The Mail fingerprint model based on .NET for authorship attribution and its visualization

Abstract:

We established the Mail fingerprint model to solve the problem of achieving authorship attribution of e-mails through handwriting analysis.

Firstly, we preprocessed the obtained mail from the Enron Email Datasets in SQL, gaining the body part, and classified the mails sent from the same mailbox. Then we merged them into the text sets. We toke 1/10 e-mails of a text sets as a test text, and the other 9/10 text sets and the text sets taken from other different mailboxes as training texts.

We selected five identifiable linguistic features (high frequency words, long sentence usage, vocabulary, adjective usage, and recipient appellation writing habits), and determined their corresponding model parameters. Through the analytic hierarchy process, the weight of each model parameter was obtained.

The mail fingerprint consists of the model parameter of each text set and its weight, which indicates the author's linguistic feature.

By comparing the mail fingerprint of the test text with the training text, the difference value D between the text sets was obtained. Authorship attribution is achieved through comparing the difference value D . Furthermore, we realized visualization of differences in linguistic features through the software MailFingerPrinter developed by ourselves.

By testing the sensitivity of the model, we determined that about at least 500 emails which contains short texts of messages were required as training texts for the model to achieve the authorship attribution.

Key words: mail fingerprint handwriting analysis linguistic features visualization authorship attribution analytic hierarchy process MailFingerPrinter

Contents

I. Introduction

1.1 Background of the problem

Handwriting analysis is an analytical tool that is essential in many kinds of situations. By analyzing existing text or electronic textual material, one can associate certain person to written evidence, and determine the author by written material handwriting, linguistic style and other information. This means of analysis has a very important application in the courts or criminal investigation.

At present a lot of linguistic evidence appears in the email so that we can not judge the author through handwriting. We can only judge the author through the linguistic style. The contents of e-mails tend to be short, and the author's linguistic style is quite obvious. However, there are few effective methods of identification so far. It is urgent to establish a highly accurate mathematical model to identify authors by capturing the linguistic features of e-mails.

1.2 Difficulties in modeling

●In addition to the mail items stored in txt format from the datasets supported by the question, we also found a large number of sql database scripts are used for some old version of MySQL which are loaded incorrectly in the new version and can not be fully loaded into memory because the script itself is too large check in.

●What we can use to refine the author's linguistic style is only the e-mail content written by the author himself. However, most of the received mail contains the contents of original message, references from other articles, and useless head information such as Message-ID and date, which must be removed and extract all the effective contents from all the emails.

●Selecting appropriate identifiable linguistic features as model parameters.

●Using rational combination of the certain linguistic features and use specific analytical methods to achieve the recognition of language style.

II. The Description of the Problem

2.1Restatement of the problem

This issue requires us to identify authors by the linguistic features of the e-mail texts. To achieve the solution, we should extract identifiable features of linguistic style in a large number of words and translate them into data for analysis, and structure a model. When we need to identify the author of an e-mail, compare its linguistic style to the one of a certain known authors' e-mails to attribute the authorship.

2.2 Steps to solve the problem

1) Get enough texts of emails from different authors.

2) Preprocess the content of the email, exclude the invalid content, and obtain the effective content of the email body.

3) Select several appropriate recognizable linguistic features as model parameters.

4) Structure a model.

5) Test sensitivity of the model.

III. Models

3.1 Terms, Definitions and Symbols

signs	definitions
A	judgment matrix
w	the eigenvector corresponding to the largest eigenvalue
λ	the maximum eigenvalue
n_1	the number of the same words of 30 words which occurred most frequently taken from a test text and a training text
x_1	the model parameter corresponding to the high frequency words
n_2	the ratio of the comma and period
x_2	the model parameter corresponding to the long sentence usage
n_3	a parameter indicating the vocabulary of a text set
x_3	the model parameter corresponding to the vocabulary
n_4	a parameter indicating the adjective usage of a text set
x_4	the model parameter corresponding to the adjective usage
n_5	a parameter indicating the recipient appellation writing habits
x_5	the model parameter corresponding to the recipient appellation writing habits
D	the difference value between two texts

3.2 Assumptions

● E-mails sent by each mailbox are written by the same person. (the total number of senders is very close to the one of mailboxes)

● All e-mails are short texts with little difference in length.

3.3 Acquisition and preprocessing of the mails

The Enron Email Datasets includes 3404 emails in txt format, a tool for visualizing the relationship of sending and receiving and a large database script in sql format.We found that there was an error in the sql format database so that it could not be read normally, so we read the sql database script by stream reading, then fixed the bug and got the data.We exported the text of the e-mails excluding invalid messages such as Message-ID, date, etc. to txt format, and finally got about 250,000 e-mails.

As mentioned before, what we need is only the body part written by the author himself and other key contents, but the e-mails we received in txt format contain historical correspondence, forwarded letters and other invalid contents. As shown below, only the contents in the red box are the valid section we need.

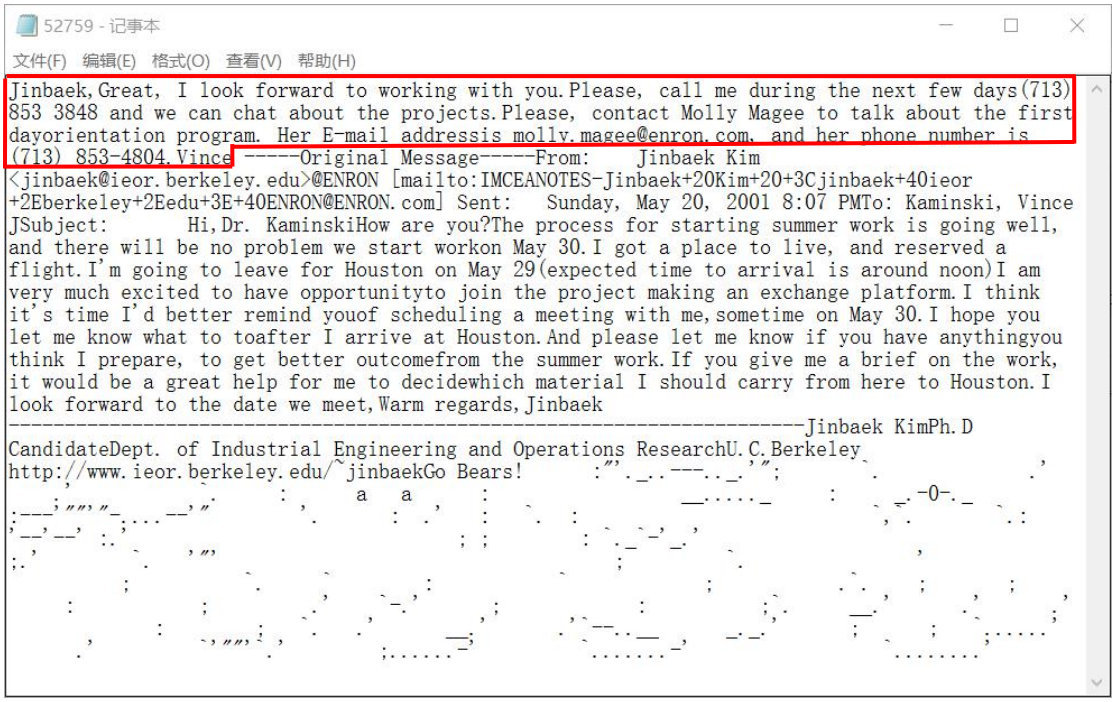

Figure 1 The original message obtained

By observing of a large number of e-mails we concluded the definition of invalid contents:

1.The text after the keyword "----- Original Message -----"

2.The text after the keyword "---------- Forwarded"

3.The text after two consecutive blank line

Based on the above rules, we removed all invalid content of the e-mails by C # programming, accessed to the valid contents.

Since most emails are short in content and few in words, analyzing each email separately can not effectively reflect the author's linguistic features. Therefore, all the

mails sent by the same mailbox which sent a large number of emails need to be merged and analyzed.

We classified all the emails according to mailboxes, and put the emails sent by the same mailboxes together.We find out the mailboxes that have sent more than 1,000 e-mails. 9/10 of all the emails in each mailbox is defined as a training set and 1/10 is defined as a test set.When calculating the model parameter values, we merged the valid contents of the training sets as the training texts, and merged the valid contents of the test sets as the test texts.

3.4 Identifiable linguistic features

We chose five recognizable linguistic features as follow: high frequency words, long sentence usage, vocabulary, adjective usage, and recipient appellation writing habits, as model parameters.

1) High frequency words

Every author has a habitual usage of some words. There is a difference between person and person. Therefore, high frequency words can be used as a recognizable language feature.

2) Long sentence usage

Some authors have an idiomatic habit to use more long sentences than short sentences when writing a message, so the long sentence usage can be used as a recognizable language feature

3) Vocabulary

Different authors have different vocabulary, so it can be used as a recognizable language feature.

4) Adjective usage

Some authors tend to use more adjectives to make the text vivid, while others do not. Therefore, the frequency of use of adjectives can be used as a recognizable language feature.

5) Recipient appellation writing habits

Different authors tend to make a big difference on the title to a recipient at the beginning of the email so the writing habit of the title can be used as a recognizable language feature.

3.5 Using Analytic Hierarchy Process to determine the weight of each parameter

Given that each parameter has a different impact on the linguistic style, we use Analytic Hierarchy Process (AHP) to determine the weight of each parameter's impact on the linguistic style. The analysis of the problem is divided into three levels. The top level is the target level which is to distinguish the linguistic style; the lowest level is the program level including each training text or test text which is marked as $P_1, P_2, P_3, \ldots, P_n$. The middle level is the criterion level which consists of the five parameters: high frequency words, long sentence usage, vocabulary, adjective usage, and recipient appellation writing habits.

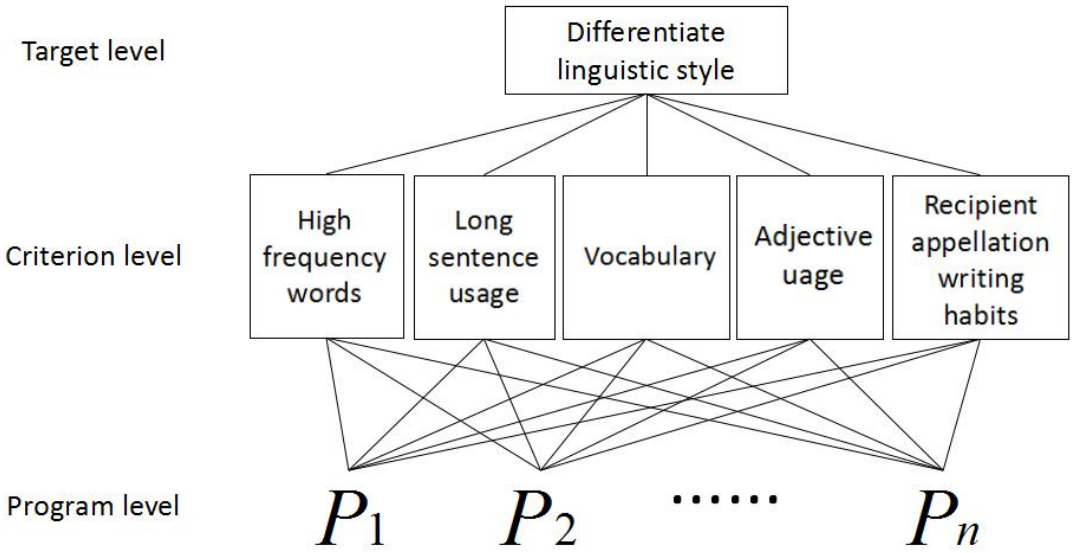

Figure 2 Hierarchy of linguistic style

When we consider the impact of different criteria C_i, C_j on the target level, we use the 1-9 scale which means the elements of the judgment matrix a_{ij} are in the range of 1, 2, ... , 9 and their reciprocal $1, \frac{1}{2}, \ldots, \frac{1}{9}$. Their meanings are as follows:

Table 1 meaning of 1-9 scale a_{ij}

scale a_{ij}	**meanings**
1	C_i and C_j have the same effect
3	the effect of C_i is slightly stronger than the effect of C_j
5	the effect of C_i is stronger than the effect of C_j
7	the effect of C_i is obviously stronger than the effect of C_j
9	the effect of C_i is absolutely stronger than the effect of C_j
2,4,6,8	The ratio of the effects of C_i and C_j is between the two adjacent levels
1,1/2, ... ,1/9	The ratio of the effects of C_j and C_i is the reciprocal of a_{ij} above

From this we get the following judgment matrix:

$$A=\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{4} & \frac{1}{5} \\ 2 & 1 & 2 & \frac{1}{2} & \frac{1}{3} \\ 2 & \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{4} \\ 4 & 2 & 2 & 1 & \frac{1}{2} \\ 5 & 3 & 4 & 2 & 1 \end{pmatrix}$$

We obtain the eigenvector corresponding to the largest eigenvalue from the judgment matrix:

$$w=（0.1285,0.2870,0.2051,0.4667,0.8008）$$

In this case, the maximum eigenvalue of the matrix is $\lambda=5.0679$, consistency index $CI=\frac{\lambda-n}{n-1}=0.01698$, random consistency indicator $CR=\frac{CI}{RI}=0.01516<0.1$. The result has gone through the consistency test which shows that the judgment matrix has a satisfactory consistency. After normalization, it can be used as the weight vector of the above indexes. The weight of each model parameter is shown in the following table:

Table 2 The weight of each model parameter

model parameters	high frequency words	long sentence usage	word richness	adjective usage	appellation writing habits
weight	0. 06806	0. 1520	0. 1086	0. 2472	0. 4241

3.6 Mail fingerprint model

3.6.1 *Calculation method of model parameters*

The definition and calculation method of the model parameters corresponding to each language feature are as follows:

1) High frequency words

30 words which occurred most frequently were taken from a test text and a training text, respectively, and the number of the same words of them were defined as n_1 . The model parameter corresponding to the high frequency words is defined as

$$x_1 = \left(\frac{30 - n_1}{30}\right)^2 \tag{1}$$

The smaller the value is, the closer the two texts are to this indicator.

2)Long sentence usage

We indicate this linguistic feature by the relationship between the number of commas and periods.

By observing a large number of text mails sent by the same mailbox, we found that if set the number of periods in the same text as an independent variable and the number of commas as a dependent variable, then draw the scatter plot, the shape of the figure is always as follows:

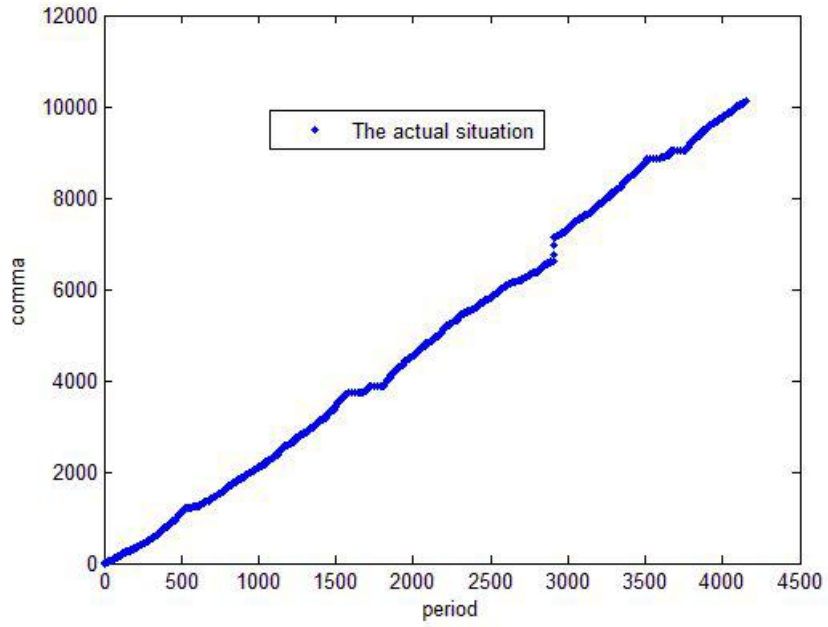

Figure 3 the relationship between the number of commas and periods

It is almost linear between the two variables, the fitted line can be seen as passing through the origin. Therefore, we use the ratio of the number of commas and periods to indicate this linguistic feature.

The greater the ratio of the comma to the number of periods is, the more long sentences is in the text. Set the ratio of the comma and period in the test text is $n_{2(te)}$, the ratio of comma to period in training text is $n_{2(tr)}$.This model parameter can be set to:

$$x_2 = (n_{2(te)} - n_{2(tr)})^2 \tag{2}$$

3)Vocabulary

By observing a large number of text mails sent by the same mailbox, we found that if set *token* (the number of all the words) in the same text as an independent variable

and *type* (the number of all different words, excluding reused words, is not case sensitive) as a dependent variable, then draw the scatter plot, the shape of the figure is always as follows:

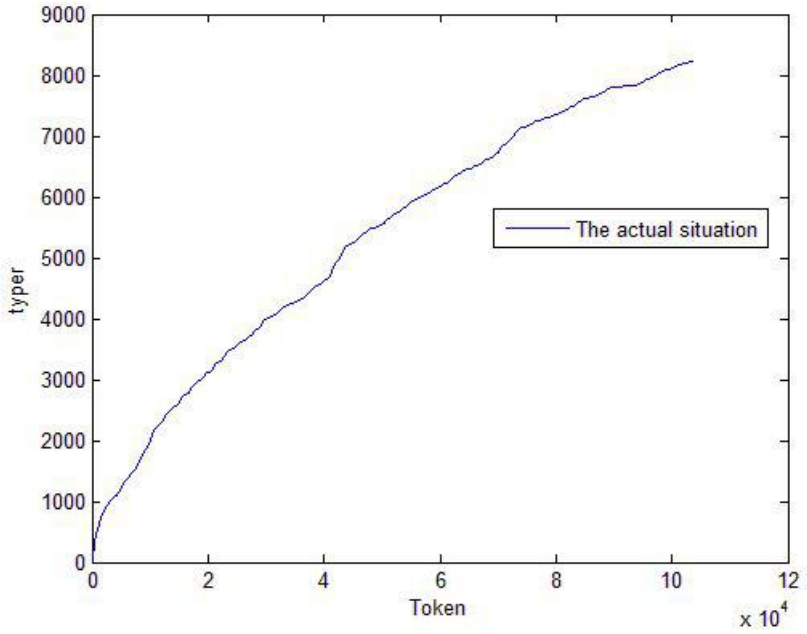

Figure 4 the relationship between type and token

Fitting equations have the form of $y = n_3 x^2$. The coefficient n_3 can be used as a parameter indicating the vocabulary of a text set. Set the coefficient of test set as $n_{3(te)}$, the coefficient of training set as $n_{3(tr)}$, and the model parameter corresponding to the vocabulary can be set as:

$$x_3 = (n_{3(te)} - n_{3(tr)})^2 \tag{3}$$

4)Adjective usage

We give an adjective affix library that we think is typical and use it to detect the number of adjectives that appear in the text.

Similar to the processing method of long sentence usage, set *token* (the number of all the words) in the same text as an independent variable and set it as x , and the total number of adjectives as a dependent variable and set it as y , the fitting equations have the form of $y = n_4 x$ .The coefficient n_4 can be used as a parameter indicating the adjective usage of a text set. Set the coefficient of test set as $n_{4(te)}$, the

coefficient of training set as $a_{4(tr)}$, and the model parameter corresponding to the adjective usage can be set as:

$$x_4 = (n_{4(te)} - n_{4(tr)})^2 \tag{4}$$

5)Recipient appellation writing habits

About the recipient appellation writing habits, we observed the characteristics of a large number of e-mails and formulated following criteria:

①The first char is 'I'

②The last char in the first line is ':'

③The first 20 char contains "Hi"

④The first 20 char contains "Dear"

⑤The first 20 char contains '-'

⑥The ratio of the number of uppercase letters and lowercase letters in the first 20 char is larger than 0.8

⑦The first 20 char (a letter or space or punctuation) contains ','

⑧The number of char in the first line is larger than 20

⑨The first 20 char contains ' '(space)

⑩The second line is blank line

We use a numerical value m in the range of [0,10] to indicate how close the recipient appellation writing habits is between two texts. If there is a rule between two texts that matches or does not match at the same time, we add 1 to the n_5 . Set

$$x_5=\left(\frac{10-n_5}{10}\right)^2 \tag{5}$$

as a model parameter corresponding to this linguistic feature. The smaller this value is, the closer the two texts are in the recipient appellation writing habits.

Supposing the weight of each model parameter obtained before is respectively $a,b,c,d,e,$ then we can finally get the difference between two texts on their linguistic style after considering the impact of every parameter. The magnitudes of x_1,x_2,x_3,x_4,x_5 are different, we multiply them by some coefficient to make them of the same order of magnitude. Define D as the difference value between two texts:

$$D=ax_1+bx_2+\frac{cx_3}{100}+10000dx_4+\frac{ex_5}{100} \tag{6}$$

Comparing the test texts we need to identify with the training texts of various different known authors, the author who corresponds to the smallest value of D is the most likely one of the test text.

3.6.2 *Solution and Result*

We selected five mailboxes which sent more than 1000 mails from Enron Email Datasets randomly to verify our model. The five mailboxes are arsystem@mailman.enron.com, carol.clair@enron.com, chris.dorland@enron.com, d..steffes@enron.com, chris.germany@enron.com. Merged text from the first four mailboxes are defined as $\alpha_1,\alpha_2,\alpha_3,\alpha_4$, 9/10 mails of the fifth mailboxes are defined as α_5 , the other 1/9 are defined as α_6 . $\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5$ are training texts, α_6 is test text.

Based on the actual contents of the six texts and our definition, we obtain the following results:

Table 3 The results

	α_1	α_2	α_3	α_4	α_5	α_6
n_1	23	22	25	26	28	30
n_2	0.001952	0.243097	0.204158	0.316510	0.410520	0.479957
n_3	0.526374	17.90920	22.77336	22.61560	25.57945	23.62603
n_4	0.040297	0.015262	0.010643	0.016793	0.013027	0.014186
n_5	7	7	7	8	10	10

We can get the relevant data as follow by bringing the above calculation results in formula（1）to formula (6).

Table 4 The calculation results

	D	x_1*a	x_2*b	$x_3*c/100$	$x_4*d*10000$	$x_5*e/100$
α_1-α_6	2.347449	0.043558	0.034730	0.579484	1.685436	0.004241
α_2-α_6	0.092614	0.007562	0.008528	0.035493	0.002862	0.038169
α_3-α_6	0.084265	0.002722	0.011562	0.000790	0.031022	0.038169
α_4-α_6	0.042648	0.003705	0.004061	0.001109	0.016809	0.016964
α_5-α_6	**0.008499**	0.000302	0.000733	0.004144	0.003320	0.000000

From the minimum value of D :0.008499, we can see that the difference between α_6 and α_5 is the smallest which means the test text is most likely from the mailbox chris.germany@enron.com. The result is realistic so the model is valid.

3.6.3 *Visualization of difference*

Mail FingerPrinter is our self-developed visual tool for analyzing the different linguistic style between different authors. We set the above five parameters as a specific author's mail fingerprint. According to the method to calculate the value of D we obtain the similar method to compare two mail fingerprints. We visualize mail fingerprints as two pentagons. If the degree of overlap of the pentagons is greater, the difference in the linguistic style of the text is smaller.

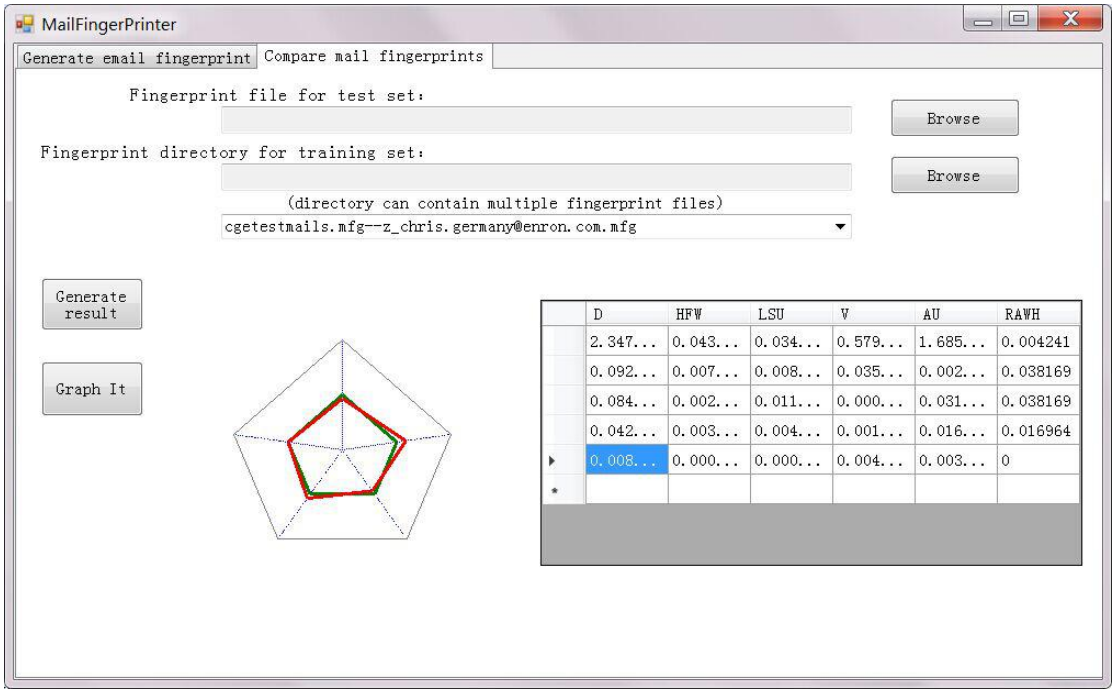

Figure 5 degree of mail fingerprint matching between α_5 and α_6

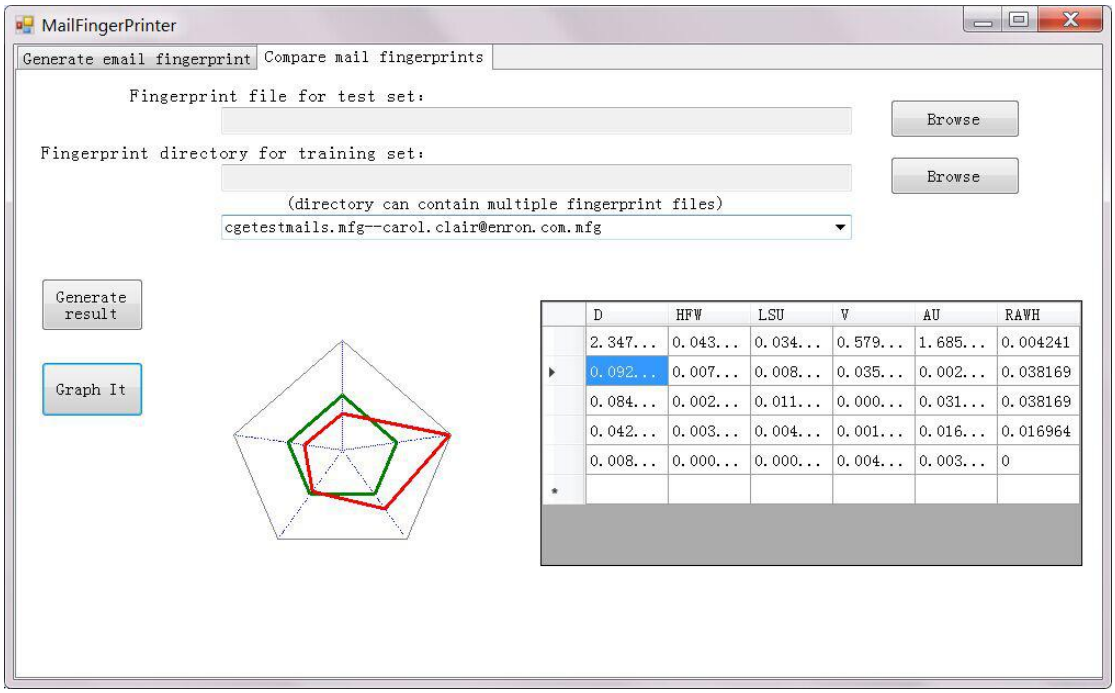

Figure 5 degree of mail fingerprint matching between α_2 and α_6

3.6.4 *Analysis of the Result*

In order to test whether the five parameters chosen is strongly representative, we consider reducing the number of training samples to see if they still reflect the differences in linguistic styles of different authors so as to attribute the authorship.

We used to randomly extract 5 mailboxes from Enron Email Datasets as a set of experiments which contain more than 1,000 emails. We also randomly selected multiple mailboxes for several experiments, all of which attribute the authorship, so we conduct the conclusion that the model is reliable at the level of over 1000 training emails.

After that, we randomly select multiple mailboxes which contains 500-1000 letters as sets of experiments, and the result can still achieve the authorship of the training text, but the difference between the test text and some other training texts is reduced.

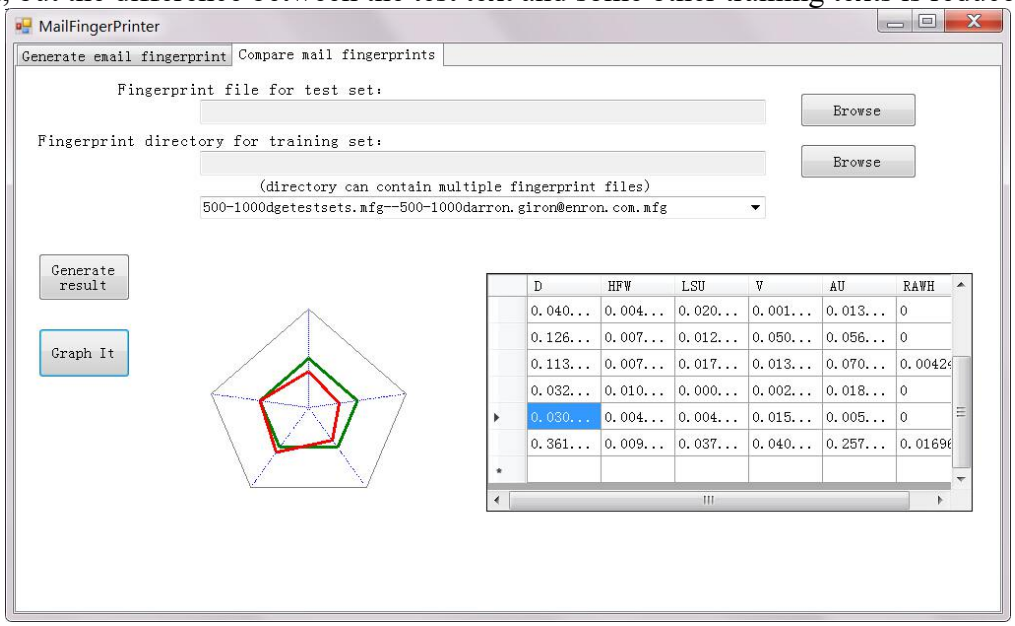

Figure 6 from the same author

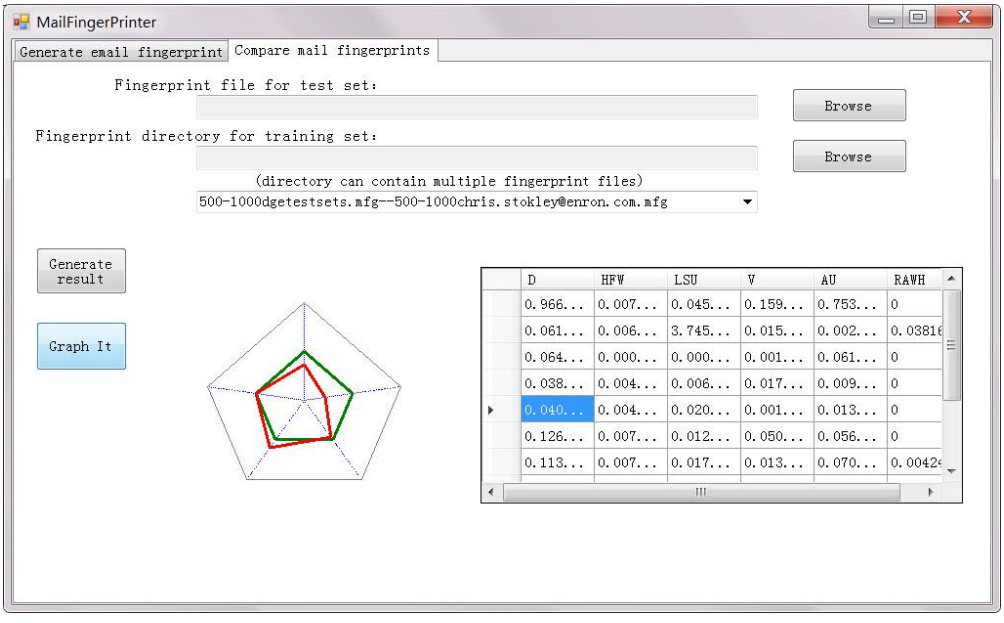

Figure 7 from different authors but the difference is small

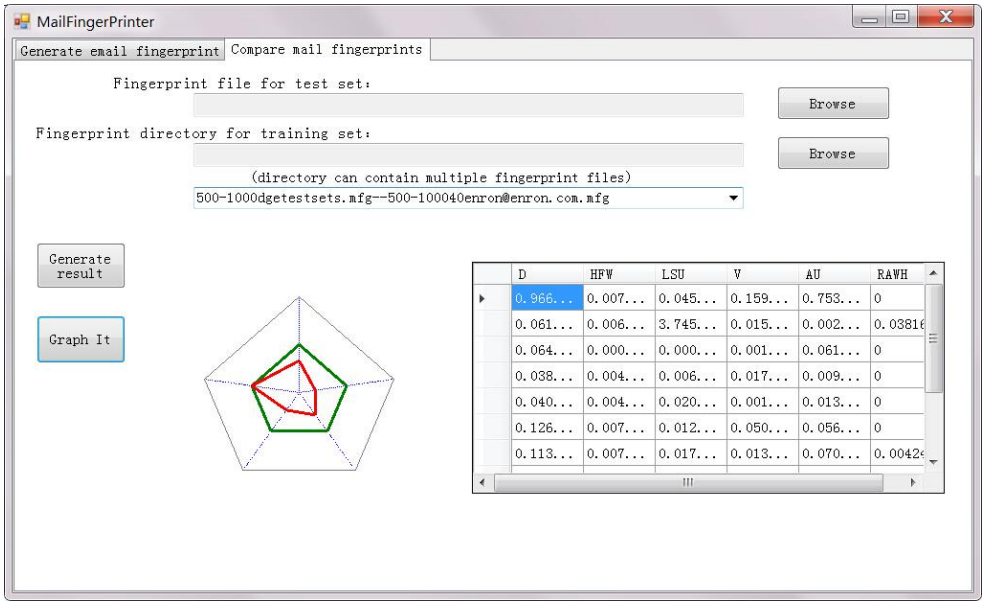

Figure 8 from different authors and the difference is big

So we conduct the conclusion that our model is basically accurate if the training text contains more than 500 mails.

3.6.5 *Strength and Weakness*

● Strength:

1.Parameters involved are comprehensive, so the model has strong representation.
2.The authorship attribution result is quite satisfactory when the sample size is sufficient.
3.Realized visualization of differences in linguistic features.

● Weakness:

1.The ranking of high frequency words wasn't taken into consideration.
2.The recipient appellation writing habits we considered is not sufficient.
3.The quantity of adjectives we considered isn't large enough.
4.The choice of weights is not objective enough.

IV. Future Work

4.1 The improvement of model

For the current model parameters, we put forward some suggestions for improvement:

●Take the ranking of high frequency words into our model parameters.
●Expand the initial writing habits and inscription writing habits of letters.

●Extend the typical prefix suffix dictionary for adjectives and extend the noun verb affix dictionary.
●The weight of different parameters should be determined more objectively.

4.2 The promotion of data sources

The sources of information can be expanded to chat log text, writing articles and more.

4.3 Application of the model to other fields

The idea of finding features and then comparing the differences in features between different objects used in this model can be applied to other areas such as voiceprint recognition.

V. References

[1] 姜启源, 谢金星, 叶俊. 数学模型[M]. 北京:高等教育出版社, 2011. 249-257
[2] Kimmo, Kettunen. Can Type-Token Ratio be Used to Show Morphological Complexity of Languages?[J]. Journal of Quantitative Linguistics, 2014, 21(3): 223-245
[3] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京:国防工业出版社, 2016. 358-360

VI. Appendix

Programs and codes

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.IO;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MailFingerPrinter
{
    public partial class FingerPrinterForm : Form
```

```
    {
        static string MailsPath = "";
        static string TrainSetPath = "";
        static string FingerPrintFilePath = "";
        static string FingerPrintTo = "";
        static List<FingerPrintResult> FingerPrintResultList = new
List<FingerPrintResult>();
        public FingerPrinterForm()
        {
            InitializeComponent();
        }
        #region tab_1 给 tab_1 的所有按钮委托写上签名
        private void bt_mailsfrom_Click(object sender, EventArgs e)
        {
            if (WhereMailIn.ShowDialog() == DialogResult.OK)
            {
                MailsPath = WhereMailIn.SelectedPath;
                tb_mailfrom.Text = MailsPath;
            }
        }
        private void bt_tomfg_Click(object sender, EventArgs e)
        {
            if (WhereToCreateMFG.ShowDialog() == DialogResult.OK)
            {
                FingerPrintTo = WhereToCreateMFG.SelectedPath;
                tb_mfgto.Text = FingerPrintTo;
            }
        }
        private void bt_create_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(MailsPath) ||
string.IsNullOrEmpty(FingerPrintTo))
            {
                MessageBox.Show("Please input the directory");
            }
            else
            {
                if (!Directory.Exists(MailsPath))
                {
                    MessageBox.Show($"The following directory does not exist,
please select once again：\r\n{ MailsPath }");
                }
                else if (!Directory.Exists(FingerPrintTo))
                {
```

```
                    MessageBox.Show($"The following directory does not exist,
please select once again：\r\n{ FingerPrintTo }");
                }
                else
                {
                    FingerPrint                    fingerprint                    =
FingerPrint.FromSourceFile(MailsPath, cb_hashead.Checked);
                    fingerprint.SaveTo(FingerPrintTo);
                    MessageBox.Show("Generate successfully！");
                }
            }
        }
        #endregion
        #region tab_2 给 tab_2 的所有按钮委托写上签名
        private void bt_testfingerprinterfrom_Click(object sender, EventArgs e)
        {
            if (WhichMFG.ShowDialog() == DialogResult.OK)
            {
                string                              ismfg                          =
WhichMFG.FileName.Substring(WhichMFG.FileName.Length - 4);
                if (ismfg != ".mfg")
                {
                    MessageBox.Show("please select a fingerprint file（.mfg）");
                }
                else
                {
                    FingerPrintFilePath = WhichMFG.FileName;
                    tb_mfgfrom.Text = FingerPrintFilePath;
                }
            }
        }
        private void bt_trainfrom_Click(object sender, EventArgs e)
        {
            if (WhereTrainMFGIn.ShowDialog() == DialogResult.OK)
            {
                var mfgs = Directory.GetFiles(WhereTrainMFGIn.SelectedPath,
"*.mfg");
                if (mfgs.Length == 0)
                {
                    MessageBox.Show("There is no fingerprint file（.mfg） in
this directory, please select again");
                }
                else
                {
```

```
                    TrainSetPath = WhereTrainMFGIn.SelectedPath;
                    tb_trainfrom.Text = TrainSetPath;
                }
            }
        }
        private void bt_getresult_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(TrainSetPath) || string.IsNullOrEmpty(FingerPrintFilePath))
            {
                MessageBox.Show("Please input the directory and file");
            }
            else
            {
                if (!Directory.Exists(TrainSetPath))
                {
                    MessageBox.Show($"The following directory does not exist, please select once again：\r\n{ TrainSetPath }");
                }
                else if (!File.Exists(FingerPrintFilePath))
                {
                    MessageBox.Show($"The following file does not exist, please select once again：\r\n{ FingerPrintFilePath }");
                }
                else
                {
                    string thisMailsPath = MailsPath;
                    string thisTrainSetPath = TrainSetPath;
                    string thisFingerPrintFilePath = FingerPrintFilePath;
                    string thisFingerPrintTo = FingerPrintTo;
                    //还原初始状态
                    Clear();
                    string[] trains = Directory.GetFiles(thisTrainSetPath, "*.mfg");
                    var trainfp = new List<FingerPrint>();
                    FingerPrint testfp = FingerPrint.FromSourceMFGFile(thisFingerPrintFilePath);
                    foreach (var train in trains)
                    {
                        trainfp.Add(FingerPrint.FromSourceMFGFile(train));
                    }
                    foreach (var fp in trainfp)
                    {
                        //将数据写入 DataGridView 中
```

```
                                var resultnow = new FingerPrintResult(testfp, fp);
                                FingerPrintResultList.Add(resultnow);
                                int indexnow = dGV_resulttable.Rows.Add();
                                dGV_resulttable.Rows[indexnow].Cells[0].Value =
resultnow.VarianceResult;
                                dGV_resulttable.Rows[indexnow].Cells[1].Value =
resultnow.HighRateWordsResult;
                                dGV_resulttable.Rows[indexnow].Cells[2].Value =
resultnow.LongRichnessResult;
                                dGV_resulttable.Rows[indexnow].Cells[3].Value =
resultnow.WordRichnessResult;
                                dGV_resulttable.Rows[indexnow].Cells[4].Value =
resultnow.AdjRateResult;
                                dGV_resulttable.Rows[indexnow].Cells[5].Value =
resultnow.FirstLineCodeResult;
                                cb_results.Items.Add(resultnow);
                            }
                            MessageBox.Show("completed. please check the results！");
                        }
                    }
                }
                #endregion
                /// <summary>
                ///  任何时候执行这个方法，即还原程序到初始状态
                /// </summary>
                void Clear()
                {
                    cb_results.Items.Clear();
                    DataTable dt = (DataTable)dGV_resulttable.DataSource;
                    if (dt != null)
                    {
                        dt.Rows.Clear();
                        dGV_resulttable.DataSource = dt;
                    }
                    MailsPath = "";
                    TrainSetPath = "";
                    FingerPrintFilePath = "";
                    FingerPrintTo = "";
                    FingerPrintResultList = new List<FingerPrintResult>();
                    tb_mailfrom.Text = "";
                    tb_mfgfrom.Text = "";
                    tb_mfgto.Text = "";
                    tb_trainfrom.Text = "";
                }
```

```
        /// <summary>
        ///  画五边形的事件
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void bt_painter_Click(object sender, EventArgs e)
        {
            int index = cb_results.SelectedIndex;
            if (index < 0)
            {
                MessageBox.Show("Select a result in the combobox!");
                return;
            }
            FingerPrintResult nowresult = FingerPrintResultList[index];
            FingerPrint a = nowresult.a;
            FingerPrint b = nowresult.b;
            Bitmap resultimg = new Bitmap(pB_resultimgbox.Width,
pB_resultimgbox.Height);
            Graphics g = Graphics.FromImage(resultimg);
            g.Clear(Color.White);
            Brush b_blue = new SolidBrush(Color.Blue);
            Pen p_black = new Pen(Brushes.Black);
            Pen p_blue = new Pen(Brushes.Blue);
            Pen p_red = new Pen(Brushes.Red, 3);
            Pen p_green = new Pen(Brushes.Green, 3);
            Pen p_dot_blue = new Pen(Brushes.Blue);
            Pen p_whitesmoke = new Pen(Brushes.Gray);
            p_dot_blue.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
            #region 计算得到五边形的中心与五角
            PointF center = new PointF(175, 175);
            PointF pointA = new PointF(175, 75);
            PointF pointB = new PointF(274, 161);
            PointF pointC = new PointF(233.8f, 256);
            PointF pointD = new PointF(116.2f, 256);
            PointF pointE = new PointF(76, 161);
            #endregion
            #region 画五边形
            //        g.FillEllipse(b_blue, 175, 175, 5, 5);
            DrawPentagon(g, p_whitesmoke, pointA, pointB, pointC, pointD,
pointE);
            g.DrawLine(p_dot_blue, center, pointA);
            g.DrawLine(p_dot_blue, center, pointB);
            g.DrawLine(p_dot_blue, center, pointC);
            g.DrawLine(p_dot_blue, center, pointD);
```

```
                g.DrawLine(p_dot_blue, center, pointE);
                #endregion
                #region 画特征点所属的五边形
                PointF aA = GetPosition(center, pointA, 0.5);
                PointF bA = GetPosition(center, pointA, nowresult.HowManyEqualInTop30 * 0.5 / 30);
                PointF aB = GetPosition(center, pointB, 0.5);
                PointF bB = GetPosition(center, pointB, a.LongRichness / b.LongRichness * 0.5);
                PointF aC = GetPosition(center, pointC, 0.5);
                PointF bC = GetPosition(center, pointC, a.WordRichness / b.WordRichness * 0.5);
                PointF aD = GetPosition(center, pointD, 0.5);
                PointF bD = GetPosition(center, pointD, a.AdjRate / b.AdjRate * 0.5);
                PointF aE = GetPosition(center, pointE, 0.5);
                PointF bE = GetPosition(center, pointE, (10 - FirstLineCode.Judge(a.FirstLineCode, b.FirstLineCode)) * 0.05);
                DrawPentagon(g, p_green, aA, aB, aC, aD, aE);
                DrawPentagon(g, p_red, bA, bB, bC, bD, bE);
                #endregion
                //将 picturebox 的 image 属性指向内存中的 bitmap
                pB_resultimgbox.Image = resultimg;
            }
            /// <summary>
            /// 用给定的五点按顺时针顺序画五边形
            /// </summary>
            /// <param name="g"></param>
            /// <param name="p">画笔</param>
            /// <param name="pointA">五边形顶角</param>
            /// <param name="pointB">五边形右上角</param>
            /// <param name="pointC">五边形右下角</param>
            /// <param name="pointD">五边形左下角</param>
            /// <param name="pointE">五边形左上角</param>
            private void DrawPentagon(Graphics g, Pen p, PointF pointA, PointF pointB, PointF pointC, PointF pointD, PointF pointE)
            {
                g.DrawLine(p, pointA, pointB);
                g.DrawLine(p, pointB, pointC);
                g.DrawLine(p, pointC, pointD);
                g.DrawLine(p, pointD, pointE);
                g.DrawLine(p, pointE, pointA);
            }
            /// <summary>
            /// 用线性插值法得到 value 值对应的点的具体位置
```

```
        /// </summary>
        /// <param name="begin"></param>
        /// <param name="end"></param>
        /// <param name="value"></param>
        /// <returns></returns>
        private PointF GetPosition(PointF begin, PointF end, double value)
        {
            float x = (end.X - begin.X) * (float)value + begin.X;
            float y = (end.Y - begin.Y) * (float)value + begin.Y;
            return new PointF(x, y);
        }

    }
    /// <summary>
    /// 封装了进行 io 文件操作的方法
    /// </summary>
    public static class IOMethod
    {
        /// <summary>
        /// 获得给定目录中所有邮件的正文（不含引用与回复文本）
        /// </summary>
        /// <param name="path">给定目录</param>
        /// <param name="hashead">邮件是否含头信息</param>
        /// <returns></returns>
        public static string GetBodies(string path, bool hashead = true)
        {
            string result = "";
            var filenames = Directory.GetFiles(path);
            foreach (var file in filenames)
            {
                result += GetBody(file, false);
                result += " ";
            }
            return result;
        }
        /// <summary>
        /// 获得给定文件中的邮件正文（不含引用与回复文本）
        /// </summary>
        /// <param name="path">给定文件路径</param>
        /// <param name="hashead">邮件是否含头信息</param>
        /// <returns></returns>
        public static string GetBody(string path, bool hashead = true)
        {
            string all = "";
```

```
        using (StreamReader sr = new StreamReader(path))
        {
            string now = "";
            int rowlimit = 2;
            int emptyrow = 0;
            if (hashead)
            {
                Regex headall = new Regex("X-FileName:");
                while ((now = sr.ReadLine()) != null)
                {
                    if (headall.IsMatch(now))
                    {
                        break;
                    }
                }
            }
            Regex tailall2 = new Regex("--- Forwarded");
            Regex tailall = new Regex("-Original Message-");
            while ((now = sr.ReadLine()) != null)
            {
                if (Regex.IsMatch(now, "^\\s+$") || now == "")
                {
                    emptyrow++;
                    if (emptyrow == rowlimit)
                    {
                        break;
                    }
                }
                else
                {
                    emptyrow = 0;
                }
                if (tailall.IsMatch(now) || tailall2.IsMatch(now))
                {
                    break;
                }
                all += now + "\r\n";
            }
        }
        return all;
    }
    /// <summary>
    /// 获得文件中的单词总数，并直接重命名文件，将其写在文件名前
    /// </summary>
```

```
        /// <param name="file">文件路径</param>
        public static void ShowTheWordsNumInFile(string file)
        {
            string body = GetBody(file);
            string[] words = GetWords(body);
            int totalword = words.Length;

            string newfilename = GetPathToFile(file) + "/(" + totalword + ")" +
GetFileName(file);
            File.Move(file, newfilename);
        }
        /// <summary>
        /// 获取文件或目录最后一个'\\'后的字符（一般为文件名或目录名）
        /// </summary>
        /// <param name="filepath">文件或目录路径</param>
        /// <returns></returns>
        public static string GetFileName(string filepath)
        {
            var namearray = filepath.Split('\\');
            return namearray[namearray.Length - 1];
        }
        /// <summary>
        /// 获取一个文件所属目录的路径
        /// </summary>
        /// <param name="file">文件路径</param>
        /// <returns></returns>
        public static string GetPathToFile(string file)
        {
            FileInfo fi = new FileInfo(file);
            return fi.Directory.ToString();
        }
        /// <summary>
        /// 获取邮件正文中所有的单词，以字符串数组形式输出
        /// </summary>
        /// <param name="body">邮件正文</param>
        /// <returns></returns>
        public static string[] GetWords(string body)
        {
            Regex allregex = new Regex("[0-9\"-\\?,\r\n:=\\.]+");
            body = allregex.Replace(body, " ");
            Regex kongbai = new Regex("\\s+");
            body = kongbai.Replace(body, " ");
            string[] allarray = body.Split(' ');
            return allarray;
```

```
        }
        /// <summary>
        /// 获取邮件正文中所有的单词，以 List 泛型形式输出，泛型值为
EWord，具体说明见 EWord 类
        /// </summary>
        /// <param name="body">邮件正文</param>
        /// <returns></returns>
        public static List<EWord> GetWordsList(string body)
        {
            #region 统计重复单词数前先进行一个排序的预处理，从而减小复
杂度
            string[] words = GetWords(body);
            Array.Sort(words);
            int len = words.Length;
            int first = 0; int second = 0;
            List<EWord> resultlist = new List<EWord>();
            for (; second < len; second++)
            {
                if (words[first] != words[second])
                {
                    resultlist.Add(new EWord(second - first, words[first]));
                    first = second;
                }
            }
            #endregion
            //排序，传入的是一个兰姆达表达式，表示按 EWord 中 times 属性
的倒序排列
            resultlist.Sort((a, b) =>
            {
                return -1 * a.times.CompareTo(b.times);
            });
            return resultlist;
        }
        /// <summary>
        /// 获得目录中的文件总数，并直接重命名目录，将其写在目录名前
        /// </summary>
        /// <param name="path"></param>
        public static void ShowTheFilesNumInPath(string path)
        {
            var directory = Directory.GetDirectories(path);
            foreach (var item in directory)
            {
                var files = Directory.GetFiles(item, "*.txt");
                int total = files.Length;
```

```
                string result = "";
                for (int i = item.Length - 1; i >= 0; i--)
                {
                    if (item[i] == '\\')
                    {
                        result += item.Substring(0, i + 1);
                        result += total;
                        result += item.Substring(i + 1);
                        break;
                    }
                }
                Directory.Move(item, result);
            }
        }
        /// <summary>
        /// 将一个目录内的文件全部移到另一个目录中
        /// </summary>
        /// <param name="apath">源目录</param>
        /// <param name="bpath">终点目录</param>
        public static void MoveAllFileFromAtoB(string apath, string bpath)
        {
            var backfilespath = Directory.GetFiles(apath);
            foreach (var item in backfilespath)
            {
                File.Move(item, bpath + "\\" + GetFileName(item));
            }
        }
        /// <summary>
        /// 以流的形式将文本写入文件中，并释放流对象
        /// </summary>
        /// <param name="FilePath">目标路径</param>
        /// <param name="Content">待写文本</param>
        public static void SaveLog(string FilePath, string Content)
        {
            using (StreamWriter sw = new StreamWriter(FilePath))
            {
                sw.Write(Content);
                sw.Flush();
            }
        }
        /// <summary>
        /// 运用完全洗牌算法将源目录内，特定数量的文件移到终点目录内
        /// </summary>
        /// <param name="MotherSetPath">源目录</param>
```

```
        /// <param name="TestSetPath">终点目录（通常是测试目录）</param>
        /// <param name="TestSetSize">移动数量（通常是测试集样本容量）</param>
        public static void MoveMotherSetToTestSet(string MotherSetPath, string TestSetPath, int TestSetSize)
        {
            //完全洗牌算法处理 allfilesname 数组，从而随机的抽出一定数量的样本
            Random m = new Random();
            var allfilesname = Directory.GetFiles(MotherSetPath);
            int minlimit = Math.Max(allfilesname.Length - 1 - TestSetSize, -1);
            for (int i = allfilesname.Length - 1; i > minlimit; i--)
            {
                int nowrandom = m.Next(0, i + 1);
                string temp = allfilesname[i];
                allfilesname[i] = allfilesname[nowrandom];
                allfilesname[nowrandom] = temp;
                File.Move(allfilesname[i], TestSetPath + "\\" + GetFileName(allfilesname[i]));
            }
        }
    }
    /// <summary>
    /// 封装测试时使用的代码，部分代码用于获取数据以供在 matlab 中进行拟合，正式项目中不使用
    /// </summary>
    public static class Lazy_MailHandle
    {
        #region getpoint for matlab painting
        public static string lazy_GetManyPointAboutWordAndAdj()
        {
            string body = IOMethod.GetBodies(@"C:\Users\Administrator\Desktop\five\spilt_five\chris.dorland@enron.com", false);
            var wordarray = IOMethod.GetWords(body);
            List<Point> result = new List<Point>();
            int adj = 0;
            for (int i = 0; i < wordarray.Length; i++)
            {
                if (AdjJudge.IsAdj(wordarray[i]))
                {
                    adj++;
                }
                if (i % 330 == 0)
```

```
                    result.Add(new Point(i, adj));
                Console.WriteLine(i);
            }
            string x = "x=["; string y = "y=[";
            foreach (var item in result)
            {
                x += item.x + ",";
                y += item.y + ",";
            }
            x += "];\r\n";
            y += "];";
            return x + y;
        }
        public static string lazy_GetManyPointAboutWordAndKinds()
        {
            string body =
IOMethod.GetBodies(@"C:\Users\Administrator\Desktop\five\spilt_five\carol.clair@
enron.comtestmails", false);
            var wordarray = IOMethod.GetWords(body);
            List<Point> result = new List<Point>();
            for (int i = 0; i < wordarray.Length; i += 330)
            {
                string[] fronti = new string[i];
                for (int k = 0; k < i; k++)
                {
                    fronti[k] = wordarray[k];
                }
                Array.Sort(fronti);
                int len = fronti.Length;
                int first = 0; int second = 0;
                List<EWord> resultlist = new List<EWord>();
                for (; second < len; second++)
                {
                    if (fronti[first] != fronti[second])
                    {
                        resultlist.Add(new EWord(second - first, fronti[first]));
                        first = second;
                    }
                }
                result.Add(new Point(i, resultlist.Count));
                Console.WriteLine(i);
            }
            string x = "x=["; string y = "y=[";
            foreach (var item in result)
```

```
            {
                x += item.x + ",";
                y += item.y + ",";
            }
            x += "];\r\n";
            y += "];";
            return x + y;
        }
        #endregion
        #region some method for test
        //高频词算法
        public static void lazy_CompareMostWords()
        {
            Console.WriteLine("tm 要多少份测试？");
            int nums = Convert.ToInt32(Console.ReadLine());

            string MotherPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\chris.dorland@enron.com";
            string aimPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\testmails";
            IOMethod.MoveMotherSetToTestSet(MotherPath, aimPath, nums);
            #region
            string allTrainText = IOMethod.GetBodies(MotherPath, false);
            var TrainTextList = IOMethod.GetWordsList(allTrainText);

            string TrainOutput = "";
            int TrainTotalLitter = 0;
            for (int i = 0; i < TrainTextList.Count; i++)
            {
                TrainTotalLitter += TrainTextList[i].times;
            }
            for (int i = 0; i < 30 && i < TrainTextList.Count; i++)
            {
                TrainOutput += TrainTextList[i].ToString(TrainTotalLitter);
            }

            string TestSetPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\testmails";
            string allTestText = IOMethod.GetBodies(TestSetPath, false);
            var TestTextList = IOMethod.GetWordsList(allTestText);
            string TestOutput = "";
            int TestTotalLitter = 0;
            for (int i = 0; i < TestTextList.Count; i++)
            {
```

```
                TestTotalLitter += TestTextList[i].times;
            }
            int howmanyhas = 0;
            for (int i = 0; i < 30 && i < TestTextList.Count; i++)
            {
                if (TestTextList[i].isExistIn(TrainTextList))
                {
                    howmanyhas++;
                    TestOutput += "存在";
                }
                TestOutput += TestTextList[i].ToString(TestTotalLitter);
            }
            TestOutput += $"测试集前 30 条一共有{ howmanyhas }条在训练集前 30 条中出现";

            //写 output

IOMethod.SaveLog($@"C:\Users\Administrator\Desktop\gaoping\test{  TestTotalLitter }.txt", TestOutput);

IOMethod.SaveLog($@"C:\Users\Administrator\Desktop\gaoping\train{  TrainTotalLitter }.txt", TrainOutput);
            Console.WriteLine($"测试集前 30 条一共有{ howmanyhas }条在训练集前 30 条中出现");
            #endregion
            //换回去
            IOMethod.MoveAllFileFromAtoB(TestSetPath, MotherPath);
        }

        public static void lazy_CompareKindOfWord()
        {
            Console.WriteLine("tm 要多少份测试？");
            int nums = Convert.ToInt32(Console.ReadLine());

            string MotherPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\chris.dorland@enron.com";
            string aimPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\testmails";
            IOMethod.MoveMotherSetToTestSet(MotherPath, aimPath, nums);
            #region
            string allTrainText = IOMethod.GetBodies(MotherPath, false);
            var TrainTextList = IOMethod.GetWordsList(allTrainText);

            string TrainOutput = "";
```

```
            int TrainTotalWords = 0;
            for (int i = 0; i < TrainTextList.Count; i++)
            {
                TrainTotalWords += TrainTextList[i].times;
                TrainOutput += TrainTextList[i];
            }

            string TestSetPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\testmails";
            string allTestText = IOMethod.GetBodies(TestSetPath, false);
            var TestTextList = IOMethod.GetWordsList(allTestText);
            string TestOutput = "";
            int TestTotalWords = 0;
            for (int i = 0; i < TestTextList.Count; i++)
            {
                TestTotalWords += TestTextList[i].times;
                TestOutput += TestTextList[i];
            }


            //写 output

IOMethod.SaveLog($@"C:\Users\Administrator\Desktop\gaoping\test{ TestTotalWo
rds }.txt", TestOutput);

IOMethod.SaveLog($@"C:\Users\Administrator\Desktop\gaoping\train{ TrainTotal
Words }.txt", TrainOutput);
            Console.WriteLine($"训练集有词:{TrainTotalWords},词种
类:{TrainTextList.Count},种类比总词:{((double)TrainTextList.Count) /
(Math.Sqrt((double)TrainTotalWords))}");
            Console.WriteLine($"测试集有词:{TestTotalWords},词种
类:{TestTextList.Count},种类比总词:{((double)TestTextList.Count) /
(Math.Sqrt((double)TestTotalWords))}");
            #endregion
            //换回去
            IOMethod.MoveAllFileFromAtoB(TestSetPath, MotherPath);
        }
        public static void lazy_CompareAdj()
        {
            Console.WriteLine("tm 要多少份测试？");
            int nums = Convert.ToInt32(Console.ReadLine());

            string MotherPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\carol.clair@enron.com";
```

```
            string aimPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\testmails";
            IOMethod.MoveMotherSetToTestSet(MotherPath, aimPath, nums);
            MotherPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\chris.dorland@enron.com";
            #region
            string allTrainText = IOMethod.GetBodies(MotherPath, false);
            var TrainTextList = IOMethod.GetWordsList(allTrainText);

            string TrainOutput = "";
            int TrainTotalWords = 0;
            int TrainAdjWords = 0;
            for (int i = 0; i < TrainTextList.Count; i++)
            {
                TrainTotalWords += TrainTextList[i].times;
                if (TrainTextList[i].isAdj)
                {
                    TrainAdjWords += TrainTextList[i].times;
                }
                TrainOutput += TrainTextList[i];
            }

            string TestSetPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\testmails";
            string allTestText = IOMethod.GetBodies(TestSetPath, false);
            var TestTextList = IOMethod.GetWordsList(allTestText);
            string TestOutput = "";
            int TestTotalWords = 0;
            int TestAdjWords = 0;
            for (int i = 0; i < TestTextList.Count; i++)
            {
                TestTotalWords += TestTextList[i].times;
                if (TestTextList[i].isAdj)
                {
                    TestAdjWords += TestTextList[i].times;
                }
                TestOutput += TestTextList[i];
            }

            //写 output

IOMethod.SaveLog($@"C:\Users\Administrator\Desktop\gaoping\test{  TestTotalWo
rds }.txt", TestOutput);
```

```
IOMethod.SaveLog($@"C:\Users\Administrator\Desktop\gaoping\train{ TrainTotal
Words }.txt", TrainOutput);
            Console.WriteLine($"训练集有词:{TrainTotalWords},其中形容词:{TrainAdjWords},形容词比总词:{((double)TrainAdjWords) / ((double)TrainTotalWords)}");
            Console.WriteLine($"测试集有词:{TestTotalWords},其中形容词:{TestAdjWords},形容词比总词:{((double)TestAdjWords) / ((double)TestTotalWords)}");
            #endregion
            //换回去
            MotherPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\carol.clair@enron.com";
            IOMethod.MoveAllFileFromAtoB(TestSetPath, MotherPath);
        }
        //句号逗号算法
        public static void lazy_CompareJuDot()
        {
            Console.WriteLine("tm 要多少份测试？");
            int nums = Convert.ToInt32(Console.ReadLine());

            string MotherPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\d..steffes@enron.com";
            string aimPath =
@"C:\Users\Administrator\Desktop\five\spilt_five\testmails";
            IOMethod.MoveMotherSetToTestSet(MotherPath, aimPath, nums);
            string allTrainText = IOMethod.GetBodies(MotherPath, false);
            int dot = 0; int ju = 0;
            for (int i = 0; i < allTrainText.Length; i++)
            {
                if (allTrainText[i] == ',')
                    dot++;
                else if (allTrainText[i] == '.')
                    ju++;
            }
            Console.WriteLine("训练集中：");
            Console.WriteLine("dot:" + dot);
            Console.WriteLine("ju:" + ju);
            Console.WriteLine("dot/ju:" + ((double)dot) / ((double)ju));
            string allTestText = IOMethod.GetBodies(aimPath, false);
            dot = 0; ju = 0;
            for (int i = 0; i < allTestText.Length; i++)
            {
                if (allTestText[i] == ',')
                    dot++;
```

```
                    else if (allTestText[i] == '.')
                        ju++;
                }
                Console.WriteLine("测试集中：");
                Console.WriteLine("dot:" + dot);
                Console.WriteLine("ju:" + ju);
                Console.WriteLine("dot/ju:" + ((double)dot) / ((double)ju));
                //换回去
                IOMethod.MoveAllFileFromAtoB(aimPath, MotherPath);
            }
            #endregion
        }
        /// <summary>
        /// 针对给定的邮件集合，能够生成特定的指纹类的实例，
        /// 同时可以用指纹类的实例比较多个邮件集合之间的相似程度，具体见 FingerPrintResult 类
        /// </summary>
        public class FingerPrint
        {
            /// <summary>
            /// 指纹文件中：
            /// 第一行：称谓书写习惯状态码
            /// 第二行：形容词使用频率
            /// 第三行：用词丰富性
            /// 第四行：长短句使用线段
            /// 后全部：高频词
            /// </summary>
            public string SourceMailName = null;
            public FirstLineCode FirstLineCode { get; }
            public double AdjRate { get; }
            public double WordRichness { get; }
            public double LongRichness { get; }
            public List<EWord> HighRateWords { get; }
            /// <summary>
            /// 私有方法，只能类内部调用，保证指纹类型的安全
            /// </summary>
            /// <param name="firstlinecode"></param>
            /// <param name="adjrate"></param>
            /// <param name="wordrichness"></param>
            /// <param name="longrichness"></param>
            /// <param name="highratewords"></param>
            private FingerPrint(FirstLineCode firstlinecode, double adjrate, double wordrichness, double longrichness, List<EWord> highratewords)
            {
```

```
        FirstLineCode = firstlinecode;
        AdjRate = adjrate;
        WordRichness = wordrichness;
        LongRichness = longrichness;
        HighRateWords = highratewords;
    }
    /// <summary>
    /// 以流的形式将实例存储到文件中，其中高频词只保留前 30 个
    /// </summary>
    /// <param name="savepath">目标路径</param>
    public void SaveTo(string savepath)
    {
        if (savepath[savepath.Length - 1] != '\\')
        {
            savepath += '\\';
        }
        string allwordsrichness = "";
        for (int i = 0; i < 30 && i < HighRateWords.Count; i++)
        {
            allwordsrichness += HighRateWords[i].ToString();
        }
        string output = FirstLineCode + "\r\n"
            + AdjRate + "\r\n"
            + WordRichness + "\r\n"
            + LongRichness + "\r\n"
            + allwordsrichness;
        string filename = IOMethod.GetFileName(SourceMailName);
        if (string.IsNullOrEmpty(filename))
        {
            filename = "unnamed.mfg";
        }
        else
        {
            filename += ".mfg";
        }
        IOMethod.SaveLog(savepath + filename, output);
    }
    /// <summary>
    /// 从邮件集合中实例化指纹对象
    /// </summary>
    /// <param name="sourcepath">包含邮件集的目录路径</param>
    /// <param name="has_head">邮件是否含有头信息</param>
    /// <returns></returns>
    public static FingerPrint FromSourceFile(string sourcepath, bool has_head)
```

```
        {
            //利用开头前两行得到二进制状态码，顺路得到全部 bodies
            var dirs = Directory.GetFiles(sourcepath);
            string nowbody = ""; string allText = "";
            List<FirstLineCode> allcode = new List<FirstLineCode>();
            foreach (var dir in dirs)
            {
                nowbody = IOMethod.GetBody(dir, has_head);
                FirstLineCode nowcode = new FirstLineCode(nowbody);
                allcode.Add(nowcode);
                allText += nowbody;
            }
            //遍历一遍对逗号和句号计数，因为在生成 List 后，标点信息会被略去
            int dot = 0; int ju = 0;
            for (int i = 0; i < allText.Length; i++)
            {
                if (allText[i] == ',')
                    dot++;
                else if (allText[i] == '.')
                    ju++;
            }
            //再遍历一遍对形容词和词种类计数
            var TextList = IOMethod.GetWordsList(allText);
            int TotalWords = 0; int AdjWords = 0;
            for (int i = 0; i < TextList.Count; i++)
            {
                TotalWords += TextList[i].times;
                if (TextList[i].isAdj)
                {
                    AdjWords += TextList[i].times;
                }
            }
            FirstLineCode thisflc = FirstLineCode.SumAll(allcode);
            double thisadjrate = (((double)AdjWords) / ((double)TotalWords));
            double thiswordrichness = (((double)TextList.Count) / (Math.Sqrt((double)TotalWords)));
            double thislongrichness = (((double)dot) / ((double)ju));
            List<EWord> thishighratewords = TextList;
            FingerPrint result = new FingerPrint(thisflc, thisadjrate, thiswordrichness, thislongrichness, thishighratewords);
            result.SourceMailName = sourcepath;
            return result;
        }
```

```
        /// <summary>
        /// 从指纹文件（.mfg）中实例化指纹对象
        /// 指纹文件中：
        /// 第一行：称谓书写习惯状态码
        /// 第二行：形容词使用频率
        /// 第三行：用词丰富性
        /// 第四行：长短句使用线段
        /// 后全部：高频词
        /// </summary>
        public static FingerPrint FromSourceMFGFile(string mfgpath)
        {
            //保证对象被释放
            using (StreamReader sr = new StreamReader(mfgpath))
            {
                int code = Convert.ToInt32(sr.ReadLine());
                FirstLineCode thisflc = new FirstLineCode(code);
                double thisadj = Convert.ToDouble(sr.ReadLine());
                double thiswordrichness = Convert.ToDouble(sr.ReadLine());
                double thislongrichness = Convert.ToDouble(sr.ReadLine());
                int top30limit = 30; string nowline = null;
                var thisewordlist = new List<EWord>();
                while (top30limit > 0 && !(string.IsNullOrEmpty((nowline =
sr.ReadLine()))))
                {
                    var ewordarray = nowline.Split(':');
                    EWord noweword = new
EWord(Convert.ToInt32(ewordarray[1]), ewordarray[0]);
                    thisewordlist.Add(noweword);
                    top30limit--;
                }
                FingerPrint result = new FingerPrint(thisflc, thisadj,
thiswordrichness, thislongrichness, thisewordlist);
                result.SourceMailName = mfgpath;
                return result;
            }
        }
    }
    /// <summary>
    /// 封装指纹实例间的比对方法，并储存比对结果（VarianceResult）
    /// </summary>
    public struct FingerPrintResult
    {
        public FingerPrint a { get; }
        public FingerPrint b { get; }
```

```
        public double HowManyEqualInTop30 { get; }
        public double FirstLineCodeResult { get; }
        public double AdjRateResult { get; }
        public double WordRichnessResult { get; }
        public double LongRichnessResult { get; }
        public double HighRateWordsResult { get; }
        //直接反应指纹实例 a 与 b 间的相似程度，越小越相似
        public double VarianceResult { get; }
        public string Name { get; }
        /// 指纹文件中：
        /// 第一行：称谓书写习惯状态码
        /// 第二行：形容词使用频率
        /// 第三行：用词丰富性
        /// 第四行：长短句使用线段
        /// 后全部：高频词
        /// </summary>
        public FingerPrintResult(FingerPrint a, FingerPrint b)
        {
            this.a = a;
            this.b = b;
            Name = IOMethod.GetFileName(a.SourceMailName) + "--" +
IOMethod.GetFileName(b.SourceMailName);

            FirstLineCodeResult = ((double)FirstLineCode.Judge(a.FirstLineCode,
b.FirstLineCode));
            FirstLineCodeResult = FirstLineCodeResult / 10;
            FirstLineCodeResult = FirstLineCodeResult * FirstLineCodeResult;
            FirstLineCodeResult = 0.4241 * FirstLineCodeResult;

            AdjRateResult = a.AdjRate - b.AdjRate;
            AdjRateResult = AdjRateResult * 100;
            AdjRateResult = AdjRateResult * AdjRateResult;
            AdjRateResult = 0.2472 * AdjRateResult;

            WordRichnessResult = a.WordRichness - b.WordRichness;
            WordRichnessResult = WordRichnessResult / 10;
            WordRichnessResult = WordRichnessResult * WordRichnessResult;
            WordRichnessResult = 0.1086 * WordRichnessResult;

            LongRichnessResult = a.LongRichness - b.LongRichness;
            LongRichnessResult = LongRichnessResult * LongRichnessResult;
            LongRichnessResult = 0.1520 * LongRichnessResult;

            double howmanyequal = 0;
```

```
                foreach (EWord item in a.HighRateWords)
                {
                    if (item.isExistIn(b.HighRateWords))
                    {
                        howmanyequal++;
                    }
                }
                HowManyEqualInTop30 = howmanyequal;
                HighRateWordsResult = (30 - howmanyequal) / 30;
                HighRateWordsResult = HighRateWordsResult * HighRateWordsResult;
                HighRateWordsResult = 0.06806 * HighRateWordsResult;

                VarianceResult = FirstLineCodeResult + AdjRateResult + WordRichnessResult + LongRichnessResult + HighRateWordsResult;
            }
            public override string ToString()
            {
                return Name;
            }
        }
        /// <summary>
        /// 封装了单词，出现频数，是否形容词的结构体
        /// </summary>
        public struct EWord
        {
            public int times;
            public string theword;
            //是否是形容词，在实例化时即判断
            public bool isAdj;
            public EWord(int thistimes, string thisword)
            {
                times = thistimes;
                theword = thisword;
                isAdj = AdjJudge.IsAdj(thisword);
            }
            /// <summary>
            /// 重写了 ToString 方法
            /// </summary>
            /// <returns></returns>
            public override string ToString()
            {
                return $"{theword}:{times}\r\n";
            }
```

```
        public string ToString(double total)
        {
            return $"{theword}:{((double)times) / total}\r\n";
        }
        //判断 this 是否存在于 EWord 集合 motherlist 中
        public bool isExistIn(List<EWord> motherlist)
        {
            int minlimit = Math.Min(30, motherlist.Count - 1);
            for (int i = 0; i <= minlimit; i++)
            {
                if (motherlist[i].theword.Equals(this.theword))
                    return true;
            }
            return false;
        }
    }
    /// <summary>
    ///  解析邮件开头两行模式的状态机
    /// </summary>
    public class FirstLineCode
    {
        /*
         * Code 码表示从第一个非空段开始，两段长度内，文章的状态
         * Code 码本身是一个 10bit 二进制数，默认是 0
         *  满足以下条件，对对应位加 1
         * 1.前 20 个 char 内有','(+1)
         * 2.前 20 个 char 内有' '(+2)
         * 3.前 20 个 char 内有"Hi"(+4)
         * 4.前 20 个 char 内有"Dear"(+8)
         * 5.前 20 个 char 内有'-'(+16)
         * 6.前 20 个 char 中大写字母总数比上小写字母总数>0.8(+32)
         * 7.第 1 个 char 是 I(+64)
         * 8.第 1 行 char 数大于 20(+128)
         * 9.第 1 行最后一个 char 是':'(+256)
         * 10.第 2 行为空(+512)
         */
        public int Code = 0;
        public const int bitsNum = 10;
        public FirstLineCode(string body)
        {
            Regex nextline = new Regex("\r\n");
            string[] lines = nextline.Split(body);
            int FirstNonNullLineIndex = 0;
            for (; FirstNonNullLineIndex < lines.Length;
```

```
FirstNonNullLineIndex++)
                {
                    if (!string.IsNullOrEmpty(lines[FirstNonNullLineIndex]))
                    {
                        break;
                    }
                    if (FirstNonNullLineIndex == lines.Length - 1)
                    {
                        return;
                    }
                }
                handlefirstline(lines[FirstNonNullLineIndex]);
                if (lines.Length <= FirstNonNullLineIndex + 1)
                {
                    handlefirstline(null);
                }
                else
                {
                    handlesecondline(lines[FirstNonNullLineIndex + 1]);
                }
            }
            public FirstLineCode(int code)
            {
                this.Code = code;
            }
            private void handlefirstline(string firstline)
            {
                #region 前期判断
                int len = Math.Min(20, firstline.Length);
                string FirstTwentiesChar = firstline.Substring(0, len);
                bool hasdot = false;
                bool hasspace = false;
                bool has_ = false;
                Regex regex_hasHi = new Regex("Hi");
                Regex regex_hasDear = new Regex("Dear");
                Regex regex_upperlitter = new Regex("[A-Z]");
                Regex regex_lowerlitter = new Regex("[a-z]");
                double upperlitter = 0, lowerlitter = 0;
                for (int i = 0; i < len; i++)
                {
                    char now = FirstTwentiesChar[i];
                    if (regex_lowerlitter.IsMatch(now.ToString()))
                    {
                        lowerlitter += 1.0;
```

```
                }
                else if (regex_upperlitter.IsMatch(now.ToString()))
                {
                    upperlitter += 1.0;
                }
                switch (now)
                {
                    case '.':
                        hasdot = true;
                        break;
                    case ' ':
                        hasspace = true;
                        break;
                    case '_':
                        has_ = true;
                        break;
                    default:
                        break;
                }
            }
            #endregion
            #region 计算 Code 值
            if (hasdot)
                Code += 1;
            if (hasspace)
                Code += 2;
            if (has_)
                Code += 16;
            if (regex_hasDear.IsMatch(FirstTwentiesChar))
                Code += 8;
            if (regex_hasHi.IsMatch(FirstTwentiesChar))
                Code += 4;
            if (upperlitter / lowerlitter > 0.8)
                Code += 32;
            if (firstline[0] == 'I')
                Code += 64;
            if (len == 20)
                Code += 128;
            if (firstline[firstline.Length - 1] == ':')
                Code += 256;
            #endregion
        }
        private void handlesecondline(string secondline)
        {
```

```
        if (string.IsNullOrEmpty(secondline))
        {
            Code += 512;
        }
    }
    private int getabit(int whichbit)
    {
        return (Code >> whichbit) % 2;
    }
    //这里所有的 bits 都是低索引在低位
    private static int getcode(int[] bits)
    {
        int result = 0;
        for (int i = bits.Length - 1; i >= 0; i--)
        {
            result = result << 1;
            result += bits[i];
        }
        return result;
    }
    static public FirstLineCode SumAll(List<FirstLineCode> allcodes)
    {
        int count = allcodes.Count;
        int _30percent = (int)(count * .7);
        int[] bits = new int[bitsNum];
        foreach (var code in allcodes)
        {
            for (int i = 0; i < bitsNum; i++)
            {
                bits[i] += code.getabit(i);
            }
        }
        for (int i = 0; i < bits.Length; i++)
        {
            if (bits[i] > _30percent)
                bits[i] = 1;
            else
                bits[i] = 0;
        }
        return new FirstLineCode(getcode(bits));
    }
    public string ToHumanString()
    {
        string result = Convert.ToString(Code, 2);
```

```
            while (result.Length < bitsNum)
            {
                result = $"0{ result }";
            }
            return result;
        }
        public override string ToString()
        {
            return Code.ToString();
        }
        public static int Judge(FirstLineCode a, FirstLineCode b)
        {
            string sa = a.ToHumanString();
            string sb = b.ToHumanString();
            int result = 0;
            for (int i = 0; i < sa.Length; i++)
            {
                if (sa[i] == sb[i])
                    result++;
            }
            return 10 - result;
        }
    }
    /// <summary>
    /// 封装了判断是否是形容词的方法，与形容词标准后缀字典
    /// </summary>
    public class AdjJudge
    {
        //包含了形容词标准后缀
        static public string[] Suffixs = { "able", "ible", "al", "ant", "ar", "ary",
            "ful", "ic", "ical", "ive", "less", "ous", "some", "ward" };
        public static bool IsAdj(string input)
        {
            foreach (var suffix in Suffixs)
            {
                //遍历字典，生成正则表达式，判断是否符合形容词标准
                Regex regex_adj = new Regex("[A-Za-z]{3}" + suffix + "$");
                if (regex_adj.IsMatch(input))
                    return true;
            }
            return false;
        }
    }
    /// <summary>
```

```
    /// 为方便在 matlab 中进行函数拟合的封装
    /// </summary>
    public struct Point
    {
        public int x;
        public int y;
        public Point(int x, int y)
        {
            this.x = x;
            this.y = y;
        }
    }
}
```