A Project Report

On

**SMS SPAM DETECTION**

Submitted in partial fulfillment of the

Requirements for the award of degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

by

**V. Akhil Reddy,**

**(15H61A05P9)**

**D. Sri Charan,**

**(15H61A05P0)**

**D. Naga Sharma**

**(15H61A05K4)**

**Under the Guidance of**

**Mr. Pallam Ravi,**

**Assistant Professor, Department Of CSE**



**Department of Computer Science and Engineering**

**ANURAG GROUP OF INSTITUTIONS**

**(Formerly CVSR College of Engineering)**

**(An Autonomous Institution, Approved by AICTE and NBA Accredited)**

**Venkatapur (V), Ghatkesar (M), Medchal(D)., T.S-500088**

**(2014-2018)**

****

**CERTIFICATE**

This is to certify that the project entitled **“SMS Spam Detection”** being submitted by **V.Akhil Reddy** bearing the Hall Ticket number **15H61A05P9** and **D.Sri Charan** bearing the Hall Ticket number **15H61A05P0** and **D.Naga Sharma** bearing the Hall Ticket number **15H61A05K4** in partial fulfillment of the requirements for the award of the degree of the **Bachelor of Technology** in **Computer Science and Engineering** to **Anurag Group of Institutions** **(Formerly** **CVSR College of Engineering)** is a record of bonafide work carried out by them under my guidance and supervision from June 2018 to Nov 2018.

The results presented in this project have been verified and found to be satisfactory. The results embodied in this project report have not been submitted to any other University for the award of any other degree or diploma.

**Internal Guide ExternalExaminer Mr.Pallam Ravi,**

**Assistant Professor**

**Project Coordinator Dr.G.Vishnu Moorthy,**

**Mr A.Obulesh, Dy. Director and Professor,**

**Assistant Professor Dept. of CSE**

**ACKNOWLEDGEMENT**

It is our privilege and pleasure to express profound sense of respect, gratitude and indebtedness to our guide **Mr.Pallam Ravi, Assistant Professor**, Dept. of Computer Science and Engineering, Anurag Group of Institutions (Formerly CVSR College of Engineering), for his indefatigable inspiration, guidance, cogent discussion, constructive criticisms and encouragement throughout this dissertation work.

We express our sincere gratitude to **Dr.G.Vishnu Murthy, Professor & Head**, Department of Computer Science and Engineering, Anurag Group of Institutions (Formerly CVSR College of Engineering), for his suggestions, motivations and co-operation for the successful completion of the work.

We extend our sincere thanks to **Dr.K.S.Rao, Director**,**Dr.V.Vijaya Kumar, Dean, Dept. of CSE & IT,** Anurag Group of Institutions (Formerly CVSR College of Engineering), Venkatapur (V), Ghatkesar (M), Medchal (Dist), for their encouragement.

V.Akhil Reddy (15H61A05P9),

D.Sri Charan (15H61A05P0),

D.Naga Sharma (15H61A05K4)

**DECLARATION**

We hereby declare that the project work entitled “ **SMS Spam Detection**” submitted to the **Anurag Group of Institutions(Formerly CVSR College of Engineering)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology (B.Tech)** in Computer Science and Engineering is a record of an original work done by us under the guidance of **Mr.Pallam Ravi, Assistant Professor** and this project work have not been submitted to any other university for the award of any other degree or diploma.

**V.Akhil Reddy (15H61A05P9),**

**D.SriCharan (15H61A05P0),**

**D.NagaSharma (15H61A05K4)**

Date:

**ABSTRACT**

In recent times, as the craze of mobile phone devices has increased, Short Message Service (SMS) has emerged into a multi-billion dollars business. Nevertheless, reduction in the cost of SMS has evolved in expansion in mercantile advertisements (spams) being sent to mobile phones. In few countries of Asia, thirty percent of text messages were spam in 2012. Shortfall of original databases for SMS spams, short length of messages and limited features, and their informal language are the factors that may cause the established email filtering algorithms to underperform in their classification. In this project, a database of real SMS Spams from UCI Machine Learning repository is used, and after preliminary processing and feature extraction, various machine learning techniques are applied to the database available. At last, the results are contrasted and the best algorithm for spam filtering for text messaging is launched. This reduce reduces the overall error rate. Among the wide range of technical measures, Naive Bayes and Random Forest filters are playing a key role in stopping email spam. In this paper, we analyze to what extent Naive Bayes and Random Forest filtering techniques used to block email spam, can be applied to the problem of detecting and stopping mobile spam. In particular, we have built SMS spam test collections of significant size, in English. We have tested on them a number of messages representation techniques and Machine Learning algorithms, in terms of effectiveness. Our results demonstrate that Naive Bayes (NB) and Random Forest (RF) filtering techniques can be effectively transferred from email to SMS spam.

**CONTENTS**

|  |  |
| --- | --- |
| **TITLE** | **PAGE NO** |
| 1. Introduction  1.1 Types of learning  1.2 Python  1.3 Numpy in python  1.4 Pandas in python | 7 |
| 2. Literature Survey  2.1 Predictions | 14 |
| 3. Proposed Method  3 .1 Bayesian filtering techniques  3.2 Application to mobile spam  3.3 Feature selection  3.4 Machine learning algorithms | 18 |
| 4. Code | 24 |
| 5. Output | 27 |
| 6. Result | 28 |
| 7. Conclusion | 29 |
| 8. References | 30 |

1. **INTRODUCTION**

**Machine learning** is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions.

Machine learning is closely related to computational statistics; a discipline that aims at the design of algorithm for implementing statistical methods on computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms is infeasible. Example applications include spam filtering, optical character recognition (OCR), search engines and computer vision. Machine learning is sometimes conflated with data mining, although that focuses more on exploratory data analysis. Machine learning and pattern recognition "can be viewed as two facets of the same field."

**1.1 TYPES OF LEARNING**:

There are three main types for learning:

**1.1.1 SUPERVISED LEARNING:**

Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value (also called the *supervisory signal*). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. Determine the input feature representation of the learned

**1.1.2 UNSUPERVISED LEARNING:**

We now consider unsupervised learning, where we are just given output data, without any inputs. The goal is to discover “interesting structure” in the data; this is sometimes called **knowledge discovery**. Unlike supervised learning, we are not told what the desired output is for each input. Unsupervised learning is arguably more typical of human and animal learning. It is also more widely applicable than supervised learning, since it does not require a human expert to manually label the data. Labeled data is not only expensive to acquire, but it also contains relatively little information, certainly not enough to reliably estimate the parameters of complex models.

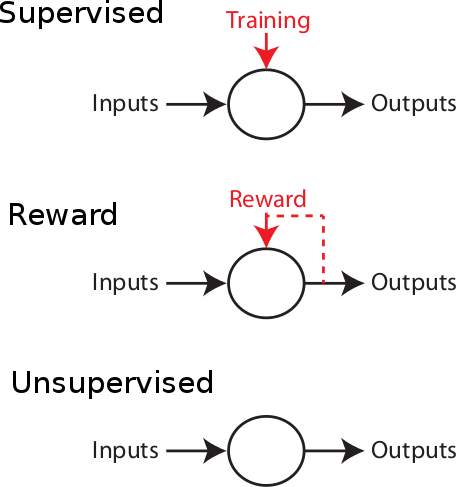


Figure 1.3.1 Types of learning

**1.1.3 REINFORCEMENT LEARNING:**

Reinforcement learning is learning what to do--how to map situations to tell which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics--trial-and-error search and delayed reward--are the two most important distinguishing features of reinforcement learning.

But to discover such actions, it has to try actions that it has not selected before. The agent has to exploitwhat it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate its expected reward.

**1.2 PYTHON:**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level programming language](https://en.wikipedia.org/wiki/High-level_programming_language) for [general-purpose programming](https://en.wikipedia.org/wiki/General-purpose_programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python has a design philosophy that emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), notably using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). It provides constructs that enable clear programming on both small and large scales. Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural](https://en.wikipedia.org/wiki/Procedural_programming), and has a large and comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [CPython](https://en.wikipedia.org/wiki/CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open_source) software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation).

Python is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language). [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of its features support [functional programming](https://en.wikipedia.org/wiki/Functional_programming) and [aspect-oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) (including by [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming) and [metaobjects](https://en.wikipedia.org/wiki/Metaobject) (magic methods)). Many other paradigms are supported via extensions, including [design by contract](https://en.wikipedia.org/wiki/Design_by_contract) and [logic programming](https://en.wikipedia.org/wiki/Logic_programming).

Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing), and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It also features dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution.

Python's design offers some support for [functional programming](https://en.wikipedia.org/wiki/Functional_programming) in the [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)) tradition. It has filter(), map(), and reduce() functions; [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension), [dictionaries](https://en.wikipedia.org/wiki/Associative_array), and sets; and [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) and [Standard ML](https://en.wikipedia.org/wiki/Standard_ML).

The language's core philosophy is summarized in the document *The*[*Zen of Python*](https://en.wikipedia.org/wiki/Zen_of_Python) (*PEP 20*), which includes [aphorisms](https://en.wikipedia.org/wiki/Aphorism) such as:

* Beautiful is better than ugly
* Explicit is better than implicit
* Simple is better than complex
* Complex is better than complicated
* Readability counts

**1.3 NUMPY IN PYTHON:**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.  
Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

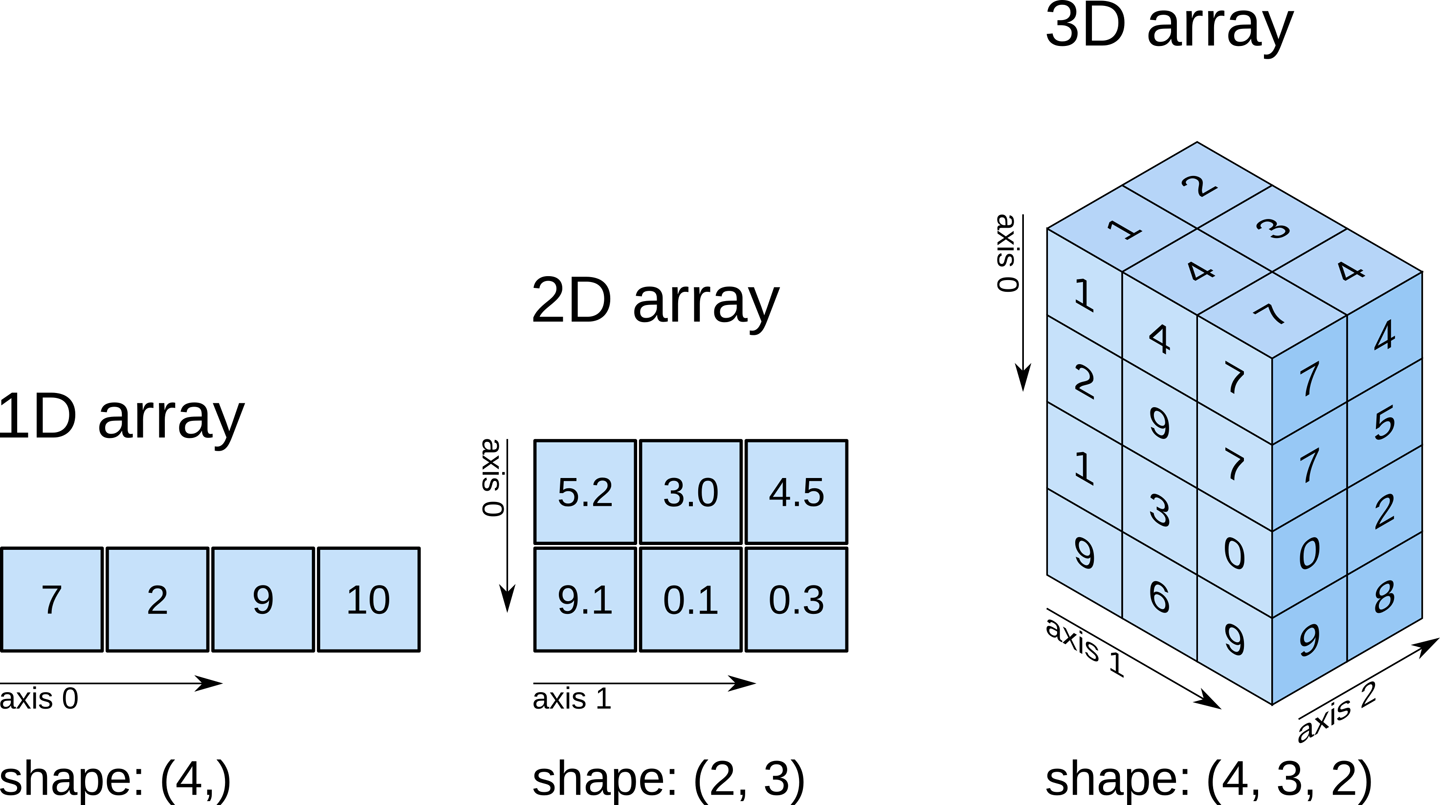


Figure 1.6.1 Different dimensional arrays

**1.4 PANDAS IN PYTHON:**

**Pandas** is a [Python](http://www.python.org/) package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

* Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
* Ordered and unordered (not necessarily fixed-frequency) time series data.
* Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
* Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, [**Series**](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html#pandas.Series) (1-dimensional) and [**DataFrame**](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, [**DataFrame**](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) provides everything that R’s data frame provides and much more. Pandas is built on top of [NumPy](http://www.numpy.org/) and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

* Easy handling of **missing data** (represented as NaN) in floating point as well as non-floating point data
* Size mutability: columns can be **inserted and deleted** from DataFrame and higher dimensional objects
* Automatic and explicit **data alignment**: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, *DataFrame*, etc. automatically align the data for you in computations
* Powerful, flexible **group by** functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
* Make it **easy to convert** ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
* Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets
* Intuitive **merging** and **joining** data sets
* Flexible **reshaping** and pivoting of data sets
* **Hierarchical** labeling of axes (possible to have multiple labels per tick)
* Robust IO tools for loading data from **flat files** (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast **HDF5 format**
* **Time series**-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.

Some other notes

* pandas is **fast**. Many of the low-level algorithmic bits have been extensively tweaked in [Cython](http://cython.org/)code. However, as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool.
* pandas is a dependency of [statsmodels](http://www.statsmodels.org/stable/index.html), making it an important part of the statistical computing ecosystem in Python.
* pandas has been used extensively in production in financial applications.

1. **LITERATURE SURVEY**

The mobile phone business has experienced a considerable extension over recent times. In the mid of 2013, a total of 432.1 million mobile phones have shipped, which exhibits a six percent year over year increase . As the usage of mobile phones has become mainstream, Short Message Service (SMS) has grown into a multi-billion dollars trade. SMS is a text communication platform that lets mobile phone users to exchange short text messages (usually less than 160 seven-bit characters). It is the most widely used data application with an estimated 3.5 billion active users, or about 80% of all mobile phone subscribers at the end of 2010. As the popularity of the platform has increased, we have seen a surge in the number of unsolicited commercial advertisements sent to mobile phones using text messaging. SMS spam is still not as common as email spam, where in 2010 around 90% of emails was spam, and in North America it is still not a major problem, contributing to less than 1% of text messages exchanged as of December 2012. However, due to increased popularity in young demographics and the decrease in text messaging charges over the years (in China it now costs less than $0.001 to send a text message), SMS Spam is showing growth, and in 2012 in parts of Asia up to 30% of text messages was spam. In middle east, some of the carriers themselves are responsible for sending out marketing text messages. Additionally, SMS Spam is particularly more irritating than email spams, since in some countries they contribute to a cost for the receiver as well. These factors along with limited availability of mobile phone spam-filtering software makes spam detection for text messages an interesting problem to look into. A number of major differences exist between spam-filtering in text messages and emails. Unlike emails, which have a variety of large datasets available, real databases for SMS spams are very limited. Additionally, due to the small length of text messages, the number of features that can be used for their classification is far smaller than the corresponding number in emails. Here, no header exists as well. Additionally, text messages are full of abbreviations and have much less formal language that what one would expect from emails. All of these factors may result in serious degradation in performance of major email spam filtering algorithms applied to short text messages. In this project, the goal is to apply different machine learning algorithms to SMS spam classification problem, compare their performance to gain insight and further explore the problem, and design an application based on one of these algorithms that can filter SMS spams with high accuracy. We use a database of 5574 text messages from UCI Machine Learning repository gathered in 2012. It contains a collection of 425 SMS spam messages manually extracted from the Grumble text Web site (a UK forum in which cell phone users make public claims about SMS spam), a subset of 3,375 SMS randomly chosen non-spam (ham) messages of the NUS SMS Corpus (NSC), a list of 450 SMS non-spam messages collected from Caroline Tag’s PhD Thesis, and the SMS Spam Corpus v.0.1 Big (1,002 SMS non-spam and 322 spam messages publicly available). The dataset is a large text file, in which each line starts with the label of the message, followed by the text message string. After preprocessing of the data and extraction of features, machine learning techniques such as naive Bayes, SVM, and other methods are applied to the samples, and their performances are compared. Finally, the performance of best classifier from the project is compared against the performance of classifiers applied in the original paper citing this dataset. Feature extraction and initial analysis of data is done in MATLAB, then applying different machine learning algorithms is done in python using scikit learn library.

**2.1 Some predictions done basing on actual values**

**Previous papers on SMS spam detection:**

El-Alfy and AlHasan [1] have proposed a model for filtering text messages for both email and SMS. They have analyzed different methods in order to finalize a feature set such that complexity can be reduced. They have used two classification algorithms i.e. Support Vector Machine (SVM) and Naïve Bayes and 11 features i.e. URLs, likely spam words, emotion symbols, special characters, gappy words, message metadata, JavaScript code, function words, recipient address, subject field and spam domain. They have evaluated their proposed model on five email and SMS datasets

Jialin et al. [2] have proposed a message topic model (MTM) for filtering Spam messages. Messages Topic Model (MTM) considers symbol terms, background terms and topic terms to represent spam messages and it is based on the probability guess of latent semantic analysis. They have used k-means algorithm to remove the sparse problem by training SMS spam messages into random irregular classes and then aggregating all SMS spam messages as a single file such that to capture word co-occurrence patterns.

Chan et al. [3] have presented two methods for SMS Spam filtering i.e. feature reweighting method and good word attack. Both methods focus on the length of the message along with considering the weight of message. Good word attack focuses on deceiving the output of classifier by using least number of characters while for feature reweighting method they have introduced a new rescaling function for rescaling the weights. They have evaluated the experiment on two datasets i.e. SMS and comment.

Xu et al. [4] have detected SMS Spam messages using content-less features. They have used 2 classification algorithms i.e. SVM and k-nearest neighbor (KNN) and feature set consisting of 3 features i.e. static, temporal and network for their experiment. They found that by combining temporal and network features SMS Spam messages can be detected more accurately and with good performance. Moreover, they also found the ways filter SMS Spam messages by using features that contain graph-topology and temporal information thus excluding the content of the message.

Uysal et al. [5] have proposed a method for SMS Spam filtering by using two feature selection based approaches i.e. chi-square metrics and information gain in order to select discriminative features. They have also developed a real time mobile application for SMS Spam filtering based on android application. They have used two different Bayesian based classification algorithms i.e. probabilistic and binary. According to the authors, their proposed system is highly accurate in detecting both spam and legitimate messages.

Yadav et al. [6] developed a model SMS Assassin for SMS Spam filtering. They have used a feature set of 20 lightweight features and two machine learning algorithms i.e. Support Vector Machine (SVM) and Bayesian learning. They have collected a dataset of 2000 messages from users within the time span of two months. In their proposed model whenever the user gets some message over his phone, then SMS Assassin initially captures that message without user’s knowledge, fetches feature values, and sends theses values to the server for classification. If the messages are reported as spam, then the user will not be able to see that message and it will be redirected to spam folder.

Hidalgo et al. [7] have analyzed that how Bayesian filtering technique can be used to detect SMS Spam. They have built two datasets one in English and another in Spanish. Their analysis shows that Bayesian filtering techniques that were earlier used in detecting email spam can also be used to block SMS Spam.

1. **PROPOSED METHOD**

**3.1 Bayesian Filtering Techniques:**

Content-based spam filters can be built manually, by hand engineering the set of attributes that define spam messages. These are often called heuristic filters, and some popular filters like Spam Assassin have been based on this idea for years. Content based filters can also be built by using Machine Learning techniques applied to a set of pre-classified messages. These so-called Bayesian filters are very accurate according to recent statistics, and their applicability to SMS spam seems immediate. Bayesian1 filters automatically induce or learn a spam classifier from a set of manually classified examples of spam and legitimate (or ham) messages (the training collection).

The learning process takes as input the training collection, and consists of the following steps:

• **Preprocessing**: Deletion of irrelevant elements (e.g. HTML), and selection of the segments suitable of processing (e.g. headers, body, etc.).

• **Tokenization:** Dividing the message into semantically coherent segments (e.g. words, other character strings, etc.).

• **Representation:** Conversion of a message into an attribute-value pairs’ vector , where the attributes are the previously defined tokens, and their values can be binary, (relative) frequencies, etc.

• **Selection:** Statistical deletion of less predictive attributes (using e.g. quality metrics like Information Gain).

• **Learning:** Automatically building a classification model (the classifier) from the collection of messages, as they have been previously represented. The shape of the classifier depends on the learning algorithm used, ranging from decision trees (C4.5), or classification rules (Ripper), to statistical linear models (Support Vector Machines, Winnow), neural networks, genetic algorithms, etc. Each new target message is pre-processed, tokenized, represented and feed into the classifier, in order to take a classification decision on it (whether it is spam or not). Current methods in Bayesian filter development are focused on the first steps, given that the quality of representation has big impact on the accuracy of the learned model. It is noteworthy that some researchers have developed highly accurate filters by employing character-level tokenization, putting nearly all the intelligence of the filter in the learning method (a for of text compression)

**3.2 Application to Mobile Spam:**

Since having a good term representation is one of the most important parts for getting a good classifier, we have to face the fact that SMS messages have not the same structure and characteristics than email messages. We have described techniques used to filter spam email messages, but we cannot state they can be also effective filtering SMS. SMS are usually shorter than email messages. Only 160 characters are allowed in a standard SMS text, and that could be a problem because using fewer words means less information to work with. Also, due to the above constraint, people tend to use acronyms when writing SMS. Moreover, the abbreviations used by SMS users are not standard for a language, but they depend on the users communities. Such language variability provides more terms or features, and a more sparse representation. We have to test if the state of the art methods used to extract terms from email messages are also suitable for SMS texts.

**3.3 Feature Selection:**

A data mining inspired approach, we have decided to feed the test with a possibly very big number of attributes, letting the attribute selection step the responsibility for deleting less informative attributes. We use Information Gain (IG) as attribute quality metric. The experience in learning-based text classification is that IG can reduce substantially the number of attributes, without no loss (or even some improvement) of accuracy. We have made experiments selecting all tokens scoring over 0 (zero) in Information Gain, and sets with 100 and 200 tokens with highest IG. Such tokens may provide information for the spam class (that is, they correlate to it), or for the legitimate class.

**3.4 Machine Learning Algorithms:**

For our experiments we have used the following algorithms:

* + 1. **Naive Bayes:**

The Naive Bayes algorithm is an intuitive method that uses the probabilities of each attribute belonging to each class to make a prediction. It is the supervised learning approach you would come up with if you wanted to model a predictive modelling problem probabilistically.

Naive bayes simplifies the calculation of probabilities by assuming that the probability of each attribute belonging to a given class value is independent of all other attributes. This is a strong assumption but results in a fast and effective method.

The probability of a class value given a value of an attribute is called the conditional probability. By multiplying the conditional probabilities together for each attribute for a given class value, we have a probability of a data instance belonging to that class.

To make a prediction we can calculate probabilities of the instance belonging to each class and select the class value with the highest probability.

Naive bases is often described using categorical data because it is easy to describe and calculate using ratios. A more useful version of the algorithm for our purposes supports numeric attributes and assumes the values of each numerical attribute are normally distributed (fall somewhere on a bell curve). Again, this is a strong assumption, but still gives robust results.

**Steps:**

1. **Handle Data**: Load the data from CSV file and split it into training and test datasets.
2. **Summarize Data**: summarize the properties in the training dataset so that we can calculate probabilities and make predictions.
3. **Make a Prediction**: Use the summaries of the dataset to generate a single prediction.
4. **Make Predictions**: Generate predictions given a test dataset and a summarized training dataset.
5. **Evaluate Accuracy**: Evaluate the accuracy of predictions made for a test dataset as the percentage correct out of all predictions made.
6. **Tie it Together**: Use all of the code elements to present a complete and standalone implementation of the Naive Bayes algorithm.

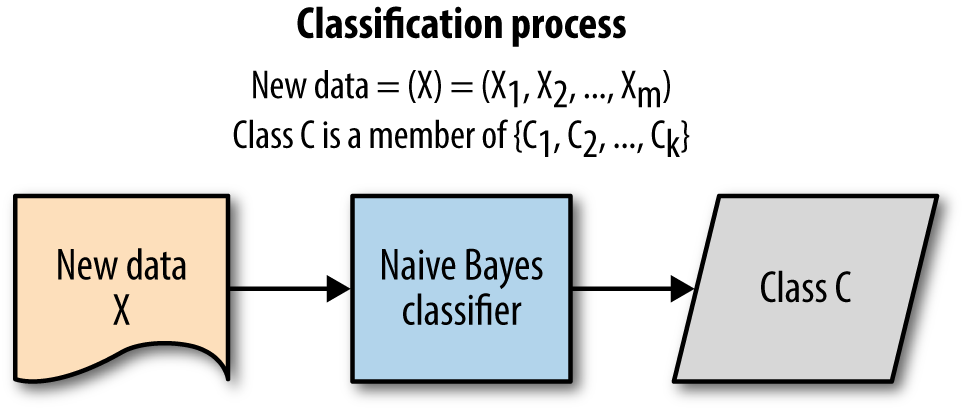
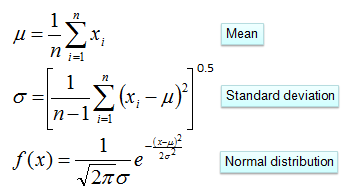


Figure 3.4.1 Classification process



* + 1. **Random forest:**

Random Forest is a machine learning algorithm used for classification, regression, and feature selection. It's an ensemble technique, meaning it combines the output of one weaker technique in order to get a stronger result.

The weaker technique in this case is a decision tree. [Decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) work by splitting the and re-splitting the data by features. If a decision tree is split along good features, it can give a decent predictive output.

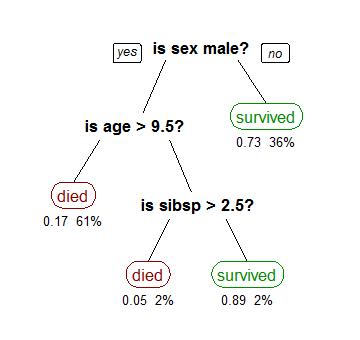
[](https://api.ning.com/files/oTw*v75BEJFE4pyzZCti2kBKk*YgwOkB1HmcMJGYGJHBq6VoZwXhgfs1xcxv7gJ-EAvQkqp39Hpdgp3DfFsn8yRJSX4dLqKN/cart_tree_titanic_survivors.png)

Figure 3.4.2 Decision tree

Random Forest works by averaging decision tree output, but it’s a bit more complicated than that. It also ranks an individual tree’s output, by comparing it to the known output from the training data. This allows it to rank features. Some of the decision trees will perform better, and so the features within the tree will be deemed more important.

Classification and regression would be the actual output of the model.  
 A good RF (meaning one that generalizes well) will have higher accuracy by each tree, and higher diversity among it’s trees.

One downfall of random forest is it can fail with higher dimensional data, because the trees will often be split by less relevant features. If you’re still intrigued by random forest, I encourage you to research more on your own! It gets a lot more mathematical.

Random forests is an averaging ensemble method for classification. The ensemble is a combination of decision trees built from a bootstrap sample from training set. Additionally, in building the decision tree, the split which is chosen when splitting a node is the best split only among a random set of features. This will increase the bias of a single model, but the averaging reduces the variance and can compensate for increase in bias too. Consequently, a better model is built. In this work, the implementation of random forests in scikitlearn python library is used, which averages the probabilistic predictions. Two number of estimators are simulated for this method. With 10 estimators, the overall error is 2.16%, SC is 87.7 %, and BH is 0.73%. Using 100 estimators will result in overall error of 1.41 %, SC of 92.2 %, and BH of 0.51 %. We observe that comparing to the naive Bayes algorithm, although the complexity of the model is increased, yet the performance does not show any improvement

1. **CODE**

import pandas as pd

import re

from nltk.stem.porter import PorterStemmer

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

dataSet = pd.read\_csv('spam.csv', encoding='cp437')

y = dataSet.iloc[:, 0].values

stemmedReviews = []

for i in range(y.size):

review = re.sub('[^a-zA-Z]', ' ', dataSet['v2'][i])

review = review.lower()

review = review.split()

ps = PorterStemmer()

review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

stemmedReviews.append(review)

cv = CountVectorizer(max\_features=3000)

x = cv.fit\_transform(stemmedReviews).toarray()

le = LabelEncoder()

y = le.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20)

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

pred = classifier.predict(X\_test)

print("\nNAIVE BAYES : \n")

print('Accuracy score: {}'.format(accuracy\_score(y\_test, pred)\*100))

print('Precision score: {}'.format(precision\_score(y\_test, pred)\*100))

print('Recall score: {}'.format(recall\_score(y\_test, pred)\*100))

print('F1 score: {}'.format(f1\_score(y\_test, pred)\*100))

classifier1 = RandomForestClassifier(n\_estimators=15, criterion='entropy')

classifier1.fit(X\_train, y\_train)

predRF = classifier1.predict(X\_test)

print("\nRANDOM FOREST CLASSIFIER : \n")

print('Accuracy score: {}'.format(accuracy\_score(y\_test, predRF)\*100))

print('Precision score: {}'.format(precision\_score(y\_test, predRF)\*100))

print('Recall score: {}'.format(recall\_score(y\_test, predRF)\*100))

print('F1 score: {}'.format(f1\_score(y\_test, predRF)\*100))

stemmedReviews = []

dataset = pd.read\_csv('test1.csv', encoding='cp437')

for i in range(0, 1):

review = re.sub('[^a-zA-Z]', ' ', dataset['v1'][i])

review = review.lower()

review = review.split()

ps = PorterStemmer()

review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

stemmedReviews.append(review)

temp = cv.transform(stemmedReviews)

predNaiveBayes = classifier.predict(temp.toarray())

predRandomForest = classifier1.predict(temp.toarray())

if predNaiveBayes == 1:

OutputNB = "Spam"

else:

print("Not spam")

OutputNB = "Not spam"

if predRandomForest == 1:

print("Spam")

OutputRF = "Spam"

else:

print("Not spam")

OutputRF = "Not spam"

print("\nOutput for Gaussian NB = {0} {1}".format(classifier.predict(temp.toarray()), OutputNB))

print("\nOutput for Random Forest Classifier = {0} {1}".format(classifier1.predict(temp.toarray()), OutputRF))

1. **OUTPUT:**

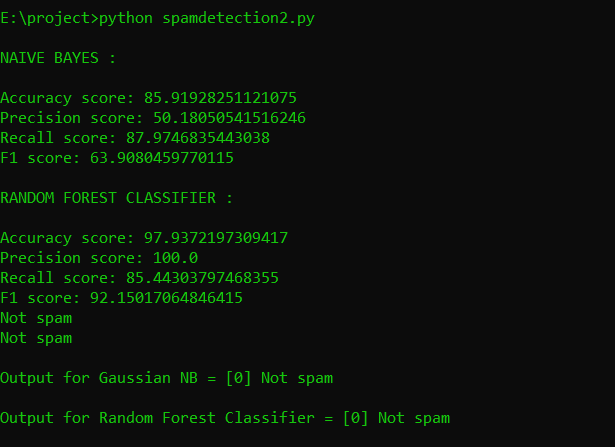
****

Figure 5.1 Output of the program

1. **RESULT**

From simulation results, multinomial naive Bayes with laplace smoothing and SVM with linear kernel are among the best classifiers for SMS spam detection. The best classifier in the original paper citing this dataset is the one utilizing Navie Bayes classifier which yields overall accuracy of 85.91%. Next best classifier in their work is boosted Random forest classifier with overall accuracy of 97.93%. Comparing to the result of previous work, our classifier reduces the overall error by more than half. Adding meaningful features such as the length of messages in number of characters, adding certain thresholds for the length, and analyzing the learning curves and misclassified data have been the factors that contributed to this improvement in results.

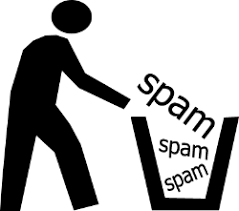


Figure 6.1 Spam bin

1. **CONCLUSION**

From this series of experiments, we can derive the following conclusions:

• Given the short size of messages, and the literature on spam email filtering, it is reasonable to define a wide range of attribute types, and let the attribute selection using IG process to select those most promising for classification. However, the number of selected attributes cannot be known in advance, although it seems proportional to the spam messages. It may be valuable to test other kinds of features (e.g. encoding all numbers, or marking telephone numbers).

• The most suitable learning algorithm for a prototype is, after in-depth evaluation, Random Forest. This is supported by our and others’ previous work in spam email detection and in text classification. Also, although we have not demonstrated this empirically, the running time of learning with Random Forest has been comparable to Naïve Bayes, and much smaller than the running time for learning rules or decision trees with high accuracy.

**8. REFERENCES**

[1] : El-Alfy, E.S.M., AlHasan, A.A.: Spam filtering framework for multimodal mobile communication based on dendritic cell algorithm. Future Gen. Comput. Syst. 64, 98–107 (2016). doi:10.1016/j.future.2016.02.018

[2] : Jialin, M., Zhang, Y., Liu, J., Yu, K., Wang, X.: Intelligent SMS spam filtering using topic model. In: International Conference on Intelligent Networking and Collaborative Systems (INCoS), pp. 380–383. IEEE (2016). doi:10.1109/INCoS.2016.47

[3] :Chan, P.P.K., Yang, C., Yeung, D.S., Ng, W.W.Y.: Spam filtering for short messages in adversarial environment. Neurocomputing 155, 167–176 (2015). doi:10.1016/j.neucom. 2014.12.034

[4] :Xu, Q., Xiang, E.W., Yang, Q., Du, J., Zhong, J.: SMS spam detection using non-content features. IEEE Intell. Syst. 27(6), 44–51 (2012)

[5] :Uysal, A.K., Gunal, S., Ergin, S., Gunal, E.S.: A novel framework for SMS spam filtering. In: International Symposium on Innovations in Intelligent Systems and Applications

(INISTA), pp. 1–4. IEEE (2012). doi:10.1109/INISTA.2012.6246947

[6] :Yadav, K., Kumaraguru, P., Goyal, A., Gupta, A., Naik, V.: SMSAssassin: crowdsourcing driven mobile-based system for SMS spam filtering. In: 12th Workshop on Mobile Computing Systems and Applications, pp. 1–6. ACM (2011). doi:10.1145/2184489. 2184491

[7] :Hidalgo, J.M.G., Bringas, G.C., Sánz, E.P., García, F.C.: Content based SMS spam filtering. In: ACM Symposium on Document Engineering, pp. 107–114. ACM (2006). doi:10.1145/ 1166160.1166191