Learning to Detect Spam: Naive-Euclidean Approach

Tony Y.T. Chan, Jie Ji, and Qiangfu Zhao

The University of Akureyri, Akureyri, 600 Iceland
The University of Aizu, Aizuwakamatsu, 965-8580, Japan
tonychanyt@gmail.com
{d8102102,qf-zhao}@u-aizu.ac.jp

Abstract. A method is proposed for learning to classify spam and non-spam emails. It combines the strategy of the Best Stepwise Feature Selection with a classifier of Euclidean nearest-neighbor. Each text email is first transformed into a vector of D-dimensional Euclidean space. Emails were divided into training and test sets in the manner of 10-fold cross-validation. Three experiments were performed, and their elapsed CPU times and accuracies reported. The proposed spam detection learner was found to be extremely fast in recognition and with good error rates. It could be used as a baseline learning agent, in terms of CPU time and accuracy, against which other learning agents can be measured.

Key words: Spam email detection, machine learning, feature selection, Euclidean vector space, 10-fold cross-validation, nearest-neighbor classifiers

1 Problem Introduction

The spam email problem is well-known, and personally experienced by anyone who uses email. National laws have been enacted to fight the problem. One difficulty is how to legally define spam. One personfs spam is another person's gem. Another difficulty is enforcement. Spammers simply move to other countries where the government and the laws are weak in this area.

Instead of depending on laws to get rid of spam emails, there are economical approaches to fight the problem. An email service provider can require the sender to do something, such as manually recognizing some fuzzy letters and digits, or pay something, such as e-pennies, before his emails will be delivered. Kuipers et. al. [8] proposed an anti-spam email protocol, called Zmail, based on the idea of the zero-sum game. Here, the players are the email service providers who keep account of the transactions. "Zmail requires the sender of an email to pay a small amount of money directly to the receiver of the email. The money earned by a user can be used to send email or can be exchanged for real money." The protocol attempts to force Zmail senders to pay and sendees to receive. Normal Zmail users receive as much as they send, so the internet Zmail load is balance. This hopefully will reduce spam.

Instead of depending on laws or email protocol to stop spam, we can also fight spam at the users' level. Email clients at the users' end commonly provide a filter subsystem to sort spam emails into a spam mailbox while the normal emails are placed into the user's inbox. "Filters of this type have so far been based mostly on manually constructed keyword patterns." [1] The user must be careful when he sets the keywords and the logic to filter the spam for his inbox. One mistake in the keyword or the logic, and his legitimate emails may end up in his spam box. In some instances, this keyword filtering approach might be asking too much analytical thinking from some users. Kuipers et. al. [8] mentioned that "the most important resource consumed by email is not the transmission process but the end user's attention." We want to minimize the user's involvement in analyzing and detecting spam emails to make the system user-friendly.

In this paper, the spam problem is treated as a classification problem, i.e., a pattern recognition problem. The user needs only to decide whether an email is spam or not. An intelligent agent will learn from his decisions to sort out whether a future email is spam or not. In [6], a simple metric model was proposed to learn to distinguish between triangles and rectangles from their pixel representation. An improved model was applied to the problem of classifying chromosomes [3], then generalized for the triple parity problem [5], and specialized for classifying irises, mushrooms, abalones, soybeans, etc. [4]. In this paper, the specialized Naive Euclidean model is tailored for the email spam problem.

For a supervised classification problem, the input is a set of examples accompanied by their correct class labels. The problem setting involves a pattern language describing a set P of (structural) objects. The object set is divided into exactly c mutually-exclusive subsets, $Q_1, Q_2, \ldots, Q_c \subset P$. Each of these subsets is assumed to be a class of objects. The teacher supplies the learning agent with c finite groups of objects to represent these classes. The union of these groups is called the example set. It can be used for training and validation testing to estimate the true error rate of a classifier. Given these groups, the intelligent agent then autonomously devises a specific classifier to distinguish the groups in the hope that when an unknown object $p \in P$ is presented to the classifier, it will be able to classify it as belonging to $Q_1, Q_2, \ldots,$ or Q_c. See Section 3 for an example of a supervised classification problem.

Suppose now we restrict the pattern language to Euclidean vectors. All objects are characterized by a sequence of D numbers or attributes. Some attributes are useful in classification; some are corrupt, noisy or redundant. During the training stage, the mean vectors for the training groups are calculated, and training objects are classified according to their nearest mean. The proposed method selects the features or subspace that maximizes the correct classification rate on the training objects. At the end of the training stage, the group mean vectors in the selected subspace are stored.

In the testing stage, the test objects are simply assigned to the class of the nearest mean vector. Other methods of assignment can be considered, particularly in cases where there are many clusters in the same class of objects. For efficiency reasons, in this paper, we shall use the simple 1-nearest-neighbor rule to the mean vectors. The proposed learner here is extremely quick to recognize and at the same time naive, partly because it does not consider multiple clusters within the same class distribution.

In this paper, the email spam problem is treated as a supervised pattern classification problem. The user labels each email as spam or non-spam. Then the Naive Euclidean model proceeds to set up a routine to detect future spam mails in the post-testing performance stage.

2 The Learner Model

In this section, we concentrate on the concepts and describe the mathematical model upon which the Fast Naive-Euclidean Learner is based.

Table 1. Criterion Function Z

$Z(\breve{Q}_1, \breve{Q}_2, \ldots, \breve{Q}_c)$
1. For each $\breve{Q}_i$, calculate the means vector m_i
2. Let $\breve{Q} = \bigcup_i \breve{Q}_i$.
3. For each $v \in \breve{Q}$,classify v according to its nearest mean vector.
4. $r \leftarrow$total number of example vectors that are correctly classified.
5. $n \leftarrow \lvert\breve{Q}\rvert$.
6. $Z \leftarrow \frac{r}{n}$.

The criterion function is called upon every time the learning agent wants to know how good a choice of attributes is. It is also called the attribute evaluator. Consider c sets of vectors in the D-dimensional Euclidean vector space. The label of each of the D axes is called an attribute. Each set represents a class of objects. Given $\breve{Q}_i, i = 1, 2, \ldots .c$, the set of example vectors for class i in a d-dimensional subspace, we can calculate the classification quality of the subspace according to the criterion function Z. Step 1 calculates the training mean vectors for all the example groups. Step 2 collects all the training vectors from all the groups. Step 3 classifies the examples using the nearest-neighbor method to the mean vectors. The last step returns the apparent accuracy of this particular classifier in the d-subspace. The whole algorithm Z takes $O(ncd)$ time where n is the size of the example set, c is the number of classes, and $d \leq D$ is the dimension of the subspace [4].

Function Z evaluates how effective the given subspace is in classifying the training vectors using the group means. The objective then is to maximize Z subject to the subspace variations. The search procedure plus the criterion procedure together requires $O(ncD^3)$ time.

Finding the best subset of attributes to optimize classification accuracy is an exponential problem in the number of attributes. Heuristic search is usually used for this process. One of the simplest is the greedy hill-climbing search. The algorithm is presented as follows:

Step 1. Begin with an empty set of selected attributes (S).

Step 2. Calculate the accuracy for each of the attributes in the space. Add to S the attribute that has the best accuracy.

Step 3. Calculate the accuracy for each of the remaining attributes in combination with S. If all the accuracies are worse than the best at Step 1, stop. Else add to S the attribute that improves the best accuracy at Step 1.

Step i. Calculate the accuracy for each of the remaining attributes not already in S in combination with the ones in S. Add to S the attribute that improves the accuracy. If none exists, stop.

3 Application: Spam Detection

The data of the spam email problem in this paper is downloaded from the UCI Machine Learning Repository. There are a total of 4601 emails in the database, i.e., the training set is of size 4601, 1813 of which are labeled as spam, the rest as non-spam. The zero-rule classification method would yield an accuracy of 60.6% based on the simple prevalence rate of the prevalence training examples. That is the minimum number for any classifier to beat.

The original emails are stored as text files of different lengths. To eliminate the problems associated with learning to classify objects of different lengths, each email is transformed into a vector of 57+1 dimensions. The last dimension represents the class label: 1 for spam and 0 for non-spam.

A set of 48 words were chosen from among the contents of this particular training set. These words were deemed to be relevant for distinguishing between spam and non-spam emails. They are as follows: make, address, all, 3d, our, over, remove, internet, order, mail, receive, will, people, report, addresses, free, business, email, you, credit, your, font, 000, money, hp, hpl, george, 650, lab, labs, 857, data, 415, 85, technology, 1999, parts, pm, direct, cs, meeting, original, project, re, edu, table, and conference.

Given an email text and a particular WORD, we calculate its frequency, i.e., the percentage of words in the e-mail that match WORD: word_freq_WORD = $100 \times r/t$, where r is number of times the WORD appears in the email and t is the total number of words in e-mail. For example, the word 'make' never occurs in email 1, so word_freq_make = 0, while 0.64% of the words in email 1 are the word_'address', so word_freq_address of email 1 = 0.64%. There are 48 such frequency variables. Each corresponds to a dimension (an axis) in the Euclidean training space.

Six characters are also chosen as special. They are ;, (, [, !, $, and #. Frequency variables are also created for these. Now training space is 48+6=54 dimensional.

Given an email text, three more variables are created, one for the average length of uninterrupted sequences of capital letters, one for the longest uninterrupted sequence length of capital letters, and one for the total number of capital letters in the email. Altogether, there are 57 attributes (variables) to describe an email, plus 1 attribute for the class label. This is how text emails are transformed into 58-dimensional vectors. For the spam email problem, 4601 emails were transformed and stored in the UCI Repository of benchmark problems.

In order to obtain an averaged unbiased accuracy estimate, we conducted 100 runs. For each run, data are completely randomized, then the database is divided into a training set and a separate test set. 90% of the example objects are used for training, while the remaining are used for testing. Every group of example objects is divided into 10 partitions. We apply the 10-fold cross-validation strategy on the Naive Euclidean model to estimate its true accuracy in each run.

The program was written in Matlab. The PC we used for experiment has a P4 2.66GHz dual core CPU, 4.00GB memory and uses windows XP x64 operation system. Since the CPU occupancy factor is 50% when the program was running, we can simply consider the CPU's frequency as 2.66GHz.

At the end of the feature selection process, Table 1 shows how many times a feature was selected. They are:

Table 2. Experiment 1: Feature Subset

	Feature	Ns	Feature	Ns	Feature	Ns	Feature	Ns	Feature	Ns	Feature	Ns
1	char_freq_$	1000	char_freq_[	153	addresses	54	direct	1	report	0	hpl	0
2	receive	997	remove	130	cs	45	make	0	free	0	george	0
3	data	751	char_freq_;	107	labs	35	address	0	business	0	re	0
4	telnet	697	money	93	pm	31	all	0	email	0	char_freq_!	0
5	conference	592	”1999”	90	”650”	18	our	0	you	0	capital_run_	
6	original	378	table	83	lab	14	over	0	credit	0	length_avg	0
7	parts	368	”857”	83	”85”	12	internet	0	your	0	capital_run_	
8	technology	282	meeting	69	3d	8	order	0	font	0	longest	0
9	char_freq_(	250	char_freq_#	58	”415”	6	mail	0	”000”	0	capital_run_	
10	project	249	edu	57	will	6	people	0	hp	0	total	0

The average number of selected feature is 6.7. Recognition is then performed on these subspaces using the two mean vectors as the representatives, one for the spam mail, and one for the non-spam mail. The 10-fold cross-validation accuracy averages 82.31%, which is significantly better than the 60.6% prevalence rate. The average elapsed CPU time for a 10-folds run training is 90 seconds. As we know, Matlab is an interpreted language, if we use C language, the elapsed CPU time will be reduced to 14.6 seconds. At recognition phase, it basically took no time, since given an unknown email, the agent only needs to calculate 6 to 8-dimensional Euclidean distances to classify it. If it is closer to mean vector 1, classify the unknown to the mean vector 1's class. If it is closer to mean vector 2, then classify the unknown to mean vector 2's class.

A second experiment is performed in the hope of obtaining a better accuracy. To find a better feature subset, backtracking is used and set to 3 levels. This allows the agent to throw away 3 attributes from the current best subset and

Table 3. Experiment 2: Feature Subset with Backtracking

	Feature	Ns	Feature	Ns	Feature	Ns	Feature	Ns	Feature	Ns	Feature	Ns
1	char_freq_$	1000	remove	241	table	123	lab	31	make	0	you	0
2	receive	874	technology	239	char_freq_!	114	”85”	31	address	0	your	0
3	conference	593	”1999”	239	”000”	111	”415”	30	all	0	hp	0
4	data	584	char_freq_[	201	hpl	107	”650”	21	our	0	george	0
5	telnet	564	money	191	cs	98	internet	14	mail	0	capital_run_	
6	parts	396	meeting	183	3d	94	order	10	people	0	length_avg	0
7	original	335	edu	154	labs	77	will	8	report	0	capital_run_	
8	char_freq_(	301	pm	142	char_freq_#	71	over	3	free	0	longest	0
9	project	272	”857”	139	credit	47	re	3	business	0	capital_run_	
10	char_freq_;	258	addresses	135	font	37	direct	1	email	0	total	0

replace them with some other hopefully better attributes. Experiments also conducted 100 times. In this case, average elapsed CPU time rises to 140 seconds. It find average 8.07 features during the training phase. The average recognition accuracy rises to 83.25%.

In the 3rd experiment, the backtracking level is set to 8. This time, the average accuracy fell down to 82.52%, average 7.8 features are found and the average elapsed CPU time is 125 seconds. This is because the backtracking procedure will be executed only when the selected features are over 8, otherwise the algorithm will give output directly, so the CPU time is smaller. The subsets which are over than 8 dimensionalities will have only one or two dimensions after backtracking, which will have a bad influence on the accuracy.

For spam detection, accuracy and time complexities are not the only criteria to judge the spam detection agent. If the agent incorrectly places an important email to the spam mailbox, the consequence is a lot worse than if the agent mistakenly places a spam into the user's inbox. Table shows some important considerations, such as False positive rate, false negative rate, precision rate and F-measure.

Table 4. Considerations of Three Experiments

Best stepwise	Accuarcy	Selected Fea	FPR	TNR	FNR	F-measure
Backtracking lever=0	0.8231	6.7	0.074	0.926	0.335	0.747
Backtracking lever=3	0.8325	8.1	0.074	0.926	0.311	0.762
Backtracking lever=8	0.8252	7.8	0.074	0.926	0.329	0.751

4 Discussion

The Naive Euclidean training procedure runs extremely fast in polynomial time. From the experiments, we have obtained reasonable accuracy rates in good running time for the training and testing phases. During the online performance phase, to recognize a spam email takes almost no time. The agent simply calculate the Euclidean distance of two distances from the unknown email to the two mean vectors.

Future works can be pursued along the following lines.

Use a non-Euclidean distance formula, such as weighted Euclidean. Certain attributes might be more weighty than other attributes in recognizing the spam emails. These can receive a higher weight.

Compare Naive-Euclidean with other methods of pattern recognition such as neural network. One thing for sure, neural network will required substantially more training and testing time compared with our low-polynomial Euclidean learning agent. Neural network also requires a bit more CPU time to classify whether an email is spam or not at the live performance stage. It is likely also that a well-trained neural network can produce better accuracy.

One way to improve Naive Euclideanfs accuracy is to find clusters within classes. Use k-nearest-neighbor rule for classification instead of the c class mean vectors, $k > c$.

Try other approximation methods for the optimization control, such as the best stepwise feature elimination algorithm [10], relevance-based algorithm [9], and genetic algorithm [11]. Perform experimentations on these method. Perhaps there is one that is particularly suitable for detecting spam emails.

Extend the model to unsupervised learning. To be successful, this is a tall order for the spam detection problem. Whether an email is spam or not often depends on the user. Some think this is spam; others think it is a gem. Unsupervised means taking away the userfs decision during the training phase and just let the learning agent decides what is spam or not. Partial success in this, however, is possible and welcome. Perhaps, partially supervised approach is the way to go.

Engineer a portable user-friendly software tool on this system using e.g., the Python language.

The benchmark dataset from UCI Repository contains 5 vectors. How does one transform a text email into a vector representation? As described in Section 3, one of the important concepts is to design a special set of keywords, or keystrings. These keywords are selected from the training set text emails. They are supposed to be chosen because of their ability to distinguish spam and non-spam emails.

The general metric model proposed in [2] can be used to find this keyword set automatically. Given a training set of text emails, the general metric mode can be used to find the keywords that possess discriminatory power more than the usual words. This keyword set can be personalized to a particular user, or group of users using the general metric model.

These are some ideas that can improve the Naive Euclidean agent, so that it would not be so naive.

5 Conclusion

The Naive Euclidean without backtracking performed better than the Zero Rule in terms of accuracy, 82.31% vs. 60.6%.

The Naive Euclidean training procedure runs extremely fast for the spam detection problem and yet the correct classification rate is reasonable. It could serve as a baseline in terms of CPU time and accuracy against which other learning methods can be compared.

References

1. Ion Androutsopoulos, John Koutsias, Konstantinos V. Cbandrinos, and Constantine D. Spyropoulos. An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. In Proceedings of SIGIR-2000, pages 160.167, 2000.
2. Tony Y. T. Chan. Inductive pattern learning. IEEE Transactions on Systems, Man, and Cybernetics. Part A: Systems and Humans, 29(6):667.674, Nov. 1999.
3. Tony Y. T. Chan. Unsupervised classification of noisy chromosomes. Bioinformatics, 17(5):438. 444, May 2001.
4. Tony Y. T. Chan. Fast Naive-Euclidean learner. In Proceedings of the 2nd IEEE International Conference on Systems, Man, and Cybernetics, page no.: Session MP1F6. IEEE Computer Society, 2002.
5. Tony Y. T. Chan. Unifying metric approach to the triple parity. Artificial Intelligence, 141(1.2):123. 135, Oct. 2002.
6. Tony Y. T. Chan and Lev Goldfarb. Primitive pattern learning. Pattern Recognition, 25(8):883.889, 1992.
7. Lev Goldfarb. An evolving model for pattern learning. Pattern Recognition, 23(6):595.616, 1990.
8. Benjamin J. Kuipers, Alex X. Liu, Aashin Gautam, and Mohamed G. Gouda. Zmail: Zero-sum free market control of spam. In Distributed Computing Systems Workshops, pages 20.26, New York, 2005. IEEE.
9. Hui Wang, David Bell, and Fionn Murtagh. Axiomatic approach feature subset selection based on relevance. IEEE Transactions on PAMI, 21(3):271. 276, March 1999.
10. Sholom M.Weiss and Casimir A. Kulikowski. Computer Systems That Learn. Morgan Kaufmann, San Francisco, 1991.
11. Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. IEEE Intelligent Systems, 13(2):44.49, March/April 1998.