Neural Networks In Email Text Classification: Categorizing The Enron Email Corpus

Martin Wunderlich, MA

Matrikelnr. 11007098

2014-04-21

CIS, LMU

Contents

List of figures 3

List of tables 3

1 Introduction 4

2 Neural networks in email text classification: Categorizing the Enron email corpus 5

2.1 Theoretical background of artificial neural networks . . . 5

2.1.1 Learning algorithms . . . 7

2.1.1.1 Derivation of delta rule for single-layer perceptrons (SLPs) . . . 7

2.1.1.2 Simple examples of ANNs to model boolean functions using SLPs . . . 9

2.1.1.3 Derivation of delta rule for multi-layer perceptrons (MLPs) . . . 10

2.1.1.4 Simple example of ANN to model boolean function using MLP . . . 11

2.2 Email classification using neural networks . . . 12

2.2.1 The problem of email classification and feature selection . . . 12

2.2.2 Introducing the Enron corpus and categories . . . 13

2.2.3 The preparational task of manual corpus classification . . . 14

2.3 Text pre-processing and normalization . . . 19

2.4 Experiments . . . 20

2.4.1 Methodology for training, testing and evaluation . . . 20

2.4.2 Description of configuration parameters . . . 21

2.4.3 Results and analysis . . . 23

2.4.4 Potential refinement and future work . . . 25

3 Conclusion 27

Appendix 1: Algorithms 28

Appendix 2: Examples from the Enron email corpus 29

Appendix 3: Detailed overview of implementation (UML diagrams) 31

List of Figures

1 A simple neural network with one hidden layer 5
2 A linearly separable classification problem 6
3 XOR problem which is not linearly separable 6
4 Plot of Sigmoid function 10
5 AND network 10
6 OR network 11
7 XOR network 12
8 Introduction page of the classification website 14
9 Categorization page of the classification website 15
10 Responses over time 16
11 Cumulative responses over time 16
12 Number of responses per respondent 17
13 Distribution of primary categories 17
14 Distribution of secondary categories 19
15 UML class diagram 1 31
16 UML class diagram 2 32

List of Tables

1 Truth table and ANN results for AND function 10
2 Truth table and ANN results for OR function 11
3 Truth table and ANN results for XOR function 12
4 Counts and distribution of assigned primary categories 18
5 Counts and distribution of assigned secondary categories 18

1 Introduction

″Rabbit's clever," said Pooh thoughtfully.
"Yes," said Piglet, "Rabbit's clever."
"And he has Brain."
"Yes," said Piglet, "Rabbit has Brain."
There was a long silence.
"I suppose," said Pooh, "that that's why he never understands anything."
A.A. Milne, Winnie-the-Pooh

The human brain is a miraculous thing. It allows us to learn in a highly efficient and robust way. These ways of learning began to unravel their mysteries on a neurobiological level around the middle of the 20th century. Based on the advances in neurology around that period, mathematicians and computer scientists began to construct artificial neural networks to mimic the learning capabilities of the human brain (cf. e.g. [Ros58], [Ros62], [MP69]) - Mathematics imitating life, so to speak.

One area in which artificial neural networks have been used successfully is natural language processing. Amongst many other tasks, such as hand-writing recognition and named entity recognition (NER), the task of text classification favourably lends itself to the use of neural networks. Given a sufficiently large set of manually pre-classified texts, a neural network can be trained on this training data and can then serve to further categorize new, uncategorized texts in an automated and scalable manner.

In this present work, artificial neural networks (ANN) are used to classify emails from the so-called "Enron corpus", a collection of more than 600.000 emails. After the Enron corporation declared bankruptcy in 2001, these emails were released to the public domain in the course of the ensuing investigations into the fraudulent accounting practices at Enron. The present work will apply supervised learning and use a subset of 1000 manually categorized emails from this corpus to train artificial neural networks, which are then used to categorize the remainder of the corpus. Various training parameters and network topologies are being tested in the experiment section in order to determine the best configuration. For each configuration, one instance of an ANN is trained and tested. The best performing network (in terms of precision, recall and F1 meassure) and the parameter configuration associated with it are stored.

The remainder of this work is structured as follows[1]: In section 2.1, "Theoretical background of artificial neural networks", the algorithms behind single-layer perceptrons (SLPs) and multi-layer perceptrons (MLPs) are explained, which are the two network topologies used in the present implementation[2]. Section 2.2, "Email classification using neural networks", will then provide details on the task of email classification and how the manually classified subset of the Enron corpus was obtained. In chapter 2.3, "Text pre-processing and normalization", the details of the pre-processing steps are explained that were applied to the raw text. Finally, various experiments on the training corpus are described in section 2.4, together with results and potential future work in order to improve on them.

[1]Note that the usual introductory section on the biological foundations of ANNs is left out in this present work; for a very detailed explanation of the biological background, see [Gra92, pp. 6-20] or [Kri07, pp. 15-34].

[2]Note that more complex topologies, such as Hopfield networks, time-delayed neural networks, Boltzmann machines or adaptive resonance theory, are not covered and would be beyond the scope of the present work

2 Neural networks in email text classification: Categorizing the Enron email corpus

″I am a brain, Watson. The rest of me is a mere appendix."
– Arthur Conan Doyle, The Adventure of the Mazarin Stone

2.1 Theoretical background of artificial neural networks

Neural networks are directed graphs which consists of a number of input neurons (nodes), to which a feature vector is applied. The results are then propagated through zero or more hidden neuron layers and an output layer will be deliver a result. The connections (edges) between two neurons are referred to as synapses. Each synapse has a value associated with it that reflects the strength of the connection and is referred to as the weight (see the following illustration[3]).

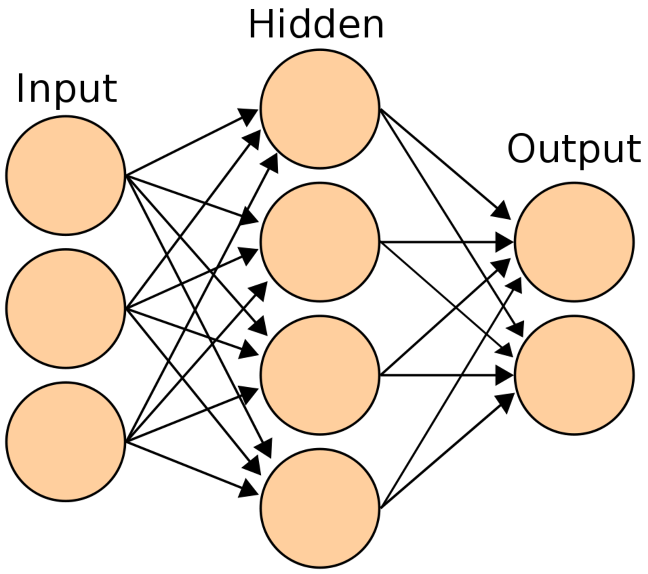


Figure 1: A simple neural network with one hidden layer

In general, an artificial neural network[4] can be described as a 4-tuple $\langle I, N, O, V \rangle$ with:

- I - set of input neurons;
- O - set of output neurons;
- N - set of neurons other than those from sets I and O (i.e. hidden layer neurons);
- V - set of directed vertices (synapses) between neurons.

[3]Image source: en.wikipedia.orgwikiFile:Artificial_neural_network.svg - last accessed: 2014-04-16; author: Colin M.L. Burnett
[4]In the following, when referring to "neural networks" we will always mean the artificial ones, not the biological ones.

A neuron is a 6-tuple $\langle x, w, f_{prop}, f_{act}, f_o, o\rangle$ with:

- x - input vector;
- w - weight vector;
- f_{prop} - a propagation function, for instance the weighted sum of inputs; (the output of the propagation function is known as the net input "net";)
- f_{act} - an activation function, for example the Fermi function as an instance of a sigmoid function[5];
- f_o - an output function, such as the identity function;[6]
- o - the output of the neuron.

A synapse is a 3-tupel $(n_i, n_j, w_{i,j})$ with:

- n_i - a neuron $\in I \cup N$;
- n_j - a neuron $\in N \cup O$
- $w_{i,j}$ - a weight on the synapse with $w_{i,j} \in \mathbb{R}$

When |N| = 0, the network contains only input and output neurons and no hidden layers. In this case, the topology is known as a single layer perceptron (SLP) [Ros62, p. 246f]. When |N| > 0, the topology is referred to as a multi-layer perceptron (MLP). Due to its high degree of interconnectedness, the MLP topology is particularly useful - and even required - for solving complex problems that are not linearly separable (cf. [Gra92, p. 111]). In order to solve such problems, a minimum of one hidden neuron layer is required[7]. The following illustrates the difference [Kaw00, p. 15f]:

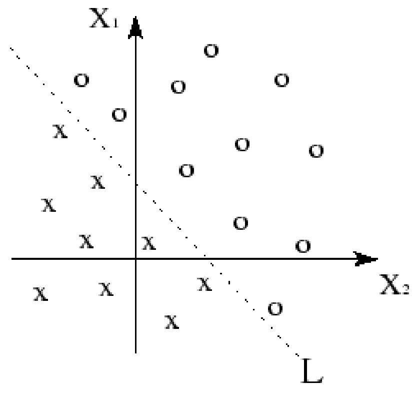
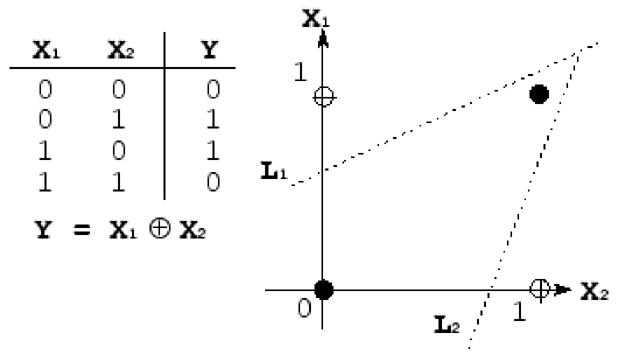


Figure 2: A linearly separable classification problem



X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$

Figure 3: XOR problem which is not linearly separable

Artificial neural networks have a number of distinct advantages when compared to other supervised learning methods. ANNs process information in a sub-symbolic manner, i.e. without explicitly encoding rules and symbols, and are therefore particularly suitable for tasks such as pattern recognition (cf. [Pao89]), where an explicit, symbolic

[5]It is important to note that the activation function must be semi-linear, i.e. it must be a continuous, monotonous and differentiable function. Cf. [RZBK98], [ISO$^+$10]

[6]These functions will be explained in detail further below in section 2.1.1.

[7]In fact, the idea of introducing hidden layers separates the development from "old" to "new" connectionism (cf. [Zau99, chapter 4]) and has lead to a revived interest in neural networks in the 1980s, e.g. described in [RM86].

coding of recognition rules for every possible pattern would not be feasible. Neural networks also have high fault tolerance and can be used for parallel information processing (cf. [Zau99, 40ff]). Furthermore, ANNs are able to map associative information and can generalize from particular data to more generic, albeit implicit, rules in an inductive manner (cf. [Zau99, p. 67]).

In fact, neural networks both have a long history of being applied in the area of computational linguistics and are very much state-of-the-art at the same time. Examples can be found in "Transition-based Dependency Parsing Using Recursive Neural Networks" [Ste13], in building language models (for instance, "A neural probabilistic language model" [BDV00] or a "Recurrent neural network based language model" [MKB^{+}10]) and various other applications (cf. [MCCD13], [MDP^{+}11] etc.).

Before moving on to the application of ANNs in this present work, we need to examine the basics of learning algorithms.

2.1.1 Learning algorithms

In general, when working with ANNs, a set of sample data - represented as a feature vector - with a known outcome is applied to the input neurons. The actual output is then compared to the known, correct output and an error value is calculated based on the deviation between them. Based on the error, the weights of the synapses are adjusted to provide a better fit of the actual output to the known output, i.e. the error is minimized. This process is referred to as the "learning" process or the "training" of the ANN.

The follow parameters are used to describe in the following detailed description of the training:

- P - set of training samples
- p - one training sample
- t - teaching input, i.e. the known desired output
- y - the actual output for one particular training sample
- $w_{i,\Omega}$ - weight for output neuron Ω

2.1.1.1 Derivation of delta rule for single-layer perceptrons (SLPs)

The basis for all neural network learning algorithms and, therefore, for the following derivation[8] is Hebb's rule, which states that the change of a weight of synapse $w_{i,j}$ from neuron i to neuron j is equal to the product of the output o_i of neuron i, the activation a_j of neuron j and a constant η known as the learning rate:

$$\Delta w_{i,j} = \eta \cdot o_i \cdot a_j \tag{1}$$

As hinted at above, when evaluating the performance of a neural network, we can state an error function Err(W) that maps a vector W of weights to a real number. This real number is the error of the network. The generalized error function for a given input vector can therefore be stated as follows:

[8]The following section is largely based on [Kri07, pp. 81ff] and [Rit91, 53ff]

$$Err : W \to \mathbb{R} \tag{2}$$

In order to determine the way in which the weights in vector W need to be modified (ΔW), we calculate the derivative of the generalized error function Err(W) - symbolized by the Nabla ∇ operator - as follows:

$$\Delta W = -\eta \nabla Err(W) \tag{3}$$

Then, to calculate the delta, i.e. the modification, for an individual weight w and a particular output neuron Ω, we calculate the partial derivative of the generalized error function Err(W) with respect to this particular weight $w_{i,\Omega}$:

$$\Delta w_{i,\Omega} = -\eta \frac{\partial Err(W)}{\partial w_{i,\Omega}} \tag{4}$$

We move on to calculate the error $Err_p(W)$ for one particular training sample by using the quadratic distance between actual output y and desired output t, summing the quadratic difference over the set of all output neurons O:

$$Err_p(W) = \frac{1}{2} \sum_{o \in O} (t_{p,\Omega} - y_{p,\Omega})^2 \tag{5}$$

Consequently, the total error Err(W) for all training samples can simply be calculated by summing up (5) over all training samples:

$$Err(W) = \sum_{p \in P} Err_p(w) \tag{6}$$

We have thus derived the error function, Err(w), which can be inserted into (4) to derive the modification for one weight, $\Delta w_{i,\Omega}$, for the entire set of training samples P:

$$\Delta w_{i,\Omega} = -\eta \frac{\partial Err(W)}{\partial w_{i,\Omega}} = \sum_{p \in P} -\eta \frac{\partial Err_p(W)}{\partial w_{i,\Omega}} \tag{7}$$

The chain rule of partial derivatives allows us to rewrite (7) as follows:

$$\frac{\partial Err_p(W)}{\partial w_{i,\Omega}} = \frac{\partial Err_p(W)}{\partial o_{p,\Omega}} \cdot \frac{\partial o_{p,\Omega}}{\partial w_{i,\Omega}} = -(t_{p,\Omega} - o_{p,\Omega}) \cdot \frac{\partial o_{p,\Omega}}{\partial w_{i,\Omega}} = -\delta_{p,\Omega} \cdot \frac{\partial o_{p,\Omega}}{\partial w_{i,\Omega}} \tag{8}$$

(note the "delta" symbol in the above which is where the Delta-rule derives its name from)

The above expression (8) can be rewritten, since all weights except the one we are interested in can be considered constants (and the first derivative of a constant being, of course, zero):

$$= -\delta_{p,\Omega} \cdot \frac{\partial \sum_i (o_{p,i} \cdot w_{i,\Omega})}{\partial w_{i,\Omega}} = -\delta_{p,\Omega} \cdot o_{p,i} \tag{9}$$

When this expression is now inserted into (6), the **delta rule** for batch learning ("offline learning") is derived, i.e. for a learning process that uses the entire set P of training samples:

$$\Rightarrow \Delta w_{i,\Omega} = \eta \sum_{p \in P} o_{p,i} \cdot \delta_{p,\Omega} \tag{10}$$

When running one training sample at a time, the "online" version of this delta rule can be written thus:

$$\Delta w_{i,\Omega} = \eta \cdot o_i \cdot \delta_\Omega = \eta \cdot o_i \cdot (t_\Omega - o_\Omega) \tag{11}$$

Based on the derivations detailed above, a simple learning algorithm for SLPs can be stated which is listed in section 3 "Conclusion". More detailed examples of ANNs to demonstrate the functionality are provided in the following section 2.1.1.2.

2.1.1.2 Simple examples of ANNs to model boolean functions using SLPs

Single-layer perceptrons can be used to model boolean functions, which serves as a useful example to understand the inner workings of ANNs. The implementation that is part of this present work (and which is described in detail in section 3) includes a test section in order to verify the correct functioning of the neural network library that is used (SNIPE[9]). These tests manually build three neural networks, one each for the boolean functions AND, OR and XOR. The first two are single-layer perceptrons, i.e. they do not include any hidden layers. The latter, however, is an MLP with one hidden layer, to be covered in detail in section 2.1.1.4.

As activation function, both networks are using the Fermi function: $sig(x) = \frac{1}{1+e^{-x}}$, which is illustrated in figure 4

The result of the propagation function net is the weighted sum of inputs and the output function is the identity function.

The weights and topology for the AND and the OR networks have been set and verified manually and are presented here in Figures 5 and 6, respectively. Note that the only difference between these two neural networks is the weight that is set on the synapse between the bias neuron "BIAS" (a neuron which always outputs a value of 1) and the output neuron "3".

[9]See www.dkriesel.com/tech/snipe

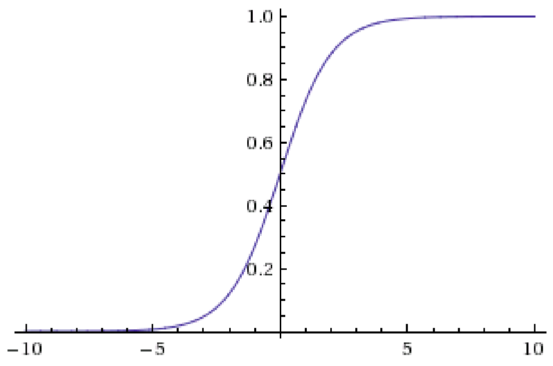


Figure 4: Plot of Sigmoid function

x1	x2	y	net	o	rounded
0	0	0	-30	0,00000	0
1	0	0	-10	0,00004	0
0	1	0	-10	0,00004	0
1	1	1	10	0,99995	1

Table 1: Truth table and ANN results for AND function

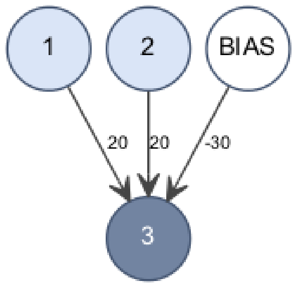


Figure 5: AND network

2.1.1.3 Derivation of delta rule for multi-layer perceptrons (MLPs)

As mentioned above, there are problems which can not be solved using a simple SLP. One such problem is the boolean Exclusive OR function (XOR), which requires at least one hidden neuron layer with two neurons a_4 und a_5 to be present in the network topology (cf. [Kri07, p. 84]).

In order to derive a modified delta rule for MLPs, additional parameters are needed:

- n - number of weights in the network;
- K - set of predecessor neurons with |K| = k (for a specific neuron h);
- L - set of successor neurons with |L| = l (for a specific neuron h);
- $w_{k,h}$ – weight of the synapse between neurons k and h,

Again, the partial derivative of the generalized error function for a weight is calculated for a particular neuron, referred to here as h:

$$\frac{\partial Err(w_{k,h})}{\partial w_{k,h}} = \frac{\partial Err}{\partial net_h} \cdot \frac{\partial net_h}{\partial w_{k,h}} = -\delta_h \cdot o_k \tag{12}$$

Using again the chain rule for composite derivatives, the delta factor can be rewritten:

$$\delta_h = -\frac{\partial Err}{\partial o_h} \cdot \frac{\partial o_h}{\partial net_h}$$

x1	x2	y	net	o	rounded
0	0	0	-10	0,00004	0
1	0	1	10	0,99995	1
0	1	1	10	0,99995	1
1	1	1	30	0,99999	1

Table 2: Truth table and ANN results for OR function

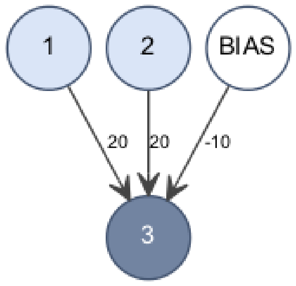


Figure 6: OR network

(13)

And the first term of (13) can be calculated like:

$$-\frac{\partial Err}{\partial o_h} = \sum_{l \in L}(-\frac{\partial Err}{\partial net_l} \cdot \frac{\partial net_l}{\partial o_h}) = \sum_{l \in L} \delta_l \cdot w_{h,l} \tag{14}$$

This leads to the general calculation of synapse weight changes used in the "backpropagation" algorithm:

$$\Rightarrow \Delta w_{k,h} = \eta \cdot o_k \cdot \delta_h \tag{15}$$

In the case of output neurons, delta is calculated like this:

$$\delta_h = f'_{act}(net_h) \cdot (t_h - y_h) \tag{16}$$

And for inner (hidden layer) or input neurons like this:

$$\delta_h = f'_{act}(net_h) \cdot \sum_{l \in L}(\delta_l \cdot w_{h,l}) \tag{17}$$

For the learning algorithm for multi-layer-perceptrons, see appendix 3.

2.1.1.4 Simple example of ANN to model boolean function using MLP

As mentioned above, the XOR function presents a problem that is not linearly separable and, consequently, cannot be modeled using an SLP. Therefore, at least one hidden layer needs to be introduced. The truth table and network topology of this function are as follows:

x1	x2	y	net	o	rounded
0	0	0	-10	0,00004	0
1	0	1	10	0,99995	1
0	1	1	10	0,99995	1
1	1	0	30	0,00004	0

Table 3: Truth table and ANN results for XOR function

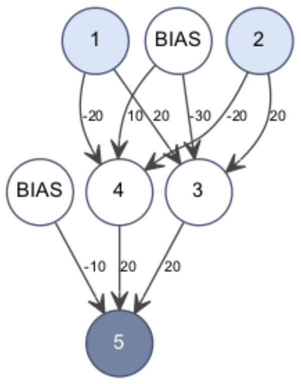


Figure 7: XOR network

2.2 Email classification using neural networks

2.2.1 The problem of email classification and feature selection

Given the flood of emails that many users have to deal with, the task of automatic email classification is not a new one. Neural networks have been used before, for instance to classify emails into categories of urgency [AZK09]. As pointed out ibid. p.1, neural networks and backpropagation in particular are effective methods: "...previous research has shown that backpropagation in neural networks (NNs) can achieve very accurate results, that are sometimes more accurate than those of the symbolic classifiers[10]." [AZK09, p. 1]. Other authors have developed methods of automatic categorization of emails into thematic folders and used these methods as benchmark experiments on Enron and SRI Corpora [BMH04].

A key question in using methods of supervised learning is the selection of appropriate features derived from the raw data. Various approaches can be found in previous related work when selecting features to be used in email classification. For instance, in [WHJ10] thread info and address groups are used to classify emails. Other possible features in addition to the text itself are meta information, such as sender, CC field, BCC field, addressee etc. (cf. [AZK09, p. 2]).

In the present work, a relatively simple "bag-of-words" feature model with unigrams is used, whereby the presence or absence of each unique word (type) is considered a binary feature. Thus, the email text can be represented as a binary vector with a length of the total vocabulary size, whereby a value of 1.0 signifies the presence of the respective word in the email under consideration and a value of 0.0 the word's absence. That is, the model used in the present work applies a binary vector space model. The frequency (absolute or relative) of words in an email is not taken into account[11].

In the present work, a total of fourteen thematic categories was used. Therefore, the task is an instance of the class of problems known as "multi-class classification" tasks. One approach to such a task is to treat the

[10]Symbolic classifiers here refers to methods such as Support Vector Machines (SVM), Maximum Entropy (ME) or Naive Bayes etc.)

[11]Such a slightly more advanced model is used, for instance, in [AZK09] who state: "The number of occurrences of a word in an email (frequency of occurrence) is the attribute's value for that email message. Email messages are therefore represented as vectors of numeric attributes where each attribute value is the frequency of occurrence of a distinct term. This set of email message vectors is often referred to as a vector space. Algorithms that operate on such representations are said to be using vector space models of the data." (cf. [AZK09, p. 3])

classification as a one-versus-all classification. That is, one classifier per class is trained and tested, assuming that no correlation exists between classes. This approach is used, for instance, in [TKB12] and is also known as "Binary relevance learning" (ibid. p. 2). However, in the present work, a single classifier ANN was created for the entire set of categories, so that any correlations between the categories could be take into account in the trained ANN. Furthermore, the task also takes the form of a multi-label classification task, since each email in the training set was manually assigned to one primary category and zero or more secondary categories[12].

2.2.2 Introducing the Enron corpus and categories

The Enron corporation declared bankruptcy in December 2001 and in the course of the ensuing investigations by the Federal Energy Regulatory Commission (FERC; a US institution that regulates the transmission of electricity, natural gas, and oil), a corpus of 619,446 messages belonging to 158 users was released to the public[13]. The corpus has since been a popular dataset with NLP researchers, since it represents the only "real" email corpus of this size that is publicly available. One of the first works using the Enron corpus was carried out by Klimt et al. and dealt with folder classification [KY04].

In the present work fourteen possible categories are used, which are based on a much more granular set used at the University of Berkeley[14]. The categories are as follows:

1. Company business, strategy, alliances etc.
2. Personal, Friends, Family
3. Logistic arrangements (meetings etc.)
4. Employment arrangements (hiring etc.)
5. News and newsletters
6. Legal and regulatory affairs
7. Jokes, humour, fun
8. Project work, general collaboration
9. Company image, PR, press releases
10. Political influence and contacts
11. IT (Information technology)
12. Empty message
13. Spam
14. Other

The total size of the raw corpus used in this present work was 388002 emails.

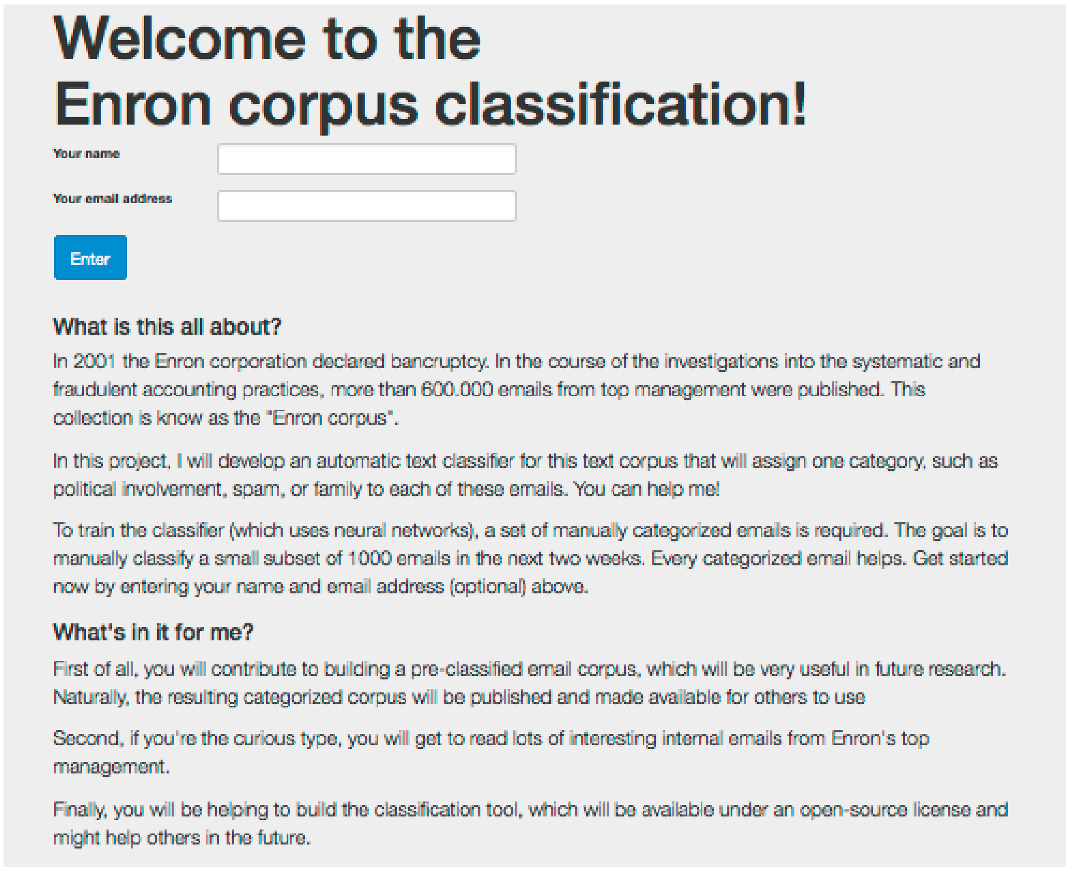
[12]More details on the manual assignment see 2.2.3, "The preparational task of manual corpus classification"

[13]This corpus is available under $www.ferc.gov/industries/electric/indus-act/wec/enron/info-release.asp$ (last accessed: 2014-04-06); the present work uses the version provided by William M. Cohen of Carnegie Mellon University ($www.cs.cmu.edu/~enron/$), which can be accessed here: $cs.bennington.edu/courses/f2012/cs4130.01/enron\_mail\_20110402.tgz$; for an example of the news coverage on the published email corpus see: $www.salon.com/2003/10/14/enron\_22/$ (last accessed: 2014-04-06) and for a chronology of the case refer to: $www.citizen.org/cmep/article\_redirect.cfm?ID = 7101$ (2014-04-06)

[14]$courses.ischool.berkeley.edu/i290-2/f04/assignments/categories.txt$ (last accessed 2014-04-06)

2.2.3 The preparational task of manual corpus classification

As effective as methods of supervised learning are, they have the drawback of requiring a pre-categorized set of data for training of the classifier. In the present work, this problem was solved by setting up a website[15] to manually classify a subset of 1000 randomly selected emails from the entire Enron corpus. The start page provided some background information on the project and allowed respondents to enter their name and an optional email address:



Welcome to the
Enron corpus classification!

Your name

Your email address

Enter

What is this all about?

In 2001 the Enron corporation declared bancruptcy. In the course of the investigations into the systematic and fraudulent accounting practices, more than 600.000 emails from top management were published. This collection is know as the "Enron corpus".

In this project, I will develop an automatic text classifier for this text corpus that will assign one category, such as political involvement, spam, or family to each of these emails. You can help me!

To train the classifier (which uses neural networks), a set of manually categorized emails is required. The goal is to manually classify a small subset of 1000 emails in the next two weeks. Every categorized email helps. Get started now by entering your name and email address (optional) above.

What's in it for me?

First of all, you will contribute to building a pre-classified email corpus, which will be very useful in future research. Naturally, the resulting categorized corpus will be published and made available for others to use

Second, if you're the curious type, you will get to read lots of interesting internal emails from Enron's top management.

Finally, you will be helping to build the classification tool, which will be available under an open-source license and might help others in the future.

Figure 8: Introduction page of the classification website

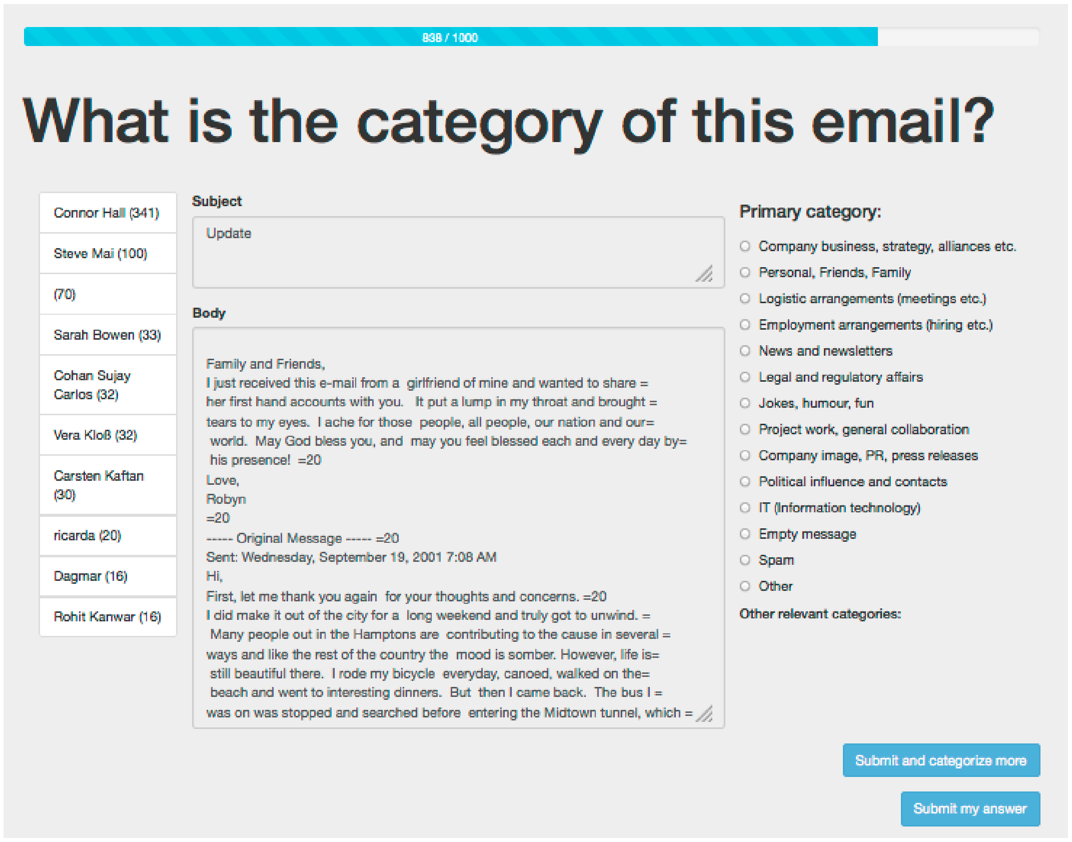
The next page showed a randomly selected email from the subset of 1000 emails with a list of possible primary categories to the right, a list of the top ten respondents to the left and a progress bar at the top, see figure 9.

The URL to the website was sent out to a number of networks of the author (friends, family, colleagues, groups on LinkedIn[16] etc.). After 18 days of the website's go-live on the 9th of March 2014, the goal of 1000 manually categorized emails had been reached. The graphics below show the (cumulative) responses over time (figures 10 and 11), the distribution of primary categories only (table 4[17] and figure 13), the distribution of primary and secondary categories combined (table 5 and figure 14), and the number of responses per respondent (figure 12). The total number of distinct named respondents was 29, with 101 responses coming from respondents who preferred to remain anonymous. The arithmetic mean of responses was 22.2 responses per respondent, the median was 9 and

[15]Published at www.martinwunderlich.com/enron/

[16]See the following group discussions: lnkd.in/dN43en8 and lnkd.in/dfcE95S (last accessed 2014-04-05)

[17]The large number of responses assigned to the first category as shown in this table indicates that this category was formulated too broadly.



What is the category of this email?

Connor Hall (341)
Steve Mai (100)
(70)
Sarah Bowen (33)
Cohan Sujay Carlos (32)
Vera Kloß (32)
Carsten Kaftan (30)
ricarda (20)
Dagmar (16)
Rohit Kanwar (16)

Subject

Update

Body

Family and Friends,
I just received this e-mail from a girlfriend of mine and wanted to share =
her first hand accounts with you.   It put a lump in my throat and brought =
tears to my eyes. I ache for those people, all people, our nation and our=
 world. May God bless you, and may you feel blessed each and every day by=
 his presence! =20
Love,
Robyn
=20
----- Original Message ----- =20
Sent: Wednesday, September 19, 2001 7:08 AM
Hi,
First, let me thank you again for your thoughts and concerns. =20
I did make it out of the city for a long weekend and truly got to unwind. =
 Many people out in the Hamptons are contributing to the cause in several =
ways and like the rest of the country the mood is somber. However, life is=
 still beautiful there. I rode my bicycle everyday, canoed, walked on the=
 beach and went to interesting dinners. But then I came back. The bus I =
was on was stopped and searched before entering the Midtown tunnel, which =

Primary category:

- ○ Company business, strategy, alliances etc.
- ○ Personal, Friends, Family
- ○ Logistic arrangements (meetings etc.)
- ○ Employment arrangements (hiring etc.)
- ○ News and newsletters
- ○ Legal and regulatory affairs
- ○ Jokes, humour, fun
- ○ Project work, general collaboration
- ○ Company image, PR, press releases
- ○ Political influence and contacts
- ○ IT (Information technology)
- ○ Empty message
- ○ Spam
- ○ Other

Other relevant categories:

Figure 9: Categorization page of the classification website

the standard deviation 53.1[18].

The responses were not cross-validated, since this would have been beyond the scoped of this present work and could have meant that the goal 1000 manually categorized emails might have been impossible to reach. In a more robust experimental setting, each email should be categorized by more than one respondent and the agreement (or disagreement) of respondents be taken into account in assigning the eventual categories. The responses are illustrated in the following figures and tables.

[18]It must be pointed out that two particular respondents (who were not previously known to the author) accounted for a large number of categorized emails, which is reflected in these values. The distributions would look rather different, if one were to subtract the responses of these two respondents.

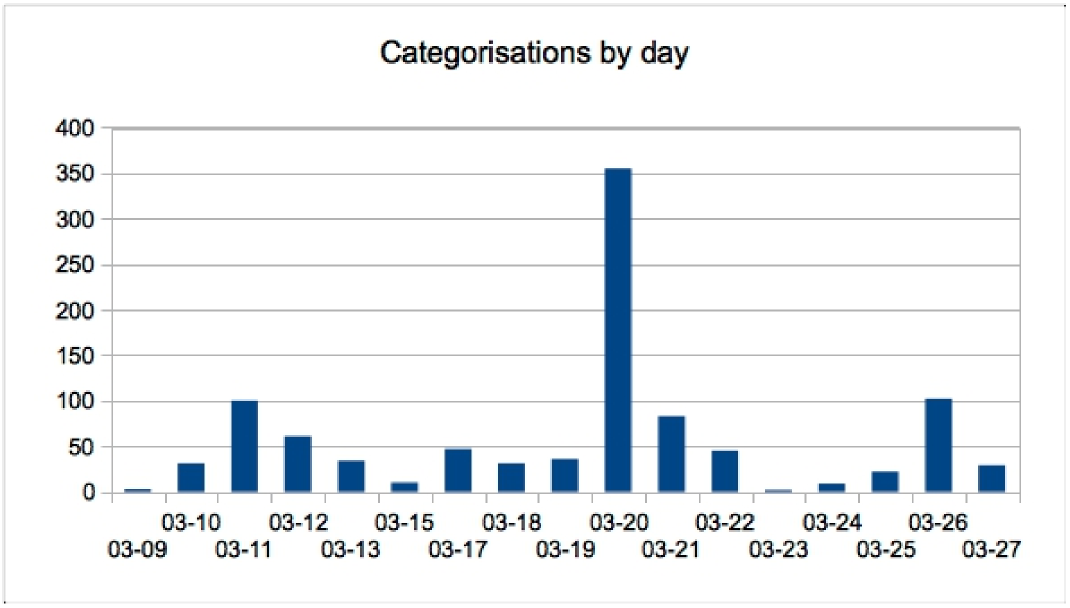


Figure 10: Responses over time

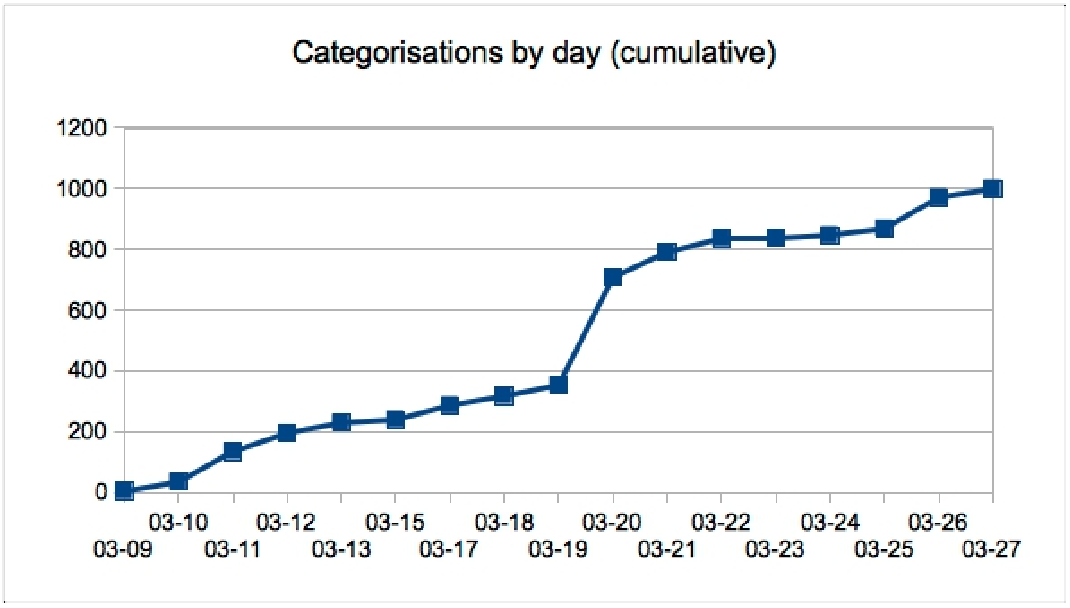


Figure 11: Cumulative responses over time

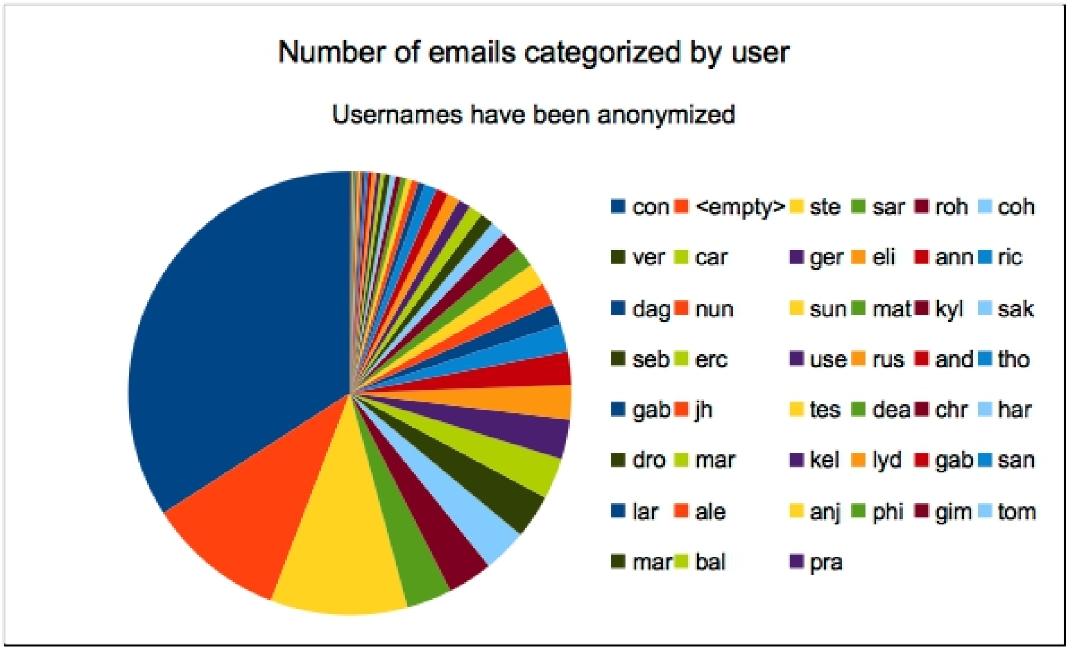


Figure 12: Number of responses per respondent

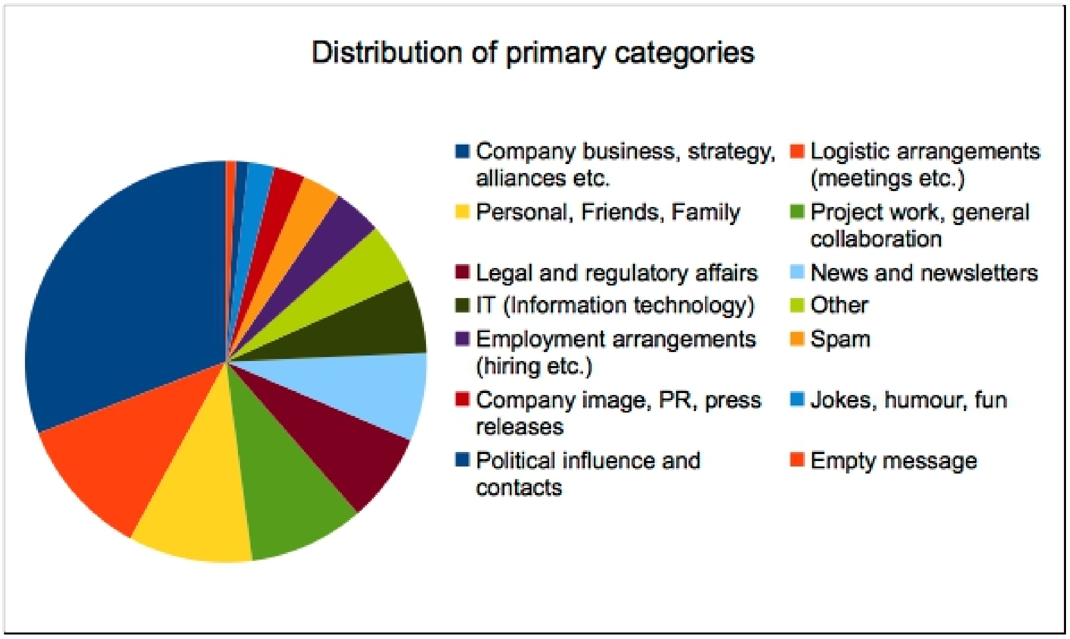


Figure 13: Distribution of primary categories

Category	Absolute	Percent
Company business, strategy, alliances etc.	308	30,8%
Logistic arrangements (meetings etc.)	113	11,3%
Personal, Friends, Family	100	10,0%
Project work, general collaboration	93	9,3%
Legal and regulatory affairs	72	7,2%
News and newsletters	71	7,1%
IT (Information technology)	60	6,0%
Other	49	4,9%
Employment arrangements (hiring etc.)	39	3,9%
Spam	31	3,1%
Company image, PR, press releases	25	2,5%
Jokes, humour, fun	21	2,1%
Political influence and contacts	10	1,0%
Empty message	8	0,8%

Table 4: Counts and distribution of assigned primary categories

Category	Absolute	Percent
Company business, strategy, alliances etc.	100	19,5%
Project work, general collaboration	93	18,1%
Logistic arrangements (meetings etc.)	51	9,9%
News and newsletters	51	9,9%
Legal and regulatory affairs	43	8,4%
Other	37	7,2%
Personal, Friends, Family	34	6,6%
IT (Information technology)	26	5,1%
Company image, PR, press releases	20	3,9%
Employment arrangements (hiring etc.)	19	3,7%
Political influence and contacts	12	2,3%
Jokes, humour, fun	11	2,1%
Spam	10	1,9%
Empty message	7	1,4%

Table 5: Counts and distribution of assigned secondary categories

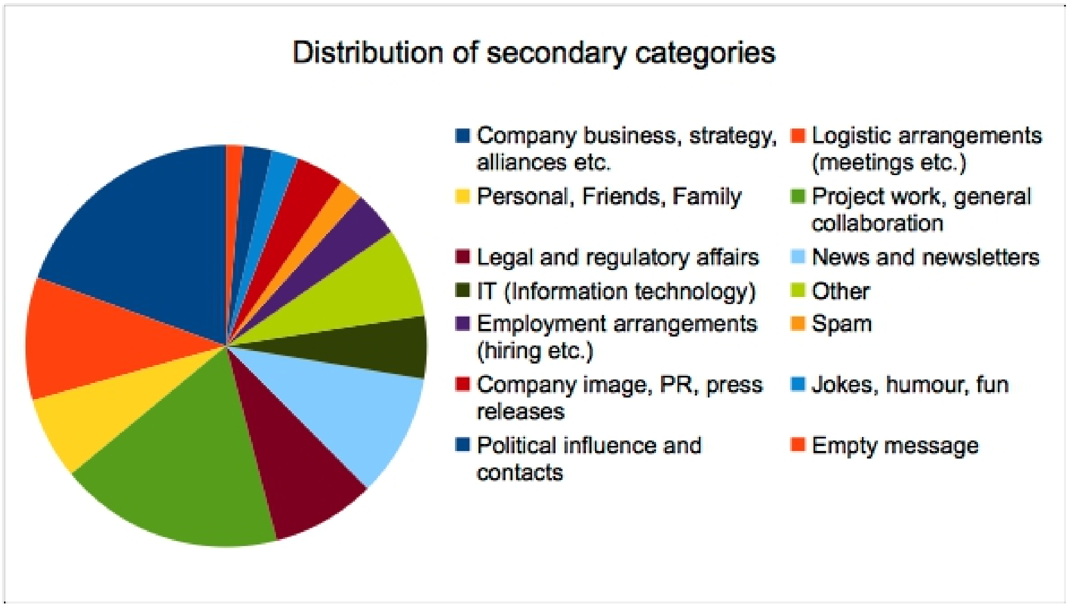


Figure 14: Distribution of secondary categories

2.3 Text pre-processing and normalization

The following section provides details on the general approach for text pre-processing and normalization that was used in the Java implementation. For more details on the classes and the actual code, please refer to the Java doc provided with the source code and also see section 3 "Conclusion".

A number of pre-processing steps were taken in order to extract only relevant information from the manually categorized sub-corpus of 1000 emails. First of all, emails from a number of directories were ignored: $sent, sent\_mail, 2\_trash, attachments, e\_mail\_bin, old\_sent, saved\_sent, sent, sent\_items$. Only subject line and the email body were extracted from the emails, removing meta-information, such as sender or recipient(s). The text contents of subject and email body were then split into sentences and each sentence was tokenized using the Apache OpenNLP library[19]. Losely following the approach taken in [AZK09], each token was then converted to all-lower case to build a unigram representation of the particular email. The types that were thus gained were also added to the vocabulary of the unigram language model [20]

[19]URL: opennlp.apache.org - last accessed 2014-04-16

[20]Several additional steps of preprocessing that are also used in [AZK09, p. 3] were implemented as well and are provided for in the configuration class and properties. In particular, these are the binary parameters of using stop words and stemming (also pointed out as a useful pre-processing step in [KY04, p. 1]: "Stemming and stop word removal are often used as they are useful in general text classification, although their usefulness in email in particular has not yet been studied.") and the mapping of various strings with little information value (such as strings of digits, dates, times and phone numbers) to a single common token using regular expressions (regarding this mapping step cf. [ZZ06, p.11]). Note that unlike [Mac13, 24-30] the intermediate steps of each preprocessing step were not stored, so that the implementation could use parameters to conditionally use a subset of preprocessing steps and compare the results of various configurations.

Other parameters for pre-processing are provided for in the Java implementation, but not yet implemented, such as text enrichment by automatically adding synonyms from WordNet or removal of most frequent words. The limitations of temporal and computational resources available for the present work did not allow to implement these parameters. More possibilities of extending this pre-processing stage are pointed in out in 2.4.4, "Potential refinement and future work".

2.4 Experiments

The application provides various runtime modes, which are described in the following. Additionally, various parameters can be configured in the properties file. In general, when using the application it should be noted that all outputs are written to the log file that is specified in the properties file. The following command line parameters can be passed to the executable JAR file:

- "setup" - selects 1000 random emails from the entire corpus and uploads them to the database of the website that was designed for the task of manual classification;
- "test" - runs some simple neural network tests, creating network topologies for three boolean functions: AND, OR and XOR. The XOR-network is also randomly initialized and trained with backpropagation.
- "train" - based on the manually categorized emails, this execution mode trains one or more neural networks. There are two sub-modes for the training:
  - "default" - uses the default training parameters from the properties file; or
  - "full" - which iterates over all possible parameter combinations based on values and ranges specified in the properties.
- "classification" - uses the trained neural network provided in the properties to classify the entire Enron corpus.

The available parameters which are supported by the implementation are described in detail in the following section 2.4.2. When selecting the training mode, the test data is split up into training data, test data, and optional cross-validation data. The split can be configured in the properties, which by default are set to use 0% of the data for cross-validation, 20% for testing and the remaining 80% for training of the neural networks. So, for the manually pre-categorized set of 1000 emails, 800 will be used for training and 200 for testing the ANNs resulting from this training. Held-out data, if present, would be used for cross-validation of the best performing networks from the training/test runs. Thus, the implementation by default runs a 5-fold cross-validation on the data. When the cross-validation percentage is set to 0%, only the best-performing networks from all test runs are reported as the result of the overall training run.

2.4.1 Methodology for training, testing and evaluation

As pointed out before, instead of treating the classification problem as a "one-vs.-all" classification and thus having to create one classifier per category, it was decided to train one ANN for all categories. This approach allows for correlations to occur between categories (cf. [TKB12, p. 2]).[21] In future work, more techniques for dimensionality reduction should be used in order to reduce the length of the input vector and thus improve performance (some examples can be found in [ZZ06, p.10]).

In order to evaluate the results from each training epoch of the n-fold cross-validation, the test run counts the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) and derives the standard measures

[21]This also reduced the need to train 14 ANNs instead of just one, which was a rather beneficial side-effect given the limited processing power and time available for this present work.

of precision, recall and F1 from these values[22]. These measures are defined as follows (cf. for instance [JM08, p. 489]).

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The configuration and the ANN that delivers the best performance in terms of precision, recall and F1 is globally stored in memory and, where applicable, updated after each test run. When the percentage for cross-validation is greater than 0.0, the best performing ANNs for each training loop - as determined locally by the test runs - is stored and the best performing ANN from each training run is validated against the data held out for cross-validation. As the global result those ANNs are stored, which perform best when checked against the CV data.

2.4.2 Description of configuration parameters

The implementation allows for a large number of parameters that can be used to configure the training of the neural network(s). These parameters can be divided into those that result from the neural network topology, such as number of hidden layers, and those that derive from the nature of the data (i.e. texts in natural language), such as application of stemming as a pre-processing step.

In detail, the following parameters can be configured:

- *hidden_layers* - number of hidden layers to be used in the network topology; must be 0 (SLP), 1, 2 or 3 (MLPs);
- *hidden_neurons* - number of neurons per hidden layer; can be set to -1 to use the same number as neurons in the input layer; note that all hidden layers have the same number of neurons. This parameter has an extremely high influence on the network's performance and should be set as low as possible.
- *allow_shortcut* - boolean value to determine whether or not shortcut connections are allowed in the network;
- *allow_self_loops* - boolean value to determine whether or not self-loop connections are allowed in the network;
- *initial_range* - initial range of synapse weights for random initialisation as a double number; the random weights will be between +/- this number;
- *loops_start, loops_end, loops_increment, default_loops* - number of training loops to be used in backpropgagation (in "full" training mode, the value increments from start to end with provided step size); note: this parameter has a strong influence on the performance in "full" training mode;
- *learning_rate_start; learning_rate_end; learning_rate_decrement; default_learning_rate* - learning rate to be used (in "full" training mode, the value decrements from start to end with provided step size);

[22] As a potential improvement, more refined evaluation metrics could be used, such as hamming loss [ZZ06, p. 6] or micro- and macro-averaging [SG09].

- *precentage_train, precentage_test, percentage_cv* - split percentages of the data for training, testing and cross-validation, as described above;
- *activation_function* - fully qualified path to the class that implements an activation function; by default the class TangensHyperbolicusAnguita is used, which implements the tanh function proposed by Anguita[23] et al.
- *stemming* - use stemming when extracting text from emails and setting up the language model[24];
- *remove_stopwords* - remove stop words when extracting text from emails and setting up the language model[25];
- *use_synonyms_for_training* - during training, create additional email instances by pulling synonyms from WordNet (not implemented, yet)
- *use_mono_class_results_only* - when evaluating results, use only those results where exactly one category has been determined by the classifier ANN (that is, no multi-label results will be considered).
- *weighted_meassure* - use a weighted meassure instead of a binary vector for applying the values of the input vector; set to "false" to use a binary vector or provide a fully qualified name to a class that implements a weighted meassure, such as tf-idf; this class must implement the interface com.martinwunderlich.neuralnetworks.neuraltextclassifier.IWeightedMeassure.
- *remove_most_frequent_words* - remove the most frequent words from emails; set to false or provide a double number which represents the fraction of words to remove; (not implemented, yet)

Additional parameters not related to the training per se[26]:

- *output_file_for_trained_network* - path to where the string representation of the trained network should be written to (in CSV format);
- *results_output_csv_file* - path to where the detailed results of the tests will be written to;
- *categories_file_for_training* - path to the text file with the list of categories; in order to ignore certain categories, lines in this file can be prefixed with '#' to comment them out. This is useful for debugging and for limiting the training data when running on machines with low performance.
- *training_email_directory* - path of where to find the root directory of the emails to be used for training; below this root directory there should be one subdirectory per category, which contains the actual emails;
- *corpus_email_directory* - path of where to find the directory with the Enron corpus emails; used in "setup" mode only;
- *password* - password for the website's database where email samples will be uploaded to for manual classification in "setup" mode.

[23]Cf. D. Anguita, G. Parodi and R. Zunino in Proceedings of the World Congress on Neural Networking, Portland, Oregon, 1993, volume 1 pages 165-168, Lawrence Erlbaum/INNS Press, 1993.

[24]Note that the possible values for the parameters of stemming and stopwords removal are not permutated in building the training configurations in training mode "full". This would inflate the number of language models to build and, thus, increase memory usage and lower performance.

[25]The tokenizer library from the Apache Lucene project was used for stemming and stopwords removal; see lucene.apache.org - last accessed 2014-04-20

[26]File and directory paths should always be absolute paths using the correct syntax for the respective operating system.

2.4.3 Results and analysis

The implementation that was created as part of the present work intended to find the best possible configuration for training an ANN. For this purposes, all possible permutations for the training parameters were created based on the preset values in the properties file. Each of these permutations was then evaluate in an n-fold cross validation process. The overall best performing ANN in terms of precision, recall and F1 was kept track of and then stored for future use in classification mode.

In detail the following configurations were found to give the best performance of the respective trained ANN in a 5-fold cross-validation (80-20-0 split):

Best results of training for single-category (obtained Sun Apr 13 19:40:56 CEST 2014)[27]*:*

Best precision: 1.0

Best precision configuration:

- hiddenNeuronCount: 100;
- hiddenLayers: 2;
- allowShortcuts: false;
- allowSelfLoops: true;
- initialRange: 1.0;
- useMonoClassResultsOnly: false;
- learningRate: 0.2;
- trainingLoops: 1000

Best precision network: N: 8547-100-100-14, S: 866314

Best recall: 1.0

Best recall configuration:

- hiddenNeuronCount: 100;
- hiddenLayers: 3;
- allowShortcuts: true;
- allowSelfLoops: false;
- initialRange: 9.0;
- useMonoClassResultsOnly: false;
- learningRate: 0.15000000000000002;
- trainingLoops: 1500

Best recall network: N: 8547-100-100-100-14, S: 876414

Best F1: 0.6358024691358024

Best F1 configuration:

- hiddenNeuronCount: 100;

[27]useStemming and removeStopwords were always set to "true" to reduce the number of input neurons; weightedMeassure, useSynonyms and removeMostFrequentWords were always set to "false", since these features were not yet available.

- hiddenLayers: 2;
- allowShortcuts: true;
- allowSelfLoops: true;
- initialRange: 3.0;
- useMonoClassResultsOnly: false;
- learningRate: 0.05000000000000002;
- trainingLoops: 1500

Best F1 network: N: 8547-100-100-14, S: 866314

Best results of training for multi-category classification (obtained Sun Apr 13 19:40:56 CEST 2014):

Best precision: 1.0

Best precision configuration:

- hiddenNeuronCount: 100;
- hiddenLayers: 1;
- allowShortcuts: true;
- allowSelfLoops: true;
- initialRange: 1.0;
- useMonoClassResultsOnly: false;
- learningRate: 0.2;
- trainingLoops: 1500

Best precision network: N: 8547-100-14, S: 856214

Best recall: 1.0

Best recall configuration:

- hiddenNeuronCount: 100;
- hiddenLayers: 3;
- allowShortcuts: true;
- allowSelfLoops: false;
- initialRange: 3.0;
- useMonoClassResultsOnly: false;
- learningRate: 0.2;
- trainingLoops: 2000

Best recall network: N: 8547-100-100-100-14, S: 876414

Best F1: 0.5463182897862233

Best F1 configuration:

- hiddenNeuronCount: 100;
- hiddenLayers: 3;
- allowShortcuts: true;

- allowSelfLoops: false;
- initialRange: 3.0;
- useMonoClassResultsOnly: false;
- learningRate: 0.10000000000000002;
- trainingLoops: 3000

Best F1 network: N: 8547-100-100-100-14, S: 876414

The best F1 network for single-category classification was then used to classify the entire corpus of 388002. The results of this classification run were as follows: Out of 388002 emails only 209 - less than 1 per thousand - were assigned to exactly one category. These assignments were:

- Company image, PR, press releases=27
- Empty message=21
- Project work, general collaboration=6
- Other=27
- IT (Information technology)=6
- Political influence and contacts=4
- Employment arrangements (hiring etc.)=35
- Jokes, humour, fun=1
- Personal, Friends, Family=5
- News and newsletters=7
- Legal and regulatory affairs=28
- Logistic arrangements (meetings etc.)=26
- Company business, strategy, alliances etc.=1
- Spam=1

As for the remainder, these were assigned to more than one category, in several cases even as many as 10 out of 14. This hints at a certain lack in precision in the trained ANN. An hypotheses that remains to be tested would be that a more advanced language model (bigram or trigram) might lead to better results. Also, the number of hidden layers, which was limited to 3 for performance reasons, could be increased, possibly yielding more accurate results.

The possibilities were further limited by the available processing power and by the architecture of the application. Training and test runs were carried on a machine with an Intel Xeon CPU X7560 with 2.27 GHz, 8 processor cores and 24GB of RAM. This machine also had to be shared with other students and researchers. Initially, a full test run for the full unigram language model with all possible parameter permutations took approx. 10 days. Only after drastically reducing the number of input nodes to less than 50% of the initial values did the processing time reach a workable duration of approx. two days for a full run. The second limitation was that both the neural network library and the application itself were not designed as multi-threaded applications, which showed itself in the CPU load during test runs being as low as approx. 10% or less. Presumably it would lead to higher performance, if the application were designed a as a multi-threaded application right from the start to make full use of the available CPU cores. For instance, it would be possible to run a train-test run for one cross-validation fold in one thread.

Thus, when using 5-fold cross-validation, it would be possible to increase the processor load by a factor of five and consequently decrease processing time. This would allow to also test more configuration parameters in a feasible way.

2.4.4 Potential refinement and future work

In addition to the configurations, parameters and experiments described above, various enhancements and refinements are possible, which shall only be hinted at here and kept in mind for potential future work.

Instead of using a simple unigram "bag-of-words" model, other N-gram language models could be used for text representation, such as bigram or trigram models. Additionally, it would be possible to use various extra-linguistic features, for instance meta-information of the email correspondence, such as recipients or date/time sent or received, linguistic meta-information, such as POS tag counts (and even limit the extracted words to specific POS only; cf. [AZK09, p. 3]) or total length/word count, number of Named Entitiies (NEs). The meta-information of the emails could also be used to create time splits, as done in [BMH04].

Going beyond mere linguistic (meta) information, it would be possible to combine the ANNs with multi-modality classifiers, e.g. by extending the feature space to include social network characteristics for the purpose of classification (cf. [HC10]).

As for the frequency measures, instead of binary vector space models, absolute and relative frequencies could be used or term frequency - inverse class frequency (TF-ICF) - cf. [AZK09, p. 4]. Also, it might be possible to use a fuzzy set approach instead of absolute assignment to classes, as described in [Pao89, pp. 57-79] and more refined frequency models, such as the Delta TF-IDF approach described in Martineau et al. [MFJP09]. More advanced weighting functions could be used, such as Dudani, Shephard's rule or other sophisticated measures for evaluation (cf. [TKB12, p. 3f]).

As mentioned before, in this present work only feedforward network topologies were used for the ANNs. It would be possible to test the current implementation against other network topologies, such as radial basis function networks, Hopfield networks or others. See [Kri07] for an extensive treatment of various possible topologies.

Finally, it would also be fruitful to compare the classification result using ANNs to other classification methods, for instance: Maximum Entropy (ME), Support Vector Machines (SVM), and Naive Bayes (cf. e.g. Yoo et al. [YYC11]) and evaluate these comparatively for performance and accuracy.

3 Conclusion

"The mind does most of its best thinking when we aren't there. The answers are there in the morning."
– Alain de Botton

The present work has demonstrated that artificial neural networks can be used successfully in text classification by training ANNs with a manually categorized set of sample emails. In doing so, the most suitable parameter values of the configuration for the best F1 meassure were found to be:

- hiddenNeuronCount: 100;
- hiddenLayers: 3;
- allowShortcuts: true;
- allowSelfLoops: false;
- initialRange: 3.0;
- useMonoClassResultsOnly: false;
- learningRate: 0.10000000000000002;
- trainingLoops: 3000

However, it must be pointed out that the F1 measure was not particularly satisfactory and more work would be required to get the classification to a level where it could be used in a production system. The hypothesis remains to be examined that the additional parameters listed in section 2.4.4, "Potential refinement and future work" would lead to better results and a more robust classification system. All materials associated with the present work are available on www.martinwunderlich.comenronresults.

Using a web-based approach for "crowd-sourcing" the manually categorized training data turned out to be straight-forward and successfully allowed gathering of the training data. The methodology developed and applied here could, of course, be applied to other text types, such as newspaper articles or social network contents, and could therefore be applied fruitfully in the area known as "Digital Humanities".

Appendix 1: Algorithms

Simple learning algorithm using delta rule for single-layer preceptrons:

```
  init:  create NN topology, randomly set weights, set learning rate $\eta$
learn:
        while there is another training sample p from P and error too large
                apply p as input to NN
                for each output neuron
                        for each incoming connection i-j for this output neuron
                                calculate difference between teaching input and actual output
                                multiply by learning rate and output from neuron i to get $\Delta w_{i,j}$
                                change weight i by $\Delta w_{i,j}$
                        end for
                end for
        end while
```

Learning algorithm for MLPs using backpropagation:

```
  init:
        create NN topology
        randomly set weights
        set learning rate $\eta$
learn:
        while there is another training sample p from P and error too large
                apply p as input to NN
                run input through NN, calculating all activations and output values
                for each weight j, i (reverse order!)
                        if i is output neuron
                                $\Delta w_{i,j} = f'_{act}(net_h) \cdot (t_h - y_h)$
                        else // input neuron or hidden neuron
                                $\Delta w_{i,j} = f'_{act}(net_h) \cdot \sum_{l \in L}(\delta_l \cdot w_{h,l})$
                        update $w_{i,j}$
                end for
        end while
```

Appendix 2: Examples from the Enron email corpus

---

1. Assigned to category "Company business, strategy, alliances etc.":
(relative path: /arnold-j/all_documents/761.)

Date: Fri, 6 Apr 2001 00:55:00 -0700 (PDT)
From: john.arnold@enron.com
To: kristi.tharpe@intcx.com
Subject: Re: Deal Cancellation
agree

Kristi Tharpe <kristi.tharpe@intcx.com> on 04/06/2001 07:41:11 AM
To: "'john.arnold@enron.com'" <john.arnold@enron.com>, "'jnnelson@duke-energy.com'" <jnnelson@duke-energy.com>
Subject: Deal Cancellation

Please reply to this correspondence to cancel deal id 131585680 between
John Neslon of Duke Energy Trading and Marketing LLC and John Arnold of
Enron North America Corp.
Product: NG Fin, FP for LD1 Henry Hub tailgate - Louisiana
Strip: May01-Oct01
Quantity: 2,500 MMBtus daily
Total Quantity: 460,000 MMBtus
Price/Rate: 5.57 USD / MMBtu
Effective Date: May 1, 2001
Termination Date: October 31, 2001
Please call the ICE Help Desk at 770.738.2101 with any questions.

Thanks,

Kristi Tharpe
IntercontinentalExchange, LLC
2100 RiverEdge Parkway, Fourth Floor
Atlanta, GA 30328
Phone: 770-738-2101
ktharpe@intcx.com

---

2. Assigned to category "Employment arrangements (hiring etc.)":
(relative path: /kaminski-v/all_documents/3175.)

Date: Wed, 15 Nov 2000 02:03:00 -0800 (PST)
From: toni.graham@enron.com
To: vince.kaminski@enron.com
Subject: F/U from Iris Mack, MBA/Phd to Enron
Vince, Iris received an error message when sending this to you so she ask I
forward it to you. She is currently living in California.
———————- Forwarded by Toni Graham/Corp/Enron on 11/15/2000 09:49 AM ———————————
"iris mack" <irismmack@hotmail.com> on 11/14/2000 11:20:15 PM
To: vincent.kaminski@enron.com
Subject: F/U from Iris Mack, MBA/Phd to Enron

Dear Dr. Kaminski,

How are you? I seemed to have lost contact with you. The last contact
was with a Ms. Graham - regarding my coming to Houston for an interview -
while I was still living in London.

I am now back in the states - where I spent the last few months working
for a dot.com. Although this internet opportunity was interesting, it was
not a viable one. I guess part of the technology firm shake out.

Therefore I am now interviewing again - with investment banks and
insurance companies. Is Enron still interested in having me come down for
an interview? If so, please let me know how to proceed.

Thank you in advance for your time and consideration.
Kind regards,
Iris Mack
925.933.7833
–
Get Your Private, Free E-mail from MSN Hotmail at http://www.hotmail.com.
Share information about yourself, create your own public profile at
http://profiles.msn.com.
- CV for Iris Mack, MBA, PhD.doc

---

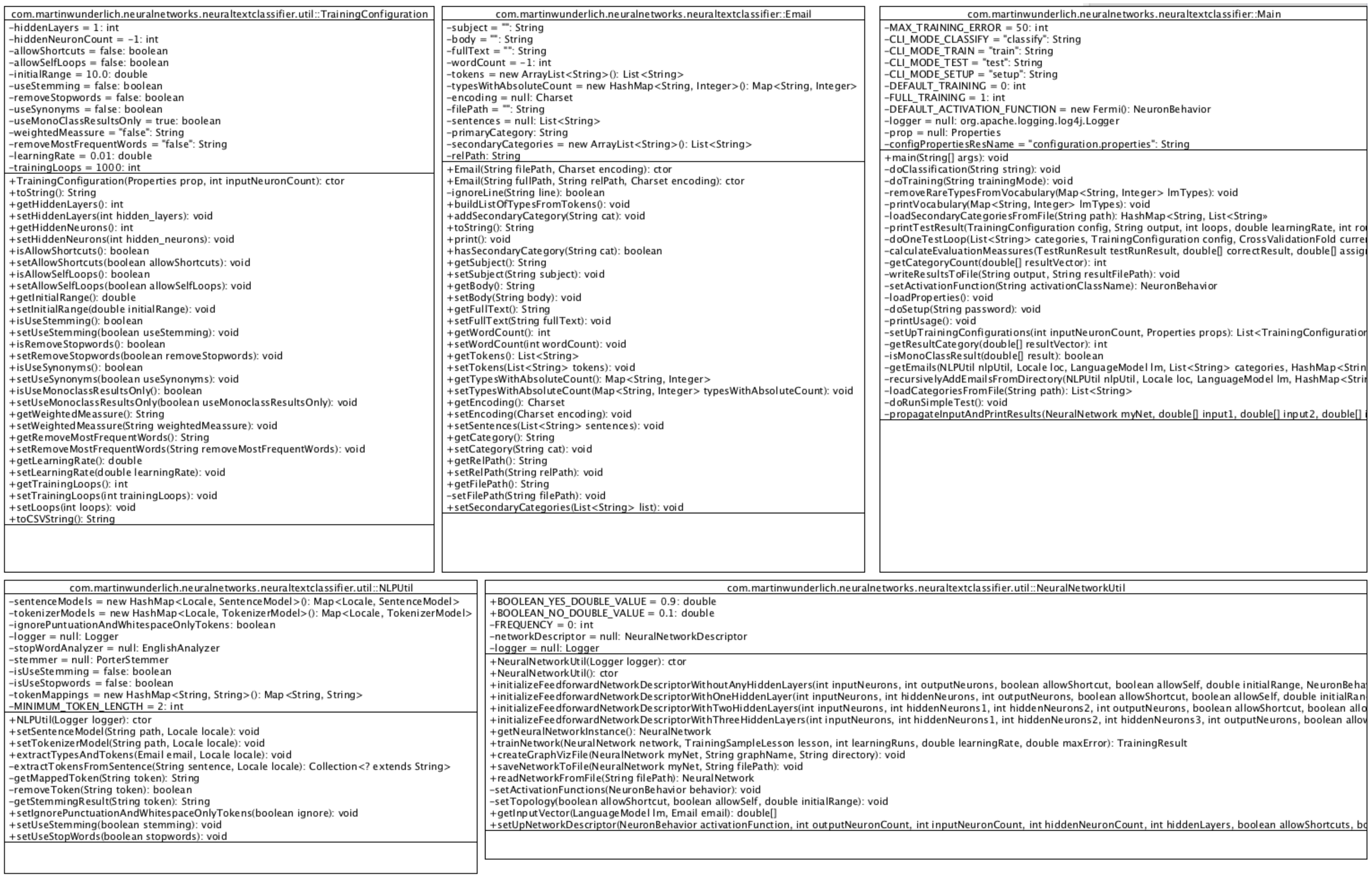
3. Assigned to category "Personal, Friends, Family":
(relative path: /guzman-m/discussion_threads/144.)

Date: Mon, 6 Nov 2000 03:02:00 -0800 (PST)
From: mark.guzman@enron.com
To: katie.trullinger@wfsg.com
Subject: RE:

Letting him stir is good in some ways but guys actually tend to like a lack
of clarity in relationships as it gives them more leeway. However, I don't
really know the dynamic in your relationship at all.

So Angela and I made up from our huge saturday morning fight before I left on
Sunday and she cried like a baby when i left. I feel so bad leaving her. She
is really broken hearted right now.

Appendix 3: Detailed overview of implementation (UML diagrams)



```
com.martinwunderlich.neuralnetworks.neuraltextclassifier.util::TrainingConfiguration
-hiddenLayers = 1: int
-hiddenNeuronCount = -1: int
-allowShortcuts = false: boolean
-allowSelfLoops = false: boolean
-initialRange = 10.0: double
-useStemming = false: boolean
-removeStopwords = false: boolean
-useSynonyms = false: boolean
-useMonoClassResultsOnly = true: boolean
-weightedMeassure = "false": String
-removeMostFrequentWords = "false": String
-learningRate = 0.01: double
-trainingLoops = 1000: int
+TrainingConfiguration(Properties prop, int inputNeuronCount): ctor
+toString(): String
+getHiddenLayers(): int
+setHiddenLayers(int hidden_layers): void
+getHiddenNeurons(): int
+setHiddenNeurons(int hidden_neurons): void
+isAllowShortcuts(): boolean
+setAllowShortcuts(boolean allowShortcuts): void
+isAllowSelfLoops(): boolean
+setAllowSelfLoops(boolean allowSelfLoops): void
+getInitialRange(): double
+setInitialRange(double initialRange): void
+isUseStemming(): boolean
+setUseStemming(boolean useStemming): void
+isRemoveStopwords(): boolean
+setRemoveStopwords(boolean removeStopwords): void
+isUseSynonyms(): boolean
+setUseSynonyms(boolean useSynonyms): void
+isUseMonoclassResultsOnly(): boolean
+setUseMonoclassResultsOnly(boolean useMonoclassResultsOnly): void
+getWeightedMeassure(): String
+setWeightedMeassure(String weightedMeassure): void
+getRemoveMostFrequentWords(): String
+setRemoveMostFrequentWords(String removeMostFrequentWords): void
+getLearningRate(): double
+setLearningRate(double learningRate): void
+getTrainingLoops(): int
+setTrainingLoops(int trainingLoops): void
+setLoops(int loops): void
+toCSVString(): String
```

```
com.martinwunderlich.neuralnetworks.neuraltextclassifier::Email
-subject = "": String
-body = "": String
-fullText = "": String
-wordCount = -1: int
-tokens = new ArrayList<String>(): List<String>
-typesWithAbsoluteCount = new HashMap<String, Integer>(): Map<String, Integer>
-encoding = null: Charset
-filePath = "": String
-sentences = null: List<String>
-primaryCategory: String
-secondaryCategories = new ArrayList<String>(): List<String>
-relPath: String
+Email(String filePath, Charset encoding): ctor
+Email(String fullPath, String relPath, Charset encoding): ctor
-ignoreLine(String line): boolean
+buildListOfTypesFromTokens(): void
+addSecondaryCategory(String cat): void
+toString(): String
+print(): void
+hasSecondaryCategory(String cat): boolean
+getSubject(): String
+setSubject(String subject): void
+getBody(): String
+setBody(String body): void
+getFullText(): String
+setFullText(String fullText): void
+getWordCount(): int
+setWordCount(int wordCount): void
+getTokens(): List<String>
+setTokens(List<String> tokens): void
+getTypesWithAbsoluteCount(): Map<String, Integer>
+setTypesWithAbsoluteCount(Map<String, Integer> typesWithAbsoluteCount): void
+getEncoding(): Charset
+setEncoding(Charset encoding): void
+setSentences(List<String> sentences): void
+getCategory(): String
+setCategory(String cat): void
+getRelPath(): String
+setRelPath(String relPath): void
+getFilePath(): String
-setFilePath(String filePath): void
+setSecondaryCategories(List<String> list): void
```

```
com.martinwunderlich.neuralnetworks.neuraltextclassifier::Main
-MAX_TRAINING_ERROR = 50: int
-CLI_MODE_CLASSIFY = "classify": String
-CLI_MODE_TRAIN = "train": String
-CLI_MODE_TEST = "test": String
-CLI_MODE_SETUP = "setup": String
-DEFAULT_TRAINING = 0: int
-FULL_TRAINING = 1: int
-DEFAULT_ACTIVATION_FUNCTION = new Fermi(): NeuronBehavior
-logger = null: org.apache.logging.log4j.Logger
-prop = null: Properties
-configPropertiesResName = "configuration.properties": String
+main(String[] args): void
-doClassification(String string): void
-doTraining(String trainingMode): void
-removeRareTypesFromVocabulary(Map<String, Integer> lmTypes): void
-printVocabulary(Map<String, Integer> lmTypes): void
-loadSecondaryCategoriesFromFile(String path): HashMap<String, List<String»
-printTestResult(TrainingConfiguration config, String output, int loops, double learningRate, int ro
-doOneTestLoop(List<String> categories, TrainingConfiguration config, CrossValidationFold curre
-calculateEvaluationMeassures(TestRunResult testRunResult, double[] correctResult, double[] assig
-getCategoryCount(double[] resultVector): int
-writeResultsToFile(String output, String resultFilePath): void
-setActivationFunction(String activationClassName): NeuronBehavior
-loadProperties(): void
-doSetup(String password): void
-printUsage(): void
-setUpTrainingConfigurations(int inputNeuronCount, Properties props): List<TrainingConfiguratio
-getResultCategory(double[] resultVector): int
-isMonoClassResult(double[] result): boolean
-getEmails(NLPUtil nlpUtil, Locale loc, LanguageModel lm, List<String> categories, HashMap<Strin
-recursivelyAddEmailsFromDirectory(NLPUtil nlpUtil, Locale loc, LanguageModel lm, HashMap<Stri
-loadCategoriesFromFile(String path): List<String>
-doRunSimpleTest(): void
-propagateInputAndPrintResults(NeuralNetwork myNet, double[] input1, double[] input2, double[] i
```

```
com.martinwunderlich.neuralnetworks.neuraltextclassifier.util::NLPUtil
-sentenceModels = new HashMap<Locale, SentenceModel>(): Map<Locale, SentenceModel>
-tokenizerModels = new HashMap<Locale, TokenizerModel>(): Map<Locale, TokenizerModel>
-ignorePuntuationAndWhitespaceOnlyTokens: boolean
-logger = null: Logger
-stopWordAnalyzer = null: EnglishAnalyzer
-stemmer = null: PorterStemmer
-isUseStemming = false: boolean
-isUseStopwords = false: boolean
-tokenMappings = new HashMap<String, String>(): Map<String, String>
-MINIMUM_TOKEN_LENGTH = 2: int
+NLPUtil(Logger logger): ctor
+setSentenceModel(String path, Locale locale): void
+setTokenizerModel(String path, Locale locale): void
+extractTypesAndTokens(Email email, Locale locale): void
-extractTokensFromSentence(String sentence, Locale locale): Collection<? extends String>
-getMappedToken(String token): String
-removeToken(String token): boolean
-getStemmingResult(String token): String
+setIgnorePunctuationAndWhitespaceOnlyTokens(boolean ignore): void
+setUseStemming(boolean stemming): void
+setUseStopWords(boolean stopwords): void
```

```
com.martinwunderlich.neuralnetworks.neuraltextclassifier.util::NeuralNetworkUtil
+BOOLEAN_YES_DOUBLE_VALUE = 0.9: double
+BOOLEAN_NO_DOUBLE_VALUE = 0.1: double
-FREQUENCY = 0: int
-networkDescriptor = null: NeuralNetworkDescriptor
-logger = null: Logger
+NeuralNetworkUtil(Logger logger): ctor
+NeuralNetworkUtil(): ctor
+initializeFeedforwardNetworkDescriptorWithoutAnyHiddenLayers(int inputNeurons, int outputNeurons, boolean allowShortcut, boolean allowSelf, double initialRange, NeuronBeha
+initializeFeedforwardNetworkDescriptorWithOneHiddenLayer(int inputNeurons, int hiddenNeurons, int outputNeurons, boolean allowShortcut, boolean allowSelf, double initialRan
+initializeFeedforwardNetworkDescriptorWithTwoHiddenLayers(int inputNeurons, int hiddenNeurons1, int hiddenNeurons2, int outputNeurons, boolean allowShortcut, boolean allo
+initializeFeedforwardNetworkDescriptorWithThreeHiddenLayers(int inputNeurons, int hiddenNeurons1, int hiddenNeurons2, int hiddenNeurons3, int outputNeurons, boolean allow
+getNeuralNetworkInstance(): NeuralNetwork
+trainNetwork(NeuralNetwork network, TrainingSampleLesson lesson, int learningRuns, double learningRate, double maxError): TrainingResult
+createGraphVizFile(NeuralNetwork myNet, String graphName, String directory): void
+saveNetworkToFile(NeuralNetwork myNet, String filePath): void
+readNetworkFromFile(String filePath): NeuralNetwork
-setActivationFunctions(NeuronBehavior behavior): void
-setTopology(boolean allowShortcut, boolean allowSelf, double initialRange): void
+getInputVector(LanguageModel lm, Email email): double[]
+setUpNetworkDescriptor(NeuronBehavior activationFunction, int outputNeuronCount, int inputNeuronCount, int hiddenNeuronCount, int hiddenLayers, boolean allowShortcuts, bo
```

Figure 15: UML class diagram 1

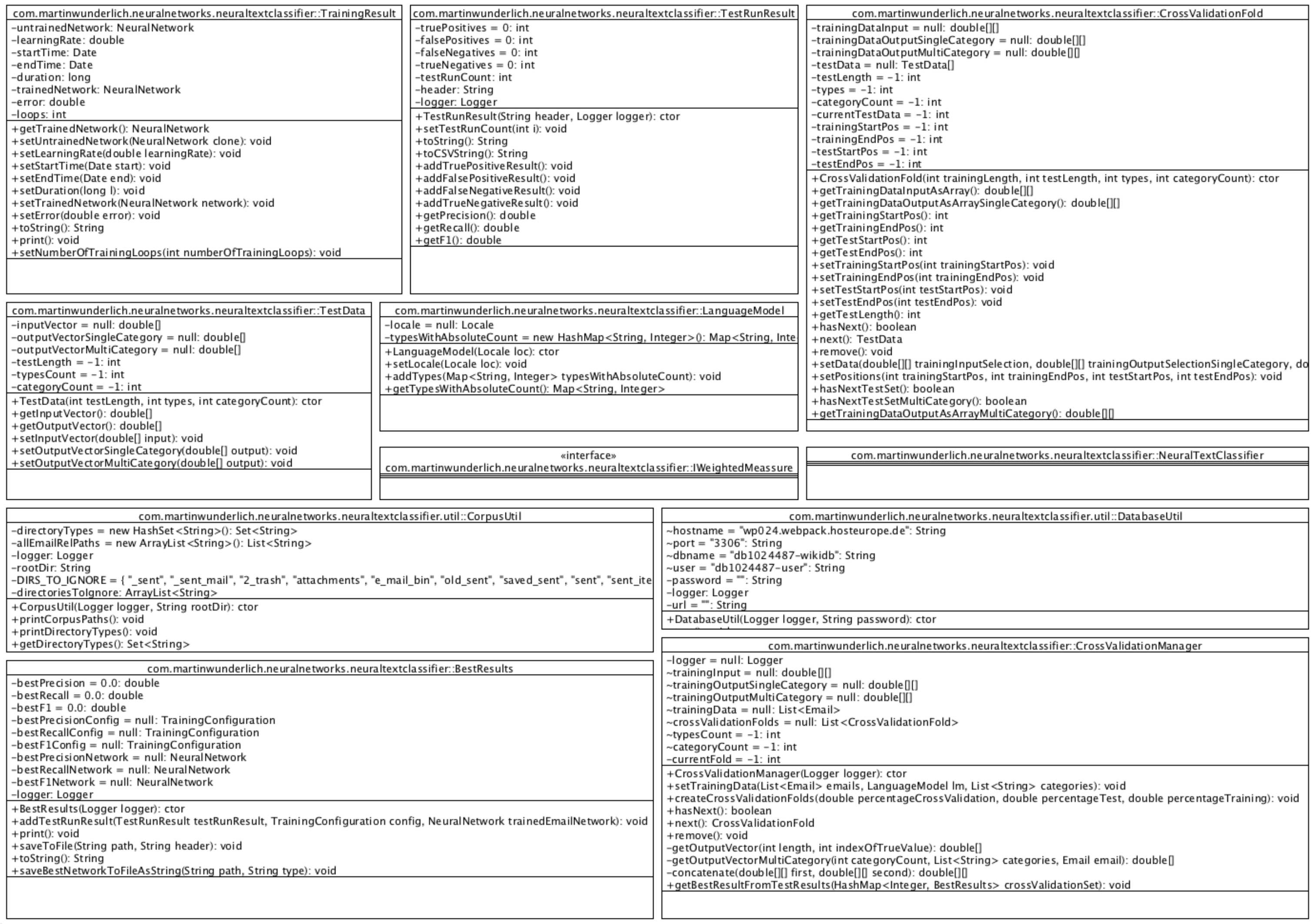

com.martinwunderlich.neuralnetworks.neuraltextclassifier::TrainingResult
-untrainedNetwork: NeuralNetwork
-learningRate: double
-startTime: Date
-endTime: Date
-duration: long
-trainedNetwork: NeuralNetwork
-error: double
-loops: int
+getTrainedNetwork(): NeuralNetwork
+setUntrainedNetwork(NeuralNetwork clone): void
+setLearningRate(double learningRate): void
+setStartTime(Date start): void
+setEndTime(Date end): void
+setDuration(long l): void
+setTrainedNetwork(NeuralNetwork network): void
+setError(double error): void
+toString(): String
+print(): void
+setNumberOfTrainingLoops(int numberOfTrainingLoops): void
com.martinwunderlich.neuralnetworks.neuraltextclassifier::TestRunResult
-truePositives = 0: int
-falsePositives = 0: int
-falseNegatives = 0: int
-trueNegatives = 0: int
-testRunCount: int
-header: String
-logger: Logger
+TestRunResult(String header, Logger logger): ctor
+setTestRunCount(int i): void
+toString(): String
+toCSVString(): String
+addTruePositiveResult(): void
+addFalsePositiveResult(): void
+addFalseNegativeResult(): void
+addTrueNegativeResult(): void
+getPrecision(): double
+getRecall(): double
+getF1(): double
com.martinwunderlich.neuralnetworks.neuraltextclassifier::CrossValidationFold
-trainingDataInput = null: double[][]
-trainingDataOutputSingleCategory = null: double[][]
-trainingDataOutputMultiCategory = null: double[][]
-testData = null: TestData[]
-testLength = -1: int
-types = -1: int
-categoryCount = -1: int
-currentTestData = -1: int
-trainingStartPos = -1: int
-trainingEndPos = -1: int
-testStartPos = -1: int
-testEndPos = -1: int
+CrossValidationFold(int trainingLength, int testLength, int types, int categoryCount): ctor
+getTrainingDataInputAsArray(): double[][]
+getTrainingDataOutputAsArraySingleCategory(): double[][]
+getTrainingStartPos(): int
+getTrainingEndPos(): int
+getTestStartPos(): int
+getTestEndPos(): int
+setTrainingStartPos(int trainingStartPos): void
+setTrainingEndPos(int trainingEndPos): void
+setTestStartPos(int testStartPos): void
+setTestEndPos(int testEndPos): void
+getTestLength(): int
+hasNext(): boolean
+next(): TestData
+remove(): void
+setData(double[][] trainingInputSelection, double[][] trainingOutputSelectionSingleCategory, do
+setPositions(int trainingStartPos, int trainingEndPos, int testStartPos, int testEndPos): void
+hasNextTestSet(): boolean
+hasNextTestSetMultiCategory(): boolean
+getTrainingDataOutputAsArrayMultiCategory(): double[][]
com.martinwunderlich.neuralnetworks.neuraltextclassifier::TestData
-inputVector = null: double[]
-outputVectorSingleCategory = null: double[]
-outputVectorMultiCategory = null: double[]
-testLength = -1: int
-typesCount = -1: int
-categoryCount = -1: int
+TestData(int testLength, int types, int categoryCount): ctor
+getInputVector(): double[]
+getOutputVector(): double[]
+setInputVector(double[] input): void
+setOutputVectorSingleCategory(double[] output): void
+setOutputVectorMultiCategory(double[] output): void
com.martinwunderlich.neuralnetworks.neuraltextclassifier::LanguageModel
-locale = null: Locale
-typesWithAbsoluteCount = new HashMap<String, Integer>(): Map<String, Inte
+LanguageModel(Locale loc): ctor
+setLocale(Locale loc): void
+addTypes(Map<String, Integer> typesWithAbsoluteCount): void
+getTypesWithAbsoluteCount(): Map<String, Integer>
«interface»
com.martinwunderlich.neuralnetworks.neuraltextclassifier::IWeightedMeassure
com.martinwunderlich.neuralnetworks.neuraltextclassifier::NeuralTextClassifier
com.martinwunderlich.neuralnetworks.neuraltextclassifier.util::CorpusUtil
-directoryTypes = new HashSet<String>(): Set<String>
-allEmailRelPaths = new ArrayList<String>(): List<String>
-logger: Logger
-rootDir: String
-DIRS_TO_IGNORE = { "_sent", "_sent_mail", "2_trash", "attachments", "e_mail_bin", "old_sent", "saved_sent", "sent", "sent_ite
-directoriesToIgnore: ArrayList<String>
+CorpusUtil(Logger logger, String rootDir): ctor
+printCorpusPaths(): void
+printDirectoryTypes(): void
+getDirectoryTypes(): Set<String>
com.martinwunderlich.neuralnetworks.neuraltextclassifier.util::DatabaseUtil
~hostname = "wp024.webpack.hosteurope.de": String
~port = "3306": String
~dbname = "db1024487-wikidb": String
~user = "db1024487-user": String
-password = "": String
-logger: Logger
-url = "": String
+DatabaseUtil(Logger logger, String password): ctor
com.martinwunderlich.neuralnetworks.neuraltextclassifier::BestResults
-bestPrecision = 0.0: double
-bestRecall = 0.0: double
-bestF1 = 0.0: double
-bestPrecisionConfig = null: TrainingConfiguration
-bestRecallConfig = null: TrainingConfiguration
-bestF1Config = null: TrainingConfiguration
-bestPrecisionNetwork = null: NeuralNetwork
-bestRecallNetwork = null: NeuralNetwork
-bestF1Network = null: NeuralNetwork
-logger: Logger
+BestResults(Logger logger): ctor
+addTestRunResult(TestRunResult testRunResult, TrainingConfiguration config, NeuralNetwork trainedEmailNetwork): void
+print(): void
+saveToFile(String path, String header): void
+toString(): String
+saveBestNetworkToFileAsString(String path, String type): void
com.martinwunderlich.neuralnetworks.neuraltextclassifier::CrossValidationManager
-logger = null: Logger
~trainingInput = null: double[][]
~trainingOutputSingleCategory = null: double[][]
~trainingOutputMultiCategory = null: double[][]
~trainingData = null: List<Email>
~crossValidationFolds = null: List<CrossValidationFold>
~typesCount = -1: int
~categoryCount = -1: int
~currentFold = -1: int
+CrossValidationManager(Logger logger): ctor
+setTrainingData(List<Email> emails, LanguageModel lm, List<String> categories): void
+createCrossValidationFolds(double percentageCrossValidation, double percentageTest, double percentageTraining): void
+hasNext(): boolean
+next(): CrossValidationFold
+remove(): void
-getOutputVector(int length, int indexOfTrueValue): double[]
-getOutputVectorMultiCategory(int categoryCount, List<String> categories, Email email): double[]
-concatenate(double[][] first, double[][] second): double[][]
+getBestResultFromTestResults(HashMap<Integer, BestResults> crossValidationSet): void

Figure 16: UML class diagram 2

References

[AZK09] Taiwo Ayodele, Shikun Zhou, and Rinat Khusainov. Email classification: Solution with back propagation technique. In *ICITST*, pages 1–6. IEEE, 2009.

[BDV00] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *NIPS*, pages 932–938. MIT Press, 2000.

[BMH04] R. Bekkerman, A. McCallum, and G. Huang. Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora. CIIR Technical Report, IR-418, University of Massachusetts, Amherst, USA, 2004.

[Gra92] Adolf Grauel. *Neuronale Netze : Grundlagen und mathematische Modellierung.* BI-Wiss.-Verl., Mannheim u.a., 1992.

[HC10] Matthijs Hovelynck and Boris Chidlovskii. Multi-modality in one-class classification. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 441–450, New York, NY, USA, 2010. ACM.

[ISO$^+$10] I.S. Isa, Z. Saad, S. Omar, M.K. Osman, K.A. Ahmad, and H.A. Mat Sakim. Suitable mlp network activation functions for breast cancer and thyroid disease detection. *Computational Intelligence, Modelling and Simulation, International Conference on*, 0:39–44, 2010.

[JM08] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence).* Prentice Hall, 2 edition, 2008.

[Kaw00] K. Kawaguchi. *A Multithreaded Software Model for Backpropagation Neural Network Applications.* University of Texas at El Paso, 2000.

[Kri07] David Kriesel. *Ein kleiner Überblick über Neuronale Netze.* 2007. erhältlich auf http://www.dkriesel.com, last accessed 2014-01-06.

[KY04] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. *Machine Learning: ECML 2004*, pages 217–226, 2004.

[Mac13] T. MacFarlane. Extracting semantics from the enron corpus. Other, Bath, U. K., November 2013. Undergraduate Dissertation.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[MDP$^+$11] Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukas Burget, and Jan Cernocký. Strategies for training large scale neural network language models. In David Nahamoo and Michael Picheny, editors, *ASRU*, pages 196–201. IEEE, 2011.

[MFJP09] Justin Martineau, Tim Finin, Anupam Joshi, and Shamit Patel. Improving binary classification on text problems using differential word features. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 2019–2024, New York, NY, USA, 2009. ACM.

[MKB$^+$10] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*, pages 1045–1048. ISCA, 2010.

[MP69] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[Pao89] Yoh-Han Pao. *Adaptive pattern recognition and neural networks.* Addison-Wesley, Reading, Mass., USA et al., 1989.

[Rit91] Helge Ritter. *Neuronale Netze.* Addison-Wesley, Bonn [et. al.], 1991.

[RM86] David E. Rumelhart and James L. McClelland, editors. *Parallel distributed processing : explorations in the microstructure of cognition.* Computational models of cognition and perception. MIT Press, Cambridge, Mass. [u.a.], 1986.

[Ros58] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[Ros62] Frank Rosenblatt. *Principles of neurodynamics; perceptrons and the theory of brain mechanisms.* Spartan Books, Washington, DC, USA, 1962.

[RZBK98] M. Rosen-Zvi, M. Biehl, and I. Kanter. Learnability of periodic activation functions: General results. *Physical Review E*, 58(3):3606–3609, 1998.

[SG09] Marina Sokolova and Lapal Guy. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage.*, 45:427–437, July 2009.

[Ste13] Pontus Stenetorp. Transition-based dependency parsing using recursive neural networks. http://pontus.stenetorp.se/res/pdf/stenetorp2013transition.pdf, 2013.

[TKB12] Muhammad Atif Tahir, Josef Kittler, and Ahmed Bouridane. Multilabel classification using heterogeneous ensemble of multi-label classifiers. *Pattern Recognition Letters*, 33(5):513–523, 2012.

[WHJ10] Man Wang, Yifan He, and Minghu Jiang. Text categorization of enron email corpus based on information bottleneck and maximal entropy. pages 2472–2475, 2010.

[YYC11] Shinjae Yoo, Yiming Yang, and Jaime Carbonell. Modeling personalized email prioritization: Classification-based and regression-based approaches. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 729–738, New York, NY, USA, 2011. ACM.

[Zau99] Detlef P. Zaun. *Künstliche neuronale Netze und Computerlinguistik.* MIT Press, Cambridge, Mass. [u.a.], 1999.

[ZZ06] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Trans. Knowl. Data Eng.*, 18(10):1338–1351, 2006.