PROJECT REPORT FOR
"*Developing a Software for Turkish Spam Mail Filtering*"

by
Huseyin OKTAY & Sedat YILDIZ

Submitted to the Department of Computer Engineering
in partial fulfilment of the requirements
for the degree of
Bachelor of Science
in
Computer Engineering

Boğaziçi University
October 2007

TABLE OF CONTENTS

1. Introduction 4
1.1. Spam Mails 4
1.2. Methods for anti-Spam Filtering 4
1.2.a. Artificial Neural Networks 4
1.2.b. Bayesian Networks 5
1.2.c. N-Gram Method 5
2. File System used for Spam Filter 6
2.1 Storage of An E-Mail 6
2.2 Storage of All Words 7
2.3 Storage of Feature Vector 7
2.4 Storage of N-Gram Probabilities 8
2.5 How our file system makes life easier for the program? 9
3. Training Phase of the Methods 9
3.1 Training in ANN Method 9
3.2 Training in Bayesian Network 11
3.3 Training in N-Gram Method 11
4. Tracing the Methods with the New File System 12
4.1 Tracing ANN Method 12
4.1.a Binary Model 13
4.1.b Probabilistic Model 14
4.2 Tracing Bayesian Network Model 15
4.2.a Binary Model 16
4.2.b Probabilistic Model 17
4.2.c Advanced Probabilistic Model 19
4.3 Tracing N-Gram Method 21
4.3.a Class General Perception (CGP) Model 22
4.3.b E-Mail Specific Perception (ESP) Model 25
4.3.c Combined Perception Refinement (CPR) Model 26
5. Integration of methods into Mail Clients 27
6. Integration of C code into C# 37
7. GUI in C# 38
8. Why re-implementing methods? 41

ABSTRACT

Project Name : Developing a Software for Turkish Spam Mail Filtering

Term : Spring 2007

Keywords : Artificial Intelligence, Spam filtering for Turkish, ANN, Bayesian, N-Gram

Summary:

With the improvements on internet, e-mails have become one of the most effective communication tools. E-mail is an easy and cheap way of communication in which one can communicate with many people simultaneously. Besides these, it is obvious that everybody receives some messages without their desires. Spam mail is the general name used to denote these types of e-mail. Hence in this project the aim is to develop software which would enable us to filter the spam e-mails from the normal e-mails.

1. Introduction

1.1. *Spam Mails*

Communicating with e-mail has become one of the inevitable ways of transferring information. A lot of people are using e-mail technology to reach some other people because using e-mails is easy and cheap. However there are e-mails called as spam mails which are sent out to thousands of recipients especially for advertisement. These types of e-mails waste most of the people's time.

Many methods have been proposed regarding this problem. There are both static and dynamic approaches for spam filtering. Since the nature of the problem is dynamic, dynamic methods are more suitable than the static ones. In this project our aim is to develop software for a spam filter program for Turkish e-mails. Our methods are based on Artificial Neural Network (ANNI), Bayesian Networks and n-gram . The methods were implemented in a prior research project and we are now making it one step further by adding an outlook interface on top of these methods as well as making some little changes in the structure of the methods.

1.2. *Methods for anti-Spam Filtering*

1.2.a. Artificial Neural Networks

Artificial Neural Networks (ANN) is a kind of machine learning algorithm by which one can fit a wide range of domains and can model any high degree exponential problem using multi-layer perceptrons. We have 2 different kinds of ANN approaches, single layer perceptron (SLP) and multi-layer perceptron (MLP). This takes the feature vector as an input and gives out estimation depending on the weights on the inputs. According to this estimation, program decides whether the given e-mail is spam or not. By the way it is obvious that we should first train the ANN with a training set to optimize the coefficients on the inputs.

We have 2 different approaches in ANN, first is binary model and second is probabilistic model. In binary model, we are concerned with only the word is in e-mail or not. Our

feature vector values are 0 or 1, stating the word does not occur in the e-mail and the word occurs in the e-mail respectively.

1.2.b. Bayesian Networks

We have also methods implemented by using the Bayesian Networks. Here we have the assumption that the words in the feature vector are independent from each other. We are calculating the probability of being normal for a given e-mail X with it is feature vector and also of being spam. And we conclude that an e-mail is spam if the probability of being spam is higher and vice versa.

We have 3 different variations of Bayesian Networks: binary model, probabilistic model and advanced probabilistic model. In binary model we are only concerned with whether words in the feature vector are in the e-mail or not. In probabilistic model we are also concerned with the number occurrence of the words in feature vector in the e-mail. Finally, in advanced probabilistic approach we are both interested in the number of occurrence of the words in feature vector as well as the total number of words in e-mail.

1.2.c. N-Gram Method

N-gram method is a way for using in spam filtering. Method makes the drastic assumption that only the previous n-1 words have an effect on the probability of the next word. While this is clearly false, as a simplifying assumption it often does a serviceable job. In this method n often be 3. Also we use first n-words heuristics in n-gram method. That heuristic comes from human behaviour of people. When we read an e-mail we look a few words and understand if it is spam or not. By using first n-words approach we will lok first n words and decide if it is spam or not.

There are 3 models in n-gram method. These are class general perception model (CGP), e-mail specific perception model (ESP) and combined perception refinement (CPR).

2. File System used for Spam Filter

Performance is an important criterion for spam filtering, hence we should design our file system so that the program will function efficiently. In this section we are going to give the details of our file system design and after that we are going to trace each method to see that our file system really makes the program work efficiently.

2.1. Storages of E-mail

We are going to store a file for each and every e-mail. In that we are going to store every different words occurring in that e-mail and the frequency of occurrence meaning that the number of times that word occurs in that e-mail. And on top of these we are going to store the total number of words in that e-mail.

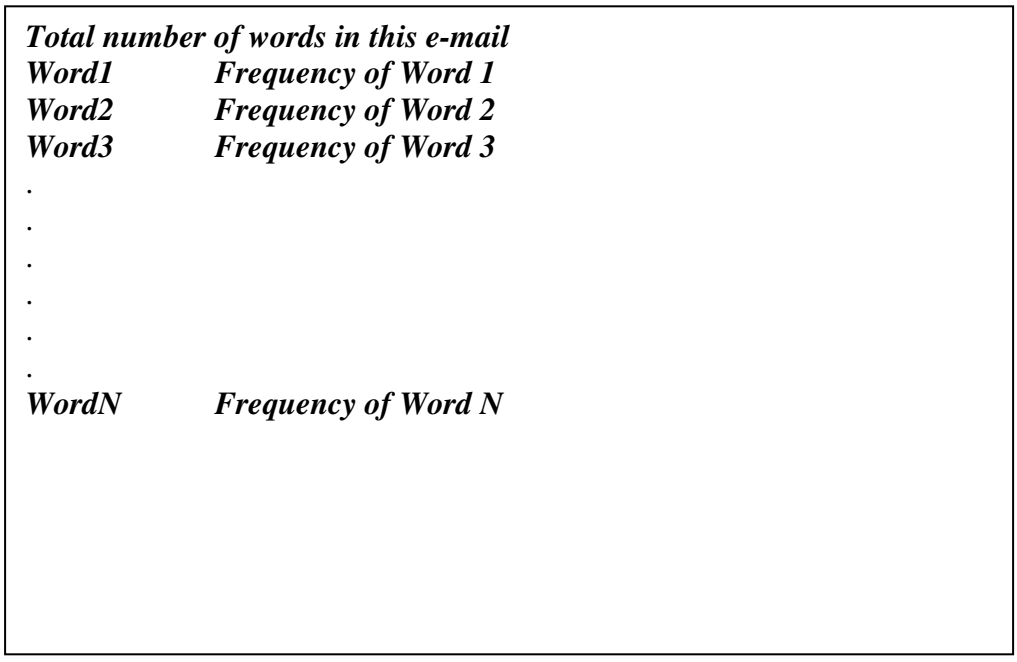


Figure 1. Structure of an example e-mail file

Here words are sorted in lexically ascending order.

2.2. Storages of All Words

We are going to store all the words in our training set in a big file. In that file in the first entry we are going to store the total number of spam mails and total number of normal mails in our training set. Then each entry will store a word, then number of spam mails that this word occurs and finally the number of normal mails that this word occurs.

Total number spam mails Total number of normal mails
Word1 # of Spam mails Word1 occurs # of Normal Mails Word1 occurs
Word2 # of Spam mails Word2 occurs # of Normal Mails Word2 occurs
Word3 # of Spam mails Word3 occurs # of Normal Mails Word3 occurs
.
.
.
.
.
.
WordN # of Spam mails WordN occurs # of Normal Mails WordN occurs

Figure 2. Structure of a file that stores all the words

Here words are sorted in lexically ascending order.

2.3. Storage of Feature Vector

We are going to simply store the feature vector words in a file after the training phase. The format of that file will be as follows:

Word 1	***# of Spam mails Word1 occurs***	***# of Normal Mails Word1 occurs***
Word 2	***# of Spam mails Word2 occurs***	***# of Normal Mails Word2 occurs***
Word 3	***# of Spam mails Word3 occurs***	***# of Normal Mails Word3 occurs***
.		
.		
.		
.		
.		
Word N	***# of Spam mails WordN occurs***	***# of Normal Mails WordN occurs***

Figure 3. Format of the file for Feature Vector

Here words are sorted in lexically ascending order.

2.4. Storage of N-Gram Probabilities

We are going to store all n-gram probabilities for spam and normal mail classes. It will be a big file but we have to know all of the n-gram probabilities. Figure 4 shows how we will store file.

#of words in spam class	***#of words in normal class.***	
Word 1	***# of occr. in spam class (u)***	***#of occr. in normal class***
Word 1 Word 2	***# of occr. in spam class (b)***	***#of occr. in normal class.***
.		
Word 1 Word N	***# of occr. in spam class (b)***	***#of occr. in normal class***
Word1 Word 2 Word 3	***# of occr. in spam class (t)***	***#of occr. in normal class.***
.		
Word1 Word2 Word N	***# of occr. in spam class (t)***	***#of occr. in normal class***
.		
.		
Word 2	***# of occr. in spam class (u)***	***#of occr. in normal class***
.		
.		
.		
Word N Word N Word N	***# of occr. in spam class (t)***	***#of occr. in normal class***

Figure 4. Format of the file for n-gram probabilities

2.5. How our file system makes life easier for the program?

Our file system brings a lot of advantages to our system. We can easily read the feature vector from the file. For binary model, we can easily check whether the word in feature vector occurs in the e-mail or not. For the probabilistic model we need the number of occurrence of the word in the e-mail. It is also easy to reach this information because we store this number in our e-mail file and read those values into the memory. Then we need the total number of words in the e-mail. This is also stored in the e-mail file. Also in our feature vector file we store number of spam e-mail that the corresponding word occurs and number of normal e-mails that the corresponding word occurs. We use those values in probabilistic Bayesian Method. We also store the total number of spam mails and total number of normal mails which are vital for advanced probabilistic model. Hence we can conclude that program has a higher performance with this file system

3. Training Phase of the Methods

3.1. Training in ANN method

Our spam filter program should be trained with the incoming e-mails to execute efficiently and effectively. In training face of the ANN method, first we calculate the words for the feature vector and we form the feature vector file. We use mutual index to determine the feature vector words. We select the words with the highest mutual index. This correspond words that have a high probability on existing in one type of e-mail but not in other. Than program optimize the coefficients of the inputs in the ANN network. We have a data set consisting of both spam and normal e-mails. By using this dataset program optimizes the coefficients in each input.

Example Scenario for Training ANN

$\Rightarrow$ Assume we have formed the file that stores the all words as in Figure 2.

$\Rightarrow$ For the all words in this file calculate the Mutual index value as:

$\Rightarrow$ $$MI(W) = \sum_{w\in\{0,1\},c\in\{spam,normal\}} P(W=w,C=c)\times\log\frac{P(W=w,C=c)}{P(W=w)\times P(C=c)}$$

- P (W = 1, C = normal) is Number of normal mails that the corresponding word occurs divided by the number of total normal mails. (All these entries are explicitly stored in the all words file.)
- P (W = 0, C = normal) is Number of normal mails that the corresponding word does not occur divided by the number of total normal mails. (All these entries are explicitly stored in the all words file.)
- P (W = 1, C = spam) is Number of spam mails that the corresponding word occurs divided by the number of total spam mails. (All these entries are explicitly stored in the all words file.)
- P (W = 0, C = spam) is Number of spam mails that the corresponding word does not occur divided by the number of total spam mails. (All these entries are explicitly stored in the all words file.)
- P (W = 1) is number of mails that the corresponding word occurs divided by the total number of mails.
- P (W = 0) is number of mails that the corresponding word does not occurs divided by the total number of mails.
- P (C = Spam) is number of spam mails divided by the total number of mails.
- P (C = Normal) is number of normal mails divided by the total number of mails.

In the file that we store all the words, we also store the total number of spam mails, total number of normal mails and for each word we store total number of normal mails that this word occurs and total number of spam mails that this word occurs. Hence with this knowledge, we can calculate the each and every entry that we need above.

⇒ Then select the n words (n = the size of the feature vector) to form the feature vector

⇒ Write the feature vector file

⇒ To optimize ANN values, for each email file, form the corresponding feature vector and train the ANN iteratively. At the end we would find the weights of the ANN module.

3.2. Training in Bayesian Method

In Bayesian Methods training phase just consists of determining the feature vector words and writing out these words into a file. The selection process of the feature vector words is same as mentioned in 3.1 section.

3.3. Training in N-Gram Method

In n-gram method training phase consists of calculating word sequences' unigram bigram and trigram probabilites. Program trains itself and when a new e-mail comes it calculates probabilies according to trained set.

Example Training in N-Gram Model

➔ Assume we have N mails in our training set.

➔ First open the first mail and get the first word.

➔ Open the n-gram occurance file (Section 2.4 Figure 4). Search for word in file.

➔If word is found ;

➔If mail is spam add 1 to the #of occurance in spam mails.

➔If mail is normal add 1 to the #of occurance in normal mails.

➔ If word is not found write the word into the file and write 1 to the # of occurance (spam or normal)

➔Continue this until N words are searched.

➔ When it finished start searching word sequences. Look at mail and take the word sequences which are length 2.

➔ Search n-gram occurance file.

➔Apply the same program flowing like above.

➔When finished start searching word sequences which are in length 3.

➔ Apply the same program flowing like above.

➔When finished training in N-Gram Method is finished also.

4. Tracing the Methods with the New File System

4.1. Tracing Artificial Neural Network Method

Below is the figure 5 in which there is a simple chart showing the flow of the program in ANN method. Our program first reads the incoming e-mail and feature vector from their files. After that it forms the feature vector values for this incoming e-mail. This feature vector values is the input for the ANN. According to the output of the ANN, program decides on whether the given e-mail is spam or not.

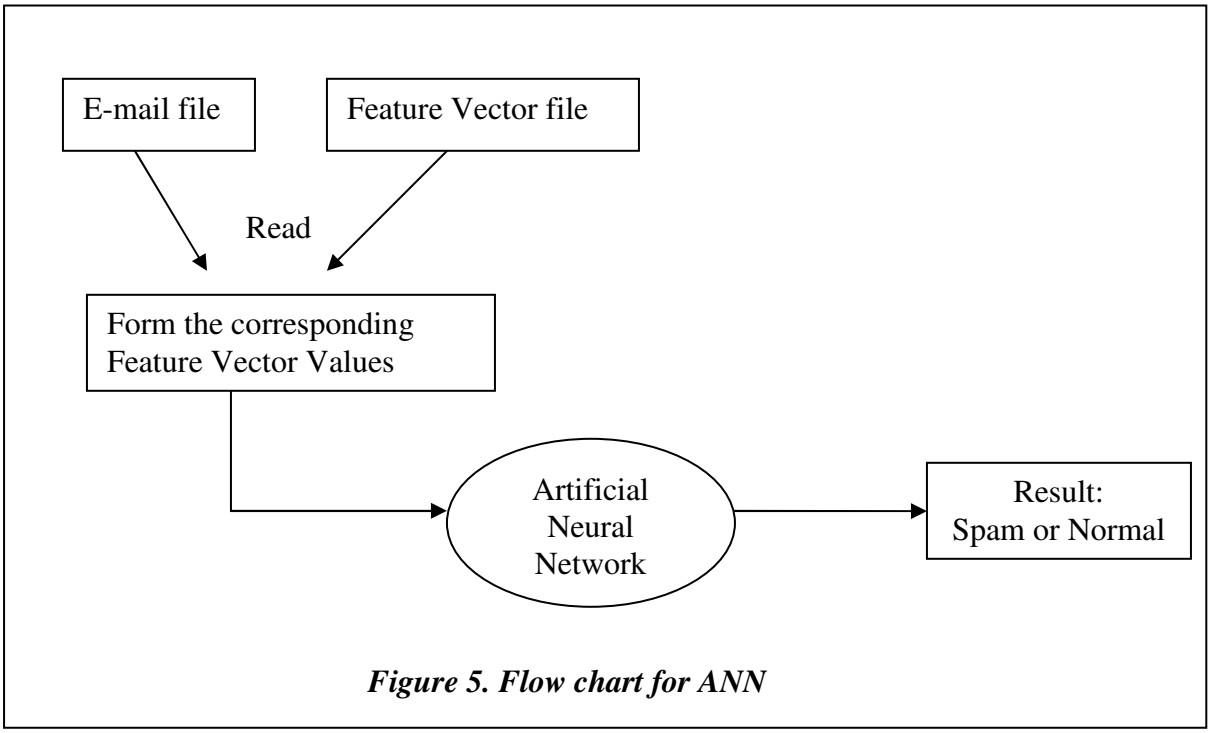


Figure 5. Flow chart for ANN

What is different between models is forming the corresponding feature vector Values for an incoming e-mail. Thus below we would mention how we form the feature vector in different models.

4.1.a. Binary Model

In binary model, figure 6, we form our feature vector values as follows: we take the each word from our feature vector and set the corresponding value to 1 if this word exists in the incoming e-mail and it is set to 0 if this word does not exist in the incoming e-mail.

```
Procedure binaryANN( )
1: Read the feature vector from the file.
2: Read the given e-mail into memory
3: For i = 0 to n do
4: If word_i is in e-mail then
5:         FeatureVectorValue[i] = 1
6: else
7:         FeatureVectorValue[i] = 0
8: end for
9: Then give this to Feature Vector to ANN
10: According to the output of the ANN, conclude result
```

Figure 6. binary ANN algorithm

Example Scenario

- ⇒ Assume we have formed the files for all words, emails and feature vector and we have an incoming email ***E1,*** with the words ***w1, w2, …… wN*** and their frequencies and ***f1, f2 …… fN.***
- ⇒ Read the feature vector file (FV) into memory.
- ⇒ Read the total number of words in this email from its file to the variable ***totalWords***.
- ⇒ Read the words and their frequencies in the incoming email file into memory.
- ⇒ Than for each word FV_i in feature vector file calculate the feature vector values as follows:
  - If the word FV_i occurs in email ***E1*** the value of the feature vector is1
  - Else the ith value of the feature vector value is 0

⇒ Than give this feature vector to the ANN and conclude the result according to the output of the ANN.

4.1.b. Probabilistic Model

In probabilistic model, figure 7, we are also interested in the frequency of the word in the incoming e-mail. We again take each word from the feature vector file, and we search for it in the e-mail file. If it exists, we get the frequency of that word in the incoming file (it is also stored in the e-mail file). We calculate the feature vector value for this incoming e-mail by dividing the frequency of that word to the total number of words in that incoming e-mail. If that word does not exist in the e-mail we again set the value to 0 for the corresponding feature vector value.

```
Procedure probabilisticAnn ( )
1:  Read the feature vector from the file.
2:  Read the given e-mail into memory.
3:  For i = 0 to n do
4:      If word_i is in e-mail then
5:          FeatureVectorValue[i] = #of occurrences of this word in e-mail / # of total words in e-mail
6:                  //  # of total words and the frequency of the words are read from the e-mail file
7:      else
8:          FeatureVectorValue[i] = 0
9:  end for
10: Then give this feature vector value to ANN
11: According to the output of the ANN, conclude result.
```

Figure 7. probabilisticANN algorithm

Example Scenario

⇒ Assume we have formed the files for all words, emails and feature vector and we have an incoming email ***E1,*** with the words ***w1, w2, …… wN*** and their frequencies and ***f1, f2 …… fN.***

⇒ Read the feature vector file (FV) into memory.

⇒ Read the total number of words in this email from its file to the variable ***totalWords***.

⇒ Read the words and their frequencies in the incoming email file into memory.

⇒ Than for each word FV_i in feature vector file calculate the feature vector values as follows:

- If the word FV_i occurs in email ***E1*** the value of the feature vector is f_i divided by totalWords.
- Else the ith value of the feature vector value is 0

⇒ Than give this feature vector to the ANN and conclude the result according to the output of the ANN.

4.2. Tracing Bayesian Network Model

Below is a simple chart, figure 8, showing the flow of the program in Bayesian method. Our program first reads the incoming e-mail and feature vector words from the files. Then by using the Bayesian formula we calculate the probability of the given e-mail of being normal and spam. We conclude that the e-mail is spam if probability of being spam is greater than normal and vice versa.

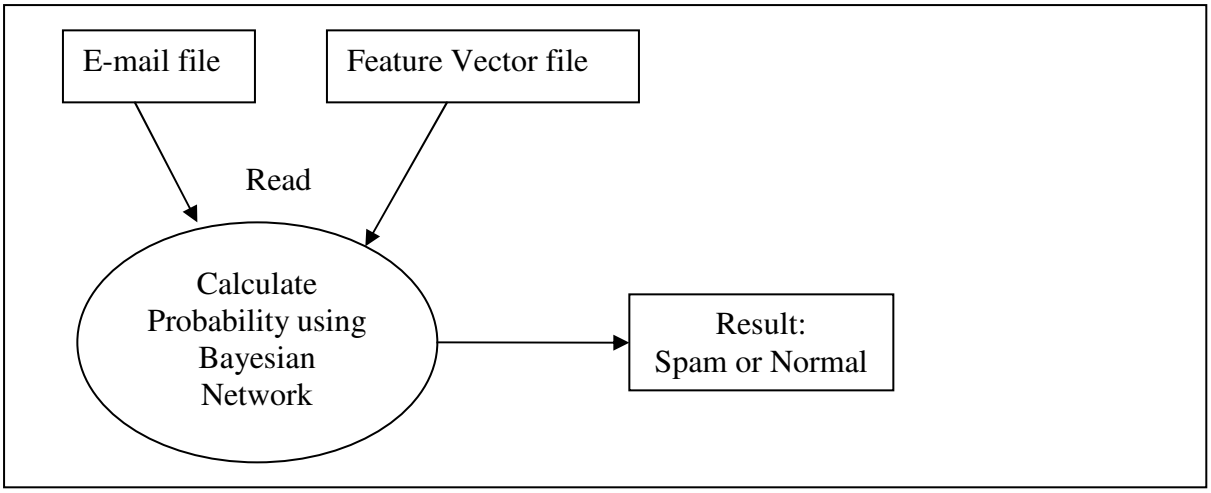


Figure 8. Bayesian Network Algorithm

What is different between methods is the values used while calculating the probability.

4.2.a. Binary Model

Here we are interested in the words of feature vector. If the word is in e-mail, we multiply its prior probability with a constant c and then add it to the probability value. This multiplication factor gives the sense of occurrence of an input word usually indicates a stronger idea for classifying e-mails than non-occurrence of that word. If the word does not exist in the e-mail then we subtract its prior probability from the total sum.

$$P(C_i/X) = \sum_{J=1}^{N} \begin{cases} cP_{ij}, & \text{If jth word of feature vector occurs in e-mail} \\ -P_{ij}, & \text{otherwise} \end{cases}$$

$$P_{ij} = \frac{\text{the number of e-mails containing the word j in class Ci}}{\text{total number of e-mail in class } C_i}$$

$c =$ is the coefficient level which indicates the level of this strength

```
procedure BinaryBayesian
1:  Read the feature Vector from the file
2:  Read the given e-mail into memory
3:  For i = 0 to classnum do
4:      For i = 0 to n do
5:          If word_i is in e-mail then
6:              Probability(C_i/X) += C*P_ij
7:          else
8:              Probability(C_i/X) -= P_ij
9:      end for
10: end for
11: if Probability(C_normal/X) > Probability(C_spam/X)
12:     E-mail is normal
13: else
14:     E-mail is spam
```

Figure 9. BinaryBayesian algorithm

Example Scenario

⇒ Assume we have formed the files for all words, emails and feature vector and we have an incoming email ***E1,*** with ***N*** words ***w1, w2, …… wN*** and their frequencies and ***f1, f2 …… fN.*** N is the first entry in the file for E1

⇒ Read the total number of spam and total number of normal mails from the file that stores the all words. These are the very first 2 entries in that file. ***spamTotal, normalTotal***

⇒ Read the feature vector file (FV) into memory. Fields are FV_j , $spamNum_j$ $normalNum_j$.

⇒ Read the total number of words in this email from its file to the variable ***totalWords***.

⇒ Read the words and their frequencies in the incoming email file into memory.

⇒ For Each class ***i*** (spam and normal)

- Than for each word FV_j in feature vector file calculate as:
  - Pij equals:
    - if i = spam, spamNumj divided by totalSpam
    - Else i = normal, $normalNum_j$ divided by totalNormal.
  - If the word FV_j occurs in email ***E1***
    - Probability(C_i) += C*P_{ij} (I is either spam or normal)
  - Else
    - Probability(C_i) -= P_{ij}

⇒ If P(C_{spam}) > P(C_{normal})

- Conclude that E1 is spam

⇒ Else

- Conclude that E1 is normal.

4.2.b. Probabilistic Model

Here we take into consideration number of times that a given word occurs in the e-mail. We are using the following formula to compute the probability by adding H_j to our existing model.

$$P(C_i/X) = \sum_{J=1}^{N} \begin{cases} c(P_{ij} H_j), & \text{If jth word of feature vector occurs in e - mail} \\ -P_{ij,} & \text{otherwise} \end{cases}$$

$$H_j = \#\text{ of occurence of j th word in the e - mail}$$

```
procedure probabilisticBayesian ( )
1:  Read the feature Vector from the file
2:  Read the given e-mail into memory
3:  For i = 0 to classnum do
4:          For i = 0 to n do
5:  If word_i is in e-mail then
6:                  Probability(C_i/X) += C* P_ij *H_j
7:              else
8:                  Probability(C_i/X) -= P_ij
9:          end for
10: end for
11: if Probability(C_normal/X) > Probability(C_spam/X)
12:       E-mail is normal
13: else
14:       E-mail is spam
```

Figure 10. Probabilistic Bayesian algorithm

Example Scenario

⇒ Assume we have formed the files for all words, emails and feature vector and we have an incoming email ***E1,*** with ***N*** words ***w1, w2, …… wN*** and their frequencies and ***f1, f2 …… fN.*** N is the first entry in the file for E1

⇒ Read the total number of spam and total number of normal mails from the file that stores the all words. These are the very first 2 entries in that file. ***spamTotal, normalTotal***

⇒ Read the feature vector file (FV) into memory. Fields are FV_j , $spamNum_j$ $normalNum_j$.

⇒ Read the total number of words in this email from its file to the variable ***totalWords***.

⇒ Read the words and their frequencies in the incoming email file into memory.

⇒ For Each class ***i*** (spam and normal)

- Than for each word FV_j in feature vector file calculate as:
  - Pij equals:
    - if i = spam, spamNumj divided by totalSpam
    - Else i = normal, $normalNum_j$ divided by totalNormal.
  - If the word FV_j occurs in email ***E1***
    - Probability(C_i) += $C*P_{ij}*f_j$ (i is either spam or normal)
  - Else
    - Probability(C_i) -= P_{ij}

⇒ If $P(C_{spam})$ > $P(C_{normal})$

- Conclude that E1 is spam

⇒ Else

- Conclude that E1 is normal.

4.2.c. Advanced Probabilistic Model

In advanced probabilistic model we take into account the total number of words in the incoming e-mail. We use the following formula to compute the probability values :

$$P(C_i/X) = \sum_{J=1}^{N} \begin{cases} c(P_{ij}\ H_j)/counter, & \text{If jth word of feature vector occurs in e - mail} \\ -P_{ij,}/\text{counter} & \text{otherwise} \end{cases}$$

$$Counter = \text{Total number of words in that e - mail}$$

```
procedure advancedProbabilisticBayesian ( )
1:   Read the feature Vector from the file
2:   Read the given e-mail into memory
3:   For i = 0 to classnum do
4:        For i = 0 to n do
5:             If word_i is in e-mail then
6:                  Probability(C_i/X) += C* P_ij *H_j / counter
7:             else
8:                  Probability(C_i/X) -= P_ij / counter
9:        end for
10:  end for
11:  if  Probability(C_normal/X) > Probability(C_spam/X)
12:       E-mail is normal
13:  else
14:       E-mail is spam
```

Figure 11.Advanced Probabilistic Bayesian algorithm

Example Scenario

⇒ Assume we have formed the files for all words, emails and feature vector and we have an incoming email ***E1,*** with ***N*** words ***w1, w2, …… wN*** and their frequencies and ***f1, f2 …… fN.*** N is the first entry in the file for E1**.**

⇒ Read the total number of spam and total number of normal mails from the file that stores the all words. These are the very first 2 entries in that file. ***spamTotal, normalTotal***

⇒ Read the feature vector file (FV) into memory. Fields are FV_j , $spamNum_j$ $normalNum_j$.

⇒ Read the total number of words in this email from its file to the variable ***totalWords***.

⇒ Read the words and their frequencies in the incoming email file into memory.

⇒ For Each class ***i*** (spam and normal)

- Than for each word FV_j in feature vector file calculate as:

- Pij equals:
  - if i = spam, spamNumj divided by totalSpam
  - Else i = normal, $normalNum_j$ divided by totalNormal.
- If the word FV_j occurs in email ***E1***
  - Probability(C_i) += $C*P_{ij}*f_j$ / N (i is either spam or normal)
- Else
  - Probability(C_i) -= P_{ij} / N

⇒ If P(C_{spam}) > P(C_{normal})

- Conclude that E1 is spam

⇒ Else

- Conclude that E1 is normal.

4.3. Tracing N-Gram Model

Below is a simple chart, figure 11, showing the flow of the program in N-Gram method. Our program first reads the incoming e-mail from the file. Then by using the formulas we calculate the probability of the given e-mail of being normal and spam. We conclude that the e-mail is spam if probability of being spam is greater than normal and vice versa

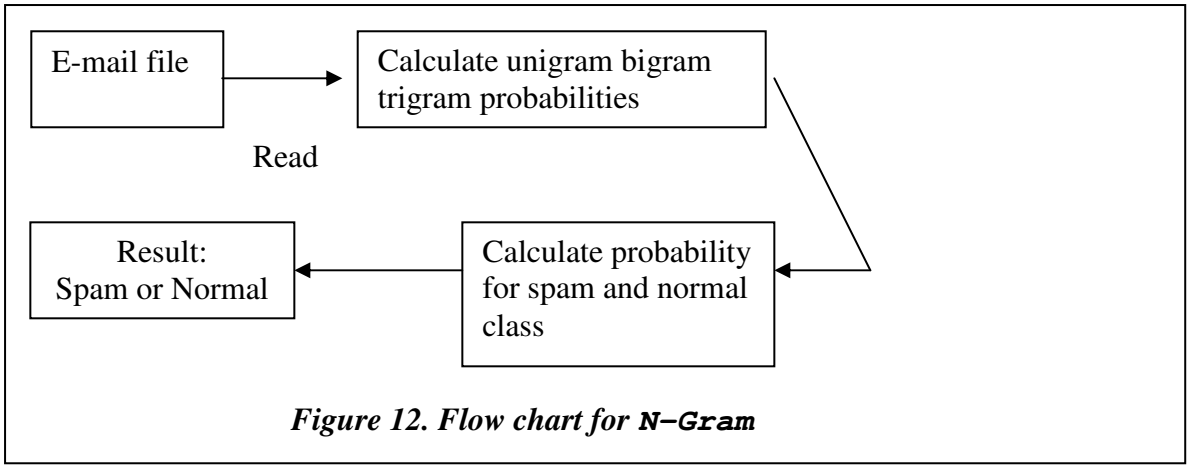


Figure 12. Flow chart for N-Gram

4.3.a. Class General Perception (CGP) Model

[2] The class general perception (CGP) model groups the e-mails in two classes: the spam class and the normal class. This is the traditional approach used in spam filtering. The goal of the perception model is then, given an incoming e-mail, to calculate the probability of belonging to the spam class and the probability of belonging to the normal class. Bayes formula enables us to compute the probabilities of word sequences (w1…wn) given that the perception is spam or normal. In addition, n-gram model enables us to compute the probability of a word given previous words. Combining these and taking into account n-grams for which $n \leq 3$, we can arrive equations that we can use for probabilities.

A common problem faced by statistical language models is the sparse data problem. We form methods by taking the sparse data problem into account. To this effect, two methods are proposed. In method-1 the unigram, bigram, and trigram probabilities are totaled for each word in the e-mail and calculate a probability. But this method is time consuming while training. So we give equal weight to each factor.Method-2 is based on the intuition that n-gram models perform better as n increases. In this way, more dependencies between words will be considered; a situation which is likely to increase the performance. We use, trigram probabilities when there is sufficient data in the training set. If this is not the case, bigram probabilities are used, and unigram probabilities are used only when no trigram and bigram can be found. It is still possible that the unigram probabilities may evaluate to zero for some words in the test data, which has the undesirable effect of making probability equal to zero. The usual solution is ignoring such words.

```
Procedure classGeneralPerception ( )
1:  Read the given e-mail into memory.
2:  For i = 0 to n do  // n is the total word number
3:          Calculate trigram bigram and unigram probabilities // formulas are in below
            according to normal and spam class
4:  end for
5:  Calculate P(E|spam) and P(E|normal) //formulas are below
6:  If  P(E|spam) > P(E|normal)
```

```
7:          e-mail is spam
8:  else
9:          e-mail is normal
```

Figure 13 Class General Perception model (CGP)

To calculate trigram, bigram and unigram probabilities we use formulas:

$$P(w_i|C) = \frac{\text{number of occurrences of } w_i \text{ in class C}}{\text{number of words in class C}}$$

$$P(w_i|w_{i-1},C) = \frac{\text{number of occurrences of } w_{i-1}w_i \text{ in class C}}{\text{number of occurrences of } w_{i-1} \text{ in class C}}$$

$$P(w_i|w_{i-2},w_{i-1},C) = \frac{\text{number of occurrences of } w_{i-2}w_{i-1}w_i \text{ in class C}}{\text{number of occurrences of } w_{i-2}w_{i-1} \text{ in class C}}$$

where C denotes class spam or normal.

There are two methods for calculating P(E|spam) and P(E|normal). In method 1 we use formula:

$$P(C|E) = \sqrt[n]{\prod_{i=1}^{n}\left[P(w_i|C) + P(w_i|w_{i-1},C) + P(w_i|w_{i-2},w_{i-1},C)\right]}$$

In method 2 we use formula:

$$P(C \mid E) = \sqrt[n]{\prod_{i=1}^{n} \eta_i}$$

where

$$\eta_i = \begin{cases} P(w_i|w_{i-2},w_{i-1},C) & \text{, if } P(w_i|w_{i-2},w_{i-1},C) \neq 0 \\ P(w_i|w_{i-1},C) & \text{, if } P(w_i|w_{i-1},C) \neq 0 \text{ and } P(w_i|w_{i-2},w_{i-1},C) = 0 \\ P(w_i|C) & \text{, otherwise} \end{cases}$$

Example Tracing for CGP model

Our e-mail is formed from 4 words. Its sequence is w1w2w3w4
Steps of the decision:

➔E-mail is read to the memory.
➔Calculate unigram bigram and trigram probabilities.
➔We find n-gram probabilities from our n-gram probability file.

(su1,nu1)=P(w1|(spam,normal))=number of occurances of w1 in (spam,normal) class / number of words in (spam,normal) class
(su2,nu2)=P(w2|(spam,normal))= number of occurances of w2 in (spam,normal) class / number of words in (spam,normal) class
(su3,nu3)=P(w3|(spam,normal))= number of occurances of w3 in (spam,normal) class / number of words in (spam,normal) class
(su4,nu4)=P(w4|(spam,normal))= number of occurances of w4 in (spam,normal) class / number of words in (spam,normal) class

➔To find unigram probabilities our program start searching n-gram occurance file. When it finds the word it looks at the occurance number of spam and normal class. (file format is word / # of occurance in normal class / # of occurance in spam class). Then it divides into the total number of words in file. (It is written at the start of the file.). By doing this program calculates the unigram normal (nu1) and unigram spam (su1) probabilites.

(sb1,nb1)=P(w2|w1, (spam,normal))=number of occurances w1w2 in (spam,normal) class / number of occurances of w1 in (spam,normal) class
(sb2,nb2)=P(w3|w2, (spam,normal))=number of occurances w2w3 in (spam,normal) class / number of occurances of w2 in (spam,normal) class
(sb3,nb3)=P(w4|w3, (spam,normal))=number of occurances w3w4 in (spam,normal) class / number of occurances of w3 in (spam,normal) class

➔To find bigram probabilites our program start searching n-gram occurance file. When it finds the word sequence(w1w2) it looks at the occurance number of spam and normal class. (file format is word sequence(w1w2) / # of occurance in normal class / # of occurance in spam class) Then program searches for w1 word in same file and find its occurance number. After that program calculates the bigram normal (nb1) and spam (sb1) probabilities.

(st1,nt1)=P(w3|w1,w2, (spam,normal))= number of occurances w1w2w3 in (spam,normal) class / number of occurances of w1w2 in (spam,normal) class
(st2,nt2)=P(w4|w2,w3, (spam,normal))= number ofg occurances w2w3w4 in (spam,normal) class / number of occurances of w2w3 in (spam,normal) class

➔Program calculates trigram probabilities like bigram and unigram. After that the message's spam probability is calculated by formula
➔$P(spam|E)=\sqrt[n]{[su1] * [su2+sb1] * [su3+sb2+st1] * [su4+sb3+st2]}$ where n = 4
➔After that the message's spam probability is calculated by formula above.
➔$P(normal|E)=\sqrt[n]{[nu1] * [nu2+nb1] * [nu3+nb2+nt1] * [nu4+nb3+nt2]}$ where n = 4
➔The message's normal probability is calculated by formula above too.
➔If P(spam|E) > P(normal |E) mail is spam otherwise normal
➔At last spam and normal probabilities are compared and mail is categorized.

4.3.b. E-Mail Specific Perception (ESP) Model

[2] In e-mail specific perception (ESP) model, each e-mail is considered as a separate class, in contrast to CGP model where we have only two generic classes. The goal is to find the similarity of an incoming e-mail to the individual e-mails in the data set. The e-mail is classified as spam (or normal) if its content is more similar to the contents of some of the spam (or normal) messages than the contents of all the normal (or spam) messages. The difference from CGP model is that we do not take an average of the similarity for the whole spam or normal class, instead we consider a few most similar e-mails and decide on the class of the incoming e-mail based on these. The intuition behind this model is that people frequently receive (especially spam) messages with same or similar content. In such cases, the correct class of a message can be determined by matching it with a highly similar previous message.

```
Procedure e-mailSpecificPerception ( )
1:  Read the given e-mail into memory.
2:  For i = 0 to n do  // n is the total spam and normal mail number
3:  Calculate trigram bigram and unigram probabilities with formulas in CGP model
            //all mails are classes in this method
4:  Calculate probability of being like that mail or not according to formula in CGP
5:  end for
6:  Take the highest 10 probabilities calculated above
7:  For i=1 to 10 do
8:  If  probability[i] is calculated with a spam mail
9:  sum=sum-1
10: Else
11: sum=sum+1
12: end for
13: If  sum>0
14: e-mail is normal
15: else
16: e-mail is spam
```

Figure 14 E-Mail Specific Perception (ESP)

Example tracing for ESP model

Our e-mail is formed from 4 words. Its sequence is w1w2w3w4
Steps of the decision:
➔E-mail is read to the memory.
➔In CGP model there was two classes normal and spam but in ESP model we take all of the mails as a class.
➔When calculating probabilities we have N different probability where N is the total number of e-mails in our training set.
➔Program first calculate probability according to CGP formulas for first e-mail. Program does not look any special file for probabilities. Because of we save all e-mails, program opens first e-mail look at unigram bigram and trigram probabilities and close it. İt is a long job so we use CPR model.
➔Then calculating goes for N mails. At last we have N different probabilities from N different mails.
➔Program sort them descending.
➔Then take the most largest 10 probabilities.
➔At last program looks if 10 probabilites come from normal mails or spam mails.
➔If more of the probabilites come from normal mails then our mail is normal else it is spam.

4.3.c. Combined Perception Refinement (CPR)

[2] ESP is a good way for spam filtering but due to its time complexity we can not use it. Instead of using only ESP, we can combine it with CGP model and can get a better model. In CGP there can be some misclassified e-mails so we can use ESP model to classify that e-mails. We call these misclassified mails as uncertain region e-mails. When an e-mail comes by using CGP model we try to clasiify it. If we can not classify we compute an upperbound and lowerbound for this uncertain region than we use ESP model to classify it as a normal or spam.

```
Procedure combinedPerceptionRefinement ( )
1:  Read the given e-mail into memory.
2:  Run CGP model for e-mail
3:  Calculate upper and lower bound for uncertain region // formulas are below
4:  If  e-mail is not in uncertain region
5:  terminate program
6:  else
7:  run ESP model for only uncertain region e-mails
```

Figure 15 Combined Perception Refinement (CPR)

To calculate upper and lower bound we use formula:

$$f_{UB} = \max\{\max\{f(E):E \in VS \text{ and } E \text{ is spam},\}, 1\}$$
$$f_{LB} = \min\{\min\{f(E):E \in VS \text{ and } E \text{ is legitimate}\}, 1\}$$

where

$$f(E) = \frac{P(\text{legitimate}|E)}{P(\text{spam}|E)}$$

This upper bound and lower bound are calculated from validation set.

Example tracing for CPR model

Our e-mail is formed from 4 words. Its sequence is w1w2w3w4
Steps of the decision:
➔E-mail is read to the memory.
➔We run CGP model as in told 4.3.a.
➔If some mails are misclassified (means if they are in upper and lower bound)
➔Then run ESP model to classify them.
➔Else terminate program.

5. Integration of these Methods with E-mail clients

For these methods to be useful for the users, we should integrate these methods into e-mail client programs. By this we can make the utilization of our methods that we have implemented. As for integration the system into e-mail client programs, we have a big problem. There are a plenty of e-mail clients for example Outlook Express, Microsoft Outlook, Thunderbird, Eudora and etc. More interestingly each and every e-mail client has its own way to store the e-mails. Outlook express uses a file with the extension ***".dbx"*** to store the e-mails. There is a file with the extension ***".pst"*** to store the e-mails in Microsoft Outlook. Finally, Thunderbird uses 2 files two store the e-mails. For the folder **Inbox** there are two files named as ***Inbox*** and ***Inbox.msf*** .

Here the problem is that we can not write a generic program which integrates our methods into all e-mail clients since every client program has its own way of storing e-mails. What we have to do is to pick up one of the clients to integrate our methods into. We picked up Outlook Express to integrate our methods.

5.1. Outlook Express

Outlook Express was an e-mail/news client that was included in versions of Microsoft Windows from Windows 98 to Windows XP Service Pack 2 (SP2); it has since been superceded by Windows Mail. Hence every windows user has implicitly installed this software. In other words Outlook Express is one of the most widely used e-mail clients. Another reason why we have picked up Outlook Express is that there is no available Turkish Spam Filter for that e-mail client.

5.1.a. Understanding the way Outlook Express stores E-mails

All of your Outlook Express (OE) mail folders and messages, local IMAP and Hotmail folders and messages, and all of your subscribed newsgroups and messages are stored in one folder called the store root, or store root folder, or store folder, and in at least one place in OE, the Store root directory. It's default location is

C:\Windows\Application Data\Outlook Express**{GUID}**
or

C:\Documents and Settings\<User>\Local Settings\ApplicationData\Identities**{GUID}**\Microsoft\Outlook Express

where **{GUID}** is the Global Unique IDentifier (technical jargon for a unique long number) used to specify an Identity. Depending on your operating system and upgrade history, your store folder might be located elsewhere.

You can use the following instructions to determine the location of your store, click ***Tools*****|** ***Options*****|** ***Maintenance*****|** ***Store folder.***

Opening the store root in Windows Explorer reveals a collection of *.dbx files. These are the files where all the messages are actually stored. The **Folders.dbx** file is the master index of the entire store. Likewise, the *.dbx files for the OE default mail folders will also be created if missing. Special attention should be paid to the Inbox, Sent Items and Deleted Items folders. It is easy for these folders to grow extremely large. OE is not a dedicated database program, and can have problems with very large files, making them prone to corruption and data loss. For this reason alone, it is not suggested to use the Inbox to store messages. Instead, creating other folders for storing mail and leave the Inbox for incoming mail is a better way.

Folders.dbx

This is the master index of an OE store folder and is required for OE to run. It stores the tree structure (nesting) of mail folders, the lists of newsgroups on each news account, and the synchronization options for all subscribed newsgroups and Hotmail folders. If it is missing when OE starts, OE will scan the store folder for all *.dbx files and then create a new Folders.dbx.

Inbox.dbx ***Drafts.dbx***
Sent Items.dbx ***Deleted Items.dbx***

These are the default mail folders, created automatically if missing. Should one of these stop working, move all messages into another folder, close OE, and delete the corresponding *.dbx file.

Offline.dbx

This stores IMAP and Hotmail actions that you carry out offline, and are created only if an IMAP or Hotmail account exists

Pop3uidl.dbx

This keeps track of messages left on a POP3 mail server, and is created automatically if missing. If OE starts downloading the same mail that you've already read, close OE and delete this file.

<name>.dbx

This represents a user-created mail folder and its messages. The number of folders is limited only by your disk space, but don't let any one folder grow above 2Mb or so or performance is likely to suffer.

<newsgroup>.dbx

This represents a newsgroup folder. All newsgroup folders are tied to the news account keys in the registry and so cannot normally be imported

5.1.b. Extracting E-mails From Outlook Express

We see the basic structure of How Outlook Express(OE) stores e-mails in the local folder. Now to integrate our program into OE, we should extract the stored e-mails from these files. As it is clearly cited above, first we should connect to the local database and get the folder properties starting from the root folder. We are using the ***msoeapi.h*** to connect to the local database.

We have IStoreNameSpace class that has the required functions to connect to the local database of OE. First we should create an object from this class to connect to the local database.

```
// Create an instance of an IStoreNamespace object.

  CoCreateInstance(
      CLSID_StoreNamespace,    // Namespace ClassID
      NULL,                   // Outer unknown that aggregates the new object
      CLSCTX_INPROC_SERVER,    // a server dll
      IID_IStoreNamespace,     // interface ID
      (LPVOID*)&pStore);       // The returned IStoreNamespace object
```

Here we should explicitly specify the interface ID in accordance with our IStoreNamespace object.

After creating the object we should initialize that object and then we can get the first subfolder in the tree structure by using the member function of IStoreNamespace class.

```
// FOLDERID_ROOT already defined
//fp Object that stores folder properties
//hEnum Object that is used for enumaration
pStore->GetFirstSubFolder(FOLDERID_ROOT, &fp, &hEnum);
```

We have the following figure(**Figure 16**) which simply summarize the way to connect to the files that store the e-mail information.

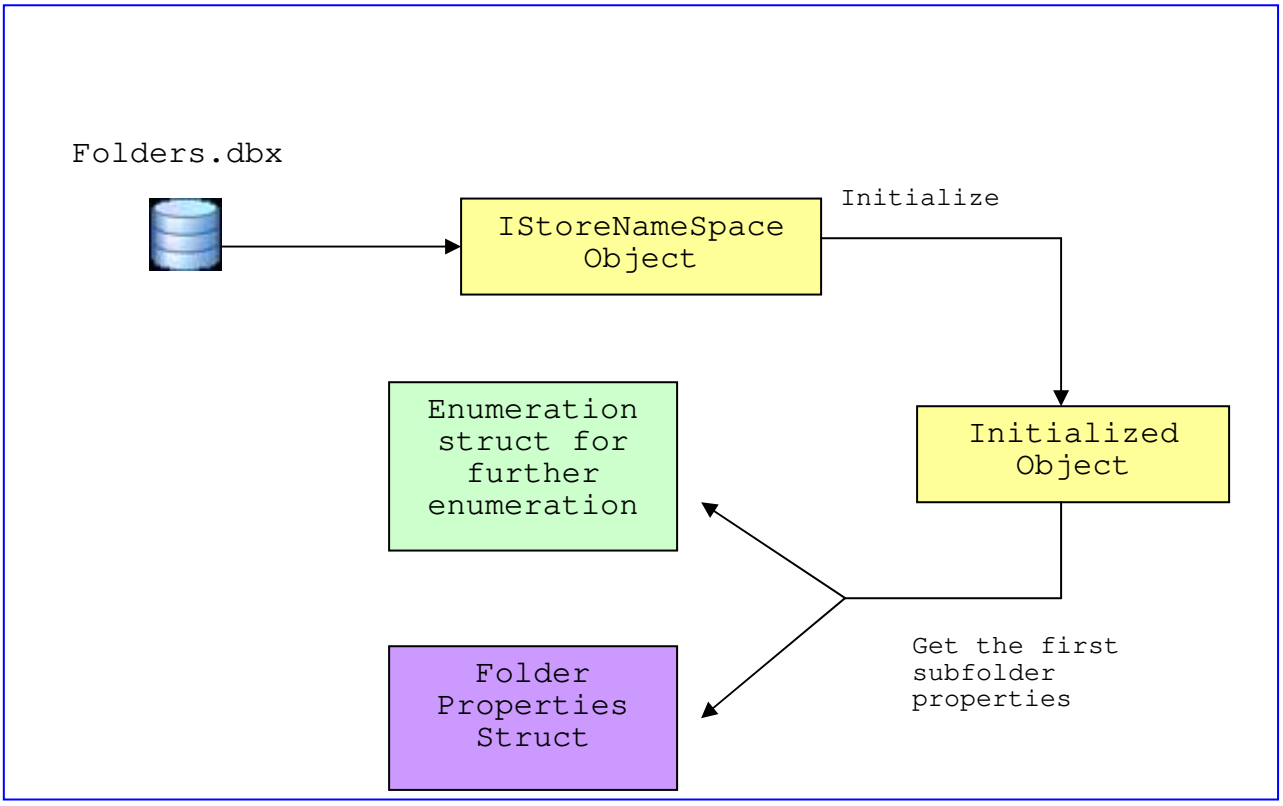


Figure 16. Getting the folder properties from OE database

After getting the folder properties, we should determine which folder to open. We can check the name of the folders from the folder properties struct. We have the following fields in FOLDERPROPS struct:

```
typedef struct tag FOLDERPROPS {
   DWORD cbSize;
   STOREFOLDERID dwFolderId;
   INT cSubFolders;
   SPECIALFOLDER sfType;
   DWORD cUnread;
   DWORD cMessage;
   CHAR szName[CCHMAX_FOLDER_NAME];
} FOLDERPROPS, *LPFOLDERPROPS;
```

cbSize
The size of this structure in bytes.
dwFolderId
ID of the folder these properties represent.
cSubFolders
Count of the number of child folders underneath this folder.
sfType
Specifies the type of message folder. Corresponds to a type in the SPECIALFOLDER enumeration.
cUnread
The number of unread messages in this folder.
cMessage
The total number of messages in this folder.
szName
The display name of this folder.

We can check the name of the folder, number of unread messages as well as the number of total messages by using this struct instance after getting the folder properties. After determining the folder to open, we can open that folder by using the OpenFolder function which is a member of IStoreNamespace class.

We should open the folder by giving the folder ID from the properties struct. We should also give a IStoreFolder instance to the function as a pointer to refer to the folder.

```
hr=pStore->OpenFolder(fp.dwFolderId, 0, &pFolder);
```

If this function returns a success and if we have e-mails to read in the folder we can open up the e-mails that pFolder is pointing to. As in the folder properties case, we have another struct to store the message properties.

```
typedef struct tagMESSAGEPROPS {
   DWORD cbSize;
   DWORD dwReserved;
   MESSAGEID dwMessageId;
   DWORD dwLanguage;
   DWORD dwState;
   DWORD cbMessage;
   IMSGPRIORITY priority;
   FILETIME ftReceived;
   FILETIME ftSent;
   LPSTR pszSubject;
   LPSTR pszDisplayTo;
   LPSTR pszDisplayFrom;
   LPSTR pszNormalSubject;
   DWORD dwFlags;
   IStream *pStmOffsetTable;
} MESSAGEPROPS, *LPMESSAGEPROPS;
```

Members

cbSize
Size of the structure in bytes.
dwReserved
Reserved value.
dwMessageId
ID value of the message.
dwLanguage
The language codepage of this message. For a list of acceptable values, see CODEPAGEID.
dwState
Message state flags. For a list of acceptable values, see Message State Flags.
cbMessage
Size of the message in bytes.
priority

The priority level of the message. May be IMSG_PRI_LOW, IMSG_PRI_NORMAL, or IMSG_PRI_HIGH.

ftReceived

The time at which this message was received.

ftSent

The time at which this message was sent.

pszSubject

The subject line of the message.

pszDisplayTo

The displayed names on the To: line of the message.

pszDisplayFrom

The displayed names on the From: line of the message.

pszNormalSubject

The normalized form of the subject line of the message.

dwFlags

MIME flags that specify the state of the message. For a list of acceptable values, see IMSGFLAGS.

pStmOffsetTable

An offset table to facilitate quicker message loading. See IMimeMessageTree::LoadOffsetTable.

As in the folder case, with the following function we get the properties of the root message in the folder.

```
pFolder->GetFirstMessage(0, 0, MESSAGEID_FIRST, &mprops, &hMEnum);
//hMEnum Object that is used for enumaration
//mprops Object that is used for storing the properties of messages
//MESSAGEID_FIRST already defined value
```

We need an object from IMimeMessage class in order to get the content of the opening message. As in the folder case we open the message with the following member function of IStoreFolder Class.

```
hr = pFolder->OpenMessage(mprops.dwMessageId, IID_IMimeMessage, (VOID**)&pMimeMessage);
```

Here we get an ImimeMessage object pointing to the recently opened message. Now, we have reached an object pointing to the message in one of the folders of Outlook Express files.

Then we get an HBODY object which points to the body of the message.

```
hr = pMimeMessage->GetBody(IBL_ROOT, 0, &m_hCurBody);
```

Then to read the content of the body first we bind the body.

```
hr = pMimeMessage->BindToObject(m_hCurBody,
                                IID_IMimeBody,
                                (LPVOID *) &pMimeBody);
```

Then we get the body as a text with the following function.

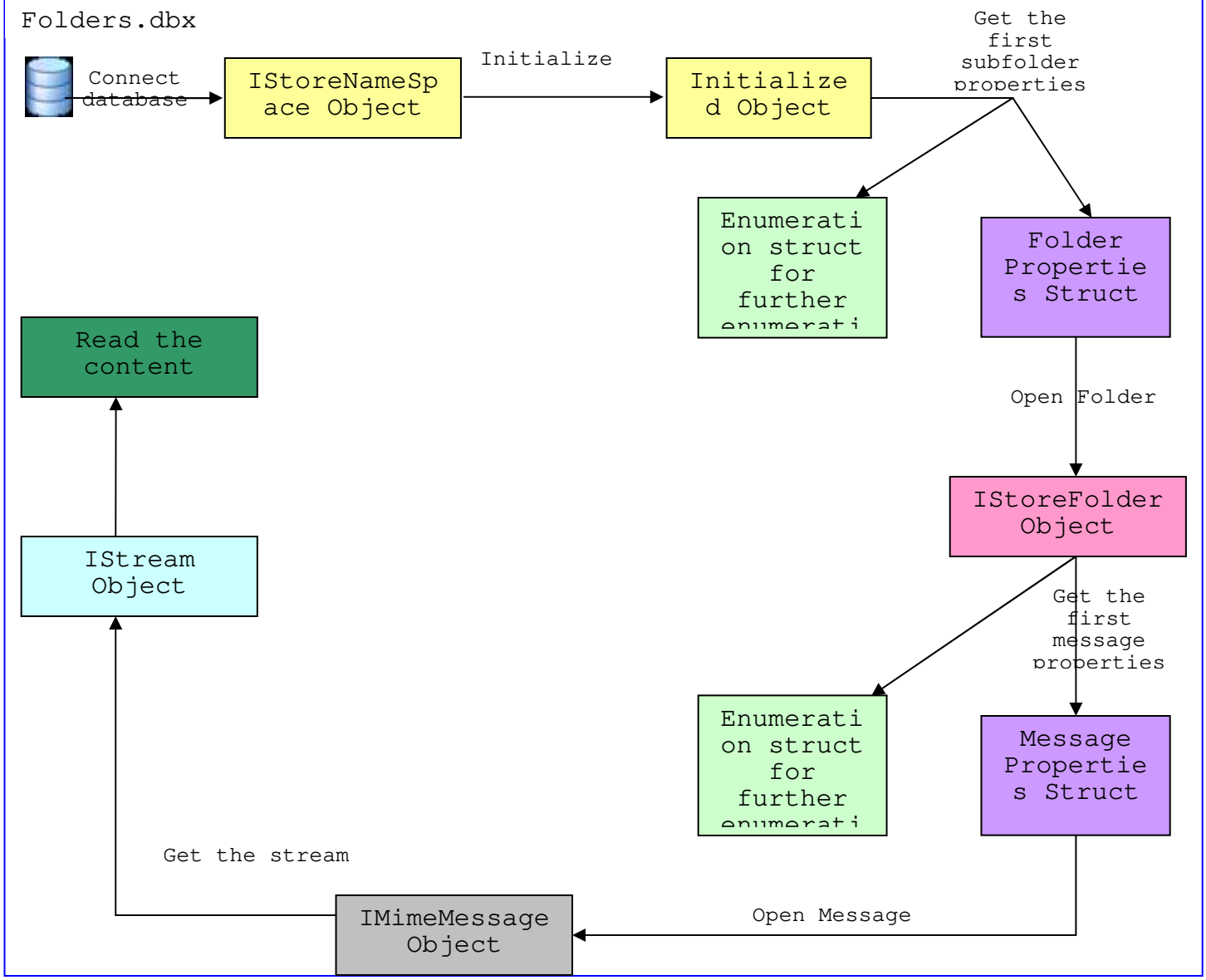
```
hr = pMimeMessage->GetTextBody( contentType,encType,&pBodyStream,phBody);
```

Notice that we have to give content type and encoding type to this function. We do this by getContentType function and from content type we extract the encoding type.

After that we can start to read the content of the message by using the IStream object.

```
hr = pBodyStream->Read(lpszwBody,sizeof(lpszwBody)-sizeof(WCHAR),&ulRead);
```

Figure 17. Extracting E-mails from Outlook Express Database



5.1.c. Processing the extracted e-mails

After extracting the emails our aim is to process those emails into the format that we have explained in the Section 2 of this document. We should form those files in order for our methods can run with these files.

By processing these extracted e-mails we should form the E-mail files, all words file ,the feature vector file and the n-gram probabilities file.

5.1.d. RoadMap of Integrating the Methods into Outlook Express

We have the following roadmap to integrate our methods into the outlook express.

A. Prepare a from in C #

B. Extract Emails from Outlook Express local Database

C. Process those extracted emails into the required format.

D. Re implement the methods to support Turkish characters(*)

E. Embed the code into the form

6. Integration of C Code into C#

C# is a higher language than C. It is more compex and more usable. We can prepare windows forms and prepare windows programmes. The methods in this report we told are written in C language and we want a user interface. In C we can not a graphical user interface so C# is a better solution.

The problem starts here. You can not call C functions from C# directly so we have to find another solution. Solution is .dll files. In C# you can import .dll files and can call its functions. Of course .dll files are writen with C language. In below I will tell how we can prepare .dll files and how we can import into C# and call functions. A small example will be enough.

6.1. Example .dll file

We will write a small C program, make it a .dll file, import into C# and call that function.

Our example starts with a small C program. In Visual Studio we open a empty .dll project and write below code.

//hello.c

```
#include <stdio.h>
```

```
extern "C"
{
  __declspec(dllexport) void DisplayHelloFromDLL()
  {
    printf ("Hello from C DLL !\n");
  }
}
```

When we compile and build the solution we have a ***hello.dll*** file

Then we open an empty C# project ***CDllcaller***. We write below code into the screen.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices; // required for loading the C-style
dll.
namespace CDllcaller
{
    class Program
    {
        // hello.dll is a C-style dll which exports the DisplayHelloFromDll
function!
        [DllImport("C:/Documents and Settings/program/hello.dll")]
        public static extern void DisplayHelloFromDll ();

        static void Main(string[] args)
        {
            DisplayHelloFromDll();

            while (true) ;}}}
```

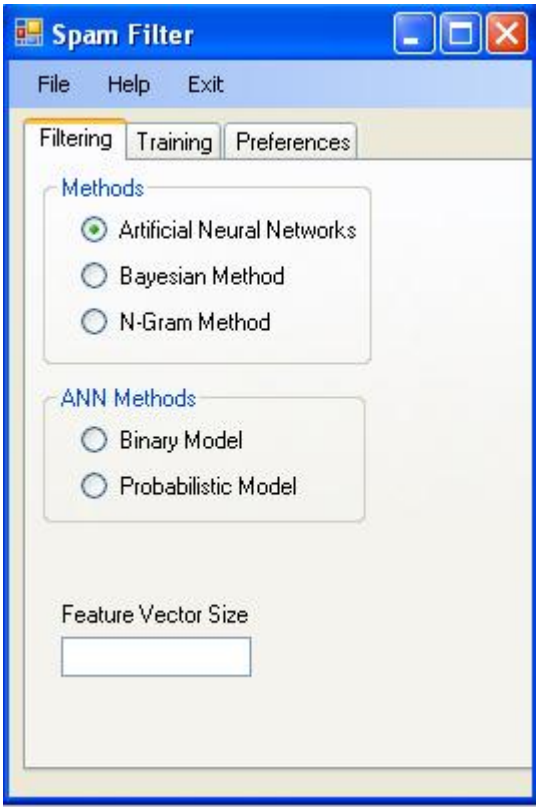
In this code segment we import hello.dll by DllImport function. Then we import the DisplayHelloFromDll() function by extern keyword. Extern keyword means that function is from dll. At last we call DisplayHelloFromDll() function in main.

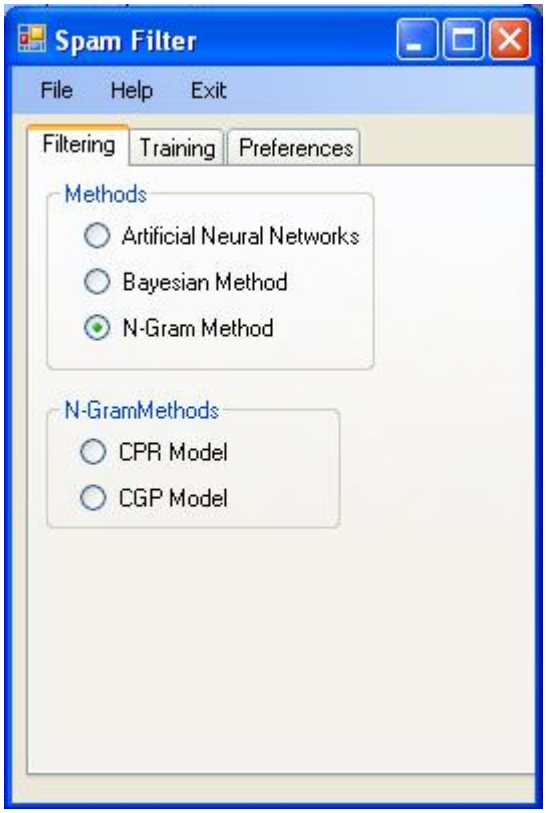
By using this way we can use C code in C# but it is costly and not dynamic. You have to use all functions in their proper way.

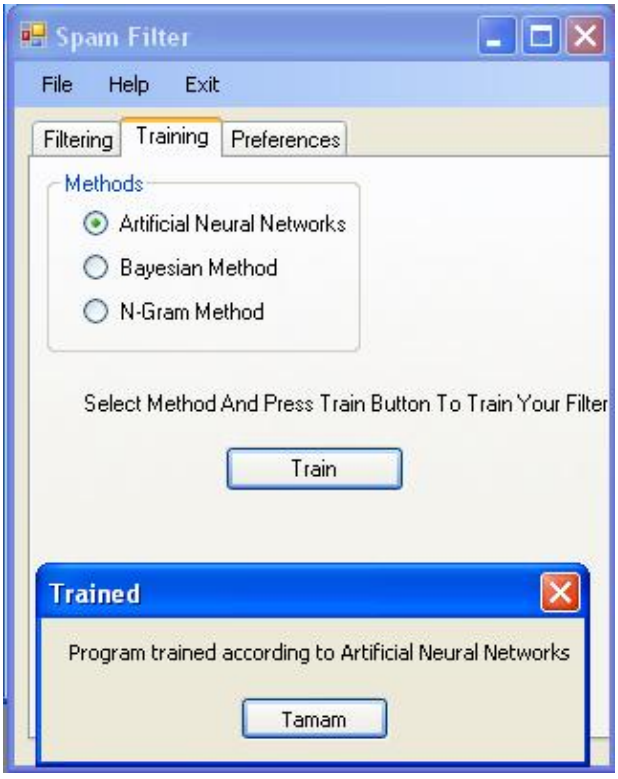
7. GUI in C#

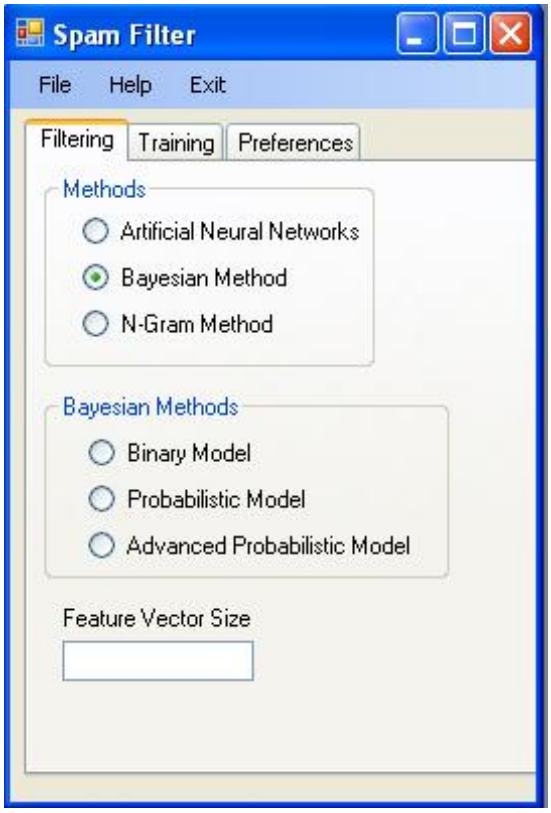
GUI for our spam filter program can be prepared in C#. For this purpose we have to know what our needs are. We implemented three different methods for filtering spams. So we have to select which method we can use for filtering. Also all methods have other ways so we have to select method's ways.

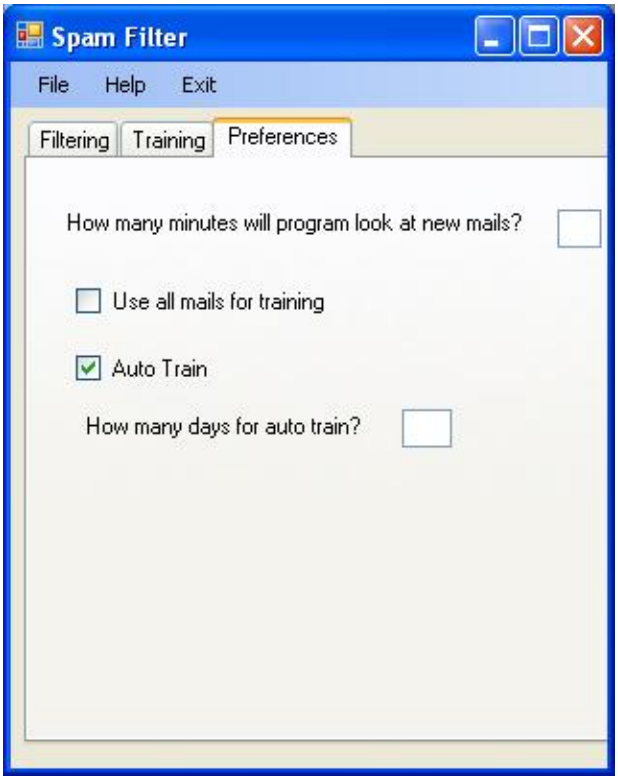
In C# we open a new windows project and prepare our form for graphical user interface.


Spam Filter
File
Help
Exit
Filtering
Training
Preferences
Methods
Artificial Neural Networks
Bayesian Method
N-Gram Method
ANN Methods
Binary Model
Probabilistic Model
Feature Vector Size


Spam Filter
File
Help
Exit
Filtering
Training
Preferences
Methods
Artificial Neural Networks
Bayesian Method
N-Gram Method
N-GramMethods
CPR Model
CGP Model


Spam Filter
File
Help
Exit
Filtering
Training
Preferences
Methods
Artificial Neural Networks
Bayesian Method
N-Gram Method
Select Method And Press Train Button To Train Your Filter
Train
Trained
Program trained according to Artificial Neural Networks
Tamam


Spam Filter
File
Help
Exit
Filtering
Training
Preferences
Methods
Artificial Neural Networks
Bayesian Method
N-Gram Method
Bayesian Methods
Binary Model
Probabilistic Model
Advanced Probabilistic Model
Feature Vector Size



8. WHY RE-IMPLEMENTING METHODS?

It is very important to integrate the methods for Turkish Spam filtering into Outlook express. For this purpose, there some advantages of re-implementing the methods.

The most important fact is that since we want to develop a spam filter for Turkish, we should undoubtedly support Turkish characters as well as Turkish words during spam filtering. Since special Turkish characters occupy 2 bytes, we should re-implement in accordance with this fact. Here storing the e-mail in the CString format may work because we already read the email content in that format and if we write our methods also in that format we are done.

Another reason to re-implement is that there are a lot of global variables and static declarations in the code that we have developed our methods onto which reduces the flexibility of methods. By rewriting the code we can prevent these and end up in a more flexible program.

Another reason for re-implementing is user interface and c synchronization. We use C# for GUI and C for method implementing. But we can not run C dlls in windows forms, only we can run as a console application. So re-implementing is a good solution for this problem.

Finally, by rewriting the code we can make use of the high level programming facilities (object oriented approach, split, regular expressions…etc) which increases the efficiency of the program.

REFERENCES

[1] Özgür, L., Güngör, T. and Gürgen, Adaptive Anti-Spam Filtering for Agglutinative Languages: A Special Case for Turkish, Pattern Recognition Letters, Vol.25(16), 2004, p.1819-1831.
[2] Güngör, T , Developing Methods and Heuristics with Low Time Complexities for Filtering Spam Messages
[3] http://en.wikipedia.org/wiki/Outlook_Express
[4] http://www.insideoe.com/files/store.htm
[5] http://msdn2.microsoft.com/en-us/library/ms715241.aspx
[6] http://msdn2.microsoft.com/en-us/library/ms709546.aspx (for detailed information)