Weather Talk,
extracting weather information by text mining

by

Vasileios Lampos

Supervisor: Professor Nello Cristianini

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

September 2008

"... Tired of lying in the sunshine, staying home to watch the rain.
You are young and life is long and there is time to kill today.
And then one day you find ten years have got behind you.
No one told you when to run, you missed the starting gun..."

Pink Floyd, Time.

Abstract

The main aim of this project was to design and implement a system able to infer the weather state of a location for a specific date by applying Bayesian inference models and statistical analysis on web observations. Additionally, we investigated various linear combinations of probabilistic schemes where traffic information, previous day's weather or a weather prior probability contribute to the final decision. As a final extension, we visualised the weather inference results on a map.

Software packages and a weather ontology were developed for data collection and preprocessing. Parameterised Bayesian belief networks formed the expression of probabilistic correlation between the inferred and the official weather observations. During training, we decide the optimal parameters and then test their absolute and relative performance. Experimental results indicate that the absolute and relative (p-values) performance in most of the schemes is significant. As a result, one may assume that similar or even more sophisticated information extraction models on different contexts will be able to deliver useful conclusions.

Despite the amount of work carried out, and the significant results we retrieved, there is still space for improvements. Recommendations for further work are given lastly.

Acknowledgements

I would like to express my deep appreciation to Professor Nello Cristianini for his support and interesting discussions throughout all the stages of the project.

I would like to thank Ilias Flaounas for allowing me to use his HTML scraper, Omar Ali for his general advice on large scale data mining projects, and Elena Hensinger for proof-reading this dissertation.

I am grateful to Dr Steve Gregory for his continuous support in several occasions.

Finally, I would like to thank all the people around me, physically or virtually, for making me feel like a person, despite my tension not to be one.

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Vasileios Lampos
September 2008

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 2005, a scientific publication indicated that the web had at least 11.5 billion indexed web pages [GS05]. This was used as a landmark in many discussions, until the recent publication of an official web article from Google's employees which stated that they have discovered 1 trillion unique **U**niversal **R**esource **L**ocators (URLs).[1] One can argue about the quality and quantity of the real content behind these URLs, but still cannot refuse that on the web lies an enormous amount of information.

This is the basis of "Weather Talk" project, the vast amount of data available on the web. Throughout this text we will refer to them as unofficial or media observations. The most important initial assumption made was that these unofficial observations will have a significant correlation with the true weather which is expressed by official weather observations. As a result, during this project we try to find the pattern between these two entities by using a weather ontology and statistical analysis on several Bayesian schemes.

However, learning previous day's weather does not possess a great interest, since it does not produce any new knowledge. It is important to notice that weather choice was motivated by the fact that our conclusions could be "easily" evaluated by the official weather observations (availability of ground truth).

In the beginning of this project, when data were not available, we have made some initial assumptions in order to support our data collection methods. Firstly, we assumed that

[1] Alpert, J., Hajaj, N. "We knew the web was big...". The Official Google Blog, 25/07/2008, http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

articles which mention a location name, may include weather information as well with a relatively significant percentage. As average statistics indicated later, this happened with a percentage of 2.56%. Obviously, articles that did not refer to a location (or locations) were not considered as their weather information (if any) would not have been of any use.[2] Overall, the elements of the weather ontology, implemented for the needs of this project, showed up with a probability of 0.087% in the collected documents' words (or phrases), indicating that more than 1,000 words were needed in order to retrieve a weather related one.

In the following sections, we state the aims and objectives of this project and we give a general description of the dissertation's outline.

1.1 Aims and objectives

The **aim** of this project is to apply statistical analysis on web observations and to design Bayesian inference schemes in order to extract information about the **weather state** of a location during a specific date. Documents may include blogs, newsgroups, and news articles but not officially weather related sources. A further challenge is the implementation of a **data fusion** model able to combine traffic information with weather observations and achieve a better performance.

The objectives for meeting the proposed aims are the following:

- **Data collection** and **preprocessing**. Web documents, official weather feeds which will be used during evaluation, and traffic statistics should be collected from the web. A scheme that extracts weather information from the documents using a weather ontology as well as a model that maps traffic data to weather states will be implemented.
- Design and implement **Bayesian weather inference schemes**. Parameterised Bayesian Belief Networks will be used for this purpose. In its final version the system should be able to combine four distinct data inputs derived from media observations, previous day's weather, traffic information, and average weather statistics.

[2] In that case, approximations about the location could be extracted from the website's context but this may be a matter of additional research.

- **Train** the schemes in order to decide the optimal values for the parameters. A prerequisite for this objective is the definition of loss measures able to quantify the performance of each scheme.
- **Test** the performance of each scheme. The absolute and relative (p-values) performance of each scheme should be tested.
- Build a **dynamic weather map** for visualising the conclusions.

The outcomes of this project will form a good base for further academic research on this field.

1.2 Dissertation outline

The dissertation consists of five parts.

Chapter 2 refers to the background and the general context of the project. Most importantly, it provides the reader with an overview on concepts such as ontologies, data fusion, and Bayesian belief networks.

Chapter 3 describes all the procedures of the project in detail. After defining the weather representation in this project, it focuses on the data collection and preprocessing methods, the Bayesian schemes and the building of the weather map.

Chapter 4 presents the experimental results of the project. Firstly, it defines the performance quantification measures and then it shows how the optimal values were chosen for each scheme. In the end, it provides the absolute and relative performance testing results.

A critical evaluation of the project and our final conclusions are provided in chapter 5. Trying to be as objective as possible, we discuss to which extent we have achieved our objectives and whether our choices were always optimal. In the end, we summarise all the outcomes of the project.

Recommendations for further work are drawn up in chapter 6.

Chapter 2

Background and general context

2.1 Introduction

The purpose of this chapter is to familiarise the reader with the theoretical background and the general context of this project. The notions of web mining, crawler, data fusion, and ontology are explained and their attachment to this work is defined. **T**erm **F**requency **I**nverse **D**ocument **F**requency (TF-IDF) scheme is described and an extended reference to **B**ayesian **B**elief **N**etworks is provided.

2.2 Web mining and crawlers

Data mining, also known as "Knowledge Discovery in Databases", is the process of extracting unknown, non obvious or hidden, but conditionally useful information from data stored in a database format using nontrivial techniques [FPSM92]. Web mining is based on the same concept and, in most cases, uses the same algorithms for retrieving information, but it is applied on web data. These data may have been derived from a server's database but without having full access to that server. Web mining applications should be able to access information located on the web either it is visible or just unprotected, meaning that the data collection process is a substantial part of the whole mining procedure. The applications responsible for data collection are called crawlers and their main strategy is to start from a set of URLs, visit the included hyperlinks (or links), enlarge this set, and then recursively

iterate through the same procedure [Cha03]. During this project, an alternative approach on crawling was used (see section 3.6).

However, data collection is only the base of a web mining system. The parts of general interest are included in higher layers of the system's structure, where the retrieved raw information is transformed to useful conclusions. There are several types of web mining systems; web structure and web usage mining discover patterns in the links or in the user's behaviour (*e.g.* clicks made, web sites visited, timing) respectively, whereas web content mining investigates a bigger challenge, to actually extract knowledge from the data [Liu07a]. The procedure of extracting conclusions from a set of data is the main focus of the project and an ontology takes a significant part in it.

2.3 About ontologies

In [Gru93] ontology is defined as a formal but restricted specification of a concept which describes predicted relationships between its elements. Applications of ontologies vary; they can be used for analysing the knowledge of a domain, defining strictly a broad concept or for separating types of knowledge, but most importantly ontologies are used for sharing noematic domain representations among human or software agents [NM01].

Most of the "ontology driven" information systems embed the ontology in a database [Gua98]. In this project we will use a weather ontology in the process of transforming unordered and heterogeneous data to usable information for a model embedded in a software agent. The ontology will be stored in the database and contain several mappings between classes, terms, and attributes. It will form a part of the "common access to information" scenario as it is presented in [JU99]. The weather ontology is presented in section 3.4.

2.4 TF-IDF scheme

Ontology is a static representation of a context, and as a result it cannot adapt to particular circumstances. For example, when the phrase "it was raining" appears inside a document ten times, the ontology will just assist the system in distinguishing this phrase from the

text ten times. However, it might have been the writer's intention to mention 'rain' a lot because of its catastrophic effects or it may have been an invalid document just repeating the same phrase. Therefore, the system must have a built in module which will enable it to determine (an approximation of) the actual importance of a word or phrase in a document or several documents.

TF-IDF weighting scheme was proposed by Salton and Yang in [SY73] and offers a standarised solution. It is an Information Retrieval (IR) vector space model for document representation. TF-IDF treats each document as a "bag of words", an approach followed by many models in IR,[1] but instead of simply counting the terms in it and assigning absolute frequencies to them, it calculates a normalised frequency over a set of documents in order to define weights of importance.

[SFW83] and [Joa96] have been used as additional references to the following description. Suppose we have a vocabulary V with n terms and a set D with m documents,

$$V = \{v_1, ..., v_n\},$$
$$D = \{d_1, ..., d_m\}.$$

For $\forall v_i \in V$, we calculate the number of its appearances in $\forall d_j \in D$. As a result, we derive a set F of size $n \times m$ which contains the raw frequencies of the vocabulary terms in the documents,

$$F = \{f_{11}, ..., f_{n1}, f_{12}, ..., f_{ij}, ..., f_{nm}\}.$$

The frequency tf_{ij} of a term v_i in a document d_j is given by

$$tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, ..., f_{nj}\}}. \tag{2.1}$$

It is a normalized frequency which depends on the maximum raw frequency of a term in a specific document. For a term v_i and a document d_j, if the raw term frequency $f_{ij} = 0$ (*i.e.* the term does not appear in the document), then the normalized frequency $tf_{ij} = 0$. At this stage, we have retrieved the set TF which includes all the normalized term frequencies

[1] In IR, the notion "bag of words" is used to denote that a model focuses on each one of the words separately, disregarding any semantic connection between them or even the order of their appearance.

for the documents,

$$TF = \{tf_{11}, ..., tf_{n1}, tf_{12}, ..., tf_{ij}, ..., tf_{nm}\}.$$

For $\forall v_i \in V$ we calculate the total number of documents in which it appears at least once. This is denoted by the document frequency df_i. All the document frequencies are included in the set DF,

$$DF = \{df_i, ..., df_n\}.$$

In order to retrieve the inverse document frequencies set IDF, we use the following formula

$$idf_i = log\frac{m}{df_i}, \tag{2.2}$$

where m is the number of documents. Similarly, the IDF set has n elements,

$$IDF = \{idf_i, ..., idf_n\}.$$

In the final stage of the scheme, we build a set W with weights for every term-document pair,

$$W = \{w_{11}, ..., w_{n1}, w_{12}, ..., w_{ij}, ..., w_{nm}\}.$$

Each w_{ij} is the result of

$$w_{ij} = tf_{ij} \times idf_i. \tag{2.3}$$

TF-IDF scheme is applied in the data preprocessing stage of the project (see section 3.7), and especially during the extraction of weather information (in the form of weather related words or phrases) from the collected web documents.

2.5 Bayesian Inference

Bayes' theorem provides a way to calculate the *a posteriori* probability $\Pr(X|Y)$,[2] from the *a priori* probability $\Pr(X)$, using $\Pr(Y|X)$ and $\Pr(Y)$.

$$\Pr(X|Y) = \frac{\Pr(Y|X)\Pr(X)}{\Pr(Y)}. \tag{2.4}$$

Using (2.4) we are able to calculate the exact probability of an event X, given a set of data Y. In case we want to decide the most probable event based on the same data set Y, then the normalizing denominator $\Pr(Y)$ is not needed as it is the same for all the events X. Our decision is the *maximum a posteriori* hypothesis s and is defined as

$$s = \underset{x_i \in X}{\operatorname{argmax}} \Pr(x_i|Y) = \underset{x_i \in X}{\operatorname{argmax}} \Pr(Y|x_i)\Pr(x_i). \tag{2.5}$$

The introductory definitions above were based on [HK01].

2.5.1 Bayesian Belief Networks

This section has been based on the relevant chapters in [PNM08]. In general, a belief network is a *directed acyclic graph* (DAG)[3] in which the nodes represent random variables and the arcs define ways of connection between the nodes [Pea86]. A BBN applies the Bayesian probabilistic theory as a way of building relationships between the nodes of the network. A formal definition of a BBN follows (Figure 2.1).

Definition of BBN. Suppose there exists a set, X, of n random variables, $\{x_1, x_2, ..., x_n\}$, and a DAG, $G(V, E)$, with $|V| = n$ vertices and at most $|E| = \frac{n(n-1)}{2}$ edges. Each node x_i has a set of parent nodes, P_i,

$$P_i = \{parents(x_i)\}, \text{ with } 1 \leq i \leq n, \text{ and } |P| \in [0, n-1].$$

[2] $\Pr(X|Y)$ denotes the probability of the event X based on the event Y.
[3] Basic information about graphs is provided in Appendix A.1.

The graph is a Bayesian belief network which represents the variables $x_1, ..., x_n$ the if

$$\Pr(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} \Pr(x_i|P_i), \tag{2.6}$$

which means that the joint probability of all the variables in X is equal to the product of all their possible joint probability distributions.

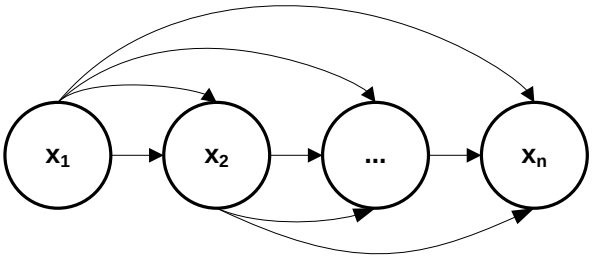

FIGURE 2.1: A complete Bayesian belief network with n nodes.

Equation 2.6 can be expressed in a recursive way as:

$$\begin{aligned} \Pr(x_1, x_2, ..., x_n) &= \Pr(x_1) \cdot \Pr(x_2, ..., x_n|x_1) \\ &= \Pr(x_1) \cdot \Pr(x_2|x_1) \cdot \Pr(x_3, ..., x_n|x_1, x_2) \\ &= \Pr(x_1) \cdot \Pr(x_2|x_1) \cdot Pr(x_3|x_1, x_2) \cdot ... \cdot Pr(x_n|x_1, ..., x_{n-1}). \end{aligned} \tag{2.7}$$

Therefore, this equation expands to every possible joint probability distribution, indicating that it can be represented by a Bayesian belief network. In this project, we will use a linear combination of Bayesian belief networks in order to infer a weather state of a location during a specific day. An example on the Bayesian networks is presented in the next subsection.

2.5.2 An example

In Figure 2.2 we present a simplistic Bayesian network that investigates the problem of a computer crash (**CC**). T, and F stand for true, and false respectively and numbers on the tables express the probability of an event. During the implementation of the Bayesian inference schemes in "Weather talk", we create parameterised versions of these tables since we cannot assume their contents and then we train them in order to retrieve their optimal representations (see section 3.8).

The model (Figure 2.2) assumes that a computer crash can be caused by a "bad" user (**BU**) or "bad" software (**BS**). It also makes the assumption that a bad user is more likely to use bad software. When both **BU** and **BS** events are true, then it is more likely for **CC** to become true.

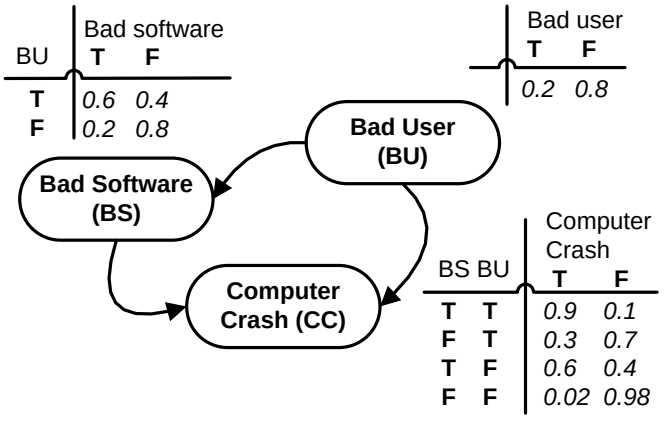

FIGURE 2.2: A Bayesian belief network that investigates the probabilistic relationships between a computer crash, a bad user, and bad software. T and F denote true and false respectively.

From the theory on the Bayesian networks in the previous section, we know that the joint probability of the three events is given by

$$\Pr(\text{CC, BS, BU}) = \Pr(\text{CC}|\text{BU,BS}) \cdot \Pr(\text{BS}|\text{BU}) \cdot \Pr(\text{BU}),$$

and as result when all the events are true their joint probability is equal to

$$\Pr_{ttt} = \Pr(\text{CC} = \text{T, BS} = \text{T, BU} = \text{T}) = 0.9 \cdot 0.6 \cdot 0.2 = 10.8\%,$$

meaning that we might have a computer crash from a bad user using bad software with probability equal to 10.8%.

The Bayesian scheme can be used for retrieving the probability of combined events, such as the probability of a good (not bad) user when we have a computer crash. This is equal to

$$\begin{aligned} \Pr(\mathrm{CC} = \mathrm{T} | \mathrm{BU} = \mathrm{F}) &= \frac{\Pr(\mathrm{CC} = \mathrm{T},\ \mathrm{BU} = \mathrm{F})}{\Pr(\mathrm{BU} = \mathrm{F})} \\ &= \frac{\Pr_{ttf} + \Pr_{tff}}{\Pr_{ttf} + \Pr_{ftf} + \Pr_{tff} + \Pr_{fff}} = 45.2\%. \end{aligned}$$

2.6 Data Fusion

Data fusion can be defined as the combination of heterogeneous kinds of information during the process of decision making in order to enhance the overall accuracy of the decision. Lee in [Lee97] proves that combinations of information will improve the final conclusion. Especially, when integrating ("fusing") non relevant contexts, the improvement is even greater [VC99]. Before we begin the fusing process, we have to choose a general data concept to fuse. A good choice is the use of observable web data about traffic, as traffic's behaviour is connected to the weather conditions.

There are two alternatives in applying data fusion on "Weather talk" project. These alternatives refer to the level of the model on which data fusion is applied. Following the definitions in [HL97], "raw data" fusion, combines data from all the resources at the time of retrieval. If we had chosen this approach, then we would have to collect data related with weather and traffic and then, using a more complex multi-ontological scheme, to map them to weather states. A better approach is to use "decision level" fusion, meaning that the fusing procedure is carried out on the already made decisions of two separate models. The implementation of "decision level" fusion will request to convert traffic information to weather conditions (see section 3.7.1). At the end, using a weight mechanism we will merge the two different based decisions into the final one. The values of each weight will be determined through experiments.

2.7 Summary

In this chapter we referred to the general background of the project and we analysed the basic theory that will support our methods. The sections followed the order of their appearance during the project's design and implementation procedures which are presented in detail in the next chapter.

Chapter 3

Project design

3.1 Introduction

This chapter describes the work carried out throughout "Weather talk" project. All the implemented or investigated theoretical and technical aspects are analysed thoroughly. Firstly, we present a general overview of the system. Then, we focus on separate modules as well as on the ways they interfere one another.

Many sections include references to the software packages implemented. Unfortunately, only a small sample of the implemented Java classes is provided in Appendix A and the reader may need to browse the actual source code for a complete understanding.

3.2 A general overview of the system

An overview of the project's general structure is drawn in Figure 3.1. Its purpose is to help the reader understand how every module contributes to the whole as well as in what ways the procedures interfere one another. At this point we would like to mention that the system will be applied on locations in the United Kingdom (UK) only.[1]

[1] The application of the system in a broader area was technically impossible as we did not possess adequate computational resources.

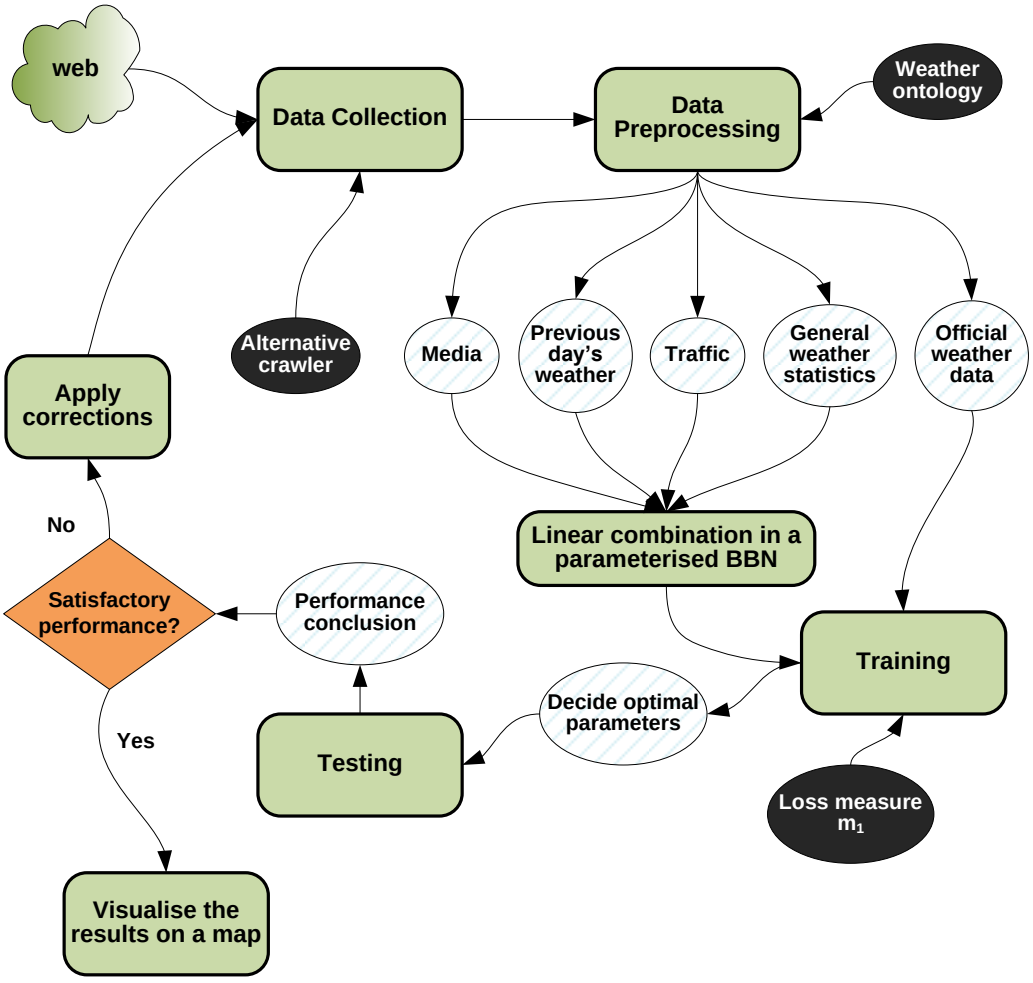

FIGURE 3.1: A general overview of the system.

The only resource of the system is the information that lies on the web. An 'alternative' crawler retrieves the input (raw) data of the system and stores them in a database (see sections 3.5, 3.6), whereas data preprocessing extracts the notions of interest from them using a weather ontology (see sections 3.4, 3.7). Media observations, official weather information which results to weather statistics, the weather of the previous day, and traffic information are combined in a parameterised Bayesian belief network (see section 3.8). Actually, we investigate more than one combinations and as a result we end up with seven Bayesian inference schemes. We train the schemes' parameters by trying to increase the accuracy of the weather inference. The loss measure used for the comparison between an inferred

weather state and the official one is the Euclidean norm of their difference (m_1 in Figure 3.1 denotes the average Euclidean norm of a training set). After deciding the values for the parameters of each scheme, we test their performance on a testing set. All the information about experiments and performance quantification measures is provided in chapter 4. If the results are not satisfactory, then main modules of the project such as the weather ontology or the crawler should be modified (manually). The inference decision for each scheme is presented on a weather map.

3.3 Definition of weather state

An important part of the project was the weather state definition because the evaluation of the final outcome is based on this. The official weather feed we use includes approximately fifty different weather conditions.[2] It would have been impossible to regard all of them separately because the difficulty of inference increases exponentially. As a result, we have decided to focus our research in six main weather conditions or **weather classes**, namely, rainy (r), cloudy (c), windy (w), fair/sunny (f), snow (s), and reduced visibility (v) (the symbol in the parenthesis denotes their representation from now on). Obviously, snow weather state was unlikely to show in our weather results, but it still was a challenge to prove that texts and articles on the web will not refer to it (or their reference will be relatively insignificant) as a weather condition taking place in a UK's location during summer.

Formally, let X be a random variable representing the weather state at a given time and location. The set of possible values for X is made of the following **weather classes** {*rainy* (**r**), *cloudy* (**c**), *windy* (**w**), *fair/sunny* (**f**), *snow* (**s**), *reduced visibility* (**v**)}.

Therefore, a weather state can be represented as a distribution of probability over X, which results to a vector of size six (whose elements add up to one).

3.4 Weather ontology

Weather talk is a semi ontology-driven framework because a big part of it depends on the weather ontology's definition. On the contrary, data collection is independent from the

[2] For retrieving official weather information, we used Yahoo! Weather feeds (http://developer.yahoo.com/weather/).

weather ontology, a fact that results to a more flexible system (changing the ontology does not affect the collected data and only changes their processing). The general idea behind the ontology was that, on the one hand, it defines the set of weather related words or phrases and on the other it assists in reducing the inference problem from a general decision over all possible weather conditions to a set of six weather classes.

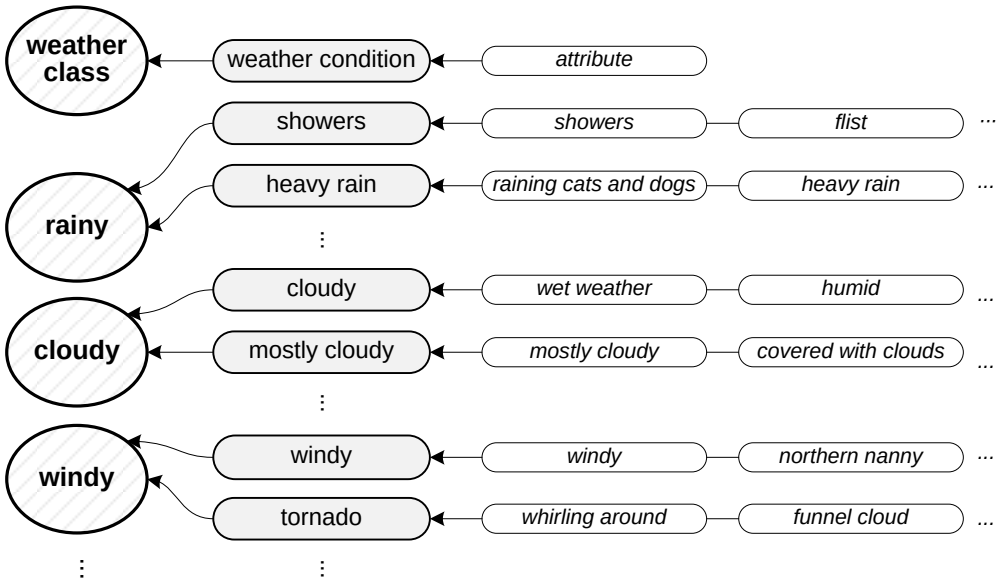

FIGURE 3.2: Weather ontology structure.

A graphical representation of the weather ontology is drawn in Figure 3.2 and part of its e**X**tensible **M**arkup **L**anguage (XML) source code is provided in Appendix B.5. Its structure is overall simple; every general weather class has mappings to specific weather conditions which in turn are connected with weather related words or phrases. The weather conditions are the same with the ones used in the official weather feed. A **D**ocument **T**ype **D**efinition (DTD) document was created for validating and keeping the structure consistent.

We tried to avoid casual or ambiguous mappings and we focused our efforts in collecting specific weather related words or idioms when possible.[3] One of the many resources for searching synonyms or definitions was WordNet.[4]

[3] *e.g.* an article with rare British or Scottish terms about the weather was used http://www.weatheronline.co.uk/reports/wxfacts/British-Weather-Terms.htm, June 2008.

[4] WordNet Search, http://wordnet.princeton.edu/perl/webwn.

3.5 Database description

Structured **Q**uery **L**anguage (SQL) and MySQL **R**elational **D**ata**b**ase **M**anagement **S**ystem (RDBMS)[5] were used for the development and maintenance of the project's database. The database schema is presented in Figure 3.3.[6]

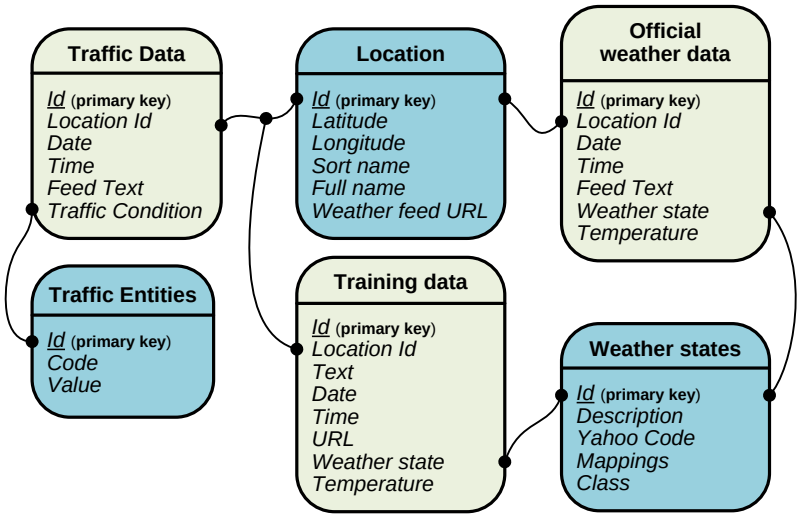

FIGURE 3.3: The main database schema of "Weather Talk". The names on the figure are not identical to the original ones. Connectors denote a foreign key relationship.

Database consists of the following six entities (or tables):

- *Location* - This table holds almost every possible location in the UK with its respective latitude and longitude values. The original source of this table was the National Geospatial-Intelligence Agency.[7] The attribute 'Weather feed URL' contains an official weather feed URL mapped to this location (see section 3.6).
- *Official Weather Data* - For every location with an official weather feed, we store periodically (every three hours), the current official weather observation. The content of this entity can be used for retrieving average weather statistics over a long time period but cannot be used for evaluating the weather of a day, since the time frame of data collection is large and it does not follow the weather variations.

[5] About MySQL RDBMS, http://www.mysql.com/about/.
[6] Database consists of a larger set of entities but here we present only the most important ones.
[7] National Geospatial-Intelligence Agency, http://earth-info.nga.mil/gns/html/cntry_files.html, June 2008.

- *Weather States* - Contains the all the possible official weather states, their mappings in the weather ontology, and the general weather class they belong to.
- *Training Data* - This entity is used for storing the system's input data. These data were used for training and testing the performance of the decisions over the weather states. Each data set consists of the publication's date/time, and the text of an article, an inferred location from the article, and the official weather observation for this inferred location at the time of data collection.
- *Traffic Data* - Stores the traffic condition for a location on a specific date and time. We focus on the severity indicator of the accidents / incidents (see section 3.7.1).
- *Traffic Entities* - Holds a map of the official traffic feed's entities. Each entity has a code and a description.

In order to secure data consistency and achieve a better public key indexing, all the public keys are unsigned big integers with an auto increment property.[8] Most of the attributes of each entity are indexed to achieve a significantly better performance during the data preprocessing stage. This slows down insignificantly the crawling procedures. The size of the database is more than 18 Gigabytes[9] and the new data insertions that take place per day are 65,000 on average.

3.5.1 Using the database

Apache's Cayenne Application Programming Interface (API),[10] which is a framework for mapping relational database schemas to Java objects,[11] was used to interfere with the database. CayenneModeler, a graphical package of Cayenne, was the tool used for re-engineering the database schema to an interface of Java classes in *weathertalk* Java package.

The main drawback of Cayenne API is that it does not handle cache problems automatically. For example, during queries which retrieve a large set of data, it tries to convert the results to Java objects (at once) without applying any paging method. As a result, Java runs out of heap space even it is manually extended during execution time. The problem was resolved

[8] In MySQL the unsigned big integer data type ranges from 0 to $2^{64} - 1$.
[9] Record taken on 01/09/2008.
[10] Apache Cayenne, http://cayenne.apache.org/.
[11] Apache Cayenne Documentation, development release 2.0, http://cayenne.apache.org/doc20/.

by applying a fixed page size manually (inside the source code). In some occasions, even this was not enough; therefore, we had to destroy (and then re-create) the objects in the cache which slowed down the operation significantly.

3.6 Data collection

Data collection provides the input of our system. Thus, it is very important to be as stable as possible. The ideal medium for data collection would have been a large-scale dedicated crawler as it gives a more flexible and customisable character to the project. However, the implementation of a crawler was not allowed by the short time available for this project or our practical resources. Therefore, we used an unconventional crawling method which uses one of the best search indices, namely Google's search engine.[12] Before crawling, we had to secure some standard resources such as the official weather feeds. Java class *MapFeedsToLoc* was implemented for that purpose; it automatically creates all the possible URLs for UK weather official weather feeds, and then parses a retrieved feed in order to map its URL to a location in the database.

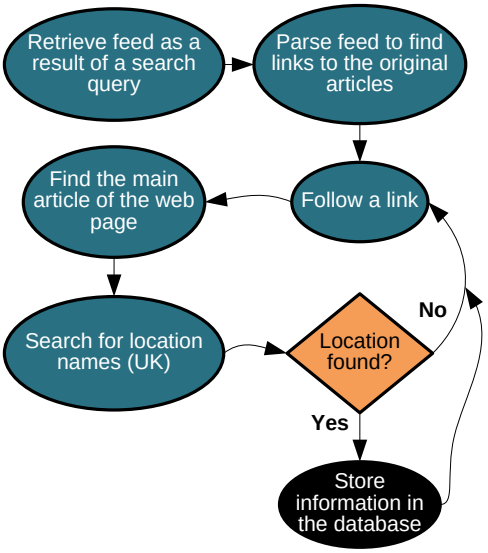

FIGURE 3.4: Data collection by an alternative crawling method.

[12] Google Inc., http://www.google.com/.

The basic steps of the crawling method used in this project are presented in Figure 3.4. The most important parts of them were implemented in *FollowRSS* class of *weathertalk* Java package.[13] After creating a search query in Google News,[14] we save the URL of the RSS feed[15] for the results of this search. We have used three different kinds of queries in order to get a more diverge set of web documents. The first one fetches the most recent or the most recently indexed articles from UK resources or from resources that may refer to UK. The second and third queries target a specific set of locations and retrieve the most recent articles or the most relative[16] ones respectively. The location set consists of fourteen cities that cover different regions on the UK's map {Belfast, Birmingham, Brighton, Bristol, Edinburgh, Glasgow, Leeds, Liverpool, London, Manchester, Norwich, Nottingham, Portsmouth, and Swansea}. These locations would be the inference target of the project (see section 3.7).

The RSS feeds created from the search queries update themselves frequently (approximately in less than one minute) and provide totally new results every two to five minutes.[17] Rome Project libraries were used to retrieve and parse RSS feeds.[18]

After retrieving the RSS feed, we parse it in order to get the links of the articles it syndicates. This is necessary because the feed only provides a summary on each article which usually is its first five lines. Then we visit the original articles' web pages and using an **H**yper**T**ext **M**arkup **L**anguage (HTML) scraper we are able to retrieve the main article text or an approximation of it.[19] Then, we scan the text looking for words that start with a letter in uppercase and when we find one we crosscheck it with the locations' names in our database for possible similarities. Locations may have the same name but different coordinates, and as a result this sometimes creates noise in our results. For the set of locations found in the text, we retrieve their official weather feed observation (at the time of collection) and

[13] FollowRSS source code is provided in Appendix B.3.

[14] "Google News is a computer-generated news site that aggregates headlines from more than 4,500 English-language news sources worldwide, groups similar stories together and displays them according to each reader's personalized interests.", Google News, http://news.google.com/intl/en_us/about_google_news.html, August 2008.

[15] RSS stands for RDF Site Summary, or Rich Site Summary, or Really Simple Syndication depending on its author and version, http://en.wikipedia.org/wiki/RSS_(file_format).

[16] Relevance in this context denotes that the articles will contain the names of the target locations with high probability.

[17] These statistics are non guaranteed approximations. They have been derived in the first stages of the project through extensive testing but depend on external factors.

[18] Rome Project, https://rome.dev.java.net/.

[19] By using the term approximation, we mean that the text may not be in its complete version or it may sometimes include HTML tags.

we store it in the database. The latter is carried out by *YahooWFParser* Java class (see Appendix B.1).

Traffic collection has a more static implementation because it uses only one resource, the BBC's UK road travel RSS feed.[20] At first, we parsed a traffic entities file[21] in order to retrieve all the possible traffic events and create an index in the database (Traffic Entities table). Having that, we implemented *weatherTalkTrafficData* Java package which collects UK road travel data from BBC's RSS feed twice a day, parses them, retrieves the actual traffic events and their respective severity indication by using the mapping from Traffic Entities table, and then stores the filtered information in the Traffic Data table (the main class of this package is presented in Appendix B.2).

3.7 Data Preprocessing

Data preprocessing is the most intensive part of the project in terms of the amount of source code written and the computational power needed. During this stage, we use the retrieved data in order to produce statistical documents. These documents will be the input of the training phase (see section 4.4). *weatherTalkStat* Java package includes all the necessary classes for data preprocessing. Class *Stats* from this package is presented in Appendix B.4.

As we mentioned in the previous section, we had to focus on a fixed number of locations since the data extraction procedures demanded a lot of resources. Data preprocessing takes place at the end of each day. Firstly, we retrieve all the web documents that are geolocated near the target location. In the beginning, a very small radius is applied. This radius increases dynamically until the number of documents retrieved allows us to make decent statistical conclusions. If the radius exceeds a specified range limit, then we do not include this location to the final conclusions output. Weather ontology is used in the second part in order to search for weather related words in the documents. Again if the weather related words are not enough for conducting conclusions, we increase the radius (until it reaches its specified limit). We apply TF-IDF scheme to assign a weight to each weather related term found in the web documents. As a result, problems of repeated terms inside documents are resolved with high probability.

[20] BBC Backstage, UK road travel data RSS feed, http://www.bbc.co.uk/travelnews/xml/tpegml_en/rtm.xml.

[21] Traffic entities resource, http://www.bbc.co.uk/travelnews/xml/tpegml_en/rtmML.ent

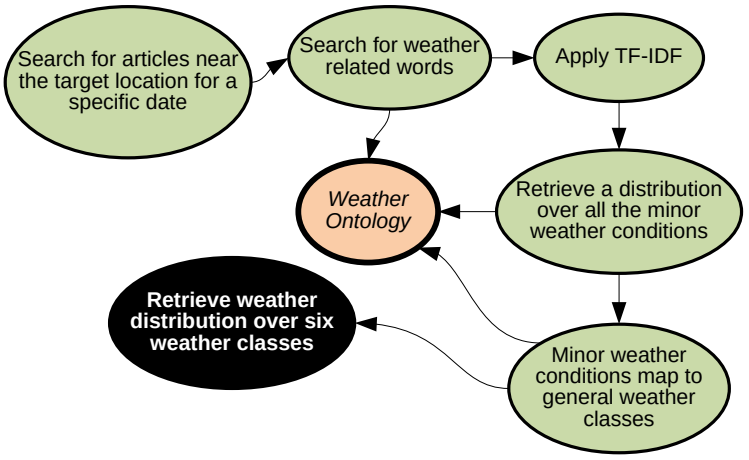

FIGURE 3.5: Data Preprocessing - Collected data are mapped to weather states.

The weights for each term form an initial weather distribution over all the possible weather conditions, which in turn results to a final weather distribution over the six predefined weather classes (weather ontology is used again to map weather words to weather conditions and then to weather classes).

For extracting the official weather state we follow a different strategy. As it was mentioned in the previous section, for all the web observations there exists an official weather indication (stored in the same tuple of the database); these indications are combined in order to extract the final official weather representation. Calculating the absolute average occurrences for each weather condition it would not have been a good approach since we may have many indications about, *e.g.* rain, but they might all be in a short time interval. Therefore, we create twelve time frames for a day (each frame has the duration of two hours) and we retrieve the absolute average for each one of them. Then, we calculate the average of these twelve time-framed official weather observations. This is the official weather representation for a location/date pair.

3.7.1 Mapping traffic information to weather states

Traffic data alone were just an observation of the severity indicators for traffic events[22] that happened during a day at a specific location. These severity indicators take four possible

[22] Event is used to refer to a traffic incident or accident.

values {slight (**sl**), medium (**med**), severe (**sev**) or very severe (**vsev**)}. We created a scheme that converts these indications to a weather state in order to be able to use traffic as an additional inference module.

At first, we defined a simple traffic score function, ts, that assigns a weight to each severity indicator, $2, 5, 7$, and 10 for sl, med, sev, and $vsev$ respectively. It produces a traffic score by multiplying the occurrences of each severity indicator by the respective weights:

$$ts(\#sl, \#med, \#sev, \#vsev) = 2 \cdot \#sl + 5 \cdot \#med + 7 \cdot \#sev + 10 \cdot \#vsev, \tag{3.1}$$

where # denotes the number of occurrences for a severity indicator.

We assign a traffic score to each location/date pair. When it comes to infer a weather state from a traffic score, the following method is applied:

- We calculate the average traffic score (α) for this location by using the traffic data of the last thirty dates. This creates an α that changes dynamically every day.
- Then, we compare the current traffic score of a location (cts) with α and we retrieve a relative traffic score Rts, $Rts = cts/\alpha$.
- Rts belongs to one of the seven relative traffic score cells as it is indicated in Figure 3.6. Every traffic cell maps to a prior weather probability.

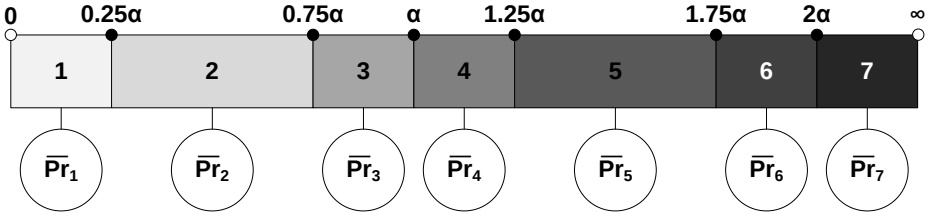

FIGURE 3.6: Traffic is mapped to weather states. α denotes the average traffic score and $\overline{Pr}_i, i \in [1, 7]$, denotes the prior weather probability for the connected traffic cell.

These prior weather probabilities are defined by performing a statistical analysis on the correlation between the relative traffic scores and the official weather states during the last thirty days. For each one of these days, we map its official weather state to a traffic score cell depending on the relative traffic score value. At the end, we calculate the average of the

official weather states sets for each traffic score cell which results to the prior probabilities (Pr_1 to Pr_7 in Figure 3.6).

3.7.2 Produce a dynamic weather prior probability

Another module that contributes to weather inference is a statistical average of the weather for a location. We refer to it as Weather prior probability because it uses previous statistical knowledge. Weather prior probability changes dynamically (like traffic inference) because it is extracted from the observations of the last thirty days for a specific location. The procedure is overall simple since we just have to retrieve the statistics of the last thirty days (for this location) and calculate their average.

3.8 A Bayesian scheme for weather inference from media observations

This section introduces the next phase of the project. The proposed method follows the same principal as the one in [Dur98] for probabilistic evolution models. At this phase, we have secured the data retrieval and preprocessing methods and we want to build appropriate schemes in order to use this input. As a first step, we design a Bayesian belief network which aims to infer the weather state for a location/date pair by using the web media observations only.

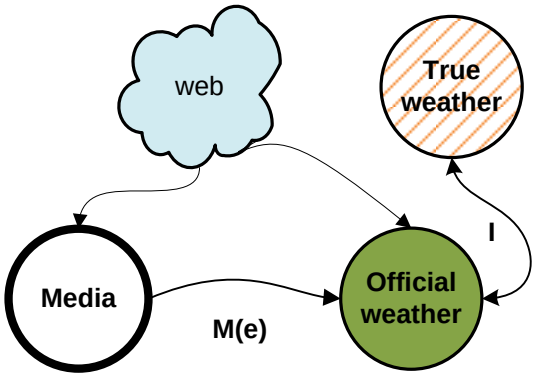

FIGURE 3.7: Bayesian belief network for weather inference from media observations

Figure 3.7 is a graphical representation of this network. Both media documents and official weather observations derive from the web. We assume that true weather is identical to the official weather feeds. Another assumption we make is that the true weather (and in turn the official weather feed) has a probabilistic correlation with the media observations.

$M(e)$ in Figure 3.7 expresses this probabilistic correlation by forming a Bayesian driven parameterised (on parameter e) matrix. It is a factor that applied to media observations makes them to converge as better as possible with the official weather and thus with the true weather.

As it was mentioned in Section 3.3, the distribution of probability for the media observations is represented as a vector, $\overline{W}_m$, of size six

$$\overline{W}_m = \begin{pmatrix} \Pr(r_m) \\ \Pr(c_m) \\ \Pr(w_m) \\ \Pr(f_m) \\ \Pr(s_m) \\ \Pr(v_m) \end{pmatrix} .$$

Similarly the official weather state is denoted by vector $\overline{W}_o$ of the same structure and size.

Let M be a matrix that holds the probabilities for all the possible transitions (conditional probabilities) between weather classes. Its format follows the Bayesian principle and it is the following:

$$M = \begin{pmatrix} \Pr(r_o|r_m) & \Pr(r_o|c_m) & \Pr(r_o|w_m) & \Pr(r_o|f_m) & \Pr(r_o|s_m) & \Pr(r_o|v_m) \\ \Pr(c_o|r_m) & \Pr(c_o|c_m) & \Pr(c_o|w_m) & \Pr(c_o|f_m) & \Pr(c_o|s_m) & \Pr(c_o|v_m) \\ \Pr(w_o|r_m) & \Pr(w_o|c_m) & \Pr(w_o|w_m) & \Pr(w_o|f_m) & \Pr(w_o|s_m) & \Pr(w_o|v_m) \\ \Pr(f_o|r_m) & \Pr(f_o|c_m) & \Pr(f_o|w_m) & \Pr(f_o|f_m) & \Pr(f_o|s_m) & \Pr(f_o|v_m) \\ \Pr(s_o|r_m) & \Pr(s_o|c_m) & \Pr(s_o|w_m) & \Pr(s_o|f_m) & \Pr(s_o|s_m) & \Pr(s_o|v_m) \\ \Pr(v_o|r_m) & \Pr(v_o|c_m) & \Pr(v_o|w_m) & \Pr(v_o|f_m) & \Pr(v_o|s_m) & \Pr(v_o|v_m) \end{pmatrix}$$

where, *e.g.* $\Pr(r_o|c_m)$ denotes the probability of weather class rainy based on the observed probability of weather class cloudy in media. Using a more strict notation,

$$\Pr(r_t|c_o) = \Pr(W_{true} = r|W_{obs} = c),$$

where W_{true} and W_{obs} denote 'true weather class' and 'observed weather class' respectively.

The inference decision vector $\overline{D}$ is the product of M and $\overline{W}_m$,

$$\overline{D} = \begin{pmatrix} r \\ c \\ w \\ f \\ s \\ v \end{pmatrix} = M \cdot \overline{W}_m. \tag{3.2}$$

It becomes obvious that the inferred probability for, *e.g.* , cloudy weather class is equal to

$$c = \sum_{j=1}^{6} \Pr(W_{true} = c|W_{obs} = ws_j) \cdot \Pr(W_{obs} = ws_j), \tag{3.3}$$

where ws is a set of all the weather classes

$$ws = \{ws_1, ..., ws_6\} = \{r, c, w, f, s, v\}.$$

Instead of using matrix M in our model, we use its parameterised version, $M(e)$. $M(e)$ assumes normal distributions for each weather class, since all the conditional (transition) probabilities have the same value $(0.2 \cdot e)$ apart from the ones that do not change weather state $(1 - e)$. It is a symmetric, Toeplitz matrix, a fact that speeds up the calculations during experiments.

$$M(e) = \begin{pmatrix} 1-e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e \\ 0.2 \cdot e & 1-e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e \\ 0.2 \cdot e & 0.2 \cdot e & 1-e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e \\ 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 1-e & 0.2 \cdot e & 0.2 \cdot e \\ 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 1-e & 0.2 \cdot e \\ 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 0.2 \cdot e & 1-e \end{pmatrix} \tag{3.4}$$

There is no specific reason for the choice of $M(e)$'s structure; this is an arbitrary structure which normalises our approach because it assumes a normal distribution of the conditional probabilities. Additionally, this choice minimises the decision problem to one parameter. A total approach that with high probability would have achieved a higher performance should focus on 25 parameters.[23]

3.8.1 Weather inference from previous day

The same method is followed in order to retrieve a prediction on the weather of a location / date by knowing the official weather of this location. The only difference in the use of a different parameter in the parameterised matrix M, denoted as f. In that case, the Bayesian belief network can be defined as a Markov chain joining two states of a weather random variable. We could have investigated this relationship further by having more states in a **H**idden **M**arkov **M**odel (HMM) but this was not one of the main interests of this project.

3.8.2 Linear combinations of Bayesian inference schemes

Figure 3.8 shows the general case of our system in which all the information available is combined in order to produce a weather inference. w_m, w_y, w_{tr}, and w_{pr} denote the weights of media observations, previous day's weather, traffic information (mapped to a probability over weather classes), and prior weather probability respectively.

In order to measure the contribution of each module to the final decision we created several intermediate schemes by combining pairs or triplets, namely (Media & Previous day), (Media

[23] Each column of M has five independent and one dependent parameter because its sum is equal to one.

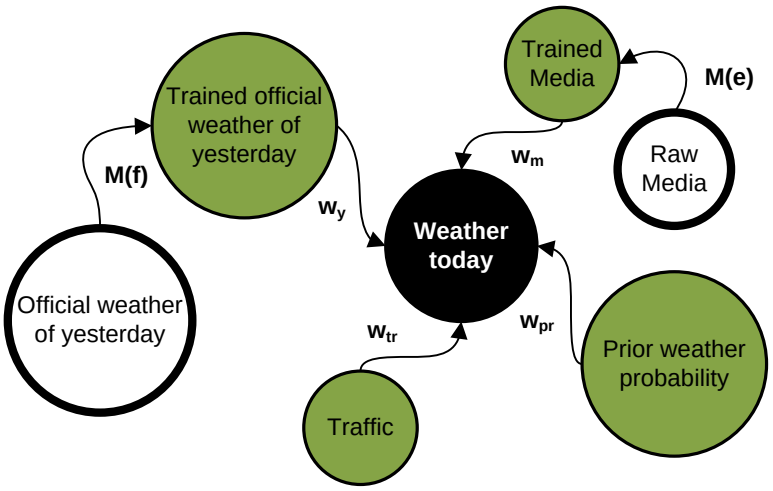

FIGURE 3.8: Bayesian belief network for weather inference from all the available input data.

& Traffic), (Media, Previous day & Traffic), (Media, Previous day, & Weather prior). During training we deal with each scheme separately (see section 4.4).

3.9 Visualising the results on a map

As a final part of this project, we created a tool[24] for visualising the inference results and be able to compare them in a more natural way with the official weather (Figure 3.9). The tool is a web application running on Apache **H**yper**T**ext **T**ransfer **P**rotocol (HTTP) server[25] developed in **PHP** (Hypertext Preprocessor) using e**X**tensible **H**yper**T**ext **M**arkup **L**anguge (XHTML), **C**ascade **S**tyle **S**heets (CSS), JavaScript, and e**X**tended **M**arkup **L**anguage (XML) input. Google Maps API[26] was used for the creation of the maps and for translating the XML input documents to points on the map.

The web site consists of two weather maps; the first one displays the official weather states for a set of locations on a specific day, whereas the second one can display the results of every weather inference scheme for that day. The panel on the right helps the user to browse through the results of all the schemes for the last forty days. For every pair of weather states

[24] The tool is available at http://weathertalk.lampos.net/, August, 2008.
[25] Apache HTTP Server Project, http://httpd.apache.org/.
[26] Google Maps API Terms of Service, http://code.google.com/apis/maps/terms.html.

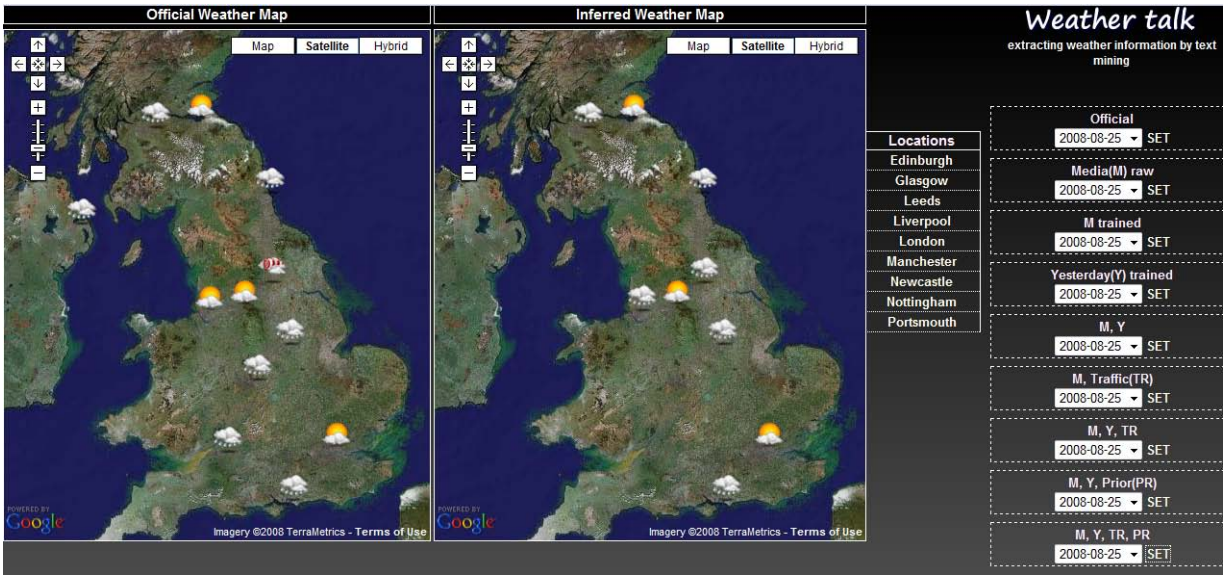

FIGURE 3.9: Weather talk web site consists of an official weather and an inferred weather map. The panel on the right is used for viewing the results of all the schemes.

there is a dedicated weather icon which is used as a marker on the weather map. By clicking this marker, a pop-up window displays the actual percentages for the three major weather states for the respective location/date pair (Figure 3.10).

The website operates as a web service and it is fully automated. The XML inputs for each scheme are created once per day as an automatic scheduled task by *Conclusions* Java class of the *weatherTalkStat* package. This class uses the extracted statistical facts from preprocessing stage, applies the Bayesian models, and records the results in appropriately formatted XML documents. The XML outputs are uploaded to pre-specified directories of the web site by *ftpUploader* Java package[27] using **F**ile **T**ransfer **P**rotocol (FTP) and then JavaScript methods translate them to points on a map. The use of XML files ensures the portability of this application.

[27] *ftpUploader* package makes use of the open source EnterpriseDT FTP package, http://www.enterprisedt.com/products/edtftpj/overview.html.

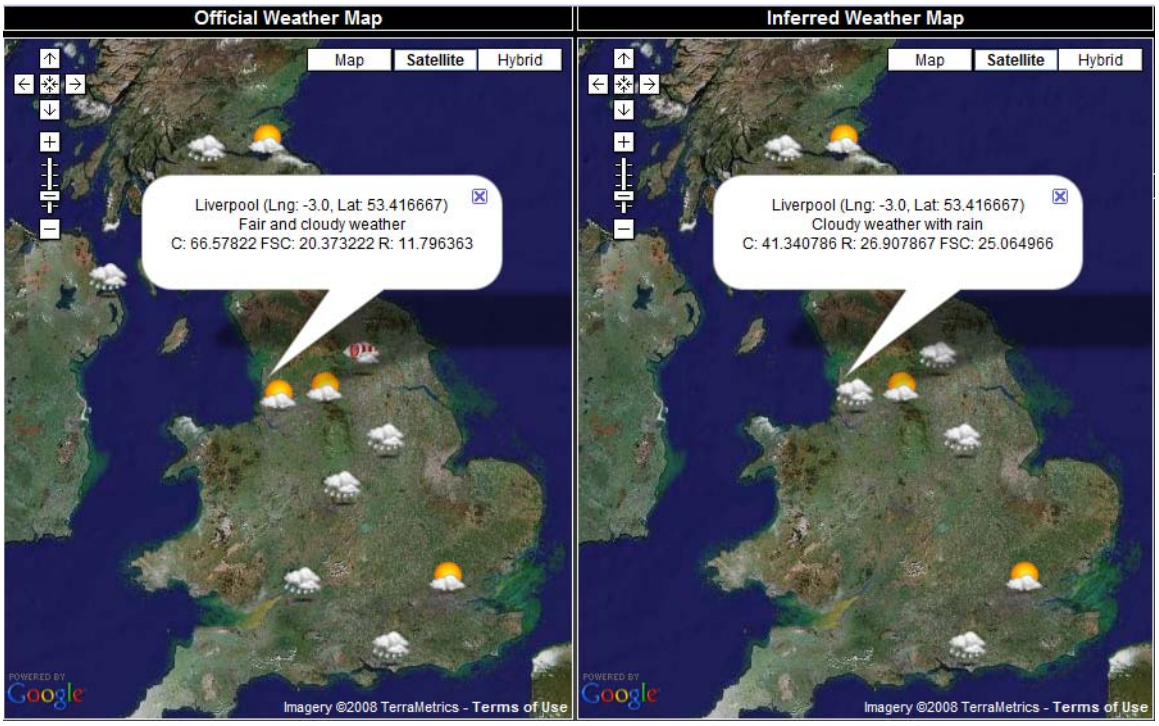

FIGURE 3.10: Weather maps provide additional information when clicking on a weather icon.

3.10 Automated procedures

Another feature that may not have become obvious from the sections above is that all the procedures of the project are automated. Data collection is performed by scheduled 'cron' jobs[28] and when a day has officially passed, preprocessing takes place by calling another scheduled cron job. Data preprocessing produces many different documents (raw statistics for a location, inferences for every scheme, XML documents for every scheme). The XML documents are uploaded on the website (by calling again an FTP uploading service). As a result, everything runs smoothly without human interference. Of course, the system has to be checked frequently because it does not depend only on itself but also on external inputs, *e.g.* a change on an official feed from its publisher may result in unexpected program failures.

[28] 'Cron' is a short version for chronograph.

3.11 Summary

In this chapter we described in detail the major part of the work carried out throughout this project. We explained how data collection works and how the weather ontology interacts with data preprocessing. The core Bayesian scheme for weather inference from media observations was analysed. In the end, we presented a tool for visualising the weather inference results. In the next chapter, we refer to the experimental results of the project during training and testing stages.

Chapter 4

Experimental results

4.1 Introduction

In this chapter, at first, we present the loss measures we applied in order to quantify the performance of each scheme. Then, we present the experiments carried out on the training data set for extracting the optimal values for the parameters of each scheme. Finally, we test the performance of the schemes using a testing as well as a randomised set. Randomised sets are used in order to define the significance level of each scheme's performance (p-value). The input data for the experiments were extracted and preprocessed using Java packages (as it was presented in chapter 3, sections 3.6 and 3.7), and the experiments were carried out in MATLAB.

4.2 Euclidean norm as a loss measure (m_1)

The weather state of a location for a specific day is represented as a distribution over six weather classes. As a result, its algebraic representation is a vector of size six. An indication that the inferred weather state, $\overline{W}_i$, is close to the official weather state, $\overline{W}_o$, can be derived by calculating the Euclidean norm (or 2-norm)[1] of their difference, $\overline{dif}$.

[1] Appendix A.2 includes the definition for a p-norm.

$$\begin{aligned} \overline{dif} &= \overline{W}_o - \overline{W}_i, \\ \|\overline{dif}\|_2 &= \sqrt{d_1^2 + ... + d_6^2}. \end{aligned} \tag{4.1}$$

During the experiments, we apply this method on large data sets. Suppose that we have a data set of size n. Then W_i, W_o, and dif are matrices of size $n \times 6$. In this case, we are interested for the average value of dif, which in turn is the average value of the Euclidean norms (of the subtraction between the inferred and the official weather distributions). We denote this loss measure as m_1 and during the experiments, we search for parameters' values which will result in a minimized value of it.

4.3 Inference of two major weather states as a performance measure (m_2)

A practical weather inference, *e.g.* the one used in weather forecasts, mentions one or two main weather conditions which represent a location for a day. By applying the following 'practical' weather quantification, we tried to evaluate this representation.

The performance measure compares the two official weather classes with the highest percentage with the two inferred ones without taking into consideration their order. Let the sets $\{i, j\}$, and $\{k, l\}$ denote the two major official and inferred weather classes respectively.

$$\begin{aligned} i &= \operatorname{argmax} \overline{W}_o, \quad j = 2 \operatorname{argmax} \overline{W}_o, \text{ and} \\ k &= \operatorname{argmax} \overline{W}_i, \quad l = 2 \operatorname{argmax} \overline{W}_i. \end{aligned} \tag{4.2}$$

If the sets $\{i, j\}$ and $\{k, l\}$ are the same, then the weather inference decision, d, is correct (value 1), otherwise it is not (value 0).

$$d = \begin{cases} 1, & \text{if } \{i, j\} = \{k, l\} \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

Consequently, for data sets of size n, we just have to relatively count how many times the weather inference is correct. This is the percentage of success in inferring the two major weather states, denoted by m_2.

4.4 Parameter tuning

Loss measure m_1 was applied on a training data set of 50 days focused on the target locations (see section 3.6) in order to decide the optimal values for parameters e, and f initially. Parameters e and f define the contents of the matrices used for weather inference from media and the weather of the previous day, respectively (see section 3.8).

Experiments on the first Bayesian scheme (weather inference from media observations only) indicated that the best performance is retrieved for $e = 0.261$ (Figure 4.1).

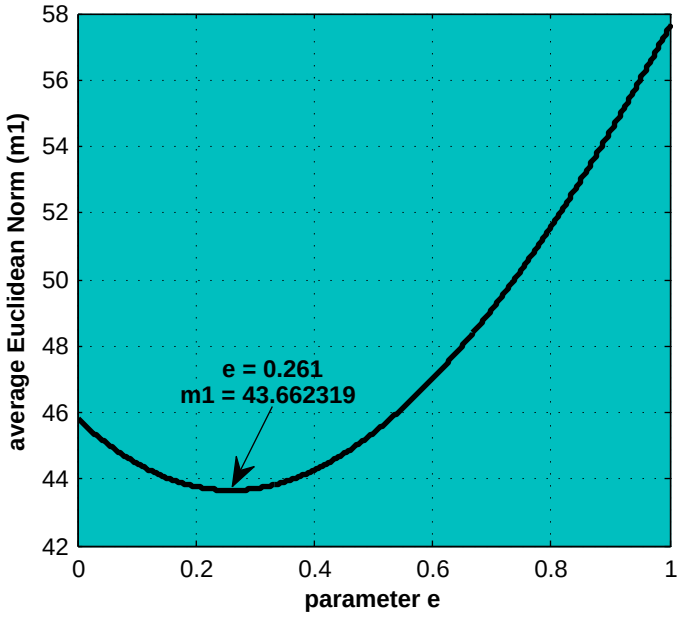

FIGURE 4.1: Weather inference from media observations - The best performance for the scheme is achieved for $e = 0.261$.

Similarly, experiments on the second basic Bayesian scheme (weather inference from the previous day's weather state) indicated that $f = 0.0969$ optimises the results (Figure 4.2). From the training results, we expected the weather of the previous day to perform better

than media observations during testing. Of course, when the weather changes rapidly from one day to another, we expect media observations to perform better.

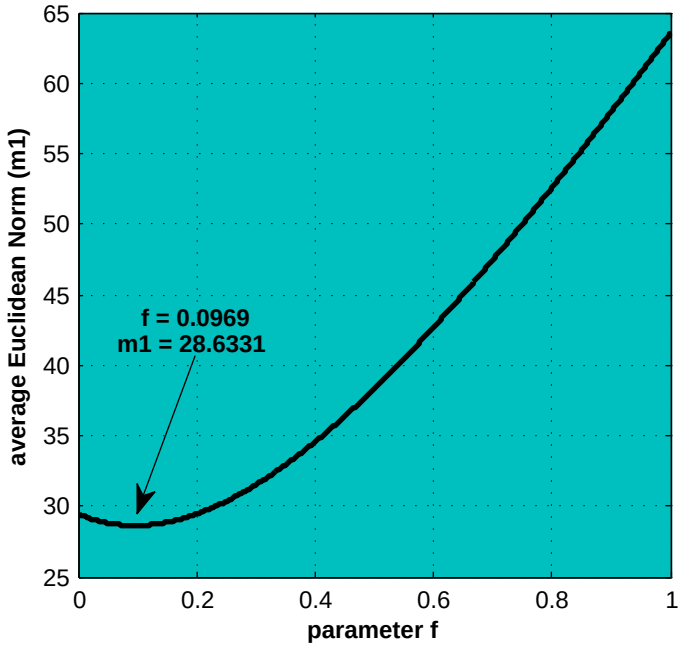

FIGURE 4.2: Weather inference from previous day's weather - The best performance for the scheme is achieved for $f = 0.0969$.

In the next five schemes, we use these optimised values of e and f trying to find the combinations of weights that provide better results than the ones retrieved from the initial two schemes. In every scheme, media observations had at least a relative averaged value.[2] Table 4.1 lists the optimal parameters for each scheme and the performance they achieved during training. The most important conclusion from this table is that the addition of a module in a scheme improves the final outcome, *i.e.* it minimises the value of m_1. Therefore, the best performance (during training) is achieved from the last scheme that combines Media, Previous day's weather, Traffic, and Weather prior modules.

[2] The relative average of the weights of three elements in a scheme is equal to $\frac{1}{3}$.

TABLE 4.1: Optimal parameter values for the schemes defined by loss measure m_1. Weights for Media, Previous day's weather, Traffic and Weather prior modules are denoted with w_m, w_y, w_{tr}, and w_{pr} respectively.

Schemes	e	f	w_m	w_y	w_{tr}	w_{pr}	m_1
Media	0.261	-	-	-	-	-	43.6623
Previous day	-	0.0969	-	-	-	-	28.6331
Media & Previous day	0.261	0.0969	0.5	0.5	-	-	31.2169
Media, Previous day's & Weather prior	0.261	0.0969	0.33	0.36	-	0.31	27.8087
Media & Traffic	0.261	0.0969	0.5	-	-	0.5	31.6135
Media, Previous day & Traffic	0.261	0.0969	0.33	0.35	0.32	-	27.7904
Media, Previous day, Traffic & Weather prior	0.261	0.0969	0.25	0.3	0.3	0.15	26.6664

4.5 Performance

For estimating the performance of each scheme, we used the trained parameters on a different (denoted as testing) set of 20 days. In this case, we retrieved the value of measure m_2 as well, which is a percentage of success (in inferring the two major weather states) for each scheme. The testing results together with the values of m_1 during training are presented on Table 4.2.

Figure 4.3 compares the behaviour of media observations for different values of e in the training and in the testing set. The overall behaviour is similar, despite the fact that testing set achieves its best performance for a significantly smaller value of e.

Figure 4.4 expresses the same comparison for the inference from previous day's weather scheme. These schemes correlate more, but again, the optimal value for the testing set is achieved for a smaller value of f than the one proposed from the training set.

The performance of the schemes is reduced when the values of e and f override a threshold which is usually below 0.25. This indicates that the transition (or conditional) probabilities of the diagonal in the matrices of probability (M) are significantly bigger ($1 - e \geq 0.75$) than the other values ($0.2 \cdot e \leq 0.05$).

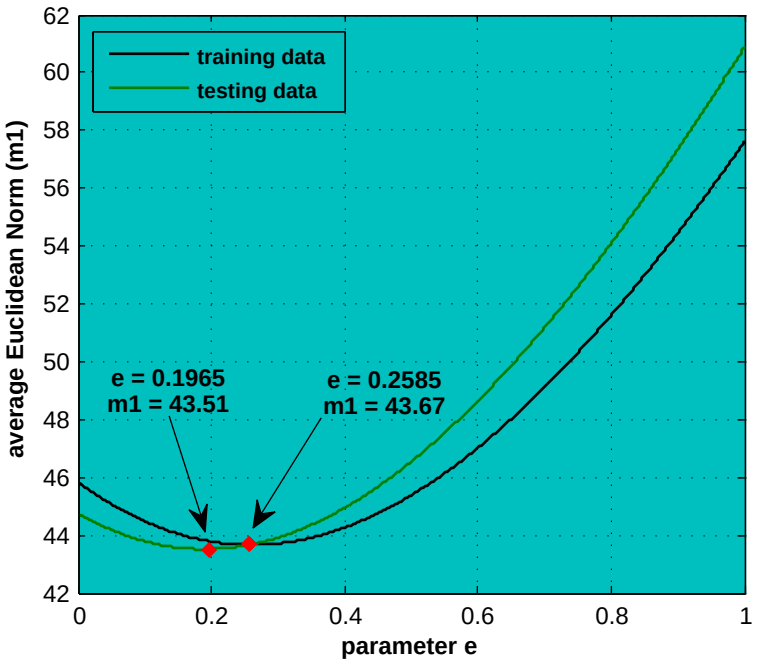

FIGURE 4.3: Compare training and testing set loss in media-only inference.

Table 4.2 demonstrates that the schemes perform slightly better in m_1 during testing than during training. It is only the Media scheme which remains approximately at the same performance level. The performance of m_2 improves after the addition of a module on the final inference result. For Media observations we predict the two major weather states correctly with a percentage of success equal to 52.22%, whereas in the final scheme this percentage is increased to 63.51%. Traffic improves Media inference by a factor of 4.5%, proving that traffic can be used effectively for inferring the weather. Surprisingly, the weather inference from the previous day does not perform as good as we expected in m_2, despite the fact that it does have the best performance in m_1.

4.5.1 P-values

Each performance quantification measure has its own p-value.[3] P-value indicates whether a scheme's performance is significant. In our case, the significance is defined as the probability that a random data set performs better than or equal to an actual/valid set of data. We

[3] The notion of p-value is explained in Appendix A.3.

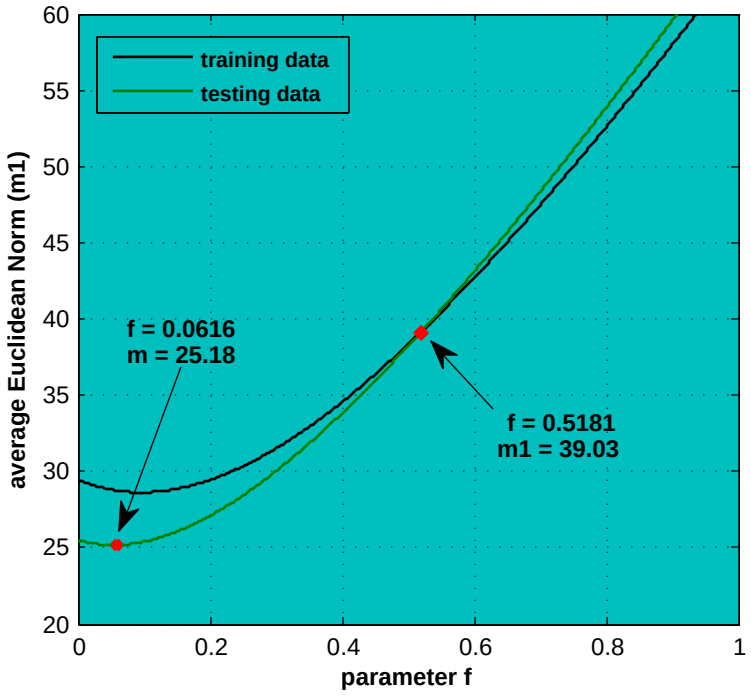

FIGURE 4.4: Compare training and testing set loss in weather inference from previous day's weather.

TABLE 4.2: Performance for each Bayesian inference scheme. An extra column with the results of loss measure m_1 during training has been added for comparing with their training equivalents.

Schemes	m_1 (testing)	m_1 (training)	m_2
Media	43.6778	43.6623	52.22%
Previous day	25.3556	28.6331	56.38%
Media & Previous day	30.6963	31.2169	60.4%
Media, Previous day & Weather prior	27.0426	27.8087	62.42%
Media & Traffic	30.5797	31.6135	56.73%
Media, Previous day & Traffic	26.9322	27.7904	62.84%
Media, Previous day, Traffic & Weather prior	25.6541	26.6664	63.51%

TABLE 4.3: P-values for performance measures m_1 and m_2 in all the schemes. The significance thresholds for m_1, m_2 are $\alpha_1 = 0.05$, $\alpha_2 = 0.07$ respectively.

Schemes	**P-value for** m_1	**P-value for** m_2
Media	0.1207	0.0684
Previous day	0.0512	0.0667
Media & Previous day	0.0248	0.0674
Media, Previous day & Weather prior	0.0173	0.0674
Media & Traffic	0.02	0.0672
Media, Previous day & Traffic	0.0164	0.0673
Media, Previous day, Traffic & Weather prior	0.0164	0.0677

define two significance thresholds α_1 and α_2 for m_1 and m_2 respectively. We set $\alpha_1 = 0.05$ (which is the most common significance threshold used in statistics) and $\alpha_2 = 0.07$ because a random choice of 2 from a set of 6 elements can be done with 15 different ways which results to a probability of 0.0667 in picking the correct two elements from the set.

Random data sets were created using the following method. MATLAB's pseudo-random number generator[4] was used for obtaining a vector of size six. The range of this vector's values is from 2^{-53} to $1 - 2^{-53}$. For every element of the vector, we calculate its relative value, which is its actual value divided by the sum of all vector values. The six relative values give us a randomised vector whose elements add up to one. Since the positioning of the elements in the vector is important, we added another layer of randomness by permutating them pseudo-randomly (using *randperm* function in MATLAB).

The p-values obtained for each scheme are presented on Table 4.3. As far as measure m_2 is concerned, every scheme's performance is significant since for a random set of data the inference outcome was proven to be random, *i.e.* close to or equal with 0.0667. In terms of loss measure m_1, the performance of Media-only scheme is not significant (see section 5.3), whereas all the other schemes indicate significant performance (with Previous day's weather scheme being slightly bigger than the threshold α_1). As before, the more modules a scheme has, the more statistically significant the outcome becomes.

[4] MATLAB uses the Mersenne Twister algorithm by Nishimura and Matsumoto [MN98] for creating pseudo-random numbers drawn from a uniform distribution, MATLAB 7.6 Help.

4.6 Summary

In this chapter we described the methods used for quantifying the performance of the project and we presented the experimental results for all the weather inference schemes. The Euclidean norm was applied during training in order to retrieve the optimal values for the parameters. Testing included an additional performance measure which indicated the percentage of success in inferring correctly the two major weather states for a location/date pair.

As the p-values indicate, the performance of almost every scheme is significant for both loss measures; media-only scheme's performance is not as good as we have expected. The reason for this is discussed in the next chapter.

Chapter 5

Project evaluation and conclusions

5.1 Introduction

In general, the aims and objectives of this project were achieved to the extent that time, resources and experience allowed us. This chapter offers initially a critical evaluation on the project. It states the extent to which the project has achieved its objectives and critically discusses all the choices made. In the end, we conclude this project by summarising the contents of each chapter and discussing the general outcomes.

5.2 Achievement of aims and objectives

This work included a high risk, since it was based in the hypothesis that the web documents in the media will correlate with the true weather. In fact, we have proven that this correlation exists, but it will need an even more sophisticated system to exploit it significantly. Nevertheless, p-values indicate that we have achieved significant performance in most of the weather inference schemes we tried. Traffic information was successfully mapped to weather conditions as it always improved the final decision's outcome, indicating that data fusion can assist an information retrieval process.

Other risks of this project were the availability of input data and the development of stable data collection software. In the end, we believe that the implemented packages for web

mining do provide a significant stability.[1] The developed weather ontology provides a stable framework for mapping weather related terms to weather classes during data preprocessing. Of course, data collection can be improved and recommendations on this are drawn up in chapter 6.

Standard Bayesian theory expressed by parameterised Bayesian belief networks assures a further improvement on the final outcome. All the weather inference schemes have been trained in order to retrieve the optimal values for their parameters (or weights). The absolute performance of each scheme and its significance were derived from testing. P-values indicated statistical significance in most of the schemes.

The weather map for visualising the final outcomes, mentioned as an optional objective in the interim report, may not bring any research interest but it certainly is a stable tool for checking the performance of each scheme everyday.

5.3 Correlation between weather states

In the beginning of the project we assumed that the true weather and thus the official weather web observations will have a 'significant' correlation with the weather in the media.

To formally exploit this assumption, we tried two different experiments, using the untrained media and previous day's observations (input sets) as well as the official weather observations (evaluation set). Firstly, we compared the media input set with its respective evaluation set and then with a random permutation of the evaluation set. We repeated the same experiment 10,000 times; the result indicated that the correlation between the media input and the random permutation of the evaluation set was better 6.56% of the times (p-value$_{1m}$ = 0.0656). This indicates an insignificant correlation (p-value$_{1m} > \alpha = 0.05$). Doing the same experiment, but using the weather of the previous day as an input set reveals that p-value$_{1y} = 0$ (significant correlation).

In the second experiment, we calculated canonical correlations (**C**anonical **C**orrelation **A**nalysis, CCA) to find the best possible correlation obtained by mapping each data set onto one dimensional spaces [Bor01].[2] Again, we tested media, and previous day's weather

[1] Programs have been working smoothly without crashing from the 12th of July, 2008 onwards.

[2] An external MATLAB m-file which implements Canonical Correlation Analysis has been used during experiments, Magnus Borga, Linkpings University, http://www.imt.liu.se/~magnus/cca/cca.m.

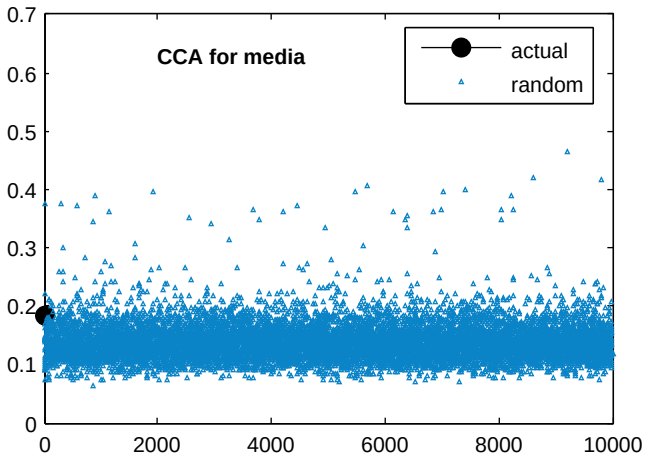

FIGURE 5.1: CCA for media data set - The performance is **not** significantly better than with random data (p-value$_{2m}$ = 0.0619).

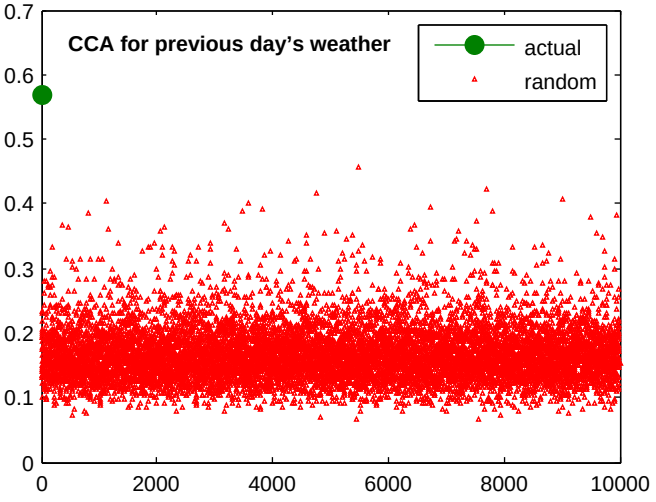

FIGURE 5.2: CCA for previous day's weather data set - The performance is significantly better than with random data (p-value$_{2y}$ = 0).

input sets against the evaluation set and its random permutations. As indicated in Figures 5.1 and 5.2, previous day's weather data performed significantly better than random (p-value$_{2y} = 0$), whereas media data sets did not succeed in performing significantly better (p-value$_{2m} = 0.0619$).

Both experiments indicate that media observations are not significantly correlated with true weather. This was the reason for retrieving an insignificant performance indication from their p-value during testing (see section 4.5.1). However, it is obvious that they do possess a signal.

5.4 Evaluation of project's modules

In this section we discuss, with hindsight, the choices made during the project. For most of them, we provide improvement recommendations in the next chapter. Obviously, this does not imply that our choices were unoptimal, but does state that operations could be performed better based on the knowledge gained from this project.

Starting with crawling methods, RSS feed crawling was overall a good choice since it provided us with relatively adequate data.[3] Google's search index is the state of the art today and as a result, only a dedicated large scale crawler might have retrieved a better input data set.

The weather ontology's development formed a significant part of this project. We tried to crosscheck the effects of each term to the final outcome (weather inference accuracy) but as the number of terms and phrases was increasing, it became almost impossible to quantify the significance of each term (at least with our current resources). On the contrary, mapping traffic feeds to weather conditions worked well. The scheme is simplistic, but the pattern it applies delivers good results.

For this project, we have chosen Bayesian scheme in order to represent the probabilistic relationship between the observed and the official weather. We gave a parameterised, normalised structure to the scheme by using only one parameter (e or f) which was not the best choice since in the matrix there exist 25 parameters that can be trained. However,

[3] It means that we could not possibly handle more data for practical reasons (resources, process time).

this was a convenient way for approximating the optimal values of the transition matrix in terms of computational power.[4]

Another decision we made was to focus our weather inference on a set of locations from the UK. We could only study a small set of locations due to the limited computational power available.[5] It would have been better to focus on a set of locations from all over the world (with different climate), but this needed broader crawling procedures.

Finally, throughout this project we assumed that official weather feeds are identical with true weather. Unfortunately, that was not always the case; updates are not as fast as the weather changes for all the locations. A good choice might have been to watch the weather reports from airports only.

5.5 Conclusions

The main aim of this project was to design and implement a system capable of inferring the weather state for a location/date pair using web media observations and statistical analysis in a Bayesian inference scheme. The decision's outcome was improved by adding weather information from the previous day and/or general weather statistics. As a further extension, we collected traffic information, and mapped it in weather conditions in order to create a data fusion (weather inference) model that enhanced the overall performance. In the end, we visualised the results on a web based weather map.

Chapter 2 of this dissertation reviewed the general background and context of this project. It defined notions such as data mining, ontology, data fusion and it explained the basic theory about BBNs and TF-IDF scheme.

Chapter 3 was a presentation of the work carried out. All the parts of the project and the ways they interfere one another were analysed thoroughly. Data collection and preprocessing by applying a weather ontology as well as the actual implementation of the Bayesian inference schemes were the main focus of this chapter.

[4] Adding one parameter results to the addition of another dimension in our calculations.

[5] Extracting statistical results for one location is a procedure that lasts approximately 20 minutes. At the same time the same machine should continue the data collection process.

Chapter 4 presented the performance quantification measures and the project's experimental results. By training the parameterised Bayesian schemes, we retrieved the optimal values for the parameters, and then we used them for testing the performance. An indication for significance in performing is provided by the p-values. The performance in most of the schemes was statistically significant.

In the current chapter, we evaluated the procedures in the project. CCA indicated that media observations and official weather did not correlate as much as we have assumed in the beginning of the project.

"Weather talk" forms a web mining framework with an ontology embedded that bases its decisions on Bayesian theory. In a period of three months, without the needed computational power, and with all the limitations that we have mentioned in this chapter, we achieved to infer the two major weather states of a location with 63.51% of success. As a result, the most important outcome of this project is that this kind of information extraction is possible and now it should be focused on different contexts.

As noted before, despite the amount of the work carried out, there is still space for improvements and extensions. Recommendations for further work are provided in the following chapter.

Chapter 6

Future development

The content of this chapter is partly based on the critical evaluation in Chapter 5. Improvements and other possible extensions of the project are discussed together with ideas for applications on totally different contexts.

6.1 Improvements and extensions on the current work

Improvements on the existing parts or extensions on the project are applicable. As we have mentioned in the previous chapter, the weather ontology could have been implemented more "carefully". A work from weather specialists and ontologists would have had better results. However, a more interesting approach would not depend on a static ontology; the creation of dynamic ontologies based on word indexing should be investigated. For example, when we know (from the official weather feeds) that it is raining in location A, we could investigate whether we can create an index of the words used in the documents that refer to A. Furthermore, can we distinguish from this index the words that map to the weather context dimension? This is a matter of further research and possible approximations to that solution may be derived from multi-dimensional **S**upport **V**ector **M**achines (SVM) classifiers.

Based on the fact that the scheme used for mapping traffic information to weather states worked overall well, we could try to improve it even more. An option would be to add more traffic cells and extend the statistical analysis on traffic to a larger period of time (more

than 2 months). A more interesting option would have been to investigate changes on the statistics for each day of the week separately because traffic has different behaviour per day.

During training, we focus on the parameters e and f of the parameterised Bayesian matrices (and then on the weights of each module). Ideally, we should have trained 25 parameters per matrix.[1] Alternatively, matrices such as K with 5 parameters to be trained may deliver a better result,

$$K = \begin{pmatrix} k & b & d & e & c & a \\ a & k & b & d & e & c \\ c & a & k & b & d & e \\ e & c & a & k & b & d \\ d & e & c & a & k & b \\ b & d & e & c & a & k \end{pmatrix}, \tag{6.1}$$

where $k = 1 - a - b - c - d - e$.

A further extension on this project could be the application of geolocating models as a feedback module during decision making. For example, in order to support a weather inference for Bristol, we can retrieve statistics from locations in a close range, extract their weather inference and then use it in another, more general scheme, which combines weather inferences about locations.

At last, an implementation of a large-scale dedicated crawler could further assist this project. The crawler should be able to distinguish dynamically which contents will provide useful information. An easy way to do that is to train the crawling targets depending on the performance of the system.

6.2 Applications on different contexts

"Weather talk" project was not about the weather. It was about investigating, designing, and implementing a model which makes a prediction that could be easily evaluated. Given the fact that the performance in most of the parts of this model was significant, the next step should be its application and adaptation to more interesting concepts and contexts.

[1] Matrices of size 6x6 result into 36 elements, but the parameters for training are 5 per column due to the stochastic structure.

A field that has received a lot of attention recently is "Opinion mining" [Liu07b], the procedure of classifying public opinions about products, individuals, places, etc. According to [HL04], one should try first to identify the different features or attributes that a document is mentioning and then decide whether an opinion about an identified feature is positive, negative or neutral. The very recent "Opinion Observer" applies an interesting and effective solution on this problem [DLY08], but going back to the problem's definition, these types of classification seem to be not enough. A more exact quantification with its attributes would be desirable, *e.g.* product A is 10% better than B because it is better priced and because of its "brand".

Traffic information enhanced the final outcome and thus more research should be dedicated on how heterogeneous kinds of information can be combined effectively for enhancing the decisions or the accuracy of the derived random variables (which in turn are defined by the nature of the inference). As an example, an interesting recent approach uses econometrics in order to determine opinions about products [GIS07].

Another challenge would be the prediction of opinions about "objects" in the future. Having a model which is able to mine opinions from the web enables us to use a large amount of training data on specific opinion targets and study how this information evolves through time. Then, we could make estimations of the transition probabilities of the system, *e.g.* what is the probability that an opinion A about a feature of an object turns to opinion B in the next state of the system. It may sound illogical but having a large amount of information at the current state may lead us to predictions of future opinions. By combining different data concepts on the same purpose (data fusion) we may end up with a very sophisticated system which could try to make targeted assumptions for the future, *e.g.* whether a sports equipment company should use a specific material or not in the future.

Appendix A

Foundations

A.1 Basic definitions from Graph Theory

The following definitions were derived from [Die00].

A **graph** $G(V, E)$ is a structure which contains a set of vertices, V, and set of edges, E, satisfying $|E| \leq |V|^2$.

A **path** in a graph is a succession of n distinct edges.

A **directed** graph is an *orientation* of an undirected graph, meaning that every edge has a direction from one vertex to another.

A directed graph is **acyclic**, when there is no path able to start and end at the same vertex.

A.2 Norm

For a vector $v = \{v_1, v_2, ..., v_n\}, v \in \mathbb{R}^n$, the *p-norm* is defined as

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}.$$

As a result, the 2-norm (known as the *Euclidean norm*) is given by

$$\|v\|_2 = \left(\sum_{i=1}^{n} |v_i|^2 \right)^{\frac{1}{2}} = \sqrt{v_1^2 + ... + v_n^2}.$$

A.3 P-value

Formally, "**p-value** is the probability of obtaining a result at least as extreme as the one that was actually observed, given that the **null hypothesis** is true".[1]

In turn, "**null hypothesis** is a hypothesis set up to be rejected in order to support an alternative hypothesis".[2]

Closer to our purpose, **p-value** is "the probability that a variate would assume a value greater than or equal to the observed value strictly by chance".[3]

As a result, in this project:

- Null hypothesis: There is no correlation between media observations and official weather observations. In other words, random input data can perform better or equal than valid input data.
- P-value: Denoting the performance as P in test set of size n, p-value p is equal to:

$$p = \frac{\#(P(random\_data) \geq P(valid\_data))}{n}.$$

Having a significance threshold α, we look for a p-value such that $p \leq \alpha$. A common value for α in statistics is 0.05.

[1] Wikipedia, http://en.wikipedia.org/wiki/P-value, September 2008.
[2] Wikipedia, http://en.wikipedia.org/wiki/Null_hypothesis, September 2008.
[3] Weisstein, Eric W. "P-Value." From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/P-Value.html, September 2008.

Appendix B

Parts of the source code

The source code of this project included a variation of programs for different platforms. Java, MATLAB, PHP - XHTML - CSS - JavaScript, XML and SQL were used. The total number of lines of code were more than 10,000. As a result, it was impossible to include all the source in the dissertation's Appendix; we only include parts of the most important source code files. We have used ellipses ("...") to denote that we have skipped a part of code. All the printing (standard output) commands have been removed. Many lines are submitted in an unconventional way in order to save space.

B.1 Weather feed parser - YahooWFParser.java

```
...
/**
 * Parses the yahoo weather feed and retrieves the requested information.
 */
public class YahooWFParser {
   private String weatherState; private String temperature;
   public YahooWFParser(String text) throws Exception{ weatherState = null; temperature = null; getElements(text); }
   ...
   /**
    * Get the weather state and temperature from a feed. @param text the text of the feed
    */
   private void getElements(String text) throws Exception{
      BufferedReader reader = new BufferedReader(new StringReader(text));
```

```
        String line; Boolean foundFlag = false; int iFlag = 0;
        try{
            while ((line = reader.readLine()) != null){
                //at the next line appear the contents of the RSS
                if (line.contains("SyndEntryImpl.description.value=")){ foundFlag = true; continue; }
                //we are interested for the contents of the third line
                if (foundFlag == true){ iFlag++; }
                if (iFlag == 3){
                    String[] weatherCond = line.split(", "); weatherState = weatherCond[0];
                    //if the temperature is not in the feed
                    if (weatherCond.length < 2){ temperature = ""; break; }
                    String[] temp = weatherCond[1].split(" C"); temperature = temp[0]; break;
                }
            }
        } catch (Exception e){ throw e; }
    }
}
```

B.2 Traffic data collection - OfficialTrafficDataDirect.java

```
...
/**
 * Uses a source from BBC in order to get traffic feeds.
 */
public class OfficialTrafficDataDirect {
    private String roadTravelDataUrlString; private String roadTravelEntitiesUrlString;
    private DataContext context;
    public OfficialTrafficDataDirect(){
        roadTravelDataUrlString = "..."; roadTravelEntitiesUrlString = "..."; context = DataContext.createDataContext();
    }
    /**
     * Gets the source code of the travel feed.
     */
    public String getRTDURLSourceCode() throws MalformedURLException, IOException{
        URL roadTravelDataURL = new URL(roadTravelDataUrlString);
        URLConnection conn = roadTravelDataURL.openConnection();
        InputStream in = conn.getInputStream(); BufferedReader buf = new BufferedReader(new InputStreamReader(in));
        String line = "", output = ""; int lineCount = 0;
        while ((line = buf.readLine()) != null){ lineCount++; output += line + "\n"; } return output;
```

```
}
...
/**
 * Parses entities resource file in order to retrieve codes and descriptions
 * for the entities of traffic feed. Then it stores them in the database.
 */
public void recordTrafficFeedEntities() throws Exception{
    String entText = getRTEntURLSourceCode();
    BufferedReader buf = new BufferedReader(new StringReader(entText));
    String line = "";
    while ((line = buf.readLine()) != null){
        if (line.contains("<!ENTITY")){
            String[] lineTokens = line.split("\""); String code = lineTokens[0].replace("<!ENTITY", "").trim();
            String desc = lineTokens[1];
            //insert entity to the database
            String queryInsert = "insert into officialtrafficentities" +
            "(Code, Value) values ('" + code + "', \"" + desc + "\")";
            SQLTemplate insertQuery = new SQLTemplate(Officialtrafficentities.class, queryInsert);
            context.performNonSelectingQuery(insertQuery);
        }
    }
}
/**
 * Gets and records the appropriate data from the traffic feeds to the
 * database.
 */
public void recordTrafficFeeds() throws Exception{
    String feedText = getRTDURLSourceCode();
    BufferedReader buf = new BufferedReader(new StringReader(feedText));
    String time = null, date = null, severityCode = null, latitude = null, longitude = null, line = "";
    Boolean feedItemStart = false, severityCodeSet = false;
    while ((line = buf.readLine()) != null){
        //first find <tpeg_message>
        if ((feedItemStart == false) && (line.contains("<tpeg_message"))){
            feedItemStart = true; severityCodeSet = false;
        }
        //get the date/time of the feed
        if (feedItemStart){
            if (line.contains("message_generation_time=\"")){
                String[] tokForDate = line.split("message_generation_time=\"");
                String[] subTokForDate = tokForDate[1].split("\"");
                String dateTime = subTokForDate[0]; String[] dateTimeTokens = dateTime.split("T");
```

```
        date = dateTimeTokens[0]; String[] tokForTime = dateTimeTokens[1].split("\\+"); time = tokForTime[0];
    }
}
//then search for severity code
if ((feedItemStart == true) && (line.contains("severity_factor=\"&"))){
    String[] lineTokens = line.split("severity_factor=\"&"); String[] lineSubTokens = lineTokens[1].split(";");
    severityCode = lineSubTokens[0]; severityCodeSet = true;
}
//proceed to the actual events only if the severity code is set
if (severityCodeSet){
    if (line.contains("latitude=") && (line.contains("longitude="))){
        String[] tokForLat = line.split("latitude=\""); String[] subTokForLat = tokForLat[1].split("\"");
        latitude = subTokForLat[0]; String[] tokForLon = line.split("longitude=\"");
        String[] subTokForLon = tokForLon[1].split("\""); longitude = subTokForLon[0];
    }
}
//a news item has been terminated
if ((feedItemStart == true) && (line.contains("</tpeg_message>"))){
    feedItemStart = false; severityCodeSet = false;
    //spot locations near lat/long
    String queryFindLoc; Double dif = 0.05;
    while (true){
        if (!longitude.startsWith("-")){
            queryFindLoc = "select LocationID from location where " +
            "((Longitude > (" + longitude + "-" + dif + ")) and " +
            "(Longitude < (" + longitude + "+" + dif + "))) and " +
            "((Latitude > (" + latitude + "-" + dif + ")) and " +
            "(Latitude < (" + latitude + "+" + dif + ")))";
        }
        else{
            queryFindLoc = "select LocationID from location where " +
            "((Longitude < (" + longitude + "+" + dif + ")) and " +
            "(Longitude > (" + longitude + "-" + dif + "))) and " +
            "((Latitude > (" + latitude + "-" + dif + ")) and " +
            "(Latitude < (" + latitude + "+" + dif + ")))";
        }
        SQLTemplate findLocQuery =  new SQLTemplate(Location.class, queryFindLoc);
        findLocQuery.setFetchingDataRows(true); List findLocQueryRows = context.performQuery(findLocQuery);
        //if no location ids returned, increase diameter and try again
        if (findLocQueryRows.size() == 0){ dif += (dif/2); if (dif <= 0.15){ continue; }else{ break; } }
        //store all the derived location ids to this arraylist
        ArrayList<Long> locIDs = new ArrayList<Long>();
```

```
            for (int i = 0; i < findLocQueryRows.size(); i++){
                DataRow row = (DataRow) findLocQueryRows.get(i);
                Location location = (Location) context.objectFromDataRow(Location.class, row, true);
                locIDs.add(location.getLocationId());
            }
            //for each one of them execute an insertion query
            for (Long locID : locIDs){
                String queryInsert = "insert into officialtrafficdata" +
                "(LocID, Date, Time, TrafficCondition) values" +
                "(\"" + locID + "\", '" + date + "', '" + time +
                        "', '" + severityCode + "')";
                SQLTemplate insertQuery = new SQLTemplate(Officialtrafficdata.class, queryInsert);
                context.performNonSelectingQuery(insertQuery);
            }
            break;
} } } } }
```

B.3 Weather talk crawler - FollowRSS.java

```
...
/**
 * Retrieves training data for weather talk.
 */
public class FollowRSS{
    private String url; private ExcludedWords exWords;
    private Stemmer stemmer; private DataContext context;
    public FollowRSS(){
        url = "..."; exWords = new ExcludedWords(); stemmer = new Stemmer();
        context = DataContext.createDataContext();
    }
    ...
    /**
     * Implements an unconventional crawler.
     */
    @SuppressWarnings("unchecked")
    public void getArticleLinks(Boolean indexWordsEnabled) throws Exception{
        Scraper scr = new Scraper(); DateTimeProcess dtProcess = new DateTimeProcess();
        ArrayList<String> checkForLocName; ArrayList<Long> inferredLocations = null;
        ArrayList<String> exWordsList = exWords.fillExWords(exWordsList);
```

```
ArrayList<String> exStartersList = exWords.fillExStarters(exStartersList);
RSSFetcher rssFetch = new RSSFetcher(); SyndFeed feed = rssFetch.fetch(url);
String feedEntriesText = feed.getEntries().toString();
BufferedReader reader = new BufferedReader(new StringReader(feedEntriesText));
String line; String articleURL = null, articleText = null; Boolean stopSearchingThisArticle = false;
//read the elements of the RSS feed line by line
while ((line = reader.readLine()) != null){
   if (line.length() > 0){
      if (line.contains("SyndEntryImpl.link=")){
         articleURL = line.split("&url=")[1].split("&cid=")[0];
         //get the article's text using the scraper
         articleText = scr.getText(articleURL); if (articleText == null){ continue; }
         articleText = removeUnwantedChars(articleText);
         if ((articleText == null) || (articleText.equals(""))){ continue; }
         String[] textTokens = articleText.split(" "); checkForLocName = new ArrayList<String>();
         for (int j = 0; j < textTokens.length; j++){
            //if the 1st letter of a word is in upper case
            if (textTokens[j] != null){
               if (!textTokens[j].equals("")){
                  //add the possible location name to the list for searching
                  if (Character.isUpperCase(textTokens[j].charAt(0))){checkForLocName.add(textTokens[j]);}
               }
            }
         }
         //if there are no words starting with a capital letter
         if (checkForLocName.size() <= 0){ stopSearchingThisArticle = true; continue;
         }
         //build the search query for the database (Location table)
         String queryText =
            "select LocationID, FullName, WeatherFeed_URL from location where " +
            "WeatherFeed_URL is not null and (";
         for (int k = 0; k < checkForLocName.size(); k++){
            if (k == checkForLocName.size() - 1){
               queryText += " Fullname = '" + checkForLocName.get(k) + "')";
            }
            else{
               queryText += " Fullname = '" + checkForLocName.get(k) + "' or";
            }
         }
         //execute the search query and fetch the locationIDs
         SQLTemplate selectQuery = new SQLTemplate(Location.class, queryText);
         selectQuery.setFetchingDataRows(true); List queryRows = context.performQuery(selectQuery);
```

```
    //collect all the locations that the article refers to
    inferredLocations = new ArrayList<Long>();
    for (int it = 0; it < queryRows.size(); it ++){
        DataRow row = (DataRow) queryRows.get(it);
        Location location = (Location) context.objectFromDataRow(Location.class, row, true);
        //add an inferred location to the list (for one article)
        inferredLocations.add(location.getLocationId());
    }
    //if no locations were inferred
    if (inferredLocations.size() <= 0){ stopSearchingThisArticle = true; }
}
//if this is the line that contains the date of the article
else if(line.contains("SyndEntryImpl.publishedDate=")){
    //if no location names were retrieved
    if (stopSearchingThisArticle == true){ stopSearchingThisArticle = false; continue; }
    //get the Date / Time of the article
    String[] tokens = line.split("Date="); String[] tokens2 = tokens[1].split(" ");
    String month = tokens2[1]; String dayNum = tokens2[2]; String time = tokens2[3]; String year = tokens2[5];
    String dateForDatabase = dtProcess.formDateBE(dayNum, month, year);
    //enter info to the database
    Boolean index_flag = true;
    for (Long locID : inferredLocations){
        System.out.println(locID);
        //retrieve URL of official weather observation feed for a locID
        String queryText =
            "select WeatherFeed_URL, LocationID, FullName from location where " +
            "LocationID = '" + locID + "'";
        SQLTemplate selectQuery = new SQLTemplate(Location.class, queryText);
        selectQuery.setFetchingDataRows(true); List queryRows = context.performQuery(selectQuery);
        DataRow row = (DataRow) queryRows.get(0);
        Location location = (Location) context.objectFromDataRow(Location.class, row, true);
        //parse contents of official weather observation feed for that location
        String officialWFText = rssFetch.fetch(location.getWeatherFeedUrl()).getEntries().toString();
        String officialWS = null, officialT = null;
        BufferedReader reader2 = new BufferedReader(new StringReader(officialWFText));
        String line2; Boolean foundFlag = false; int iFlag = 0;
        while ((line2 = reader2.readLine()) != null){
            //at the next line appear the contents of the RSS
            if (line2.contains("SyndEntryImpl.description.value=")){foundFlag = true; continue;}
            //we are interested in the contents of 3rd line
            if (foundFlag == true){ iFlag++; }
            if (iFlag == 3){
```

```
                    String[] weatherCond = line2.split(", "); officialWS = weatherCond[0];
                    //this is the temperature
                    String[] temperature = weatherCond[1].split(" C"); officialT = temperature[0];
                    break;
                }
            }
            String qText = removeUnwantedChars(articleText);
            String qWFText = officialWFText.replace("'", " "); qWFText = qWFText.replace("\"","" ");
            //the text of the RSS feed was removed from the insert command
            //in order to save space (OWS_Text) = qWFText = ""
            String insertQueryText =
                "insert into trainingdata(Text, URL, LocID, Date, Time, " +
                "OWS_Text, OWS_State, OWS_Temperature_C) values " +
                "('" + qText + "', \"" + articleURL + "\", '" + locID + "', '" +
                  dateForDatabase  + "', '" + time + "', '', '" + officialWS + "', '" + officialT + "')";
            SQLTemplate insertQuery = new SQLTemplate(Trainingdata.class, insertQueryText);
            context.performNonSelectingQuery(insertQuery);
            if (index_flag == true){
                if (indexWordsEnabled){
                    wordsIndexing(officialWS, qText, articleURL, dateForDatabase, time,
                        officialT, locID, exWordsList, exStartersList);
                }
                index_flag = false;
            }
        }
        continue;
} } } } ... }
```

B.4 A statistical analysis sample - Stats.java

```
...
/**
 * Creates the statistics files used for weather inference.
 */
public class Stats {
    private DataContext context; private DbQueries dbQueries; private HashMapFunctions hashMapFun;
    private ArrayList<String> mapTokens; private HashMap<String, ArrayList<String>> wsMappings;
    //Intepolation parameters
    private static final Double LonUKRange = 17.5416, LatUKRange = 15.566667;
```

```
private Double iFLon = LonUKRange * 0.001, iFLat = LatUKRange * 0.001, iFLonMax = 0.075, iFLatMax = 0.06;
public Stats(){
    context = DataContext.createDataContext(); dbQueries = new DbQueries();
    hashMapFun = new HashMapFunctions(); mapTokens = dbQueries.getMapTokens();
    wsMappings = dbQueries.getWSMappings();
}
...
/**
 * Create the stat file for a location / date. If tfIdfenabled is true then it implements tf−idf.
 */
public HashMap<String, Double[]> getStatsForLocDate(String date, String locationName,
        Boolean tfIdfEnabled) throws Exception{
    HashMap<String, String> outputs = new HashMap<String, String>();
    int articlesWords = 0, numOfWeatherRelatedWords = 0;
    ArrayList<String> weatherWordsFoundList = new ArrayList<String>();
    Set<String> wsDescriptions = wsMappings.keySet();
    HashMap<String, Double> wsDescWordCount = new HashMap<String, Double>();
    //initialize HashMap
    for (String temp: wsDescriptions){ wsDescWordCount.put(temp, 0.0); }
    //get the longitude, latitude for the specified location
    Double[] lonLatOutput = dbQueries.getLonLatFromLocName(locationName);
    if (lonLatOutput == null){ return null; }
    Double longitude = lonLatOutput[0]; Double latitude = lonLatOutput[1];
    // TF − IDF Parameters //
    int numOfDocuments = 0;
    //document id −> wsMappingInDocCount
    HashMap<Integer, HashMap<String, Integer>> docStats = new HashMap<Integer, HashMap<String, Integer>>();
    //document id −> wsMappingInDocWeight
    HashMap<Integer, HashMap<String, Double>> docWeights =
        new HashMap<Integer, HashMap<String, Double>>();
    //name of the wsMapping and number of docs in which it appears
    HashMap<String, Integer> wsMappingNumOfDocs = new HashMap<String, Integer>();
    //initialization
    for (String wsDescription: wsMappings.keySet()){
        for (String wsMapping : wsMappings.get(wsDescription)){ wsMappingNumOfDocs.put(wsMapping, 0); }
    }
    //derive all the locationIDs that are close to this location
    ArrayList<Long> locIDs = new ArrayList<Long>(); List getLocIDsQueryRows = null;
    while (true){
        getLocIDsQueryRows = dbQueries.getLocIdFromLonLat(longitude, iFLon, latitude, iFLat);
        iFLon = iFLon * 1.03; iFLat = iFLat * 1.03;
        //if the limits are reached stop
```

```
    if ((iFLon > iFLonMax) && (iFLat > iFLatMax)){ break; }
}
//add the location IDs to an ArrayList
for (int i = 0; i < getLocIDsQueryRows.size(); i++){
    DataRow row = (DataRow) getLocIDsQueryRows.get(i);
    Location location = (Location) context.objectFromDataRow(Location.class, row, false);
    Long locID = location.getLocationId(); locIDs.add(locID);
}
if (locIDs.size() == 0){ return null; }
//For every location find (if it exists) a traffic record for the
//specified day − count the occ of the records/events
HashMap<String, Integer> OTRCounts = new HashMap<String, Integer>();
OTRCounts = dbQueries.getOTRCounts(locIDs, date);
//For every ID and for the specified date retrieve the training data
ArrayList<String> OWS = new ArrayList<String>();
HashMap<Integer, ArrayList<String>> OWSPer2Hours = new HashMap<Integer, ArrayList<String>>();
//initialise
for (int init = 1; init <= 12; init++){ OWSPer2Hours.put(init, new ArrayList<String>()); }
List getOWSQueryRows = dbQueries.getTDForLocDate(locIDs, date);
//TF−IDF − Set the number of documents
numOfDocuments = getOWSQueryRows.size();
for (int i = 0; i < numOfDocuments; i++){
    DataRow row = (DataRow) getOWSQueryRows.get(i);
    Trainingdata td = (Trainingdata) context.objectFromDataRow(Trainingdata.class, row, true);
    String ows = td.getOwsState(); Date owsTime = td.getTime();
    //get the time frame
    int timeFrame = getTimeFrameNum(owsTime);
    //hashmap time interval id −> ows
    if ((ows != null) && (!ows.equals("Unknown")) && (!ows.isEmpty())){
        OWS.add(ows); OWSPer2Hours.get(timeFrame).add(ows);
    }
    String tdText = td.getText().toUpperCase();
    //get the number of words for the article
    articlesWords += tdText.split(" ").length;
    //name of the wsMapping and the number of appearances in a document
    HashMap<String, Integer> wsMappingInDocCount = new HashMap<String, Integer>();
    for (String wsDescription: wsMappings.keySet()){
        for (String wsMapping : wsMappings.get(wsDescription)){
            //count the number of occurrences of wsMapping in the text
            int temp2 = numOfSubStrings(tdText, wsMapping);
            //counter of weather related words in the text
            numOfWeatherRelatedWords += temp2;
```

```
            if (temp2 > 0){
                //TF−IDF − Store the # of this wsMapping appearances in the doc
                wsMappingInDocCount.put(wsMapping, temp2);
                //TF−IDF − Increase # of documents in which wsMapping appears
                int wsMappingDocs = wsMappingNumOfDocs.get(wsMapping); wsMappingDocs++;
                wsMappingNumOfDocs.put(wsMapping, wsMappingDocs);
                if (!tfIdfEnabled){
                    for (int temp4 = 0; temp4 < temp2; temp4++){
                        weatherWordsFoundList.add(wsMapping); Double temp6 = wsDescWordCount.get(wsDescription);
                        temp6++; wsDescWordCount.put(wsDescription, temp6);
                    }
                }
            }
        }
    }
    //all the doc is parsed − we add its statistics
    docStats.put(i, wsMappingInDocCount);
}
//check the number of weather related words if < 30, terminate
if (numOfWeatherRelatedWords < 30){ return null; }
if (tfIdfEnabled){
    for (Integer docID : docStats.keySet()){
        //find the wsMapping of the document with the most appearances
        if (!(docStats.get(docID).size() > 0)){ continue; }
        int maxAppInDoc = hashMapFun.getMaxHashMapValue(docStats.get(docID));
        //name of the wsMapping and its calculated weight in a document
        HashMap<String, Double> wsMappingInDocWeight = new HashMap<String, Double>();
        for (String wsMapping : docStats.get(docID).keySet()){
            Double tF = (double)((double)(docStats.get(docID).get(wsMapping))) / ((double)(maxAppInDoc));
            Double iDF; Double wsMappingWeightForDoc;
            if ((tF > 0) && (wsMappingNumOfDocs.get(wsMapping) > 0)){
                iDF = Math.log((double)numOfDocuments/wsMappingNumOfDocs.get(wsMapping));
                wsMappingWeightForDoc = tF * iDF;
            }
            else{ wsMappingWeightForDoc = 0.0; } wsMappingInDocWeight.put(wsMapping, wsMappingWeightForDoc);
        }
        docWeights.put(docID, wsMappingInDocWeight);
    }
    //collecting the totals
    HashMap<String, Double> wsMappingsTotalWeights = new HashMap<String, Double>();
    for (Integer p : docWeights.keySet()){
        for (String wsMapping : docWeights.get(p).keySet()){
```

```
                if (!wsMappingsTotalWeights.containsKey(wsMapping)){ wsMappingsTotalWeights.put(wsMapping, 0.0); }
                Double addWeight = docWeights.get(p).get(wsMapping);
                Double currentTotalWeight = wsMappingsTotalWeights.get(wsMapping);
                currentTotalWeight += addWeight; wsMappingsTotalWeights.put(wsMapping, currentTotalWeight);
            }
        }
        for (String wsDescription: wsMappings.keySet()){
            for (String wsMapping : wsMappings.get(wsDescription)){
                if (wsMappingsTotalWeights.keySet().contains(wsMapping)){
                    weatherWordsFoundList.add(wsMapping); Double temp6 = wsDescWordCount.get(wsDescription);
                    temp6 += wsMappingsTotalWeights.get(wsMapping); wsDescWordCount.put(wsDescription, temp6);
  }  }  }  }...}
  /**
   * Counts the number of occurences of a substring in a string.
   */
  public int numOfSubStrings(String string, String subString) throws Exception{
      int count = 0;
      for (int i = 0; i < string.length(); i++){
          if (string.substring(i).length() >= subString.length()){
              if ((string.substring(i, i + subString.length())).equals(subString)){ count++; }
          }
      } return count;
  }
  ...
}
```

B.5 Weather ontology, a sample

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE weatheront SYSTEM "wont.dtd">

<weatheront>
  <entity>
    <wclass>FAIR_SUNNY_CLEAR</wclass>
    <condition>clear</condition>
    <wcmapping id="1">clear weather</wcmapping>
    <wcmapping id="2">blue sky</wcmapping>
    ...
  </entity>
```

```
  <entity>
    <wclass>CLOUDY</wclass>
    <condition>cloudy</condition>
    <wcmapping id="1">cloudy</wcmapping>
    <wcmapping id="2">wet weather</wcmapping>
    ...
  </entity>
  <entity>
    <wclass>RAINY</wclass>
    <condition>drizzle</condition>
    <wcmapping id="1">drizzle</wcmapping>
    <wcmapping id="2">dimpsey</wcmapping>
    ...
  </entity>
  ...
</weatheront>
```

Bibliography

[Bor01] M. Borga. Canonical correlation: a tutorial. *Online Tutorial*, 2001. [http://people.imt.liu.se/~magnus/cca/tutorial/tutorial.pdf].

[Cha03] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*, chapter 2, pages 19–34. Morgan Kaufmann, 2003.

[Die00] R. Diestel. *Graph Theory*, chapter 1, pages 1–26. Springer-Verlag, 2000.

[DLY08] X. Ding, B. Liu, and P.S. Yu. A holistic lexicon-based approach to opinion mining. *Proceedings of the international conference on Web search and web data mining*, pages 231–240, 2008.

[Dur98] R. Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, chapter 8, pages 192–205. Cambridge University Press, 1998.

[FPSM92] W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus. Knowledge Discovery in Databases: An Overview. *AI Magazine*, 13(3):57–70, 1992.

[GIS07] A. Ghose, P. Ipeirotis, and A. Sundararajan. Opinion Mining using Econometrics: A Case Study on Reputation Systems. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 416–423, 2007.

[Gru93] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GS05] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. *International World Wide Web Conference*, pages 902–903, 2005.

[Gua98] N. Guarino. Formal Ontology and Information Systems. *Proceedings of FOIS '98*, pages 3–15, 1998.

[HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*, chapter 6, pages 310–315. Morgan Kaufmann, 2001.

[HL97] DL Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[HL04] M. Hu and B. Liu. Mining and summarizing customer reviews. *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177, 2004.

[Joa96] T. Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. 1996.

[JU99] R. Jasper and M. Uschold. A framework for understanding and classifying ontology applications. *Proceedings 12th Int. Workshop on Knowledge Acquisition, Modelling, and Management KAW*, 99:16–21, 1999.

[Lee97] J.H. Lee. Analyses of multiple evidence combination. *ACM SIGIR Forum*, 31:267–276, 1997.

[Liu07a] B. Liu. *Web data mining: exploring hyperlinks, contents, and usage data*, chapter 1, pages 4–7. Springer, 2007.

[Liu07b] B. Liu. *Web data mining: exploring hyperlinks, contents, and usage data*, chapter 11, pages 411–447. Springer, 2007.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[NM01] N.F. Noy and D.L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, 2001.

[Pea86] J. Pearl. Fusion, Propagation, and Structuring in Belief Networks. *Artificial Intelligence*, 29(3):241–288, 1986.

[PNM08] O. Pourret, P. Naim, and B. Marcot. *Bayesian Networks: A Practical Guide to Applications*, chapter 1, pages 1–13. Wiley, 2008.

[SFW83] G. Salton, E.A. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[SY73] G. Salton and CS Yang. On the Specification of Term Values in Automatic Indexing. 1973.

[VC99] C.C. Vogt and G.W. Cottrell. Fusion Via a Linear Combination of Scores. *Information Retrieval*, 1(3):151–173, 1999.