General text classifier (Project specification)

Erik Lux

28^{th} October 2011

Basic information

The main goal of this work is to implement a general algorithm classifying an input text into categories, constructed by a user. The category is created according to the examples of documents containing an appropriate material describing its content. It is in the competence of the user to provide such examples.

General description

The project is based on creating a library, applicable to the variety of different texts(e.g. including newspaper articles). The library should recognize given text and categorise it correctly due to its knowledge. The most essential part of the library includes way of information retrieval, considering learning from examples and testing its capability. It will be built upon the Nave Bayes classifier, a simple and quite accurate probabilistic classifier applying Bayes' theorem. [5]

Project application

Moreover, the resultant project forms a library application, designed to classify certain texts. The application, implemented as a plug-in project, could be usable by an open-source email client. It is developed in a way to categorise incoming mail.

Technical documentation

Project design

The Project is divided into two parts. The first part is a general library, where the main classification program is implemented. And the second - application part, which calls the library by certain methods. The methods include creating a new category, training the category, testing the category and categorising a document into the category.

Text conversion

Documents could come in a lot of different forms such as a plain text, html documents, xml documents, email documents,... Classification functions work only with plain text documents. Therefore, the library has an internal package for text conversion. The conversion is called through the main converting function which recognises the type of a document and convert its content into plain text. There are three converters implemented in the package. The html, the xml converters, and the email converter. However, the package could be easily extended by adding another converter class. The process of the conversion returns a new data type, containing metadata and a plain text document, extracted from the content of the text.

Metadata

The conversion tool extracts not only the plain text but basic information about the document as well. This operation remains quite important as it stores the metadata of the text for the informational or any other purposes. The data are represented as dictionary or a hashmap of a key-value pair. The main key Type shows the type of the document and other keys are actually derived from the type of the document. For example, an html document contains a key *Title*. Therefore, an email document would have a key, named *From*, for instance.

Data preprocessing

A string vector is expected as a classification program input. Therefore, an extremely high dimensionality of text data could flow through the program. However, not all of it is useful. To reduce a given data set, the program proceeds in 5 steps. Firstly, an ordered vector of word occurences is constructed. Second one includes leaving out stopwords(modal verbs, conjunctions etc.).

Thirdly, the program takes care of stemming words(training, train, trained, etc.) and uses just the most common format. Right then, a BIF (best individual features) method makes a decision for every single word, if it will be included in a preprocessed data according to a document frequency criterion, term frequency criterion, mutual information, χ^2 statistic or information gain. There will be available one implementation of feature subset selection and that would be forward feature selection with mutual information. This feature vector is now available for next processing. [1][3][7]

Feature selection

This tool for dimensionality reduction is quite popular. It evaluates each vector string according to the criterion, sorts the vector and takes the given number of string values for feature vector. Its popularity comes not only from the fast and easy implementation but merely from the small computational time. It is set to be used individually or by the criterion combination. Here is the implemented criterion record. [3]

1. Term frequency - this criterion evaluates terms in the vector according to their frequency in the training data. [1]
2. Document frequency - terms are evaluated according to the number of documents, they are part of. It is a very simple criterion and mainly easy to implement. [1]
3. Mutual Information - The evaluation proceeds for each term by the formula:

$$MI_{avg}(t) = \sum_{i=1}^{m} P(c_i) I(t, c_i), \tag{1}$$

   where t is a term, m represents a number of categories and $I(t, c_i)$ can be computed as follows:

$$I(t, c) = \log \frac{P(t \wedge c)}{P(t) * P(c)}, \tag{2}$$

   where P(c) is the probability of the category c in the document and P(t) is a term probability. [1][9]
4. Informational gain is considered to be a statistically approved criterion. The calculation for each string goes as the formula shows:

$$IG(t) = -\sum_{i=1}^{m} P(c_i) \log P(c_i) + \sum_{i=1}^{m} P(c_i|t) \log P(c_i|t)$$

$$+\sum_{i=1}^{m} P(c_i|\neg t)\log P(c_i|\neg t), \tag{3}$$

where t represents term t, P are the probabilities. [1][9]

5. χ^2 statistic evaluates vector terms by the measure of the independence between the term t and the category c. The formula right under demonstrates the calculation.

$$\chi^2_{avg}(t) = \sum_{i=1}^{m} P(c_i)\chi^2(t, c_i), \tag{4}$$

where $\chi^2(t, c_i)$ is computed as follows:

$$\chi^2(t, c) = \frac{N * (AD - CB)^2}{(A + C) * (B + D) * (A + B) * (C + D),} \tag{5}$$

where N is the number of documents in the training data. A is the number of documents, where term t and class c cooccur. B represents data, where t occurs and c does not. C is the number, where c is represented but t is not. D is the number of documents without any occurrence. [1][9]

6. Sequential feature selection takes into account the precalculated value of mutual information. Let V be the initial input vector. After MI preprocessing, the method finds the word w' maximazing the MI(w). Let U be $V\backslash\{$w'$\}$ and $S = \{$w'$\}$. The next selection step is a cycle running until it reaches the desired number of features:
   (a) For w $\in U$, compute MI(w,S).
   (b) Find the word w" that maximizes MI(w,S), set $U = V\backslash\{$w"$\}$ and $S \cup \{w''\}$.

   However, the calculation of MI(w,S) is quite difficult. One of the best solution is suggested by maxMI algorithm which substitues the problematic assignment with the formulala:

$$maxMI(w, S) = MI(w) - \max_{s \in S} I(w, S), \tag{6}$$

   which provides almost the same algorithm except the maximizing part. [1][8]

Expected time complexity

Let n be the length of trained vocabulary and l the length of classified vector. The stemming and stopword process takes O(l) time multiplied by the constant of the stopword's vector or the stemming calculation.

Let us have a look at feature selection methods: In every single method the complete system of categories is searched. The system of documents follows with all of the words in its vector, this means $|categories| * |documents| * |vector|$, approximately equals O(n). Classified vector iteration takes O(l) as expected.

1. Document frequency - There is a classified vector iteration over the vocabulary. This takes O(l)O(n). Assuming that the classified vocabulary vector is certainly much shorter than the vocabulary, this takes linear time.
2. Term frequency - The same iteration process like the document frequency, it takes linear time as well.
3. Mutual information - $|categories| * |classified - vector|documents| * |vector| * |categories| * |documents| * |vector|$. Counting all together, one gets $O(l)O(n^2)$. This means quadratic time.
4. Informational gain - One has to iterate over the whole vocabulary two times during a single cycle. This takes quadratic time as well.
5. χ^2 statistic - $|classified-vector|*|categories|*|documents|*|vector|*|categories|*|documents|$. Although the second iteration lacks $|vector|$ the complexity remains quadratic in time.
6. Sequential feature selection using pre-calculated mutual information - Let k be the final number of features in the feature vector. The thing, one knows for sure is that the comlexity of the calculation would be at least quadratic as the mutual information takes the time. As the calculation for each of the k features could be quadratic, this takes $O(k)O(n^2)$ which means quadratic asymptotically.

[9]

Data storage

A newly created vocabulary vector of a certain document could become a training material or testing material for certain class. Suppose, it is a training material. A new instance class of such vectors is built with certain probability parameters for classification purpose and it is serialised into a file. If the vector is a testing material, vocabulary vectors of tested classes are deserialised from the file. The vector(the document) is categorised into a class according to the probability information and the deserialised vectors. Right then, a new class vocabulary(extended by the testing vocabulary vector) is serialised with the other unchanged class vocabulary vectors into the file.

Document classification

The multinomial Nave Bayes model is implemented. It treats a documents as an ordered vector of word occurrences. The program finds a probability that a document d belongs to to a certain class c:

$$P(c|d) = P(c) \prod_{1 \leq k \leq n_d} (P(t_k|c). \tag{7}$$

t_k represents k^{th} term in a document vocabulary. [5][6]

Calculation process

The class Class with the highest probability is chosen. And the document d is categorised into the class:

$$Class = argmax_{c \in C} \bar{P}(c|d) = argmax_{c \in C} \bar{P}(c) \prod_{1 \leq k \leq n_d} (\bar{P}(t_k|c). \tag{8}$$

However, it is used $\bar{P}$ instead of P because the values are not real, they are estimated from the training data. Meanwhile, there is a lot of multiplying in the formula which can cause a problem, especially in the floating point underflow. Therefore, the formula is changed according to:

$$log(xy) = log(x) + log(y), \tag{9}$$

using the logarithmic monotony to the:

$$Class = argmax_{c \in C} [\log \bar{P}(c) + \sum_{1 \leq k \leq n_d} \log \bar{P}(t_k|c)]. \tag{10}$$

The parameters $\bar{P}(c)$ and $\bar{P}(t_k|c)$ are estimated:

$$\bar{P}(c) = \frac{N_c}{N}, \tag{11}$$

where N_c is the number of documents in the class c and N is a number of all documents.

$$\bar{P}(t|c) = \frac{O_c t + 1}{\sum_{\bar{t} \in V} O_{c\bar{t}} + 1}, \tag{12}$$

where O_{ct} is the number of occurrences of term t in the training documents from class c. It is added 1 to the numerator and denominator because of the possibility that the value of the probabilities could be zero, so it could damage all the calculation. [5][6]

Project API

General

A classification library is implemented as a set of java packages working together. The packages are all exported as a jar file. To use the library, one has to set a build path of their project to this external jar file and import the main package *NaiveBayes*. The jar provides public methods to access the functionality of the library, described respectively.

A method *train*, designed to expand classifier's knowledge capabilities by providing sample text data. There are implemented four variants of the method, differing in the input parameters. A general scheme of the methods accepts three parameters: a category name, a document type and an input text. First two parameters apply to each of the method. The document type flags the input format in order to process text further. The third parameter specifies the form of the input. Method *trainFile* accepts a file name of the existing input. Method *trainString* accepts the input as a string. Furthermore, two overloaded variants are available, excepting the third parameter as the vector of file names or the vector of strings respectively.

A method *classify*, constructed to decide on a proper category for the input text. There are implemented two variants of the method: *classifyFile* and *classifyString*. The general scheme of the methods accepts three parameters. The first one specifies if the input text as a string value or a string file name. The second parameter, a document type, meets the same function as in a document training. The third one, a selection type, allows to choose a feature selection method from a given set or their combination.

The trained material is stored by a serialization process into a file. The file is named *catfile*. By default, there is only a single file and the program uses its name. However, the property methods *setcategoriesFileName*, *addcategoriesFileName* and *removecategoriesFileName* provide an interface to control a serialization and enable even more training tasks, where the task could be described as a training system of files related to a certain topic.

The last library function available, is a possibility of changing the default length of a feature vector by *setnumFeat* method accepting an int value parameter.

Main package

This namespace provides a public accessibility to the library using the functionality of all packages. It is designed in a way to be easily extended. Therefore, each method represents a logical part in a document processing. This basically means that each package could be extended by a functional class doing quite a different job but under two condition. The class has to provide a general method to be accessed properly by the namespace methods and must preserve a return type as well. Moreover, if the library functions lack any kind of a logical block, a new package could be written and added.

Data types package

A set of categories is implemented as a category vector, where the category is a new data type. Each category has a name, its length and a vector of documents. Additionally, it consists of a few setters and getters to add, rename or change the category content.
In fact, a document is implemented as a new data type too. It contains a property vector, applicable as a metadata storage, where the piece of metadata information is one property. There also belongs a vocabulary vector, which is internaly represented as a hashmap of string words and the number of their occurence in the document text .

Plain text package

A document conversion is provided by Plain text. Each class represents a different type of a conversion tool, recognizing the document type and changing it to the plain text . The html, email and xml conversion tools are included:

1. Html conversion tool removes all the html tags and its parameters. It parses the plain text included between two of the tags. The unknown tags are not removed. The parser simply ignores the wrong written html code(missing brackets, tags, etc.). This could lead to a more serious problem as the parts of the tags remain untouched and harm the training or classification process. The main reason for not implementing the control over wrong code is the variety of mistakes, it could contain. Trying to handle the unpredictable mistakes could be less efficient than leaving them out.
2. Xml parser just removes the specific symbols of xml tagging(/, <, >, etc.) because it understands all tag names and attributes as the part of the document. Furthermore, the wrong written xml document remains

untouched.
3. Email converter just parses the body section of the mail if not given explicitely and passes it to the next proceedings.

The package is freely extensible by other conversion tools or any more suitable conversion method. Moreover, a metadata parser is implemented as a part of the package, allowing to extract meta information from html or email documents when neccessary.

Vector conversion package

The removal of redundant words and their stemming can be procceded by the package. Two classes are available, stopwords and stemming words.

The input text is parsed according to the default stopword file deleting all its words' occurences. A stopword file can be easily changed or updated by an overloaded method removing stopwords from the package, excepting the new file name.

Stemming a word means reducing a derived word into its stem or base, which can differ a lot in many situation. A freely distributed Porter stemming algorithm, slightly adjusted, deals with the stemming in the package. [10]

Serialize package

By default, the current instance of the category set is serialised into a file specified by the internal variable *catfile*. All of this is obtained by the package using serialising and deserialising method: *serjavaObject* and *deserjavaObject* respectively. The serialisation is built in a way to be used as little as possible because it takes a certain amount of time. Nevertheless, it might be used after each serialisation process if neccessary.

Feature vector package

Anyway, a feature vector construction tends to be the most peculiar part of all. Therefore, it is necessary to give the package a certain amount of abstraction. One expects implementation of 5 BIF(Best Individual Feature) methods: term frequency, document frequency, mutual information, information gain and χ^2 statistics. In addition, a SFS(Subset Feature Selection) method, using precalculated mutual information, will be implemented as well. In order to ensure portability and an easy extension, the best solution would

be to write each method as a part of a single class in the package. However, it seems that each class should implement some kind of a general scheme, dictating the exact method form, supporting any public variables, provided outside as a result of the method's calculation process.

Therefore, a class *selector* is meant to meet the function. It contains the field of the final feature vector, the virtual function(selecticting top features), the hashmap(having one feature as the key and its actuall count in the input as the value). It also provides the class, for comparing two hashmap key features by their value. The class implements the Comparator interface.

Nevertheless, the entrance point of the package is another single class, specifying the type of the feature method selection. The selection type is a string variable. Every char position in the string is a integer number defining the method selection. The individual instances of selection classes are created according to the parameter. If only one method is desired, the procedure calls the instance function to calculate top vector features and these are returned as the result.

Otherwise, the number of other methods could be accessed. First of all, the instance methods calculate its top feature vectors which are passed further as the paramenter. The vectors' length is defined by the user. Secondly, mentioned methods merge these features into a single feature set having the same length like before. The difference is in the way, the methods merge these features.

The features are notably sorted by their counts in the vector. Methods iteratete over the 1^{st} vectors' position, then 2^{nd} and so on, until the final position in the vector is reached. However, two general approaches are taken into an account. If a string feature, which does not belong to the final feature vector yet, is iterated, it is immediately added to the vector either by one occurence or by the actuall number of occurences in the input. The first approach takes the iterated item as the single feature while the second approach tries to strengthen the position of the topmost features. The approach then supports two variants of unsuccessful iteration(the feature already belongs to the final vector). Either by moving the iterator to the next vector in the same position or by moving the iterator to the next feature of the vector.

The next topic deals with the selection method classes. Their names correspond to the appropriate selection method. They contain derived variables and and methods from the *selector* class. Furtermore, they generally

implemet various string double hasmaps, taking certain information about input vocabulary, supporting the final calculation of the top feature vector. The hasmap key is a concatenated string value of the probable category, the $ mark(separator) and the appropriate vocabulary term. The current probabilities or any other values for further calculation are stored in them as doubles. The features are then put into the map derived from the *selector* with their real probabilistic values. The calculation is based on the details mentioned in the section *Feature Selection*. Subsequently, the map is ordered by the implemented comparer and the topmost features are considered as the return vector. The number of the topmost feature is provided by the user.

Vector classifier

The namespace provides two methods, using the current categories instance and the vocabulary feature vector as their parameters firstly to calculate conditional probabilities and secondly to find the most probable class, the right category. The *Calculation process* section describes the algorithm implemented by the methods. The final return type is a string value which corresponds to the existing category name.

Extension package

It provides an independent graphical interface structure to build a plugin application with the library project functionality.
!!!Still working here.!!!

Last package builds a graphical interface for the project application to access library in a convenient way. It defines the frames and the visual style of the library's interface in the most general way with an additional functionality for the email client which could be modified or extended by requirements of other applications instead.

Exceptions

Generally, there are two exception types possible. Internal exceptions, occuring right in the packages, do not affect the program a lot. They mean that the calculation cannot be procceded exactly in the way the user expects it to. If the new stopword file cannot be found, the program uses the default one there is no need in propagating the exception up the methods' hierarchy.

On the other hand, if the exception handles the serialization process, it could seriously affect the calculation. The approach propagates the exception with the well defined description and quickly terminates the program.

Project application

By the open source project, there exists an additional classification API for document categorisation processing. The API defines its own shape independently of the project it extends.
There are two main areas to deal with. A classification menu defines current categories. It enables a user to see the content of each category in a separate frame. A new category can be created here in the menu. A dialog for a document training is also available.
On the other hand, incoming text is automatically sorted in a category as soon as the category is trained and it is marked by a class label right in the inbox folder. !!!still working here!!!

Libraries

The program will use mainly some of the dynamically loadable libraries, that can be called in runtime, from the Java Class library. These are java.io, java.utils, java.lang and java.math, java.awt.

Programming language and platform

The code will be written in Java due to its functionality on several platforms including (Unix, Windows etc). As a developer tool, the Eclipse IDE will be used. Implementation of the algorithm will be written under the Linux Platform as a java project.

References

[1] S. Kotsiantis, E. Athanasopoulou, P. Pintelas, Logitboost of Multinomial Bayesian Classifier for Text Classification, International Review on Computers and Software (IRECOS), Volume: 1(3), Pages = 243 – 250, November 2006.

[2] Dmitry Pavlov, Ramnath Balasubramanyan, Byron Dom, Shyam Kapur, Jignashu Parikh, KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data miningsification and Clustering with Mixture of Multinomials, Pages = 829 – 834, ACM, ISBN:1-58113-888-1, 2004.

[3] Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng, Some Effective Techniques for Naive Bayes Text Classification, Ieee Tranasactions on knowledge and data engineering, Volume: 18 Issue: 11, pp. 9185444, ISSN: 1041-4347, November 2006.

[4] Michael W. Berry and Malu Castellanos, Survey of Text Mining: Clustering, Classification, and Retrieval, Second Edition, Editors. Springer, January 2008.

[5] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schtze, Introduction to Information Retrieval, Cambridge University Press, ISBN: 0521865719, 2008.

[6] Ethem Alpaydn, Introduction to Machine Learning Second Edition, Massachusetts Institute of Technology, Cambridge, Massachusetts, London, England, The MIT Press, ISBN-10: 0-262-01243-X, 2010.

[7] Peter Jackson and Isabelle Moulinier, Natural language processing for online applications Text retrieval, Extracton and Categorization, John Benjamins Publishing Company, ISBN-10: 1588112500, July 2002.

[8] Jana Novovicov, Anton Maly, and Pavel Pudil, Feature Selection using Improved Mutual Information for Text Classification, Institute of Information Theory and Automation, Department of Pattern Recognition, Academy of Sciences of the Czech Republic; The University of Economics, Faculty of Management; Czech Technical University, Faculty of Electrical Engineering; Prague, Czech Republic; Springer, 2004.

[9] Yiming Yang and Jan O. Pedersen, A Comparative Study on Feature Selection in Text Categorization, School of Computer Science, Volume: 20, Issue: 15, Publisher: Citeseer, Pages: 412 – 420, ISBN:1-55860-486-3, 1997.

[10] Willett, P., The Porter stemming algorithm: then and now. Program: electronic library and information systems, Volume: 40 (3), Pages: 219 – 223, 2006.