Contents

Introduction

The problem of spam has been around since the early days of e-mail services and has since evolved to numerous forms. Spam can vary by platform (e-mail, instant messaging, social network messaging), by intentions (from advertisement to phishing and ransom), by distribution targets (from a single individual or general public), etc.

While the number of messaging domains is growing rapidly, the prime purpose of spam remains the same: to reach out to the recipient by any means or to trick him into acting on the message by disguising it as a legitimate message. This leads to a very clear practical definition of spam from user's perspective as *any unsolicited message*, whether it acknowledges its origin (e.g. advertisement) or is crafted to gain the user's trust (e.g. phishing and ransom attacks).

Given the large number of messaging systems to which spam is applicable, the problem of detection of unwanted correspondence has evolved along with distribution methods and now includes additional metadata like attached media and hyperlinks. While they may seem to increase the complexity of spam detection, they in fact give a basis for additional means of filtering which are often much easier to perform than the raw text classification.

While the message metadata can provide additional information, in practice text classification is inevitable. It is usually the final step in classification process as it tends to be most expensive. Since the metadata processing is different from case to case, it is safe to say that the general problem of spam filtering comes down to the problem of binary text classification. Therefore, it makes sense to develop universal methods for spam classification of text messages that can be later adopted to any domain.

In this paper we give an overview of a number of common statistical methods of performing spam filtering on text messages. In particular we focus on Bayesian

classifier - one of the most common and effective approaches to spam filtering. We analyze existing approaches and develop a number of improvements that provide stronger theoretical results.

Additionally, we analyze common problems for training on text messages like feature selection. Finally, we provide a performance comparison of the detailed algorithms in practice.

Chapter 1

Statement of the Problem

1.1 Scope and Applications

A social networking service (or SNS) is a platform to build social networks or social relations among people who share similar interests, activities, backgrounds or real-life connections (definition from Wikipedia). The ultimate purpose of any social networking service is fast and efficient exchange of information, often with intention to present it to the largest audience possible.

The vast majority of most popular social network services rely on text messages as the main form of exchange of information. Because of their openness, social networks can be extremely useful in spreading malicious messages across wide audiences, both via private (addressed to a particular individual) and public messages. Conceptually social network spam is no different from e-mail spam as private messaging services of popular social networks are equivalent in their functionality to e-mail. Hence, we can generalize the problem of classifying spam messages for both these cases.

In general, the problem of spam detection depends heavily on the application's domain and can benefit from additional metadata available along with the text message. For example, in case of e-mails the mail header is the source of metadata. Modern spam filtering systems detect the vast majority of malicious mails by simply checking the sender's reputation before proceeding with analysis of the message body.

This, of course, applies to all messaging services. Maximum effectiveness can not be achieved without using all available data in addition to the message text.

However, in most cases text analysis is the second stage preceded by a domain-specific filter. Therefore, we can further focus on statistical classification of spam for text messages without specific constraints.

Let us give a number of definitions that will be used throughout the paper.

M - the set of all messages.

$L \in M$ - the set of legitimate messages.

$S = m \setminus L$ - the set of spam messages.

Note that we will often speak of M, L and S in context of the set of training messages.

$f : M \to \{S, L\}$ - classification function that given a message determines its class. This function is the ultimate goal of any classification algorithm.

$T = \{(m_1, c_1), (m_2, c_2), ..., (m_k, c_k)\}, m_i \in M, c_i \in \{S, L\}, 1 \leq i \leq k$ - the training set of already classified messages.

$T_c = T \cap c, c \in \{L, S\}$ - training set of legitimate (T_L) or spam (T_S) messages.

$x = (x_1, x_2, \ldots, x_n), x_i \in \{0, 1\}, 1 \leq i \leq n$ - binary feature vector of a message. Each coordinate specifies whether a specific feature applies to the message.

1.2 Feature Selection

The entities we need to classify are text messages that are given in the form of strings. Raw strings are not convenient objects to handle in this case. Most machine learning algorithms can only classify numerical objects or otherwise require a distance metric or other measure of similarity between the objects.

Before proceeding with machine learning we have to convert all messages to numerical vectors called *feature vectors*, and then classify these vectors. The simplest example of a feature vector is the vector of the numbers of occurrences of certain words in a message or the vector of flags specifying whether each word occurs in the message or not.

Extraction of features usually means that some information from the original message is lost. On the other hand, the way feature vector is chosen is crucial for the performance of the filter. If the features are chosen so that there may exist a spam message and a legitimate mail with the same feature vector, then any machine learning algorithm will make mistakes no matter how good it is.

Even the ultimate feature vector containing frequencies of *all* words in the message does not preserve semantical information. This problem however is out of scope and is in the domain of Natural Language Processing. In most practical applications the most basic vector of word frequencies or its modification is sufficient.

Note that at the stage of feature selection it is possible to include the features from the available metadata along with features from message text. However, we will only focus on plain text as any of the described algorithms can be easily expanded to include more features.

Chapter 2

Basic Methods

2.1 Naive Bayes Classifier

Consider the simple case of text classification based on the presence or absence of just one word W. Suppose we know that the word W only occurs is spam messages. This gives us confidence that any message containing W is spam. This approach can be generalized to the probability of a message feature vector occurring in the message.

Suppose we have two classes L and S corresponding to legitimate and spam messages, and that there is a known probability distribution of feature vectors $P(x|c), c \in \{L, S\}$. In general it is hard to define such distribution, but it is often possible to provide an approximation. What we need to obtain is the class that the given message belongs to, or the probability $P(c|x)$. This can be done using the Bayes' formula

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} = \frac{P(x|c)P(c)}{P(x|L)P(L) + P(x|S)P(S)}$$

.

where $P(x)$ is the a-priori probability of a message with feature vector x and $P(c)$ is the probability of class c, i.e. the probability that any given message belongs to c. Given the values $P(c)$ and $P(x|c)$ for $c \in \{L, S\}$ one can calculate the probability $P(c|x)$ which can then be used in a classification rule.

The most basic classification rule is to classify message to the category with bigger probability.

Definition 2.1.1. *Maximum a-posteriori probability (MAP) rule: if* $P(S|x) >$

$P(L|x)$ then classify x as spam, otherwise classify as legitimate message.

The MAP rule can be transformed to

If $\frac{P(x|S)}{P(x|L)} > \frac{P(L)}{P(S)}$ then classify x as spam, otherwise as legitimate message.

The ratio $\frac{P(x|S)}{P(x|L)}$ is known as the *likelihood ratio* and is denoted as $\Lambda(x)$.

This approach can be too simplistic for certain applications. For example, in case of e-mail spam filtering, false positives (classifying legitimate message as spam) are usually much more unwanted than false negatives (classifying spam as legitimate message). The following generalization allows to take such restrictions into account.

Definition 2.1.2. *A cost function $\mathcal{L}(c_1, c_2)$ denotes the cost of misclassifying a message of class c_1 as the one belonging to class c_2.*

Then we can express the expected risk of classifying a message x belonging to class c in the above terms.

Definition 2.1.3. *The function $R(c|x) = \mathcal{L}(S, c)P(S|x) + \mathcal{L}(L, c)P(L|x), x \in M, c \in \{L, S\}$ is called the risk function.*

Now we can define a natural classification rule in terms of expected risk.

Definition 2.1.4. *Bayes' classification rule: if $R(S|x) < R(L|x)$ then classify x as spam, otherwise as legitimate message [2].*

It can be shown that Bayesian classifier f minimizes the average risk

$$R(f) = \int \mathcal{L}(c, f(x))dP(c, x) = P(L) \int \mathcal{L}(L, f(x))dP(x|L) + P(S) \int \mathcal{L}(S, f(x))dP(x|S)$$

so in this sense Bayesian classifier already is optimal [1].

Naturally, the loss of classifying the message correctly is zero, thus $\mathcal{L}(S, S) = \mathcal{L}(L, L) = 0$. Then the Bayes's classification rule can be rewritten as

$$\text{If } \Lambda(x) > \lambda \frac{P(L)}{P(S)} \text{ classify as spam otherwise as legitimate message.}$$

Here $\lambda = \dfrac{\mathcal{L}(L,S)}{\mathcal{L}(S,L)}$ is the additional parameter that specifies the risk of misclassifying legitimate messages as spam. As the value of λ increases, the classifier produces fewer false positives.

While the classification process is straightforward, the practical applications of Bayes's classifier are limited by our ability to approximate the a-priori probabilities $P(x|c)$ and $P(c), c \in \{L, S\}$ from the training data. Therefore, while the Bayes's classifier is optimal in the sense of minimizing the loss of classification for given a-priori probabilities, the quality of spam detection depends on the feature selection and approximation of these probabilities.

$P(L)$ and $P(S)$ can be easily approximated by the ratio of legitimate and spam messages respectively. $P(x|c)$ is non-trivial and depends on the contents of selected feature vector. Consider the simplest case where the feature vector x_w is 1 if the message contains w and 0 otherwise. Then the probability $P(x_w = 1|S)$ can be approximated by the ratio of spam messages containing w to the ratio of all spam messages in a training set. This is sufficient to be used by the Bayes's classifier, so we can outline the training and selection process for this model.

Training process

1. Calculate probabilities $P(c)$, $P(x_w = 0|c), P(x_w = 1|c), c \in \{L, S\}$.
2. Using Bayes's formula calculate $P(c|x_w = 0)$ and $P(c|x_w = 1)$.
3. Calculate $\Lambda(x_w), x_w = 0, 1$, calculate $\lambda \dfrac{P(L)}{P(S)}$ and store these values.

Classification process

1. Determine feature vector x_w for message m.
2. Retrieve the stored value $\Lambda(x_w)$.
3. Use Bayes's decision rule to determine class of the message.

Now we need to generalize this classifier to include more features than just the presence of a single word. The simplest way (and a very common one) is to choose a set of most common words $w_1, w_2, \ldots, w_n$ and define the feature vector $x = (x_1, x_2, \ldots, x_n), x_i =$, $x_i = 1$ if the message contains w_i, $x_i = 0$ otherwise.

The problem with this approach is that it requires calculation and storing of all possible values of the feature vector, and there are 2^n such vectors, which is not feasible. A common way to remove this requirement is to assume that the individual components of the vector are independent [1]. This assumption is not formally correct, but in practice it is a good compromise between formal correctness and computational requirements. We will consider other options in later chapters.

Because of independence of features:

$$P(x|c) = \prod_{i=1}^{n} P(x_i|c)$$

$$\Lambda(x) = \prod_{i=1}^{n} \Lambda_i(x_i)$$

This classifier is known as Naive Bayesian Classifier due to assumption of independence of features. Training and classification are very simple computationally.

Training process

1. For all $w_i \in W$ calculate and store $\Lambda_i(x_i), x = 0, 1$.
2. Calculate and store $\lambda \frac{P(L)}{P(S)}$.

Classification process

1. Determine feature vector x for message m.
2. calculate $\Lambda(x)$ using the stored values $\Lambda_i(x_i)$.
3. Use Naive Bayes's decision rule to determine class of the message.

In terms of word selection for the feature vector, usually words that are too common or too rare are excluded. For simple cases when performance is not critical, all words form the training set can be used. In later chapters we will consider ways to select words with maximum mutual information.

Another benefit of naive Bayesian filter is that it is very easy to expand the feature vector to include additional available metadata. In case of e-mails, for

example, it would be contents of e-mail headers. It is possible to include additional components either to the calculation of the a-priory probability of the vector or to combine the risk of Bayesian classifier with additional risk calculated from metadata when making a decision.

2.2 k Nearest Neighbors Classifier

k Nearest Neighbors (or k-NN) is a modification of the classical Nearest Neighbors algorithm. The idea behind this classifier is to first define a metric on feature vectors and then classify the message according to classes of k nearest messages in the training set. The metric is often chosen to be Euclidean, but Hamming or l_p can also be used for this purpose.

Training process

1. Store feature vectors of training messages in two sets L and S.

Classification process

1. For message with feature vector x determine k nearest neighbors from messages in the training set. If there are more legitimate messages among them, classify x as legitimate message, otherwise classify as spam.

Since the algorithm does not require any preprocessing of the training dataset, the training process is trivial. The classification process, however, requires calculation of distances to all messages in the training set, and for feature vectors of length m the most trivial implementations take $O(mn)$ time for set of n messages in case of Hamming or l_p metrics. Performing indexing on the training set can decrease the running closer to $O(n)$ [1]. However, if the size of the set increases over time, the algorithm might not be feasible in practice for certain applications.

The k-NN classifier is widely applicable in general classification problems, partially because it is one of so called *universally consistent* rules. Consider the training set s_n of n samples, and let use denote the k-NN classifier over that set as f_{s_n}. Similar to Bayesian classifier, we can determine the average risk $R(f_{s_n})$ of the classifier. The risk value is always greater than or equal than the Bayesian risk R_* (recall that Bayesian classifier is optimal in this sense), however for large values of n $R(f_{s_n})$ will be close to R_*.

Definition 2.2.1. *A classification rule is called consistent if the expectation of the average risk $E(R_n)$ converges to the optimal (Bayesian) risk R_* as n goes to infinity:*

$$E(R_n) \xrightarrow[n\to\infty]{} R_*$$

. The classification rule is called universally consistent if it is consistent for any distribution of (x, c).

Theorem 2.2.1 (Stone, 1977)**.** *If* $k \to \infty$ *and* $\frac{k}{n} \to 0$*, then* k*-NN classification rule is universally consistent.*

Consistency of k-NN rule allows to increase the quality by increasing the size of the training set. Stone theorem guarantees that as the size of the training set increases with constant value of k, the selection of messages for the training set does not matter. In addition to this, small values of k prevent quadratic complexity $O(n^2)$ when computing nearest neighbors.

Despite theoretical results, k-NN classifiers are performing worse than competition in practice in spam classification and are computationally expensive.

2.3 Artificial Neural Network Classifier

Artificial neural networks (ANN) is a family of models inspired by biological neural networks which are widely used in classification, regression and density estimation by approximating functions that can depend on a large number of inputs and are generally unknown. A neural network is a complex function that may be decomposed into smaller units called neurons and represented graphically as a network. Many functions fall under such criteria, however the most common kinds of neurons are perceptron and multilayer perceptron.

The perceptron produces a linear function of the feature vector $f(x) = w^T x + b$ where $f(x) > 0$ for vectors of one class and $f(x) < 0$ for vectors of another class. Here w is the vector of weights, or *bias*, $w = (w_1, w_2, \ldots, w_n)$. This vector will be determined by the training process.

If we denote the classes by number -1 and +1, we can use $d(x) = sign(w^T x + b)$ as decision function. This allows us to represent the decision function graphically as a neuron with n inputs and a single output. A system of one perceptron is an example of the simplest neural network.

Suppose the feature vector is two-dimensional, $x \in \mathbb{R}^2$. Then we can represent these feature vectors as points on the plane. Then the decision function can be represented as a line dividing the plane in two parts, each corresponding to one of the classes. Similarly, the decision boundary for three-dimensional feature vectors is a plane, etc. In general, for n-dimensional feature vector the decision boundary is a n - dimensional hyperplane.

The learning process for a perceptron is iterative. The initial values of parameters (w_0, b_0) can be arbitrary, as they are updated on each iteration. On the k-th iteration of the algorithm a training sample (x, c) is chosen such that the current decision function does not classify it correctly, i.e. $sign(w_k^T x + b_k) \neq c$. Then the parameters (w_k, b_k) are then updated according to the rule:

$$w_{k+1} = w_k + cx$$

,

$$b_{k+1} = b_k + c$$

.

The algorithm terminates when a decision function that correctly classifies all training set is found. If the training set is linearly separable, the perceptron algorithm converges. It is known as Perceptron Convergence Theorem proven by Frank Rosenblatt in 1962 [4]. If, however, the set is not linearly separable, the algorithm will never converge. In this case it is possible to still use the perceptron, but the training loop needs to stop when the number of misclassification becomes small.

We can now outline the training and classification phases of the perceptron.

Training process

1. Initialize the values of w and b with random values or 0.
2. Find a sample from the training set (x, c) such that $sign(w^T x + b) \neq c$. If there are no such samples, terminate as the training is completed and all training samples are being classified correctly, else proceed to the next step.
3. Update (w, b) with new values $w := w + cx$, $b := b + c$ and go to the previous step.

Classification process

1. For message with feature vector x classify it as $sign(w^T x + b)$.

As mentioned before, perceptrons can be combined in multiple layers to form *multilayer perceptrons* which are non-linear classifiers. Neurons of the first layer which takes in the input parameters are called *input neurons*, similarly neutrons of the last layer which provide the function result value are called *output neurons*. All layers between input and output are called *hidden layers*.

Each neuron in the networks is similar to a perceptron: for input vector $x = (x_1, x_2, \ldots, x_n)$ it calculates output value by the formula

$$o = \phi(\sum_{i=1}^{n} w_i x_i + b)$$

where w_i and b are weights and bias of the neuron respectively, ϕ is a nonlinear function that approximates binary output of the perceptron. Most often $\frac{1}{1+e^{ax}}$ or

$\tanh(x)$ are used as ϕ as they tend to give a good approximation while being mathematically convenient.

Like in the case of a single perceptron, training of a neural network is searching for the values of weights and biases for all neurons that minimize the output error. Let us denote $f(x)$ as the output of the neural network. Then for training samples $(x_i, c_i), 1 \leq i \leq k$ the training has to minimize the *total training error*

$$E(f) = \sum_{i=1}^{k} |f(x_i) - c_i|^2$$

.

An iterative algorithm can be used to perform this minimization. The most common one is the gradient descent which in case of neural networks is called *error backpropagation.* The theory of backwards propagation of errors is well developed and has many implementations in practice [3].

The main reason to use multiple layers of neurons is that the multilayer neural network is a non-linear classifier. As a result, they are applicable for tasks with training data that is not linearly separable, particularly when the number of features is relatively small. However, in case of spam detection with multiple words being used as features the data is often linearly separable, thus using neural network will have no noticeable benefits over a simple perceptron.

Performance of the neural network is proportional to the number of neurons. Thus, the large number of features directly impacts performance as it translates to increased number of input neurons and thus the complexity of the network in total. In practice the number of features would have to be more strictly limited than in case of a perceptron, which for spam detection means the trade-off between non-linear decision boundaries and the amount of information loss.

Because of the above reasons and due to the large number of parameters that require tuning. the multilayer perceptron is hard to use in practice for spam detection. It has been successfully used for that purpose [1], but it is not easily applicable in general case as it is hard to reconfigure. For the purposes of this paper we shall focus on a simple perceptron.

2.4 SVM Classifier

Support Vector Machines (SVM) is a class of widely used algorithms for classification and regression. The theoretical foundation of SVM is the Statistical Learning Theory that gives certain guarantees of performance. Let us consider classification problem with SVM for linearly separable data.

SVM works in a similar manner to perceptron in terms of finding a linear boundary that separates test data according to their classes. However, the purpose of SVM is not to find any of these boundaries if they exist, but to find the maximal margin separating hyperplane, for which the distance to the closest training sample is maximal.

Definition 2.4.1. *Let* $X = \{(x_i, c_i)\}$, $x_i \in \mathbb{R}^n$, $c_i \in \{-1, +1\}$ *be the set of training samples. Suppose* (w, b) *is a separating hyperplane* $sign(w^T x_i + b) = c_i$ *for all* $1 \leq i \leq k$. *The margin* m_i *of a training sample* (x_i, c_i) *with respect to the separating hyperplane is the distance from* x_i *to the hyperplane*

$$m_i = \frac{|w^T x_i + b|}{||w||}$$

.

The margin m *of the separating hyperplane for training set* X *is the smallest margin of an instance in the training set*

$$m = \min_i m_i$$

.

The maximal margin separating hyperplane for training set X *is the separating hyperplane with maximal margin with respect to the training set [1].*

Because the hyperplane given by parameters (x, b) is the same as the hyperplane given by parameters (kx, kb), we can safely bound our search by only considering canonical hyperplanes for which $\min_i |w^T x_i + b| = 1$. It is possible to show that the optimal canonical hyperplane has minimal $||w||$, and that in order to find a canonical hyperplane it suffices to solve the minimization problem: minimize $\frac{1}{2} w^T w$ under conditions

$$c_i(w^T x_i + b) \geq 1, i = 1, 2, \ldots, k$$

The problem may be transformed to a certain dual form: maximize

$$L_d(\alpha) = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{k} \alpha_i \alpha_j c_i c_j x_i^T x_j$$

with respect to dual variables $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k)$, $\alpha_i \geq 0 \forall i$ and $\sum_{i=1}^{k} \alpha_i c_i = 0$.

This is a classical quadratic optimization problem. It mostly has a guaranteed unique solution, and there are efficient algorithms for finding this solution. Once we have found the solution α, the parameters (w_o, b_o) of the optimal hyperplane are determined as

$$w_o = \sum_{i=1}^{k} \alpha_i c_i x_i$$

,

$$b_o = \frac{1}{c_m} - w_o^T x_k$$

where m is an arbitrary index for which $\alpha_m \neq 0$.

The resulting hyperplane is defined by the training samples that are at minimal distance to it, so called *support vectors*. Training process is quite complex, while classification is extremely simple.

Training process

1. Find α that solves the dual problem (maximizes L_d under named constraints).
2. Determine w and b for the optimal hyperplane and store the values.

Classification process

1. For message with feature vector x classify it as $sign(w^T x + b)$.

2.5 Boosting

AdaBoost.

Chapter 3

Modified Bayesian Classifier

3.1 Non-Naive Bayesian Classification

3.1.1 Overview Recall the naive Bayesian classifier described earlier. Let us ignore feature selection for now and instead only consider classification of feature vectors. Bayesian classifier is optimal in the sense of minimization of expected risk, however it is a common practice to assume that individual components of the feature vector are independent. Such assumption, while significantly simplifying implementation and improving training and classification performance, does compromise the optimality of the classifier. In this chapter we will analyze the possibility of an effective Bayesian classifier without assumption about feature independence and will develop an algorithm that allows to balance learning/classification speed with optimality guarantees.

Let us return to Bayesian model for the case of a single feature.

Training process

1. Calculate probabilities $P(c)$, $P(x_w = 0|c), P(x_w = 1|c), c \in \{L, S\}$.
2. Using Bayes's formula calculate $P(c|x_w = 0)$ and $P(c|x_w = 1)$.
3. Calculate $\Lambda(x_w), x_w = 0, 1$, calculate $\lambda\dfrac{P(L)}{P(S)}$ and store these values.

Classification process

1. Determine feature vector x_w for message m.

2. Retrieve the stored value $\Lambda(x_w)$.

3. Use Bayes's decision rule to determine class of the message.

In the naive version assumption about independence of features was made. With this assumption the required probability can be easily found from a-priori probabilities of separate features:

$$P(x|c) = \prod_{i=1}^{n} P(x_i|c)$$

Now we again need to generalize this classifier to include multiple features, however this time without making any assumptions about the underlying distribution. Let us ignore the feature selection for now as it will be described in the following chapter.

Consider the feature vector $x = (x_1, x_2, \ldots, x_n)$, $x_i = 1$ if the message contains w_i, $x_i = 0$ otherwise. As mentioned before, calculating and storing this value for all 2^n possible combinations of feature vector x is not feasible in practice, so this has to be done during classification process.

The probability $P(x|c)$ for any given $c \in \{L, S\}$ is equal to the ratio of messages in c with feature vector x to all messages in c. To calculate it efficiently, we need to be able to quickly determine the number of messages in the set that have specified feature vectors. This can be done in a number of ways.

3.1.2 Feature Trie To be able to perform lookup of the number of messages with specific feature vector, we shall store the known feature vectors in a trie where each node corresponds to one of the features, starting from x_1. Given a node at position x_i the left subtree will contain vectors with $x_i = 0$ and the right subtree - vectors with $x_i = 1$. The leaf nodes will contain the number of messages in the set with feature vectors that match the path from root to leaf. Since we need to calculate both $P(x|L)$ and $P(x|S)$, we shall maintain two tries, one for legitimate messages and one for spam.

With this representation adding a message to the trie takes time $O(n)$ (where n is the dimension of feature vector) since the length of the path from root to the corresponding leaf node is n. Therefore, the whole training process takes $O(nk)$, where k is size of the training set.

Lookup of the feature vector also takes linear time $O(n)$ since we only follow the path of length n from root to a leaf node. It is also easy to show that the lower bound is $O(n)$ because each of the feature vectors has to be examined at least once. Therefore, our algorithm is optimal.

The amount of extra storage required by the trie is $O(nk)$ (consider the set of one message). Obviously, some fractions of the paths will be shared (different messages with identical feature vectors will share the whole path). One way to increase the number of shared nodes is to sort the features in descending order by the number of messages that have the same value of this feature, then more messages will share common beginning of the path.

Of course, this is not the only way to store and perform lookup on feature vectors. We can also use a hash table. Calculating hash of the feature vector is $O(n)$. Construction and lookup are $O(nk)$ and $O(n)$ on average respectively, however they can be $O(n^2k)$ and $O(n^2)$ in the worst case. For the purposes of this paper we will focus on trie approach.

Let us outline the updated training and classification algorithms. Let $M(c, x), c \in \{L, S\}, x \in 2^{|n|}$ denote the power of subset of c with feature vector x.

3.1.3 Non-Naive Bayesian Classification Algorithm Let us now outline the updated algorithm for training and classification.

Training process

1. Calculate $|L|$ and $|S|$. Time complexity: $O(k)$.
2. Calculate likelihood ratio $\lambda\frac{P(L)}{P(S)}$. Time complexity: $O(1)$.
3. For both L and S construct corresponding tries that allow to perform lookup of $M(c, x)$. Time complexity: $O(nk)$. Space complexity: $O(nk)$.

Classification process

1. Determine feature vector x for message m. Time complexity: $O(|m|)$ where $|m|$ denotes the number of words in message m.
2. Perform lookup of $M(L, x)$ and $M(S, x)$ using the tries. Time complexity: $O(n)$.

3. Calculate $P(x|L) = \frac{M(L,x)}{|L|}$ and $P(x|S) = \frac{M(S,x)}{|S|}$. Calculate $\Lambda(x) = \frac{P(x|S)}{P(x|L)}$. Time complexity: $O(1)$.

4. Use Bayes's decision rule to determine class of the message. Time complexity: $O(1)$.

3.1.4 Generalizing Probability of Feature Vectors In case of naive approach only probabilities of separate features need to be stored. They are sufficient to determine the overall probability of the feature vector for all 2^n combinations, regardless of whether they are present in the training set. In the approach described above, however, only the information about concrete feature vectors from the training set is stored. While it is possible to calculate probability of any feature vector in the training set by the algorithm above, in its current version probability of any feature vector that is not in the training set to be in either L or S is 0. In fact, probability of any message m to belong to c will be non-zero if and only if the training set contains at least one message with identical feature vector.

This can be interpreted as excessive 'formality', i.e. the algorithm produces the optimal result, but in the context of given training set. We would naturally want to get a reasonable approximation of the probability of messages that are not present in the training set. Note that out of 2^n possible feature vectors at most k will be present in the constructed trie, and the probability that the algorithm will produce the expected result is $\frac{k}{2^n}$ at best. Therefore, without providing an alternative probability for unrecognized feature vectors these modifications will be useless in practice.

One possible solution is to fall back to naive Bayesian classifier for all feature vectors that are not present in the training set. In this case, our modified classifier will only produce different results for a small fraction of possible messages ($\frac{k}{2^n}$), making these modifications negligible in practice. For example, even if we assume that the training set contains k_u unique feature vectors, for $k_u <= 2^30$, (i.e. the training set is very unlikely to contain more than 1 billion messages in practice) n has to be not larger than $log_2(k) = 30$. Obviously, we want to include significantly more features.

3.2 Semi-Naive Bayesian Classification

At this point we have an optimal algorithm for any message such that its feature vector is present in the training set. In this section we will describe a number of ways to give an approximation of conditional probabilities for all possible feature vectors.

3.2.1 Chain rule We can express probability $P(x|c), c \in \{L, S\}$ in terms of conditional probabilities using the chain rule:

$$P(x|c) = P(x_1|c)P(x_2|x_1, c)P(x_3|x_1, x_2, c)\dots P(x_n|x_1, x_2, \dots, x_{n-1}, c) = \prod_{i=1}^{n} P(x_i|\bigcap_{j=1}^{i-1} x_j, c)$$

Let us first consider any of the probabilities $P(x_i|x_1, x_2, \dots, x_{i-1}, c)$ for given values of $x_1, \dots, x_{i-1}$ and c. The constraint of c limits the set of messages to either only legitimate messages or only spam. The constraints on the fist $i-1$ features require the first $i-1$ components of the feature vector to be equal to given values $(x_1, x_2, \dots, x_{n-1})$.

We will now extend the function M from the previous section to be able to compute the number of features with constraints on some indexes. Let $M(c, x_1, x_2, \dots, x_l)$, $1 \leq l \leq n$ denote the power of subset of c with the first l features equal to $x_1, x_2, \dots, x_l$. Then

$$P(x_i|x_1, x_2, \dots, x_{i-1}, c) = \frac{M(c, x_1, x_2, \dots, x_i)}{M(c, x_1, x_2, \dots, x_{i-1})}$$

The value of $M(c, x_1, \dots, x|i)$ is the number of messages in the training set of legitimate or spam messages (depending on c) feature vectors of which start with $x_1, x_2, \dots, x_i$. One way to find this value is to go through all training messages of category c and count the number of those with feature vectors that match $x_1, \dots, x_n$. This would mean $O(nk)$ time complexity for each probability in the chain, and overall complexity for $P(x|c)$ would be $O(\sum_{i=1}^{n} nk) = O(n^2k)$, which is, of course, not feasible in practice.

However, we can reuse the trie that we already are using for calculating $M(c, x_1, \dots, x_n)$, only in this case we need to retrieve the number of messages feature vector of which matches the first i given coordinates. This can be done

in $O(n)$ if in each node we store the sum of values of all terminal nodes in the subtree rooted at that node. This will give us exactly the number of messages feature vector of which satisfies the constraints on the first i coordinates.

In fact, as we follow the path in the trie, on each step we will calculate exactly one of the probabilities in the chain. Taking their product gives us the final probability with overall complexity of $O(n)$. Essentially, this approach is just a different computation of the same non-Bayesian classifier, but it can be modified more easily.

3.2.2 Approximation for Feature Vectors Outside The Training Set

The chain rule allows us to compute the exact value of $P(x|c)$, but it does not solve the problem of feature vectors that are not in the training set having zero probability. In the previous section we suggested that a priori probability (like in naive Bayesian classifier) can be used for the whole feature vector.

We can apply the same idea, but to each of the probabilities in the chain separately: if any of the probabilities $P(x_i|x_1, \ldots, x_{i-1}, c)$ are equal to zero, they are replaced with a priori $P(x_i|c)$.

As before, probabilities of all feature vectors that are present in the training set will not change after such modification. To prove this, let us consider each of the probabilities in the chain rule for some message $m \in c$ for fixed c. For $1 \leq i \leq n$:

$$P(x_i|x_1, \ldots, x_{i-1}, c) = \frac{|m \in T_c : x_1^m = x_1, \ldots, x_i^m = x_i|}{|m \in T_c : x_1^m = x_1, \ldots, x_{i-1}^m = x_{i-1}|}$$

Note that $1 \leq \forall i \leq n$ neither nominator nor denominator are equal to 0 since the corresponding subsets of T_c contain at least the message m. Therefore, the chain rule Bayesian classifier compared to non-naive classifier only changes the probability of messages that would otherwise have zero probability.

One can notice that after the feature with zero conditional probability the chain rule classifier behaves identically to naive Bayesian classifier, i.e. for the rest of the features it only considers the a priori probability. It is still possible to compute the a posteriori probability while ignoring some features, therefore avoiding the situation with zero probability. However, such approach can be extremely expensive since for each component of the chain $O(2^n)$ potential subsets of conditions can be considered. Also, recall that while the trie allows to compute

any of the probabilities in the chain in $O(n)$ time, it is only the case for all features in order.

3.2.3 Chain Rule Bayesian Classification Algorithm Let us outline the final version of semi-naive Bayesian classifier based on chain rule computation.

Training process

1. Calculate $|L|$ and $|S|$. Time complexity: $O(k)$.
2. Calculate likelihood ratio $\lambda\frac{P(L)}{P(S)}$. Time complexity: $O(1)$.
3. For both L and S construct corresponding tries that allow to perform lookup of $M(c, x_1, x_2, \ldots, x_i), 1 \leq i \leq n$. Time complexity: $O(nk)$. Space complexity: $O(nk)$.

Time complexity: $O(nk)$. Space complexity: $O(nk)$.

Classification process

1. Determine feature vector x for message m. Time complexity: $O(|m|)$ where $|m|$ is the number of words in m.
2. Calculate $P(x|L) = \prod_{i=1}^{n} P(x_i|\bigcap_{j=1}^{i-1} x_j, L)$ and $P(x|S) = \prod_{i=1}^{n} P(x_i|\bigcap_{j=1}^{i-1} x_j, S)$ using the corresponding paths in the tries. Time complexity: $O(n)$.
3. Calculate $\Lambda(x) = \frac{P(x|S)}{P(x|L)}$. Time complexity: $O(1)$.
4. Use Bayes's decision rule to determine class of the message. Time complexity: $O(1)$.

Time complexity: $O(n)$. Space complexity: $O(1)$.

Note that the asymptotic complexity of classification is the same as in case of naive Bayesian algorithm.

3.2.4 Limited Number of Conditional Probabilities Some scenarios might require additional constraints on time or space complexity of the training phase. The proposed algorithm does allow for trade-offs between the number of conditions and speed/space.

The first modification that can significantly improve training and classification speed of chain rule classifier is to only calculate a limited number of conditional probabilities, using naive assumption for the rest. It is similar to what the classifier is already doing for most messages, but with a strictly defined limit on the number of extra calculations.

Let d, $1 \leq d \leq n$ be the limit on features for which we will calculate conditional probabilities. For the first d features in the feature vector we will construct the trie, for the rest we will compute unconditional probabilities.

Training process

1. Calculate $|L|$ and $|S|$. Time complexity: $O(k)$.
2. Calculate likelihood ratio $\lambda \frac{P(L)}{P(S)}$. Time complexity: $O(1)$.
3. For all $w_i \in W$ calculate and store $P(x_i), d+1 \leq i \leq n$. Time complexity: $O(k(n-d))$, space complexity: $O(n-d)$.
4. For both L and S construct corresponding tries that allow to perform lookup of $M(c, x_1, x_2, \ldots, x_i), 1 \leq i \leq d$. Time complexity: $O(dk)$. Space complexity: $O(dk)$.

Time complexity: $O(nk)$. Space complexity: $O(dk)$.

Classification process

1. Determine feature vector x for message m. Time complexity: $O(|m|)$ where $|m|$ is the number of words in m.
2. Calculate $\prod_{i=d+1}^{n} P(x_i)$ using the stored values. Time complexity: $O(n-d)$.
3. Calculate $P(x|L) = \prod_{i=1}^{d} P(x_i | \bigcap_{j=1}^{i-1} x_j, L)$ and $P(x|S) = \prod_{i=1}^{d} P(x_i | \bigcap_{j=1}^{i-1} x_j, S)$ using the corresponding paths in the tries. Time complexity: $O(d)$.
4. Calculate $\Lambda(x) = \frac{P(x|S)}{P(x|L)}$. Time complexity: $O(1)$.
5. Use Bayes's decision rule to determine class of the message. Time complexity: $O(1)$.

Time complexity: $O(n)$. Space complexity: $O(1)$.

The above modification allows to limit the space requirements regardless of the number of features. This can be useful for applications with multiple customized classifiers each based on a separate training set.

3.2.5 Limited Number Of Conditions Another possible modification is to limit the number of conditions for each probability in the chain with some constant d. The easiest example would be to exclude features $x_{d+1}, \ldots, x_n$ from conditions.

The algorithm would be almost identical to the previous case. Calculation of probabilities of the first d elements of the chain remains unchanged. Calculation of the rest is the same as in default chain rule classifier, however conditions after x_d are ignored. This results in the following conditional probability:

$$P(x|c) = \prod_{i=1}^{d} P(x_i|\bigcap_{j=1}^{i-1} x_j, c) \prod_{i=d+1}^{n} P(x_i|\bigcap_{j=1}^{d} x_j, c)$$

The core algorithm remains the same, the only difference is that the depth of the search trie will be limited by d.

Training process

1. Calculate $|L|$ and $|S|$. Time complexity: $O(k)$.
2. Calculate likelihood ratio $\lambda\frac{P(L)}{P(S)}$. Time complexity: $O(1)$.
3. For both L and S construct corresponding tries that allow to perform lookup of $M(c, x_1, x_2, \ldots, x_i), 1 \leq i \leq d$. Time complexity: $O(nk)$. Space complexity: $O(dk)$.

Time complexity: $O(nk)$. Space complexity: $O(dk)$.

Classification process

1. Determine feature vector x for message m. Time complexity: $O(|m|)$ where $|m|$ is the number of words in m.
2. Calculate $P(x|L) = \prod_{i=1}^{n} P(x_i|\bigcap_{j=1}^{\min(i-1,d)} x_j, L)$ and $P(x|S) = \prod_{i=1}^{n} P(x_i|\bigcap_{j=1}^{\min(i-1,d)} x_j, S$ using the corresponding paths in the tries. Time complexity: $O(n)$.
3. Calculate $\Lambda(x) = \frac{P(x|S)}{P(x|L)}$. Time complexity: $O(1)$.

4. Use Bayes's decision rule to determine class of the message. Time complexity: $O(1)$.

Time complexity: $O(n)$. Space complexity: $O(1)$.

One potential modification in this direction would be to split the features (words in our case) into groups with high correlation within each group but weak correlation between groups, and then compute conditional probabilities between features of the same group.

Chapter 4

Feature Selection

4.1 Mutual Information Based Feature Selection

Words, formulas.

Chapter 5

Performance Comparison in Practice

5.1 Classification Quality

Words, tables, graphs, pictures, code.

Conclusion

Words.

References

[1] Konstantin Tretyakov. Machine Learning Techniques in Spam Filtering. Institute of Computer Science, University of Tartu. Data Mining Problem-oriented Seminar, MTAT.03.177, May 2004, pp. 60-79.

[2] V. Kecman. Learning and Soft Computing. 2001, The MIT Press.

[3] S. Haykin. Neural Networks: A Comprehensive Foundation. 1998, Prentice Hall.

[4] N. Cristianini, J. Shawe-Taylor. An Introduction to Support Vector Machines and other kernel-based learning methods. 2003, Cambridge University Press. http://www.support-vector.net

[5] Xavier Carreras, Lluis Marquez. Boosting Trees for AntiSpam Email Filtering. TALP Research Center, LSI Department, Universitat Politecnica de Catalunya.