Solution based on a truly online machine learning

GENPACT EMAIL CLASSIFICATION IDEATION CONTEST STATEMENT

yedtoss

March 23, 2016

The client, "Genpack" has a lot of emails that they need to classify into categories so that it can be handled by the correct team. Right now, they are doing it manually. The objective of the client for this contest is to perform an automatic email classification to avoid this manual and time consuming process.

In the scientific literature, this is known as "document categorization", "text classification", "email classification" or "email foldering". In this document, we described the general techniques used by all standard state-of-the-art algorithms. We then proposed a specific algorithm that can be used by the client together with some recommendations. Finally, we presented software packages that can be used for the implementation.

1 General strategy for email classification

Figure 1: General email foldering strategy

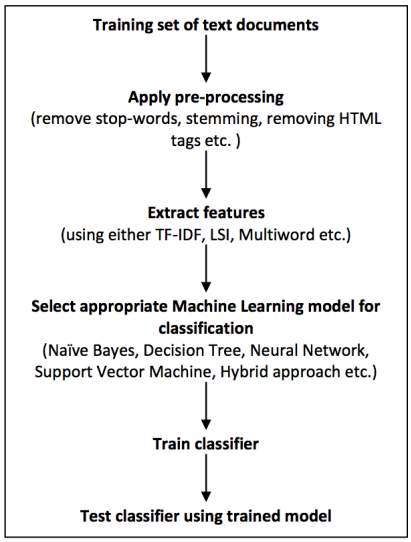

Figure 1 presents the general technique used to perform email classification. We will describe below the pre-processing, the features extraction as well as the classification steps.

1.1 Pre-Processing

In a first step, the emails need to be processed to remove any useless information. This includes different processes explained below.

1.1.1 Tokenization

Tokenization is the process of breaking up a stream of text into words, symbols, or other meaningful elements called tokens. In its basic form, token can be identified as whitespace separated words for

English. However, not every language separate words by space (e.g. Chinese, Japanese) and even for English, white space alone is not sufficient. For this reason, it is recommended to use better algorithms which are implemented in NLTK or Pattern [15] for example.

1.1.2 Stop Words Removal

Many of the most frequently used words in English (or other languages) are useless in Information Retrieval (IR) and email categorization. Those words are called "Stop words". Stop-words, which are language-specific functional words, are frequent words that carry no information. For example, it can include pronouns, prepositions or conjunctions ("the", "of", "and", "to" for example). One of the first step during text categorization is to remove those words. One of the most complete list of stop words for English and other languages is provided by Savoy [14].

1.1.3 Stemming

Stemming techniques are used to find out the root of a word. Behind stemming, the hypothesis is that words with the same root mostly describe the same or relatively close concepts in text. For example, the words "user", "users", "used", "using" can all be stemmed to the word 'USE'. One of the most used stemming algorithms for English is the Porter stemmer [10]. A list of stemmer implementations in C or Java for various languages are provided by Savoy [14]. Also, an easy to use python implementation is available in the NLTK module nltk.stem.snowball [1].

1.1.4 Lemmatization

Lemmatization has the same goal as stemming which is to convert words to their root forms. However, they differ in the techniques used to achieve this conversion. Where stemming used an heuristic, lemmatization tries to do things properly with the use of a vocabulary and morphological analysis of words. So, lemmatization leads to a better root although the algorithm is usually slower than a stemmer. For example, the words "caring" and "cars" in English would be reduced to "car" in a stemming process whereas in a lemmatization process they would be replaced by "care" and "car" respectively. To properly work, a lemmatizer needs to know the context of the word. For example, if confronted with the word "saw", lemmatization would attempt to return either "see" or "saw" depending on whether the use of the token was as a verb or a noun. This context is provided by a process called part of speech tagging. A lemmatizer for English is available in Spacy [2] and NLTK [1]. Others languages lemmatizer are available in Pattern [15] from CLIPS.

1.1.5 Part of Speech Tagging

A Part-Of-Speech Tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. Examples of part of speech includes noun, verb, adjective although generally computational applications use more fine-grained tags such as 'noun-plural'. A Part of Speech Tagger for English is available in Spacy [2] and NLTK [1]. Others languages tagger are available in Pattern [15] from CLIPS or Stanford Tagger [16].

1.2 Features Extraction

Once preprocessed, the email need to be converted to a vector representation called features. Most text categorization algorithms used a Bag-of-words (BOW), Bag-n-grams or its derivatives (term frequency–inverse document frequency aka tf-idf . . .). In BOW, an email is represented by the count of his words. Additionally, binary (presence/absence of a word from the vocabulary) is used. With BOW the word order is lost. Even though bag-of-n-grams considers the word order in short context, it suffers from data sparsity and high dimensionality. Bag-of-words and bag-of-n-grams have very little sense about the semantics of the words or more formally the distances between the words. This means that words "powerful," "strong" and "Paris" are equally distant despite the fact that semantically, "powerful" should be closer to "strong" than "Paris."

This is why a better representation for documents called *Paragraph Vector* [8] have been derived. It's goal is to transform text of arbitrary length to a vector of fixed size such that two documents with the same meaning or topics are closer according to a similarity measure called cosine distance.

1.3 Selecting a Classifier

Once the relevant features are extracted, a learning algorithm is applied to detect the category. Popular algorithms for text categorization are Support Vector Machines (SVM), Naives Bayes, K-Nearest Neighbor .... In the literature, SVM is believed to give generally the best performance. However, some authors has pointed that if correctly tuned other algorithms will reach the same accuracy as SVM. The issues with SVM is that it is computationally intensive, does not work well with imbalance class (when there are more emails in one categories than others) and when the number of features increases. Furthermore, it can't handle feedback on classification errors without a full re-training. This is why online classification methods have been developed. An online classifier is a method of machine learning which updates his prediction as data come in a sequential order. One of the state-of-the-art online classifier is the Exact Soft Confidence Weighed (SCW) [7] which beats SVM in many experiments.

1.3.1 Exact Soft Confidence Weighed (SCW [7])

SCW [7] classifier works by maintaining a multivariate Gaussian Distribution with a mean vector μ and a covariance matrix Σ whose dimension is the number of features. After seeing one training example, SCW will predict the category of this example. To Predict the category of an example of features x, a weight vector w is drawn from the Gaussian Distribution. Then, the sign of the dot product between w and x is returned for binary classification. For multi categories classification, we can use the One vs One, One vs All or the Gibbs sampling proposed in [3].

2 Algorithm Description

For each email, the subject and the body of the messages will be used. All others fields such as sender, receiver or attachments will be discarded. The following process should be done for the subject and body merged. We will simply refer to the merged version as *text*.

2.1 Preprocessing

The preprocessing steps consist of the following:

1. Remove all HTML tags from the text using BeautifulSoup [13].
2. Detect the language of the text using langdetect [5]. All the remaining steps depends on the language found.
3. Convert all foreign characters or words to that language into their canonical form or remove them.
4. Convert the text to lowercase if the language has the concept of lower and upper case.
5. Tokenize the text into a vector of words/tokens using its language as parameter. Do not change the original order of the words using Pattern [15].
6. Perform Part of Speech tagging using the language and the vector of words using Pattern [15].
7. For each words in the vector convert it to its lemma using a Lemmatizer(e.g. Pattern [15]).
8. Remove stops words from the vectors and making sure the original order of words is kept using NLTK [1].

2.2 Features extraction

At the end of the preprocessing steps of section 2.1, we have a vector of words. We can transform it back to a string or sentence.

Now, convert this sentence to a vector of fixed length using the *ParagraphVector* model with Gensim [11] module gensim.models.doc2vec.Doc2Vec. Then, update the *ParagraphVector* model according to section with Gensim function gensim.models.doc2vec.Doc2Vec.train.

2.3 Classification

At the end of the features extraction step described in section 2.2, we have a vector of features. Classify the text using the Exact Soft Confidence Weighed algorithm [7] and update it using this email. Now if we receive immediately the true or reference category of this text, we can update our algorithm model according to section 2.4. However, in practice we might not have the feedback immediately due to the fact that the user has not yet open the email or has not classified it. In this case, once we received the feedback at a later time, we need to first re-classify the email and update the algorithm only in case of error.

We recommend a custom implementation for this module with the L1-hinge loss, single constraint, and diagonal matrix according to [7]. However an implementation is available in [12].

2.4 Updating the model and classifier

Note that we can update the *ParagraphVector* model without receiving any feedback because it does not need to know about the (true) category of the text. Also, we don't need the update of the *ParagraphVector* model or SCW classifier to complete before executing potential further steps needed to return a result to the user. When updating SCW we only change the mean μ and covariance Σ is updated in case of mis-classification according to Proposition 1 in Hoi et al. [7].

2.5 Pre-training the model and classifier

Although, the *ParagraphVector* model and *SCW* classifier can both be used without pre-training and will learn as new emails are coming, we recommend to perform a one time large scale pre-training using Gensim for the former. The input emails or documents need to be preprocessed according to section 2.1 and the features generated as explained in section 2.2. This will allow the system to perform well from the beginning.

2.6 Dealing with removed or new categories

The *ParagraphVector* model does not need any special treatment or any modification in case a new category is added or removed because it is unsupervised and does not depend on categories. This is not the case for the *SCW* classifier for which we propose the following solutions:

- When using the Gibbs sampling to handle multiclass classification, no changes is needed.
- When using the One vs One or One vs All to handle multiclass classification, train the added models due to the new category or removed the obsolete models corresponding the the deleted category.
- If the categories added or removed are significant behavioral changes in the classification of the emails (For example, lot of previously classified emails will change categories) it is recommended to completely re-train the classifier to get good results from the beginning of this significant change.

2.7 Dealing with multiple languages

Although, a single *ParagraphVector* model and *SCW* classifier can be used with no modifications on different languages, we have not seen any use case in the literature. We believe that doing so, will

decrease the accuracy of the overall algorithm for each language. Our intuition is confirmed by some works stating that the vector representation of a word using the *ParagraphVector* model in two different languages are related by a linear combination However, practical experiments still need to be done to infirm or confirm this intuition.

So at this stage, our recommendation is to use one *ParagraphVector* model and *SCW* classifier pair for each languages. Each pair can be hosted on the same Salesforce.com instance belonging to a single client.

2.8 Dealing with multiple clients

For the *ParagraphVector* model it is perfectly valid to use the same model for all clients. However we are not sure whether or not using the same model will increase accuracy or decrease it. Indeed, using one model per client will mean that the model will understand more about the jargon of this client which can improve performance. However, by using the same model for all clients the *ParagraphVector* will have a larger training set which and c the jargon of other clients can also improve performance. Our intuition here is to recommend using one *ParagraphVector* model for all clients. We will even recommend to train the *ParagraphVector* model on external data such as Wikipedia or publicly available emails collections. This intuition is backed by the fact that most experiments on *ParagraphVector* model used a lot of external training data. Again, practical experiments still need to be done to infirm or confirm this intuition.

Due to the fact that each client can have different meaning for the categories or even completely different categories, we recommend to use one *SCW* classifier per client.

2.9 Hosting the algorithm

There are two different possibilities of hosting the algorithm per client:

- Hosting the algorithm of each client on its Saleforce.com instance
- Hosting all algorithms on a mother Salesforce.com instance. Then, implement a REST API to expose its functionality to hosts Saleforce.com instance for individual client.

2.10 Dealing with very long email messages

Although, the proposed algorithm will work just fine on a full long email with no penalty in speed, we believe that classification accuracy can be improved by splitting the long emails into parts of fixed size and classifying each part independently. The final classification can be a Majority Vote for example. This intuition is inspired by the fact that some emails might be a multi-categories email where different part belongs to different categories. Also, doing this would help with email containing text in multiple languages.

2.11 Miscellaneous and Disadvantages

- The proposed algorithm ignores the possibility that emails can contain mistakes, misspelling errors or abbreviations. It would be interesting to check if correcting the errors or abbreviations in an email will improve the classification results or not.
- The proposed solution also ignores the To Field, From Field and other meta data available in emails. It will be interesting to explore solutions to handle these fields in the classification using for example the SIMOVERLAP and SIMSUBSET algorithms [6].

2.12 Scalability, Advantages and Past Uses cases

The two main algorithms used are the *ParagraphVector* model and *SCW* classifier.

The main advantages of *ParagraphVector* are:

- It has been used by Google for classification and sentiment analysis with a huge accuracy improvement over bag of words.
- It can be distributed.
- It has been trained with a database of gigabyte (Wikipedia) and is fast.
- It is an unsupervised algorithm which do not need labeled emails.
- It does not need to keep the whole documents in memory and is an online algorithm.
- All these can be verified in [8] and [4].

More descriptions are available in section 1.
The main advantages of *SCW* classifier are:

- It has been used for text and emails categorization and show accuracy improvement over Support Vector Machines.
- It is significantly faster and only keep a memory linear to the number of features and not the number of documents
- All these can be verified in [7] and [9].

More descriptions are available in section 1.

3 Software and Libraries

All the softwares we proposed are open source and are described in the references section. It is Pattern [15], Gensim [11], NLTK [1], BeautifulSoup [13] and [5]. We also proposed some alternative implementations described in section 1.

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009. URL `http://www.nltk.org/`. [Version 3.0, Apache 2.0 License].

[2] SYLLOGISM CO. Spacy, Industrial strength NLP with Python and Cython. `https://spacy.io/`, 2015. [Version v0.100, License: MIT].

[3] Koby Crammer, Mark Dredze, and Alex Kulesza. Multi-Class Confidence Weighted Algorithms. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 496–504, Singapore, August 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology-new/D/D09/D09-1052.bib`.

[4] Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document embedding with paragraph vectors. *CoRR*, abs/1507.07998, 2015. URL `http://arxiv.org/abs/1507.07998`.

[5] Michal Danilak. langdetect. `https://pypi.python.org/pypi/langdetect`, 2015. [Version 1.0.5, License: Apache v2].

[6] Mark Dredze, Tessa Lau, and Nicholas Kushmerick. Automatically classifying emails into activities. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 70–77, New York, NY, USA, 2006. ACM. ISBN 1-59593-287-9. doi: http://doi.acm.org/10.1145/1111449.1111471. URL `http://portal.acm.org/citation.cfm?id=1111449.1111471`.

[7] Steven C. H. Hoi, Jialei Wang, and Peilin Zhao. Exact soft confidence-weighted learning. In *ICML*. icml.cc / Omnipress, 2012. URL `http://dblp.uni-trier.de/db/conf/icml/icml2012.html#HoiWZ12`.

[8] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196. JMLR Workshop and Conference Proceedings, 2014. URL `http://jmlr.org/proceedings/papers/v32/le14.pdf`.

[9] Hung Ngo, Matthew Luciw, Ngo Anh Vien, and Jürgen Schmidhuber. Upper confidence weighted learning for efficient exploration in multiclass prediction with binary feedback, 2013.

[10] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980. doi: 10.1108/eb046814.

[11] Radim Rehurek. Gensim, Software Framework for Topic Modelling with Large Corpora. `https://radimrehurek.com/gensim`, 2015. [Version 0.12.4, License: LGPL].

[12] Radim Rehurek. Hivemall: Hive scalable machine learning library. `https://github.com/myui/hivemall`, 2015. [Version 0.11, License: Apache v2].

[13] Leonard Richardson. Beautiful Soup. `http://www.crummy.com/software/BeautifulSoup/`, 2015. [Version 4.4.1, License: MIT].

[14] Jacques Savoy. IR Multilingual Resources at UniNE. `http://members.unine.ch/jacques.savoy/clef/index.html`, 2005. [Online; last accessed 22-March-2016].

[15] Tom De Smedt and Walter Daelemans. Pattern for python. *Journal of Machine Learning Research*, 13:2063–2067, 2012. URL `http://www.clips.ua.ac.be/pattern`. [Version 3.6, License BSD].

[16] Stanford. Stanford Log-linear Part-Of-Speech Tagger. `http://nlp.stanford.edu/software/tagger.html`, 2015. [Version 3.6.0, Dual License:, GNU full GPL v2 for open source software and commercial license for closed source software].