# B.E. Project
# CROSS DOMAIN RECOMMENDATION SYSTEM
## Using Weather Data and Commercial Sales Data


**Submitted by:**

**SONAALI MITTAL (786/IT/13)**

**SRISHTI JAIN (787/IT/13)**

**SWATI GARG (791/IT/13)**


**Under the guidance of**

**DR. SHAMPA CHAKRAVERTY**
**MS. AMARJEET MALHOTRA**

**DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF ENGINEERING**


**DEPARTMENT OF INFORMATION TECHNOLOGY**

**Netaji Subhas Institute of Technology**

**University of Delhi**


**2016-2017**

# SELF DECLARATION

The project entitled “Cross Domain Recommendation System using Weather Data and Commercial Sales Data” is a bonafide work carried out by **Sonaali Mittal**, **Srishti Jain**, **Swati Garg** in **Department of Information Technology, Netaji Subhas Institute of Technology, Delhi** under the supervision and guidance of **Dr. Shampa Chakraverty** and **Ms. Amarjeet Malhotra** in partial fulfilment of the requirement for the Degree of Bachelor of Engineering in Computer Engineering, University of Delhi for the year 2016-2017.

The content of this report is to the best of our knowledge and hasn’t been used for any other academic activity.

Date:

| **Sonaali Mittal** | **Srishti Jain** | **Swati Garg** |
|---|---|---|
| (786/IT/13) | (787/IT/13) | (791/IT/13) |

नेताजी सुभाष प्रौद्योगिकी संस्थान
NETAJI SUBHAS INSTITUTE OF TECHNOLOGY
(An Institution of Govt. of NCT of Delhi-Formerly, Delhi Institute of Technology)
Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi - 110 078
Telphone : 25099050; Fax : 25099025, 25099022 Website : http://www.nsit.ac.in

# CERTIFICATE

This is to certify that the project entitled "**Cross Domain Recommendation System using Weather Data and Commercial Sales Data**" is a bonafide work done by Ms. Sonaali Mittal, Ms. Srishti Jain and Ms. Swati Garg, students of eighth semester, B.E. Information Technology, Netaji Subhas Institute of Technology, Delhi.

This project work has been prepared as a partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology, University of Delhi, in the academic year 2016-2017.

This work has not been presented for any other academic purpose before.

I wish them luck for all their future endeavours.

Date:

**Dr. Shampa Chakraverty**

**Professor and Head, Department of COE**
**Netaji Subhas Institute of Technology**

**Ms. Amarjeet Malhotra**

**Assistant Professor, Department of IT**
**Netaji Subhas Institute Of Technology**

# ACKNOWLEDGEMENT

We would like to take this opportunity to acknowledge the support of all those without whom the project wouldn't have been possible.

We sincerely thank our mentors, **Dr. Shampa Chakraverty** and **Ms. Amarjeet Malhotra** for their guidance, criticism and encouragement which led to the completion of the project. The regular brainstorming sessions with them gave us a deep insight into the topic and evolved our ideas. We thank them for giving us this opportunity and supporting us throughout.

We would also like to express our gratitude to the **lab assistants in CADLAB** for cooperating with us and providing us with the required resources.

We wish to express heartfelt gratitude to our college, **Netaji Subhas Institute of Technology** for giving us this opportunity for research and development.

Lastly, we would also like to thank our parents who supported us through our thick and thin. Their trust in our capabilities helped us in giving our best.

# INDEX

# LIST OF TABLES

## Chapter 2: Literature Survey



## Chapter 3: Project Design



## Chapter 4: Implementation

# LIST OF FIGURES

## Chapter 1: Introduction



## Chapter 2: Literature Survey



## Chapter 3: Project Design



## Chapter 4: Implementation

Figure 4.4: GITHUB REPOSITORY USED FOR THE PROJECT

# Chapter 5: Observations and Results

Figure 5.1: OBSERVED HMM AND ITS PARAMETERS, EXAMPLE 1

Figure 5.2: OBSERVED HMM AND ITS PARAMETERS, EXAMPLE 2

# ABSTRACT

With the increasing benefits of cashless transactions and ease of online shopping, more and more customers are shifting to e-commerce websites for their shopping needs. These websites create a great experience for customers by not just providing them all kinds of products but also by recommending them the right kind of products. These recommendations seek to be as close as possible to the customer's personal choice and preference. We seek to make a similar kind of recommendation system catering to the personal choices and preferences of customers not by using their history of shopping but by the history of shopping of all the customers put together based on the time of the year and weather conditions.

Several psychological studies have suggested that the weather around us has a profound impact on our mood and subsequent behavioral patterns[1][2]. For example, Seasonal affective disorder (SAD) is a very real kind of depressive disorder (referred to as a depressive disorder with seasonal pattern) wherein a person's major depressive episode is connected to a specific season. These studies reveal that quite a lot of our actions, our moods and our shopping patterns get affected by the weather around us.

We seek to gauge the common changes in moods depending on the weather and the subsequent changes in the shopping patterns that it causes. We do this by using historic shopping trends and analysing them with the weather conditions of the corresponding location and date. Using this analysis, we seek to provide accurate recommendations and check them with actual trends.

# CHAPTER 1: INTRODUCTION

## 1.1 Objective

Weather is one of the most important factors that affect the consumer behavior. It affects almost every part of consumer's life be it clothes, food, travel, shopping and even his mood. The demand for number of products and services is largely shaped by weather conditions. However, these correlations vary according to locality, seasonality, and deviation from seasonal norms [3].

| Products | No Sunlight (Willingness to pay in $) | After Sunlight (Willingness to pay in $) |
|---|---|---|
| Green tea | 3.35 | 4.61 |
| Orange juice | 2.90 | 3.51 |
| Gym membership | 32.89 | 41.67 |
| Airline Ticket | 400.00 | 517.98 |
| Newspaper subscription | 11.41 | 17.79 |

Figure 1.1: THE EFFECT OF WEATHER ON CONSUMER SPENDING [3]

Our objective is to make a recommendation system that correlates trends during the past with the weather during that time period. This correlation is then used to make recommendations to a user when similar kind of weather is encountered in the present. The trends comprise of items belong to a category that topped the chart in that category (top sellers) during the past. The number of items to be picked up from the top can be changed by the user.

## 1.2 Motivation

The project is about building a recommendation system that takes data from multiple domains and makes some recommendation to the user based on it. The recommendations can be movies, music, restaurants, books, generic purchases etc. Currently the recommendation systems exist but they make recommendations based on data from a single domain. For example *MovieLens* makes movie recommendations, *Jester* recommends jokes, *Book-crossings* recommends books and *Last.fm* recommends music.

Our primary focus is on weather as a domain, as we intend on studying the effect of weather on purchases; a previously less explored field.

## 1.3 Organisation of Chapters

In the upcoming chapters, we will provide more details to provide an insight into the work done. Chapter 2 talks about the literature survey conducted before implementing each functionality. In Chapter 3, we explain the detailed design of the recommendation system by giving its software requirement specification and also, the database and methods used for fetching the data. In Chapter 4, we talk about the implementation details of the project. Chapter 5 covers the analysis, observations and the results. Finally, in Chapter 6 we present the limitations, future scope and the conclusion of our work.

# CHAPTER 2 : LITERATURE SURVEY

## 2.1 Cross-Domain Recommendation Systems

Traditional recommender systems are recommendation systems which seek to suggest items belonging to a single domain. For example movies in *Netflix* or songs in *Last.fm*. This method of using a single domain is not perceived as a limitation, but as a focus on a certain market.

Nowadays, users provide feedback for items of different types, for example in Amazon we can rate books, DVDs. Sometimes they express their opinions on different social media and different providers, for example Facebook, Twitter, Amazon, Netflix, TripAdvisor etc. On the other hand providers wish to cross-sell products and services or provide recommendations to new users.
Cross domain recommendation systems leverage all the available personal data provided in distinct domains to generate better recommendations for their users.

A Single-Domain recommendation system treats each domain independently and in the Collective-Domain, we seek to merge domains and treat them as a single domain Whereas in Cross-Domain recommendation system, Knowledge is transferred from source to target with the assumption that information overlap between users and/or items across different domains - overlaps of users, items, attributes etc.

## 2.2 Hidden Markov Models (HMMs)

- A **hidden Markov model** (**HMM**) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (*hidden*) states. An HMM can be presented as the simplest dynamic Bayesian network.
- In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but the output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states. The adjective 'hidden' refers to the state sequence through which the model passes, not to the parameters of the model; the model is still referred to as a 'hidden' Markov model even if these parameters are known exactly.
- HMMs are especially known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging etc.

## 2.2.1 Architecture

1. The random variable $x(t)$ is the hidden state at time $t$. The random variable $y(t)$ is the observation at time $t$. The arrows in the diagram (often called a trellis diagram) denote conditional dependencies.

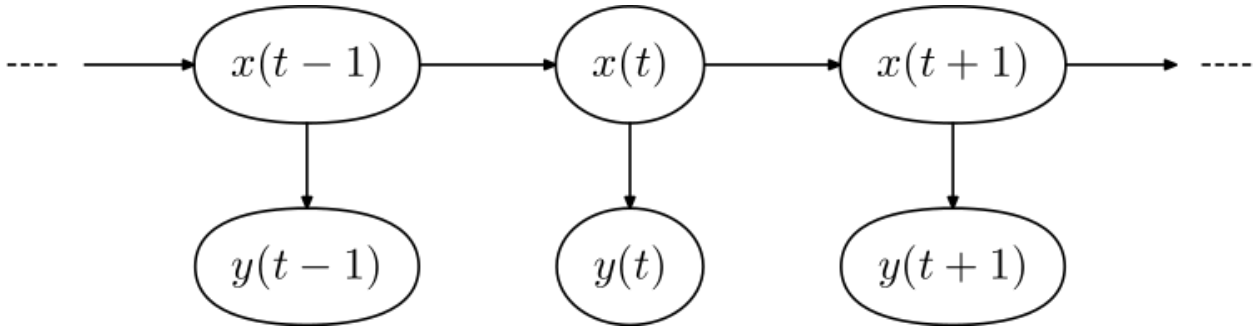



Figure 2.1: TRELLIS DIAGRAM FOR A STANDARD HIDDEN MARKOV MODEL

2. Conditional probability distribution of the hidden variable *x*(*t*) at time *t*, given the values of the hidden variable *x* at all times, depends *only* on the value of the hidden variable *x*(*t* − 1); the values at time *t* − 2 and before have no influence. This is called the *Markov property*. Similarly, the value of the observed variable *y*(*t*) only depends on the value of the hidden variable *x*(*t*) (both at time *t*).
3. The state space of the hidden variables is discrete, while the observations themselves can either be discrete (typically generated from a categorical distribution) or continuous (typically from a Gaussian distribution).
4. The parameters of a hidden Markov model are of two types, transition probabilities and emission probabilities (also known as output probabilities). The *transition probabilities* control the way the hidden state at time *t* is chosen given the hidden state at time *t-1*
5. The hidden state space is assumed to consist of one of N possible values, modeled as a categorical distribution. This means that for each of the N possible states that a hidden variable at time t can be in, there is a transition probability

from this state to each of the N possible states of the hidden variable at time *t+1* , for a total of $N^2$ transition probabilities. Note that the set of transition probabilities for transitions from any given state must sum to 1. Thus, the *N x N* matrix of transition probabilities is a Markov matrix.

6. For each of the N possible states, there is a set of *emission probabilities* governing the distribution of the observed variable at a particular time given the state of the hidden variable at that time. The size of this set depends on the nature of the observed variable.

## 2.2.2 Probability of an Observed Sequence

The task is to compute in a best way, given the parameters of the model, the probability of a particular output sequence. This requires summation over all possible state sequences:

The probability of observing a sequence

$$Y = y(0), y(1), \ldots, y(L-1)$$

of length *L* is given by

$$P(Y) = \sum_{X} P(Y \mid X) P(X)$$

where the sum runs over all possible hidden-node sequences

$$X = x(0), x(1), \ldots, x(L-1).$$

### 2.2.3 Implementation of HMMs

*hmmlearn* implements the Hidden Markov Models (HMMs). The HMM is a generative probabilistic model, in which a sequence of observable X variables is generated by a sequence of internal hidden states Z. The hidden states are not be observed directly. The transitions between hidden states are assumed to have the form of a (first-order) Markov chain. They can be specified by the start probability vector π and a transition probability matrix A. The emission probability of an observable can be any distribution with parameters θ conditioned on the current hidden state. The HMM is completely determined by π, A and θ.

There are three fundamental problems for HMMs:

- Given the model parameters and observed data, estimate the optimal sequence of hidden states.
- Given the model parameters and observed data, calculate the likelihood of the data.
- Given just the observed data, estimate the model parameters.

The first and the second problem can be solved by the dynamic programming algorithms known as the Viterbi algorithm and the Forward-Backward algorithm, respectively. The

last one can be solved by an iterative Expectation-Maximization (EM) algorithm, known as the Baum-Welch algorithm.

### 2.2.4 Example

Alice believes that the weather operates as a discrete Markov chain. There are two states, "Rainy" and "Sunny", but she cannot observe them directly, that is, they are hidden from her. On each day, there is a certain chance that Bob will perform one of the following activities, depending on the weather: "walk", "shop", or "clean". Since Bob tells Alice about his activities, those are the observations. The entire system is that of a hidden Markov model (HMM).

Table 2.1: CODE SNIPPET FOR HMM CREATION FOR THE EXAMPLE

```
states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')

start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

transition_probability = {
   'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
   'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
   }
```

```
emission_probability = {
   'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
   'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
   }
```

In this code, start_probability represents Alice's belief about which state the HMM is in when Bob first calls her (all she knows is that it tends to be rainy on average). The particular probability distribution used here is not the equilibrium one, which is (given the transition probabilities) approximately {'Rainy': 0.57, 'Sunny': 0.43}. The transition_probability represents the change of the weather in the underlying Markov chain. In this example, there is only a 30% chance that tomorrow will be sunny if today is rainy. The emission_probability represents how likely Bob is to perform a certain activity on each day. If it is rainy, there is a 50% chance that he is cleaning his apartment; if it is sunny, there is a 60% chance that he is outside for a walk.

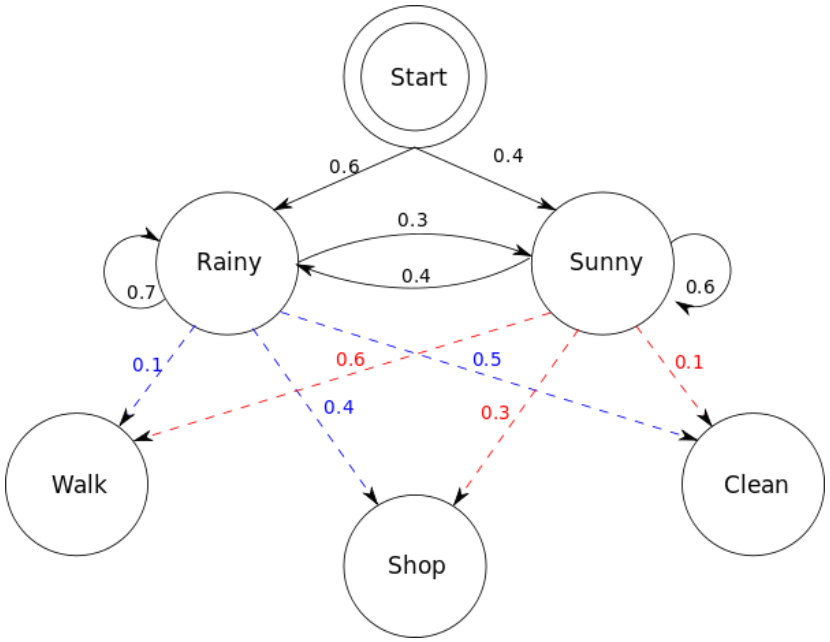



Figure 2.2:TRELLIS DIAGRAM FOR THE EXAMPLE

## 2.3 Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones.

Typically, neurons are connected in layers, and signals travel from the first (input), to the last (output) layer. Modern neural network projects typically have a few thousand to a few million neural units and millions of connections; their computing power is similar to a worm brain, several orders of magnitude simpler than a human brain. The signals and state of artificial neurons are real numbers, typically between 0 and 1. There may be a threshold function or limiting function on each connection and on the unit itself, such that the signal must surpass the limit before propagating. Back propagation is the use of forward simulation to modify connection weights, and is sometimes done to train the network using known correct outputs. However, the success is unpredictable: after training, some systems are good at solving problems while others are not. Training typically requires several thousand cycles of interaction.

### 2.3.1 The MCP neuron

The neuron in Fig 2.3 is the **McCulloch and Pitts model (MCP)**. The inputs are 'weighted', the effect that each input has at decision making is dependent on the weight of the particular input. The weight of an input is a number which when multiplied with the input gives the weighted input. These weighted inputs are then added together and if they exceed a preset threshold value, the neuron fires. In any other case the neuron does not fire.

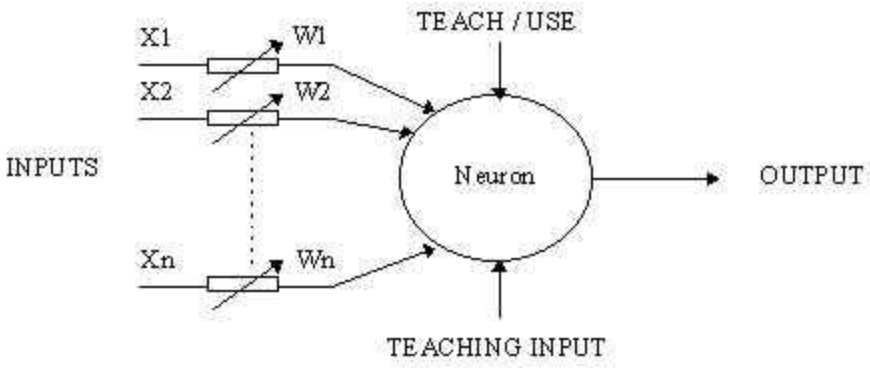



Figure 2.3. An MCP neuron

In mathematical terms, the neuron fires if and only if;

X1W1 + X2W2 + X3W3 + ... > T

The addition of input weights and of the threshold makes this neuron a very flexible and powerful one. The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold. Various algorithms exist that cause the neuron to 'adapt'; the most used ones are the Delta rule and the back error propagation. The former is used in feed-forward networks and the latter in feedback networks.

## 2.3.2 The Back-Propagation Algorithm

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The backpropagation algorithm is the most widely used method for determining the EW.

- Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements(neurons) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.
- Conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.
- Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations. The backpropagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those

weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, we can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives backpropagation its name. Once the EA has been computed for a unit, it is straightforward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection.

### 2.3.3 Neural networks versus conventional computers

- Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve.
- Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements(neurons) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that

because the network finds out how to solve the problem by itself, its operation can be unpredictable.

- On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.
- Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks, require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency.

## 2.4 Python as a scripting language

The following points highlight why Python can be considered as a beneficial replacement for bash scripting:

- Python is installed by default on all the major Linux distributions. Opening a command line and typing python immediately will drop you into a Python interpreter. Python has a very easy to read and understand syntax. Its style

emphasizes minimalism and clean code while allowing the developer to write in a barebones style that suits shell scripting. [4]

- Python is an interpreted language, meaning there is no compile stage. This makes Python an ideal language for scripting. Python also comes with a Read Eval Print Loop, which allows you to try out new code quickly in an interpreted way. This lets the developer tinker with ideas without having to write the full program out into a file.
- Python is a fully featured programming language. Code reuse is simple, because Python modules easily can be imported and used in any Python script. Scripts easily can be extended or built upon. [5]
- Python has access to an excellent standard library and thousands of third party libraries for all sorts of advanced utilities, such as parsers and request libraries. For instance, Python's standard library includes date-time libraries that allow you to parse dates into any format that you specify and compare it to other dates easily.
- Python can be a simple link in the chain. Python should not replace all the bash commands. It is as powerful to write Python programs that behave in a UNIX fashion (that is, read in standard input and write to standard output) as it is to write Python replacements for existing shell commands, such as cat and sort.

## 2.5 Web Scraping

Web scraping (web harvesting or web data extraction) is data scraping used for extracting data from websites. Web scraping software may access the World Wide Web directly using the Hypertext Transfer Protocol, or through a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying, in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when you view the page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Once fetched, then extraction can take place. The content of a page may be parsed, searched, reformatted, its data copied into a spreadsheet, and so on. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. An example would be to find and copy names and phone numbers, or companies and their URLs, to a list (contact scraping).

Web scraping is used for contact scraping, and as a component of applications used for web indexing, web mining and data mining, online price change monitoring and price comparison, product review scraping (to watch the competition), gathering real estate listings, weather data monitoring, website change detection, research, tracking online presence and reputation, web mashup and, web data integration.

## 2.6 10-Fold Cross Validation

In 10-fold cross-validation, the original sample is randomly partitioned into 10 equal sized subsamples. Of the 10 subsamples, a single subsample is retained as the validation data for testing the model, and the remaining 9 subsamples are used as training data. The cross-validation process is then repeated 10 times (the *folds*), with each of the 10 subsamples used exactly once as the validation data. The 10 results from the folds can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used[6], but in general it could be anything and is termed as k-fold cross validation.

## 2.7 Git - An Open Source Version Control System

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. **Git** is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for software development, but it can be used to keep track of changes in any set of files. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows. Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Thus, Git makes collaborative work very convenient and simplified.

Git is implemented in a manner such that it implements the following characteristics:

- **Strong support for non-linear development**

  Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers. In Git, branches are very lightweight: a branch is only a reference to one commit. With its parental commits, the full branch structure can be constructed.

- **Distributed development**

  Git gives each developer a local copy of the full development history, and changes are copied from one such repository to another. These changes are imported as added development branches, and can be merged in the same way as a locally developed branch.

- **Efficient handling of large projects**

  Git has been known to be very fast and scalable, and performance tests done by Mozilla show it is an order of magnitude faster than some version control systems, and fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server.

- **Cryptographic authentication of history**

  The Git history is stored in such a way that the ID of a particular version (a *commit* in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed.

# CHAPTER 3 : PROJECT DESIGN

## 3.1 Introduction

### 3.1.1 Purpose

The purpose of this document is to present a detailed description of the recommendation system. It will explain the purpose and features of the system, the interfaces of the system, the algorithms used in implementation, functionality of the system, and the constraints under which it must operate. The system will be used to recommend products on the basis of the location of the user and weather at that place.

### 3.1.2 Scope

This specification gives the details of the recommendation system. It explains how the retail dataset is obtained and processed. It also explains how the weather data will be assimilated. It highlights the creation of HMMs and choosing them.

### 3.1.3 Definitions, Acronyms, and Abbreviations

| | |
|---|---|
| DTA | Data file (File format) |
| CSV | Comma separated values (File format) |
| DAIRY | Dairy Sales in Dollars |
| DELIEXPR | Deli Express Sales |
| DELISELF | Deli Self Service Sales |

| | |
|---|---|
| FLORAL | Floral Sales |
| FTGCHIN | Food-to-Go Chinese Sales |
| FTGITAL | Food-to-Go Italian Sales |
| GM | General Merchandise Sales |
| HABA | Health and Beauty Aids Sales |
| MEATFROZ | Meat-Frozen Sales |
| PHOTOFIN | Photo |
| SSDELICP | Self Service Deli Sales |
| VIDEOREN | Video Rentals |
| df | A pandas DataFrame: a 2-D labeled data structure with column(s) of one or more types, like a spreadsheet or an SQL table |
| CONVFOOD | Conventional Foods Sales |

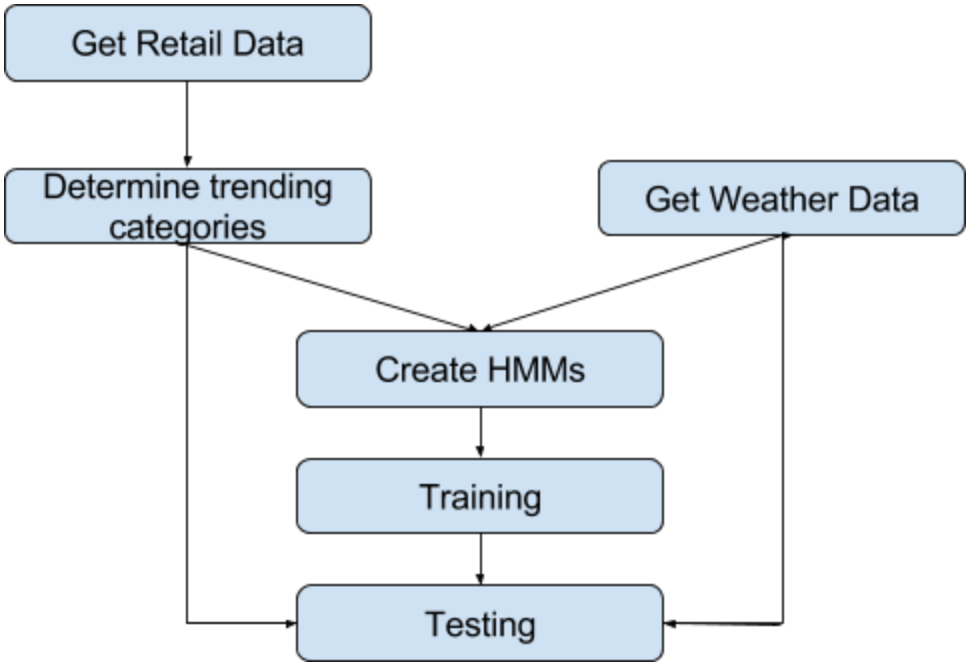
### 3.1.4 Overview

After the theoretical analysis, the sequential execution of goals will broadly be in this order:

- Acquire, filter, process and refine the data

- Create the HMMS
    - For each HMM, the input will be the weather parameters
    - The output of each HMM will be the trending products on that weather pattern
    - There may be several inputs for an HMM, but only one output-therefore, Gaussian HMMs will be used.
- Train the Neural Network for classification
    - The weather data will be fed to the input neurons
    - The output classes for the network will be the various HMMs
- Test the trained classifier

Figure 3.1: MODULAR FLOW OF WORK



## 3.2 Dataset

### 3.2.1 Product Data

We will use a detailed scanner database from Dominick's supermarkets in Chicago to perform the analysis. The Dominick's Finer Foods (DFF) database was obtained from the Kilts Center for Marketing at the University of Chicago, Booth School of Business.[7][10] It contains approximately nine years of store-level data on the sales of more than 3,500 UPCs (Universal Product Codes) distributed in a 100-store chain and includes information on product and consumer characteristics. This data is unique for the breadth of its coverage and for the information available on retail margins.

Table 3.1: ALGORITHM FOR DATASET CLEANUP

1. Import sales and store data as DTA
2. Clean up sales data:
    a. Remove rows with non-numeric values
    b. Drop rows with sales by coupons, 'mvpclub', 'promo'
    c. Ignore negligible rows of 21st century
    d. Convert date from UNIX timestamp to YYYYMMDD
3. Merge sales and store data, based on store ID to get geographical location of each store
4. Output retail data CSV

### 3.2.2 Weather Data

For weather, we will be scraping data for each unique combination of zip code and date from the Web. Farmers' Almanac (www.almanac.com) has historical weather data for each date. Farmer's Almanac seemed like the only available choice as most weather websites or data point don't store weather data that is this old and frequent, or have restrictions over the number of queries in a day.

It gives the following statistics for each query date and zip code:

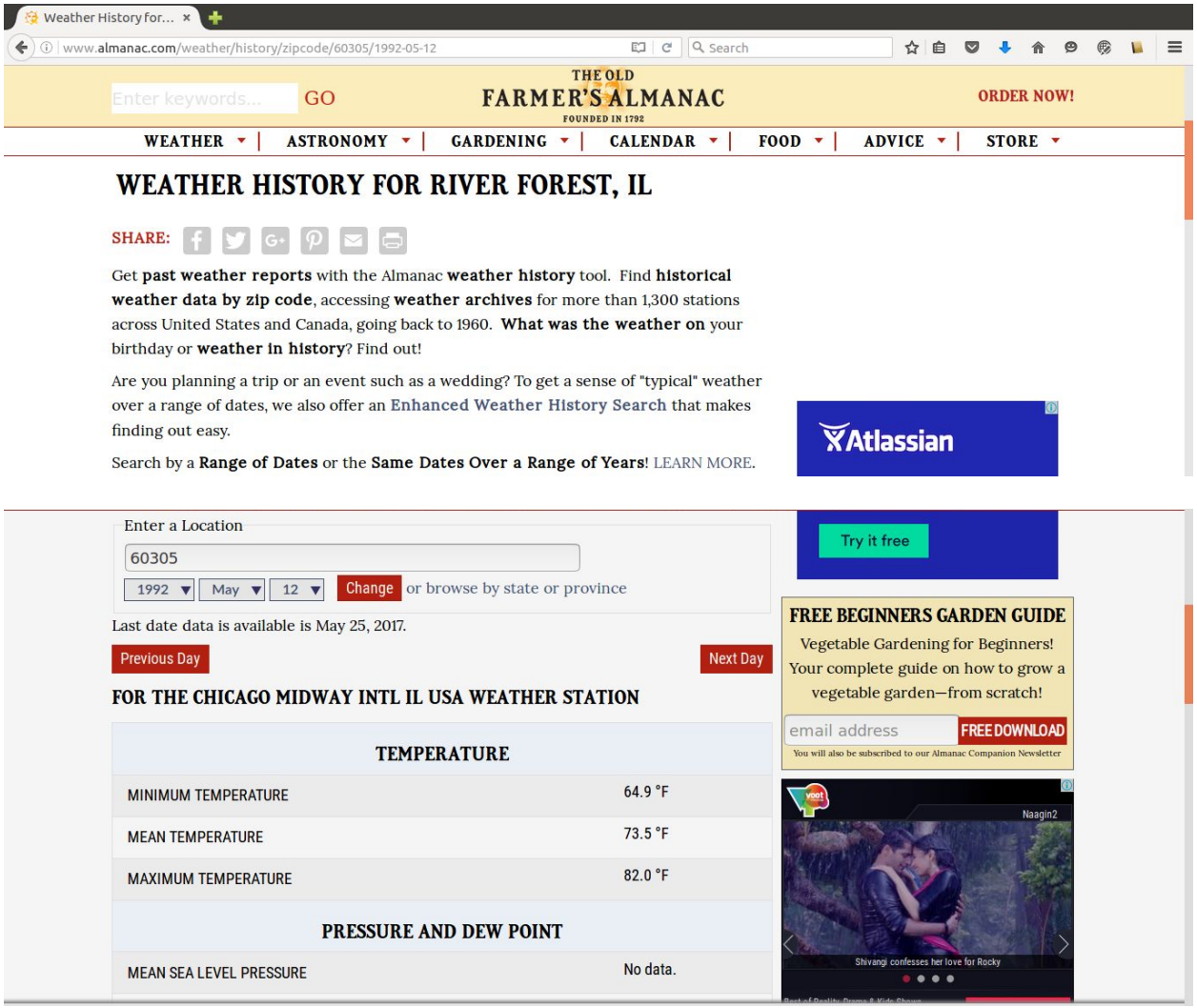
- Minimum Temperature
- Mean Temperature
- Maximum Temperature
- Mean Sea Level Pressure
- Mean Dew Point
- Total Precipitation (Rain and/or melted snow)
- Visibility
- Snow Depth
- Mean Wind Speed
- Maximum Sustained Wind Speed
- Maximum Wind Gust

Table 3.2: ALGORITHM TO FETCH DATA FROM FARMER'S ALMANAC

1. Import retail data CSV
2. Find all unique combinations of date and zip code from the csv as "dates"
3. Import old output data if it exists. Create empty dataframe otherwise.
4. For each tuple x in dates:
    a. Get weather:
        i. Read html data as table from

           http://www.almanac.com/weather/history/zipcode/(x.zip)/(x.date)
        ii. Save the second row only and transpose it
        iii. Insert x in the first column of the table
        iv. Rename column indices
    b. Skip if weather not available
    c. Append data to weather data's table
    d. Write onto CSV and print time elapsed for every 10th tuple
5. Set the column names for weather data

Figure 3.5: THE WEATHER DATA ON *ALMANAC.COM* FOR ZIP 60305 ON 12 MAY, 1992



| PRESSURE AND DEW POINT | |
|---|---|
| MEAN SEA LEVEL PRESSURE | No data. |
| MEAN DEW POINT | 63.1 °F |
| **PRECIPITATION** | |
| TOTAL PRECIPITATION<br>Rain and/or melted snow reported during the day. | No data. |
| VISIBILITY | 10.5 MI |
| SNOW DEPTH<br>Last report for the day if reported more than once. | No data. |
| **WIND SPEED AND GUSTS** | |
| MEAN WIND SPEED | 12.54 MPH |
| MAXIMUM SUSTAINED WIND SPEED | 17.26 MPH |
| MAXIMUM WIND GUST | 25.32 MPH |



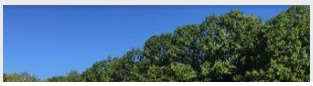
**LIVE WEBCAMS**

**VISIT THE ALMANAC HQ WEBCAM**


| WIND SPEED AND GUSTS | |
|---|---|
| MEAN WIND SPEED | 12.54 MPH |
| MAXIMUM SUSTAINED WIND SPEED | 17.26 MPH |
| MAXIMUM WIND GUST | 25.32 MPH |


**VISIT THE ALMANAC HQ WEBCAM**

**VISIT THE ALMANAC GARDEN WEBCAM**



Weather data from the National Climatic Data Center Global Surface Summary of Day. Information from the NCDC may be incomplete. Not every station reports every day, and some stations never report certain values. To learn more about weather station terminology, please consult the Weather Observation Station page of the NCDC.

Are you planning a trip or an event such as a wedding? To get a sense of "typical" weather over a range of dates, we also offer an **Enhanced Weather History Search** that makes finding out easy.

Search by a **Range of Dates** or the **Same Dates Over a Range of Years**! LEARN MORE.

Previous Day Next Day

Table 3.3: ALGORITHM TO CREATE HMMs FROM FETCHED DATA

1. Create a folder saveHMM.
2. Create new dataframe for HMM records.
3. Import weather data and trending data CSV
4. Remove testing data rows from trending data
5. Find all group of keys(Zip/Date) with same 'MaxByNormalizedQty' values. Name this dataframe as grouped data. It contains the value of MaxByNormalizedQty and corresponding list of keys having the same MaxByNormalizedQty values.
6. For each tuple x in grouped data:
    a. Get list of keys in this tuple x.
    b. Find length of keys.
    c. Divide length in multiples of 100, name it as m.
    d. For idx in 1 to m:
        i. Get list of keys in $idx^{th}$ slot, name it as 'keys'
        ii. Create new dataframe. Get the weather data from the CSV corresponding to 'keys'
        iii. Concatenate and stack the weather data as a matrix (X) with row as weather values for different key.
        iv. Create an HMM model with X as observed states
        v. Save HMM to a folder saveHMM
        vi. Append HMM data to HMM records dataframe
        vii. For every 100th HMM, display the time elapsed

6. Save HMM records file.

Table 3.4: ALGORITHM TO PREDICT TRENDING DATA

1. Load HMMs and trending data values from saveHMM folder and HMM records file respectively.
2. Create new dataframe for predicted data.
3. Load trending data CSV. Get the value of keys for testing rows.
4. Load weather data CSV.
5. For each tuple x in testing data:
    a. Get value of key for tuple x
    b. Fetch weather data for the key from weather data CSV
    c. Pack the weather values in the form of matrix
    d. Find the HMM which gives maximum score for given weather sequence
    e. Append the data of corresponding HMM to the dataframe for predicted data.
    f. For every 100th record, display the time elapsed
7. Save predicted data records file.

# CHAPTER 4: IMPLEMENTATION

## 4.1 Programming Languages and Tools

- Programming Language - Python
    - Libraries
        - Sklearn

          Tools for machine learning, data mining and data analysis
        - Hmmlearn

          Contains a set of algorithms for unsupervised learning and inference of Hidden Markov Models.
        - Beautiful Soup, HTML5Lib

          For parsing / pulling data from HTML and XML files
    - Packages
        - Pandas

          Provides data structures designed to make working with relational / labeled data fast and easy
        - Numpy

          For scientific computing with Python
        - Datetime

          For manipulating dates and times with Python
        - OS

          Provides functions to read, write, open and create files

- Code Editor - Gedit, Geany
- Code Execution - Command Line Interface, using Python's distribution for Linux

## 4.2 Components

Broadly, there are four components:

1. Date Acquisition and reorganisation
2. Creating HMMS
3. Prediction using HMMs
4. Calculating the accuracy of prediction

### 4.2.1 Data Acquisition and Reorganisation

#### 4.2.1.1 Product Data

The Dominick's Finer Foods (DFF) database obtained from the Kilts Center for Marketing at the University of Chicago, Booth School of Business contains approximately nine years of store-level data on the sales of more than 3,500 UPCs (Universal Product Codes) distributed in a 100-store chain and includes information on product and consumer characteristics. This data is unique for the breadth of its coverage and for the information available on retail margins.

- **ccount.dta**

  This contains information about in-store traffic. The data is store specific and on a daily basis. The data refers to a total dollar sales and total coupons redeemed

figure, by DFF defined department. These figures are compiled daily from the register/scanner receipts.

The initial file has 327045 rows of data. After eliminating rows with any non-numeric values, and those with improper date, 266872 entries remain. Also, the columns with sales by coupons were eliminated since they are not very weather dependent -- sale season does not coincide with actual seasons for such product data.

Table 4.1: CODE SNIPPET FOR DATA CLEANUP

```
df = pd.read_stata('ccount.dta')
df = df.apply(pd.to_numeric, errors = 'coerce').dropna()
df = df.drop(df.columns.str.contains('coup').split() + ['mvpclub', 'promo'],
axis=1)
df = df[df['date'] > 500000]
df['date'] = pd.to_datetime(df['date'] + 19000000, format='%Y%m%d')
```

Table 4.2: PRODUCT CATEGORIES IN RETAIL DATA AFTER CLEANUP

| | | | | | |
|---|---|---|---|---|---|
| grocery | dairy | frozen | bottle | meat | meatfroz |
| ssdelicp | ftgital | ftgchin | custcoun | miscscp | bulk |
| spirits | wine | beer | photofin | camera | haba |
| videoren | cosmetic | jewelry | gm | convfood | saladbar |
| video | pharmacy | bakery | cheese | floral | deli |

| | | | | | |
|---|---|---|---|---|---|
| deliself | deliexpr | produce | fish | | |

- **demo.dta**

  This file contains the volume movement of products. We have used this to determine map the sales data to the location of its store, based on the store's unique ID.

Table 4.3: CODE SNIPPET FOR ADDING LOCATION DATA

```
df2 = pd.read_stata('demo.dta')
df = df.merge(df2[['store', 'city', 'zip', 'lat', 'long']], on=['store']).dropna()
df.to_csv('retailData.csv')
```

### 4.2.1.2 Weather Data

Farmers' Almanac (www.almanac.com) has historical weather data for each date, comprising of:

1. **Temperature** - Minimum, Mean, Maximum
2. **Pressure and Dew Point** - Mean Sea Level Pressure, Mean Dew Point
3. **Precipitation** - Total Precipitation (Rain and/or melted snow reported during the day)
4. **Visibility**
5. **Snow Depth** (Last report for the day if reported)

6. **Wind Speed and Gusts** - Mean Wind Speed, Maximum Sustained Wind Speed, Maximum Wind Gust

There were 235959 unique Zip Code-Date combinations in total, of which we could find and fetch data for 201285 combinations from Farmer's Almanac. There were several zip codes and dates for which weather data was not available.

Table 4.4: CODE SNIPPET FOR DETERMINING UNIQUE COMBINATION OF ZIP AND DATES

```
df = pandas.read_csv('retailData.csv')
dates = df.drop_duplicates(subset=['zip', 'date']).filter(['zip', 'date'])
```

Table 4.5: CODE SNIPPET FOR SCRAPING HISTORICAL WEATHER DATA FROM ALMANAC

```
for counter, date in dates.iterrows():
        site = "http://www.almanac.com/weather/history/zipcode/" +
                str(date['zip']) + "/" + str(date['date'])
        weather = pd.read_html(io=site, encoding='utf-8')
        weather = weather[1].T
        weatherTable = weatherTable.append(weather.iloc[1])
```

After cleanup and removal of unnecessary columns, the collected weather data was stored in a CSV (Comma Separated Values) file with UTF-8 encoding.

- All the retail data for which weather data was unavailable on almanac.com was removed. After this, only 232190 of 266872 entries remained.

Table 4.6: CODE SNIPPET FOR REMOVING DATA FOR WHICH WEATHER DATA IS UNAVAILABLE

```
df1 = pd.read_csv('CompiledWeather.csv', usecols=['Zip/Date'])
df2 = pd.read_csv('retailData.csv')
df2['Zip/Date'] = df2['zip'].apply(int).apply(str) + "/" + df2['date'].apply(str)
df2 = df2.merge(df1, on='Zip/Date') #inner join
df2.drop(['Zip/Date'], axis=1, inplace=True)
```

## 4.2.1.3 Determining Trending Data

From the filtered retail data, we determined the most popular *n* categories, on a daily basis. To compute the popularity, we used four methods:

1. **By Quantity**

   Simply filter out the top n categories by *quantitative sales* ie. the categories which had the maximum sales in day.

2. **By Normalised Quantity**

Normalise the sales quantities by dividing each value by the total or mean sales of its category. This ensures that more *variations* in sales are highlighted and categories like bread, groceries and other staples, with regular sales, do not hog the most trending data.

3. **By Maximum Increase**

   Determine the top n categories having most *increase in quantitative sales* from the previous day's data for that category, for the same location (Zip Code)

4. **By Maximum Percentage Increase**

   Determine the top n categories having most *percentage increase* in quantitative sales from the previous day's data for that category, for the same location (Zip Code)

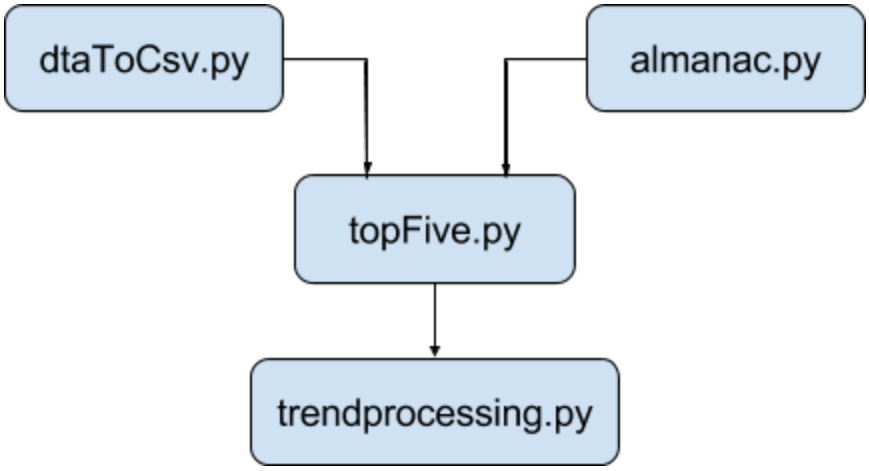
We chose to use normalised quantity to finally determine the trending categories, as it gave more importance to variations and had a moderate amount of sensitivity to change.

Table 4.7: CODE SNIPPET TO DETERMINE TOP *N* TRENDING CATEGORIES

```
def topN(row):
        row = row.sort_values(ascending=False)[:n] #sort and choose first n values
        return str(row.index.values)
df = pd.read_csv('retailData.csv')
df.set_index(['date', 'zip']) #sort by date and zip
df = df.drop(['mmid', 'store', 'city', 'zip', 'lat', 'long', 'date', 'week', 'custcoun'], axis=1)
```

```
df1 = df[['date', 'zip']]

df1['MaxByQty'] = df.apply(topN, axis=1)

df1['MaxByNormalizedQty'] = df.div(df.sum(axis=1), axis=0).apply(topN, axis=1)

df1['ByMaxIncrease'] = df.diff().reset_index(drop=True).apply(topN, axis=1)

df1['ByMaxPercentIncrease'] = df.pct_change().apply(topN, axis=1)

df1.to_csv('top'+str(n)+'trending.csv', index=False)
```

Figure 4.1 : SOURCE CODE FILES USED FOR DATA ACQUISITION AND PROCESSING




## 4.2.2 Creating HMMs

The data was divided into training and testing data as 9:1 ratio, as mentioned in 10 fold cross validation. The training data (222300 rows) were grouped according to their 'MaxByNormalizedQty' value. There were around 780 unique groups. For each group, a new HMM was created. For groups with more than 100 weather data rows, they were split into multiples of 100.

Table 4.8: CODE SNIPPET FOR CREATING HMMs

```
HMMno = 0
TRAINING_SLOT = 1 #this was done for 1 to 10
VALUES_IN_EACH_SLOT = 24700
START_OF_TESTING_SLOT = VALUES_IN_EACH_SLOT * (TRAINING_SLOT - 1)
df = pd.DataFrame.from_csv('top5trending.csv', index_col=None)
dfWeather = pd.DataFrame.from_csv('CompiledWeather.csv', index_col=None)
df['key'] = df['zip'].apply(int).apply(str) + "/" + df['date'].apply(str)
dfWeather['key'] = dfWeather['zip'].apply(int).apply(str) + "/" +
dfWeather['date'].apply(str)

df = df[df.duplicated(subset=['MaxByNormalizedQty']) |
df.duplicated(subset=['MaxByNormalizedQty'], keep='last')] #remove unique entries
#create df with list of dates having same trending products
df2 = df.groupby('MaxByNormalizedQty')['key'].apply(list).to_frame().reset_index()
```

```
#New data frame to save data about HMMs.
columns = ["hmmno", "key", "MaxByNormalizedQty"]
dfHmm = pd.DataFrame(index=None, columns=columns)


for oindex,row in islice(df2.iterrows(), 0, None):
    keys = row['key']
    trendingItem = row['MaxByNormalizedQty']
    m = len(keys)/100
    if len(keys)%100 != 0:
        m = m+1
    start = 0
    for ind in range(0,m):
        if (len(keys) < start+100):
            end = len(keys)
        else:
            end = start+100
        key = keys[start:end]
        start = end+1
        #Get weather data for particular ‘Zip/Date’
        df3 = dfWeather.loc[dfWeather['key']==key[0]]
        for k in range(1,len(key)):
```

```
            df3 = pd.concat([df3, dfWeather.loc[dfWeather['key']==key[k]]])
        for index,d in df3.iterrows():
            # Pack data for training.
            Col = [‘MinTemp’, ‘MeanTemp’, ‘MaxTemp’, ‘Mean Sea Level Pressure’, ‘Mean
Dew Point’, ‘Total Ppt’, ‘Visibility’, ‘Snow Depth’, ‘Mean Wind Speed’, ’Max Sustained
Wind Speed’,’Max Wind Gust’]
            X = np.column_stack([[i] for i in Col])

            # Make an HMM instance and execute fit
            model = GaussianHMM(n_components=len(X), covariance_type="diag",
n_iter=1000).fit(X)
            #Save HMM to a folder saveHMM
            joblib.dump(model, "saveHMM/hmm" + str(HMMno)+ ".pkl")
            newItem = {"hmmno": HMMno, "hmm": model, "key":key,
"MaxByNormalizedQty":trendingItem}
            HMMs.append(newItem)
            HMMno = HMMno + 1

#Save information regarding HMMs
hmmno,key,mx = [],[],[]
for i in HMMs:
    hmmno.append(i["hmmno"])
```

```
    key.append(i["key"])
    mx.append(i["MaxByNormalizedQty"])
dfHmm["hmmno"] = hmmno
dfHmm["key"] = key
dfHmm["MaxByNormalizedQty"] = mx


dfHmm.to_csv(HMMfile, encoding="utf-8", header=True, index=False)
```

### 4.2.3 Predicting data using HMMs

The trending data for testing rows (Rows from 222301 to 247000, when TRAINING_SLOT is 10) is predicted using the models we created in the above step. A prediction score is calculated using the score() method of hmm, which calculates the log probability of a sequence under the model. The model with maximum score is selected, and the corresponding trending items are predicted for the given sequence.

Table 4.9: CODE SNIPPET FOR PREDICTING THE TRENDING DATA

```
HMMs = []
def loadHMMs():
        df = pd.DataFrame.from_csv('hmmRecords.csv', index_col=None)
        for index,row in islice(df.iterrows(), 0, None):
                model = joblib.load("saveHMM/hmm" + str(row["hmmno"]) + ".pkl")
```

```
                newItem = {"hmmno": row["hmmno"], "hmm": model, "key":row["key"],
"MaxByNormalizedQty":row["MaxByNormalizedQty"]}
                HMMs.append(newItem)


loadHMMs()
columns = ['key', 'MaxByNormalizedQty', 'score', 'trendKey']
key=[]
MaxByNormalizedQty=[]
trendKey = []
score = []


df_ = pd.DataFrame(index=None, columns=columns)
for index, d in islice(dfWeather.iterrows(),START_OF_TESTING_SLOT, None):
    Col = [‘MinTemp’, ‘MeanTemp’, ‘MaxTemp’, ‘Mean Sea Level Pressure’, ‘Mean Dew
Point’, ‘Total Ppt’, ‘Visibility’, ‘Snow Depth’, ‘Mean Wind Speed’, ’Max Sustained Wind
Speed’,’Max Wind Gust’]
    X = np.column_stack([[i] for i in Col])
    maxPredictionScore = HMMs[0]["hmm"].score(X)
    hmmNo = 0
    for i in range(1,len(HMMs)):
        predictionScore = HMMs[i]["hmm"].score(X)
        if predictionScore > maxPredictionScore:
```

```
            maxPredictionScore = predictionScore
            hmmNo = i
        key.append(d['key'])
        trendKey.append(HMMs[hmmNo]['key'])
        MaxByNormalizedQty.append(HMMs[hmmNo]['MaxByNormalizedQty'])
        score.append(maxPredictionScore)
df_['key'] = key
df_['MaxByNormalizedQty'] = MaxByNormalizedQty
df_['score'] = score
df_['trendKey'] = trendKey


df_.to_csv('predictedData.csv', encoding="utf-8", header=True, index=False)
```

## 4.2.4 Calculating Accuracy of method

The accuracy of method is calculated using **10 fold cross validation**. The mean for each fold is calculated using this script:

Table 4.10: CODE SNIPPET TO DETERMINE TOP *N* TRENDING CATEGORIES

```
df1 = pd.read_csv('top5trending.csv', index_col=None)
df2 = pd.read_csv('predictedData.csv', index_col=None)
df1['key'] = df1['zip'].apply(int).apply(str) + "/" + df1['date'].apply(str)
def match(x):
```

```
    list1 = str(x['MaxByNormalizedQty_x']).split(' ')
    list2 = str(x['MaxByNormalizedQty_y']).split(' ')
    return len(set(list1) & set(list2))

df3 = df1.merge(df2, on='key') #inner join
df3['match'] = df3.apply(match, axis=1)
print df3[['match']].mean()
df3.to_csv("sample.csv", header=True, index=False)
```

The above script compares the elements of trending Data in top5trending.csv and predictedData.csv. df3[['match']].mean() prints the mean of the accuracy of method.

Figure 4.2 : SOURCE CODE FILES USED FOR CREATING THE DATASET, TRAINING, TESTING AND EVALUATION

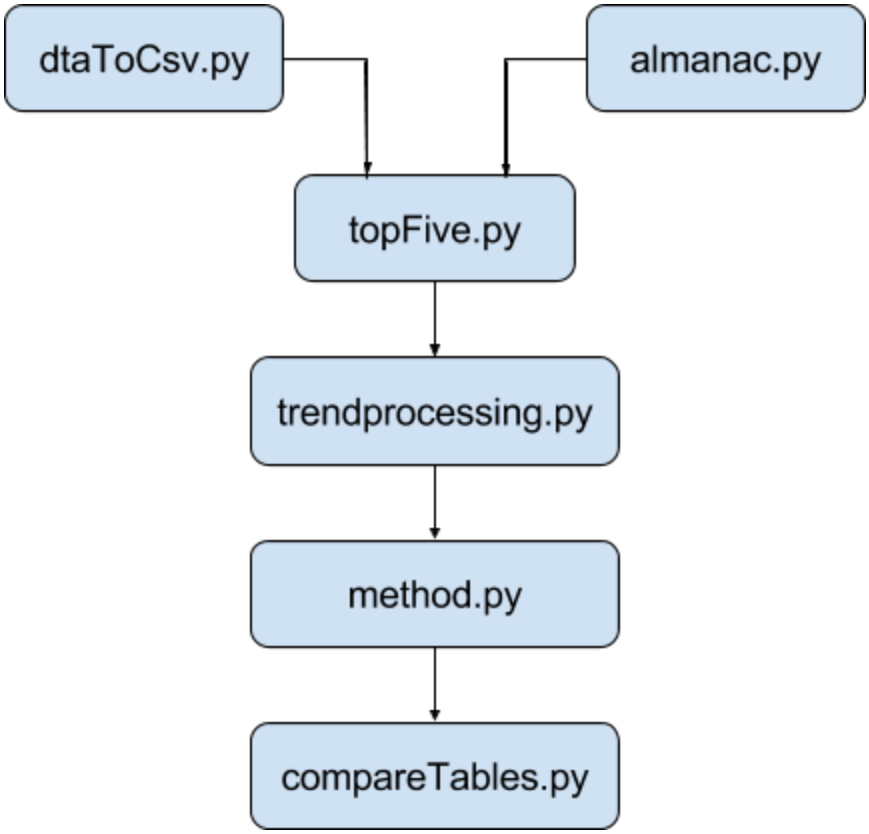
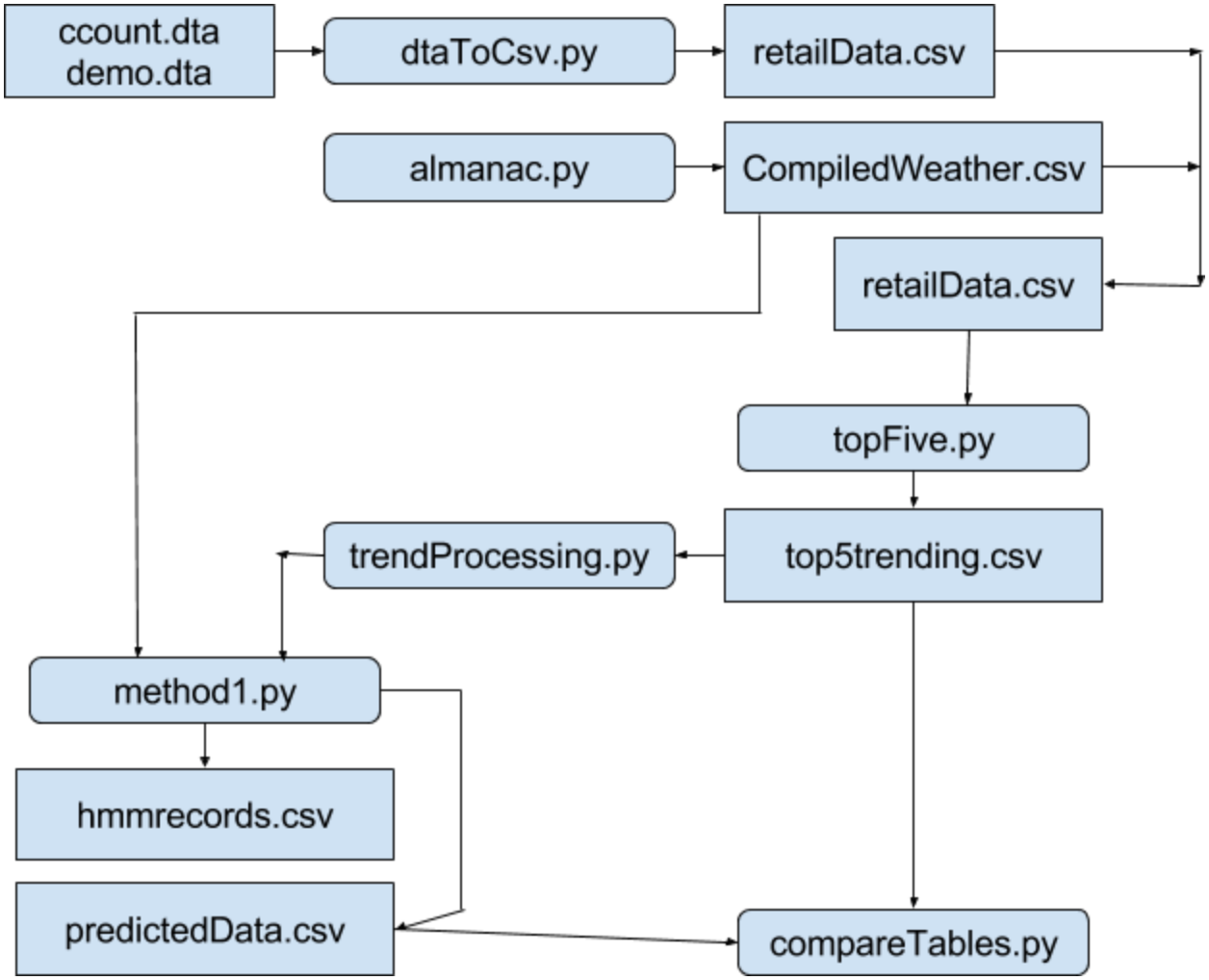



Figure 4.3: FLOW OF DATA FILES ACROSS SOURCE CODE FILES




### 4.2.5 Using Git

We maintained a private git repository on github for working collaboratively.

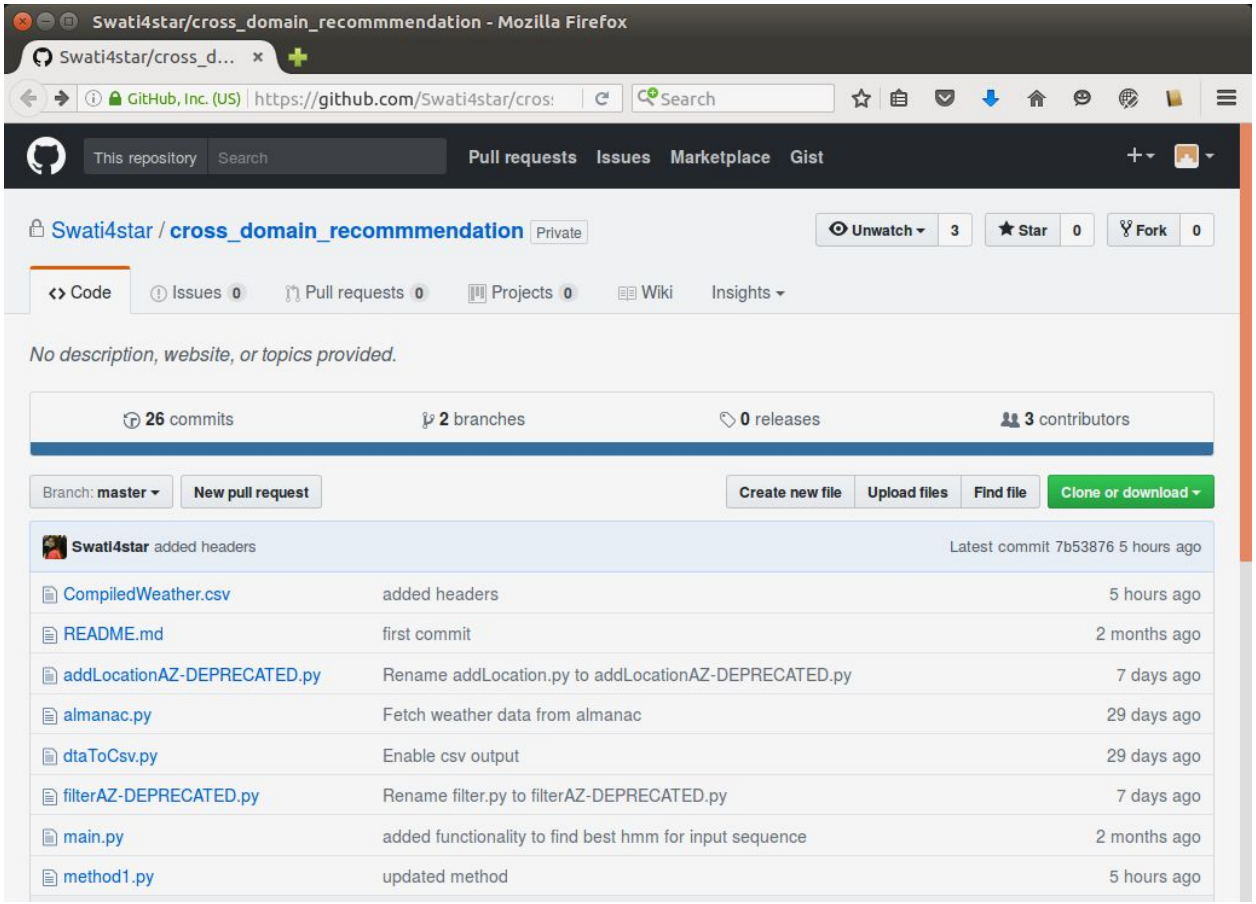



Figure 4.4: GITHUB REPOSITORY USED FOR THE PROJECT

The codebase is available at

*https://github.com/Swati4star/cross_domain_recommmendation* (shown in Figure 4.21).

# CHAPTER 5: OBSERVATIONS AND RESULTS

## 5.1 Observed HMM parameters

Figure 5.1: OBSERVED HMM AND ITS PARAMETERS, EXAMPLE 1

```
Row number :  14 Keys: 13
[[ 21.9   26.    27.9    0.     0.     7.3    0.    10.59  13.58   0.  ]
 [ 15.1   25.3   52.    29.83  20.8    7.9    0.    17.26  21.86  31.07]
 [ 14.9   29.5   37.    30.1   19.4    7.1    0.     6.67  13.81  19.56]
 [ 20.1   28.1   37.     0.    19.7   21.6    0.    10.36  17.26   0.  ]
 [ 24.1   33.3   35.1   29.94  30.8    3.2    1.2    6.79  12.77   0.  ]
 [ -2.9    8.7   18.    30.53  -1.5   12.1    0.4   15.42  21.86  33.37]
 [ 18.    25.8   35.1   30.27  14.5   10.6    0.    13.23  23.02  27.62]
 [  3.9   11.6   21.9    0.    -0.7   21.2    0.    13.81  18.3   26.47]
 [ 18.9   22.    25.     0.    17.6    2.9    0.     7.94  17.26   0.  ]
 [ 38.8   41.    43.9    0.    32.9   15.4    0.    12.66  17.26  21.86]
 [ 32.    39.    47.8    0.    34.6    8.4    0.    14.85  20.6    0.  ]
 [ 32.    39.    47.8    0.    34.6    8.4    0.    14.85  20.6    0.  ]
 [ 27.    32.4   43.     0.    18.9   15.8    0.     8.4   13.58   0.  ]
 [ 27.    32.4   43.     0.    18.9   15.8    0.     8.4   13.58   0.  ]
 [ 19.4   28.7   39.2    0.    21.4    9.4    0.    10.13  17.26   0.  ]]
Transition matrix
[[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]

Means and vars of each hidden state
0th hidden state
mean =  [ 18.    25.8   35.1   30.27  14.5   10.6    0.    13.23  23.02  27.62]
var =  [ 0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01]

1th hidden state
mean =  [ 18.9   22.    25.     0.    17.6    2.9    0.     7.94  17.26   0.  ]
var =  [ 0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01]

2th hidden state
mean =  [ 20.1   28.1   37.     0.    19.7   21.6    0.    10.36  17.26   0.  ]
var =  [ 0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01]

3th hidden state
```

Figure 5.2: OBSERVED HMM AND ITS PARAMETERS, EXAMPLE 2

```
Row number :  13 Keys: 15
[[-11.     4.1   12.9   30.43  -3.8   10.     1.2   14.73  18.3   23.02]
 [ 21.    26.8   37.9   30.42  20.4    7.9    1.2    7.6   11.39   0.  ]
 [ 37.9   39.3   46.    30.27  35.8    7.1    0.    16.69  21.86  25.32]
 [ 37.9   39.5   45.    30.28  30.4    6.4    0.    10.47  16.11   0.  ]
 [ 37.9   39.3   46.    30.27  35.8    7.1    0.    16.69  21.86  25.32]
 [ 19.    30.1   36.    30.06  18.    14.9    0.4   10.93  16.11   0.  ]
 [ 21.9   26.3   30.    30.13  19.9    9.1    0.4   12.66  17.26  21.86]
 [  0.1    6.2   14.2    0.     0.8   15.7    0.    12.31  17.26   0.  ]
 [ 36.9   38.2   39.9    0.    36.7    5.8    0.    15.08  19.45  25.32]
 [ 12.2   16.6   19.4    0.     8.9    6.7    0.    18.07  23.02  28.77]
 [ 37.9   40.4   41.9    0.     0.     6.7    0.    14.15  17.26  28.77]
 [ 27.9   31.6   37.9    0.     0.    16.3    0.    14.5   18.3   27.62]
 [ 32.    39.    47.8    0.    34.6    8.4    0.    14.85  20.6    0.  ]
 [  2.1    7.7   15.3    0.    -3.7   19.3    0.    14.96  23.02   0.  ]
 [ 10.    21.5   32.    30.17  13.6   11.     2.    11.85  17.26   0.  ]]
Transition matrix
[[ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]

Means and vars of each hidden state
0th hidden state
mean =  [ 37.9   39.3   46.    30.27  35.8    7.1    0.    16.69  21.86  25.32]
var =  [ 0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01]

1th hidden state
```

When weather data parameters are:

```
X = [[60.1 68.9  79.  0.     63.2 4.8   0. 8.4  12.77  0.  ]
     [52.  57.6  61.  0.     42.8 17.9  0. 8.4  17.26  0.  ]
     [28.  40.4  48.9 30.12  34.3 10.1  0. 8.86 17.26  28.77]]
```

This represents data for 3 different days, that correspond to same trending data.

The HMM parameters for above sample are :

Transition matrix

```
[[ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]]
```

**Means and variances** of each hidden state

**1. 0th hidden state**

```
mean =  [28.  40.4  48.9  30.12  34.3  10.1  0.  8.86  17.26
28.77]
var =  [0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01
0.01]
```

**2. 1th hidden state**

```
mean =  [ 60.1   68.9   79.     0.    63.2    4.8    0.     8.4
12.77   0.  ]
var =  [ 0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01
0.01]
```

**3. 2th hidden state**

```
mean =  [ 52.    57.6   61.     0.    42.8   17.9    0.     8.4
17.26   0.  ]
var =  [ 0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01
0.01]
```

## 5.2 10-fold cross validation results

| S.No | Test Data Training Slot | Accuracy |
|---|---|---|
| 1. | 10th Subsample | 66.751926% |
| 2. | 9th Subsample | 68.71178% |
| 3. | 8th Subsample | 66.04458% |
| 4. | 7th Subsample | 66.43724% |
| 5. | 6th Subsample | 59.32842% |
| 6. | 5th Subsample | 60.79738% |
| 7. | 4th Subsample | 61.58264% |
| 8. | 3rd Subsample | 59.24696% |
| 9. | 2nd Subsample | 68.58624% |
| 10. | 1st Subsample | 62.29150% |
| Mean | | 63.98178% |

# CHAPTER 6: CONCLUSION AND FUTURE WORK

This chapter discusses the limitations and future scope of the project and finally summarises the conclusion.

## 6.1 Conclusion

Thus, we were able to implement a recommendation system which has multiple domains and uses commercial retail / products data and weather data.

We were also able to classify HMMs with a decent accuracy.

## 6.2 Limitation

- The domains for training can be expanded to include newer methods of mining data.
- The same recommendations are made for people in the same, broad location. This can be further fine-tuned by incorporating data from more domains.
- The current dataset has location specific categories from a chain of stores that is no longer in business. More sources of recent data may be helpful.
- Only one normalisation technique could be experimented with due to constraint in resources and available tools. More combinations could be tried.
- Feed-forward classification, though giving acceptable results, may not be the the best technique for choosing and classifying to HMMs.

## 6.3 Future Scope

Improvements over the current work and future plans for the project to make better recommendations:

- **Internet of Things (IoT)**

  The system can be used with IoT to determine weather data in a place and use it to make recommendations. The data may also be used to train the networks.

- **Social Media**

  The current implementation recommends the same products for any person in the same zip code on the same day. It does not take into account personal moods and choices which are set to vary more. Taking data from social media sites with open APIs like Twitter and with mood analysis of a person's post, their preferred item can be guessed more accurately. Hence it will be able to recommend items depending on personal choices more than common trends.

- **Larger, varied datasets**

  Have more varied number of shopping items and corresponding data from which to analyse from, so that recommendations are better and more varied across different buying items. They can even be categorized into different categories like genre, price range, quality and hence depending on the person's choice specific items can be recommended. Also, the current dataset may not reflect usable values as the data is a few years old and may not reflect current products that are popular.

- **More and better Normalization Techniques for the data**

- **Experimenting with algorithms for computing and choosing the most probable HMM**

  Have complex techniques like deep neural networks, back propagation networks to find out the most suitable HMM for a given observed sequence.

# REFERENCES


1. Hannak, Aniko, et al. "Tweetin'in the Rain: Exploring Societal-Scale Effects of Weather on Mood." ICWSM. 2012.
2. Thelwall, Mike. "Heart and soul: Sentiment strength detection in the social web with sentistrength." Proceedings of the CyberEmotions (2013): 1-14.
3. Weatherunlocked.com- white paper titled “The Complete Guide to Weather based Marketing”
4. “The History of Python: A Brief Timeline of Python". Blogger. 2009-01-20. Retrieved 2016-03-20.
5. Downey, Allen B (May 2012). Think Python: How to Think Like a Computer Scientist (Version 1.6.6 ed.). ISBN 978-0-521-72596-5.
6. McLachlan, Geoffrey J.; Do, Kim-Anh; Ambroise, Christophe (2004). Analyzing microarray gene expression data. Wiley.
7. #Database: https://research.chicagobooth.edu/kilts/marketing-databases/dominicks
8. #sales data(ccount.dta): http://kilts.chicagobooth.edu/dff/store-demos-customer-count/ccount_stata.zip
9. #store data(demo.dta): http://kilts.chicagobooth.edu/dff/store-demos-customer-count/demo_stata.zip
10. Dominicks Manual and Codebook: KiltsCenter 2013

11. Veningston, K., and R. Shanmugalakshmi. "Personalized location aware recommendation system." Advanced Computing and Communication Systems, 2015 International Conference on. IEEE, 2015.
12. Véras, Douglas, et al. "Context-Aware Techniques for Cross-Domain Recommender Systems." Intelligent Systems (BRACIS), 2015 Brazilian Conference on. IEEE, 2015.
13. Guo, Ying, and Xi Chen. "A Framework for Cross-domain Recommendation in Folksonomies." Journal of Automation and Control Engineering Vol 1.4 (2013).
14. Zhenzhen, Xu, et al. "Cross-domain item recommendation based on user similarity." Computer Science and Information Systems 13.2 (2016): 359-373.
15. Lawrence R. Rabiner “A tutorial on hidden Markov models and selected applications in speech recognition”, Proceedings of the IEEE 77.2, pp. 257-286, 1989.
16. Jeff A. Bilmes, “A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models.”, 1998.
17. "Old Farmer's Almanac". http://www.almanac.com/