lambda_1^{|a \wedge j|} + (1-\lambda_1)^{|a \wedge \neg j|} + \lambda_2^{|\neg a \wedge j|} + (1-\lambda_2)^{|\neg a \wedge \neg j|}) =$$

$$= |\alpha| \log \lambda_0 + |\neg \alpha| \log(1-\lambda_0) + |\alpha \wedge \neg j| log \lambda_1 + |a \wedge \neg j| \log (1-\lambda_1) + |\neg a \wedge j| \log \lambda_2 + |\neg a \wedge \neg j| \log(1-\lambda_2)$$

The next step is to find the derivates of λ_i as we mentioned. This is calculated as follows.

$$\frac{\partial L}{\partial \lambda_0} = \frac{|a|}{\lambda_0} - \frac{|\neg a|}{1-\lambda_0}$$

$$\frac{\partial L}{\partial \lambda_1} = \frac{|a \wedge j|}{\lambda_1} - \frac{|\neg a \wedge j|}{1-\lambda_1}$$

$$\frac{\partial L}{\partial \lambda_2} = \frac{|a \wedge j|}{\lambda_1} - \frac{|\neg a \wedge j|}{1-\lambda_1}$$

Step 3:

Setting these equal to 0 and solving for the λ_i yields.

$$\lambda_0 = \frac{|a|}{|a| + |\neg a|}$$

$$\lambda_1 = \frac{|a \wedge j|}{|a \wedge j| + |a \wedge \neg j|}$$

$$\lambda_2 = \frac{|\neg a \wedge j|}{|\neg a \wedge j| + |\neg a \wedge \neg j|}$$

Summarizes, λ^* corresponds to the parameters that maximize the likelihood function (maximum likelihood estimation). These parameters are $\lambda_0, \lambda_1, \lambda_2$.

2.6.2 The structure learning problem

So far, we have analyzed the parameter estimation problem which has been studied in this thesis. In this subsection we will discuss the problem of structure learning from data in Bayesian networks even though we did not study this problem in this thesis. We perform this discussion for the reader of this thesis in order to have a complete view of the learning problem in Bayesian networks. Therefore, important techniques have been developed in order to learn the structure of a model from a given data set. So, we give a short brief in structure learning technique. This technique was not used in our system in the learning procedure, because our model has known structure of DAG based on algorithms that we have constructed. Furthermore, in the following bullets we will give some important definitions of structure learning technique [De Raedt, 2008].

Given Elements:

- We have a set of examples **E**
- We have also, a language $\boldsymbol{L_M}$ of possible models of the form **M = (S, λ)** with parameters **λ** and structure **S**
- A probabilistic coverage relation **P(e|M)** that computes the probability of observing the example **e** given the model **M**
- A scoring function **score (E, M)** that employs the probabilistic coverage relation **P(e|M).** We use, the same function as we presented in parameter estimation problem. So, this function is used to quantify the fitting of a Bayesian Network.

Asked Elements:

In the following example we will find the model M = (S, λ) that maximizes score (E, M) that is

$$M = argmax\ score(E, M)$$

This problem is a search problem, because, there is a space of possible models to be considered, defined by the language L_M , and the goal is to find the best one, according to the scoring function (model with the maximum score). In Bayesian Networks there are two possible cases:

1. The first case is when network is fully connected (where there is an edge between any pair of random variables)
2. The second case is when no links are contained at all. To evaluate a candidate structure S, the parameters λ are first estimated and then the scoring function is used to determine the overall score of the resulting model.

The problem with the scoring function, is that always prefers a fully connected network. To evaluate a candidate structure S, the parameters λ are first estimated and then the scoring function is used to quantify the overall score of the resulting model. So, in structure learning problem the goal is to find the model or hypothesis with the maximum posterior probability such as follows [De Raedt, 2008]:

$$H = argmax\ P(H|E) = argmax \frac{P(E|H)P(H)}{P(E)} = argmaxP(E|H)P(H) = argmax\ log\ P(E|H) + \log P(H)$$

2.7 Other Graphical Models

In this subsection, we will make a brief introduction to the other type of probabilistic graphical models, i.e. the Markov Networks.

Markov Networks: is a graphical model for the joint distribution of a set of variables X=$\{X_1, X_2, \dots, X_n\} \in$ X. Markov model can represent each random variable X_i with a directed graph. In the following bullets we mention the properties of Markov models.

- We have a set of states: $\{s_1, s_2, \dots, s_n\}$.
- Process moves from one state to another generating a sequence of states such the following $s_{i1}, s_{i2}, s_{i3}, \dots, s_{ik}, \dots$
- ***Chain property***: probability of each subsequent state depends only on what was the previous state, so, the following is the formula for computing the probability.
  $P(s_{ik} | s_{i1}, s_{i2}, \dots, s_{ik-1}) = P(s_{i1} | s_{ik-1})$
- To define Markov model, the following probabilities have to be specified, *transition probabilities* $a_{ij} = P(s_i \mid s_j)$ and *initial probabilities* $\pi_i = P(s_i)$.

So, to make it more understandable, we will give an example. Suppose that we have two states 'Rain', and 'Dry' which represent random variables. In addition, we have transition probabilities P('Rain'|'Rain') = 0.3, P('Dry'|'Rain') = 0.7, P('Rain'|'Dry) = 0.2, P('Dry|'Dry') = 0.8. And final, we have the initial probabilities P('Rain') = 0.4, P('Dry') = 0.6.

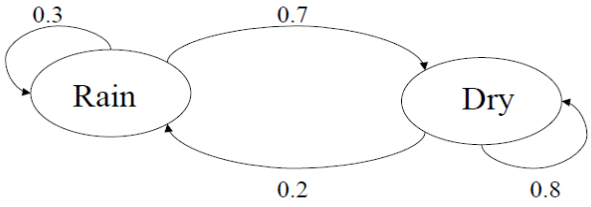

Figure 4 Markov Model

Let have the following sequence of states {'Dry', 'Dry', 'Rain', 'Rain'}

According to the aforementioned formula we have the following calculations:

$$P(s_{ik,} \mid s_{ik-1})P(s_{ik,} \mid s_{ik-1})....P(s_{i2,} \mid s_{i1})P(s_{i1})=$$

$$P(\text{'Rain'} \mid \text{'Rain'})P(\text{'Rain'} \mid \text{'Dry'})P(\text{'Dry'} \mid \text{'Dry'})P(\text{'Dry'})=$$

$$0.3*0.2*0.8*0.6=\mathbf{0.0288}.$$

The above Markov model with two states Rain-Dry and the transition probabilities in ProbLog is expressed as follows:

Program 1 Markov Model in ProbLog

%initial probabilities

0.4::init(rain).

0.6::init(dry).

%transition probabilities

0.3::stateTransition(rain,S,rain).

0.7::stateTransition(rain,S,dry).

0.2::stateTransition(dry,S,rain).

0.8::stateTransition(dry,S,dry).

2.8 Estimate of parameters in statistical models

In this sub-unit we will mention shortly some algorithms that can estimate the parameters of statistical models. The most popular algorithms are *Bayesian Estimation* and *Expectation Maximization (E.M*). The Bayesian Estimation algorithm is the alternative principle to maximum likelihood estimation which has been discussed in Section 2.5. The steps of the *Bayesian Estimation algorithm* are the following:

Steps of Bayesian Estimation Algorithm

- Start with a prior distribution and use experience (from dataset) to update the distribution.
- Collapse the posterior distribution to the mean value and use this as the final value of the parameter.
- If we have binary values (True False), let X be a binary value, and we have performed a number of independent experiments of which **n** turned up True and **m** turned up False. Then starting with even prior distribution for **θ** the Bayesian estimate for P(X=True) is

$$\frac{n+1}{n+m+2}$$

The basic steps of Bayesian *Estimation algorithm* have been presented. The detail presentation of *Expectation Maximization algorithm follows*. This algorithm has been used by our system learning system. The EM algorithm concerns the case where the *data are not fully observable*, but some of them are observable (Table 4) [De Raedt, 2008].

	a	**j**
e1	true	true
e2	true	?
e3	false	false
e4	?	true

Table 4 Dataset with missing values

Missing data occur when some values of the random variables are occasionally unobserved.

Latent state occurs when the values for some random variables are always unobserved. This state is illustrated in table Table 5. The latent variable is the random variable **a** because in all examples of Table 5 the values are unobserved.

	a	**j**
e1	?	true
e2	?	?
e3	?	false
e4	?	true

Table 5 Latent State

After we mention the above attributes that may have any dataset, we use the log-likelihood in the data which is illustrated in Table 4 . Let's we have the log-likelihood function with parameter λ,

called Q(λ). The most difficult with this function is its dependence on unobserved values, i.e. the unobserved values are represented in Table 4 as "?". According to [De Raedt, 2008] the natural way of dealing with these values (unobserved – observed) is to compute the expected likelihood function Q(λ). The expectation is taken over the missing values of the examples that corresponds to Table 4. Let's assume that we have examples of the following form:

$$e_i = x_i \cup y_i \ \ (1 \leq \mathrm{i} \leq 4)$$

That is, each example e_i $(1 \leq \mathrm{i} \leq 4)$ is composed of an *observed part* x_i and an *unobserved part* y_i. The *expectation maximization algorithm* consists of the following two steps.

Steps of *Expectation Maximization Algorithm.*

- **E-step**: called the *expectation step*. In this step, the observed data and the present parameters of the model M, compute a distribution over all possible completions of each partially observed data case.
- **M-step**: called the *maximization step*. This step of the algorithm is described [De Raedt, 2008] as follows, "in M-step using each completion as a fully observed data case weighted by its probability, compute the updated parameter values using frequency counting (these frequencies over the completions are called the *expected counts*)".

In order to use the expectation maximization algorithm we assume that there is a current model $M(\lambda_j)$ which is used to compute the expected values of unobserved part of dataset y_i (Table 4). In addition, it is used to compute $Q(\lambda)$. The next formula illustrates the computation of Q(λ). The **E** in the formula corresponds to the expectation which is taken with regard to the current model $M(\lambda_j)$ and the missing values y_i of dataset in Table 4. The $\boldsymbol{L}(\boldsymbol{\lambda})$, corresponds to the likelihood as a function of parameters λ which we have mentioned in Equation 2.2.

Equation 2.3

$$Q(\lambda) = \boldsymbol{E}[L(\lambda)] = \boldsymbol{E}\left[\log P\left(E\middle|M(\lambda)\right)\right] = \boldsymbol{E}\left(\sum_{e_i \in E} \log\left(P\left(e_i\middle|M(\lambda)\right)\right)\right) =$$

$$= \boldsymbol{E}\left(\sum_{e_i \in E} \log\left(P\left(x_i, y_i\middle|M(\lambda)\right)\right)\right) = \sum_{e_i \in E} P\left(y_i\middle|x_i, M\left(\lambda_j\right)\right) log P\left(x_i, y_i\middle|M(\lambda)\right)$$

According to the above formula (Equation 2.3), first need to compute the $P\left(y_i\middle|x_i, M\left(\lambda_j\right)\right)$. These values are computed by normal inference procedures. In the estimation step, we estimate the likelihood that the examples x_i are completed with unobserved part y_i given the current

model M (λ_j). When these estimates are known, the expected likelihood of function $Q(\lambda)$ is computed. We give an example that illustrates the steps of E.M algorithm.

Illustration of EM algorithm by an example.

Let have the dataset that is presented in Table 4.

E-step of E.M. algorithm: *The first step of E.M algorithm is to complete the data set*. As we mention, in the dataset there are missing values (in e_2, e_4 Table 4). More specifically, the result of the example e_2 is split into two examples $e_{2,1}$ $e_{2,2}$. The *first*, $\boldsymbol{e_{2,1}}$ has the value *true* for **j** (j corresponds johnCalls random variable) and receives the probability of P (j | a). The second, $\boldsymbol{e_{2,2}}$ has the value *false*, and receives the probability of P ($\neg j \mid a$). These fractional examples can be used for computing the expected counts and perform the maximization step. For example, in parameter λ_o can be re-estimated as (the same process corresponds to the all of parameters of the model $\lambda_0\lambda_1\lambda_2$):

$$\lambda_0 = \frac{ec(a)}{ec(a) + ec(\neg a)}$$

After we re-estimate the parameter λ_0 it follows the maximization step.

M-step of E.M. algorithm: As we mentioned, in this step we compute the updated parameter values using expected counts. So, $ec_\lambda(a)$ is the expected count of the number of occurrences of *a* given the current set of parameters $\lambda = (\lambda_0, \lambda_1, \lambda_2)$, now replaces |a| with the following:

$$ec_\lambda(a) = \sum_{i\in\{1,\dots,4\}} P(a \mid e_i) = 2 + P(a \mid e_4)$$

Where P (a | e_4) has to be estimated using λ, with the following:

$$P(a \mid e_4) = P(a|j) = \frac{P(a \wedge j)}{P(a)} = \frac{\lambda_0\lambda_1}{\lambda_0\lambda_1 + (1-\lambda_0)\lambda_2}$$

So, the above is the process that follows the E.M algorithm.

2.9 Basic Features of ProbLog

In this subsection we will discuss some basic features of ProbLog [De Raedt et, al., 2007]. As we mentioned earlier, ProbLog is a probabilistic extension of Prolog. It can be used as a standalone tool or as a library in Python. In our system, it has been used in connection with Python from the

library of Python. ProbLog allows us to encode the uncertainties that are inherent in real world applications. A ProbLog program consists -as Prolog- of a set of definite clauses. Moreover, in ProbLog every clause C_i is labeled with probability Pi. Let have the following example. Suppose there is a burglary in a house, with probability 0.7 and an earthquake with 0.2 probability. We have the following rules in pseudo-code.

- **if** there is a burglary and an earthquake **then** the alarm rings with probability 0.9
- **if** there is only a burglary **then** it rings with probability 0.8
- **if** there is only an earthquake **then** it rings with probability 0.1
- **if** there is neither a burglary nor an earthquake **then** the alarm doesn't ring.

In **Figure 5** illustrates the Bayesian Network of given example.

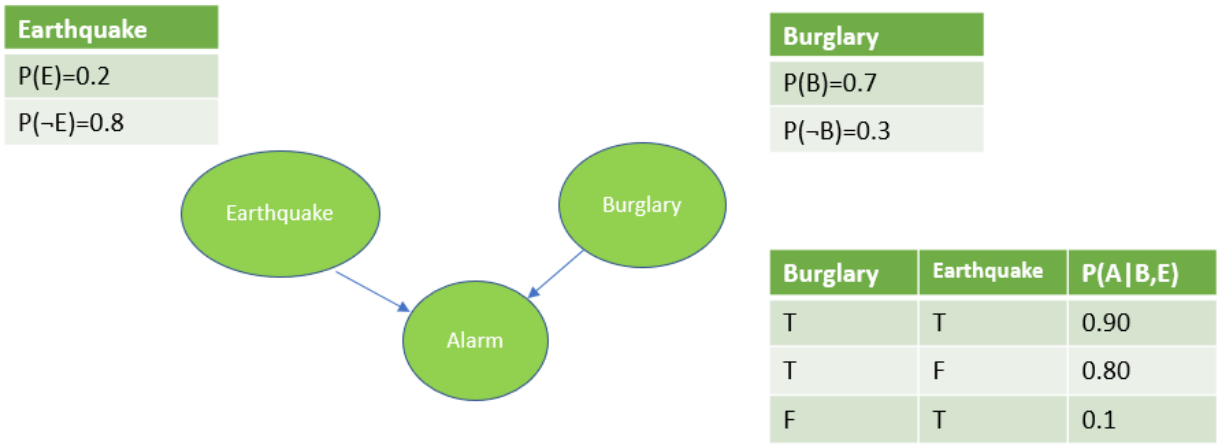

Figure 5 Burglary Example

The aforementioned rules in pseudo-code and the **Figure 5**, can be modeled in two different approaches. In the first approach, the modelling is performed by using probabilistic facts and rules. More specifically, **Program 1** contains *probabilistic facts* (called also *random variables*) and *rules.*

Program 2: Create rules using probabilistic facts.

%**probabilistic facts**

0.7::burglary.

0.2::earthquake.

0.9::p_alarm1.

0.8::p_alarm2.

0.1::p_alarm3.

%**rules**

%if there is a burglary and an earthquake the alarm rings with probability 0.9

alarm :- burglary, earthquake, p_alarm1.

%if there is a burglary only, it rings with probability 0.8

alarm :- burglary, \+earthquake, p_alarm2.

%if there is an earthquake only, it rings with probability 0.1

alarm :- \+burglary, earthquake, p_alarm3.

% we know that the alarm rang

evidence(alarm).

%what is the probability of a burglary

query(burglary).

The program **Program 2** can also represented in the follow link

https://dtai.cs.kuleuven.be/problog/editor.html#task=prob&hash=564f615f5f657deb2eacb993d59d69f6

In the second approach, the modelling is performed by using annotated disjunctions in rules. In this approach we can replace the clause "alarm :- burglary,earthquake,p_alarm1." by the following clause "0.9::alarm :- burglary, earthquake.".

In every rule, probability is placed in the head of clause. More specifically, **Program 3 models the same problem as the one of Program 2**, in addition it uses rules that are annotated with probability.

Program 3 Create rules with annotated disjunctions

0.7::burglary.

0.2::earthquake.

*%*probabilistic rules

0.9::alarm :- burglary, earthquake.

0.8::alarm :- burglary, \+earthquake.

0.1::alarm :- \+burglary, earthquake.

% we know that the alarm rang

evidence(alarm).

%what is the probability of a burglary

query(burglary).

The program **Program 3** can also represented in the follow link

https://dtai.cs.kuleuven.be/problog/editor.html#task=prob&hash=9d7aff7dc8facf16e617edcb256641ce

We know the E.M uses iteration to estimate the parameters of statistical models, so for that reason ProbLog uses three forms for defining the iteration. We represent these forms that use ProbLog. In other words, when learning the probability annotation in a probabilistic fact-random variable or rule- with ProbLog we can use three possible forms. We give an example to represent that three forms. Let have the following program **Program 4**.

Program 4 the possible forms that use ProbLog

t(_)::burglary.

t(0.50)::earthquake.

0.6::*alarm.*

Based on **Program 4,** we represent the following forms that can use in ProbLog program. These three forms are the following [De Raedt et, al., 2007]:

- **Probability to be learned corresponds to the form t(_)**: The form of "t(_)", as in the instance "t(_)::*burglary*", indicates that *the probability of this fact-random variable has to be learned from data*. In the first iteration of EM, each random variable is initialized with a random probability.
- **Intermediate value of the probability corresponds to the form t(p)**: The form "t(p)", as in the instance "t(0.50)::earthquake", indicates that *the probability of this fact has to be learned from data but in the first iteration the EM algorithm it has been assigned 0.5 probability*.
- **Fixed probability corresponds to the form p**: The form of "p", as in the instance "0.6::alarm" indicates that the *probability of this fact is fixed (it is not learned)*.

After we represent the possible forms that use ProbLog, now it is important to illustrate the example with a Bayesian Network with some unknown parameters. So, let we have the following network:

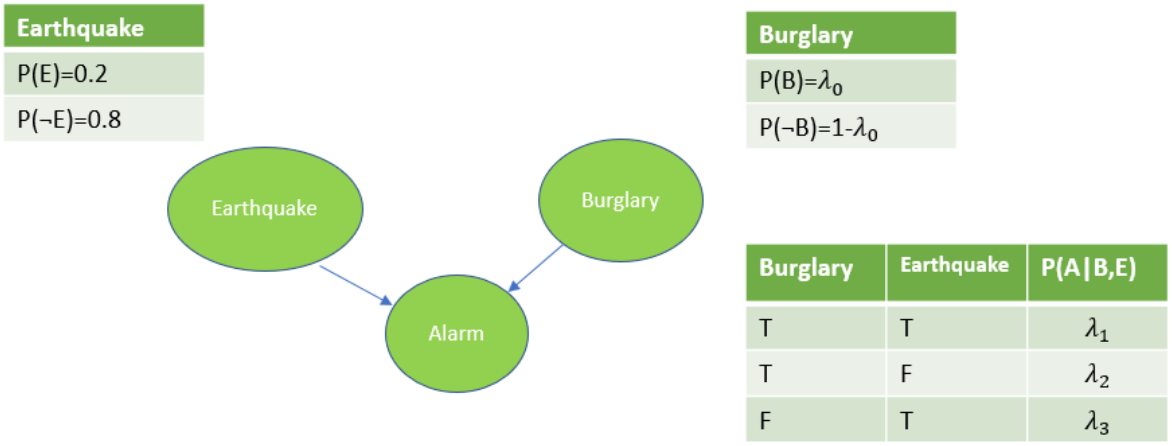

Figure 6 Bayesian Network with unknown parameters

This network **Figure 6**, corresponds model M, with the following below program.

Program 5 Untrained Network with annotated probabilities in every fact

t(_)::burglary.

0.2::earthquake.

t(_)::p_alarm1.

t(_)::p_alarm2.

t(_)::p_alarm3.

%the untrained rules

t(_)::alarm :- burglary, earthquake, p_alarm1.

t(_)::alarm :- burglary, \+earthquake, p_alarm2.

t(_)::alarm :- \+burglary, earthquake, p_alarm3.

Also, we have the following table as set of examples E.

	B	E	A
e1	false	?	false
e2	true	false	true
e3	false	?	?

Table 6 set of Examples E of Bayesian Network

The **Table 6** in ProbLog is represented as follows:

Program 6 Examples (specified as evidence, separated by ---)

% representation of the 1st row of Table 5.

evidence(burglary, false).

evidence(alarm,false).

% representation of the 2nd row of Table 5.

evidence(earthquake,false).

evidence(alarm,true).

evidence(burglary,true).

% representation of the 3rd row of Table 5.

evidence(burglary,false).

Based on the examples – evidences Table 6 that are mentioned earlier, we use lfi/2 function. This function stands for learning from interpretations. Generally, the interpretations are all probabilistic facts and evidences that are created. More specifically, the interpretations that use lfi/2 are the rules and the examples (evidences). So, lfi/2 takes the following as input:

- Evidences: is the set of examples that represent events that are true or false. More specifically, take the Program 6 as input.
- Rules-random variables are the rules and random variables of Bayesian Network corresponds to **Figure 6**.

When, lfi/2, use E.M. algorithm to estimate the parameters of the model **Figure 6** , then it returns the trained model as output. This trained model is illustrated in **Program 7**

Program 7 Trained Bayesian Network

%trained random variables

0.33::burglary.

0.2::earthquake.

0.27::p_alarm1.

1.0::p_alarm2.

0.35::p_alarm3.

%trained rules

0.39::alarm :- burglary, earthquake, p_alarm1.

1.0::alarm :- burglary, \+earthquake, p_alarm2.

0.0::alarm :- \+burglary, earthquake, p_alarm3.

Also, we can run the above program in the following link. If we run the program many times we will notice that we will have different results and this is because it does not do the same iterations in the learning process.

https://dtai.cs.kuleuven.be/problog/editor.html#task=lfi&hash=9f4b2a12d6828366886de83acef53156&ehash=e722d0902bfa7c3be7515561dba78c67

2.10 Related Work

Our work is related with Statistical Relational Learning (SRL) called also Probabilistic Logic Learning. As we have described earlier we use Bayesian Logic and ProbLog library in order to create a machine learning system. In this section, we will give a short brief review of machine learning systems that use SRL and other scientific methods such Deep Learning, NLP.

As it is known, SRL combines expressive representation formalisms, able to model complex relational networks. Some of the formalisms in this domain are logic programs with annotated disjunctions (LPAD) [Vennekens, et al., 2004], probabilistic horn abduction (PHA) [Poole, et al.,

1993], ProbLog [De Raedt et, al., 2007]. First, in the next paragraphs we represent systems that use Probabilistic Logic in order to create machine learning systems.

Probabilistic Soft Logic over the Social Graph: In [Jiwei, et al., 2014] proposed a framework that use probabilistic reasoning over the social network graph. This framework answers questions about Twitter users such the following:

- *Does this user like cheese cake*?
- *Is this user a Barcelona F.C. fan*?

According to the article, the above questions are answered by building a probabilistic model that reasons over the user's attributes in Twitter such as gender and home location and the social network of the user, such as the user's friends and spouse. It is mentioned that it is more likely that a user is a fan of Barcelona if he/she comes from Spain. Also, it is more likely that a user likes cheese cake if his/her spouse or a friend likes cheese cake. For extracting user's attributes i.e. gender, spouse, home location, and preferences (like-dislike) from text they use semi-supervised data harvesting and vector space models. For probabilistic reasoner they use Probabilistic Soft Logic. In short terms, we give an example how to extract the attribute Education/Job of a user. So, in order to identify the Education and Job attributes, it is mentioned in the article that they use Google+ API2 service. More specifically, for each user, they obtained his/her full name into the Google + account. Most of the users of Google+, they use many important attributes, such education and job that used in the aforementioned framework. Also, mention that, the most important challenge is to match user's Twitter accounts to Google+ accounts. In addition, they adopted the friend shared strategy. With this strategy, taken in that if more that 10 percent of and at least 20 friends are shared by Google + circles and Twitter followers, they assume that the two accounts point to the same person [Jiwei, et al., 2014].

Information about user's attributes and preferences in predicate form is specified as the following: Spouse(UsrA,UsrB), Friend(UserA,UserB), Like(UsA,Entity1).

In addition, in order to model complex relations, they use Probabilistic Logic such as the following formulas:

Friend(A,B)∧ Friend(B,C)=>Friend(A,C) friends of friends are friends

Couple(A,B) ∧ Friend(B,C)=>Friend(A,C) the friend of one member of a couple is also friend of the other member of the couple.

FRIEND(A,B) ∧ LKE-SPORTS(A) => LKE-SPORTS(B) If A and B are friends and a sport likes A, then that sport likes to his friend B.

The important advantage of the aforementioned framework is that it can be answer questions from different topics by extracting information from Twitter. On the other hand, the disadvantage is that the export of user attributes is based on web platforms that are not so popular like Google +.

In addition, important formalisms in SRL are the Bayesian Networks and Markov Logic. So, in the next paragraphs we discuss a brief review of some systems that use these formalisms in order to model complex real-life situations.

Using Bayesian networks in social networks: in article [Xu, et al., 2019] they propose a framework for identifying trustworthy users in social networks using the Bayesian approach. In other words, they use Bayesian Networks to model user profiles and historical records so that their system recognizes the trustworthy users. The general idea of the system was to employ user features in order to create a classifier by formalizing trust prediction as a classification problem. In addition, they mention that "we emphasize the inner reasoning of people when they endeavor to judge whether a person can be trusted based on their past records rather than focusing on the complex process of building a trust network". Also, they assumed all the aforementioned idea can be well defined in a directed graph (i.e. Bayesian Network). In this directed graph each of the nodes represent features extracted from user's records. Also, each edge of the Network corresponds to a cause-and-effect relationship. Finally, the main contributions of the aforementioned work is the following:

1. They apply the Expectation-Maximization (EM) algorithm, in order to handle the latent components.
2. In their experiments, they use two different datasets for evaluation of the model. One was from Facebook by [Cambria, et al. 2015] and the other was from Twitter by [Chen, et al., 2015].
3. They conduct, several experiments on Facebook and Twitter and compare the performance of their method with other machine learning algorithms (i.e. Random Forest [Cambria, et al. 2015], Naive Bayes [Amor, et,al., 2004], Decision Trees [Amor, et, al., 2004]).

Learning Bayesian Networks from data: in article [Jie, et al., 2002] they proposed algorithms that use an information-theoretic analysis to learn Bayesian net., from given data. More specifically, they developed a three-phase analysis algorithm called Three-Phase Dependency Analysis (TPDA). They mention that algorithm TPDA requires at most O (N^4) conditional independence tests to learn an N-variable Bayesian Network. In addition, they use the TPDA-Π algorithm (i.e. this algorithm expects an ordering of the nodes), and requires at most O (N^2) conditional independence tests to learn an N- variable Bayesian Network. The aforementioned two algorithms have been implemented in a Bayesian Network learning system called BN *PowerConstructor*. This system used in order to evaluate the aforementioned algorithms and the results show that these algorithms are efficient and accurate.

Modeling the Infectiousness of Twitter Hashtags: in the research [Skaza, et al., 2017] they proposed a system that applies dynamic and statistical techniques in order to quantify the proliferation and popularity of trending hashtags on Social Media. In addition, the social network used for this research is Twitter. More specifically, it is mentioned that using timeseries data reflecting actual tweets in New York City and San Francisco, they present estimates for the dynamics (i.e., rates of infection and recovery) of several hundred trending hashtags using an epidemic modeling framework coupled with Bayesian Markov Chain Monte Carlo (MCMC) methods. So, by using the Bayesian method their approach has two aspects. Firstly, it can quantify

the spread of certain trending hashtags on Twitter, and second, the same methodology can be used in a predictive context.

Probabilistic Inference in Twitter Data: in article [Rao, et at., 2016] they propose a framework called SocialKB. This system is used to model and justify social media posts to determine their truthfulness, a first step towards identifying emerging cyber threats. In addition, the SocialKB is based on Markov Logic Networks. Their framework develops a knowledge base in a cohesive manner regarding the social media posts (i.e. tweets) and actions of people (i.e. user's behavior). In the following bullets we describe the main aspects for the aforementioned framework.

- *Domain Expert defines the input KB*: initially, they use a domain expert/user for defining the input of the knowledge base (KB). This KB contains predicates and first order logic formulas.
- *Automatically generate evidences*: when the predicates are known, the system generates automatically evidences, based on the social media data.
- *Modeling tweets using a KB*: they define a set of different types of predicates in the knowledge base. The first type makes closed – world assumption. In that case, anything that is not proven to be true, it is assumed to be false. The second type of predicates makes open-world assumptions, (i.e. what is not known, it may or may not be true).

For example, the predicate *tweeted(userID,tweetID)*, states whether a user posted a specific tweet or not, the predicate *containsHashtag(tweetID,hashtag)*, is true when the particular tweet, i.e. *tweetID*, contains hashtags. Finally, for learning process (i.e. learn the weighs of formulas) they use Tuffy and it required 14 hours to learn the weights of formulas.

A deep learning framework for named entity recognition: In this research [Xusheng, et al., 2018], the authors present a novel method for bacterial named entity recognition. As they mention this application domain is very important because many human diseases have been associated with bacteria. The proposed method combines domain features and deep learning models. In other words, integrates the domain features in a deep learning framework combining two different neural network architecture. One is the long short-term memory (LSTM) that uses entire sequences of data (i.e. video or speech) and the other architecture is convolutional neural network. In the evaluation process, two different metrics are used. The first measure is when domain features are not added, and the second measure is when part of speech (POS) and dictionary features are added. In the first metric, F1-mesearue is 89.14% and the second is 89.7%. With these metrics prove that their model achieves an advanced performance in bacterial NER.

In the next paragraph we give a very short brief review of a system that uses Natural Language Processing and Linguistic Analysis in order to conduct opinion mining from Social Media.

Opinion Mining from Social Networks: a system which uses Twitter data for mining user opinions about products or services is proposed in [Khyati, et al. 2014]. That is, they present an approach to extract data from Twitter by performing linguistic analysis on them. The linguistic analysis is performed by using techniques of Artificial Intelligence and NLP. In addition, the user opinions

about products or services they classified as negative, positive and neutral. Some of the main aspects in this work are the following:

1. This system can classify each sentence in a review.
2. In addition, for each sentence, this system can recognize the subjects of the feeling and the feature(s) being described.
3. Finally, the aforementioned system, does not need a training set since it depends on linguistic analysis.

3 Overview of our Machine Learning System

In this chapter we will discuss the main features of our system. After that, we will compare the features of our system with the ones of related work. In the final subsection of chapter 3 we will discuss the limitations and future extensions of our system.

3.1 The Main Features of our System

In this subsection we will discuss the main features of our system. We use data from social media to train our model. As we have mentioned earlier, the social network that we used in order to learn our model is Twitter. As it is known Twitter is very popular social network nowadays. In addition,

users' data from Twitter are available for analysis. Every developer can use Twitter API for constructing web applications that use Twitter data. More specifically, in our system we use Tweepy, that is a python library [Tweepy, n.d.]. Initially, we construct a python program that get tweets about vacation in Crete based on hashtags. In other words, with our program, we collect the top tweets used by people that visited Crete or probably like to visit Crete in near future. In addition, according the web site [Best-Hastags, n.d.] the most popular hashtag that user is used for vacation in Crete, are the following: #welovecrete, #visitcrete, #creteisland, #crete, #travel. These hashtags are stored in a python list, and with iterative process we get tweets for each one of the aforementioned hashtags. After execution the aforementioned program, we collect more than 1000 tweets. In addition, for each tweet, our system gets the text, the tweet creation date and the user's home location, that we can defined as data elements. We use these data elements, to create and train the Bayesian Network. In order to get additional information from the collected data we use two scientific methods, that is Sentiment Analysis and Entity Recognition. The sentiment analysis is performed by incorporations into our system the Valence Aware Dictionary and sEntiment Reasoner called 'VADER' which is a Python package. The sentiment analysis concerns how much positive or negative is each tweet. For, sentiment analysis, the VADER tool, uses a sentiment lexicon. More specifically, VADER use a list of lexical features, e.g. words, which are generally labelled according to their semantic orientation as either negative or positive (how positive or negative a sentiment is). We used this tool because it is applied very easily to text type of social media and it doesn't require any training data [Hutto, et at. 2014] . For example, given the following input:

"I want to go in Crete, because it's a great place #visitcrete#creteisland"

The sentiment analysis tool returns positive sentiment score (range of score is -1 – 1) because the text contains positive words such "want" and "great".

The sentiment analysis tool returns positive sentiment score because the text contains positive words such "want" and "great". VADER returns a real value in the interval [-1, 1] where -1 stands for 100% negative sentiment and 1 stands for 100% positive sentiment. The other number are rated analogously, i.e. 0 stands for 50% negative sentiment and 50% positive sentiment.

Each user of Twitter can mention his/her home location. As we have mentioned earlier, the user's home location is one of the *data elements* that are used for the derivation of the Bayesian Network. In addition, our system refers to users who are not from Crete. So, for that reason, we have in a csv file all toponyms of Crete, i.e. a list of more than 500 names of villages and cities, and our system checks if the user of each tweet is from Crete or not based on these toponyms [data-gov, n.d.].

Our system uses the Entity Recognition method that we mentioned in Chapter 1. This method performs the task of information extraction that seeks to locate and classify named entities in unstructured text into some pre-defined categories. Our system uses the following pre-defined categories: *tourism, location_in_tweet, vacation, political_issues, hotel, restaurant*. The next example shows us how Entity Recognition works in our system. Let's assume that we have the following unstructured text:

"I want to go Crete, because it has very good motels"

Our system recognizes the following pre-defined categories in the above text:

- ***location_in_tweet***: this category is recognized on the text because the text contains the location "Crete".
- ***hotel*:** this category is recognized on the text because the text contains the related word "motel" which belongs to the more general category "hotel".

So, we construct algorithms that use the aforementioned two scientific methods (i.e. Sentiment Analysis, Entity Recognition) for generating the random variables and the evidence set based on tweets that are stored in CSV file. The data sentiment analysis is performed by using the VADER tool. This is done by creating in order to create the next two random variables.

- ***positiveSentiment***: this random variable is true when the tweet is positive. That is, it contains positive words such "good", "perfect", etc.
- ***negativeSentiment***: this random variable is true when the tweet is negative. That is, it contains negative words such "bad", "terrible", etc.

We have constructed an algorithm which performs sentiment analysis. It is shown in Chapter 5 Algorithm 1. After the sentiment analysis, our system perform Entity Recognition. In order to perform Entity Recognition it creates some extra random variables. These extra random variables are the following:

- ***hotel***: if a tweet contains some word related to 'hotel' then this random variable, i.e. category, is true.
- ***political issues***: if a tweet contains some word related to 'political issues' then this random variable, i.e. category, is true.
- ***health care***: if a tweet contains some word related to 'health care' then this random variable, i.e. category, is true.
- ***vacation***: if a tweet contains some word related to 'vacation' then this random variable, i.e. category, is true.
- ***visit***: if a tweet contains some word related to 'visit' then this random variable, i.e. category, is true.
- ***travel***: if a tweet contains some word related to 'travel' then this random variable, i.e. category, is true.
- ***restaurant***: if a tweet contains some word related to 'restaurant' then this random variable, i.e. category, is true.

All tweets of the training set are stored in a CSV file. This file is given as input into our system in order to derive some of the aforementioned random variables. Moreover, our system uses the method of incremental learning. This mean that our system can enhance the derived model by applying additional training on the model. This is a dynamic training technique. It is mentioned in [Gepperth, et al. 2016] that "dynamic technique is the one that can be applied when input data is continuously used to extend the existing model's knowledge (to further train the model)". More details about all of features of our model in Chapter 5.

3.2 Comparison of Our System with Related Work

Our work capitalized knowledge on Information Extraction on social Media and Statistical Relational Learning, offering a unique combination of the aforementioned technologies. We know that many researchers use Twitter API to export user data. The most valuable attributes that a user can have in social media are *location, gender, education/job, published posts* (tweets, Facebook posts). In our system, we use *user home location* because we want to know if a user is from Crete or not. At the stage of prediction, our system uses published posts (tweets) from users and performs opinion mining about visiting Crete. The derived opinion is a possible value because it is given its truth probability.

In statistical relational learning, many researchers use Bayesian Logic and Markov Logic in order to model uncertainty. These methods can be used to answer queries such as the following:

- "*will this user visit Crete*?"
- "has this *patient flu*?"
- "will *it rain tomorrow*?"

The most popular formalisms for modeling the uncertainty are the following: a) *logic programs with annotated disjunctions* (LPAD) [Vennekens, et al., 2004], b) *probabilistic horn abduction* [Poole, et al., 1993] and c) *ProbLog* [De Raedt et, al., 2007]. Our system uses ProbLog to model uncertainty. The main task of this research work involves the development of a module which *creates automatically evidence set and rules* based on tweets that are stored in the CSV file. Evidences and rules are the basic features that use probabilistic graphical models for inference and learning procedures. In our system, we have used Bayesian Networks in order to model uncertainty due the important benefits of this method. The advantages of this method are the following.

- **Suitable from small and incomplete data**: It is demonstrated in [Kontkanen, et al. 1997] that Bayesian Networks can show good prediction accuracy even with few samples in the training set. In addition, according to Myllymäki there are no minimum sample size which is required in order to perform the analysis [Myllymäki, et al. 2002]. We considered that it is very important for our approach to be effective even with a small sample of data. In order to estimate the conditional probabilities of the Bayesian model the E.M algorithm is used. The E.M. algorithm requires only the model structure to be known, not the estimation of the parameters. Then, given the data and the structure of the model it iteratively calculates the maximum likelihood estimates for the parameters.
- **Structure learning possible (as future work)**: structure learning, is the process that can use any Bayesian Network. So, it is possible to use data to learn the structure of a Bayesian Network (BN). Our system is given the structure of the Bayesian Network in order to derive the trained model. The derivation of the structure of the BN from the data is left as extension of the system in future work. Specifically, we could apply this scientific method in order to create the conditional dependencies between the random variables, based on training set i.e. tweets. As it is known, the conditional dependences in Bayesian Network are defined by the

structure of DAG. So, in other words, we can use structure learning problem (as future work) in order to create the DAG of the Bayesian Network based on the training set.

- **Combining different sources of knowledge**: The most important feature of Bayesian Models is the use of *prior information*. It is mentioned in [Uusitalo, 2007] that "***priors*** reflect our knowledge of the subject before the research is conducted, and can be either highly informative and detailed, in case there is a lot of knowledge about the subject already, or very uninformative, if not much is known. These priors are then updated with data, to obtain a synthesis of old knowledge and new data. This synthesis can then be used as a prior in a new study. This mechanism makes the scientific learning process explicit, and also makes the assumptions made by the scientists transparent and open to discussion". These networks, have the advantage that they can combine expert knowledge with the data. These data could have been created by different sources of knowledge.
- **Fast Responses**: Once a Bayesian Network is compiled, it can provide fast responses to queries. In other words, a BN contains a conditional probability distribution for every random variable in the network which is used by the inference procedure. In addition, it can provide any probability distribution instantly. This has been taken into account in our system, because a lot of new data (new tweets) are given to the system for prediction and we would like to get immediate response from the system. Therefore, we think this feature as a very important advantage of Bayesian Networks.

3.3 Limitations and Extensions of our System

Systems that use machine learning techniques usually have some limitations. The most common limitation that a machine learning system has is the dataset that use for training. More specifically, the training set, may not have enough information to properly train the model. We identified some limitations during our research. In Twitter, each user is assigned a user location in a text field. This location is the user's home location, but we cannot confirm that the user lives in that place. This is the most important limitation in our work because inaccurate location declaration by the user can result in incorrect predictions. For the training of our model we used specific hashtags from the tweeters of the training set, i.e. #visitcrete, #travel, #Crete, #creteisland [Best-Hashtags, n.d.]. We have selected these hashtags as the most appropriate for training the model, on the other hand users of Twitter may have used different hashtags when are intended to visit Crete. In other words, it is not certain that the user who is going to visit Crete will use the aforementioned tags in his/her tweets.

In this section we will discuss future extensions of our system. These extensions are discussed in the following paragraphs:

- ***Multiple Bayesian Models***. We can create multiple Bayesian Models in order for our model to be able to answer queries on different topics. For example, we may have a Bayesian model that can answer questions such the following "is this user intended to visit Paros?" for tourism topic. Also, may have Bayesian Model that can answer questions as the following one "Is this user suffering from depression?" for a medical topic.

- ***Structure learning***. In our system, the structure of the Bayesian Network is given as part of the design of the system. The development of a structure learning procedure from the training set has been left as extension to our system in future work. More specifically, we could give to our system, a training set of tweets and the system will derive the structure and the parameters of the Bayesian model.

4 Architecture of our Machine Learning System

In this chapter, we will discuss the architecture of our machine learning system. As we mentioned before, a Python script has been constructed that it gets all tweets related with vacation in Crete, with specific hashtags. The Python program stores all tweets (more than 1000) in a CSV file. The first module of our system is called *pre-processing phase*. In this module, initially our system gets tweets as input in the format of an array of Python. Then, it returns as output *untrained rules* and *random variables*. The untrained rules and the random variables are derived by our algorithms. The next module of our system, called *the training phase*, takes as input the untrained rules and random variables and it returns as output the *trained rules* and *random variables*. These two modules are represented in Figure 7.

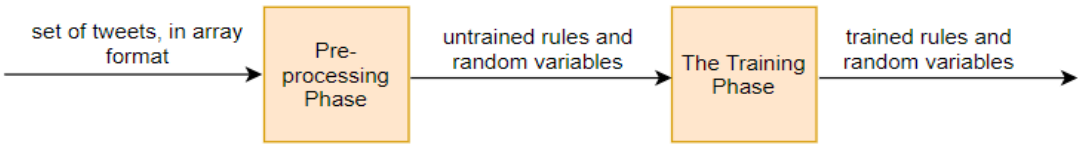

Figure 7 The two modules of our system

As we mentioned earlier, our system performs incremental learning. This feature of our learning system is very important because it can improve the derived model by applying *incremental training*. In this way the trained model is improved in a stepwise manner. The incremental retraining of the trained model is depicted in figure, Figure 8 **Error! Reference source not found.**

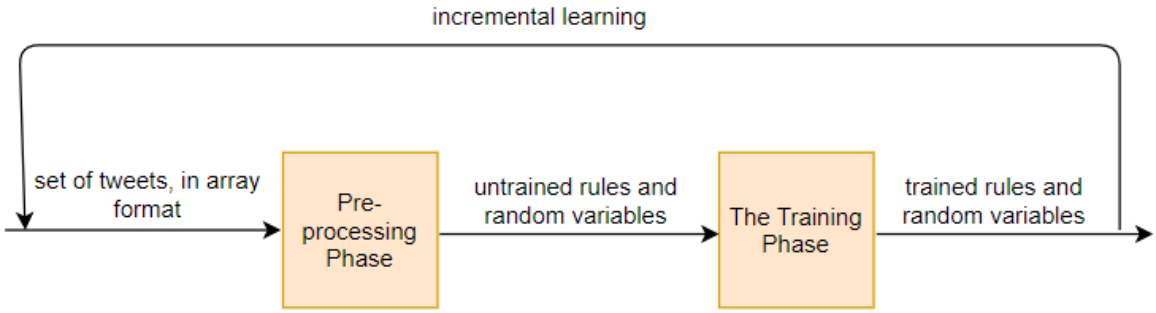

Figure 8: Incremental training of the model

After the derivation of the trained model, our system takes as input new tweets for evaluation by the trained model. This is the prediction stage. The output of this module is the prediction of the model by giving the probability that the tested user will visit or not Crete in the near future or he/she has already visited Crete. The overall architecture of our system illustrating both stages, i.e. the training stage and the prediction stage, is illustrated in figure, Figure 9.

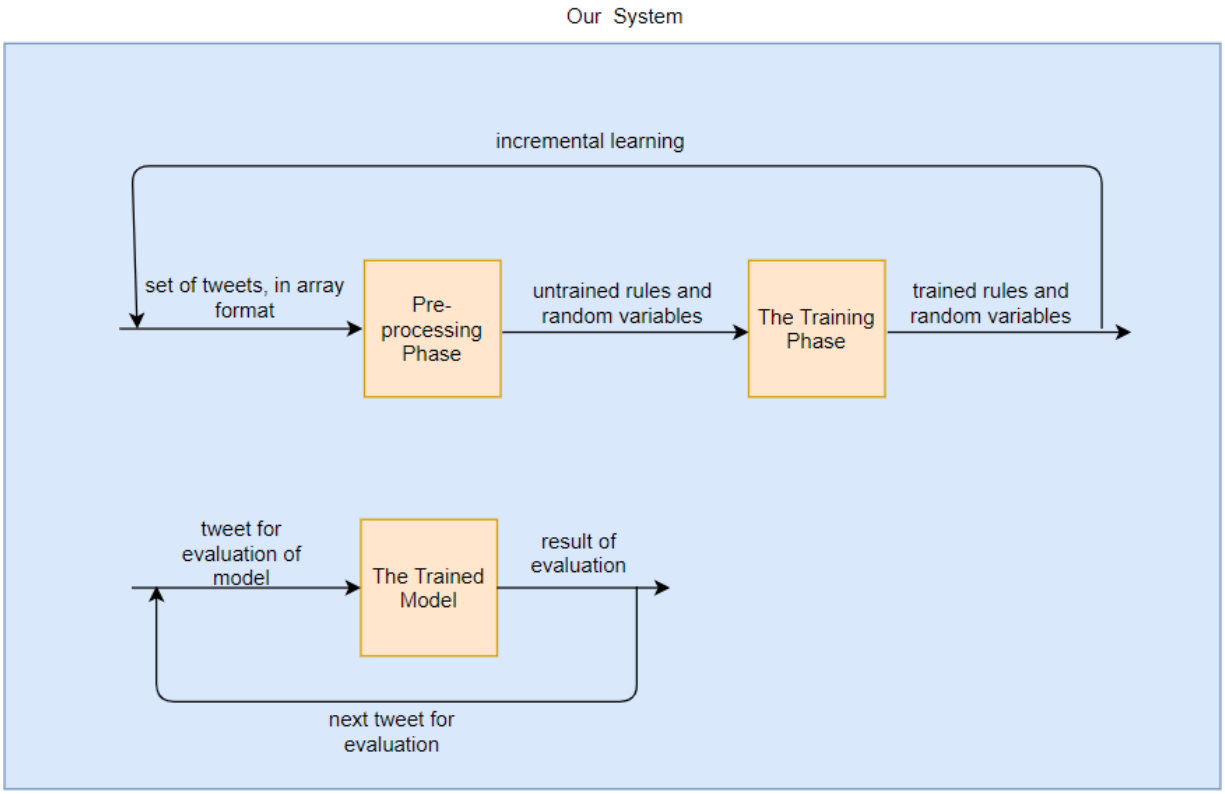

Figure 9 Overall architecture of our system

5 Detail Presentation of the Components of our Machine Learning System

In this chapter we will discuss thoroughly the components of our machine learning system. In the first section we will present the preprocessing phase. In the second section we will present the

training phase. In the third section we will discuss and present the trained model. In the final section we will discuss implementation issues and we will illustrate all algorithms that we have constructed for our system.

5.1 The Pre-processing Phase

The most important phase of our system is the preprocessing phase. We have constructed algorithms that call procedures (i.e. methods in object-oriented terminology) which perform *Sentiment analysis and Entity recognition* in order to create automatically random variables and rules based on the given train set. Initially, the algorithm that performs Sentiment Analysis, which is used to create random variables based on the sentiment of each tweet. More specifically, the sentiment analysis algorithm uses the VADER tool and it can create automatically up to two random variables. We have also constructed an algorithm that calls the Entity Recognition procedure, i.e. a method in object-oriented terminology. With this algorithm our system can identify some pre-defined categories in unstructured text, i.e. text in tweets. These categories correspond to the random variables that our algorithm creates. After the execution of the code corresponding to these two algorithms, ProbLog is used to estimate the parameters of the untrained model. More specifically, all random variables based on tweets that are stored in table Table 9 are created in the preprocessing phase of our system.

5.2 The Training Phase

In this subsection we will present the training phase. In this phase we use the ProbLog tool. More specifically, we call the lfi/2 function. This function, takes as input rules and evidence set and it returns the trained model. This function uses the Expectation Maximization algorithm in order to perform training of the model. In this phase, we use ProbLog tool only. The function lfi/2, takes as input the program Program 8 and the table Table 7. As we mentioned earlier, this function lfi/2 uses the Expectation Maximization algorithm to get the parameters of the random variables. After the estimation process, lfi/2 returns the trained random variables and rules as output Program 9, this program is illustrated in the next section.

Program 8: The untrained random variables and rules

t(_)::*userLocation.*

t(_)::*positiveSentiment*.

t(_)::*negativeSentiment*.

t(_)::*location*.

t(_)::*vacation*.

t(_)::*tourism*.

t(_)::*hotel*.

t(_)::*restaurant*.

t(0.33)::visitLocation :-

userLocation, positiveSentiment, tourism, location_in_tweet.

t(0.31)::visitLocation :-

userLocation, positiveSentiment, location_in_tweet.

t(0.23)::visitLocation :-

userLocation, positiveSentiment.

t(0.13)::visitLocation :-

userLocation, positiveSentiment, vacation, tourism, location_in_tweet.

Evidence set instance
{(userLocation, True), (positiveSentiment, True), (negativeSentiment, False), (location, True), (vacation, False), (tourism, True) }
{(userLocation, True), (positiveSentiment, True) }

{(userLocation, True), (positiveSentiment, True), (negativeSentiment, False), (location, True), (vacation, False), (tourism, False), (health_care, False)}

Table 7 Instance of evidence set

5.3 The Trained Model

Our system uses the lfi/2 function of ProbLog in the training phase. The lfi/2 function gets as input the random variables, rules, and an evidence set. The lfi/2 function performs machine learning by applying the E.M. algorithm for training the model. Actually, it estimates the parameters of our Bayesian network. This function returns the following program as output Program 9.

Program 9 Instance of trained model

```
0.96::userLocation.

0.94::positiveSentiment.

0.05::negativeSentiment.

0.30::vacation.

0.15::hotel.

0.42::tourism.

0.10::restaurant.

0.75::location_in_tweet.

0.45:: visitLocation :-

    userLocation, positiveSentiment, \+negativeSentiment,location, \+vacation, tourism.

0.17::visitLocation :- userLocation,positiveSentiment.

0.15::visitLocation :- userLocation,  positiveSentiment, \+negativeSentiment, location,

                \+vacation, \+tourism, \+health_care.
```

The derived trained model is illustrated by the figure, Figure 10 .

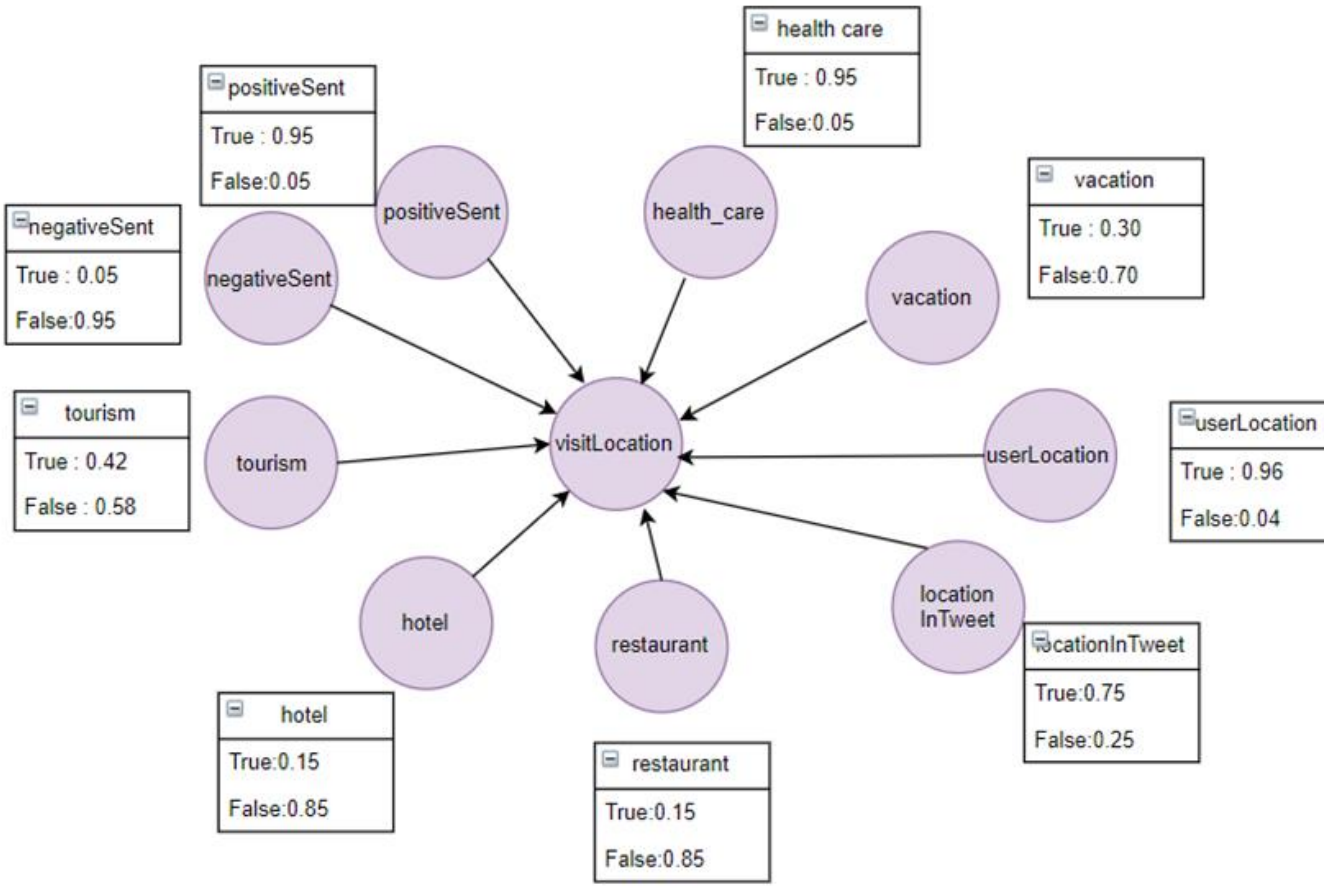

Figure 10 Trained Bayesian Network

5.4 Implementation issues

In this subsection, we will present the implementation issues of our system. Initially, it is important to mention the main programming components of our system.

- "readCSV.py": We have a CSV file, "CSV_tweets.csv", which has more than 1000 tweets. Sample transactions of this file are illustrated in **Error! Reference source not found.**. The program "readCSV.py" reads the CSV file "CSV_tweets.csv" row by row, and then it stores each element of the row of the CSV file, in a Python array. We essentially convert

the CSV file "CSV_tweets.csv", i.e. **Error! Reference source not found.** , to a Python array, i.e. Table 9.

- "createModel.py": This program has the code which implements the algorithms that perform Sentiment Analysis and Entity Recognition and they create automatically the evidence set, the random variables and the rules.

5.4.1 Read CSV and store data into array

The tweets that are used to train our model are stored in a CSV file called "CSV_tweets.csv". A sample of entries from that file are illustrated in the following table Table 8**Error! Reference source not found.**.

Tweets	Username	Create_at	User Location
Last dawn on the island of Crete#timeline#visitCrete	Christof34	9/7/2019 6:09	Texas
Sunset in Axus #sunset#rethymno	Jeanne_8j	10/7/2019	Pretoria
September on the beach, is great#rethymno	mako84	3/8/2019 17:32	China

Table 8 CSV_tweets.csv

The first stage of the preprocessing phase is to read the csv file "CSV_tweets.csv" with the program "readCSV.py", and then it stores all tweets in an array format. A sample of entries from the created array are shown in Table 9. The array of this table has been created by the procedure "readCSV.py".

text	username	create_at	userLocation	hashtags
Last dawn on the island of Crete	Christovb7000	9/7/2019 6:03:27 AM	Texas	#timeline #visitCrete
Sunset In Axus	Jeanne_8j	10/7/2019 6:03:27 AM	Pretoria, South Africa	#sunset #rethymno #crete
September On The Beach, is great	mako671178	3/8/2019 6:03:27 AM	China	#rethymno #create #seascape

Table 9 Array of tweets for processing(example)

5.4.2 Implementation of random variables and rules

After the construction of the array of tweets (Table 9), a group of algorithms create automatically random variables and rules. As it is known a discrete random variable X in a sample space Ω is a function which maps every element of the random experiment to a value, usually a real value. For each random experiment a random variable gets an element ω where $\omega \in \Omega$ and maps it a value.

The elements of Ω are usually mapped to a set of (real) values. For example, the random variable userLocation takes as values places names from everywhere in the world and returns True if the place of the user is not in Crete otherwise it returns False. For example, for values such as London and Athens the userLocation returns the value True, i.e. userLocation(London) = True and userLocation(Athen) = True. On the other hand, for places like Kisamos, Sitia it gets the value False, i.e. userLocation(Kisamos) = False and userLocation(Sitia) = False. We express it in a short way by saying that the random variable gets the value True or False accordingly. In addition, the random variable hotel, takes as values synonyms words and phrases of hotel. For example the word 'motel' is synonym word of random variable hotel such the following hotel(motel) = True. In the other hand the word girl is not synonym of hotel, and that express it such hotel(girl)=False. Also, the random variable vacation, takes as values synonyms words and phrases of vacation. For example, the word 'holiday' is synonym word of vacation, and illustrated as follows vacation(holiday)=True. In the other hand , the word test is not synonym word of vacation, that represented as follows vacation(test)=False. In our case, the set of values of random variables are synonyms of random variables (i.e. hotel, health, political issues, vacation) and toponyms of Crete.

As we mentioned earlier, the discrete random variables are created automatically using the algorithms that we constructed and we present below. The algorithm Algorithm 1 creates random variables for sentiment analysis (positive - negative). Initially, in list *randomVariables* the random variable ***userLocation*** is entered. This random variable consists the user's home location. Then, the VADER sentiment analyzer is called, it takes the array of tweets as input Table 9. If there is a negative sentence-tweet in the array of tweets, then the function returns -1 and creates the random variable ***negativeSentiment***. Also, if there is a positive sentence-tweet in the array of tweets, then the function returns 1 and creates the random variable ***positiveSentiment***. We illustrate below the algorithm, Algorithm 1, for sentiment analysis.

```
procedure sentimentIdentificationWithVADER(in: arrayOfTweets; out: randomVariables)
begin
      randomVariables  := ["userLocation"];
      if (sentimentAnalizer(arrayOfTweets) = 1) then
                randomVariables :=  append(randomVariables, [ "positiveSentiment"]);
      if (sentimentAnalizer(arrayOfTweets)  = -1) then
              randomVariables :=  append(randomVariables, [ "negativeSentiment"]);
end
```

Algorithm 1 Sentiment analysis with VADER

- The variable *negativeSentiment* is created if the function of VADER after sentiment analysis of tweet returns number in range [-1,0), this random variable gets the value true.
- The variable *positiveSentiment* is created if the function of VADER after sentiment analysis of tweet returns a number in range (0,1], this random variable gets the value true.

- If the function of VADER after sentiment analysis of tweet returns 0, this tweet is further processed in order to be classified as either positive or negative.

Summarizing for the random variables *positiveSentiment* and *negativeSentiment*, when the program that we have created finds in array of tweets Table 9 positive sentence-tweet it returns a value for the random variable ***positiveSentiment.*** Also, when it finds negative sentiment-tweet it returns a value for the random variable ***negativeSentiment***. When VADER returns 0, our system use the additional insights of Entity Recognition method for classification of tweet.

The algorithm Algorithm 2 categorizes tweets based on pre-defined categories. More specifically, the Algorithm 2 it gets the array of tweets as input and it returns a subset of the random variables that are identified. Each of these categories corresponds to a random variable. The remaining random variables – categories are the following: ***location_in_tweet***, ***tourism***, ***vacation***, ***health care***, ***political issues***, ***hotel*** and ***restaurant***. The algorithm Algorithm 2 is based on the idea of the Entity Recognition method. As it is known, this task is a task of information extraction that seeks to locate and classify named entities mentioned in unstructured text (such tweet, newspaper) into some pre-defined categories. The categories that our system uses are presented in the following bullets.

- ***location_in_tweet***: If in the text of the tweet is mentioned some location of Crete, then our program will identify it.
- ***tourism***: If in the text of the tweet is mentioned some word related to tourism then our program will identify it.
- ***vacation:*** If in the text of the tweet is mentioned some word related to vacation then our program will identify it.
- ***health care:*** If in the text of the tweet is mentioned some word related to health care domain then our program will identify it.
- ***political issues:*** If in the text of the tweet is mentioned some word related to political issues then our program will identify it.
- ***hotel*** : If in the text of the tweet is mentioned some word related to hotels then our program will identify it.
- ***restaurant*** : If in the text of the tweet is mentioned some word related to restaurants then our program will identify it.

Initially, we will describe how our program "createModel.py", identify the above random variables-categories in tweets based on Algorithm 2. We have a CSV file, called "synonymsTable.csv", that is illustrated in Table 10. All the random variables-categories are presented in the first row of the file "synonymsTable.csv". In the remaining rows of the csv file "synonymsTable.csv" Table 10 are possible values of the categories that presented in the first row [Power-thesaurus, nd]. In other words, for each random variable the column of its synonym words corresponds to its domain. i.e. the values of the random variable. Possible values stands for words with similar meaning. In the category ***location_in_tweet*** we have a list of possible values which

do not have similar meaning. The only similarity they have is that all are locations-toponyms of Crete, i.e. villages, cities, places etc.

	A	B	C	D	E	F
1	**tourism**	**location_in_tweet**	**vacation**	**political_issues**	**hotel**	**restaurant**
2	travel	Sfakia	holiday	policy issues	temporary	eating place
3	sightseeing	Paleochora	leave	politics	room	eating place
4	tourist	Kissamos	break	political matters	flat	eating house
5	tour	Frangokastello	rest	political aspects	house	place
6	tourist industry	Kolimbari Village	time off	policy concerns	hostel	eatery
7	visitors	Kalyves Village	recess	political agenda	place	grill
8	tourists	Platanias Village	leisure	policy issue	home	diner
9	journeying	Anopolis	vacationing	policy questions	inn	eating place
10	leisure industry	Akrotiri	holidays	political affairs	lodge spot	bistro

Table 10 Synonyms table for Entity Identification

```
procedure tweetsClassification(in: arrayOfTweets, synonymsCSVtable; out:
categoriesList )
begin
 for each tweetText ∈ arrayOfTweets do
    for each rowSynonyms ∈ synonymsCSVtable do
               for fieldValue := 1 to size_of_rowSynonyms do
                   if (prefix(tweetText, rowSynonyms[fieldValue]) or
                          infix(tweetText, rowSynonyms[fieldValue]) or
                          postfix (tweetText, rowSynonyms[fieldValue])) then
                              if fieldname ∌ categoriesList then
                                categoriesList := append(categoriesList, fieldname);

end
```

Algorithm 2 create random variables-categories

Note: the functions prefix/2, infix/2, and postfix/2 defined as follows. These functions, take a tweet tweetText as input, and the value of a field of a row, i.e. rowSynonyms[fieldValue], from the table. They return true if the value of field of the row exists at the beginning, in the middle or at the end of the tweet. In that case, the corresponding value of the field is inserted in the list categoriesList (if doesn't exist already).

We give an example of Algorithm 2. Let's we have the following tweet that is stored in table Table 9.

'*I want to go in Crete because it is wonderful place #visitCrete#travel*'

The algorithm, Algorithm 2, will be tested if there exists a value of a random variable, such as ***tourism***, ***vacation***, ***health care***, ***political issues***, ***location_in_tweet***, in the above text. In that case,

the algorithm will identify the random variable(s) whose values have been found in the text of the tweet. In this case, values for the random variables ***location_in_tweet*** and ***tourism*** have been found. That is:

- The algorithm, **Error! Reference source not found.**, finds that the text contains the value 'Crete' , so,the random variable (category) ***location_in_tweet*** because it refers to a location in Crete.

It also finds that the text has two values 'visit' and 'travel' of the random variable (category) ***tourism***.

Then algorithm which finds values for the remaining random variables is the algorithm Algorithm 3. That is, this algorithm finds values for the RVs ***vacation***, ***health care***, ***political issues***, ***hotel*** and ***restaurant***. The Algorithm 3**Algorithm 3** if it finds some values of the aforementioned random variables in the tweet (array of tweets - Table 9) then it assigns them to the corresponding random variable only if the RV ***tourism*** has already been created.

```
procedure   createRandomVariables(in arrayOfTweets; out randomVariables)
begin
  for each tweetText ∈  arrayOfTweets do
    for each rowSynonyms ∈ synonymsCSVtable do
       for fieldValue := 1 to size_of_rowSynonyms  do
           if (prefix(tweetText, rowSynonyms[fieldValue])  or
               infix(tweetText, rowSynonyms[fieldValue])  or
               postfix(tweetText, rowSynonyms[fieldValue])) then
                    if fieldname ∌ randomVariables  then
                       randomVariables:=append(randomVariables,  fieldname);

end
```

Algorithm 3 Create Random Variables with E.R method

The code of the algorithms **Error! Reference source not found.**, **Error! Reference source not found.**, and Algorithm 3, is included in the Python program "createModel.py". When this program runs it creates the following Bayesian Network, It is shown in Figure 11 Bayesian Network of our system**Error! Reference source not found.** with the corresponding random variables.

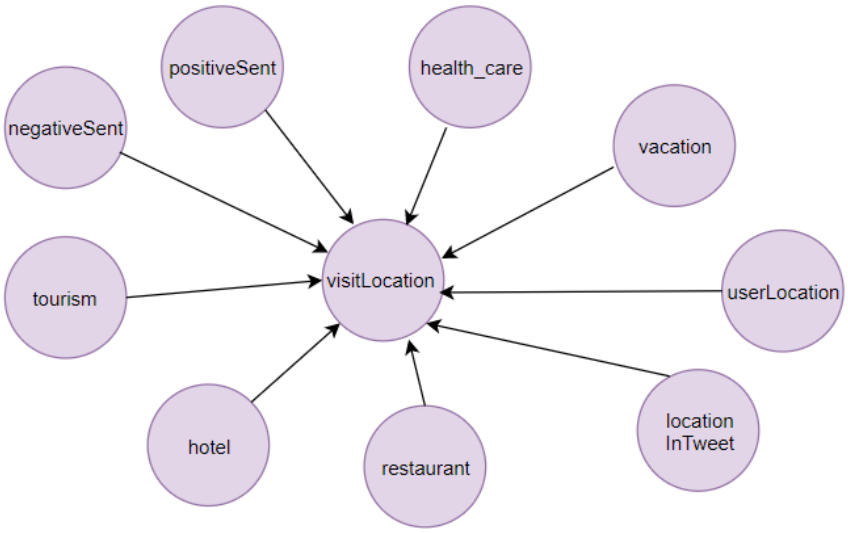

Figure 11 Bayesian Network of our system

The evidence set is an array that contains the random variables and their values True-False. That is, the values they map to the values of the experiment as they have been discussed above. After the execution of algorithms **Error! Reference source not found.**, **Error! Reference source not found.** and Algorithm 3

The evidence set is returned by the program which implements the algorithms **Error! Reference source not found.**, **Error! Reference source not found.** and Algorithm 3. The evidence set has values of each one of the random variables.

The general form of the *evidence set* is as follows:

{(***userLocation***, Value), (***positiveSentiment***, Value), (***negativeSentiment***, Value), (***location_in_tweet***, Value), (***vacation***, Value), (***tourism***, Value) , (***hotel***, Value) , (***political_issues***, Value) }

The possible values of a RV are True, False or Null. If a random variable in the evidence set does not appear in instance of the evidence set then it means that the corresponding instance of the evidence set has a value Null.

Table 11 contains in each row an instance of the above general form of the evidence set as well as the number of occurences.

We give an example for that process. Let have the following text:

' *I booked tickets for Chania today #holiday#visitCrete*'

The evidence set with the random variables and their values is shown below:

{(***tourism***,True), (***health_care***,False), (***location_in_tweet***,True), (***vacation***,True)}.

Evidence set instance	Number of occurrences
{(userLocation, True), (positiveSentiment, True), (negativeSentiment, False), (location, True), (vacation, False), (tourism, True) }	975
{(userLocation, True), (positiveSentiment, True) }	375
{(userLocation, True), (positiveSentiment, True), (negativeSentiment, False), (location, True), (vacation, False), (tourism, False), (health_care, False)}	330

Table 11 Evidences (example)

In order to create the evidence set, (i.e. Table 11) , **Error! Reference source not found.**, **Error! Reference source not found.** and Algorithm 3 runs for every tweet. The second column of the Table 11 has the repetitions of the occurrences of each evidence set instance in the array of tweets, i.e. Table 9. The number of occurrences of each instance of evidence set is used to give each rule a probability. The probability of each rule is derived as follows:

probabilityOfRule = numberOfOccurances/sizeOfArrayOfTweets

"numberOfOccurance" is the number of occurrences of each evidence in the array of tweets and "sizeOfArrayOfTweets" is the size of the array of tweets. The size of the array of tweets corresponds the sum of all tweets in dataset. So, the probability of the first rule of the constructed modes is derived from the corresponding evidence set in Table 11 as follows:

probabilityOfRule = 975/2145=0.45.

For each element of the table a rule is constructed. The construction procedure is illustrated by algorithm, Algorithm 4

```
procedure   createRulesBasedOnEvidence (in evidenceSet; out rules)
begin
```

```
  rules:= [ ] ;
  headOfRule := ‘’;
  for each (instanceOfEvidence, numberOfEccurances) ∈  evidenceSet do
    probabilityOfRule = numberOfOccurances/sizeOfArrayOfTweets
    stringConcat(probabilityOfRule,’visitLocation:-’,headOfRule)
    rules := append(rules,headOfRule)
    for each (randomVariable,Value) ∈ instanceOfEvidence do
        if value = True then
           rules:=append(rules, [randomVariable])
        else
           rules: =append (rules, [\+randomVariable])

end
```

Algorithm 4 Create rules based on evidence set

5.4.3 Implementation of incremental learning method

Our system uses an incremental learning procedure. We constructed an algorithm, Algorithm 5, that applies incremental learning in the trained Bayesian network model. The trained model is stored in txt file called “model.txt”, also the evidence set is stored in an array of Python language called “evidence.npy”. More specifically, the ‘model.txt’ file contains all trained random variables and rules of our system. The “evidence.npy’ array, contains all evidences that were used for constructing the trained model ‘model.txt’. Our system uses incremental learning, to further train the model over time with additional data sets. In other words, we give a new train set to the trained model in order to be trained further in an incremental way. The new training starts from the point where the previous training had stopped. It is not done from the beginning all over again with a larger data set. Our system applies a clear Incremental Learning method. The trained rules, they are based on the random variables, and the new evidence set are created by using the algorithms Algorithm 1 and Algorithm 2 and with the function of ProbLog lfi/2. Let’s call the new trained rules “newModel” and the new evidence set “newEvidenceSet”. At this point, our system has two trained models and two evidence set for this reason we have the following:

- “newEvidenceSet”: this set contains all the new evidences that emerged from the new training data.
- “evidenceSet.npy”: this set contains all the previous evidences in order to train the previous “model.txt”.
- “setOfRandomVariablesOfModel”: this set contains the random variables of the previous model “model.txt”.
- “setOfRandomVariablesOfNewModel” this set contains the random variables of the new model “newModel”.

The incremental training is performed by algorithm Algorithm 5. The algorithm get as input two sets of random variables (the random variables of the model “model.txt” and the one of the

"newModel"). It also takes as input two evidence sets, i.e. "evidenceSet.npy" and "newEvidenceSet" and it returns the final trained model. The first step of this algorithm is to join the two arrays of evidences into one (finalEvidenceSet). The second step is to create the updated probabilities of random variables based on the two sets that we mentioned earlier, i.e. setOfRandomVariablesOfModel and setOfRandomVariablesOfNewModel. Lets assume that we have the following use case of Algorithm 5:

numberOfSamplesOfModel: is the number of previous samples(tweets) that was trained the model 'model.txt'

numberOfSamplesOfNewModel: is the number of new samples (tweets) that is trained the model "newModel".

We have the trained model "model.txt" that was trained with 900 samples-tweets. Also, we have the trained model "newModel" that was trained with 500 samples. So, the total samples-tweets of two modes that was trained is 1400.

totalSamples = numberOfSamplesOfModel + numberOfSamplesOfNewModel = 900 + 500 = **1400**

Also, we have the percentage of two models based on total samples.

percentageOfModel = ((100 * 900) / 1400)/100 = **0.64**

percentageOfNewModel= 1-percentageOfModel = **0.36**

Also, let we have the following set of random variables

setOfRandomVariablesOfModel={(***userLocation***,0.90),(***travel***,0.40)}

setOfRandomVariablesOfNewModel = {(***userLocation***,0.60), (***travel***,0.80)}

Calculate the new probability is based on percentage of each model :

updateProb = 0.90 * 0.64 = **0.57**

updateProb = newProb + (0.60*0.36) = **0.792**

This procedure is executed for each random variable of two sets, i.e. setOfRandomVariablesOfModel and setOfRandomVariablesOfNewModel.

```
procedure  incrementalLearning (in setOfRandomVariables; in numberOfSamplesOfModel in
setOfNewRandomVariables; in numberOfSamplesOfNewModel in evidenceSet;
in newEvidenceSet; out trainedModel; out finalEvidenceSet)
begin
  finalEvidenceSet:=[]
```

```
  finalRandomVariables:={}
  finalEvidenceSet = append (finalEvidenceSet, evidenceSet)
  finalEvidenceSet = append (finalEvidenceSet, newEvidenceSet)
  totalNumberOfSamples = numberOfSamplesOfModel + numberOfSamplesOfNewModel
  percentageOfModel = (100 * numberOfSamplesOfModel) / total
  percentageOfNewModel= 100 - percentageOfModel
  for each (randomVariable, probability) ∈ setOfRandomVariables
      newProb = percentageOfModel * probability
      tempProb= readSet(randomVariable, setOfNewRandomVariables)
      newProb = newProb + tempProb * percentageOfNewModel
      finalRandomVariables := append((randomVariable,newProb))

end
```

Algorithm 5 Incremental Learning In Our System

***Note**: the function readSet/2, takes as input a random variable and a set of random variables and it returns the probability of random variable that has been given as input. If the random variable that that has been given as input doesn't exist in set of random variables then it returns 0.

5.4.4 Derive Evidences in Trained Model

In this subsection we will present how to derive evidences for evaluation. The program "evaluationOfModel.py" in our system takes as input a new tweet-text message and it returns the opinion about visiting Crete or not based on evidences that are derived from the new tweet. The program "evaluationOfModel.py" uses some algorithms that we have constructed which create evidences (such as the training of the model). The algorithm **Algorithm 6** which uses the VADER tool takes as input a new tweet-text and returns a list with the values of random variables ***negativeSentiment***, ***positiveSentiment***. Let's assume that we have as input to this algorithm the following tweet-text.

'*I'm going to a fantastic place #visitCrete #travel*'

Its output is the following:

positiveSentiment = True

negativeSentiment = False.

The **Algorithm 6** is shown below:

```
procedure sentimentIdentificationWithVADER(in:newTweet; out: randomVariables)
begin
    randomVariables := [];
   if (sentimentAnalizer(newTweet) = 1) then
```

```
          randomVariables:=append([(positiveSentiment,True),randomVariables])
          randomVariables:=append([(negativeSentiment,False),randomVariables])


  if (sentimentAnalizer(newTweet)  = -1) then
          randomVariables:=append([(positiveSentiment,True),randomVariables])
          randomVariables:=append([(negativeSentiment,False),randomVariables])


end
```

Algorithm 6 Derive evidences from text (using Sentiment Analysis)

In addition, the program "evaluationOfModel.py" uses the algorithm **Algorithm 7** in order to extract evidences based on the Entity Recognition method. This algorithm, takes as input a new tweet-text and it returns the list with the values that the random variables-categories will have. Let have the following text of a tweet:

'I'm going to a fantastic place #visitCrete #travel'

The algorithm returns the list with the following values:

For the RV ***vacation*** it returns True because referred in the text the related word 'travel'

as input, ***health_care*** it returns False because not referred in the text some related word of health care domain

as input, ***location_in_Tweet*** it returns True because referred in the text the word 'Crete'

```
procedure evidenceOfTweet (in: newTweet, synonymsCSVtable; out: randomVariablesList)
begin
   for each rowSynonyms ∈ synonymsCSVtable do
       for fieldValue := 1 to size_of_rowSynonyms  do
          if (prefix(newTweet, rowSynonyms[fieldValue])  or
           infix (newTweet, rowSynonyms[fieldValue])  or
            postfix (newTweet, rowSynonyms[fieldValue])) then
             randomVariablesList:=append((fieldname,True), randomVariablesList);

end
```

Algorithm 7 Extract evidences from text (using Entity Recognition)

The algorithms **Algorithm 6 Algorithm 7** are implemented in the Python program "evaluationOfModel.py". After the successful execution of this program the ProbLog function get_evaluate/1 is called. This function, takes as input evidences by using the Bayes' rule it returns the probability of visiting Crete. For the above example, it will return the following answer.

yes, will visit Crete (visitLocation:0.7)

6 Sample Sessions of our System

In this chapter we will illustrate some sample sessions of our system. First, we present the graphical environment of our system. We use ‘Tkinter’ libraryof Pyhton. The Tkinter module is the standard Python interface to the TK GUI toolkit [docs.python, nd][Data-camp,nd] . The home window has 3 buttons, “train model”, “trained model” and “exit” Figure 12.

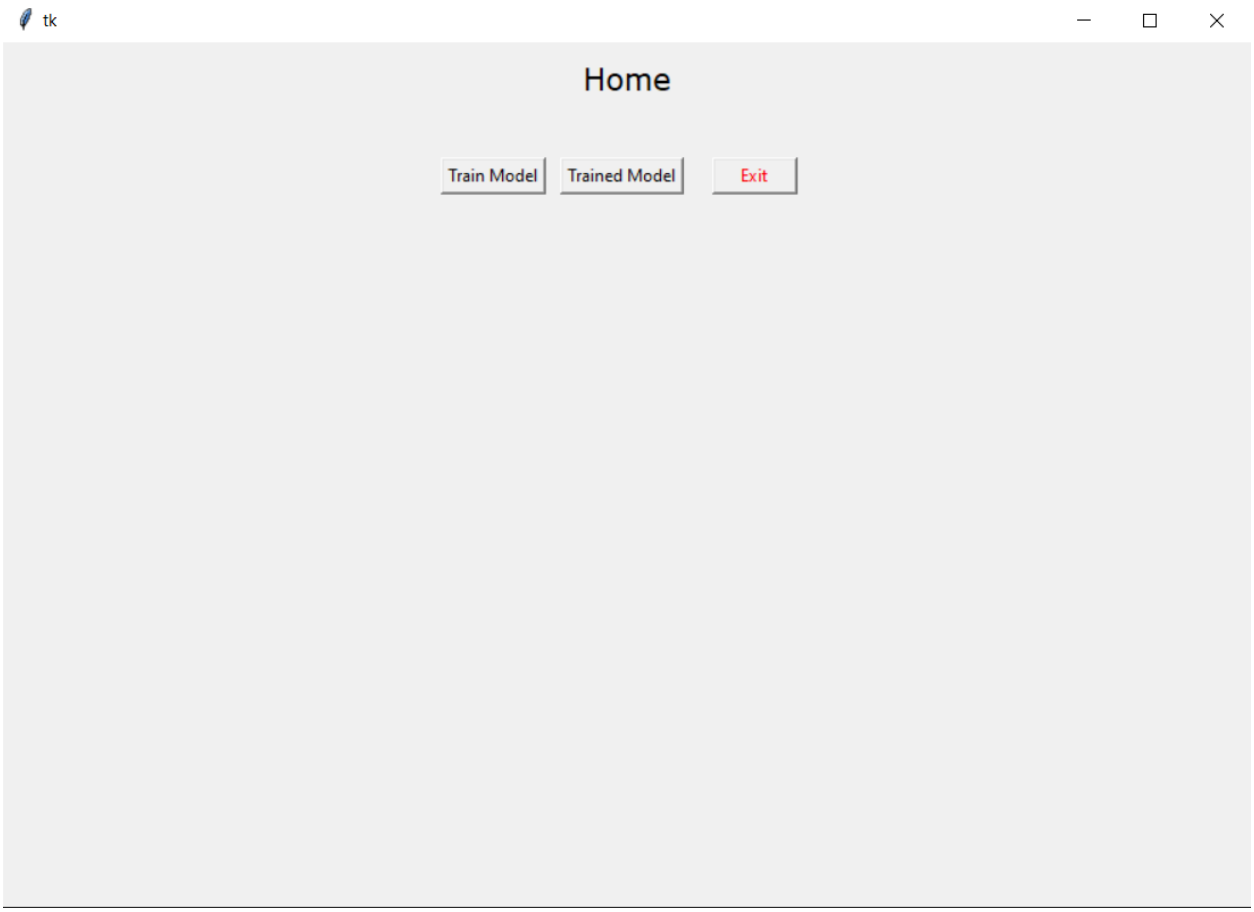

Figure 12 Home Window of our system

6.1.1 Train the Model

When user press the button “Train Model”, the train model window appears. Because, is this sample session there are not have any trained model, we inform the user with appropriate message(i.e. ‘There is no trained model’ green arrow in Figure 13).

Let us have the follow sample session of our system that is illustrated in Figure 13. We have a dropdown list, that user can choose the trainset for train the system. We choose the dataset “trainSet.csv”, and then we press the button “Incremental Learning”. The response is illustrated in Figure 14. Our system informs the user with an appropriate message when the model has been successfully trained (Figure 14 red arrow). Also, informs the user the number of samples that they were used to train the model (Figure 14 green arrow).

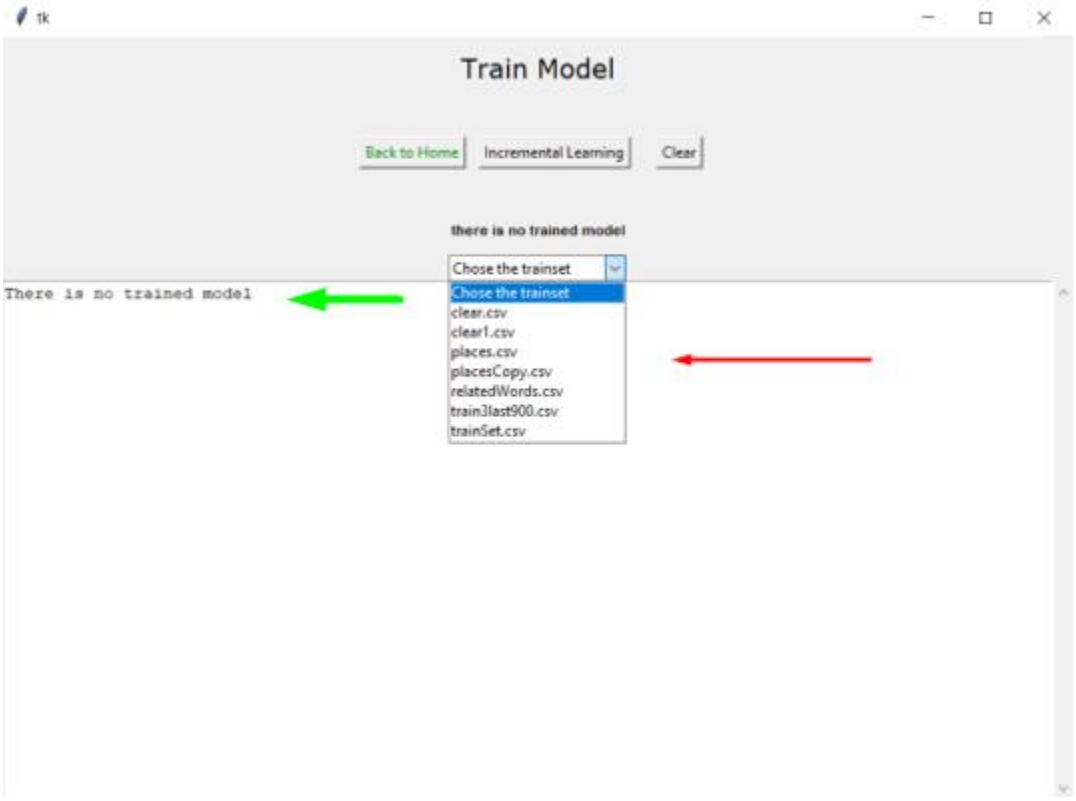

Figure 13 Train Model session

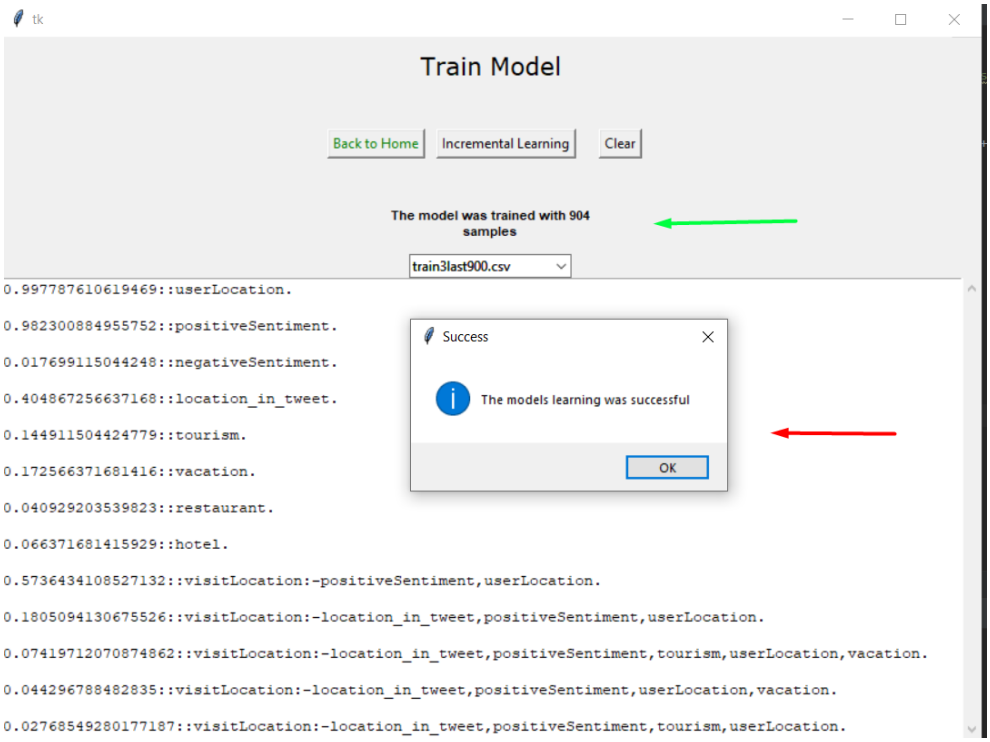

Figure 14 The result after press Incremental Learning Button

6.1.2 Incremental Learning sample session

In this subsection we will illustrate a sample session of incremental learning method. Our system uses incremental training when it already has a trained model. A user of this system can select an option from the option list, i.e. the option list has all datasets that they have been mentioned in Section 6.1.1. Let's assume that the file "test20.csv" is selected for further training the model, Figure 15. By selecting 'incremental learning' the response of our system is a pop-up window with an appropriate message such the one in Figure 15. The training process takes a few seconds to complete and to return its response. In this window there is a button back to the home window, i.e. "Back to Home".

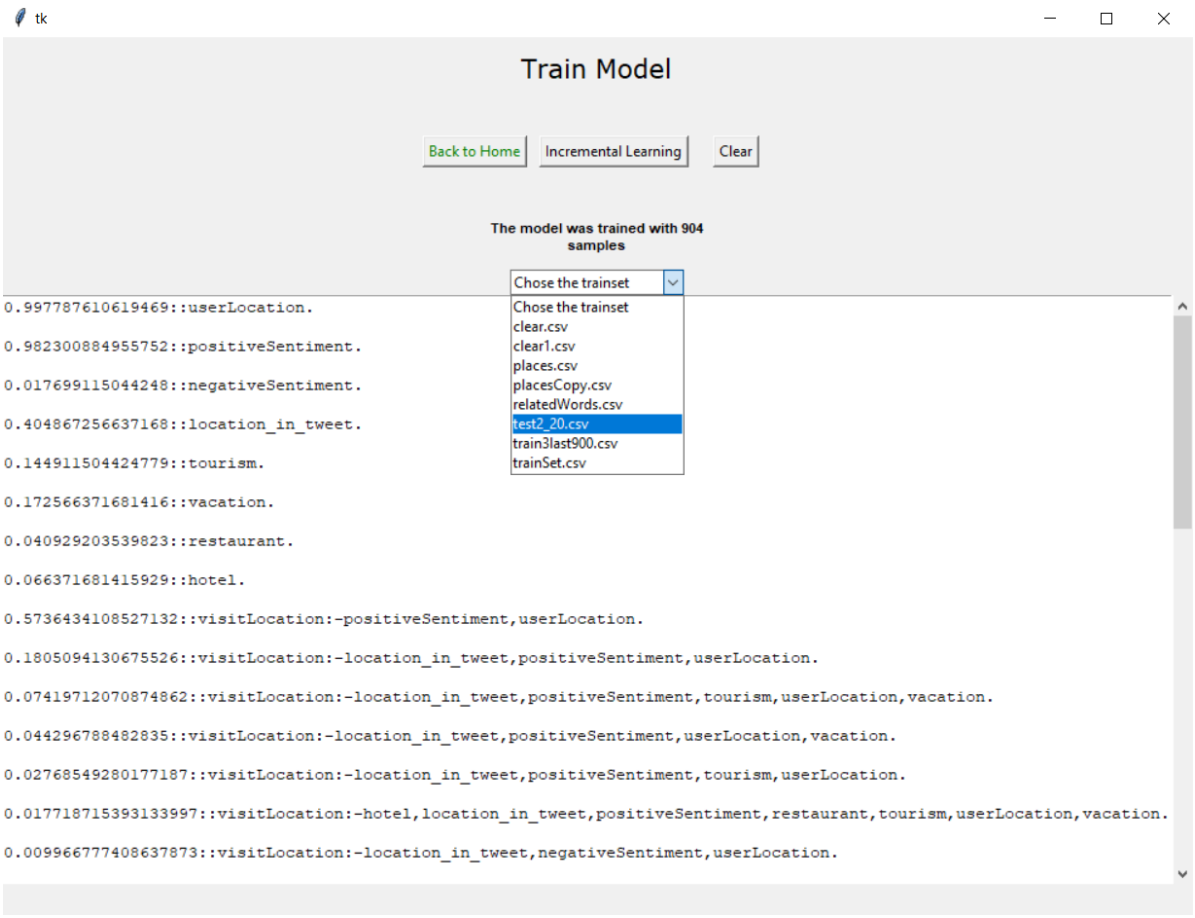

Figure 15 Incremental Learning sample Session

6.1.3 Sample Session of Trained Model

In home window, the user by selecting the "trained model" button, the trained model window appears. In this window, the user can write a text message, that is a tweet, and he is asked also to insert his/her home location. Let's consider the following example. The user writes the following text message.

"*I think Chania is perfect destination!! I want to go there!! #visitCrete#vacation#chania*"

The home location of the user is "*Texas*". This sample session is illustrated in figure Figure 16 Test the trained model sample 2. Then, the user presses the "Evaluate" button. The system derives its answer in the text field "Opinion for tweet". The derivation of the answer is based on the user's answer. When the user press the "clear" button, he/she can write a new text message for evaluation by the trained model. Another example will be shown in which a user does not mention anything about Crete. Let's assume that the user gives the following text message in the trained model.

"*FOX POLL:*

54% of Americans want #trump impeached. 50% want him impeached and removed from office.

I'm sure trump will be attacking FOX for its faulty polling. The thing is,

this is exactly what all major polls are finding."

The answer of the system is shown in figure Figure 16

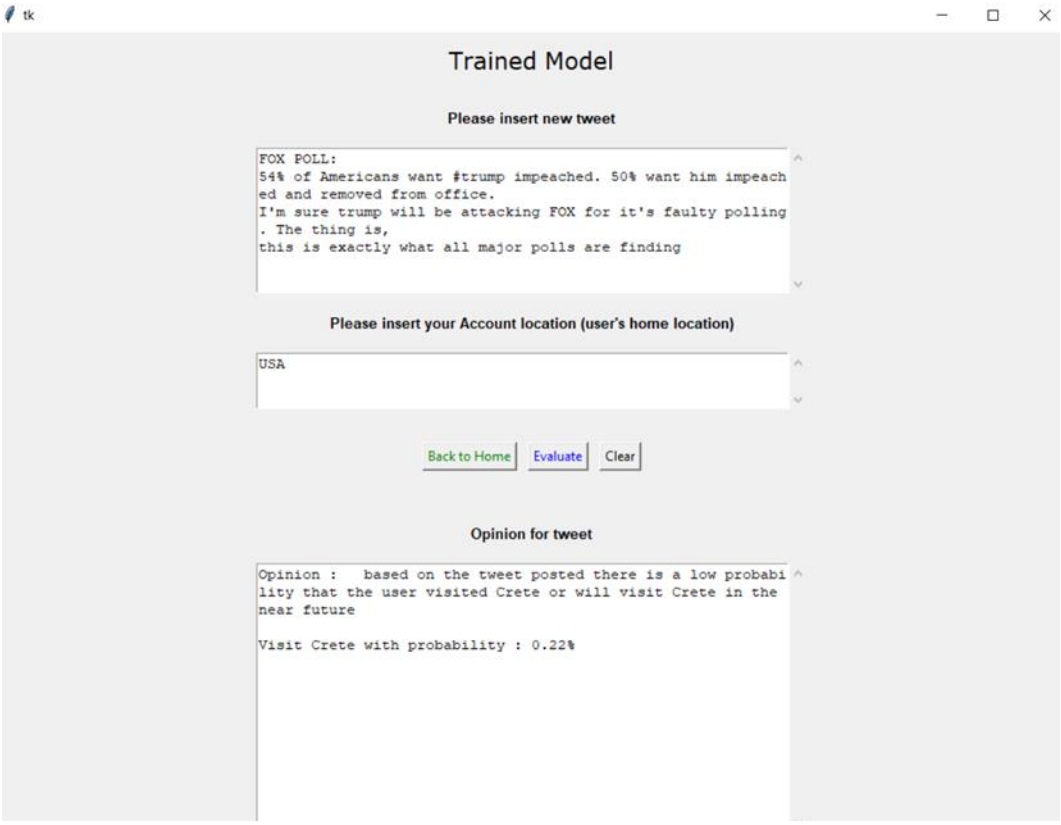

Figure 16 Test the trained model sample 2

7 Evaluation of our System

In this chapter, we will discuss the evaluation of our system. As it is known, Machine learning continues to be an increasingly integral component of our lives, whether we're applying the techniques to research or to business problems. Systems of this kind are able to give accurate predictions in order to create real value for a given organization. Many, researchers, mention that the key step in machine learning systems is the training of the model of the system. We think, it is also important the evaluation of system. More specifically, it is very crucial to know whether the trained model actually works effectively in real applications, that is it makes trustworthy predictions. There are two approaches for evaluating the performance of a model, ***holdout*** and ***Cross-validation***. Both of these approaches use *test data* to evaluate the system. The evaluation data have to be different from the data that were used to build the model. In the *holdout* approach the dataset is divided in the following subsets:

- The *training set* which is a subset of the dataset that is used to build predictive models.
- The *validation set* which is a subset of the dataset that is used to assess the performance of the model built in the training phase.
- The *test set*, or unseen data, which is a subset of the dataset that is used to assess the likely future performance of the model.

This approach is useful because of its speed, simplicity, and flexibility. On the other side, the most common method in *cross-validation* approach (called also, rotation estimation), is k-fold cross validation. In this popular method, the original dataset is partitioned into k equal size subsamples, called *folds*. The k is specified by the user (usually 5-10 preferred). This is repeated k times, such that each time, one of the k subsets is used as the test set/validation set and the other k-1 subsets are put together to form a training set. The error estimation is averaged over all k trials to get the total effectiveness of our model. In our system the dataset for evaluation is tweets that are not included in the training set. In the following subsection we discuss the evaluation of our system. More specifically, in the first subsection, we will discuss the evaluation criteria of our machine learning system with all metrics that are used. In the second subsection we will present the results of the evaluation metrics. These metrics are created by the Python program called "evaluationOfSystem.py".

The above sub-sections (7.1, 7.2) presented as follows. In sub-section we present the evaluation criteria of regression models. In sub-section 7.2 we present the evaluation results in our system based on metrics that we present in 7.1.

7.1 Evaluation criteria of regression models

In this sub-section we present the evaluation criteria of regression models. More specifically, we will first briefly describe the basic metrics of regression models. These metrics are the mean absolute error (MAE) , root mean squared error (RMSE), mean squared error(MSE). These three

metrics we choose to use in our machine learning system for evaluation where we present it in the next section. So, in this section we describe the aforementioned metrics.

Mean Absolute Error (MAE) is a measure of difference between two continuous variables [Willmot, et al. 2005] [Willmot, et al. 2006]. This measure is also called MAE and it is very popular metric for the accuracy of model. Furthermore, in general terms, MAE follows the next formula:

Prediction Error := Actual Value - Predicted Value

***Actual Value**:* is the value that is obtained by observation or by measuring the available data. It is also called the observed value.

***Predicted value**:* is the value of the variable predicted based on the regression model.

This prediction error is taking for each record in dataset after which we convert all error to positive. This is achieved by taking Absolute value for each error as below:

Absolute Error := |Prediction Error|

Finally, we calculate the mean for all recorded absolute errors (Average sum of all absolute errors).

$$mae = \frac{\sum_{i=1}^{n} abs(y_i - \lambda(x_i))}{n}$$

An example using this metric is discussed in order to illustrate its accuracy. So, let have the following actual values of houses based on bedrooms that contains each house.

Let have the following **actual costs** (assumed actual cost of houses in this example also the actual costs corresponds to the actual value in mae metric):

The house with two bedrooms costs — $200K

The house with three bedrooms costs 3 bedroom — $300K

The house with four bedrooms costs 4 bedroom — $400K

The house with five bedrooms costs 5 bedroom — $500K

So, let we have the following **predicted costs** by an regression model:

The house with two bedrooms costs — $230K

The house with three bedrooms costs — $290K

The house with four bedroom costs — $740K

The house with five bedrooms costs bedroom — $450K

We calculate the ERROR as follows:

The house of two bedroom house calculate the Error such as follows:

Actual Price = $200K

Predicted Price = $230K

Error := Actual Price — Predicted Price

Absolute Error 1 := |Error| (Absolute or positive value of our error) = |200-230| = **30**

So, this measurement is performed on each the aforementioned houses of demonstration example. According to the next formula the mae metrics is the following:

$$mae = \frac{\sum_{i=1}^{n} abs(y_i - \lambda(x_i))}{n} =$$

$$= \frac{30\$ + 10\$ + 340\$ + 50\$}{4} = \mathbf{107.5}\$$$

The measure of the evaluation of the model has been calculated by the above formula. We are therefore able to say that, averagely, our model-example predictions are off by approximately $107.5K.

Root Mean Squared Error (RMSE): this is a measurement to accuracy of a machine learning model. More specifically, it's the square root of the average of squared differences between prediction and actual observation. It is very similar to MAE measure, the most common similarity of the two measurements are the negatively-oriented scores, which means the closer to 0 the measurement is, the better the model accuracy [Willmot, et al. 2005]. The root mean squared error is calculated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(p_j - o_j)}$$

When p is the predicted value and o is the observed value. So, we give an illustration examples of the aforementioned metrics. Let have the same example , with the five houses. The actual costs (that corresponds the observed of the four houses are the following:

The house with two bedrooms costs — $200K

The house with three bedrooms costs 3 bedroom — $300K

The house with four bedrooms costs 4 bedroom — $400K

The house with five bedrooms costs 5 bedroom — $500K

And also, we have the following **predicted costs** by an regression model:

The house with two bedrooms costs — $230K

The house with three bedrooms costs — $290K

The house with four bedroom costs — $740K

The house with five bedrooms costs bedroom — $450K

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{4}(p_j - o_j)} =$$

$$\sqrt{\frac{1}{4}\ (230\$ - 200\$) * (290\$ - 300\$) * (740\$ - 400\$) * (450\$ - 500\$)} =$$

$$= \mathbf{1,275,153}$$

Mean Squared Error (MSE): called also Mean Squared Deviation (MSD). In statistics, MSE measures the average of the squares of the errors. It is also non-negative, closer to 0 the measurement is, the better the model accuracy. With above formula calculate the mean squared error in statistical model, when p is predicted value and o is observed value.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(p_j - o_j)$$

The only difference between RMSE and MSE metrics, is that RMSE formula use root to calculate the measure. So, in the same example with the two aforementioned metrics are the following:

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(p_j - o_j) =$$

$$\frac{1}{4}(230\$ - 200\$) * (290\$ - 300\$) * (740\$ - 400\$) * (450\$ - 500\$) =$$

$$\mathbf{5,100,000}$$

So, we gave the example of the aforementioned metrics (RMSE, MSE, MAE) because is the evaluation criteria of our system. We choose these metrics, because in our test set, we know the actual value of each tweet (e.g. the true value is the maximum probability, 1). The predicted value, as we mentioned earlier, is the value predicted of machine learning system. Also, the output of our model is a continue value (range 0.0-1.0), which is the main feature of regression models. Finally, in the next sub-section, we give the evaluation results of our machine learning system.

7.2 Evaluation results

In this subsection we will present the evaluation results of our system. We use the metrics that they were presented in subsection 7.1. More specifically, we create the program "evaluationOfOurSystem.py". In this program we use the *scikit-learn* library to evaluate our system. It is a Python library integrating classical machine learning algorithms in the tightly-knit world of scientific Python packages. Initially, we collect tweets which mention that their users have visited Crete. We have collected 100 tweets in "testTweets.csv" file. Then, the next step is to use scikit-learn library. More specifically, for Mean Squared Error measure we use the mean_squared_error/2 function. This function takes two arrays as input such as the following:

- **y_true**: This array contains the true value. The true value is the maximum probability(i.e. 1) that the tweet posted by the user who actually visit Crete.
- **y_predict**: This array contains the predicted value. The predicted value in our system has the probability that has each tweet in the test set "testTweets.csv".

Also, we use mean_absolute_error/2 to calculate the mean absolute error (MAE) of our model. Mean_absolute_error/2 also take two arrays as input (**y_true**, **y_predict**), and returns the MAE. For root mean squared error (RMSE) we also use mean_squared_error/2 function but with root instead.

Our evaluation program 'evaluationOfModel.py' returned the following metrics.

Mean Squared Error: **0.11**

Root Mean Squared Error: **0.34**

Mean Absolute Error: **0.33**

The visualization of the predicted values is illustrated in the next figure, Figure 17 Visualization of predicted values The y-axis has the probabilities with range 0-1 and the x-axis has the number of samples that are in the "testTweets.csv" file.

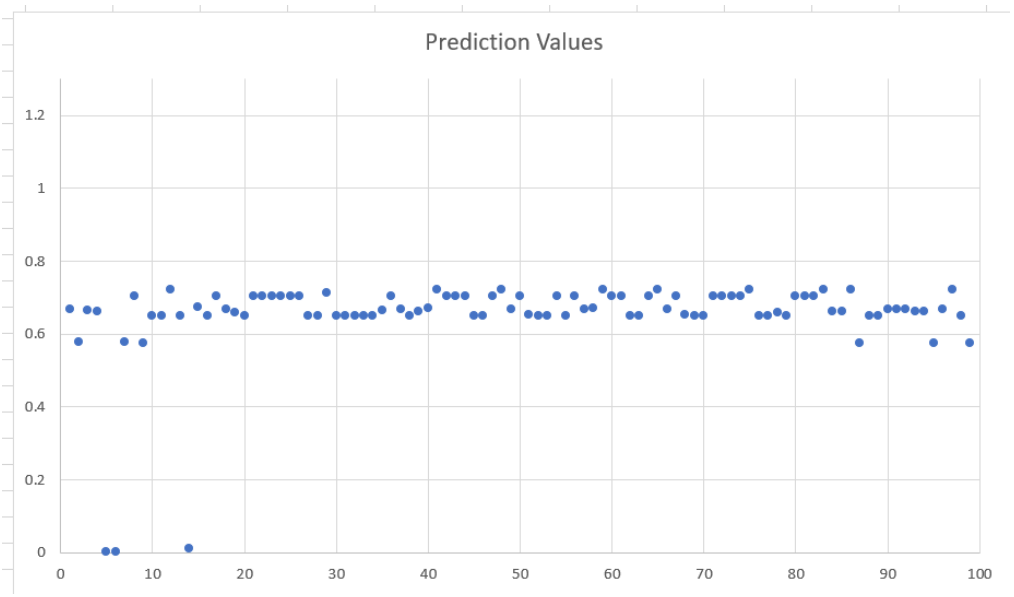

Figure 17 Visualization of predicted values

8 Conclusions and Future Work

Conclusions

In this research work, we represent a system for applying probabilistic logical reasoning for opinion mining in problems related to social media. As a use case we used Twitter data, describing a scenario where we would like to predict whether a user is intended to visit Crete with obvious applications for travel agencies and in all domains of the tourism industry. We use Sentiment Analysis and Entity Recognition methods in order to automate important tasks such as the following:

- Create random variables.
- Create rules.
- Create evidence set.

These aforementioned tasks are the basic features of a probabilistic graphical models like the Bayesian Network. After the completion of these automated tasks by our system it proceeds to the training of the model using the ProbLog toolbox. After that, new Tweets can be classified according to the desired outcome, i.e. whether their users will visit Crete with some probability. The evaluation of the system was based on metrics that has any regression model. More specifically, we use root mean square error, mean absolute error and mean squared error to measures the average of the errors that has our system. With these metrics, we conclude that our system, has derived a satisfactory model but not perfect.

We will summarize the benefits of our system which we have discussed in various sections of the thesis. An important feature of our system is its ability to be very easily adapted to many topics in social media in order perform opinion mining. In our approach, we used Twitter but our approach and system can also be used for any other social network such Facebook, Instagram etc. Furthermore, in the train procedure, our system Finally, our system supports incremental learning so the derived model can be improved.

Future work

In order to be able our system to answer queries on different topics we can create multiple Bayesian Models. This idea can be considered as a future work. More specifically, we may have a Bayesian Model that can answer question such “is this user intended to visit Paros?” for tourism perspective. Also, may have a Bayesian Network that can answer question such “Is this user suffering from depression?” for medical topic. Also, we can create multiple Bayesian Models to answer queries of the same topic such the topic of tourism i.e. same topic, but for different places. For example, we can have a Bayesian Model to answer query such “is this user intended to visit Paros?” and other Bayesian Model to answer question such “is this user intended to visit Crete?”. Also, as future work can be considered the structure learning procedure. As we mentioned in the first Chapters, the structure of our model is known and given. So, we can developed the structure learning and for that reason, we could give to our system, a training set of tweets and the system will derive the structure.

9 Bibliography

[Agarwal, et al, 2012]

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., & Passonneau, R.J. (2011). *Sentiment Analysis of Twitter Data,* LSM '11 Proceedings of the Workshop on Languages in Social Media,USA, pp 30-38

[Agarwal, et al, 1993]

Agrawal, R., Imielinski, T., & Swami, A.N. (1993). *Mining association rules between sets of items in large databases*. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, USA, pp 207-216.

7[Amor, et, al., 2004]

Amor, N.B., Benferhat, S., Elouedi, Z., *Naive Bayes vs decision trees in intrusion detection systems*, Proceedings of the 2004 ACM symposium on applied computing, Cyprus, pp. 420-424.

[Barber, 2007]

Barber, D. (2011). Bayesian Reasoning and Machine Learning. Cambridge University Press.

[docs.python, nd]

https://docs.python.org/3/library/tkinter.html

[Best-Hashtags,n.d]
http://best-hashtags.com/

[Cambria, et al. , 2013]

Cambria, E., Schuller, B.W., Xia, Y., & Havasi, C. (2013). *New Avenues in Opinion Mining and Sentiment Analysis*. IEEE Intelligent Systems, 28, 15-21.

5.[Cambria, et al. 2015]

5.Cambria, E., & Hussain, A. (2015). Sentic Computing: A Common-Sense-Based Framework for Concept-Level Sentiment Analysis. Springer- DOI 10.1007/978-3-319-23654-4

6.[Chen, et al., 2015]

Chen C., Zhang J., Chen X., Xiang, Y., Zhou, W., *6 million spam tweets: a large ground truth for timely twitter spam detection*, IEEE international conference on communications, London, pp. 7065-7070

[Cunningham, et al., 2002]

Cunningham, H., Maynard, D., Bontcheva, K., & Tablan, V. (2002). *GATE: an Architecture for Development of Robust HLT applications*, Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02, USA, pp168-175

[Data-camp, n.d.]

https://www.datacamp.com/community/tutorials/gui-tkinter-python

[data-gov, n.d]

http://www.data.gov.gr/dataset/poleis-xwria-ths-krhths.

[De Raedt et, al., 2007] 2

2.Raedt, L.D., Kimmig, A., & Toivonen, H. (2007). *ProbLog: A Probabilistic Prolog and Its Application in Link Discovery*, Proceedings of the 20th International Joint Conference on Artificial Intelligence. India, pp 2468-2473

[De Raedt, 2008]

Raedt, L.D. (2008). *Logical and Relational Learning*. Springer-Verlag 2010.

[De Raedt et, al. 2008]

Raedt, L.D. and Kersting, K., Probabilistic Inductive Logic Programming (2008). In *Probabilistic Inductive Logic Programming - Theory and Applications*, edited by Raedt, L.D., Frasconi, P., Kersting, K., & Muggleton, S., Springer.

[Esposito, et al. 2012]

Esposito, F., Ferilli, S., Basile, T.M., & Mauro, N.D. (2012). *Social networks and statistical relational learning: a survey*. International Journal of Social Network Mining Volume 1 Issue 2, pp 185-208.

[Farasat et, al. 2015]

Farasat, A, Nikolaev, A, Srihari, S, & Blair, R.H, (2015), Probabilistic graphical models in modern social network analysis, Social Network Analysis and Mining, **5,** 62 (2015). https://doi.org/10.1007/s13278-015-0289-6

[Gepperth, et al. 2016]

Gepperth, A., & Hammer, B. (2016). *Incremental learning algorithms and applications*. European Symposium on Artificial Neural Networks 2016 proceedings, Belgium, pp 357-368.

[Hutto, et at. 2014]

Hutto, C.J., & Gilbert, E. (2014). *VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text*. Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media. Ann Arbor, MI.

[Jie, et al., 2002]

Jie, C., Greiner, R., Kelly, J., Bell, D., Liu, W., (2002) *Learning Bayesian networks from data: An information-theory based approach,* Artificial Intelligence 137, pp 43-90

[Li, et al., 2016]

Li, J., Ritter, A., & Jurafsky, D. (2014*). Inferring User Preferences by Probabilistic Logical Reasoning over Social Networks,* In 19th International Conference, DS 2016, Bari, Italy, edited by Calders, T., Ceci, M., Malerba, Springer.

[Khyati, et al. 2014]

Khyati, D., Surbhi, C., Ashika, S., (2014) *Opinion Mining from Social Networks*, International Journal of Computer Science and Network Vol (3) Issue(6), pp 554-558

[Kontkanen, et al. 1997]

Kontkanen, P., Myllymäki, P., Silander, T., Tirri, H., (1997), *Comparing predictive inference methods for discrete domains*, Sixth International Workshop on Artificial Intelligence and Statistics pp 311-318

[Lourentzou, et, al., 2017]

I. Lourentzou,I., Morales, A., Zhai, C., *Text-based geolocation prediction of social media users with neural networks*, 2017 IEEE International Conference on Big Data (Big Data), Boston, USA, pp. 696-705.

[Medium, nd]

https://medium.com/@abhinav.mahapatra10/probability-vs-likelihood-bab5b2b42150

[Myllymäki, et al. 2002]

Myllymäki, P., Silander, T., Tirri, H., & Uronen, P. (2002). *B-Course: A Web-Based Tool for Bayesian and Causal Data Analysis*. International Journal on Artificial Intelligence Tools, 11, 369-387.

[Nilsson, 1986]

Nilsson, N.J. (1986). *Probabilistic logic" artificial intelligence*, Proceedings Second National Conference on Artificial Intelligence, Pittsburgh, Volume 28, Issue 1, pp 71-87

[Power-thesaurus, n.d.]

https://www.powerthesaurus.org/

[Rao, et at., 2016]

Rao, P., Katib,A., Kamhoua,C., Kwiat K., Njilla,L., *Probabilistic Inference on Twitter Data to Discover Suspicious Users and Malicious Content*, (2016), IEEE International Conference on Computer and Information Technology (CIT), Nadi, pp. 407-414.

[Ritter, et al., 2011]

Ritter, A., Clark, S., Mausam, & Etzioni, O. (2011). *Named Entity Recognition in Tweets: An Experimental Study*, Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, Scotland, pp 1524-1534

[Satya , et al., 2016]

Satya P.R.B., Lee K., Lee D, (2016). *Uncovering fake likers in online social networks*. CIKM' 16 Proceedings of the 25th ACM international on conference on information and knowledge management, Indianapolis, pp 2365–2370.

[Skaza, et al., 2017]

Skaza, J., & Blais, B., (2017). *Modeling the Infectiousness of Twitter Hashtags*, Physica A: Statistical Mechanics and its Applications, Volume 465, pp 289-296

4[Xu, et al., 2019]

4.Xu,C., Yuyu, Y., Mehmet, O., (2019), *Using Bayesian networks with hidden variables for identifying trustworthy users in social networks*, Journal Inference Science, pp 1-16

[Xusheng, et al., 2018]

Xusheng, L.,, Chengcheng, F., Ran, Z., Duo Z., Tingting, H., Xingpeng, J., (2018) *A hybrid deep learning framework for bacterial named entity recognition with domain features*, IEEE International Conference on Bioinformatics and Biomedicine 2018, Spain, pp 1-9.

[Tweepy, nd]

https://www.tweepy.org/

[1 Vennekens, et al., 2004]

1.Vennekens, J., Verbaeten, S., Bruynooghe, B., (2004). *Logic programs with annotated disjunctions*. International Conference on Logic Programming, Springer, Berlin, Heidelberg, pp 431-445.

[Shindle, et al, 2018]

Shinde, P. P., Shah, S., (2018), *A Review of Machine Learning and Deep Learning Applications*, Fourth International Conference on Computing Communication Control and Automation, India, pp 1-6.

[Willmott, et al. 2005]

Willmott, C.J., & Matsuura, K. (2005). *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performan*ce, Climate Research, Vol 30,pp 79-82.

[Willmot, et al. 2006]

Willmott, C.J., & Matsuura, K. (2006). *On the use of dimensioned measures of error to evaluate the performance of spatial interpolators*. International Journal of Geographical Information Science, 20, 89-102.

[Uusitalo, 2007]

Uusitalo, L. (2007). *Advantages and challenges of Bayesian networks in environmental modelling*, Ecological Modelling, Vol 223, pp 312-318.

Appendix A

A.1. The implementation of sentiment analysis procedure with VADER

The above function corresponds to the algorithm **Algorithm 1** that illustrated in chapter 5.

```
facts = """t (_)::userLocation.\n"""
neg = False
pos = False
randomVariables = {}
for i in range (1, len(tweetsList)):
      if sentiment_analyzer(tweetsList[i].text, analyzer) == -1 and neg == False:
         facts = facts + "t (_)::negativeSentiment.\n"
         neg = True
      if sentiment_analyzer(tweetsList[i].text, analyzer) == 1 and pos == False:
         facts = facts + "t(_)::positiveSentiment.\n"
         pos = True
if pos == True and neg==True:
    break
```

A.2. Implementation of Entity Recognition method

The above function corresponds to the algorithm **Algorithm 2**

```
def readRelatedWordsDict(sentence,tempdictionary):
  tempList = returnFirstColByRelatedWords()
  with open ('datasets/relatedWords.csv', 'r', encoding='utf-8-sig') as csvFile:
     reader = csv.reader(csvFile)
     for row in reader:
        for i in range(0, len(tempList)):
           if row[i] in sentence and row[i] not in "" and checkKey(tempdictionary,tempList[i]) == 1:
```

```
            tempdictionary[tempList[i]] = True
            break
    return tempdictionary
    csvFile.close()
```

A.3 Check if user's home location is from Crete or not

```
def checkPlace(text,randomVariables):
    str = []
    with open('datasets/placesCopy.csv', 'r', encoding='utf-8-sig') as csvFile:
        reader = csv.reader(csvFile)
        str = text.split(",")
        for row in reader:
            for i in range(len(str)):
                if row[0] == str[i].capitalize():
                    randomVariables['userLocation']=False
                    return randomVariables
                    csvFile.close()
    randomVariables['userLocation'] = True
    return randomVariables
    csvFile.close()
```

A.4 Create Evidence set based on array of tweets

The above part of program "createModel.py" use the aforementioned function to create the evidence set based on array of tweets.

```
  for i in range(1, len(tweetsList)):
    tempDict = initialDict.copy()
    tempDict = sentiment_analyzer_scores(tweetsList[i].text, analyzer,
                                tempDict)
    tempDict = readRelatedWordsDict(tweetsList[i].text,
                               tempDict)
    tempDict = checkPlace(tweetsList[i].location, tempDict)

    orderedDictionary = collections.OrderedDict(sorted(tempDict.items()))
    orderedDictionary = {Term(k): v for k, v in orderedDictionary.items()}
examples.append([(key, value) for key, value in orderedDictionary.items()])
```

A.5 The implementation of incremental learning procedure

The above code corresponds to the algorithm **Algorithm 5** that we mentioned earlier in chapter 5.

```
def incrementalLearning (currentRandomVars, sizeOfCurrentSet, currentEvidence):
    oldRandomVariables = {}
    newRandomVariables = {}
    evidenceOld = {}
    newEvidence = []
    txtEvidence = {}
    content = []
    if os. path. exists('models/model.txt'):
        with open('models/model.txt') as f:
            content = f.readlines()
            content = [x.strip() for x in content]
    if os.path.exists('models/examples.npy'):
        numpyArray = np.load("models/examples.npy")
        oldEvidence = []
        oldEvidence = convertToSet(numpyArray)
        newEvidence = oldEvidence + currentEvidence
 else:
        newEvidence = currentEvidence
 if len(content) > 0:
        total = int(content[0]) + sizeOfCurrentSet
        percentageOfCurrentDataset = (100 * sizeOfCurrentSet) / total
        percentageOfOldDataset = 100 - percentageOfCurrentDataset
        for i in range(1, len(content)):
            if ":-" in content[i]:
```

```
            break
        variable = content[i].split("::")
        tempVar = variable[1].split(".")
        oldRandomVariables[tempVar[0]] = variable[0]
else:
    percentageOfCurrentDataset = 100

facts = ""
if os.path.exists('models/examples.npy'):
    for key, value in currentRandomVars.items():
        if checkKey(oldRandomVariables, key) == 1:
    sumWeight = float((percentageOfOldDataset / 100))*float(oldRandomVariables[key])
    sumWeight = sumWeight + float((percentageOfCurrentDataset / 100)) * float(value)
        newRandomVariables[key] = sumWeight
        else:
            sumWeight1 = float(value)*100/total
            newRandomVariables[key] = sumWeight1
    for key,value in oldRandomVariables.items():
        if checkKey(currentRandomVars,key)==0:
            sumWeight2= float(value)*100/total
            newRandomVariables[key]=sumWeight2
    for key, value in newRandomVariables.items():
        facts = facts + str(value) + "::" + key + ".\n"
else:
    for key, value in currentRandomVars.items():
        facts = facts + str(value) + "::" + key + ".\n"
np.save("models/examples.npy", np.array(newEvidence))
```

```
return newEvidence, facts, len(newEvidence)
```

A.6 Test the trained model with new set of tweets

```
for i in range(1, len(tweetsList)):
    evidence = []
    evidenceDict = {}
    evidenceDict = randomVariables.copy()
    tweet1 = NewTweet(tweetsList[i].text, tweetsList[i].create, tweetsList[i].location)
    evidenceDict = sentiment_analyzer_scores(tweet1.text, analyzer,
                                    evidenceDict)

    evidenceDict = readRelatedWordsDict(tweet1.text,
                                evidenceDict)
    evidenceDict = checkPlace(tweet1.location,
                          evidenceDict)

    evidenceDict = {Term(k): v for k, v in evidenceDict.items()}
    evidence = [(key, value) for key, value in evidenceDict.items()]
    lf = engine.ground_all(db, evidence=evidence, queries=[query])
    result = get_evaluatable().create_from(lf).evaluate()
```