

Learning from the Classics

Machine Learning Laboratory Assignment

2011 FALL TERM



Enginyeria en Informàtica

Departament de Llenguatges i Sistemes Informàtics



FIB

Facultat d'Informàtica
de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Inductive Learning Laboratory Assignment

This is the laboratory assignment for the second half of the course *Machine Learning (Aprenentatge)*.

This assignment will involve the use of supervised inductive algorithms in a specific dataset. The dataset for this assignment can be downloaded from the webpage of the course using the link that you will find in the laboratory page.

This assignment has to be solved in groups of **two** students. You can also solve the assignment individually, but it will be more work.

The work of this assignment includes performing different experiments using the algorithm explained in theory class and writing a report explaining the work done, the results obtained and answering the set of questions that appear in this assignment.

The report has to be delivered electronically as a PDF file, using the assignment delivery application from the racó as will be explained in a notice that will be posted in the racó before Christmas break.

The deadline for delivering the report will be January 9th 2012.

Text processing

2.1 Introduction

The areas of text mining and information retrieval are gaining relevance since the internet is providing a huge source of documents of any kind. There are a lot of tasks and problems that can be solved using documents and machine learning algorithms are frequently used to solve them.

The tasks in the area of text mining are very diverse and some times present unique opportunities for building applications for tasks that can not be solved in other ways. Think for example of these different scenarios:

1. You are a book seller that want to recommend books not by their popularity (the more buyers a book has the better the book is, yeah right!) but by the similarity in style or content. You can use information from the text in the books to classify other books and discover authors for example with similar style. With this you can answer questions like for example, is the author X as easy to read as author Y?
2. Imagine that you are subscribed to different feeds in your RSS news reader and you do not have time to read all the messages that you daily receive. Could it be built an application that reads the messages for you and able to tell you whether it will be of your interest or not? Or that automatically classifies and regroups the news using keywords obtained from the messages?
3. Imagine that you have a document and you do not know who is the author, but you have documents from a set of possible authors. It could be possible to learn from the documents the difference in use of words and styles of the authors to find out who is the real author. This task is usually done by experts in linguistics and obviously there are not many of them, can this task be automated?

2.2 Representing documents as examples

In order to use documents and text in machine learning usually a transformation process has to be used to have a representation suitable for the learning algorithms.

Most of the learning algorithms need an attribute-value representation, something that is very different from the information that appears in documents. The usual representation for text is the representation called **bag of words**.

To obtain the *bag of words* of a set of documents, all different words that appear in the text are treated as attributes and some information is computed for each word from the documents to be used as the value of this attribute.

This means that each document will be represented by all the words that appear in the set of documents and for each attribute some information representing the significance of this word for the

document will be its value. This transformation adapts any set of documents to the attribute-value representation.

2.3 Significance of a word

The value of an attribute (in this case a word) can be computed in different ways. The usual ways are:

TF-IDF: This value combines two different measures, the *term frequency* (TF) and the *inverse document frequency* (IDF).

The term frequency is the proportion of appearance of a word in a document and is computed as:

$$tf_{ij} = \frac{w_{ij}}{\sum_k w_{kj}}$$

The inverse document frequency is the importance of a word in the set of documents and is computed as:

$$idf_i = \log \frac{|D|}{\{d : w_i \in D\}}$$

being $|D|$ the number of documents and $\{d : w_i \in D\}$ the number of documents that have the word i .

The TF-IDF measure is computed as:

$$TFIDF_{ij} = tf_{ij} \times idf_i$$

Term frequency: The term frequency can be also a measure of significance, the more times a word appears in a document the more important is for the document.

Term occurrences: The term frequency is relative to the number of words in a document and can be misleading depending on the length of the document. Sometimes is better con use the number of occurrences of words in the document.

Binary term occurrence: It can be also informative wether if a word occurs or not occurs in a document independently of the number of occurrences.

None of these measures is perfect and usually the patterns that can be obtained from the documents change with the measure used.

2.4 Filtering the words from a document

The first process for obtaining the words from a document is to *tokenize* the text. The usual way is to create a token for each string that is separated by a blank or any punctuation sign, but there are other possibilities depending on the nature of the documents.

The usual result of the tokenizing process is several thousand of tokens (each token is an attribute). This is one of the characteristics of learning from documents, the number of attributes is usually larger than the number of examples. This is also a problem for the learning process because some algorithm do not scale well with the increase of attributes and the quality of the learning process is also degraded by large numbers of irrelevant attributes.

Before applying techniques for attribute selection there are some preprocesses that can be applied to reduce the number of attributes.

The first one is eliminating all the words that are *stop words*. This task is language dependent, so a list of stop words is needed for the specific language of the documents. This process will eliminate some words that are very frequent in the documents but that do not have any significance for the learning task (pronouns, prepositions, adverbs, articles, ...).

Another process related to the specific language of the documents is *stemming*. The problem with words is that you can have different attributes for the same word because it can be written in different ways, for example, singular or plural nouns, verbs in different tenses, masculine or feminine words, prefixes, suffixes, ... Each word is counted as a different attribute, stemming processes the words and substitutes all occurrences for the same token. This means that the occurrence count of each word will be more accurate.

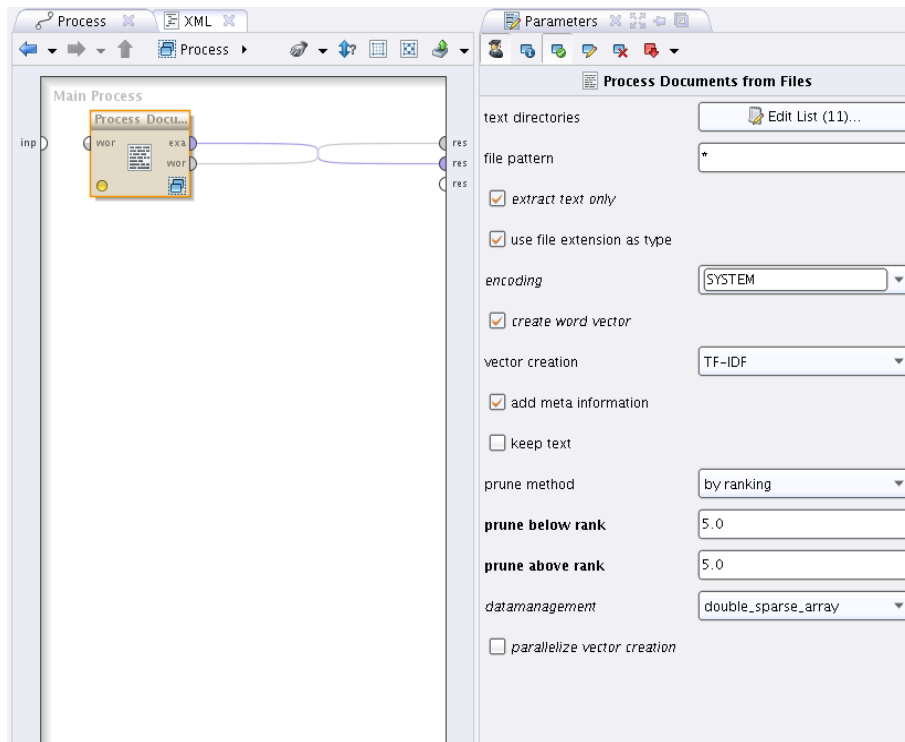
There are other processes that need more specialized knowledge, for example, for eliminating names of entities from the text. This can be done by having a dictionary of names or by eliminating from the text all words that do not appear in a dictionary of the language used in the documents.

Other simple preprocess is to eliminate all the words that appear a lot of times or only a few times, this can be done by frequency, eliminating words with absolute frequency (for all documents) more or less than a certain value, by absolute appearance, eliminating words that appear in less or more documents than a specific value, or by rank, eliminating a percentage of the most and less frequent words.

2.5 Rapidminer text processing plug in

A plug in for text processing can be installed in rapidminer using the update server. After reinitializing there will be a new folder named *Text Processing* in the operators tab.

The simplest way to process documents with this plug in, is to put all documents in a set of directories (one for each class) and to use the operator *Process documents from files*. These are the parameters of this operator:



For the *text directories* parameter, the edit list button allows you to introduce the labels to use for classifying the documents and the directories where the documents are. The plug in assumes that each directory has a set of text documents to be transformed.

The plug in can read all the files or only the ones that follow an specified pattern (*file pattern*). It can extract only text or also include xml or html tags (*extract text only*). It can also recognize the type of the file by its extension, assuming to be plain text if it does not know the type (*use file extension as type*).

The files can be transformed to the bag of words representation (*create word vector*). The values to use as attribute values for the words can be selected (*vector creation*) from the measures explained in section 2.3 (TF-IDF, Term frequency, Term occurrence, Binary text occurrence).

Also the set of words used to represent the text files can be pruned (*prune method*) by percentual, absolute value or ranking.

- Percentual pruning will eliminate words below or above a frequency. Notice that words with a very low frequency sometimes are just noise, but also include specific words that can be important for classification. Also words with large frequency can be irrelevant because can nor separate among classes, but are also general attributes that can help to predict unseen examples.
- Absolute pruning will eliminate words that appear in more or less than a number of documents
- Ranking pruning will eliminate a percentage of the more and less frequent words.

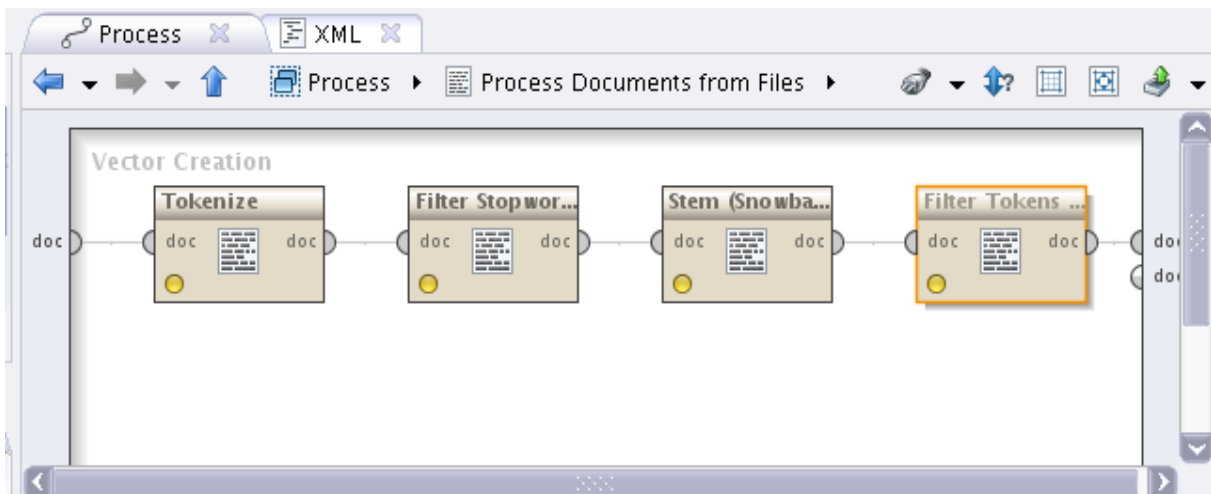
This operator is a subprocess, this means that internally the text has to be tokenized and some processes can also be applied to the tokens.

2.6 Obtaining and processing the tokens

In order to obtain the bag of words from the text files the fist task to perform is tokenization. There is an operator for tokenization that extracts the words from the text. It has one parameter that specifies how to identify words in the text. For plain text the usual way is to use non letters as word separators, but special characters and regular expressions can also be used.

Tokenization is the only task necessary inside this subprocess, but other operators can be applied to reduce the number of words and to filter words that do not add useful information.

The following figure shows a more complete preprocess.



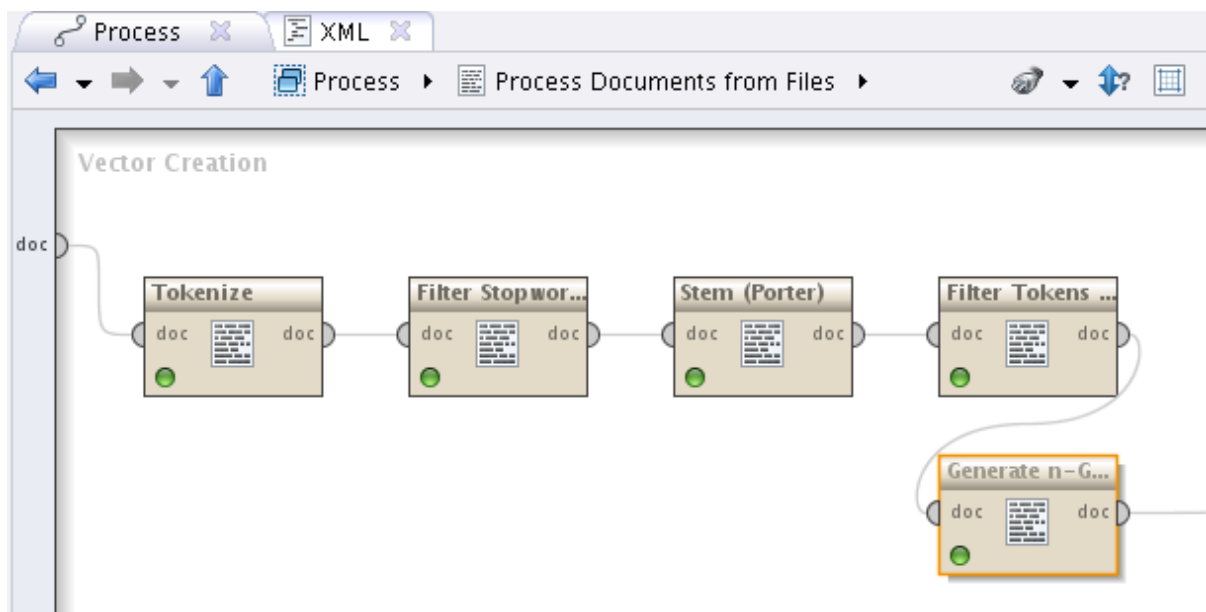
After tokenization appears the operator to filter stop words (inside the filter folder). There are different operators for different languages, and also a dictionary can be provided to include other words (like for example names).

The next operator in the process is stemming (inside the folder stemming). Stemming transforms different occurrences of the same word to the same string. There are different stemming algorithms (snowball, porter, lovins), some work for different languages, some work only for the english language.

This process will reduce the number of tokens and also will identify all the occurrences of the same word into the same token.

The last process filters the tokens shorter or longer than a specific length. Sometimes words too short or words too long are unusual words that are not interesting for characterizing the text (think for example of variables of functions in mathematical formulas, or names of chemical compounds, or random characters due to errors in the text acquisition).

An interesting operator is the one that is able to generate *n-grams*. An *n-gram* is a groups of *n* words that appear together. Usually individual words are not enough for the learning process and *n-grams* can capture interesting relations. Commonly the value used for *n* is 2. The next figure is a process that includes the generation of *n-grams* using the operator **Generate N-Grams (Terms)** (inside the folder Transformation), the attributes obtained for the dataset will include individual words and attributes representing groups of words found together in the text:



Additionally to filtering words, new attributes can also be obtained from the text (extraction folder). For example, the length of the tokens from the document, the number of tokens in the document, or any information about the structure of the document that can be obtained by string matching or regular patterns (specially in XML or HTML documents).

Learning from the classics

3.1 The domain

The purpose of this assignment is to learn from fragments of texts extracted randomly from books of classical authors. The authors are from different ages and the genres of the works are also diverse.

For almost all authors the fragments are from different books, usually three or more. The idea is to have different documents that represent the literary style of the author. But the assumption is not only that the personal style of the authors will show in the documents, but also that the different words that are used in the documents will represent collectively the different genres and other interesting characteristics.

3.2 The goals

We can classify the authors from different perspectives and we are going to experiment if these characteristics can be learned from the documents. The learning goals will be the following:

1. The obvious one is to experiment if it is possible to differentiate one author from another. Each author has a different style, even if they write about the same subject or belong to the same genre.
2. Another obvious classification is the literary genre. To make things easy authors will be classified under the following genres: Philosophy, Politics, Children, Drama, SFTerrorAdventure. A good assumption is that in different genres are used different kinds of words, for example in adventure books probably will appear more verbs of action than in philosophy.
3. Also authors from different ages write differently, we will test the classification: pre16th century, 16th century, 17th century, 18th century, 19th century, 20th century
4. A less obvious classification is if the documents are in the original language of the author or are translations. All documents are in english, so in this case the original language is english. This classification may seem difficult at first, because usually the translations of classical authors are fairly good, but it can also be argued that the vocabulary used in the translations may be more limited and perhaps this shows in the documents.

The task is to use the different algorithms and preprocesses explained in class to see if these goals can be learned and with what accuracy.

3.3 The dataset

The zip file that you have available has a set of directories with text documents. There is a directory **Train** with a subdirectory for each author plus two directories with test examples, there are also two additional files that contain the classification labels corresponding to the authors.

For each author the **Train** directory has 25 text files, each one is a fragment from a book from the author. The files have different sizes and the size of the text does not follow any pattern.

The directory **Test** contains also a subdirectory for each author with 5 document for each one of them, usually from a different book than the training set documents. The directory **New** contains documents from authors that are not in the dataset.

The goal is to learn a model from the train documents and test the accuracy and generality of the classifiers in different scenarios.

The files from the **Test** directory will test the accuracy of the classifiers with data similar to the training data, we can expect similar accuracy to the crossvalidation results, but we are not sure if the classifiers can be used for more than classifying these specific authors.

The files from the **New** directory will be more challenging. We are testing if we have learned classifiers general enough to classify any work from any author in the domain.

For this last set of examples we can only test accuracy for the goals 2, 3 and 4, but we can also informally check if the authors that are given as labels for the first goal are similar somehow to the real authors.

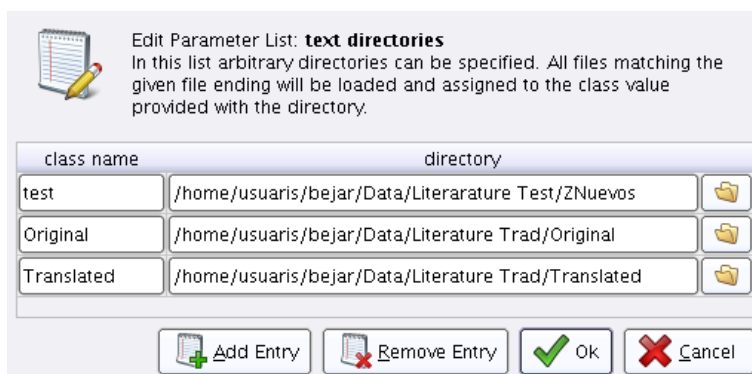
3.4 Experimenting and learning from the dataset

The tasks of the assignment involve preprocessing the dataset to obtain a set of examples, experiment with the learning algorithms that have been explained in theory class and measure the accuracy obtained using crossvalidation with the training data and applying the models learned to the two test datasets.

3.4.1 Reading the data

The directories in the zip file are arranged to learn the authors task. In order to obtain the datasets to learn the rest of the goals you will need to rearrange the files in new sets of directories. For example, if you want to learn the genre you will need to create a directory for each genre and put the corresponding files inside each directory¹.

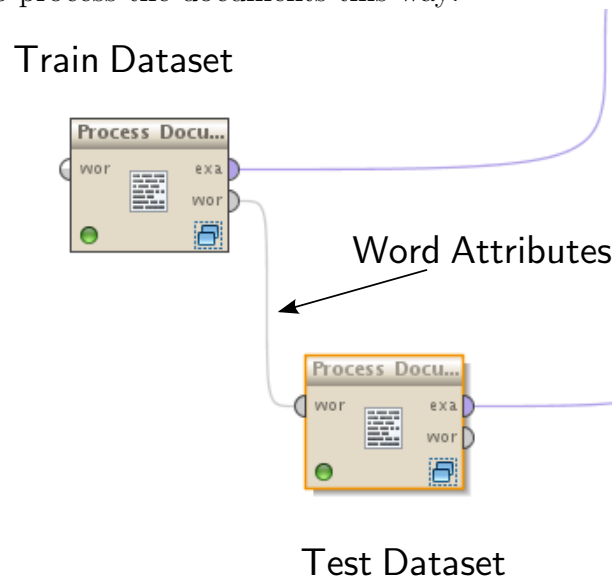
For example, this is the selection of classes and the files for the *process documents from files* operator for learning if the documents are in the original language or are translated:



Notice that the files from the **Test** and **New** directories also have to be rearranged to perform the tests. Notice also that in order to test the accuracy of the classifiers with the test data you have to use the output set of words obtained from the training set so they have the same set of attributes.

¹To know what file goes in each directory you only need to look in the files that have the labels for each author.

In order to obtain a dataset with the same attributes for the training and test set you will have to connect the operators to process the documents this way:



Inside the operator that processes the test set you only need to include the tokenizer operator and the n-grams operator if you are generating n-grams for the dataset.

3.4.2 Preprocessing

There are different preprocess that can be done to the dataset, but the main goal of the preprocess will be to reduce the number of attributes. In learning from text, one of the main problems is that the number of attributes is larger than the number of examples. This can be problematic for some learning algorithms.

In this case the dataset has over one thousand examples, the number of attributes depending on the preprocess can range from several tens of thousands of attributes to some hundreds of attributes or even only tens of attributes.

The first set of preprocessing steps can be performed using the *Process documents from files* operator. This operator has to tokenize the data, and this is the minimum thing that has to be done. Only doing this, the number of attributes is near forty thousand. If you include 2-grams the number of attributes increases to hundreds of thousands and is obviously unfeasible to process the dataset in reasonable time.

As was explained in the previous chapter, the tokens can be reduced by eliminating stop words, stemming and eliminating tokens shorter or longer than an specific size. Doing this will reduce the number of attributes almost in half, but event with this reduction generating 2-grams is not feasible.

This means that the pruning option of the operator has to be used to reduce more the number of attributes. Notice that each author has 25 examples for selecting the limits for pruning attributes. Think also that generating 2-grams will increase a lot the total number of attributes.



You can think about what are the effects of this pruning and what do you gain or lose from doing this. For example, filtering words that appear only in a few documents can eliminate specific words like the name of the characters, but also can eliminate the words that configure the personal style of the author. Words that appear in many documents probably will result in more general classifiers because they are words that more probably will appear in new documents. Also different goals probably will need more or less attributes to be learned and different number of specific and general attributes.

The last thing that you will need to decide for this operator is what information to use as attribute values.



The different values that can be assigned to each attribute can result in different performances for the learning task. You probably will need to test what happens for each one. Also you will need to think about how each learning algorithm processes the different types of attributes. Some algorithms increase their time complexity if the attributes are for example continuous.

This preprocessing steps will reduce the number of attributes not taking in account the learning goal. Attribute selection algorithms can be used to reduce more the size of the dataset. You can experiment with some of the filter and wrapper algorithms that are implemented in RapidMiner.



You will need to think about the time complexity issues for this kind of algorithms. Obviously filter methods are less expensive, but the way the ranking functions is computed is important. For this algorithms you will need also to decide how many attributes to keep.

It will take a lot of time to process thousands of attributes using a wrapper, this means that if you want to use it, you will need to reduce the number of attributes before applying it. Is it enough to eliminate attributes using the pruning of the *Process documents from files*? Can a filter be a previous step to a wrapper? Is it necessary to evaluate a lot of attribute subsets to reduce the number of attributes?

You will also need to be careful with the learning algorithm you use in the wrapper. What algorithm will be more efficient?

The different preprocesses that you have performed allows you to have datasets of very different sizes



Usually the more the data, the better the results. But in this case probably a lot of attributes are just noise, irrelevant or redundant. Is this so? You will need to experiment the impact of the number of attributes in the final performance of the classifiers you will learn.

You can also perform other preprocesses, For example, if the attributes that you are using are continuous. You can think a little bit more about how is their actual statistical distribution.



Does the values of the attributes really cover all the possible range of values? Maybe a discretization of the attributes can have a good impact on the learning process. You can experiment with the discretization operators of RapidMiner and see what happens.

After all this preprocessing you can save some datasets for the learning process using the store operator from the Repository Access folder. Remember that in order to evaluate the test data you have to preprocess train and test dataset at the same time so they have the same attributes.

3.4.3 Learning and evaluating

You have generated different datasets that you think can be used to learn the different tasks and now you have to test if the different algorithms that you know from theory class are able to generate good models. Specifically you must test at least:

- Decision trees (use the implementation from weka folder *W-J48*)
- Random Forest (use the implementation from weka folder *W-RandomForest*)
- Boosting and bagging with decision trees
- K nearest neighbours
- Naïve Bayes

But you can experiment with other algorithms to see what happens (can a neural network compete with these algorithms in this domain?). Obviously if you use other algorithms it will suppose an extra on the grade of the assignment.



Some algorithms have some parameters to adjust, others have none. You have to think about the meaning of the parameters and decide what values of the parameters can be adequate for this scenario. Some parameters will be very obvious from the characteristics of the data (what is the maximum number of neighbours that can be used in K-nn for the authors task?). You will need to experiment to test the effect of your assumptions and to adjust the less obvious parameters.

Be careful because some values will make the experiment to take a lot of time to obtain the result. Is sampling a good idea? Can the results obtained with sampling be extrapolated to the whole dataset?

Some things that are interesting to know about this specific domain are:

- Usually naïve bayes is used in spam filtering, that is a similar task, but estimating probabilities with very few examples is hard (for example for the author task)
- The distance functions that usually work well for text classification are the cosine distance and the Jaccard similarity, this means that probably these distances will have better performance for k-nearest neighbours than others such as the euclidean distance (is that so?)
- Decision trees can have a hard time if there are many continuous attributes, discretization can usually help
- The decision trees used for boosting and bagging do not have to be good for the combination to give good results

The goal is to measure the accuracy that can be achieved in the different learning tasks using different algorithm and different scenarios (more or less preprocessing, attribute selection, discretization).

You have to measure the accuracy using 10-fold cross validation with the whole dataset for the parameters that you think work best for each algorithm. If the time needed for the experiments is large, you can use a sampled dataset for the experiments to help you to decide the best parameters. **Include** this information in you report.

You have also to measure the accuracy of the final classifiers for the two test sets.

To perform the crossvalidation is straightforward, you only have to use the crossvalidation process operator from the evaluation folder and choose the algorithm to be used to evaluate its accuracy. You can include the evaluation of the test sets in the same process or save the model and perform the test separately.

For example if you work with the training and test set at the same time you can use a process like in figure 3.1.

Do not expect astonishing accuracies, you have to consider that the dataset has been build from random chunks of text, but probably you will be surprised in some tasks.

Also think that any result that is no better than to predict the majority class or predict randomly is a bad result. It is possible that some algorithms do not obtain a better result than that, but you have to test enough variety of parameter values to be confident that an algorithm can not learn the target of the problem. This means that in your experiments is not enough to try a couple of things for an algorithm and if it does not work move to the next algorithm. You have at least to try to explain why the algorithm does not work well.

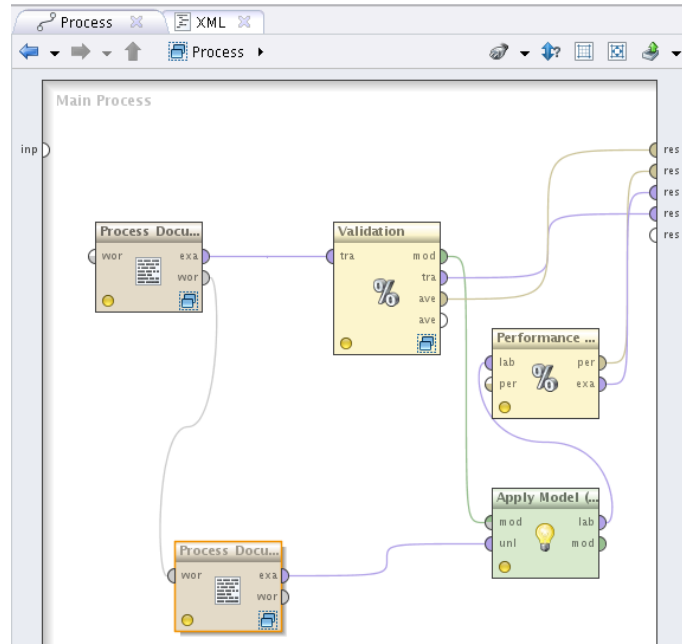


Figure 3.1: Processing the training and test set

3.5 Reporting your work

The problem has four different learning tasks, you have to perform experimentation for each one. Being all four different, probably you will need to do different things to obtain a good performance.

You have to write a report that explains all the experimentation work that you have done in detail, putting special emphasis in the questions that appear in the different sections of this assignment. You have also to describe the RapidMiner processes that you have used in your experiments.

Do not left any decision or explanation out from the report, and do not hesitate to include any ideas or thoughts about the results that you have obtained.

The additional questions that you will need to answer are the following:

1. Are some tasks more difficult to learn than others? What do you think are the reasons?
2. What tasks need more attributes to obtain a better performance? Are these the more difficult ones?
3. What algorithm performs better? What do you think are the reasons?
4. Are the crossvalidated accuracy related to the accuracy from the test examples? If not, try to guess the reasons.
5. Inspect the attributes used in the best classifiers for the genre (you can do this only with decision trees). Are they too specific (names, places) or are really words used exclusively in each genre?
6. Look into the author predictions for the examples of new authors. Are they really similar?

RapidMiner

A.1 What is Rapidminer

RapidMiner is an open source environment for machine learning and data mining. You can do with RapidMiner all the processes involved in a project: data acquisition, data transformation, data selection, attribute selection, attribute transformation, learning/modeling and validation.

RapidMiner has also some plug-ins available developed by the community for processing different types of data like temporal data, text processing or web mining.

A.2 Where can I download RapidMiner?

RapidMiner can be download from the webpage of Rapid-I (<http://rapid-i.com/>). You can download the windows version if you are going to work with windows (ugh!) or the other systems version that can be installed in any OS that has java installed. In order to download the software you do not need to register.

You can also download RapidMiner directly from source forge, the link is:

<http://sourceforge.net/projects/rapidminer/>

The plug-ins can be download once you have installed RapidMiner using the automatic update.

A.3 Where can I find more information about RapidMiner?

There are plenty information in the rapid-I website. You will find a small manual and several videos about how to use the basic operators of RapidMiner. You will find also links to other sites with more specialized videos.

A.4 How can I increase the amount of memory that RapidMiner uses?

By default RapidMiner uses an amount of memory that can be small for some experiments. You can increase the amount of memory used by defining the environment variable `MAX_JAVA_MEMORY` to the value that you need.

You can also modify this value inside the scripts that run Rapidminer. You can find this files in the directory `scripts` inside the installation directory. The files that you have to modify are the ones named `RapidMinerGUI`, there is one for unix and other for Windows. You just need to define the value of the `MAX_JAVA_MEMORY` variable that you will find inside the file.