"In Pursuit of Technical Competitiveness"

DETECTION OF ADVERTISEMENTS USING MACHINE LEARNING APPROACH

For the Degree of
Bachelor of Engineering in
Information Technology

By

ADITYA MAHAMUNI [BE15F06F032]
MAYUR DESHMUKH [BE15F06F011]
VINOD YEVATIKAR [BE15F06F059]
JAYESH MUTHA [BE15F06F038]
SHUBHAM BAKLIWAL [BE15F06F006]

Under the Guidance of
Ms. Anjana N. Ghule

Department of Information Technology
Government College of Engineering, Aurangabad
Maharashtra State, India

(2018-2019)

"In Pursuit of Technical Competitiveness"

DETECTION OF ADVERTISEMENTS USING MACHINE LEARNING APPROACH

For the Degree of
Bachelor of Engineering in
Information Technology

By

ADITYA MAHAMUNI	**[BE15F06F032]**
MAYUR DESHMUKH	**[BE15F06F011]**
VINOD YEVATIKAR	**[BE15F06F059]**
JAYESH MUTHA	**[BE15F06F038]**
SHUBHAM BAKLIWAL	**[BE15F06F006]**

Under the Guidance of
Ms. Anjana N. Ghule

Department of Information Technology
Government College of Engineering, Aurangabad
Maharashtra State, India

(2018-2019)

CERTIFICATE

This is to certify that the project report entitled **Detection of Advertisements using Machine Learning Approach**, which is being submitted herewith for the award of the **Bachelor of Engineering in Information Technology** affiliated to Dr.Babasaheb Ambedkar Marathwada University, Aurangabad. This is the result of the original work and contribution by **Aditya Rajkumar Mahamuni, Mayur Subhashrao Deshmukh, Vinod Vijaykumar Yevtikar, Jayesh Paras Mutha, Shubham Bakliwal** is bonafied work completed under my supervision and guidance.

Place: Aurangabad

Date: / /2019

Mrs. Anjana N. Ghule
Guide
Information Technology
Department

Prof. C.M. Gaikwad
HOD
Information Technology
Department

Prof. Dr. P.B. Mrunal
Principal
Govt. College of Engineering,
Aurangabad

DECLARATION

We hereby declare that the project work entitles “Detection of Advertisements using Machine Learning Approach” submitted to the Government College of Engineering, Aurangabad, is a record of an original work done by us under the guidance of Ms. Anjana N Ghule, our project guide.

Aditya Mahamuni [BE15F06F032]

Mayur Deshmukh [BE15F06F011]

Vinod Yevatikar [BE15F06F059]

Shubham Bakliwal [BE15F06F006]

Jayesh Mutha [BE15F06F038]

Index

Table of Contents

List of Abbreviations

ML	Machine Learning
SVM	Support Vector Machines
k-NN	k-Nearest Neighbour
AD	Advertisement
URL	Uniform Resource Locator
FP	False Positive
FN	False Negative
TP	True Positive
TN	True Negative
NB	Naïve Bayes
LR	Logistic Regression
BoW	Bag of Words

List of Figures

List of Tables

Abstract

Due to the significant development of online advertising, malicious advertisements have become one of the major issues to distribute scamming information, click fraud and malware. Most of the current approaches are involved with using filtering lists for online advertisements blocking, which are not scalable and need manual maintenance. This paper presents a lightweight online advertising classification system using lexical-based features as an alternative solution. In order to imitate real-world cases, three different scenarios are generated depending on three different URL sources. Then a set of URL lexical-based features are selected from previous researches in the purpose of training and testing the proposed model.

The objective of the study is to analyse the distributions of ads provided by different sources, so we could have a rough idea about the environment of Internet. Online advertisement filtering is a cost-sensitive classification task. Misclassifying legitimate web content is generally costly then misclassifying unwanted ads. To deal with this problem we can use cost sensitive accuracy.

1. INTRODUCTION

1.1 Introduction

There are lots of advertisements over the Internet, who have become one of the major approaches for companies to advocate their products and for webmasters to make money by posting these ads. Some big players in this market are Google and Facebook, with a lot of other companies.

These ads can sometimes be useful, while can also be very annoying at some other occasions. For almost as long as the commercial world wide web has existed, annoying advertisements have competed with desired content for users' attention. To combat these annoyances, users have created methods for filtering out the unwanted ads.

Figure 1.1: Amazon Advertisement

In the modern ages, advertisements are widely used to actively persuade customers to buy products or service. There is thus a deep relationship between advertisements and the human society. There are three main viewpoints to describe such relationship. The first one is that advertisements create demands. Some say that advertisements create a lot of useless demands, but the opposite side says that these demands already exist but people didn't know before. The second viewpoint is that advertisements affect people's social behaviours, while the opposite side thinks that such affection is weak.

The third viewpoint is that advertisements cause overspending. But some researchers think that advertisements only reflect reality of the society. From these arguments, we realize that there are complex relationships between advertisements and the human society. Advertisement studies thus play an important role to realize macro human behaviours.

However, tremendous amounts of advertisements in various forms impede efficient advertisement studies. In this work, we develop a visual analysis system that can be used to analyse large-scale advertisements and thus facilitate advertisement studies.

Fast machine-learning online advertisements detection is needed for a few reasons. In order to efficiently access information on internet it is necessary to separate superfluous web content (like ads) from legitimate one. For dealing with thousands evolving adverts we need automated solution that can be easily adapted to the new conditions. Finally, to deal with the voluminous data streams in real-time we need fast solution—thus, classification based only on URLs is a proper approach to this requirement.

Types of Online Advertisements:

1. Display Advertising
2. Search Engine Marketing & Optimization (SEM) & (SEO)
3. Social Media
4. Native Advertising
5. Pay Per Click (PPC)
6. Remarketing
7. Affiliate Marketing
8. Video Ads

1.2 Necessity

Advertisement plays an important role in the human society. Advertisement studies are related to many important issues in economics, social science, and marketing.

In a large online advertising system, adversaries may attempt to profit from the creation of low quality or harmful advertisements. In this paper, we present a large-scale data mining effort that detects and blocks such adversarial advertisements for the benefit and safety of our users. Because both false positives and false negatives have high cost, our deployed system uses a tiered strategy combining automated and semi-automated methods to ensure reliable classification.

A natural reaction is for users to install ad blockers which prevent advertisers from tracking users or displaying ads. Traditional ad blocking software relies upon hand-crafted filter expressions to generate large, unwieldy regular expressions matched against resources being included within web pages.

This process requires a large amount of human overhead and is susceptible to inferior filter generation.

1.3 Objectives

The objective of the study is to analyse the distributions of ads provided by different sources, so we could have a rough idea about the environment of Internet. Online advertisement filtering is a cost-sensitive classification task. Misclassifying legitimate web content is generally costly then misclassifying unwanted ads. To deal with this problem we can use cost sensitive accuracy [1].

We employ strategies to address the challenges of learning from highly skewed data at scale, allocating the effort of human experts, leveraging domain expert knowledge, and independently assessing the effectiveness of our system.

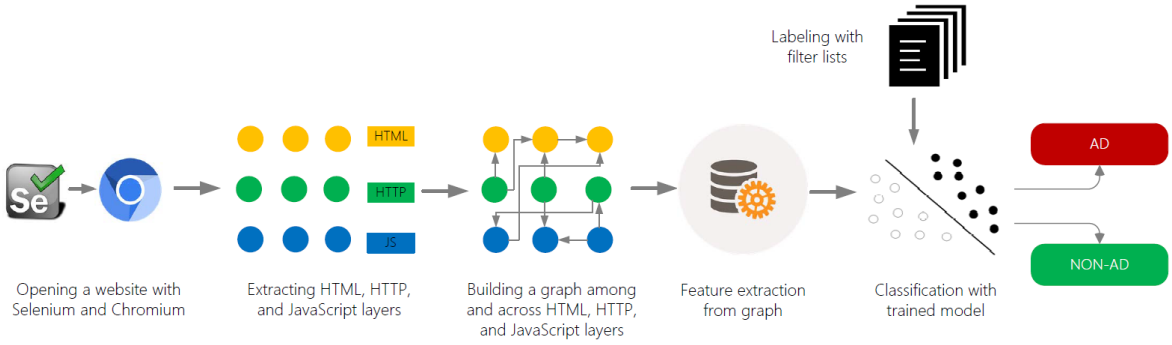


Figure 1.2: Approach for classification of AD and NON-AD URLs.

We propose a framework to conduct advertisement detection, segmentation, and classification. An alternate approach which leverages machine learning to bootstrap a superior classifier for ad blocking with less human intervention. We show that our classifier can simultaneously maintain an accuracy similar to the hand-crafted filters.

1.4 Challenges

The problem of detecting adversarial advertisements is complicated by scale. With millions of advertisers and billions of advertiser landing pages,

automated detection method is clearly needed. However, unlike many data-mining tasks in which the cost of false positives (FP's) and false negatives (FN's) may be traded off, in this setting both false positives and false negatives carry extremely high misclassification cost. Thus, both FP and FN rates must be driven toward zero, even for difficult edge cases.

• High cost of both FP's and FN's.
In our setting, both FP's and FN's have high cost; we cannot trade off one against the other. Using a combination of automated and semi-automated effort helps drive both FP and FN rates towards zero.

• Training many models at scale.
At a high level, our system may be viewed as an ensemble composed of many large-scale component models. Each of these models must be frequently trained, evaluated, calibrated, and monitored.

• Capturing expert knowledge.
To cope with constantly evolving adversarial tactics, our system needs to be able to capture and leverage expert knowledge quickly and efficiently. Using experts to label examples is one such method. Section 5 details additional approaches including the use of active learning, providing exploratory tools to experts, and enabling experts to develop rule-based models for fast response.

• Independent evaluation.
Because we rely on human experts for ground truth, regular independent evaluations are critical to ensure that our ground truth understanding is accurate and comprehensive.

1.5 Organization

The Report is organized into four chapters as

Chapter 1: The content of this chapter includes Introduction, Necessity and Objectives of the proposed system.

Chapter 2: This chapter focuses on Terms, Concepts, i.e. it includes the study of Literature Review of past research papers.

Chapter 3: This chapter is devoted for System Development, and also contains the algorithmic flow of proposed work.

Chapter 4: This chapter consists Conclusions, Applications of the proposed system and future scope of further Research.

2. LITERATURE REVIEW

2.1 Related Works

Related works from 2004 to 2015 have been reviewed in order to identify a collection of commonly used features that could yield high detection accuracy.

1. Lawrence Kai Shih and David R. Karger [2] in 2004 proposed a URL-based approach for webpage identification for ad-blocking and content recommendation. They used four training systems for data labelling. The four systems are: WebWasher, Redirect, Learn-WW and Learn-RD. And the two type of features they used are URL-based feature and Network-based feature.

2. Jianping Zhang in 2006, M. Indra Devi in 2007 [3] and Eda Baykan in 2009 [4] studied Lightweight Online Advertising Classification System using Lexical-based, studied the problem of URL-based webpage classification with machine learning techniques. With the application of SVM, Naive Bayes (NB), both of the authors broke the URL into a sequence of tokens, and treated the URL tokens as the feature.

3. Piotr L. Szczepanski in 2013 presented a URL-based automated framework as a solution to the problem of online advertisements detection. Their experiments were performed on the following popular classifier, K-Nearest Neighbour (KNN), NB, Bayesian Network, SVM, Decision Tree, Random Forest, and AdaBoost with lexical based features.

4. Sruti Bhagavatual in 2014 [5] designed a machine learning based approach to classify ad-URLs using the AdBlockPlus classification system. The features they used in their study can be categorized into lexical-based feature and URL based feature. Their evaluation on the classification models was conducted by the classifiers such as NB, SVM, LR and KNN, along with the textual features.

5. Min-Yen Kan and Hoang Oanh Nguyen Thi in 2005 [6] demonstrated the use of URL alone in performing webpage classification, such as identification of online advertising. They applied machine learning classifiers like Support Vector Machines (SVM) and Maximum Entropy (ME) to evaluate the performance of the textual based features.

2.2 Detection of Advertisements

Detection of advertisements has been a topic of much research in recent years. The main contribution of this research is the characterization of online advertisements.

At this point, the problem is pretty much simplified to a standard text classification problem. One step further, if we think a little deeper about the nature of advertisement links, they usually have a stable pattern, or even exactly same URL. The desirable output of a machine-learning algorithm is an effective classification of a given URL as no-Ad or ad.

In our studies we scrutinized the following popular classifiers. For each of them we examine many different configurations and scope of our experiments is briefly described in the next paragraphs.

k-Nearest Neighbour:

k-nearest neighbour classifier picks the most common class among k closest neighbours of a given instances.

Naïve Bayes:

Naive Bayes classifier maximizes a conditional probability of a new instance belonging to a particular class by looking at all features individually. Using Bayes' theorem this problem can be expressed with probability of instance's features occurring in specified class, which in turn can be obtained using statistics. To reduce problem even further we naively assume feature independence. This approach was tested on three configurations. We discretize continuous features using an equal frequencies method assuming a normal distribution.

Support Vector Machine (SVM):

Support vector machine finds a single hyperplane that separates all the instances with maximum distance between plane and instances. Internally this is done by expressing classification task as a constrained optimization problem and solving it with quadric programming. If no such plane exists, SVM can relax a separability constraint by introducing penalty function for misclassified instances. Moreover, the dimension of feature space can be greatly increased using kernel function trick. Higher dimension separating plane projected back to the original feature space can model the variety of shapes.

Decision Tree:

Decision tree classifier is a tree that divides whole feature space into hyper-rectangular areas corresponding to particular classes. Inner nodes of the tree represent separating planes and each leaf indicates a class assigned to instances occupying that region of the feature space. Decision trees are built recursively. Starting in the root node, algorithm must decide whether to stop branching and create a leaf node or continue division. Tests for inner nodes are selected greedily based on the information gain estimation, which is very often expressed as an entropy reduction. For better generalization after initial generation tree nodes can be pruned and substituted with leaves.

Random Forest Algorithm:

Random forest algorithm is a meta-classifier based on the number independent decision trees created from the random subset of features. The output of this classifier is chosen during voting between individual trees. Random forest does not suffer from overfitting as much as single decision trees built in greedy way.

2.3 Feature Engineering

Feature Engineering in which the raw dataset is transformed into flat features which can be used in a machine learning model.

Feature engineering is a key component of effective data mining; the following is a listing of the features extracted from advertisements during training and classification.

1. Natural language features are extracted from the text of advertisement keywords, creatives, and landing pages. These include term-level features, and semantically related terms and topic-level features.

2. String-based features are intended to avoid the possibility that adversaries may exploit alternate spellings or typographical manipulation to avoid detection. We incorporate features that allow inexact string matching.

3. Crawl-based features are extracted from the results of the http fetch of the landing page.

For the classifier to be effective, it is imperative to identify features that may differentiate an ad-related URL from a non-ad-related URL. These include characteristics of the structure of the URL, keywords present in the URL, the container it was requested from on a page, and other page properties. All features are either binary or real-valued between 0 and 1. Conveniently, this approach allows all features to be given equal weight at training time.

#	Category	Feature Name	Description
F1	Textual-based Feature	Textual length of URL	The total length of the URL
F2	URL-based Feature	URL component presence	If query component or user information component present in the given URI
F3	Textual-based Feature	Token occurrences	The occurrences of each word in URL
F4	URL-based Feature	Token occurrences by URL component	The count of token occurrences for each component
F5	Textual-based Feature	Sequential n-gram	URL first split into tokens, then derive sequence of n tokens from them
F6	Textual-based Feature	Full token n-gram	The count of occurrences of tokens with regard to succession relation
F7	URL-based Feature	Token count (total and per URL component)	Ads are likely to have many parameters in query component
F8	Textual-based Feature	Numeric tokens count (total and per URL component)	The count of occurrences of numeric values in URL
F9	Textual-based Feature	Ad-related keywords	Such as 'ad', 'advert', 'popup','banner', 'sponsor', 'iframe', 'googlead', 'adsys' and 'adser'
F10	Textual-based Feature	Suspicious symbol presence	If the URL contain semicolons to separate parameters? If the URL contains valid query? If any suspicious symbol just like @ present?
F11	URL-based Feature	Original page related	If the base domain name is present in the query of the URL? If the requested URL is on the same domain?
F12	URL-based Feature	Size of Ads in URL	Indicating the size of the ads it going to display
F13	URL-based Feature	Dimensions of the URL	Indicating the dimensions of the screen or browser
F14	URL-based Feature	Iframe container	Indicating if the URL is requested from within an iframe either in the context of the page or in the context of nested iframes
F15	URL-based Feature	Proportion of external requested resources	The proportions of external iframe, script and resource requests
F16	Textual-based Feature	Textual length of hostname	The length of hostname in given URL
F17	Textual-based Feature	Dots occurrences	The number of dots in the URL
F18	Host-based Feature	IP address	Is the IP address in the blacklist?
F19	Host-based Feature	WHOIS properties	What is the date of registration, update and expiration? Who is the registrar? Who is the registrant? Is the WHOIS entry blocked?
F20	Host-based Feature	Domain name properties	What is the TTL for DNS records associated with the hostname?
F21	Host-based Feature	Geographic properties	In which continent/country/city does the IP address belong?
F22	JavaScript Feature	JavaScript source code	Including code obfuscation, dynamic code and URL generation, code structure, function call distribution, event handling, script origin, presence of keywords
F23	URL-based Feature	URL tree	The left-most item (http:) becomes the root node of the tree, successive tokens in the URL become the children of the previous token
F24	Host-based Feature	Blacklist membership	Is the IP address in a blacklist?
F25	Host-based Feature	Connection speed	What is the speed of the uplink connection?
F26	Textual-based Feature	Presence of IP address	IP is not included in a normal URL
F27	Textual-based Feature	Unknown noun presence	Domain names are not created by using some random letters
F28	URL-based Feature	Misplaced top level domain	If the domain name is present in the path section?
F29	Network-based Feature	URL redirection	If the URL has been used to redirect to many other pages?
F30	Network-based Feature	Traffic received	The amount of web traffic that each website gets
F31	Network-based Feature	HTML table tree	The visual/physical placement of links on a page
F32	Textual-based Feature	Precedence Bigram	The left-to-right precedence of two tokens in the URL
F33	Network-based Feature	URL redirect path	The redirection chain of a set of URLs
F34	Network-based Feature	Domain redirect path	The redirection chain of a set of domains
F35	Host-based Feature	Domain frequency	The number of publishers that associated with the domain each day
F36	Host-based Feature	Domain-pair frequency	The frequency of two neighboring URLs/domains
F37	Textual-based Feature	Dash count in hostname	The number of dash present in hostname

Figure 2.1: List of Detection Features.

Below is a summary of the key feature categories which can be used to train our classifier:

1. Ad-related keywords:
   A URL is more likely to be ad-related than not if it contains certain keywords. We check for: ‘ad’, ‘advert’, ‘popup’, ‘banner’, ‘sponsor’, ‘iframe’, ‘googlead’, ‘adsys’, and ‘adser’.

2. Lexical features:
   This feature set contains two features that come from the character arrangements in the URL itself. First, whether it contains semicolons to separate parameters. It could either be in the place of actual query parameters usually separated by ‘&’ or a string of semicolon parameters as part of the query parameters or even the path itself.

3. Related to the original page:
   This feature set contains two features that contain information about relationships between the requested URL and its base page domain.

4. Size and dimensions in URL:
   Ad URLs very often contain information about the size of the ad it should display or the dimensions of the screen or browser it is going to be displayed on.

2.3.1 Pre-Processing

Data pre-processing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviours or trends, and is likely to contain many errors

The first step of pre-processing which is used to presents the text documents into clear word format. The documents
prepared for next step in text classification are represented by a great number of features. Commonly the steps taken are:

1. Tokenization:
   A document is treated as a string, and then partitioned into a list of tokens.

2. Removing stop words:
   Stop words such as “the”, “a”, “and”, etc. are frequently occurring, so the insignificant words need to be removed.

3. Stemming word:
   Applying the stemming algorithm that converts different word form into similar canonical form.

This step is the process of conflating tokens to their root form, e.g. connection to connect, computing to compute.
Data cleaning, also called data cleansing or scrubbing. Fill in missing values, smooth noisy data, identify or remove the outliers, and resolve inconsistencies. Data cleaning is required because source systems contain "dirty data" that must be cleaned.

2.4 Datasets

Viktor Krammer [7] chose the Alexa Global Top 500 list of popular websites in their study. A total of 502 pages were examined in their experiment and 314 (63%) of them displayed some forms of advertising.

Sruti Bhagavatula selected the Alexa top 500 US sites from February 2014 as their dataset input, and crawling all the links from a page up to depth 2 from the source page, which led to a total of 60,000 URLs with 50% ad-related sources.

A dataset which includes normal URLs (Non-ad), Benign-ad URLs and Malicious-ad URLs. Three types of URLs sources are collected in order to conduct the classification task.

Source I (Malicious-ad URLs):

Examples of malicious advertising URLs from a URL blacklist service website, www.urlblacklist.com. The data sources are all about malicious advert servers and banned URLs.

Source II (Benign-ad URLs):

Benign advertising URLs from an advertising dataset, www.code.google.com/archive/p/open-advertisingdataset/, which is created by the University College London as an independent computational advertising dataset from the publicly available sources.

Source III (Non-ad URLs):

A crawler to fetch all the non-advertising URLs from Alexa Top 5000. For each Alexa domain (such as google.com), the crawler visits at most 20 pages which are originally linked with Alexa top page, from August 28th, 2016 to September 10th, 2016. After that, 5000 non-advertising URLs are selected manually from the above fetched URLs.

Piotr L. Szczepanski collected 55*k* instances of URLs from internet by browsing the most popular websites. They used proxy server to filter all URLs that have occurred during web surfing. Then, they constructed training set by applying *Adblock Plus* to all collected instances. This solution imposes limitation to the real-life accuracy of our experiments, while even this best rule-based ad detector is fallible. However, in order to test the usefulness of URL based online ad detection this approach is sufficient. This experiment shows that near 6% of all instances are online advertisements.

To generate training and ground truth labels for a collection of URL data, Sruti Bhagavatula and the others compared URLs against old and new EasyList filters to get two different sets of positive and negative examples. This resulted in two datasets each with the same URLs and features but with a small percentage of differing class labels for certain examples. For the nonrecent or "old" EasyList, most recent evaluations used the version from September 22nd, 2013 and the version from February 23rd, 2014 as the current or "new" filter list. They considered filters from February as "new", relative to September, to analyse prediction accuracy over time with respect to the present. They crawled the Alexa top 500 US sites from February 2014 up to depth 2 and checked each requested page resource URL against the new EasyList filters. Crawling all the links from a page up to depth 2 from the source page allowing them to capture normal advertisement activity on pages other than the index pages and on several other sites beyond the Alexa top 500. For each requested URL, while crawling, they stored features that were computed within the context of the page as well as various URL-based and lexical features to create their feature vectors. They checked each URL against the old EasyList and new EasyList to obtain two sets of feature vectors for their "old" and "new" datasets respectively with differing class labels. They collected about 60,000 total URLs; roughly 30,000 of which were ad-related and 30,000 non-ad related URLs according to the "new" filters.

2.5 Machine Learning

Machine learning (ML) is a category of algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.

The processes involved in machine learning are similar to that of data mining and predictive modelling. Both require searching through data to look for patterns and adjusting program actions accordingly. Many people are familiar with machine learning from shopping on the internet and being served ads related to their purchase. This happens because recommendation engines use machine learning to personalize online ad delivery in almost real time. Beyond personalized marketing, other common machine learning use cases include fraud detection, spam filtering, network security threat detection, predictive maintenance and building news feeds.

Let's explore fundamental machine learning terminology.

Labels:

A label is the thing we're predicting—the y variable in simple linear regression. The label could be the future price of wheat, the kind of animal shown in a picture, the meaning of an audio clip, or just about anything.

Features:

A feature is an input variable—the x variable in simple linear regression. A simple machine learning project might use a single feature, while a more sophisticated machine learning project could use millions of features, specified as:

$$x1,x2,...,xN$$

In the spam detector example, the features could include the following:

1. words in the email text
2. sender's address
3. time of day the email was sent.

Models:

A model defines the relationship between features and label. For example, a spam detection model might associate certain features strongly with "spam". Let's highlight two phases of a model's life:

1. Training means creating or learning the model. That is, you show the model labelled examples and enable the model to gradually learn the relationships between features and label.

2. Inference means applying the trained model to unlabelled examples. That is, you use the trained model to make useful predictions (y'). For example, during inference, you can predict medianHouseValue for new unlabelled examples.

Types of Machine Learning Methods:

a) Supervised Learning
b) Unsupervised Learning

a) Supervised Learning:

Supervised learning is the Data mining task of inferring a function from labelled training data. The training data consist of a set of training example. 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).

b) Unsupervised Learning:

Unsupervised Learning is a class of Machine Learning techniques to find the patterns in data. The data given to unsupervised algorithm are not labelled, which means only the input variables(X) are given with no corresponding output variables.

At this point, the problem is pretty much simplified to a standard text classification problem. One step further, if we think a little deeper about the nature of advertisement links, they usually have a stable pattern, or even exactly same URL.

Text Classification is an example of supervised machine learning task since a labelled dataset containing text documents and their labels is used for train a classifier. An end-to-end text classification pipeline is composed of three main components:

1. **Dataset Preparation:**
   The first step is the Dataset Preparation step which includes the process of loading a dataset and performing basic pre-processing. The dataset is then split into train and validation sets.

2. **Feature Engineering:**
   The next step is the Feature Engineering in which the raw dataset is transformed into flat features which can be used in a machine
   learning model. This step also includes the process of creating new features from the existing data.

3. **Model Training:**
   The final step is the Model Building step in which a machine learning model is trained on a labelled dataset.

2.6 Text Classification Algorithms

The problem is pretty much simplified to a standard text classification problem. One step further, if we think a little deeper about the nature of advertisement links, they usually have a stable pattern, or even exactly same URL.

Classification Algorithm:

Classification is the process of discovering a model for the class in terms of the remaining attributes. The objective is to use the training data set to build a model of the class label based on the other attributes such that the model can be used to classify new data not from the training data set attributes.

The objective is to use the training data set to build a model of the class label based on the other attributes such that the model can be used to classify new data not from the training data set.

1. Our classifier is a binary valued function
   f: Rd → {-1,1} chosen on the basis of the training set alone.

2. So how can we find classifiers that generalize well?
   The key is to constrain the set of possible binary functions we can entertain. In other words, we would like to find a class of binary functions such that if a function in this class works well on the training set, it is also likely to work well on the unseen images.

Text Classification Algorithms:

Text classification (also known as text categorization or topic spotting) is the task of automatically sorting a set of documents into categories from a predefined set.
Following are the popular classifiers relevant to our problem domain:

Naïve Bayes:

In machine learning, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the colour, roundness, and diameter features.

An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

Linear Classifier:

In the field of machine learning, the goal of statistical classification is to use an object's characteristics to identify which class (or group) it belongs to. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector. Such classifiers work well for practical problems such as document classification, and more generally for problems with many variables (features), reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use.

A regression model predicts continuous values.

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function.

SVM:

In machine learning, support vector machines are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis.

Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier(although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. The model extracts a best possible hyper-plane / line that segregates the two classes.

Decision Tree Classifier:

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of three types of nodes:

1. Decision nodes – typically represented by squares
2. Chance nodes – typically represented by circles
3. End nodes – typically represented by triangles

K-Nearest Neighbour:

In pattern recognition, the *k*-nearest neighbours algorithm (*k*-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:

1. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its *k* nearest neighbours (*k* is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbour.
2. In k-NN regression, the output is the property value for the object. This value is the average of the values of its *k*-nearest neighbours.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all machine learning algorithms.

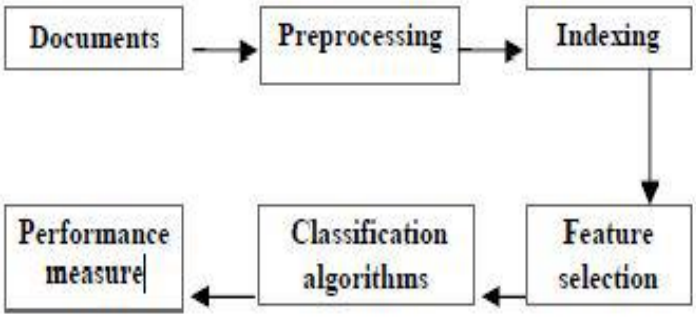


Figure 2.2 Document Classification Process

3. SYSTEM DEVELOPMENT

The system development consists of three phases:

1) System Engineering
2) Software Requirement Analysis
3) Software Design

Requirement analysis is a software engineering task that bridges the gap Between system-level software allocation and software design. Requirement analysis enables the system engineer to specify software functions and performance, indicate software's interface with other system elements, and establish constraints that software must meet. Requirement analysis allows the software engineer to refine the software allocation and build the models of the data, functional, behavioral domains that will be treated by software. Requirement analysis provides the software designer with models that can be translated into the data, architectural, interface and procedural design. Finally, the requirements specifications provide the developers and customers with the means to assess quality once software is built.

The most creative and challenging face of the system development is system design. It provides the understanding and procedural details necessary for the logical and physical stages of development. In designing a new system, the system analyst must have a clear understanding of the objective which the system is aiming to fulfil. The first step is to determine how the output is to be produced and in what format. Second, input data and master files have to be designed to meet the requirements of the proposed output. The operational phases are hand-led through program construction and testing.

Design of the system can be defined as a process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization. Thus, system design is a solution to "how to" approach to the creation of new system. This important phase provides the understanding and the procedural details necessary for implementing the system recommended in the feasibility study.

3.1 System Requirements

3.1.1 Hardware Requirements (Min.)

Processor	Core i5 processor
RAM	4GB

3.1.2 Software Requirements (Min.)

Technology -
Python ML Libraries:
Pandas, Scikit-learn.

Application Software –
Spyder, Jupyter Notebooks, WEKA.

Programming Language –
Python (2.7/3.6).

Operating System –
Windows 7 or later (32/64 bit)
Linux

3.2 Approach

Compared with other types of features, such as host-based features or network-based features, using lexical features alone can lead to a comparable classification accuracy without any time latency issues or resource consuming issues. The high accuracy and the lightweight properties make lexical-based features potential candidates for classifying malicious advertisements.

We test and evaluate the performance of the features that have been previously used to identify online advertisements i.e. the lexical text-based features and URL based features.

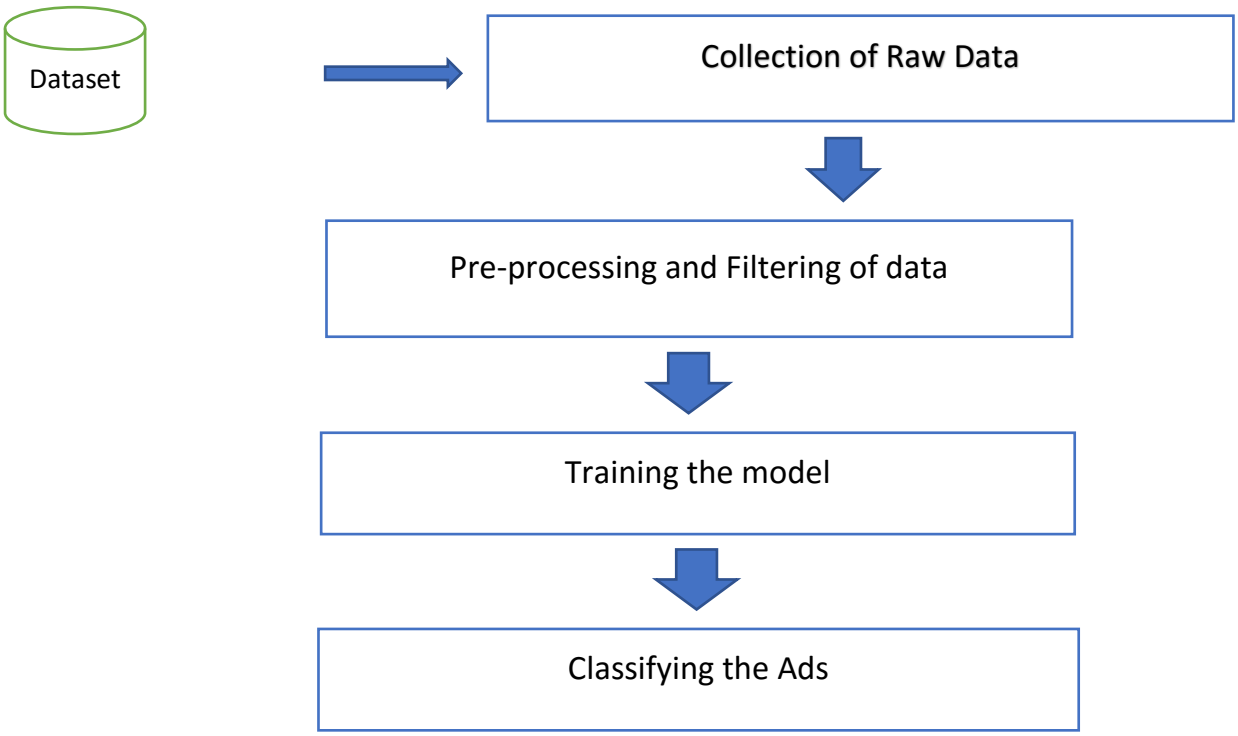


Figure 3.1 Data Flow Diagram

Features may be considered as noisy features if they do not contribute to the detection performance. Instead of improving the performance of classifiers, the presence of such noisy features will be harmful, resulting in an increase in training time and error rates. Based on different evaluation approaches, feature selection heuristics can separate the useful features from the unimportant ones. The selected features can deepen our knowledge about the nature of different data sources.

In our experiment, different machine learning classifiers namely Naïve Bayes (NB), Support Vector Machines (SVM), Decision tree algorithm (C4.5), K-Nearest Neighbour (KNN) could be used. The algorithms could be trained and tested on the Selected Feature Set and the Full-feature Set

The Accuracy (Acc) is used as evaluation metrics to assess the performance of the system.

3.3 Dataset

We crawled Alexa's top 100 India sites for creating the dataset and to generate training and ground truth labels for a collection of URL data, we compared URLs against EasyList filters. EasyList is a list containing possible keywords that an advertisement may contain. We checked each URL for presence of a keyword in EasyList as to classify it as ad or non-ad.

3.3.1 Crawler Development

A Web crawler, sometimes called a spider or spiderbot and often shortened to crawler, is a program that systematically browses the World Wide Web, typically for the purpose of Web indexing (web spidering).

Web search engines and some other sites use Web crawling or spidering software to update their web content or indices of others sites' web content. Web crawlers copy pages for processing by a search engine which indexes the downloaded pages so users can search more efficiently.

Generally, the advertisements on a webpage are contained in the script tag. We developed a crawler program which accessed the given sites page source and downloaded it. Then, it separated the script tag and then the source URL of that script. Then it stored the URLs in a file for further processing.

Development of Crawler –

We selected the Alexa Global Top 500 from February 2018 list of popular websites in our study as our dataset input. A total of 500 pages were examined in our experiment and almost 300 (63%) of them displayed some forms of advertising.

3.3.2 Feature Set

From the stored URLs we were able to extract following features for the dataset –

1. **Textual Length of URL**: The length of a URL without considering ‘www.’.

2. **Numeric Tokens Count**: A binary value indicating if there are any numeric tokens in a given URL.

3. **Dots Occurrences**: The number of dots in given URL.

4. **Dash Count in Hostname**: The number of dash occurrences in the hostname of a given URL.

5. **Longest Token Length**: The length of longest token in a given URL.

6. **Average Token Length**: The average length of all the tokens in a given URL.

7. **URL Token Count**: The number of tokens in a given URL.

8. **URL Pattern-based Features**: We want to check if there is any specific pattern for online advertisement URL.

9. The following features are considered as URL pattern-based features.
    a. **Letter-digit-letter**: If there is a digit presents between two letters in the given URL.

10. **Special characters**: A binary value indicating if there is any suspicious symbol in a given URL, such as “@;”.

11. **Number of special characters**: The number of symbols in a given URL, such as “() [] // - + =. / ? : ! ,”

3.4 Bag of Words Model

A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modelling, such as with machine learning algorithms.
The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents. A bag-of-words is a representation of text that describes the occurrence of words within a document.

It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

It is called a "bag" of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.
A very common feature extraction procedures for sentences and documents is the bag-of-words approach (BOW). In this approach, we look at the histogram of the words within the text, i.e. considering each word count as a feature. The intuition is that documents are similar if they have similar content. Further, that from the content alone we can learn something about the meaning of the document.

The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words.

Example of the Bag-of-Words Model -

Let's make the bag-of-words model concrete with a worked example.

Step 1: Collect Data –

Below is a snippet of the first few lines of text from the book "A Tale of Two Cities" by Charles Dickens, taken from Project Gutenberg.
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,

For this small example, let's treat each line as a separate "document" and the 4 lines as our entire corpus of documents.

Step 2: Design the Vocabulary

Now we can make a list of all of the words in our model vocabulary.
The unique words here (ignoring case and punctuation) are:

- "it"

- “was”
- “the”
- “best”
- “of”
- “times”
- “worst”
- “age”
- “wisdom”
- “foolishness”

That is a vocabulary of 10 words from a corpus containing 24 words.

Step 3: Create Document Vectors

The next step is to score the words in each document.
The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model.
Because we know the vocabulary has 10 words, we can use a fixed-length document representation of 10, with one position in the vector to score each word.
The simplest scoring method is to mark the presence of words as a Boolean value, 0 for absent, 1 for present.
Using the arbitrary ordering of words listed above in our vocabulary, we can step through the first document (“It was the best of times “) and convert it into a binary vector.

The scoring of the document would look as follows:

- “it” = 1
- “was” = 1
- “the” = 1
- “best” = 1
- “of” = 1
- “times” = 1
- “worst” = 0
- “age” = 0
- “wisdom” = 0
- “foolishness” = 0

As a binary vector, this would look as follows:
1 [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

The other three documents would look as follows:
1 "it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
2 "it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
3 "it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]

All ordering of the words is nominally discarded and we have a consistent way of extracting features from any document in our corpus, ready for use in modelling.

New documents that overlap with the vocabulary of known words, but may contain words outside of the vocabulary, can still be encoded, where only the occurrence of known words is scored and unknown words are ignored.

Managing Vocabulary

As the vocabulary size increases, so does the vector representation of documents.
In the previous example, the length of the document vector is equal to the number of known words.
You can imagine that for a very large corpus, such as thousands of books, that the length of the vector might be thousands or millions of positions. Further, each document may contain very few of the known words in the vocabulary.
This results in a vector with lots of zero scores, called a sparse vector or sparse representation.
Sparse vectors require more memory and computational resources when modelling and the vast number of positions or dimensions can make the modelling process very challenging for traditional algorithms.
As such, there is pressure to decrease the size of the vocabulary when using a bag-of-words model.

There are simple text cleaning techniques that can be used as a first step, such as:

- Ignoring case
- Ignoring punctuation
- Ignoring frequent words that don't contain much information, called stop words, like "a," "of," etc.
- Fixing misspelled words.
- Reducing words to their stem (e.g. "play" from "playing") using stemming algorithms.

A more sophisticated approach is to create a vocabulary of grouped words. This both changes the scope of the vocabulary and allows the bag-of-words to capture a little bit more meaning from the document.
In this approach, each word or token is called a "gram". Creating a vocabulary of two-word pairs is, in turn, called a bigram model. Again, only the bigrams that appear in the corpus are modelled, not all possible bigrams.
An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (more commonly called a trigram) is a three-word sequence of words like "please turn your", or "turn your homework".

For example, the bigrams in the first line of text in the previous section: "It was the best of times" are as follows:

- "it was"
- "was the"
- "the best"
- "best of"
- "of times"

A vocabulary then tracks triplets of words is called a trigram model and the general approach is called the n-gram model, where n refers to the number of grouped words.
Often a simple bigram approach is better than a 1-gram bag-of-words model for tasks like documentation classification.
a bag-of-bigrams representation is much more powerful than bag-of-words, and in many cases proves very hard to beat.

Scoring Words

Once a vocabulary has been chosen, the occurrence of words in example documents needs to be scored.
In the worked example, we have already seen one very simple approach to scoring: a binary scoring of the presence or absence of words.

Some additional simple scoring methods include:

- **Counts**. Count the number of times each word appears in a document.
- **Frequencies**. Calculate the frequency that each word appears in a document out of all the words in the document.

- **Evaluation Metrics -**

TF-IDF

A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much "informational content" to the model as rarer but perhaps domain specific words.
One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like "the" that are also frequent across all documents are penalized.
This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:

Term frequency

In the case of the term frequency tf(t,d), the simplest choice is to use the raw count of a term in a document, i.e., the number of times that term t occurs in document d. If we denote the raw count by $f_{t,d}$, then the simplest tf scheme is $tf(t,d) = f_{t,d}$. Other possibilities include[5]:128

- Boolean "frequencies": tf(t,d) = 1 if t occurs in d and 0 otherwise;
- term frequency adjusted for document length : $f_{t,d} \div$ (number of words in d)
- logarithmically scaled frequency: $tf(t,d) = \log (1 + f_{t,d})$;[6]
- augmented frequency, to prevent a bias towards longer documents, e.g. raw frequency divided by the raw frequency of the most occurring term in the document:

Inverse document frequency

The inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient).

5. Bag of Words Model

1. Logistic Regression

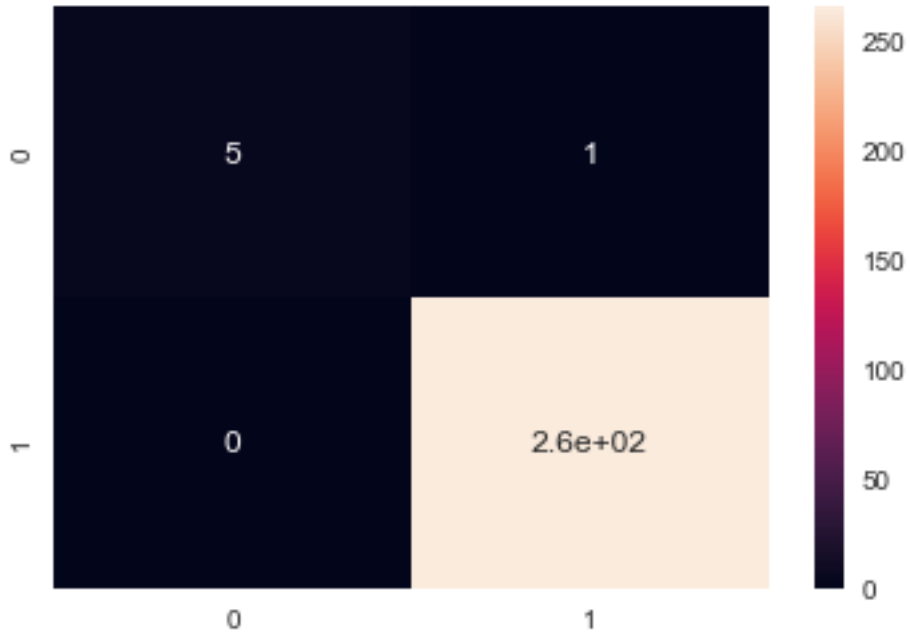
```
0.996309963099631
             precision    recall  f1-score   support

          a       1.00      0.83      0.91         6
          n       1.00      1.00      1.00       265


avg / total       1.00      1.00      1.00       271

Out[15]:
(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
         dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
         lowercase=True, max_df=1.0, max_features=None, min_df=1,
         ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
         stop_words=None, strip_accents=None, sublinear_tf=False,
         token_pattern='(?u)\\b\\w\\w+\\b',
         tokenizer=<function getTokens at 0x0000017E7203D840>, use_idf=True
,
         vocabulary=None),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=N
one,
             max_features=None, max_leaf_nodes=None,
             min_impurity_decrease=0.0, min_impurity_split=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, presort=False, random_state=0,
             splitter='best'),
 array([[  5,   1],
```

```
       [  0, 265]], dtype=int64))
```



2. SVM

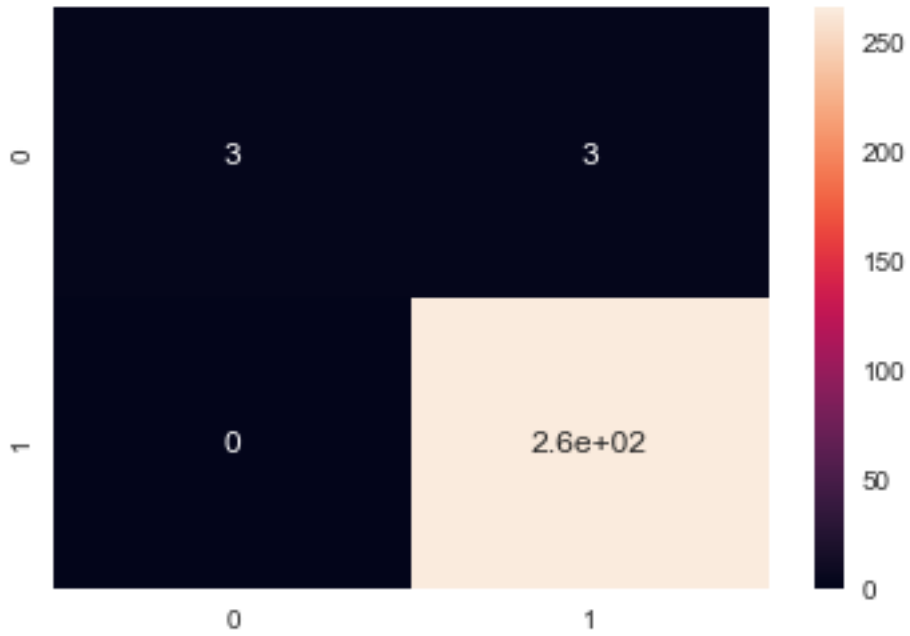
```
0.988929889298893
             precision    recall  f1-score   support

          a       1.00      0.50      0.67         6
          n       0.99      1.00      0.99       265

avg / total       0.99      0.99      0.99       271
```

```
Out[12]:
(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
         dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
         lowercase=True, max_df=1.0, max_features=None, min_df=1,
         ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
         stop_words=None, strip_accents=None, sublinear_tf=False,
         token_pattern='(?u)\\b\\w\\w+\\b',
         tokenizer=<function getTokens at 0x0000017E7203D840>, use_idf=True
,
```

```
        vocabulary=None),
 SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
   decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
   max_iter=-1, probability=False, random_state=0, shrinking=True,
   tol=0.001, verbose=False),
 array([[  3,   3],
        [  0, 265]], dtype=int64))
```



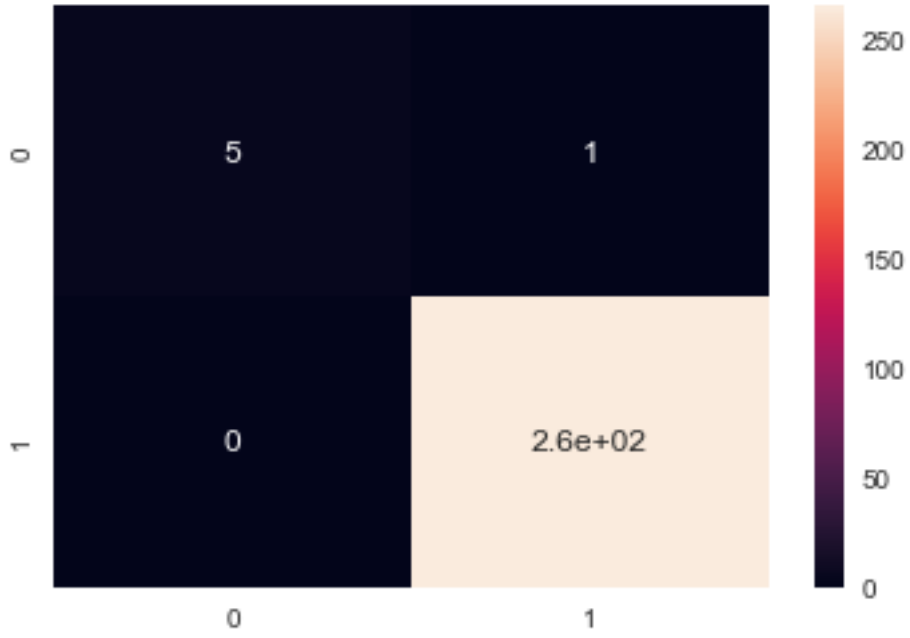
3. Decision Tree

```
0.996309963099631
             precision    recall  f1-score   support

          a       1.00      0.83      0.91         6
          n       1.00      1.00      1.00       265

avg / total       1.00      1.00      1.00       271
```

```
Out[14]:
(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
         dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
         lowercase=True, max_df=1.0, max_features=None, min_df=1,
         ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
         stop_words=None, strip_accents=None, sublinear_tf=False,
         token_pattern='(?u)\\b\\w\\w+\\b',
         tokenizer=<function getTokens at 0x0000017E7203D840>, use_idf=True
,
         vocabulary=None),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=N
one,
             max_features=None, max_leaf_nodes=None,
             min_impurity_decrease=0.0, min_impurity_split=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, presort=False, random_state=0,
             splitter='best'),
 array([[  5,   1],
        [  0, 265]], dtype=int64))
```



4. PERFORMANCE ANALYSIS

4.1 Performance Metrics

1. **Recall or Sensitivity or TPR (True Positive Rate)**:
   Number of items correctly identified as positive out of total true positives- TP/(TP+FN).

2. **Specificity or TNR (True Negative Rate):**
   Number of items correctly identified as negative out of total negatives- TN/(TN+FP).

3. **Precision:**
   Number of items correctly identified as positive out of total items identified as positive- TP/(TP+FP).

4. **False Positive Rate or Type I Error:**
   Number of items wrongly identified as positive out of total true negatives- FP/(FP+TN).

5. **False Negative Rate or Type II Error:**
   Number of items wrongly identified as negative out of total true positives- FN/(FN+TP).

6. **F1 Score**:
   It is a harmonic mean of precision and recall given by-
   F1 = 2*Precision*Recall/(Precision + Recall).

7. **Accuracy:**
   Percentage of total items classified correctly- (TP+TN)/(N+P).

Accuracy –

The first metric we are going to discuss is, perhaps, the simplest one, the accuracy. It answers the question:
"How often is the classifier correct?"

It can be obtained simply using the following formulae:

$$\text{Accuracy} = \frac{\#\text{correctly classified items}}{\#\text{all classified items}}$$

Confusion matrix –

The confusion matrix is another metric that is often used to measure the performance of a classification algorithm. True to its name, the terminology related to the confusion matrix can be rather confusing, but the matrix itself is simple to understand (unlike the movies).
In this post, let's focus in binary classifiers as with the spam filtering example, in which each email can be either spam or not spam. The confusion matrix will be of the following form:
The predicted classes are represented in the columns of the matrix, whereas the actual classes are in the rows of the matrix.

We then have four cases:

- **True positives (TP)**: The cases for which the classifier predicted 'ad' and the ads were actually benign ads.
- **True negatives (TN)**: The cases for which the classifier predicted 'not ads' and the ads were actually real.
- **False positives (FP)**: The cases for which the classifier predicted 'ads' but the ads were actually real.
- **False negatives (FN)**: The cases for which the classifier predicted 'not ads' but the ads were actually not ads.

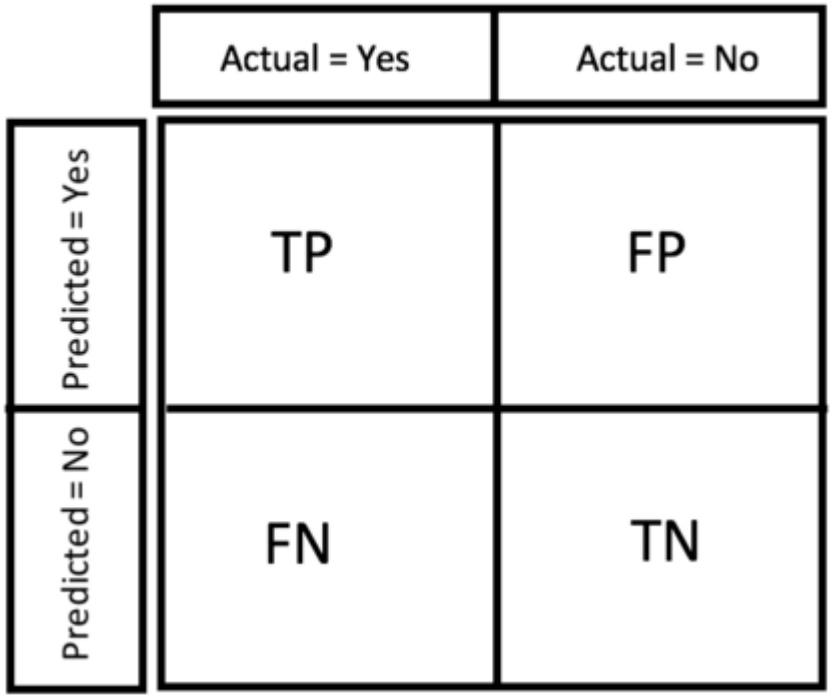


Figure 4.1 Confusion Matrix

Precision -

It answers the question:

"When it predicts the positive result, how often is it correct?"

This is obtained by using the following formulae:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Precision is usually used when the goal is to limit the number of false positives (FP).

Recall –

It answers the question:

"When it is actually the positive result, how often does it predict correctly?"

This is obtained by using the following formulae:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Recall is usually used when the goal is to limit the number of false negatives (FN

f1-score -

This is just the harmonic mean of precision and recall:

$$f_1\text{-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

ROC curve –

A more visual way to measure the performance of a binary classifier is the receiver operating characteristic (ROC) curve. It is created by plotting the true positive rate (TPR) (or recall) against the false positive rate (FPR), which we haven't defined explicitly yet:

$$\text{FP rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

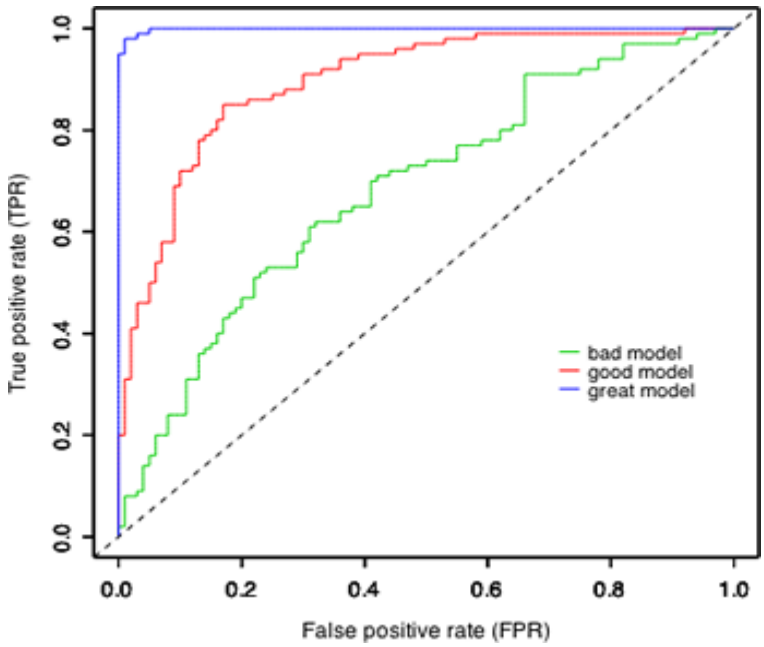


Figure 4.2 ROC Curve

4.2 Classification and Evaluation

1. Logistic Regression

Output – Accuracy - 0.996309963099631

	precision	recall	f1-score	Support
a	1.00	0.83	0.91	6
n	1.00	1.00	1.00	265
avg / total	1.00	1.00	1.00	271

Table 4.1 Output for Logistic Regression

2. SVM

Output – Accuracy-0.988929889298893

	precision	Recall	f1-score	Support
a	1.00	0.50	0.67	6
n	0.99	1.00	0.99	265
avg / total	0.99	0.99	0.99	271

Table 4.2 Output for SVM

3. Decision Tree

Output – Accuracy-0.996309963099631

	precision	Recall	f1-score	Support
a	1.00	0.83	0.91	6
n	1.00	1.00	1.00	265
avg / total	1.00	1.00	1.00	271

Table 4.3 Output for Decision Tree

	Performance Metrics			
Algorithm	**Accuracy**	**Precision**	**Recall**	**Support**
Logistic Regression	0.99630	1.00	1.00	271
SVM	0.98	0.99	0.99	271
Decision Tree	0.996	1.00	1.00	271

Table 4.4 Comparison of different Algorithms.

5. CONCLUSION

Nowadays, online advertisement is one of the largest annoyances for users in web surfing and reading. Current tools are using a large list of regular expression filters for matching every requested URL to detect the
malicious-ads. So, it is necessary to have an automated solution that can be adapted to thousands evolving adverts easily and fast. To tackle this issue, in this research we proposed a machine learning model which will detect whether an ad is malicious or not on the basis of the URL. This helps us to know the malicious ads before downloading its content.

While the traditional ad blocking model has been successful and efficient since its creation in 2002, the manual efforts behind the scenes might be better placed elsewhere. Undesirable resources that could be irritating, privacy-violating or maliciously loaded as ads, need a more automatic way to be detected in the future. The landscape of ads is continuously changing either to embody more security threats or simply to combat ad blockers. We have developed a machine learning based classifier for ads which was able to automatically learn currently known ads with a high accuracy and up to 50% of new ads that would otherwise necessitate manual filter creation. Our classifier was also able to detect a large number of ad and tracking URLs that current filter lists aren't able to identify, either due to their lack of defining URL tokens from which filters can be constructed or because the filters matching these haven't been manually identified yet. We believe that this line of research has the potential to further enable user choice regarding what ads, tracking beacons, and other undesirable web assets are loaded on their machines, improving several aspects of the experience and web security for web users.

6. FUTURE SCOPE

While we were able to achieve high accuracy, precision and low false positive rates on old and new advertisements, ad URLs are constantly updating and changing their structure to combat ad blockers and hence there needs to be a way for a classifier to periodically train on new data. To improve the new-ad accuracy, the classifier would need to be able to keep up with the new types of ads and update its model periodically to accommodate these new rules. A possible implementation would be to have a feedback mechanism browser extension, which allows a user to identify ad elements, similar to feedback-based spam filters in email clients. All the URLs requested within this element could be added to the trainer with their features computed. To maintain a balance of examples between the two classes, for every ad URL label incoming, one non-ad URL can be added to the training data. This sort of feedback mechanism learning could be applied to our proposed semi-supervised online learning approach with initial training done on the most recent EasyList data.

7. References

[1] Piotr L. Szczepanski, Adrian Wisniewski and Tomasz Gerszberg, "An Automated Framework With Application To Study URL Based Online Advertisement Detection", Journal of Applied Mathematics, Statistics and Informatics (JMASI), Issue 9 ,2013.

[2] Shih L. K. and Karger D.R., "Using URL's and Tables Layout for Web Classification Task", WWW 13th Conference, pages 193-202, 2004.

[3] Zhang J. Qin, Yan Q., "The Role of URL's in Objectionable Web Content Categorization", IEEE/WIC/ACM Conference on Web Intelligence, page 277-283, 2006.

[4] Eda Baykan, Monika Henzinger, Ingmar Weber, "A Comprehensive Study for URL-Based Web Page Language Classification", ACM Transactions on the Web, Volume 7, No. 1, Article 3, March 2013.

[5] Sruti Bhagavatula, Christopher Dunn, Chris Kanich, Minaxi Gupta, Brian Ziebert, "Leveraging Machine Learning to Improve Unwanted Resource Filtering", ACM Artificial Intelligence and Security, page 95-102, 2014.

[6] Kan M.Y. and Thi H.O.N., "Fast Webpage Classification Using URL Features", 14th Conference on Information and Knowledge Management, page 325-326, 2005.

[7] Krammer V., "An Effective Defense Against Intrusive Web Advertising", IEEE 6th conference on Privacy, Security and Trust, page 3-14, 2008.

ACKNOWLEDGEMENT

A project is creative work of many minds. A proper synchronization between individual is must for any project to be completed successfully. We are extremely great full to all the helping contributors for the successful presentation of our project.

We owe our deep gratitude to our guide Mrs. Anjana N.Ghule. She gave us valuable guidance with a touch of inspection and motivation to help our way through quite a substantial barrier between unwieldy early ideas and something that resembles a fine project. Her invaluable presence was a great boost for all of us in achieving up a goal.

We are greatly thankful to principal Dr. P.B. Murnal providing necessary Facilities for carrying out this work.

We are thankful to all our staff's members of our department for their co-operation. we are thankful to all library staff, which extended every facility to us. Last but not least thankful to all those who helped us in completing this project work. To them, please accept our apology and thank you.

Aditya Rajkumar Mahamuni
[BE15F06F032]

Mayur Deshmukh
[BE15F06F011]

Vinod Yevatikar
[BE15F06F059]

Jayesh Mutha
[BE15F06F038]

Shubham Bakhliwal
[BE15F06F006]