Data classifier CX8
(Project specification)

Erik Lux

23^{th} November 2011

Basic information

Data classifier is a java toolkit for training, classifying and testing file-format independent documents. It applies only one classification algorithm but supports its performance with a variety of feature selection methods.
CX8 classifies documents into user-defined categories. The category is a document set also containing some metadata information. Document examples falling within the same category share a common theme. It is consequently in a user competence to provide such examples.

General description

The main project component is 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 process includes the way of information retrieval, learning from examples and testing its capability. The component is built upon the Nave Bayes classifier, a simple and quite accurate probabilistic classifier applying Bayes' theorem [5].

Project application

The other component is a library application. This application adds extra graphical interface. This interface provides user comfort to handle the library. The component could be used by any kind of existing software as its extension. In this case, however, an open source email client is considered to be the one. The client is extended in a way to categorise incoming mail.

Technical documentation

Project design

As mentioned earlier, the CX8 is based on two component model. The application component needs to call the library component to perform classification tasks. These two have to work together. All of this is ensured by the public library interface. It provides methods, to create a new category, to train the category, to test the category and to classify a document into the category.

Text conversion

Documents could come in a lot of different forms such as plain text, html, xml, email-form,... Classification methods work only with plain text documents. Therefore, the library has its own package for a text conversion. The conversion tool recognises the document type and converts its content into the plain text. There are three converters available yet: the html, the xml and the email converter. However, the package could be easily extended by adding another converter.

Metadata

Conversion tools not only return a plain text but parse metadata information about the document as well. Be aware that such operation is far from redundant as it stores document information for technical and statistical purposes. The data are represented as the dictionary or the hashmap of the key-value pair. The main key *Type* represents the document type. Other keys are actually derived from the document type. For example, an html document contains the key *Title*. An email document contains the key *From* or the key *To* for instance.

Data preprocessing

A string vector is expected as the classifier input parameter. The length of the vector is not known in advance. Therefore, an extremely high dimensionality space of text data could flow through the program. However, not all of it is useful. To reduce the given data set, the program proceeds in 4 steps [1][3][7].

1. create an ordered vector of word occurences.
2. leave out stopwords(modal words, conjunctions, ets.)

3. stemm words in the vector(training -> train, train -> train, trained -> train)
4. find top features.

Feature selection

The tool for a 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 to construct the feature vector. Its popularity lies not only in the fast and easy implementation but merely in small computational time. The criteria could be applied individually or together. The implemented criteria description is presented over here [3].

1. Term frequency - the criterion evaluates terms in the vector according to their frequency in training data [1].

2. Document frequency - terms are evaluated according to the number of documents, they are part of [1].

3. Mutual Information - The criterion is used in the statistical modelling, describing word associations and related applications [9]. 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 the number of categories and $I(t, c_i)$ displays the term-category relation:

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

where P(c) is the probability of a category c in a document d, P(t) is the term probability and P(t∧c) is a coocurence probability of t and c in d [1][9].

4. Informational gain is frequently employed as a term goodness criterion. It measures the number of bitsof information obtained for category prediction by knowing the presence or absence of a term in a document [9].

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

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

where t represents a term, P(c) is a category probability and P(c—t) is a conditional probability of a category c in a document provided by a probability of a term t being the part of the document as well. [1].

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 training data. A is the number of documents, where a term t and a 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 $V\backslash\{\text{w'}\}$ and $S = \{\text{w'}\}$. 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 $U = V\backslash\{\text{w"}\}$ and $S \cup \{w''\}$.

   The calculation of MI(w,S) or finding a direct relation between w and all the set of words is quite difficult. Therefore, several ways are presented nowdays to approximate the onerous MI. One of the best solution is suggested by maxMI algorithm. It substitues the problematic assignment with the formulala:

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

which provides a handy way to deal with calculation of MI for a term and a term set [1][8].

Expected time complexity

Let n be the length of trained vocabulary and l the length of the 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 closer look at feature selection methods. Each method searches all the system of categories. 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 the 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 the 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. The calculation for each of k features could be quadratic at worst. This takes $O(k)O(n^2)$ which means quadratic asymptotically.

[9]

Data storage

A newly created vocabulary vector could become a training material or a testing material. Suppose, it is the training material. The feature vector is either added to the existing category or a new category instance is initialized. The vector then becomes the part of the instance. In the end, the category is serialised into the file. If the vector is the testing material, vocabulary feature vectors are deserialised from the file. The vector is classified into the category according to the probability, specified by deserialised items. It extends one of the categories, which could be sebsequently serialised back into the file, if expected.

Document classification

CX8 implements the multinomial Nave Bayes model which treats a document as an ordered vector of word occurrences. It is based on two step algorithm:

1. find a conditional probability that a category c is a searched one provided by the probability that a document d belongs to c.
2. determine the most probable category applying Naive Bayes formula.

Conditional probabilities are estimated:

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

P(c) is a category probability. t_k represents the k^{th} term in a document. P(t_k|c) is a conditional probability describing the t_k as the part of a document d provided by a category c being the part of d as well [5][6].

Calculation process

Naive Bayes formula is applied to the newly calculated conditional probabilities.

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

The formula contains the probability of $\bar{P}$(c|d) instead of P(c|d) and $\bar{P}$(c) instead of P(c). That is just emphasis on the fact the probabilities are not

real but estimated from trained data.
Meanwhile, there is a lot of multiplying in the formula which can cause a problem, especially in the floating point underflow. Therefore, the logarithm function is applied to the formula using a rule:

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

Thanks to the logarithmic monotony, one can rewrite the formula as follows:

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

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

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

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

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

O_{ct} is a number of a term t occurrences in training documents belonging a class c. However, the probabilities could equal zero in the denominator, leading to an undefined expressions. Therefore, 1 is added to both the numerator and the denominator [5][6].

Project API

General

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

The 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: the category name, the document type and

the 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 the file name of the existing input. Method *trainString* accepts the input as the string. To provide multiple training just by one function, two overloaded variants of previous methods are available, excepting the third parameter as the vector of file names or the vector of strings respectively.

The method *classify*, constructed to decide on the 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 the input text is the string value or the string file name. The second parameter, the document type, meets the same function as in the document training. The third one, the selection type, allows to choose a feature selection method from the given set or their combination.

The trained material is stored into a file. The file's name is *catfile*. By default, there is only one file. The property methods *setcategoriesFileName*, *addcategoriesFileName* and *removecategoriesFileName* provide an interface to control a serialization, to store trained data in more than one file and to enable even more training tasks, where the task could be described as a training system of files related to the certain topic.

The last library function available is *setnumFeat*. It changes the default length of a feature vector accepting a new value as the int parameter.

Main package

This package provides a public interface using the functionality of all packages. It is designed in a way to be easily extended. Therefore, each method represents a logical part in the document processing. This basically means that each package could be extended or changed independently of others but under two conditions. The package has to provide a function visible enough for the main package and this function has to perserve the required return type. Moreover, if suitable, adding a new logical block, is also an option.

Data types package

A set of categories is internaly represented as a vector of category instances. The category is a new data type. Each category has items such as a name, a length and a vector of documents. Additionally, a few setters and getters to add, rename or change the category content, are the part of the category to set these items.
In fact, a document is implemented as another data type. It contains a property vector, applicable as a metadata storage. A piece of metadata information is a property. There also belongs a vocabulary vector, which is internaly represented as a hashmap of string words with the number of their occurence in a document.

Plain text package

A document conversion is provided by Plain text. Each class represents a specific type of a conversion, recognizing the document type and changing it to a 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 the 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. However, 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 level.

The package is freely extensible by other conversion tools or any conversion methods. Moreover, a metadata parser is implemented as a part of the package, allowing to extract meta information from html or email documents when neccessary. It searches for appropriate meta tags in documents. If they are found, the parser takes information inside the tags to create a new metadata property.

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 the stopwords. The stopword file can be easily changed or updated by an overloaded variant of the method, removing stopwords from the package, excepting a new file name as the input parameter.

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

The current instance of the category set is serialised into the file specified by the private variable *catfile*. All of this is obtained by the package using serialising and deserialising methods: *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. By default, any changes are not serialised implicitly. Nevertheless, they might be if neccessary.

Feature vector package

Anyway, a feature vector construction tends to be the most peculiar part of all. Therefore, it is necessary to design the package by a certain amount of abstraction.

One expects implementation of 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, is implemented as well. In order to ensure portability and an easy extension, the best solution would be to write each method into a different class in the package. In any case, one class should implement some kind of a general scheme, dictating the exact method form, supporting any public variables, provided outside as the result of the method's calculation.

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 public package interface is provided another class, specifying the type of the feature method selection. The selection type is a string variable. Every position in the string contains an integer number defining the method selection. The 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 a 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 use-defined. Secondly, mentioned methods merge these features into a single feature set of the same length . 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. 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 methods from the *selector* class. Furtermore, they generally implemet various string double hasmaps, taking certain information about input vocabulary, supporting the final selection of the top features. The hasmap key is a concatenated string value of the category name, the $ mark(separator)

and the appropriate vocabulary term. The current probabilities or any other values for next 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 user-defined.

Vector classifier

The namespace provides two methods, using the current categories instance and the vocabulary feature vector as their parameters to calculate conditional probabilities and to find the right category. The *Calculation process* section describes the algorithm implementation. The final return type is a string value corresponding 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 speaking, 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 which quickly terminates the program right after.

Project application

By the classification library, there is 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 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

CX8 uses mainly some of the dynamically loadable libraries 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 is written in Java due to its functionality on several platforms including Unix, Windows, etc. As a developer tool, the Eclipse IDE is used. Implementation of the algorithm will be written under the Linux Platform as a java project.

Classification techniques

There is a wide variety of document classification mechanisms using quite different approaches. These approaches handle an input data in their own specific way which means choosing the most appropriate method for each of the tasks. One can see preprocessing input data as the first task of the categorisation algorithms. We will discuss this topic in the next paragraph. Right then, certain techniques except Naive Bayes classifier will be issued.

Moreover, their advantages and disadvantages as well as their specifics will be argued.

Data pre-processing

The process of text retrieval usually begins with indexing an input document. Every word in the document takes an index or the position to simplify its handling. This attempt for document well handling is called Vector space document representation. However, one of the greatest problems that this representation does not solve by itself, is a high dimensionality of vectors.

Before the problem could be solved, the document can be pre-processed to eliminate some of the vocabulary parts that are not necessarily essential in the text. These include stop words(conjunctions, prepositions ...) and stemming words(words like train, trained, training ...). To work the high dimensionality problem out, a feature text representation is used instead. However, it is not such a different approach as the transformation of the text into the vector makes the basis here as well. So, this technique adds some functions to reduce the dimensionality of vectors. First one, best individual features uses evaluation function that is applied to a single word. Scoring individual words can be performed by some measures like term frequency, document frequency, mutual information ... The conclusion of these individual features is processed and the top features are selected afterwards. On the other hand, the subset feature selection functions firstly select the top scoring word and then add one word at a time to fullfil the number of words. Another, quite a polular approach is a feature transformation which does not measure words' weights and processes the top features but compacts the vocabulary based on feature occurrencies. Its aim is to learn discriminative transformation matrix in order to reduce initial vector space.

The big advantage of feature vector representation is that it could be used by both, instance-based and model-based classifiers. However, this method does not capture all important structural information. Therefore, it is not convenient enough for web documents.

There exists one other way to represent a document that statistically outperforms the others and also satisfies the categorisation of web pages. It is a recently developed graph based document representation, using the k-

nearest neighbour classification algorithm. Although it presented much better performance on web documents, the problem is, that the eager, model based classifiers, cannot use this method directly. So, there seems to appear new hybrid text classification methods combining both the vector and the k-nearest neighbour approach.

Classification vector models

Even though my decision for classifier is Naive Bayes, there are still ways how to approach the classifier. The ways differs mainly in vocabulary vector and the information they consist of. Older way considers input as a binary feature vector representing whether a current word is present or absent in the vocabulary. This is called multivariate Naive Bayes. Although this way is the closest one to the native Naive Bayes classifier, it still lacks the ability to utilize term frequencies in the document.

Thus a multinomial model was introduces as an alternative, representing the number frequencies of each term in the document. However, as time passes two serious problems are encounter with a multinomial Naive Bayes classier. First is a rough parameter estimation. In general. the testing data are merged into one big document and the probabilities are calculated from this big document. Well, there comes a consequence that bigger document influences the probabilities more than the smaller one, although that it does not have anything to do with the document importance. Second is that with the insufficient amount of training data the classifier cannot perform well enough as some of the categories are pretty rare to be equipped with the data of certain amount. Here come some nixed or improved techniques to improve the performance of Naive Bayes.

Classification

The most suitable approaches for document classification can be divided into two groups. One could be named model-based classifiers and the other instance-based classifiers.

Model-based classifiers are built upon a mathematical model. Therefore, their calculation steps use the basic principle of the current model. They generalize the problem and apply the principles.These include classifiers such as Naive Bayes, expectation maximisation, latent semantic indexing, artificial neural network, decision trees ...

On the other side, instance-based learning or memory-based learning is a family of learning algorithms that, instead of performing explicit generalisation, compare new problem instances with instances seen in training data, which have been saved directly into the memory. Instance-based learning is a kind of lazy learning. It is known under the name of instance-based because it makes hypotheses directly from the training set instances. As the main examples for this type of classifiers could be considered k-Nearest neighbour algorithms.

Similar projects

Here is a list of projects which have been made, using the same technique I am implementing. Surely, with all their pros and cons, I will try to fully discuss their permanence as well as the preprocessing techniques they use. Furthermore, I will have a look at the classification techniques they adopt.

1. **Data mining software in Java named Weka, using Naive Bayes Classifier**

   Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data preprocessing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Moreover, Weka provides access to SQL databases with java connectivity. It is not directly matched with the input tables for categorization process but can be simply transformed into these tables. The area which is still not included in weka algorithms is sequence modeling.
   This cross-platform software provides user except Naive Bayes and other basic algorithms even the implementation of Expectation Maximalisation algorithm or K-means algorithm. It could be easily manipalated by a graphical interface. Simply, this software is a present of text categorization.

   Weka homepage

2. **Classifier4J**

   Classifier4J is a Java library designed to do text classification. It comes with an implementation of a Bayesian classifier, and now has some other features, including a text summary facility. It makes the use of vector data reprezentation. Its API could work as a spam filter or blog cl.

   Classifier4J homepage

3. **A Naive Bayesian Classifier in C# - NClassifier**

   NClassifier is an open source product. It is a .NET library that supports text classification and text summarization. It is a very extensible library consisting largely of interfaces. It includes, out of the box, an implementation of the Bayesian classification algorithm. The classifier is synched closely with Classifier4J project. For example, the database handling in java is replaced with ADO.Net solution. The future perspective of the project is undetermined. It could at least stay with Classifier4J or, at the most, cut into its own direction.

   NClassifier homepage

4. **The RDP Classifier - a nave Bayesian classifier**

   The Ribosomal Database Project (RDP) Classifier, a nave Bayesian classifier, can rapidly and accurately classify bacterial 16S rRNA sequences into the new higher-order taxonomy proposed in Bergey's Taxonomic Outline of the Prokaryotes (2nd ed., release 5.0, Springer-Verlag, New York, NY, 2004). It provides taxonomic assignments from domain to genus, with confidence estimates for each assignment.

   The RDP classifier homepage

5. **Mallet**

   Mallet is a java-based software for natural language text processing. It provides efficient routines for feature vector conversion. Mallet includes a wide variety of classification tools(Naive Bayes and decision trees techniques for instance). Furthermore, Mallet has got code for evaluating its classification algorithms and its efficiency.

Mallet homepage

6. **jBNC**

   It is a java toolkit, providing methods for training, testing and applying Bayesian Network Classifiers. This software was manly tested in artifiacial inteligence and machine learning tasks. It performed well.

   jBNC homepage

7. **Orange**

   Orange is a component-based data mining and machine learning software suite, featuring friendly yet powerful and flexible visual programming front-end for explorative data analysis and visualization, and Python bindings and libraries for scripting. It includes comprehensive set of components for data preprocessing, feature scoring and filtering, modeling, model evaluation, and exploration techniques. It is implemented in C++ (speed) and Python (flexibility). Its graphical user interface builds upon cross-platform Qt framework. Orange is distributed free under the GPL. Orange includes a component based Naive Bayes Classifier.

   Orange Component Naive Bayes Classifier

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.