ABSTRACT

MALLIK, NUPUR. Classification and Visualization of Wildfire Tweets. (Under the direction of Dr. Christopher Healey).

The world of data is growing and changing dramatically. This increase in data increases our need to better store, process, extract and explore. Data classification is an important aspect in many fields such as data management, business intelligence and machine learning. The classification that we focus on is this thesis pertains to the field of machine learning and statistics, where the problem refers to identifying which set of categories a new observation belongs to, based on properties of past observations. Our specific goal is to classify tweets from Twitter, an online microblogging social network. We have collected all tweets since May 14, 2013 containing the keywords “wildfire” or “forest service”. We seek to categorize these tweets into those that discuss wildfires (*wildfire* tweets), and those that simply contain one of the keywords but do not discuss wildfires (*other* tweets).

A wildfire or wildland fire is an uncontrolled fire in an area of combustible vegetation. Wildfires are usually extensive in size and can spread at great speed from their original source. Every year there are numerous wildfires, many of which are discussed on social media. Unfortunately, the term “wildfire” is not used exclusively for wildland fires. For example, it can also refer to a song, an item (e.g., the HTC Wildfire cell phone), or just a common phrase such as something “spread[ing] like a wildfire”.

To address this ambiguity, we applied a supervised classification technique to separate wildfire tweets from other tweets. Classification alone was not sufficient, since the total number of tweets was still huge. Manually exploring for particular information is time consuming and prone to error. To assist with this task, we built a web application designed to assist with data exploration. The application visualizes the location and content of geotagged tweets on a map, and provides ways to filter the information based on date range and additional keywords (e.g., to focus on the dates of a particular wildfire, or to filter based on a wildfire’s name). This greatly enhances the user experience and provides an effective data exploration mechanism.

Classification and Visualization of Wildfire Tweets

by
Nupur Mallik

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Science

Raleigh, North Carolina

2016

APPROVED BY:

_____	_____
Dr. Rada Chirkova	Dr. Robert St. Amant

_____
Dr. Christopher Healey
Chair of Advisory Committee

DEDICATION

To my family and Mr. Priyank Vyas

BIOGRAPHY

Nupur Mallik graduated from Bengal College of Engineering and Technology, Durgapur, India with a Bachelors degree in Computer Science and Engineering. She came to North Carolina State University to pursue Masters in Computer Science. She also interned at Apple Inc. during the summer of 2015. After completing her studies, she is planning to work as a Software Developer.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Christopher Healey for his constant guidance and support during my thesis. You are an excellent mentor and a great instructor and researcher and it was an honor working with you.

I would also like to thank Dr. Robert St. Amant and Dr. Rada Chirkova for giving me their time to review my thesis.

I am also thankful to my friends Priyank Vyas, Ritwika Roychoudhury, Prashant Trivedi, Dhara Desai and Deblina Gupta for their support.

Lastly, I would like to thank my family for their constant support and motivation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

This thesis describes a combined approach to: (1) classify social media text; then (2) visualize geotagged text from a particular category in a web application that supports visual exploration and discovery. Our practical need for this capability involves a collaboration with researchers from public policy and anthropology who are studying risk management and communication prior to and during wildland fire events. We were curious, is it possible to engage in a social network conversation with a community building structures in high-risk wildfire areas, to offer suggestions to mitigate potential risk to people and property? In addition, if a wildfire event occurs, is it possible to use the same social network to identify and prioritize a community's need for information, to help public information officers (PIOs) during their briefings to the public?

To address this problem, we have been collecting tweets with the keywords "wildfire" and "forest service" since March 14, 2013, producing approximately 3.1 million tweets to date. Unfortunately, tweets with these keywords do not always refer to wildland fires, so a method is needed to separate wildfire tweets from other tweets. Once this is done, wildfire tweets with geotag information are visualized on a Google map, allowing researchers to examine individual posts and the narratives they form during specific wildfire events.

1.1 Thesis Goals

This thesis focuses on capturing spatial-temporal data from a social media site (in our case, Twitter), classifying the data, and then visualizing the results in a web application.

Spatial-temporal data is data that contains both geographic location and time information, for example, historical tracking of plate tectonic activity or tracking of moving objects that typically can occupy only a single position at a given time. As with many other

forms of data, spatial-temporal data can often be efficiently explored and analyzed with appropriate visualization techniques. There are many software tools to visualize spatial-temporal data. These tools use the modern computer technologies, and they incorporate legacy knowledge from cartography research.

Social media mainly refers to web-based communication tools that enable people to interact with each other by both sharing and consuming information. Some prominent examples of social media are Facebook, Twitter, Google+, Wikipedia and LinkedIn. In our data analysis, we have captured Twitter data from May 14, 2013 containing the keywords "wildfire" and "forest service" as our spatial-temporal dataset. A wildfire or wildland fire is an uncontrolled fire in an area of combustible vegetation. We quickly observed that there could be tweets that contain the term "wildfire" that do not discuss wildland fires. Our first goal was to extract the true *wildfire* tweets from *other* tweets. We used a supervised classification method to separate the wildfire tweets from the non-wildfire ones.

Next, we needed a method to explore wildfire tweets. Because they were still numerous, our second goal was to build a web application to visualize geotagged tweets. Together with filters for date ranges and additional keywords, this allows our research collaborators to focus on specific wildfires (e.g., by name) or times of high wildfire risk, then examine the location and content of the tweets posted during the given wildfire events.

In summary, our research goals in this thesis are:

1. Construct a method to classify a tweet either as a wildfire tweet or as other tweet.
2. Visualize geotagged wildfire tweets on a web-based map that includes user interface elements to filter by date range and additional keywords.

Although our particular practical domain is risk management and communication during wildfire events, our techniques could easily generalize to a wide range of domains, as long as geotagged social media text for these domains was available.

1.2 Proposed Approach

We chose a supervised classification approach to build a classifier that separates tweets about actual wildland fires from other tweets.

Classification is the task of choosing a correct *class label* for a given input. Classification can be either supervised or unsupervised. A classification method is supervised if we use a set of training examples whose *class labels* are already to known to design and build the classifier. We can then use this classifier to predict the *class labels* for a set of records whose class labels are unknown. A classification method is unsupervised if we do not build the classifier with training data, but rather use properties of the data itself to classify it into separate classes, then use domain knowledge to judge the correctness of the results.

In our case, we decided to use supervised classification to build our classifier. We tested several well-known supervised classification methods against our training dataset to design a classifier, and then chose the method that gave the most accurate class label results. The classification methods that we tested were:

- Decision trees
- Naive Bayes
- Multinomial naive Bayes
- Naive Bayes for multivariate Bernoulli models
- Support vector machines

For testing the results of the different approaches, we used *k*-fold cross-validation. In our case we chose k=10 by dividing the training data into ten subsets. We used nine of the ten sets as training data and the remaining set to test the resulting model. We repeated the same process by rotating the ten subsets so each acted exactly once as a test set. Comparing the accuracy measure that we obtained from each of the above methods, we chose a multinomial Naïve Bayes classifier. After classification of the data, we designed and implemented a web application to visualize the results. The main components of this web app are as follows:

- **Google Map.** We used the Google Maps API to represent geotagged tweets on a map of the United States. The tweets are represented using SVG

circles. The map allows the user to zoom in or zoom out. When a user hovers on the SVG circle, it shows a pop-up containing the content of the tweet represented by the circle.

- **Number of tweets.** The web application also shows the number of tweets over a particular window of time.
- **Slider.** We provide a double-ended slider for filtering tweets based on a particular date range. This helps users narrow the results to gain information regarding a particular wildfire or wildfires that occurred at a particular time.
- **Keyword filtering.** The application also provides an autocomplete input-box that helps us to look for tweets containing particular keywords. For example, most of the wildfires have names, so if we are trying to find tweets regarding a particular named fire we can filter for it with the help of this feature.

1.3 Results

The wildfire tweet training set that we used for building the classifier consisted of 519 tweets. The training set consisted of the following attributes:

- The user ID or twitter handle of the tweet's author
- The latitude and longitude geolocation of the tweet
- The body of the tweet
- The timestamp the tweet was posted

For building the classifier we manually added a binary column to the dataset to indicate whether a given tweet is about a wildfire or not, where '1' indicates wildfire and '0' indicates other. Of the 519 tweets in the training set, 281 were classified as wildfire and 238 were classified as other. The classifier showed an average accuracy of 89.4117 percent based on our 10-fold training procedure.

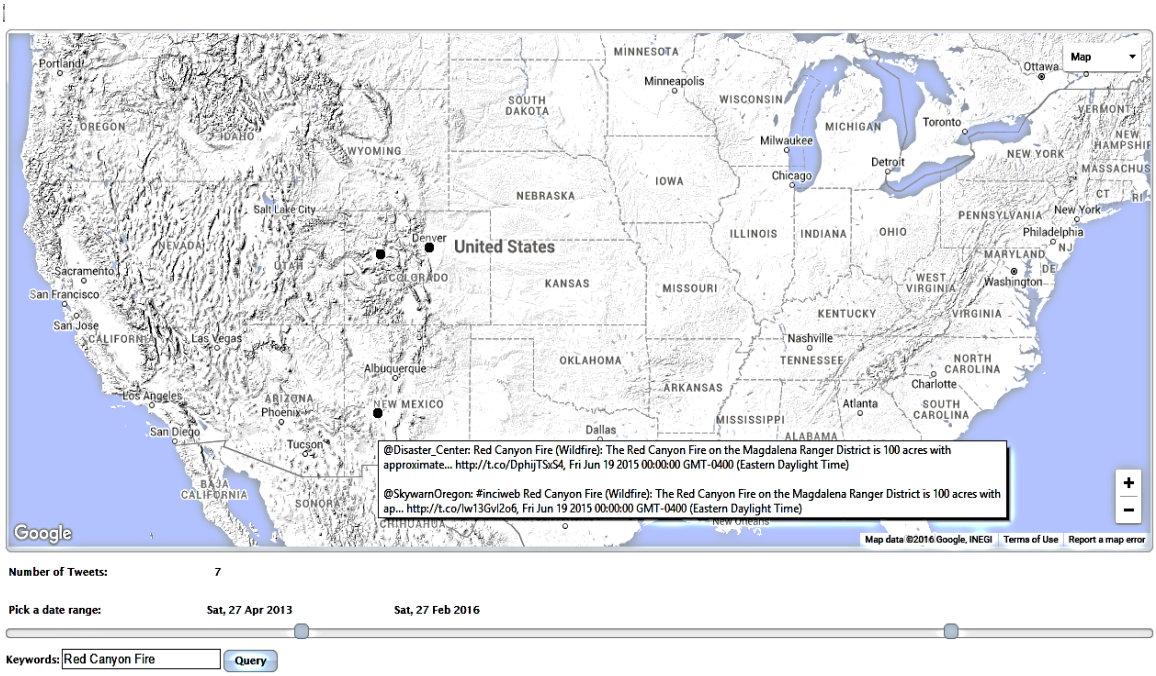


Figure 1. A visualization of tweets (shown as a circles) discussing the Red Canyon fire, selected using the date range slider and a keyword filter of "Red Canyon Fire"

The web application provides an effective visualization of the geotagged tweets. The time window implemented by the double-ended slider allows for focused results regarding wildfires that happened during a given time span. Additional keyword filtering helps to further narrow down the tweets being visualized, allowing users to gain information regarding a particular wildfire.

CHAPTER 2

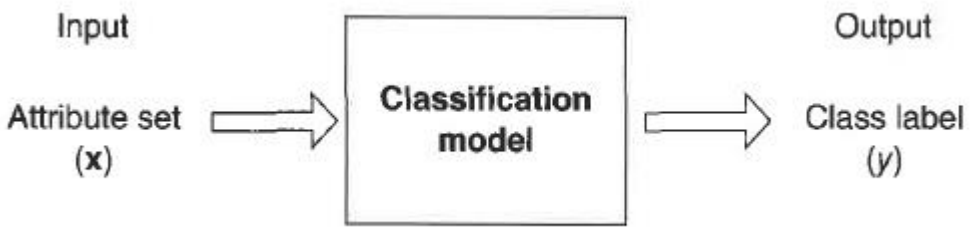
Background

2.1 Classification

Classification is a general task of categorization or the process of assigning labels to elements from one of the several predefined categories. Listed below are few examples of classification:

- Detecting spam emails based on the message header and content.
- Categorizing cells as malignant or benign based upon the results of an MRI scan.
- Categorizing an animal as a *vertebrate* or *invertebrate* depending on the presence or absence of a backbone.

In terms of data science classification is the task of mapping an input attribute set **x** into its class label *y*. The input data for a classification task is a set of records. Each of these records are also known as an instance or example and is characterized by a tuple (**x**, *y*) as shown in Figure 2 below, where **x** is the attribute set and *y* is the special attribute known as the *class label*.

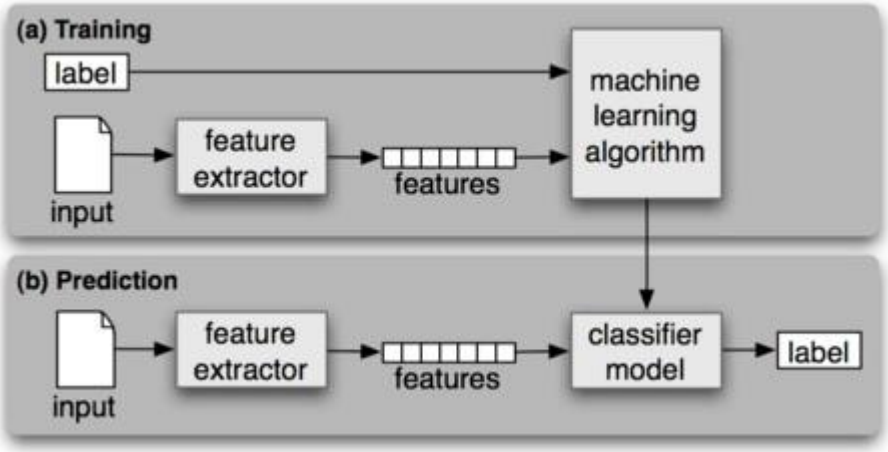


[17]

Figure 2. Classification model that takes an attribute set (**x**) as input and maps it to a class label (*y*)

Classification can be either supervised or unsupervised. In supervised classification a classifier is built based on a training set, i.e. the set of tuples where we know both **x** and *y*.

The model that results from training the classifier is used to classify other set of tuples where the *y* is unknown. The framework used by supervised classification is shown in Figure 3.



[17]

Figure 3. Framework used by supervised classification

During training, a feature extractor is used to convert each input value into a set of features, capturing the basic information about each input that should be used to classify it. Pairs of feature sets (**x**) and labels (*y*) are fed into the machine-learning algorithm to generate a model. For prediction, the same feature extractor is used to convert unseen inputs to feature sets. These feature sets are then fed into the model, which generates predicted labels.

In unsupervised classification we do not have any training dataset. The data is divided into several groups. The results of such classification are data driven and it is difficult to judge the correctness of the results in a conclusive way. Thus, it is important for a user with domain knowledge to be available to assess correctness. For example, cluster analysis is sometimes referred to as unsupervised classification.

There are several well-known supervised classification approaches. The ones we used in this thesis are described below.

2.1.1 Decision Trees

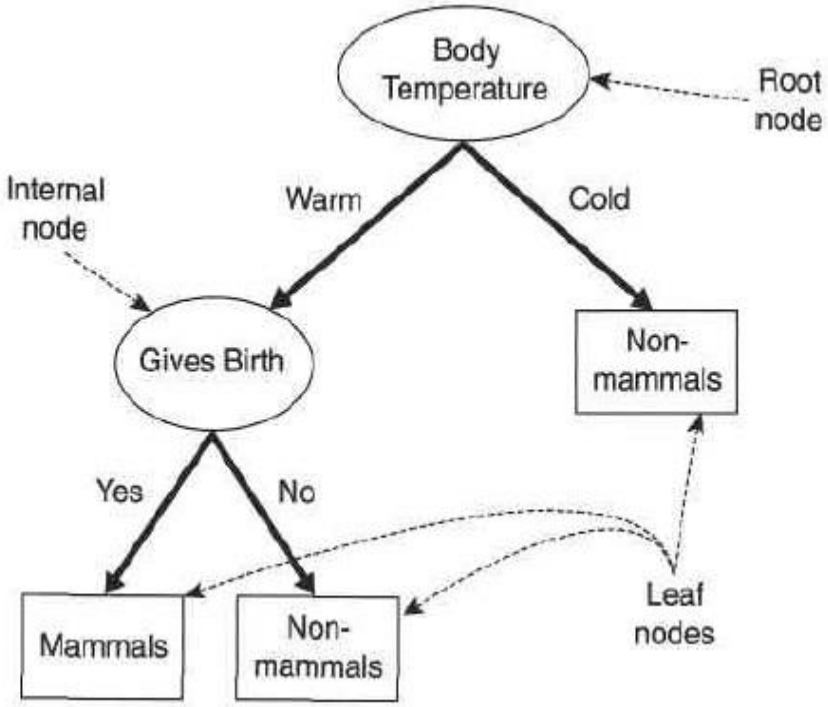
Decision tree learning are a method for assigning class labels to discrete-valued target functions, in which the learned function is represented as a decision tree [1]:

- **Root Node.** The originating node of the tree that has no incoming edges and zero or more outgoing edges.
- **Internal node.** Nodes that have exactly one incoming edge and two or more outgoing edges.
- **Leaf /terminal nodes**. Nodes that have exactly one incoming edge and no outgoing edges. These represent the class labels.

To assign a class label to an unlabeled record, its attribute values are compared at the root and different internal nodes in the tree. Edges correspond to an attribute value or range of values. Based on the record's value for the given attribute, it chooses the appropriate branch, continuing to apply this logic to form a path from the root to a leaf node. Once a leaf node is reached, the class value stored at that leaf is assigned to the record.

Figure 4 shows a decision tree that uses the attributes Body Temperature and Gives Birth to determine whether an animal is a mammal or a non-mammal. The root node splits on Body Temperature: Cold immediately leads to a leaf node with the class Non-Mammal. Warm leads to an internal node that performs a second split on Gives Birth: Yes leads to a leaf node of Mammal, and No leads to a leaf node of Non-Mammal. These rules can be represented by if-then conditions that form a logical representation of the conjunction and disjunction constraints on the attribute values.

We construct the decision tree using a training set that contains pre-classified records. Once built, this tree can then be applied to new records whose class labels are unknown. Although our goal is to build a perfect decision tree, which correctly classifies all training



[3]

Figure 4. A decision tree for classifying animals as mammals or non-mammals

records, finding an optimal decision tree is an NP hard problem. Instead, we can use a greedy approach where at each internal node of the tree the best splitting attribute is chosen. This assumes that choosing the best attribute will tend towards an optimal decision tree. All of the data attributes (except for the classification attribute) are considered as a splitting variable at each node. Mutual information is used to determine which attribute best splits the dataset to maximize mutual information (or minimize entropy) at the split.

Entropy measures the impurity of the training set. Suppose we have a training set as given in Table 1. To start, we need to calculate the entropy e of the class label, in our example, the Mammal/Non-Mammal label. Entropy is calculated as:

$$e(\text{attribute}) = \sum_{i=1}^{n} -p_i \log_2 p_i \qquad (1)$$

Table 1. Training Set for classification of Mammals and non-Mammals

Body Temperature	Gives Birth	Animal Type
Warm	Yes	Mammal
Warm	No	Non-Mammal
Cold	Yes	Non-Mammal
Cold	No	Non-Mammal
Warm	No	Mammal

for the *i* different values or ranges of values of the given attribute. For the Animal Type attribute, we have $n = 2$ values: Mammal and Non-Mammal. This produces a classification label entropy $e(\text{Animal Type}) = -0.4 \log_2 0.4 - 0.6 \log_2 0.6 = 0.97$.

Given the classification entropy, we can now choose the best split attribute by calculating the entropies of the attributes Body Temperature and Gives Birth. Entropy for a split attribute *A* (e.g., *A* = Body Temperature) is:

$$e(A) = \sum_{a \in A} p(a)\, e(a) \tag{2}$$

where, p_a is the probability of attribute value a in the dataset, and e_a is the entropy of a.

If we use Body Temperature to split the training set we will get four sets of records, with the following occurrence frequencies for each (Body Temperature, Animal Type) pair:

		Animal Type	
		Mammal	Non-Mammal
Body Temperature	Warm	2	1
	Cold	0	2

If we calculate the overall entropy after the split using Eq. 2, we obtain:

$$e(\text{Body Temperature}) = p(\text{Warm})\, e(\text{Warm}) + p(\text{Cold})\, e(\text{Cold}) \\ = (0.6 \times 0.92) + (0.4 \times 0) = 0.55 \quad (3)$$

Similarly, $e(\text{Gives Birth}) = 0.95$.

Information gain is the classification entropy minus the split attribute's entropy. This means the information gain for Body Temperature is $e(\text{Animal Type}) - e(\text{Body Temperature}) = 0.97 - 0.55 = 0.42$, while information gain for Gives Birth is $0.97 - 0.95 = 0.02$. Since the information gain for Body Temperature is higher, we use this as the split attribute at the root of the decision tree.

Once the split is made, the training set is divided based on the split attribute's values (in this case, Warm and Cold), and the process is repeated until the reduced training set contains records with only a single classification value. For example, in the Cold reduced training set, we have two records:

(Body Temperature = Cold, Gives Birth = Yes, Type = Non-Mammal)
(Body Temperature = Cold, Gives Birth = No, Type = Non-Mammal)

Since both records are classified as Non-Mammal, this forms a leaf node with classification Non-Mammal. The Body Temperature = Warm node does not produce a single classification, so it is further split on Gives Birth to produce two leaf nodes: Mammal and Non-Mammal (see Figure 4).

Unclassified records use the decision tree to obtain a class label. For example, a record (Warm, Yes), the decision tree would split left on Warm, and then split left on Yes, giving a result of Mammal. A record (Cold, No) would split right on Cold, giving a result of Non-Mammal. Additional records would traverse the decision tree in a similar manner to determine their class labels.

2.1.2 Bayesian Classifier

A Bayesian classifier uses conditional probabilities over a set of independent data attributes to train a classification model. The Bayesian classifier considers the value of each attribute independently, combining the evidence it collects to reach an overall class label recommendation. For example, if an animal's Body Temperature is Cold and Gives Birth is No, the Bayesian classifier will consider both of these properties independently as indicating a Non-Mammal, leading to Non-Mammal as the final recommended class label.

Equation 3 shows the Bayes theorem stated mathematically: [2]

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{4}$$

where A and B are events, $P(A)$ and $P(B)$ are probabilities of A and B independent of each other, $P(A|B)$ is the conditional probability of event A given that B is true, and $P(B|A)$ is the probability of event B given that A is true.

2.1.3 Naive Bayes Classifier

The Naïve Bayes classifier uses Bayes' theorem to solve the classification problem by applying an independence assumption: that the input features or attributes are conditionally independent of each other given the class label or classification. The mathematical representation of the conditional independence assumption is: [3]

$$P(X|Y = y) = \prod_{i=1}^{d} P(X_i|Y = y) \tag{5}$$

where, $X = \{X_1, X_2, \ldots, X_d\}$ is the feature set or set of d attributes, $Y = y$ is a class label value of y, $\prod_{i=1}^{d} P(X_i|Y = y)$ product of the individual conditional probability of each attribute X_i given a value y for the class label.

Using the same example of classification of animals in Table 1, the independence assumption is that the conditional probabilities of the attributes Body Temperature and Gives Birth are independent of one another given the class label. Consider building a Naïve Bayes classifier for Animal Type using Eq. 4. To define the classifier, we must compute three different probabilities.

1. **Prior Probability.** $P(A)$, known as the prior probability or the *prior*, is the probability of each of the classes in the training set.

$$P(A) = \frac{\text{Total examples of class A in training set}}{\text{Total records in training set}} \quad (6)$$

Applying this to the Animal Type attribute, we compute $P(\text{Mammal}) = \frac{2}{5} = 0.4$ and $P(\text{Non-Mammal}) = 0.6$.

2. **Likelihood.** $P(B|A)$, known as the likelihood of B given A, is the probability of generating the instance B given the different class labels. Consider B: (Body Temperature = Warm, Gives Birth = Yes). Using Eq. 5 produces

$P(\text{Body Temperature} = \text{Warm, Gives Birth} = \text{Yes} \,|\text{Mammal}) =$

$P(\text{Body Temperature} = \text{Warm} \mid \text{Mammal}) \times P(\text{Gives Birth} = \text{Yes} \mid \text{Mammal})$

$$= 1 \times 0.5 = 0.5$$

Similarity, $P(B \mid \text{Non-Mammal}) = 0.11$.

3. **Posterior Probability.** $P(A|B)$, known as the posterior probability, is the probability of class label A given an unknown instance B. We calculate posterior probabilities for each class label, and then assign B to the class with the highest posterior probability.

$$\text{posterior} \propto \text{likelihood} \times \text{prior} \quad (7)$$

We usually ignore the individual probability of the new record $P(B)$ because it is constant across all posterior probabilities.

Once the classifier's model is trained, it can be used to assign class labels to unlabeled records. Applying Eq. 7 to B, we obtain the following probabilities.

1. $P(\text{Mammal} \mid B) \propto P(B \mid \text{Mammal}) \times P(\text{Mammal}) = 0.5 \times 0.4 = 0.2.$
2. $P(\text{Non-Mammal} \mid B) \propto P(B \mid \text{Non-Mammal}) \times P(\text{Non-Mammal}) = 0.067.$

As we can see the $P(\text{Mammal} \mid B\,) > P(\text{Non-Mammal}| B)$. Thus, we can classify the new record B as Mammal.

Naïve Bayes classifiers are fast to train and classify, are insensitive to irrelevant features, and handle real, discrete, and streaming data. The conditional independence assumption can be a disadvantage because we lose the ability to exploit interactions between features. Also, if there are no occurrences of a class label with certain attribute values (e.g. Mammal and Body Temperature = Cold) then the frequency-based probability estimate will be zero, resulting in a posterior probability estimate of zero.

2.1.4 Bernoulli Naïve Bayes

Bernoulli Naïve Bayes classifiers follow a Bernoulli distribution. The probability mass function of this distribution over possible outcomes *k* is:

$$f(k;p) = \begin{cases} p & if\ k = 1 \\ 1 - p & if\ k = 0 \end{cases} \tag{8}$$

Below is an example of text classification using a Bernoulli Naïve Bayes classifier. Here, a document is represented by a feature vector of binary elements 0 and 1, corresponding to the absence or presence of a word in a document, respectively. We consider a set of tweets, each of which is about either a wildfire event (W) or some other event (O). Given a training set of five tweets, we want to estimate a Naïve Bayes classifier, using the Bernoulli document model, to classify the tweets as W or O. We first define a vocabulary of eight words.

$$V = \begin{bmatrix} w_1 = \text{wildfire} \\ w_2 = \text{torches} \\ w_3 = \text{spread} \\ w_4 = \text{home} \\ w_5 = \text{restaurant} \\ w_6 = \text{burning} \\ w_7 = \text{song} \\ w_8 = \text{drought} \end{bmatrix}$$

Individual words are selected from the five-tweet training set.

Tweet	Tweet Body	Classification
T_1	Wildfire by drake is still a good song to this day	other
T_2	The wide wildfire has spread over 7,000 acres	wildfire
T_3	North Texas Already Facing Drought Concerns for wildfire	wildfire
T_4	Douglas Co. wildfire torches homes, still burning strong	wildfire
T_5	I'm at Wildfire Restaurant near home	other

The training data is converted to a matrix for each class. Each row represents an eight dimensional feature vector.

$$B_{\text{wildfire}} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$B_{\text{other}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Suppose we have a new tweet B, “The wildfire has been burning strong, torched many homes, and has spread to 5,000 acres”, that we want to classify. The feature vector b for this tweet is $b = (1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0)$.

The prior probabilities from the training set are $P(W) = 0.6$ and $P(O) = 0.4$.

Word counts in the training set are the number of times $n_k(w_i)$ each word w_i in V occurs for each class k. For the class W the counts are:

$$n_w(w_1) = 3, n_w(w_2) = 1, n_w(w_3) = 1, n_w(w_4) = 1,$$
$$n_w(w_5) = 0, n_w(w_6) = 1, n_w(w_7) = 0, n_w(w_8) = 1$$

For the class O the counts are:

$$n_o(w_1) = 1, n_o(w_2) = 0, n_o(w_3) = 0, n_o(w_4) = 1,$$
$$n_o(w_5) = 1, n_o(w_6) = 0, n_o(w_7) = 1, n_o(w_8) = 0$$

The likelihoods are the probabilities of each word $P(w_t|C)$ given the document class C assigned in the training set, calculated as:

$$P(w_t|C = k) = \frac{n_k(w_t)}{N_k} \tag{9}$$

where, N_k is the total number of samples of class *k* present in the training set (i.e., N_w = 3 and N_o = 2).

Using Eq. (9) we can calculate, for example, the likelihood for $w_1 = \text{wildfire}$ as $P(w_1|W) = \frac{3}{3} = 1$. For all eight terms, the likelihoods for class W are $(1, 0.33,\ 0.33,\ 0.33,\ 0,\ 0.33,\ 0,\ 0.33)$. The corresponding likelihoods for class O are $(0.5,\ 0,\ 0,\ 0.5,\ 0.5,\ 0,\ 0.5,\ 0)$.

Finally, to estimate posterior probabilities and classify our unlabeled tweet B, we apply the following formula.

$$P(C|B) \propto P(b|C)\,P(C) \propto P(C)\prod_{i=1}^{|V|} b_i\,P(w_i|C) + (1 - b_i)(1 - P(w_i|C)\,) \tag{10}$$

Applying Eq. 10 for class W and $w_1 = \text{wildfire}$ gives $b_1 P(w_1|W) + (1 - b_1)(1 - P(w_1|W)) = (1 \times 1) + (1 - 1)(1 - 1) = 1$. Calculating $P(w_i|W)$ and $P(w_i|O)$ for all eight terms in V allows us to compute $P(W|B) \propto (0.6\ \times 1 \times 0.33 \times 0.33 \times 0.33 \times 1 \times 0.33 \times 1 \times 0.667) = 0.0049$ and $P(O|B) = 0$. Since $P(W|B) > P(O|B)$, we classify B as a wildfire tweet with class label W.

2.1.5 Multinomial Naïve Bayes

Multinomial naïve Bayes classifiers are assumed to follow a multinomial distribution. The probability mass function of this distribution is:

$$f(x_1, \ldots, x_k; n, p_1, \ldots, p_k) = \begin{cases} \dfrac{n!}{x_1! \ldots x_k!} p_1^{x1} \ldots p_k^{xk}, & if \sum_{i=1}^{k} x_i = n \\ 0 & otherwise \end{cases} \tag{11}$$

In this classifier, a document is represented by a feature vector of integers whose values represent the frequency of words in the document. We follow the same steps as outlined for Bernoulli Naïve Bayes to calculate posterior probabilities for each class, and then compare

them to classify new records. The equations for calculating the prior probabilities, likelihoods and posterior probabilities are as follows [4]:

$$P(C = k) = \frac{N_k}{N} \; \forall \; k \in C \tag{12}$$

$$P(w_t|C = k) = \frac{\sum_{i=1}^{N} x_{it} \, z_{ik}}{\sum_{s=1}^{|V|} \sum_{i=1}^{N} x_{it} \, z_{ik}} \tag{13}$$

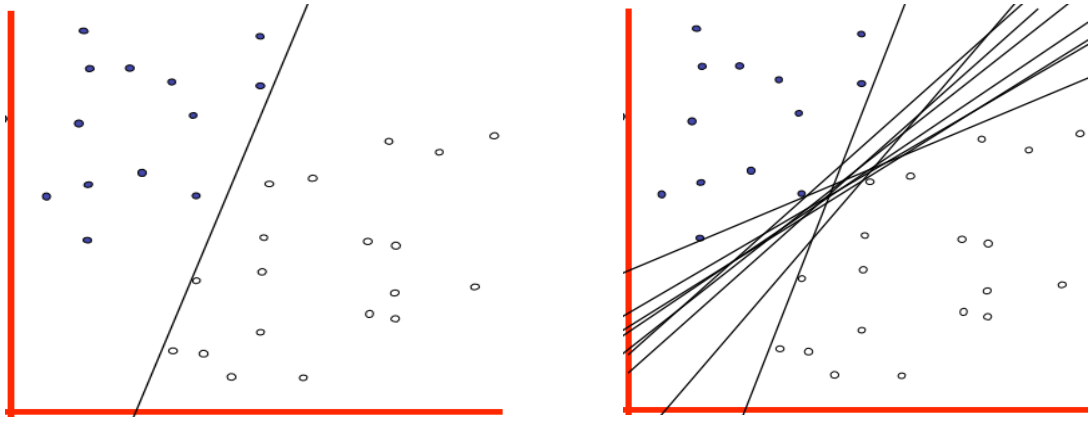
$$P\left(C\middle|D_j\right) = P\left(C\middle|x_j\right) \; \propto P\left(x_j\middle|C\right) P(C) \; \propto P(C) \prod_{t=1}^{|V|} P(w_t|C)^{x_{it}} \tag{14}$$

where N is the total number of documents in the training set, z_{ik} is an indicator variable set to 1 when document D_i is of class k, and x_{it} is the frequency of word w_t in document D_i.

For multinomial Naïve Bayes we also make the conditional independence assumption, where the probability of each word occurring in the document is independent of occurrences of the other words. Unlike the Bernoulli model, words that do not occur in the document (i.e., for which x_{it} = 0) do not affect the probability (since $p^0 = 1$). More information about the multinomial Naïve Bayes classification is provided in the Classification section of this document.

2.1.6 Support Vector Machines

Support vector machines are supervised learning models with an associated learning algorithm. SVM has its roots in statistical learning and has shown promising empirical results in practical applications like handwritten digit recognition and text categorization. SVM is a classifier that can be formally defined by a separating hyperplane. If we provide the SVM a labeled training set, it will return an optimal hyperplane that categorizes new examples. Consider Figures 5a and 5b below, assuming the points to represent a training set with two classes: filled points and open points. We can see a clear separation between the two classes' data points. Figure 5b shows many boundaries that can separate the two classes. SVM helps us obtain this optimal hyperplane [5].



(a) (b)

Figure 5. (a) two classes and one boundary; (b) two classes and many boundaries

Finding the optimal hyperplane involves calculating the margin of a linear classifier. This represents the width that a boundary that can be increased before it hits a data point (e.g., the yellow area shown in figure 6 below). Among all the boundaries shown in Figure 5b, the one that has the maximum margin is called the maximum margin linear classifier or optimal hyperplane. The data points that the margin touches are called the support vectors. In Figure 6, the data points encircled by the red circles are support vectors. This particular class of SVM is one of the simplest to solve, and is known as linear SVM. We choose the maximum margin because it gives the least chance of misclassification of a data point.

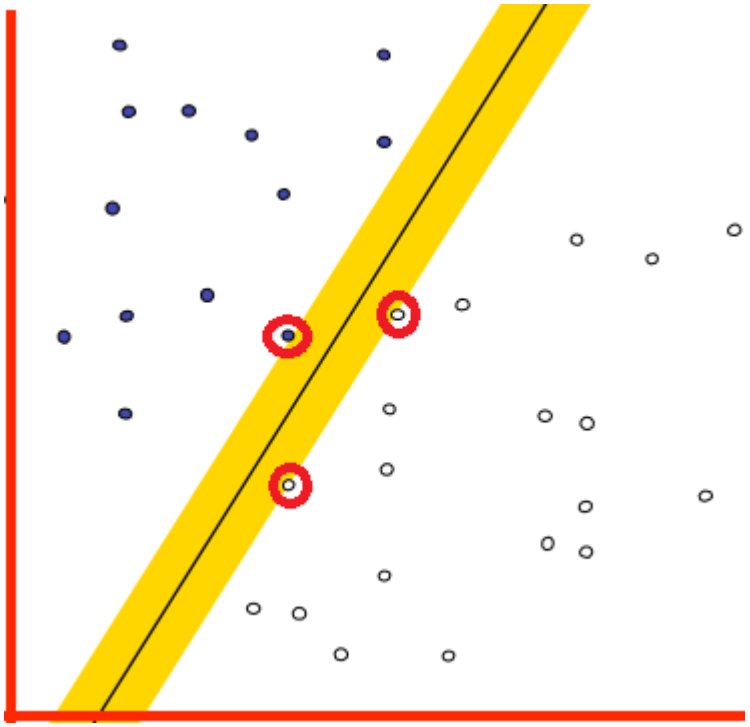


Figure 6. Margin of the SVM boundary, with support vectors circled in red

Mathematically, let us assume for our example the classes are + (filled circles) and – (clear circles). A hyperplane can be formally described as $f(x) = (w.x + b)$, where w is the vector perpendicular to the median of the maximal margin hyperplane [6] and x represents the training examples closest to the hyperplane (i.e., its support vectors).

The distance between a point x and the hyperplane can be calculated geometrically as $\frac{1}{||w||}$, the inverse of the magnitude of the vector w. Since the margin is equidistant on both sides of the hyperplane the width of the margin M is $\frac{2}{||w||}$. Finding the maximum margin hyperplane is equivalent to maximizing M or minimizing a function $L(w)$ subject to some constraints. These constraints require that the hyperplane classify correctly as many training examples x_i as possible. This corresponds to a Lagrangian optimization problem using

Lagrange multipliers to obtain w and b, formally stated as $\min_{w,b} L(w) = \frac{1}{2}||w||$, subject to $y_i(w.x_i + b) \geq 1 \; \forall \; i$, where y_i represents the labels of the training examples [7]. Once we calculate w and b, we can use them to classify unknown data points. Classifying a new data point u involves taking a dot product of u and w and adding it to b. If the result is greater than or equal to 0 then u is classified as +, otherwise it is classified as –.

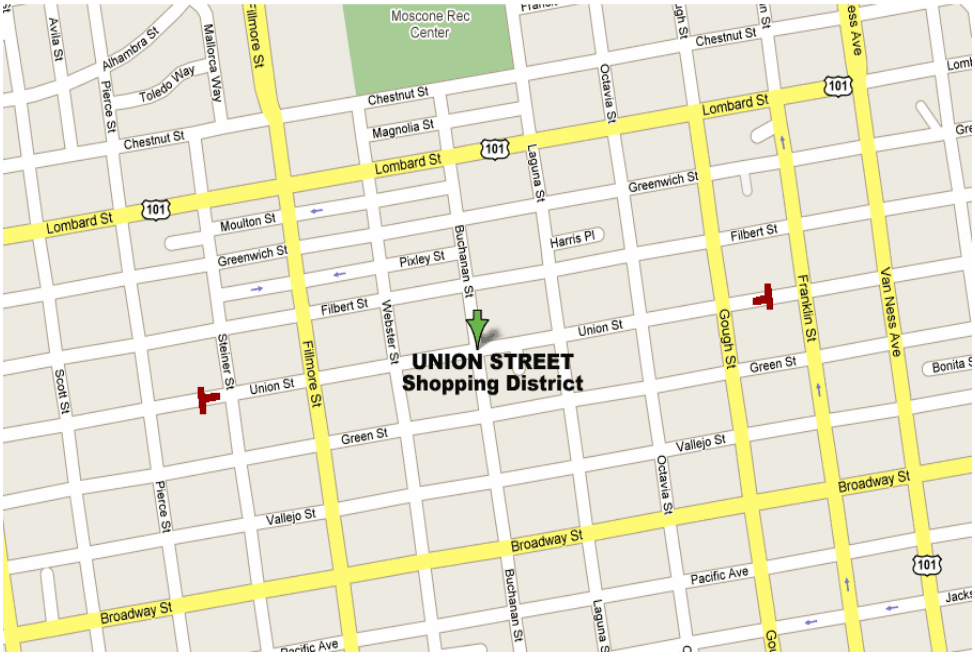
2.2 Map Visualization

A map is symbolic representation of selected characteristics of a place and their relationships, usually drawn on a flat surface. They represent information about the world in a simple visual way. Maps are one of the oldest and best-studied methods for visualizing data. Types of maps are characterized based on the specific tasks they are designed to support. Maps can be broadly classified into two categories.

2.2.1 General

General maps are designed for general use, and are usually focused on ease of understanding. These maps typically portray the physical environment and a variety of cultural elements for a geographic area at a particular point in time. The maps in this category usually show a geographic area larger than a city or town and do not display any particular subject that is part of one of a thematic category. Some good examples of general maps are street and tourist maps as shown in Figures 7(a) and (b) below.

(a)



(b)

Figure 7. (a) a tourist map (b) a street map

2.2.2 Thematic

Thematic maps are focused on specific theme or subject. They are designed for very specific data and analysis tasks. The different types of thematic maps are:

- **Choropleth.** Choropleth maps show data aggregated over predefined regions, such as counties or states, by coloring or shading these regions. For example, the choropleth map of water use in the US where the darker regions indicate more water usage is shown in the map in Figure 8. This technique assumes a relatively even distribution of the measured phenomenon within each region. Generally speaking, differences in hue are used to indicate qualitative differences, such as land use, while differences in saturation or lightness are used to indicate quantitative differences, such as population.

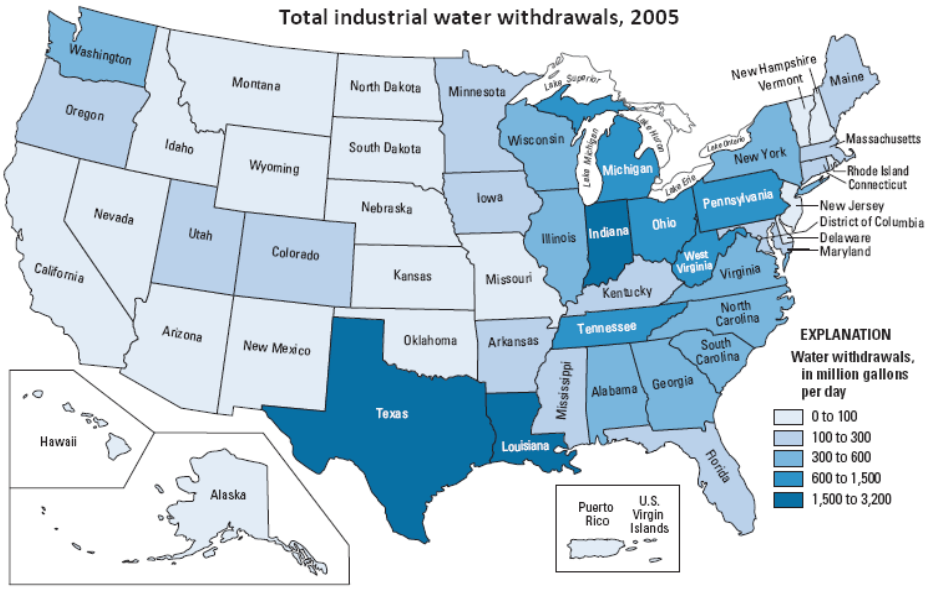


Figure 8. Choropleth map of total industrial water withdrawal in 2005

- **Isarithmic or Isopleth.** Isarithmic or isopleth maps are used to depict smooth continuous phenomena such as precipitation, surface temperature, or elevation. They are also known as contour maps. Each line or contour on this type of map represents a path with the same value. For example, on a surface temperature map like the one shown in Figure 9, each temperature line indicates a path over the country with the labelled temperature. If contour lines are close to one another (e.g. in center and southern California in Figure 9), the value the contours represent is changing rapidly. Where contour lines are spaced farther apart (the southeast in Figure 9), the value is changing more slowly.

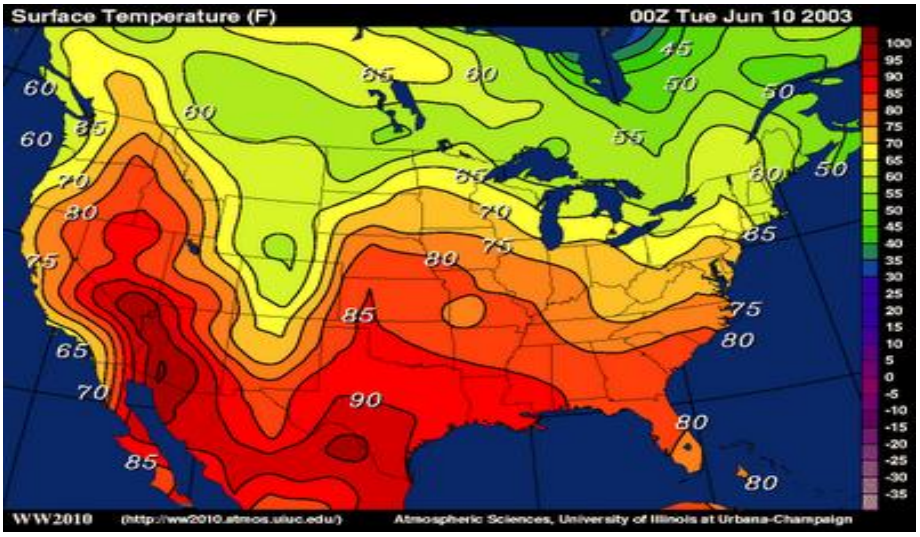


Figure 9. Isarithmic or isopleth map of surface temperature boundaries

- **Proportional Symbol.** These maps use symbols of different sizes to represent data associated with different areas or locations within the map. For example, a circle may be shown within each state on a map (Figure 10 below), where the area of the circle visualizes the Hispanic population of the respective states.

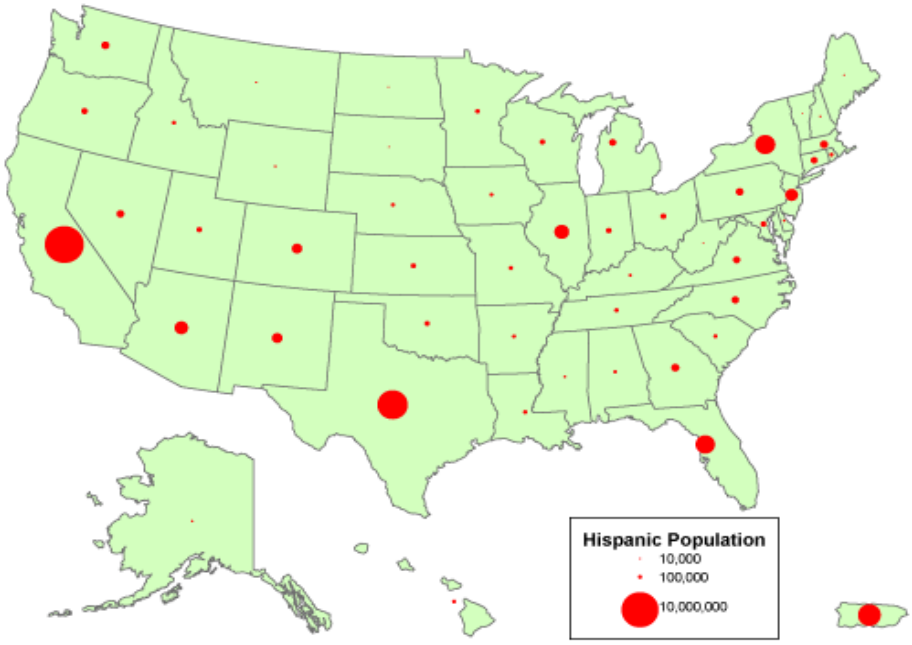


Figure 10. Proportional symbol map showing the distribution of Hispanic population across the United States

- **Dot.** A dot map or a dot distribution map is used to represent the presence or absence of an event at a geographic location. It can be used to locate each occurrence of a phenomenon. A dot may indicate any number of entities, for example, one dot for every 100 voters. For example, the map shown in Figure 11 below depicts cell phone tower locations in US.

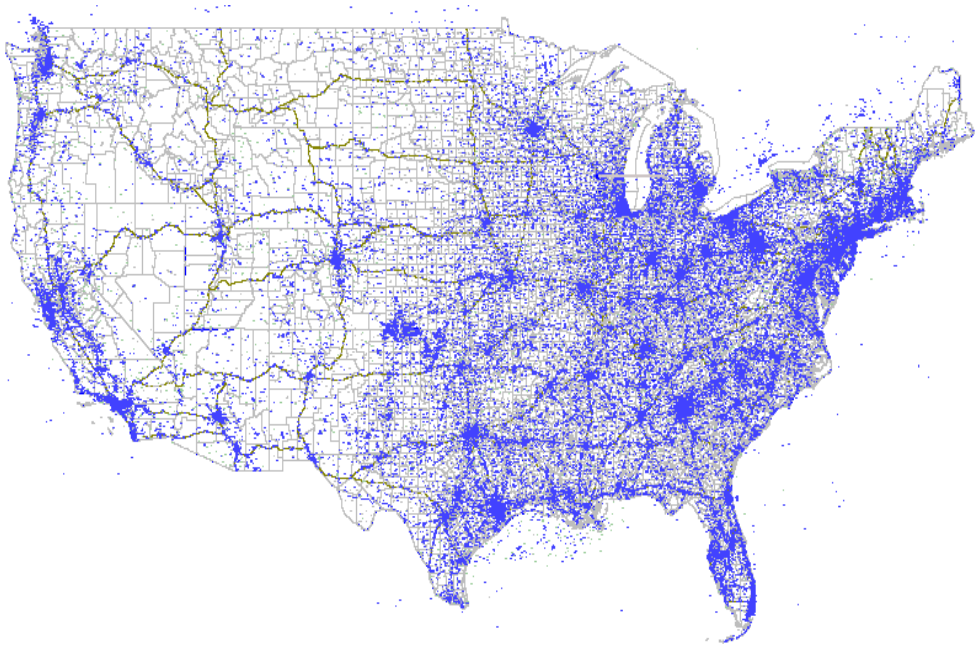


Figure 11. Dot map showing cell phone towers in the United States

- **Dasymetric.** A dasymetric map is an alternative to a choropleth map. In a choropleth map, the population density of a region is represented by varying hue or luminance across different regions. A dasymetric map warps the size of the region to represent the value being visualized within that region. For example, the map shown in Figure 12 below shows the crimes in Greater London.

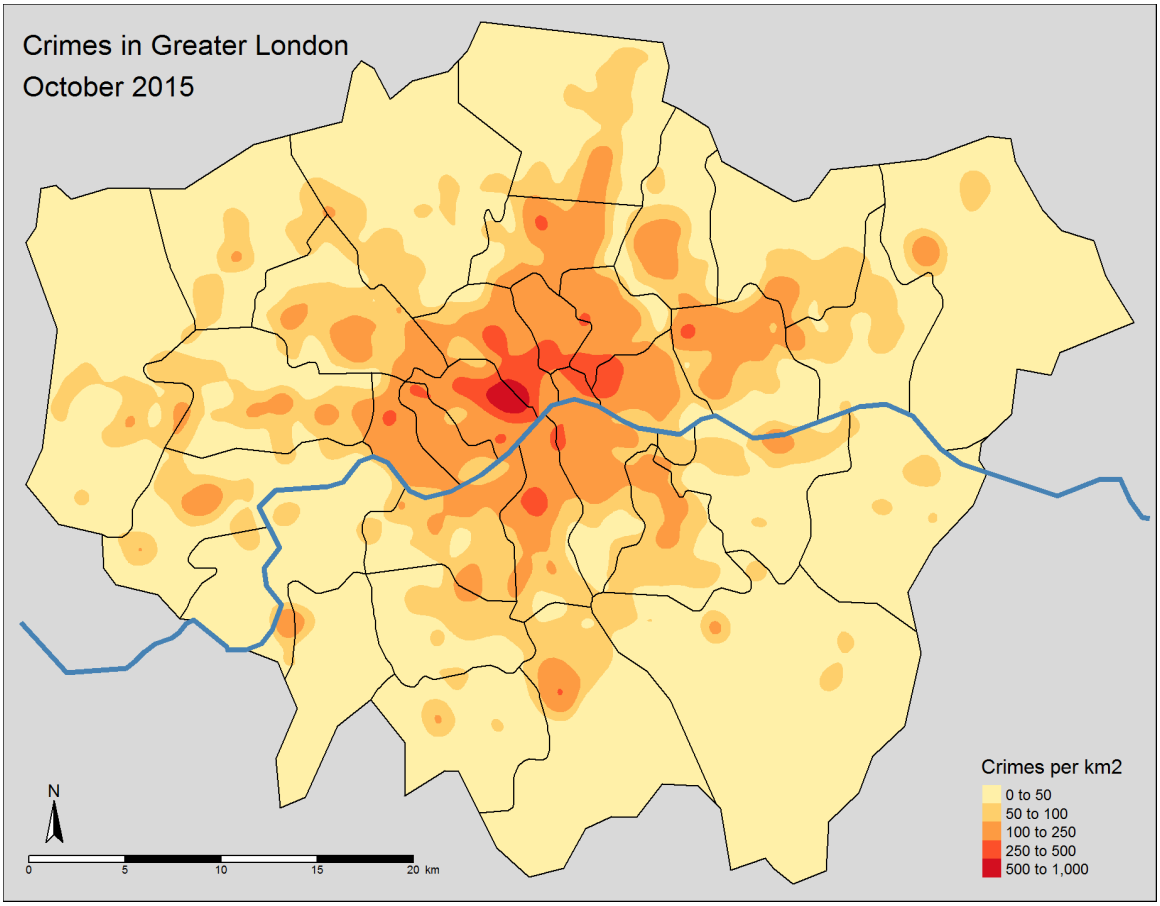


Figure 12. Dasymetric map showing the crimes in Greater London in October 2015 with districts warped such that district area represents crime rate

2.3 Web-based Visualization

Visualization is the process of representing data in the form of a picture or an image that can help in better understanding of the data. Visualization has many applications in various fields such as science, education, engineering, interactive, multimedia, medicine, and data analysis. The primary goal of data visualization is to communicate information clearly and efficiently to users via statistical graphics, plots, information graphics, tables, charts, and maps. There are many different tools available to help with data visualization. Traditionally these tools were designed to be run as native programs for specific desktop platforms, but

more recently web-based visualization tools have become popular. These tools can be easily accessed through modern web browsers, helping address issues like accessibility, portability and distribution.

Web-based applications can be designed using a variety of technologies supported by current web browser engines. These include:

- **SVG.** SVG stands for Scalable Vector Graphics. SVG is used to define vector-based graphics for the Web in an XML format. Vector graphics are useful, since they do not lose quality if they are zoomed or resized. Elements and attributes in SVG files can be animated based on a W3C recommendation that integrates with other W3C standards such as the DOM and XSL. There are other advantages of SVGs over alternative formats such as JPEGs and GIFs. SVG images can be created and edited with a text editor and can be searched, indexed, scripted, and compressed. The disadvantages of SVG are that it is difficult to develop and there are limited authoring tools (e.g., we cannot edit SVGs in Photoshop).
- **JavaScript Canvas.** The HTML <canvas> element is used to dynamically render graphics on a web page, usually via JavaScript scripting. The <canvas> element is a container for pixel-based graphics. The script is used to draw the graphics. The canvas has several methods for drawing different shapes and for adding images. Like SVG, it is supported by most modern web browsers.
- **D3.js.** D3.js (D3, for Data-Driven Documents) is a JavaScript library for producing interactive data visualizations in a web browser. It makes use of SVG, HTML5, and CSS standards within its algorithms. In D3, data is attached to DOM (Document Object Model) elements within a web page specification, then manipulated and displayed using SVG. The data can be made interactive with D3's data-driven transformations and transitions. D3 uses a functional programming style to support code reusability and the flexibility to add new functions as per a designer's requirements. Figure 13 shows a choropleth map

visualization of unemployment rates from 2008 with a quantize scale ranging from 0 to 15%, created using D3.

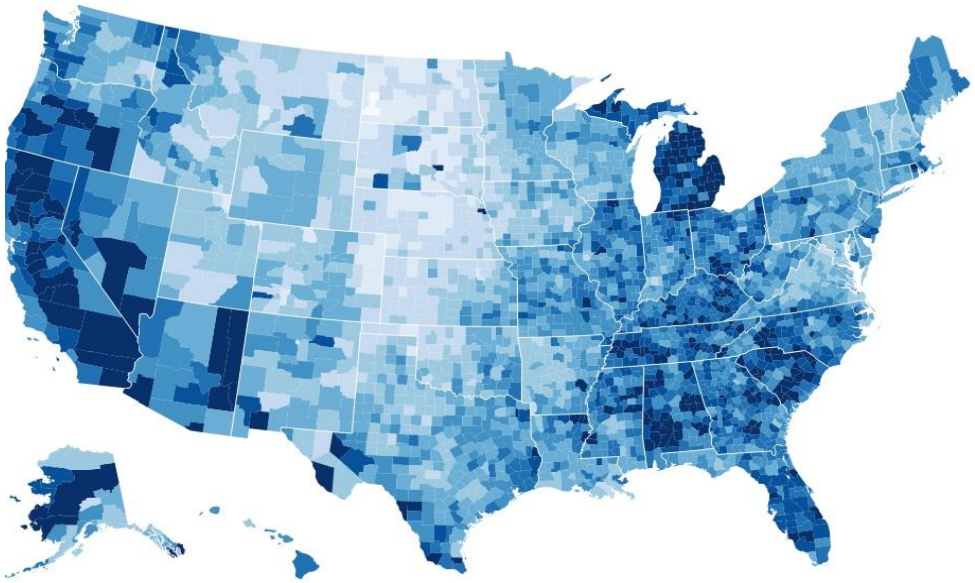


Figure 13. Choropleth map visualization encoding unemployment rates from 2008 with a quantize scale ranging from 0 to 15% created by D3.js

- **Google Maps.** Google Maps is a web mapping service developed by Google. It offers satellite imagery, street maps, 360° panoramic views of streets (Street View), real-time traffic conditions (Google Traffic), and route planning. Originally developed at Where 2 Technologies, Google Maps was later acquired by Google in 2004 and converted into a web application. It uses JavaScript, XML, and Ajax for the front-end display engine. The Google Maps API allows maps to be embedded on third-party websites, and offers various functionality like city locators and source–destination route requests. The Google Maps API also allows developers to visualize data overlaid on an underlying Google Map, allowing it to act as a flexible map visualization library. Figure 14 uses the Google Maps JavaScript API to visualize the number of Syrian–American businesspersons who migrated to different parts of the United States in 1908–1909.

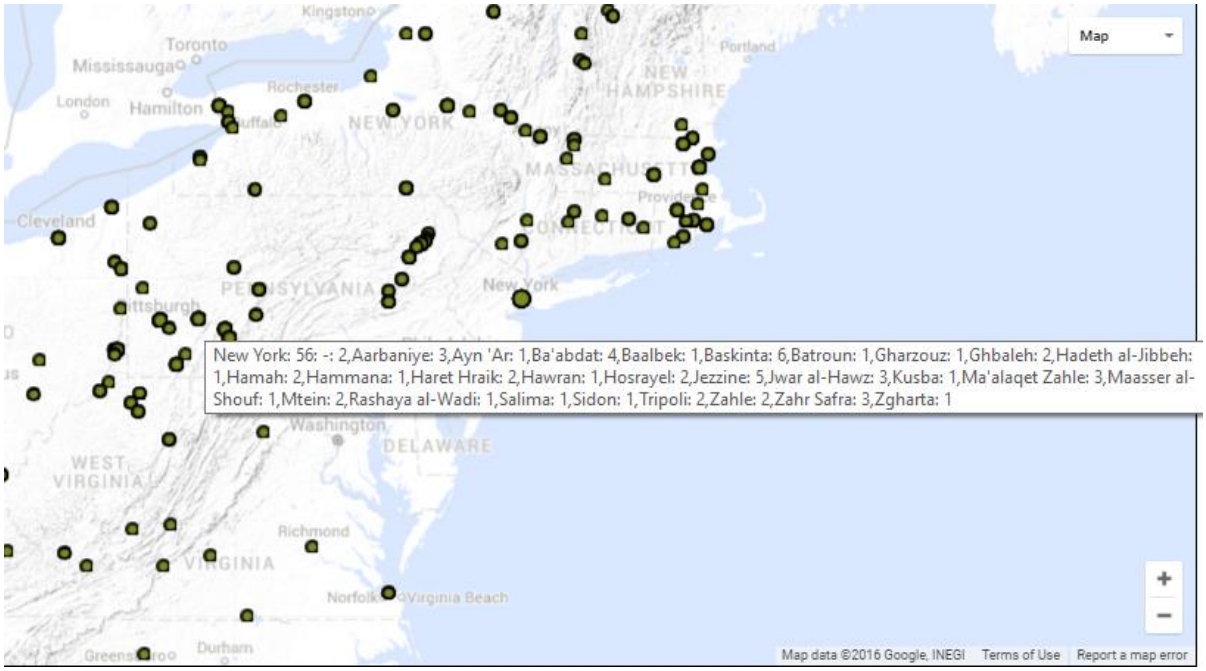
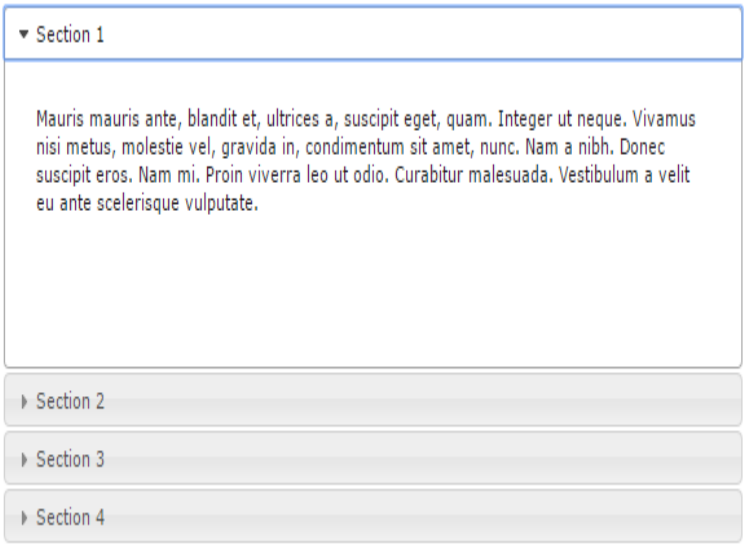
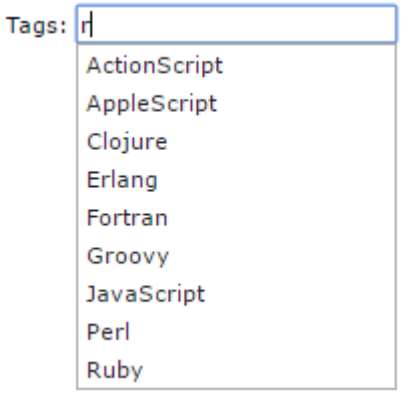
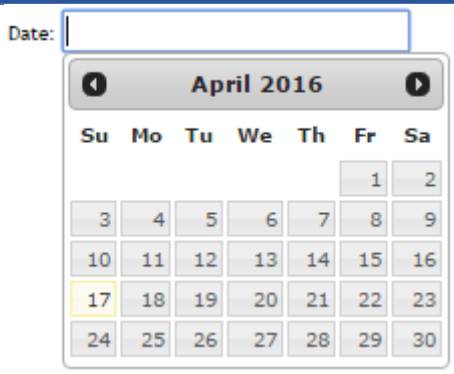
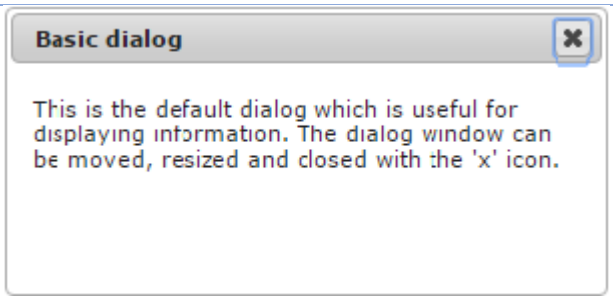
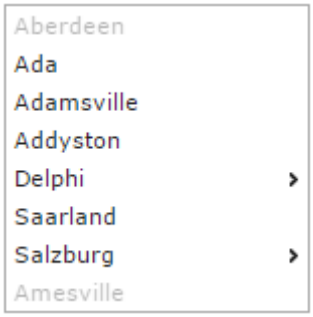
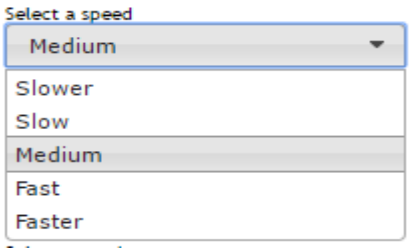


Figure 14. Map visualization showing the number of Syrian–American businessmen who migrated to different parts of the United States in 1908–1909, created using the Google Maps JavaScript API

- **jQuery.** jQuery is a cross-platform JavaScript library designed to simplify client-side scripting of HTML. It is a fast, small and feature rich. Its syntax makes it efficient to navigate an HTML document's DOM elements and develop Ajax applications. jQuery also provides a fully featured set of UI widgets, which can be used to provide standard UI functionality on a web site. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and Web applications. Example of jQuery widgets are as shown in the table below: [8]

Table 2. Different jQuery widgets

Widgets	Images
Accordion	
Autocomplete	
Button	

Widgets	Images
Datepicker	
Dialog	
Menu	
Progressbar	
Selectmenu	

CHAPTER 3

Classification

In this thesis, we use supervised classification to build a classifier to separate wildfire tweets from other tweets that do not discuss wildfire events. The steps undertaken to build this classifier are as follows:

1. **Data pre-processing.** Extract and structure tweets from the tweet dataset upon which the classification methods will be applied.
2. **Classifier.** Identify the best possible classifier by applying several supervised classification techniques and choosing the one that provides the best accuracy.
3. **Training and Testing.** To assess the performance of each of the classifiers we performed *k*-fold cross validation with *k*=10.

3.1 Data pre-processing

Data pre-processing is a data mining technique where raw data is transformed into a structured format in preparation for text analytics algorithms. It is normally the first step in a data mining or data analytics tasks. Examples including tasks like:

- Stop word removal, where words that provide little or no semantic information are removed from the text (e.g., article like “a” or “an,” or prepositions like “in” or “after.”)
- Part of speech tagging, where words are assigned their appropriate part of speech (e.g., noun, verb, adverb, etc.)
- Stemming, where words with a common semantic meaning are truncated into a common form (e.g., “runs”, “running”, “ran” are stemmed to the common form “run.”)

In our case, we choose not to apply cleansing to the tweets, but we did have to extract geo-tagged tweets from the wildfire tweet database that contained the term “wildfire.” Twitter geo-tagging is “opt-in,” so unless users permit location information to be attached to their tweets, it is not included. The opt-in rate for Twitter is low, often 1% or less. Out of approximately 3.1 million tweets, we found 28,772 tweets that that includes a geolocation, which is slightly less than 1% of our total dataset.

To classify the 28,772 tweets into “wildfire” and “other” tweets, we chose a subset of 519 tweets from the 3.1 million tweets that contained both geo-tagged and non-geo-tagged tweets. Our assumption is that tweets are returned via Twitter’s API independently, that is, tweets adjacent in time are not assumed to be related to one another. Based on this assumption, we chose the 519 tweets uniformly from the 3.1 million tweet dataset. We used these 519 tweets as a training set for building the supervised classifier.

The data columns that were present in the original dataset are:

- Twitter tweet ID
- Author ID
- Geolocation, represented as a latitude and longitude, if included
- Tweet body
- Date and time the tweet was posted

Based on human examination of the tweet body, we manually added a binary column to indicate whether a given tweet was about a wildfire event or not, where ‘1’ indicates a wildfire and ‘0’ indicates other.

3.2 Classifier

We began with a Python-based multinomial Naive Bayes classifier to construct a model that classifies tweets as “wildfire” or “other.” We then compared the results to three other classification algorithms: simple Naïve Bayes, decision trees and support vector machines.

Multinomial Naïve Bayes implements the Naive Bayes algorithm for data that has a multinomial distribution. The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class *y*, where *n* is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i | y)$ of feature *i* appearing in a sample belonging to class *y*.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, for example, relative frequency counting:

$\widehat{\theta_{yi}} = \frac{N_{yi}+\alpha}{N_y+\alpha n}$ where, $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature *i* appears in a sample of class *y* in the training set *T*, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all features for class *y*.

The smoothing priors $\alpha \geq 0$ account for features not present in the training set and prevents zero probabilities in future computations on previously unseen tweets. Setting $\alpha = 1$ performs Laplace smoothing. $\alpha < 1$ is called Lidstone smoothing.

To construct our multinomial Naïve Bayes classifier we performed the following:

1. From a CSV file containing the 519 training tweets we read the columns containing the tweet body and the binary indication of whether the given tweet is about an actual wildfire.
2. Tweet bodies and wildfire tweet flags were store in a text corpus.
3. We applied the multinomial naive Bayes classifier from scikit-learn [9] [10] to the text corpus. Since the classifier expects numeric feature vectors with a fixed size and not raw text, we first used TfidfVectorizer() to transform the tweet bodies in the corpus to fixed sized TF-IDF weighted term vectors. Each term's weight corresponds to its importance in distinguishing this tweet from other tweets in the corpus.
4. The text corpus entries were divided into ten subsets, used as training sets and test sets. Ten-fold cross validation was used to train the classifier on nine subsets, and then evaluate the accuracy of the resulting model on the tenth

subset. Training and test subsets were rotated ten times to produce a final accuracy score.

3.3 Training and Testing

Training and testing are two important aspects of supervised classification techniques. In our case, we used 10-fold cross validation to train and test our classifier. As stated above we had 519 tweets in our training dataset: 281 about actual wildfires and 238 about other topics. We divided the training dataset into ten subsets. During each cross-validation step, we chose nine of the subsets to act as a training set, and then tested the resulting model on the tenth subset by using it to predict the class labels of its tweets. The training and test subsets rotated, for example, the first cross-validation step used subsets 1–9 for training and subset 10 for testing, and the second cross-validation step used subsets 1–8 and 10 for testing and subset 9 for training, and so on. We checked for model correctness by using an accuracy measure on the class labels proposed for the test subset.

$$Accuracy = \frac{Number\ of\ tweets\ correctly\ classified\ by\ the\ classifier}{Total\ number\ of\ tweets\ tested}$$

After we repeated the above process of training and testing ten times, we calculated the overall accuracy of the classifier by taking an average of the accuracies of the ten cross-validation steps. In case of the multinomial Naïve Bayes classifier the overall accuracy was 89.4%

3.4 Comparison to other methods

As noted earlier, we tested several supervised classification techniques: multinomial Naïve Bayes, Naïve Bayes, Bernoulli Naïve Bayes, decision trees, and support vector machines. We compared their accuracies to identify the classifier that gave the best overall accuracy on the training set. Table 3 contains all ten cross-validation accuracies, as well as each method's overall performance:

Table 3. Comparative results of the accuracy of the different classifiers

Test Set	Naive Bayes	Naïve Bayes w/o weight	Decision Tree	Bernoulli Naïve Bayes	SVM	Multinomial Naïve Bayes
subset 1	78.43%	78.43%	72.55%	66.67%	33.33%	94.12%
subset 2	80.39%	80.39%	80.39%	76.47%	58.82%	94.12%
subset 3	76.47%	76.47%	68.63%	62.75%	41.18%	88.24%
subset 4	72.55%	72.55%	66.67%	72.55%	52.94%	88.24%
subset 5	49.02%	49.02%	35.29%	29.41%	13.73%	70.59%
subset 6	82.35%	82.35%	76.47%	72.55%	47.06%	96.08%
subset 7	52.94%	52.94%	52.94%	52.94%	35.29%	88.24%
subset 8	78.43%	78.43%	82.35%	84.31%	76.47%	90.20%
subset 9	84.31%	84.31%	88.24%	80.39%	9.80%	94.12%
subset 10	79.25%	79.25%	77.36%	79.25%	7.55%	90.20%
Overall Results:	**73.41%**	**73.41%**	**70.09%**	**67.73%**	**37.62%**	**89.41%**

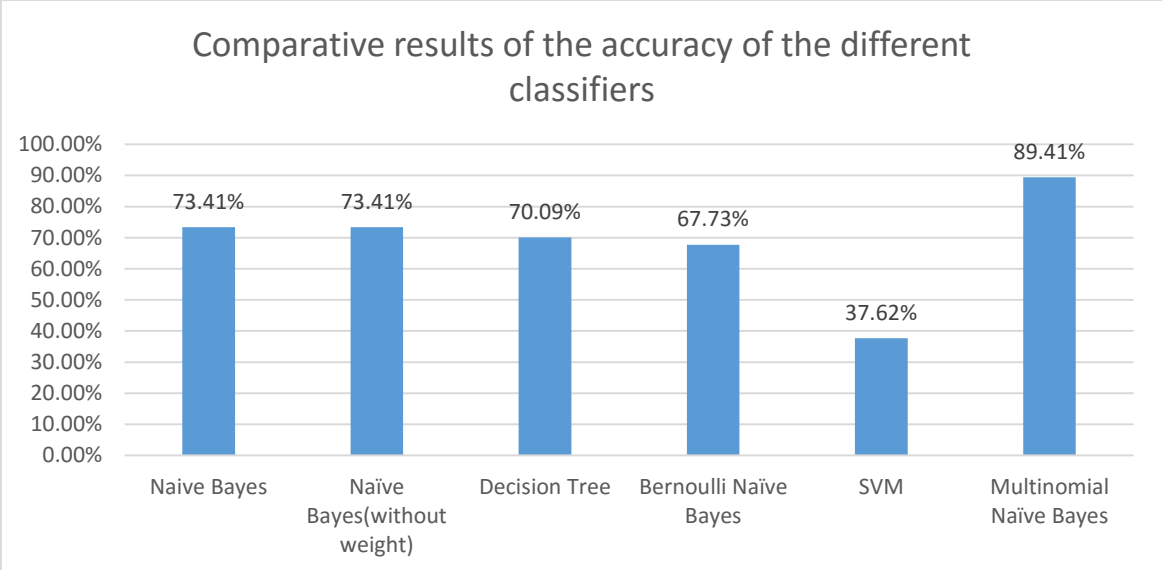


Figure 15. Bar graph showing the comparative results of the accuracy of the different classifiers

Based on the accuracy results multinomial Naïve Bayes classifier gave the best results. We performed a one-way ANOVA (analysis of variance) to compare the algorithms for

significant differences in performance. For accuracy, we obtained $F = 12.61$, $p < 0.001$, showing a significant difference in accuracy based on the classification technique we applied [11]. Post-hoc analysis using Tukey's range test was then applied to pairs of classifiers to identify which classifiers performed better than the others did. Based on the table below we see that decision trees, Naïve Bayes, Naïve Bayes without weights, Bernoulli Naïve Bayes, and multinomial Naïve Bayes are all significantly more accurate than SVM. Multinomial Naïve Bayes also performs significantly better than Bernoulli Naïve Bayes, but there is no significant accuracy difference between Bernoulli Naïve Bayes, Naïve Bayes, and Naïve Bayes without weights. Based on this, we chose to use multinomial Naïve Bayes to build our wildfire classification model.

Table 4. Tukey's range test

Group 1	Group 2	Mean Difference	Lower	Upper	Significant
Decision Tree	Naïve Bayes	0.0332	-0.1666	0.2331	no
Decision Tree	Bernoulli NB	-0.0236	-0.2235	0.1763	no
Decision Tree	Multinomial NB	0.1933	-0.0066	0.3931	no
Decision Tree	Naïve Bayes w/o weight	0.0332	-0.1666	0.2331	no
Decision Tree	SVM	-0.3247	-0.5246	-0.1248	yes
Naïve Bayes	Bernoulli NB	-0.0569	-0.2567	0.143	no
Naïve Bayes	Multinomial NB	0.16	-0.0399	0.3599	no
Naïve Bayes	Naïve Bayes w/o weight	0	-0.1999	0.1999	no
Naïve Bayes	SVM	-0.358	-0.5579	-0.1581	yes
Bernoulli NB	Multinomial NB	0.2169	0.017	0.4167	yes
Bernoulli NB	Naïve Bayes w/o weight	0.0569	-0.143	0.2567	no
Bernoulli NB	SVM	-0.3011	-0.501	-0.1012	yes
Multinomial NB	Naïve Bayes w/o weight	-0.16	-0.3599	0.0399	no
Multinomial NB	SVM	-0.518	-0.7179	-0.3181	yes

CHAPTER 4

Web Application

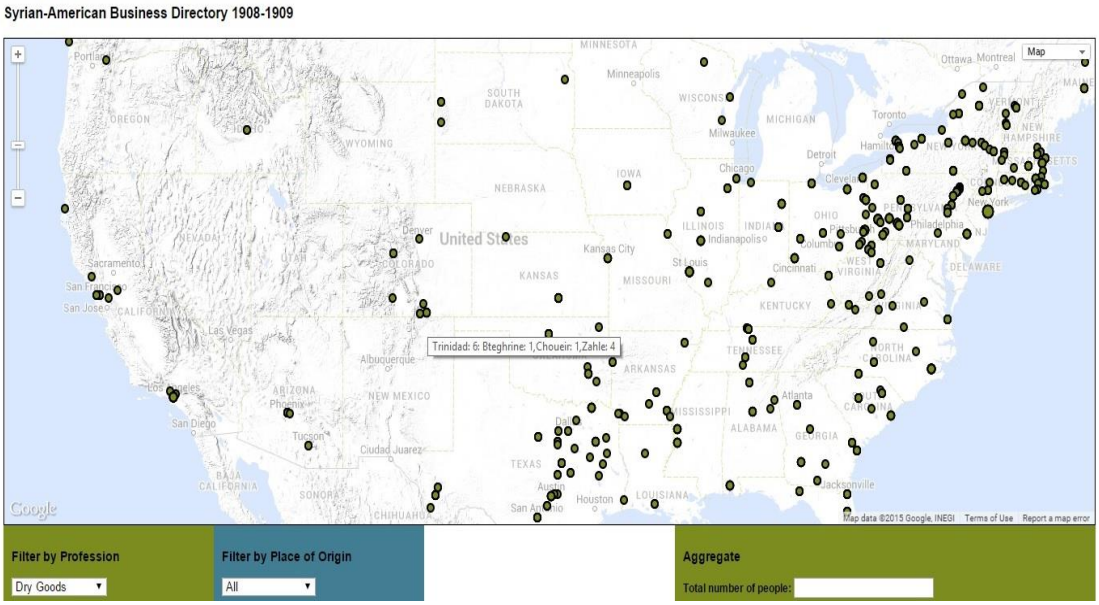
In this thesis, we developed a web application to visualize the results of our classifier. We used a web application to represent the results because these type of applications are easily accessible for users. They require an internet connection and a modern web browser. This makes them efficient to disseminate, and usable across a variety of hardware devices and operating systems.

We initially developed a web application for Dr. Akram Khater, a University Faculty Scholar and Professor of History who holds the Khayrallah Chair in Diaspora Studies at North Carolina State University, where he serves as the Director of the Khayrallah Center for Lebanese Diaspora Studies. This web application was designed to visualize data related to Syrian–American businesses from 1908–1909 on an underlying Google Map. Information about the various Syrian business owners in the U.S. was represented with simple SVG glyphs that varied their appearance to represent different data attribute values. This web application proved to be very helpful, and acted as a model when we started work on a method to visualize data from the wildfire.

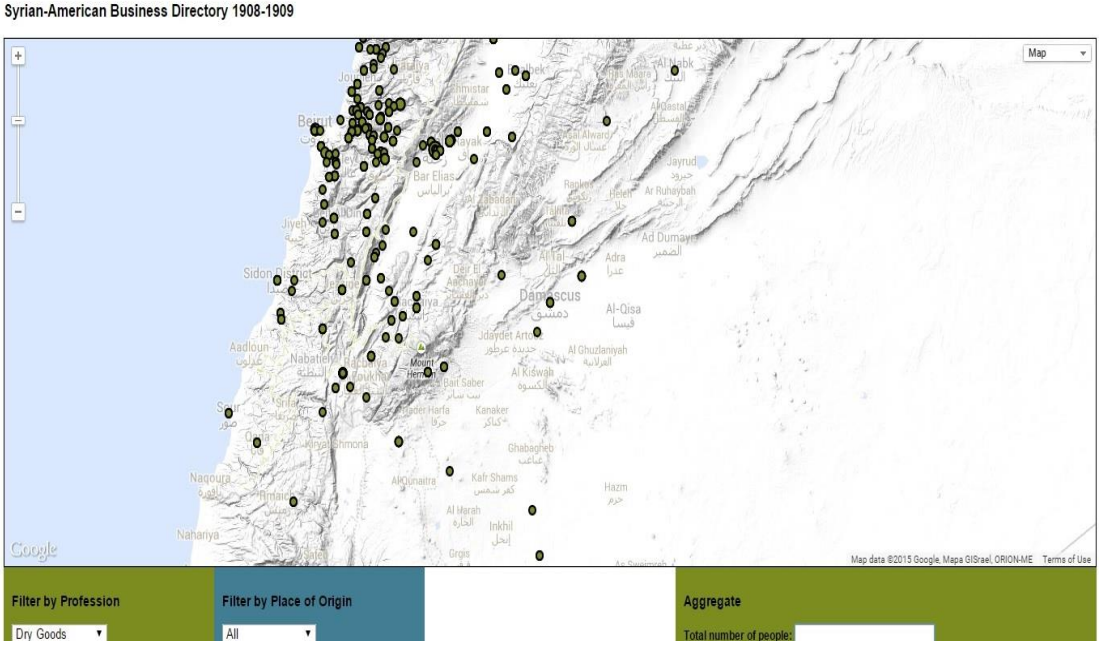
4.1 Preliminary Work

The web application that we developed for Dr. Khater was primarily used to depict data related to Syrian-American businesses from 1908–1909 on an underlying Google Map. We used HTML, JavaScript and the Google Maps JavaScript API for the development. The application consisted of three main parts.

1. **Google Map.** We used two proportional symbol maps, one for the United States and the other for Syria, as shown in Figures 16 (a) and (b). SVG circles that varied in size were used to visualize the number of people in a particular region. Hovering over the SVG circles generated a tooltip with the following information:



(a)



(b)

Figure 16. (a) a map of the United States showing the Syrian–American Business Directory from 1908–1909 (b) a map of Syria showing the Syrian–American Business Directory from 1908–1909

For the U.S. Map:

Location. A count of the number of people in the given location.

Origin. The different locations in Syria where the immigrants migrated from, followed by the number of immigrants, followed by their professions

For the Syrian Map:

Migration. The different places in the U.S. where the immigrants migrated to, followed by the number of immigrants, followed by their professions.

2. **Drop-down menu.** The application included two drop down menus to filter the data being shown on the maps. One menu applied to the places of origin (e.g., Aleppo, Beirut, Damascus, and so on) and the other applied to profession (e.g., dry goods, rugs, silks, and so on). If a user selected any of these options, for example, Damascus, then the U.S. map would only show Syrian–American businesspersons who had migrated from Damascus and the Syrian Map would only show people from Damascus who had migrated to the U.S. Similarly, if a user selected Dry Goods from the profession menu, then only businesspersons who sold dry goods would be shown on the maps. Users could filter the data using a combination of both place of origin and profession to focus on specific information.
3. **Text box.** The application provided a text box, which reported the total number of people being shown on the map at any instance in time.

Based on anecdotal feedback from Dr. Khater and his associates, the web application proved to be very useful, allowing numerous inferences to be drawn by examining the maps. Examples included where higher concentrations of immigrants came from, and where in the United States the majority of the immigrants settled. Hovering over the SVG circles gave additional, more detailed information about the places of origin and professions of the people living in a particular area. The drop down menus assisted in focusing on specific details. For example, if users wanted to see how many doctors there were in the U.S. who had migrated

from Damascus, they could very easily obtain that information within the application. The application also helped users determine answers to questions like “What are the most and least practiced professions for people who have migrated from Syria to U.S.?” Since the application was made publically accessible over the web, it proved immensely helpful for people who wanted to know more about Syrian–American businesspersons who migrated to the United States.

4.2 Wildfire Application Design

The web application we designed for visualizing wildfire tweets was inspired by the Syrian Migration application, and was developed using similar technologies. In this domain, we required a more flexible interface, which we implemented using the jQuery UI library. Our web application consists of the following main components:

1. **Google Map.** We used a Google Map to position geo-tagged tweets about wildfires, identified with our classification algorithm. SVG circles on the map indicated the location of each tweet based on its latitude and longitude, as shown in Figure 17 below:

Figure 17. Google Map showing the location of four geo-tagged tweets represented by the SVG circles

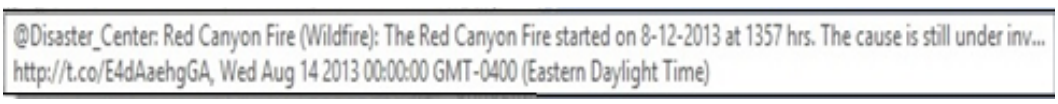


Figure 18. Text displayed when we mouse-over an SVG circle on the map

When we mouse-over any of these SVG circles, text similar to Figure 18 is shown in a tooltip.

The dialog text has the following format:

Twitter Handle. The Twitter ID of the tweet's author.

Tweet. The body of the tweet.

Date and Time. The datestamp when the tweet was posted.

2. **Time Range Slider.** We provided a double-ended slider for filtering tweets based on a user-chosen date range. This slider helps users narrow or widen the number of tweets being displayed on the map, for example, to focus on information regarding a particular wildfire that occurred over a specific range of dates, or to investigate all wildfires that occurred over a particular time span. Figure 19 provides an example of the slider, set to a particular date range.

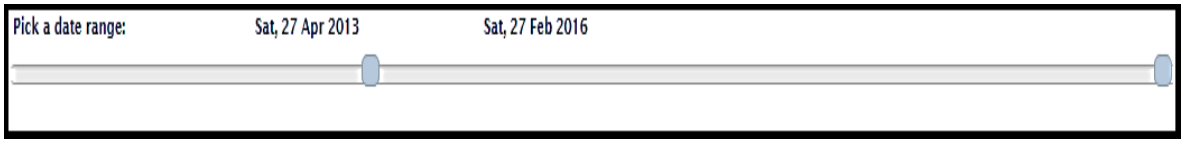


Figure 19. Double-ended slider with a date range from Saturday, 27th April 2013 to Saturday, 27th February 2016

3. **Keyword Filter.** In our web application we provide an autocomplete input box and corresponding Query button to allow users to further filter tweets being displayed based on a constraint of containing one or more keywords. For example, if a number of wildfires happened during overlapping time spans, but users need to focus on only one of the fires, they can filter on the wildfire's name using the keyword filter field. Entering a keyword in the input box and clicking the Query button removes all the tweets that do not contain that keyword. The keyword filter allows users to control the level of detail at which they explore wildfires for more specific informations. Figure 20 below demonstrates how the autocomplete input box functions.

Figure 20. The autocomplete input box and Query button used for keyword filtering

4. **Number of tweets.** The current number of tweets being displayed at any instance in time is overlaid on the map, as shown in Figure 21.

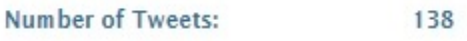


Figure 21. The number of tweets displayed on the map (currently 138)

4.3 Google Maps

We used the Google Maps JavaScript API to represent geo-tagged wildfire tweets. To begin using the API we first need to obtain an API key for our application from the Google Developers site. This is needed to authenticate loading the Google Maps API, using a script tag like the following:

```
<script
  type="text/javascript"
  src="https://maps.googleapis.com/maps/api/js?key=OUR_API_KEY&sensor=false">
</script>
```

Figure 22. Example showing a call to the Google Maps API

The URL contained in the script is the location of the JavaScript file that loads all of the symbols and definitions needed for using the Google Maps API. The key parameter contains our application's API key.

We synchronously load the API, so our application does not block as the API is downloaded, avoiding halting the application. Once the API is loaded, we display the map in a DIV location defined in the HTML for the web page containing the map. A reference to this element in the browser's document object model (DOM) is provided by the DIV's id parameter, as shown below.

```
<div id="map-canvas"
    style="height: 98.7%;width: 99.7%;z-index: 0;position:absolute;">
</div>
```

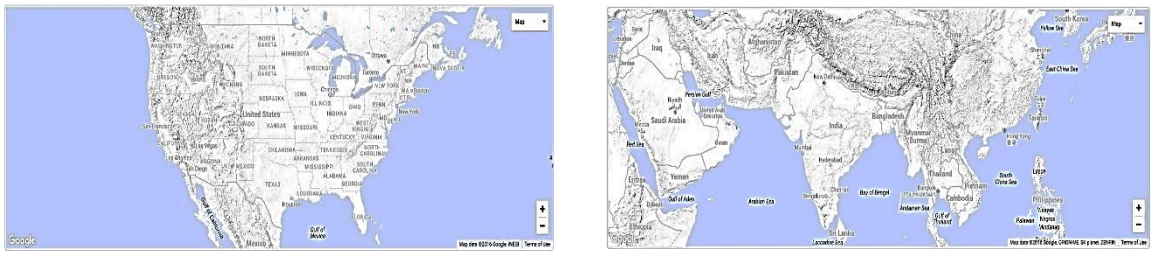
Figure 23. Example showing how to display the Google Map inside a div element

Map options and controls are used to set and control the positioning and the look of the map. The JavaScript code snippet in Figure 24 shows how we can set some of these options.

```
var  opt = {          // Map options
       center: new google.maps.LatLng( 38.267, -97.806 ),
       zoom: 4,
       mapTypeControlOptions: {
         mapTypeIds: [
           google.maps.MapTypeId.ROADMAP,
           google.maps.MapTypeId.TERRAIN,
           google.maps.MapTypeId.HYBRID,
           google.maps.MapTypeId.SATELLITE
         ],
         position:
           google.maps.ControlPosition.TOP_RIGHT,
         style:
           google.maps.MapTypeControlStyle.DROPDOWN_MENU
       },
       mapTypeId: google.maps.MapTypeId.TERRAIN,
       panControl: false,
       streetViewControl: false
     };
```

Figure 24. JavaScript code snippet showing how to set the various map options

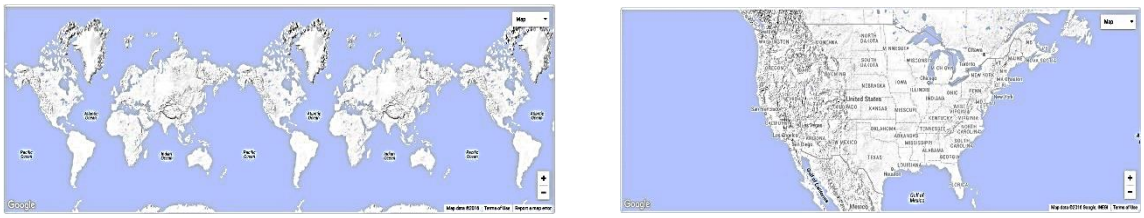
There are two required map options for every map: *center* and *zoom.* The center property displays the appropriate part of the map by setting the latitude and longitude for the center of the display region. For example, if we set the latitude and the longitude of the center of the map to (38.267° N, 97.806° W), the map would display the United States, Figure 25 (a), whereas a center of (22.5726° N, 88.3639° E) would display India and its surrounding countries, Figure 25 (b).



(a) (b)

Figure 25. (a) Map with center at 38.267° N, 97.806° W (b) Map with center at 22.5726° N, 88.363° E

The zoom property is used to set the zoom level of the map. Figures 26 (a) and (b) show the same map with zoom levels of 1 and 4, respectively.



(a) (b)

Figure 26. (a) Map at zoom level 1 (b) Map at zoom level 4

The maps displayed through the Google Maps JavaScript API contain UI elements to allow user interaction with the map. These elements are known as *controls.* We included a subset of these controls in our application. Alternatively, we could do nothing and let the Google Maps JavaScript API use its default control behavior. The controls that we used selected for our application were:

- The *Zoom control*, which displays "+" and "-" buttons for changing the zoom level of the map. This control appears by default in the bottom right corner of the map.
- The *Map Type control*, available in a dropdown or horizontal button bar style, allowing the user to choose a map type (ROADMAP, SATELLITE, HYBRID, or TERRAIN). This control appears by default in the top left corner of the map, but we moved it to the top right and chose a dropdown style and gave the user the option to choose a map type of SATELLITE or TERRAIN, as shown in Figure 27.

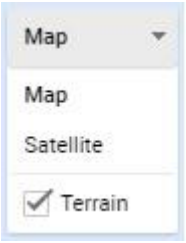


Figure 27. Map type control

- The *Street View control* presents a Pegman icon that can be dragged onto the map to enable Street View. This control appears by default near the bottom right of the map. In our application we did not require this functionality, so we disabled this control.

We modified the map's MapOptions fields to access and modify the map controls to set the visibility and presentation of the controls. We adjusted these controls during map instantiation and by using appropriate API map options and controls.

In our web application, we use a styled map to apply colors and changes to the default map provided by the Google Maps API. The two main concepts that we use to do this were:

- **Map features.** Features are geographic elements that can be included on the map, for example, roads, parks, ad bodies of water, as well as their labels.

- **Stylers.** Stylers control the color and visibility of map features, defined through a combination of hue, color, and lightness/gamma values.

Map features and stylers are combined into a style array, which is passed to the default map's MapOptions object as shown in the JavaScript code snippet in Figure 28.

```
    map.set( "styles", [
  {
    featureType: "landscape",     // Simplified, white landscape
    elementType: "geometry",
    stylers: [ {
      color: "#ffffff",
      visibility: "simplified"
    } ]
  },

//  Turn off all administrative features, then selectively turn some
//  text and boundaries back on

  {
    featureType: "administrative",
    elementType: "all",
    stylers: [ {
      visibility: "off",
    } ]
  },
  {
    featureType:          // Restore country names
      "administrative.country",
    elementType: "text",
    stylers: [
      { lightness: 70 },
      { visibility: "on" }
    ]
  },
```

Figure 28. JavaScript code snippet showing the style array consisiting of features and stylers

We used eight different features and their respective stylers on our map. These features are as follows:

Table 5. Table of Map features and Stylers

Map Features	Stylers
Landscape	We set the landscape to “simplified” and set its color to white.
Administrative	We turned off all the administrative features like countries, states, cities, etc.
Administrative Country	We then reset the counties (turned off in the previous step) to be visible.
Administrative Province	We reset the states and provinces.
Administrative Locality	We reset the large city names.
Water	We set the lightness and saturation of the water to make it pale blue.
Points of Interest	We turned off all points of interest.
Roads	We turned off all roads.
Transit	We turned off all transit.

We used the Google Maps API’s marker option to show the geo-tagged tweets on the map. The google.maps.Marker constructor takes a single Marker options object literal, specifying the initial properties of the marker. The fields that we set while constructing the markers were:

Table 6. The different marker options and their uses

Fields	Uses
Position (required)	We used this to specify the latitude and longitude of the location of the marker.
Map (optional)	We used this to specify the particular map on which the marker should be placed.
Icon (optional)	By default a marker uses a standard image. In our case, we used the google.maps.SymbolPath.CIRCLE.
Title (optional)	The marker's title appears as a tooltip. We used this field to set information shown about the tweet when a user hovered over the tweet's SVG circle.

4.4 jQuery

We used the jQuery JavaScript API to create the interface for our web application. This API provides a theme gallery the defines the look-and-feel of the widgets and other elements of the jQuery UI (our application uses the theme "Cupertino"). Each theme consists of the same sets of widgets, utilities and effects but with different colors, backgrounds and visual appearances. Based on the jQuery theme, we developed our application using it as a foundation. The main widgets we used in our application have been discussed in the Design section.

The data to be visualized must be stored and loaded into our application. We hosted our data on a Google Spreadsheet inside Google Drive. The interaction between the user interface and the data happens through the Google Visualization API. The interaction is a two-step process:

1. **Queries.** To send a request for data to the Google spreadsheet we instantiate a Query object with the URL of our spreadsheet. When we send the query we specify a callback handler that will be invoked when the requested data is received as shown in Figure 29.

```
q = new gocgle.visualization.Query("https://docs.google.com/spreadsheets/d/URL_Of_Google_Spreadsheet" );
q.setQuery( "SELECT B,C,D,F,H WHERE E = " + 1 + " AND todate(H) >= date " + "'" + sDate + "'" + "AND todate(H) <= date " +
q.send( viz_pop );
```

Figure 29. JavaScript code snippet showing instantiation of a Query object with a callback handler

2. **Processing.** The response handler is called when the data request returns. A google.visualization.QueryResponse parameter is passed into our handler function. If the request was successful, the response contains a data table (a google.visualization.DataTable object). If the request failed, the response contains information about the error. The data table included in the response can be retrieved using the getDataTable() function.

In our application, the query sent to the Google spreadsheet is modified based on the date range slider and any keywords included in the filter text field. After we retrieve the data, we count the number of rows and display the result as the number of tweets currently being visualized.

If a user starts typing in the input box, the autocomplete feature suggests different options in the form of a dropdown that the user can choose from. Once the user hits the Query button (Figure 20), the button's onclick event issues a new query to reinitialize the Google Map with the query's results. A similar query is built and sent when the user changes either handle of the date slider.

The main function being invoked to connect to the database, retrieve data, and visualize the results on the Google Map is shown below.

```
function init_map() {
  var  q;
  q = new google.visualization.Query("https://docs.google.com/spreadsheets/d/URL_Of_Google_Spreadsheet" );
  var searchString2 = document.getElementById('autocomplete').value.replace(/'/g, "\\'");
  var startDate = new Date(document.getElementById('event-start').innerHTML);
  console.log(startDate);
  var sDate = startDate.getFullYear() + "-" + (startDate.getMonth()+1) + "-" + startDate.getDate();
  console.log(sDate);
  var endDate = new Date(document.getElementById('event-end').innerHTML);
  var eDate = endDate.getFullYear() + "-" + (endDate.getMonth()+1) + "-" + endDate.getDate();
  console.log(eDate);
  if(searchString2 == "")
  { q.setQuery( "SELECT B,C,D,F,H WHERE E = " + 1 + " AND todate(H) >= date " + "'" + sDate + "'" +
                "AND todate(H) <= date " + "'" + eDate + "'" + "ORDER BY C,D");
    q.send( viz_pop );}
  else if(searchString2 != "")
  { q.setQuery( "SELECT B,C,D,F,G WHERE E = " + 1 + " AND F LIKE " + "'%" + searchString2 + "%'" +
                " AND todate(H) >= date " + "'" + sDate + "'" + "AND todate(H) <= date " + "'" + eDate + "'" + "ORDER BY C,D" );
    q.send( viz_pop );}
}
```

Figure 30. JavaScript code snippet to initialize/reinitialize the Google Map

In the above code snippet, ***searchString2*** fetches values from the input box when the Query button is clicked. ***startDate*** and ***endDate*** fetch the values of the handles of the date range slider, and then formats them into ***sDate*** and ***eDate*** respectively to match the date format of the wildfire database. Based on whether ***searchString2*** is empty or not two different queries are fired.

1. ***searchString2*** empty. In this case we query the columns B (Author ID), C (Latitude), D (Longitude), F (Tweet Body), G (Datestamp), and E (wildfire = 1, other = 0). We visualize tweets where column E (wildfire) is 1 for all tweets that fall within ***sDate*** to ***eDate*** inclusive.
2. ***searchString2*** non-empty. In this case we query the columns B (Author ID), C (Latitude), D (Longitude), F (Tweet Body), G (Datestamp), and E (wildfire = 1, other = 0). We visualize tweets where column E (wildfire) is 1 and column F (Tweet Body) contains the value of ***searchString2***, for tweets with post dates that fall within ***sDate*** to ***eDate*** inclusive.

CHAPTER 5

Results

This thesis consists of two major parts: *classification* and *visualization*, as discussed in the earlier sections of this document. The results that we achieved in our classification task are summarized here and the results that we obtained from the web application built for our visualization task are presented in Section 5.1.

Classification results:

Table 7. Summary of the classification

Method Used	Multinomial naïve Bayes
Training and Testing Method Used	10–fold cross validation
Total number of tweets	3.1 million
Number of geo-tagged tweets	28,772 tweets
Size of training set	519 tweets (Wildfire tweets: 281, Other tweets: 238)
Accuracy of the classifier on the training set	89.42 %
Number of geo-tagged tweets classified as "Wildfire tweets"	10,158
Number of geo-tagged tweets classified as "Other (non-wildfire) tweets"	18,613

We use absolute accuracy, as well as precision and recall, to characterize the performance of our classification results.

- **True Positive (TP).** Tweets classified as wildfire, and about wildfires.
- **False Positive (FP).** Tweets classified as wildfire, but about other.
- **True Negative (TN).** Tweets classified as other, and about other.
- **False Negative (FN).** Tweets classified as other, and about wildfires.

- **Precision.** $\frac{TP}{TP+FP}$, the percentage of tweets classified as a wildfire that were actually about wildfires.
- **Recall.** $\frac{TP}{TP+FN}$, the percentage of all wildfire tweets that were correctly classified as about a wildfire.

Table 8. Precision, recall, and accuracy results for the multinomial Naïve Bayes classifier

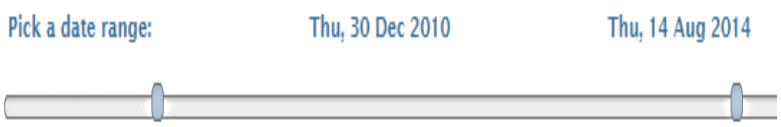
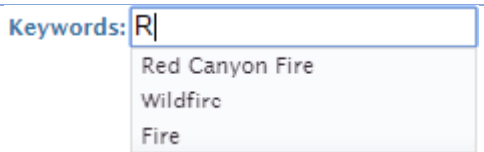
Test Set	TP	FP	TN	FN	Precision	Recall	Accuracy
subset 1	17	3	31	0	0.85	1.0	94.12%
subset 2	29	2	19	1	0.94	0.97	94.12%
subset 3	20	5	25	1	0.80	0.95	88.24%
subset 4	25	4	20	2	0.86	0.93	88.24%
subset 5	5	13	31	2	0.28	0.71	70.59%
subset 6	24	2	25	0	0.92	1.0	96.08%
subset 7	19	6	26	0	0.76	1.0	88.24%
subset 8	38	4	8	1	0.90	0.97	90.20%
subset 9	45	1	3	2	0.98	0.96	94.12%
subset 10	42	0	4	5	1.0	0.89	90.20%
Overall Results:					**0.83**	**0.94**	**89.41%**

Based on the precision, recall, and accuracy results, we concluded that the classifier was performing well. In 89.41% of the cases, our algorithm correctly classified a wildfire tweet as an actual wildfire.

5.1 Images

This section is focused on the various widgets of our web application. Table 9 shows the images of each of them.

Table 9. Images of the widgets in our web application

Widgets	Images
Google Map	
SVG Circle to represent tweets on the map	
Text Box that appears when we hover over a circle	
Number of tweets	
Date range information and Date range slider	
Empty Autocomplete Input box	
Autocomplete Input box when some text is typed in it	
Query Button	

5.2 Web Application Demonstration

Here we provide a brief example of using our web application to identify relevant results. These are the steps that we followed to explore different wildfires. It is not necessary to use the same sequence of steps, or to investigate the same issues we use in our demonstration. Any of the features of the web application can be accessed independently, and any set of features can be combined to produce more focused results.

1. Figure 31 shows the web application when it is initially loaded. The autocomplete text box is empty and the date range slider is set from Saturday, 11 May 2013 to Sat, 6 February 2016. We pre-selected this date range because all our tweets fall within this range. The number of tweets being visualized is 10,158, which is the same number of wildfire tweets that we identified after classification.

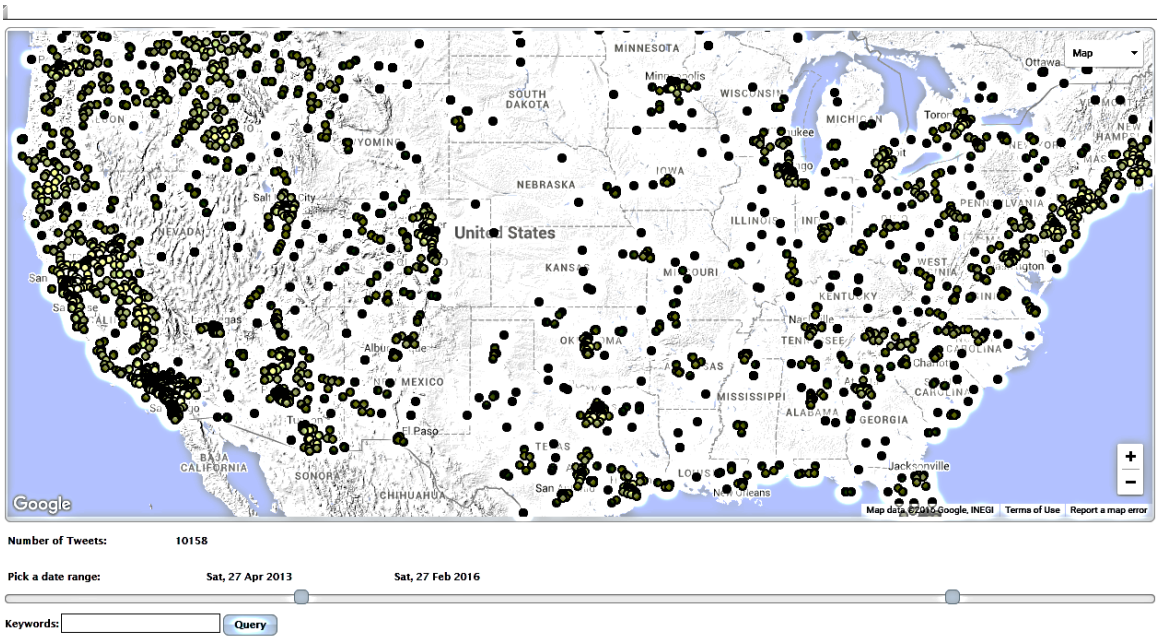


Figure 31. Our web application when it is initially loaded

2. In our example, we want to examine tweets about fires that occurred over a one-year period. We select a date range of 28th December 2013 to 27th December 2014 using the slider. The Google Map reloads and visualizes all tweets within this date range (Figure 32). Now, we see 3,836 tweets on the map.

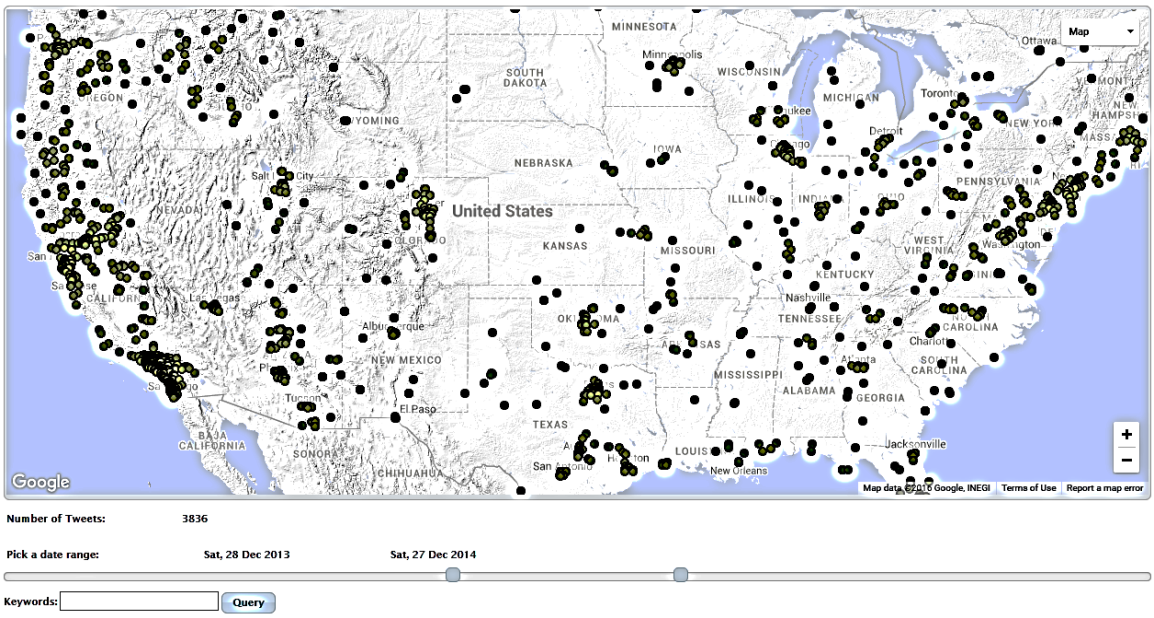


Figure 32. Web application once the date range [28th December 2013, 27th December 2014] is selected

3. Next, we decide to examine all the wildfires that happened in California during this period (28th December 2013, 27th December 2014). Therefore, we filter the tweets by entering “California wildfire” in the autocomplete input box. After filtering, we retain 210 tweets shown in Figure 33.

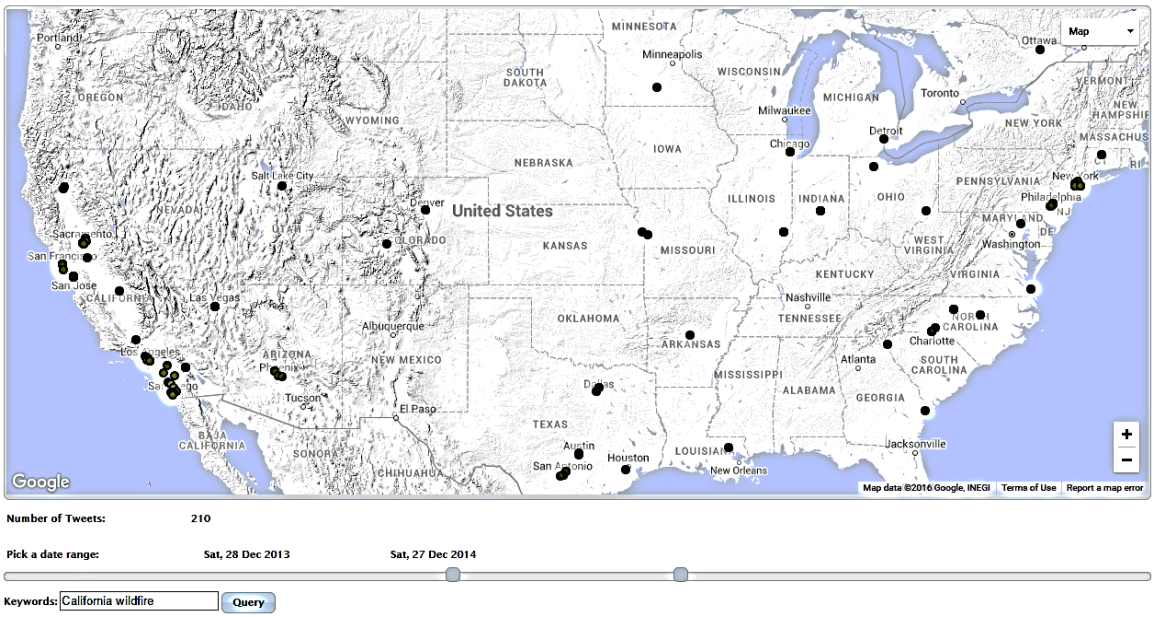


Figure 33. Tweets about wildfires in California in 2014

4. Next, we decide to examine the May 2014 San Diego County wildfires, a swarm of fourteen wildfires that erupted during May 2014 in San Diego County during severe Santa Ana Wind conditions, historic drought conditions, and a heat wave. The main event during mid-May was preceded by a precursor fire that ignited on May 5. These fires burned from May 5, 2014–May 22, 2014, growing to a size of approximately 26,000 acres. This wildfire's date range was selected to restrict the visualization to tweets in the appropriate timeframe. Additional tweets were filtered by entering "San Diego County" in the autocomplete input box. The Google Map reloads and shows only the tweets with the keywords San, Diego and County that were posted from May 5, 2014 to May 22, 2014, inclusive. As we can see in the Figure 34, there are 14 tweets about this fire. We can see by hovering over various tweet circles that the majority of the tweets are indeed about the San Diego County wildfires.

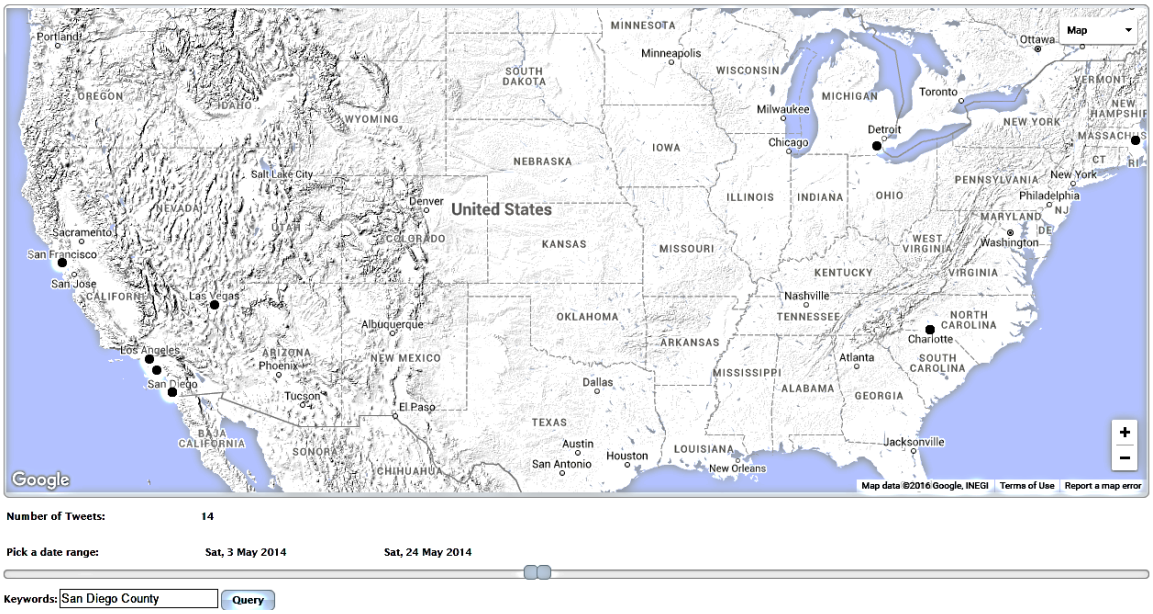


Figure 34. Web application following keyword filtering on "San Diego County" along with the date range [5th May 2014, 22nd May 2014] selection

5.3 Examples

Table 10 identified two of the significant fires that happened during 2013 to 2015.

Table 10. Short list of wildfires that happened between 2013 to 2015

Wildfire Name	State	Spread (Acres)	Started On	Burned Till
Valley Fire	California, USA	76,067	September 12, 2015	October 15, 2015
Black Forest Fire	Washington, USA	14,280	June 11, 2013	June 20, 2013

Valley Fire. In our initial dataset, we manually identified 20 tweets from the 28,771 in the dataset that contained the words "Valley Fire". Out of these tweets, one was about the Meadow Valley fire (2013) and two others were about the Rye Valley Fire (2013). The remaining 17 were about the Valley Fire mentioned in Table 10. Our classifier correctly classified all of these 20 tweets as "wildfire". Then when we tried to explore Valley Fire

tweets through our web application. Initially we identified the 20 tweets, shown in Figure 35(a). Then we filtered by the date range of the Valley Fire to identify the 17 Valley Fire tweets (Figure 35 (b)).

(a)

(b)

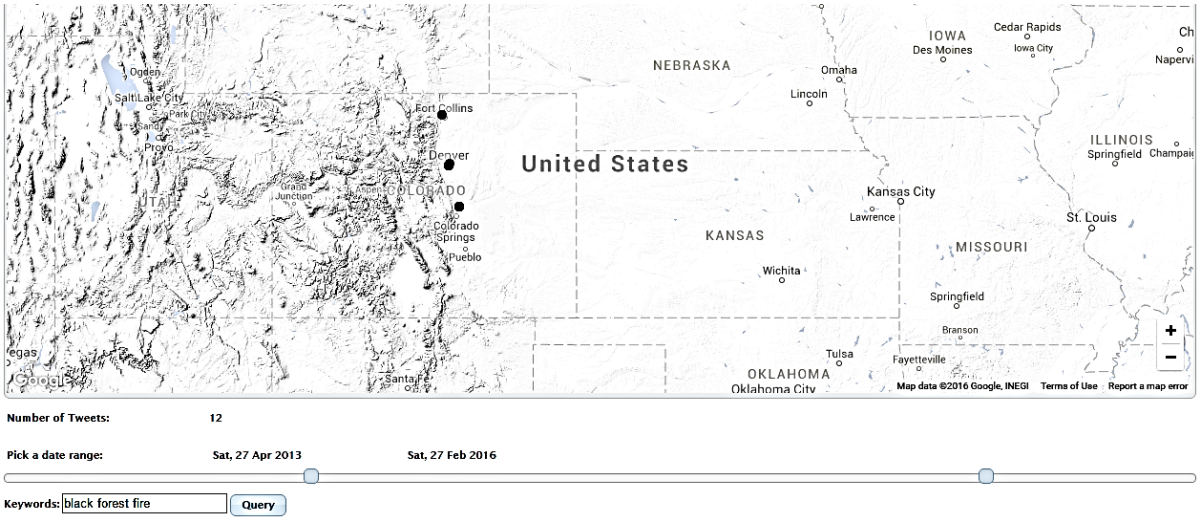
Figure 35. (a) Valley Fire without date range filtering (b) Valley Fire with date range filtering

Table 11 contains snapshots of the some of the tweets shown by the web application.

Table 11. Snapshot of the tweets about "Valley Fire"

@SanFranciscoCP: Valley Fire Now 3rd Most Destructive Wildfire In California History: Ten days after it started, the Valley Fireâ€¦ http://t.co/o9gN80vi7T, Tue Sep 22 2015 00:00:00 GMT-0400 (Eastern Daylight Time)
@SanFranciscoCP: Valley Fire 69 Percent Contained As Firefighters Gain Ground; 888 Homes Burned: A wildfire that has killed three,â€¦ http://t.co/E4XO16AZIK, Mon Sep 21 2015 00:00:00 GMT-0400 (Eastern Daylight Time)
@SanFranciscoCP: Return Of Hot Weather Could Flare Up Valley Fire Flames: Â As crews increase their chokehold on a deadly wildfireâ€¦ http://t.co/WngoC9oosn, Fri Sep 18 2015 00:00:00 GMT-0400 (Eastern Daylight Time)
@SanFranciscoCP: Valley Fire: 70,000 acres torched, 35% contained: The fight continues on day 6 of the massive wildfire that hit theâ€¦ http://t.co/f8jCivNlyF, Thu Sep 17 2015 00:00:00 GMT-0400 (Eastern Daylight Time)
@SanFranciscoCP: Valley Fire Ranked 9th Most Damaging Wildfire Ever In State: The 70,000-acre Valley Fire in Lake, Sonoma and Napaâ€¦ http://t.co/...

Black Forest Fire. In our initial dataset, we manually identified 12 tweets from the 28,771 that contained the words “Black Forest Fire” in them. Our classifier correctly classified all of these tweets as “wildfire”. We then visualized Black Forest Fire tweets through our web application to show the 12 Black Forest Fire tweets (Figure 36(a)). When we filtered by the date range of the Black Forest Fire we only saw six tweets (Figure 36 (b)), suggesting that discussion of the Black Forest Fire continued after the fire was contained.



(a)

(b)

Figure 36. (a) Black Forest Fire without date range filtering (b) Black Forest Fire with date range filtering

Table 12 contains the snapshots of some of the tweets as shown by the web application for the Black Forest Fire.

Table 12. Snapshot of the tweets about Black Forest Fire

@Disaster_Center: Black Forest Fire (Wildfire): The Black Forest Fire started on Tue, June 11th. The cause is undertermined.... http://t.co/0TXG4NLzGR, Wed Jun 19 2013 00:00:00 GMT-0400 (Eastern Daylight Time) @Disaster_Center: Black Forest Fire (Wildfire): The Black Forest Fire started on Tue, June 11th. The cause is undertermined.... Error, Tue Jun 18 2013 00:00:00 GMT-0400 (Eastern Daylight Time)
@dougp01: \'@850KOA:Largest wildfire is the Black Forest fire north of Colorado Springs. http://t.co/OaFBhj60Oa\' Largest? Don't forget #RoyalGorgeFire, Wed Jun 12 2013 00:00:00 GMT-0400 (Eastern Daylight Time)
@damon9851: Everyone please say a prayer for the many families that have lost there homes in the Black Forest fire in Colorado...huge wildfire...awful, Fri Jun 14 2013 00:00:00 GMT-0400 (Eastern Daylight Time)
@DenverCP: Black Forest Fire has destroyed â€˜as many as 100â€² homes: EL PASO COUNTY, Colo. â€" A fast-moving wildfire in the... http://t.co/9X5JPh0LAP, Wed Jun 12 2013 00:00:00 GMT-0400 (Eastern Daylight Time) @DenverCP: Black Forest Fire burning in El Paso County: EL PASO COUNTY, Colo. â€" A 15-acre wildfire is burning in El Paso... http://t.co/enju9CbtwB, Tue Jun 11 2013 00:00:00 GMT-0400 (Eastern Daylight Time)

CHAPTER 6

Conclusions and Future Work

This thesis describes an attempt to classify tweets from the microblogging social network Twitter into “wildfire” and “other” (non-wildfire). Approximate 3.1 million tweets containing the keywords “wildfire” or “forest service” were collected, resulting in 28,772 tweets with geolocations. All 28,772 tweets were classified, producing 10,158 wildfire tweets that were then visualized using a Google Map web application. Classification was performed using the Scikit-learn [9] Python multinomial naïve Bayes classifier. The web application was developed using SVG, HTML, JavaScript, jQuery, and the Google Maps JavaScript API. Since the application runs on a web browser, it is accessible from desktop, tablet, and mobile devices. The application provides keyword filtering and data range selection to help to focus on specific wildfires by restricting tweets based on a wildfire’s name or the time it was active.

6.1 Limitations

We encountered a number of the limitations while performing the classification and visualization tasks. Although the tweet database contains 3.1 million records, we were only able to use 28,771 of them for the classification and visualization tasks. This is because attaching a geolocation to a tweet is an opt-in choice by a user. Normally, only about 1% of tweets that are posted contain a geolocation. Correspondingly, slightly less than 1% of the data that we collected included the latitude and longitude of the location where the tweet was posted.

For web application, it is still difficult to get an ideal browsing experience across different web browsers. The speed with which the visualization displays depends on a number of factors like the amount of data being visualized, network speed, the computational capacity of the device, and so on. Screen size also affects the visualization significantly. For

example, if the screen size is small it can be difficult to view details of the visualization accurately.

6.2 Future Work

We plan to extend out work by providing ways to estimate the geolocations of additional tweets. This would improve the analysis, since we would be able to use more of the 3.1 million tweets in our classification and visualization tasks.

We would also like to provide a better classification of the tweets, as well as explore how we can extend and generalize our work to data from other social networks.

We plan to improve our web application by making it more scalable, so that it can handle larger amounts of data. Features like tracing the path of a wildfire through time, showing trends of different wildfires, for example in regions where wildfires occur frequently or during times when most wildfires occur, or providing possible reasons about why the particular wildfires started could improve the analysts' exploration experience.

REFERENCES

[1] A. Storkey, "Learning from Data: Decision Trees," School of Informatics, University of Edinburgh, 2004.

[2] M. G. Kendall, A. Stuart and J. K. Ord, Kendall's Advanced Theory of Statistics, Volume 1, Distribution Theory, Oxford University Press, 1994.

[3] P.-N. Tan, M. Steinbach and V. Kumar, Introduction to DATA MINING, Pearson, 2006.

[4] "The University of Edinburgh," [Online]. Available: http://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn-note07-2up.pdf.

[5] A. W. Moore, "Support Vector Machines," 23 November 2001. [Online]. Available: http://www.csee.umbc.edu/courses/pub/WWW/courses/graduate/678/spring14/svm.pdf.

[6] P. Winston, *MIT 6.034 Artificial Intelligence, Fall 2010,* MITOPENCOURSEWARE, 2010.

[7] "Open Source Computer Vision Library," Open Source Computer Vision Library, [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html.

[8] S. González, J. Zaefferer , F. Nagel, M. Sherov, R. X. de Souza and et al., "jQuery user interface," [Online]. Available: https://jqueryui.com/demos/.

[9] Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825--2830, 2011.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel *et al.*, "Scikit-learn: Machine Learning in Python," 1 February 2010. [Online]. Available: http://scikit-learn.org/.

[11] "Clever Owl," 1 7 2015. [Online]. Available: http://cleverowl.uk/2015/07/01/using-one-way-anova-and-tukeys-test-to-compare-data-sets/.

[12] H. Deng, G. Runger and E. Tuv, "Bias of importance measures for multi-valued attributes and solutions," in *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN)*, 2011.

[13] L. Hyafil and R. Rivest, "Constructing Optimal Binary Decision Trees is NP-complete," *Information Processing Letters,* vol. 5, 1976.

[14] D. I. C. Mihaescu, "Software Engineering Department, Central University of Venezuela," 07 05 2010. [Online]. Available: http://software.ucv.ro/~cmihaescu/Lab4-NaiveBayes.pdf.

[15] A. Gut, Probability: A Graduate Course, New York, NY: Springer International Publishing AG, 2013.

[16] "Stanford University," [Online]. Available: https://web.stanford.edu/class/cs124/lec/naivebayes.pdf.

[17] S. Bird, E. Loper and E. Klein, Natural Language Processing with Python, O'Reilly Media, 2009.