Automatic web content categorization
Machine Learning

Masumi Mutsuda Zapater

February 2012

Abstract

In the information era where large amounts of content are being generated in a daily basis, autonomous processes of sorting and categorizing become of high interest. This project consists in the design and implementation of a web content classification application including well-known document classification algorithms such as Naïve Bayes and Maximum Entropy and inspired by proven tools like Mallet or Rapidminer. Accuracy and scalability are the main goals of the project.

Contents

Chapter 1

Introduction

"FlowSight is a Massively Parallel Processing platform that scales using Commercial Off-The-Shelf Hardware and Directly Attached Storage." Near the end of the year 2011 the company was asked by one of their Telecom clients to extend the functionality of one of the features their application was offering: they wanted to know not only which sites where most visited by their subscribers, but also a generic categorization of those top sites.
One way of solving this task would be doing it manually by someone in the company by reading those top sites and applying categories to them. The problem of such a manual approach is that maybe today the client asked for the top 10 sites, but you never know when will they ask for the top 100000. Another way could be using the metadata information provided by the website itself, but not every site provides its category in the metadata. The best approach to solve this task in a real scenario ended up being the artificial intelligence. This project is the result of applying the power of machine learning algorithms to a real case scenario.

Chapter 2

Supervised learning

The main goal of the project was to be able to automatically classify webpages into different categories depending on their content. By the goal itself it is clear that Machine Learning algorithms might be the best approach to try to solve the problem. We could have tried to use some unsupervised algorithms to solve the task, but in this case we have a clear knowledge about what we want to classify, and we don't need to extract new knowledge from the data.

2.0.1 Client requirements

Being part of a real case scenario brought to this project some requirements we had to take into account. The original set of categories that we were asked to identify was the following.

`Social Networking`, `Email`, `Sport`, `News`, `Shopping`, `Weather`, `Travel`, `Maps`, `Photos`, `Instant Messaging`, `Entertainment`, `Financial`, `Adult`, `Other`

Before going any further and just looking at the categories we were asked to classify, we can observe that some of them could be more difficult to classify than others. Some categories like `Adult` have a clearly characteristic language in their content whereas other categories like `Email` might contain generic language. We can also see that the category `Entertainment` might be overlapping with others like `Photos`, `Travel`, `Social Networking`, etc. Another of the difficulties of the task seems to be categorizing something as `Other`, because generating a training set for such a category might be difficult. But before being able to do any training, we needed labeled data, that is content from where to learn.

2.0.2 Crawling training data

Supervised Machine Learning algorithms need a set of training (labeled) data to infer the function that will later determine whether a new example is from one class or another. In this case we needed web content for each of the categories. There were different approaches to solve the problem. One of them could have been querying categories in search engines or navigating through blog entries matching each category. We finally decided to use the **Yahoo! Directory**[6] as a source for this data. Before search engines like Google existed, people used directories to find what they were

looking for. These directories are characterized as being a hierarchical structure for navigating through categories and subcategories. Under each category there is a list of URL related to it. Crawling every URL under a given category provided a huge and reliable corpus to train the model. I used a python to automate the process of crawling the Yahoo directory, manually checking which categories I wanted to crawl, and writing those initial URL into a seed file that was later automatically parsed.

```
...
http://dir.yahoo.com/Recreation/Sports/,sport
http://dir.yahoo.com/News_and_Media/,news
http://dir.yahoo.com/Business_and_Economy/Shopping_and_Services/
    Auctions/,shopping
http://dir.yahoo.com/Business_and_Economy/Shopping_and_Services/Music
    /,shopping
http://dir.yahoo.com/Business_and_Economy/Shopping_and_Services/
    Jewelry/,shopping
http://dir.yahoo.com/news_and_media/weather/,weather
http://dir.yahoo.com/Recreation/Travel/,travel
http://dir.yahoo.com/science/geography/cartography/maps/,maps
http://dir.yahoo.com/arts/visual_arts/photography/,photos
http://dir.yahoo.com/computers_and_internet/software/internet/
    instant_messaging/,instant_messaging
http://dir.yahoo.com/business_and_economy/finance_and_investment/,
    financial
http://dir.yahoo.com/business_and_economy/shopping_and_services/sex/
    adult_galleries/,adult
...
```

Once I had the training set and for initial testing purposes that could help taking later decisions I used rapidminer[5], a fast-to-setup and easy-to-use machine learning tool that, among other things, allowed me to test different machine learning algorithms with the same datasets in order to compare their performance.

2.0.3 Tests with Rapidminer

Rapidminer[5] is a widely used, easy to use and powerful machine learning environment. It was the perfect tool to do some initial tests in order to diagnose how might some categories be overlapping, detecting which algorithms performed better, etc. Almost all training processes were performed with a 10 fold cross validation in order to prevent overfitting and obtaining realistic accuracy values.

Feature vector creation

In order to obtain a precise and accurate document representation, and before selecting the features, several pre-processes need to/can be applied to the word set.

- **Tokenize**: the first step is to tokenize all documents. That is, split the text into a sequence of tokens.
- **Transform cases**: transforms all words to lowercase.
- **Stopword filter**: this process will remove common words. This step is language dependant.
- **Stem**: this process reduces inflected or derived words to their stem, base or root. It is also language dependant.

Figure 2.1: Rapidminer process

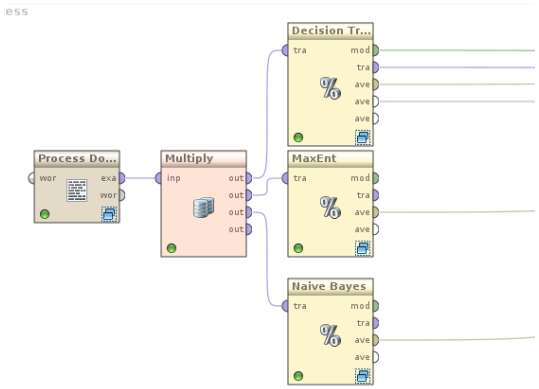
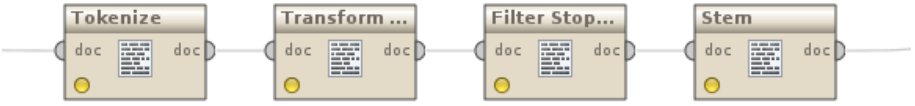


Figure 2.2: Vector creation



Number of features

The first experiment consisted in determining whether the amount of features was important and which was the optimal amount. It was also useful to determine which vector creation algorithm was more suitable for the process. Starting with 100 files for each category, the following plot represents the model accuracy over number of documents in the training set, using two different feature selection methods: `TD-IDF` and `Term Frequency`.

The conclusion of this initial test was that the more data the better. We can also observe that when the training set size is small it is not clear which feature selection method is better, but as the dataset grows it is clear that word frequency performs better, and also faster.

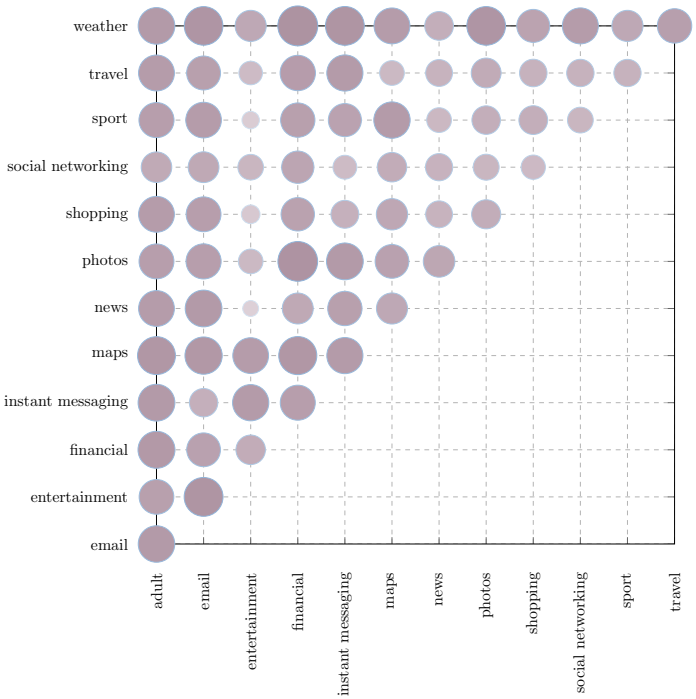
K nearest neighbours

Face to face accuracy

Another experiment consisted in comparing all classes binarily to determine which classes are more likely to be confused. In the following graph each circle represents a classification between the class in the y-axis and the class in the x-axis. Smaller and more transparent circles represent worse accuracy, and bigger and stronger circles represent better accuracy.

As we can see, classes like `Entertainment` are more likely to be confused with any other class, being the distinction between the classes `News` and `Entertainment` the

Figure 2.3: Accuracy heat graph



most difficult to perform. The reason why the `News` are so easy to confuse might be the fact that their content depends on what is trending, and their sites do not contain specific language. `Entertainment` is likely to be confused with `Sport` or `Shopping` because a lot of people consider them entertainment activities. As a concrete example, the following tree represents the WJ-48 pruned tree output that classifies `Weather` and `News`.

```
weather <= 0.068359
|   forecast <= 0.030773
|   |   snow <= 0.035962: news (511.0/30.0)
|   |   snow > 0.035962
|   |   |   stori <= 0.033615
|   |   |   |   watch <= 0.01856: weather (14.0/1.0)
|   |   |   |   watch > 0.01856: news (3.0)
|   |   |   stori > 0.033615: news (8.0)
|   forecast > 0.030773
|   |   engin <= 0.017747
|   |   |   edit <= 0.015559: weather (19.0/1.0)
|   |   |   edit > 0.015559: news (2.0)
|   |   engin > 0.017747: news (4.0)
weather > 0.068359
|   news <= 0.153393: weather (163.0/17.0)
|   news > 0.153393
|   |   copi <= 0.068359: weather (4.0)
|   |   copi > 0.068359: news (7.0)
```

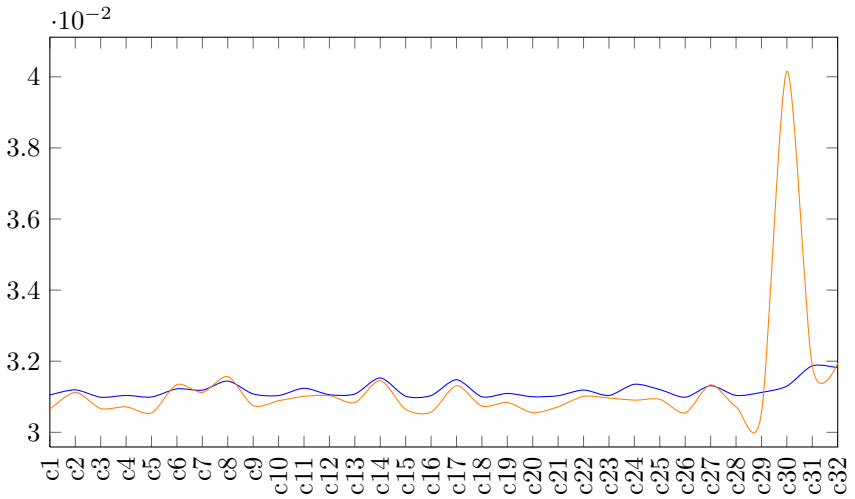
Entertainment into subcategories

As we saw in the previous section, the category `Entertainment` is one of the most difficult to distinguish. This could be due to the fact that it is a wide category containing a lot of different subcategories. We could try to split `Entertainment` into subcategories (such as `Music`, `Movies`, `Literature`...) to get a more precise dataset. In order to meet the client requirements those subcategories should be remapped to the original `Entertainment` category after the classification process.
After splitting the `Entertainment` category into subcategories, these were the new classes I found:
`Music`, `Literature`, `Theme parks`, `Movies`, `Food & Drink`, `Games`, `Television`
As new classes are added, it is clear that the overall accuracy of the classification may be worse, but with this new structure the confusions are likely to be between categories belonging to the same supercategory, which is not a problem.

The `Other` category

There are several approaches in order to represent the `Other` category, one of them could be creating a new class named `Other` and filling it with content from categories that differ from the ones we want to classify.
Another approach is taking the accuracy of a classification into account, and using a threshold to decide whether some sample should be considered of a any class or assume it should be considered of another class. By calculating the variance of the probability distribution through classes, we can determine the strength with which a classification is done.

Figure 2.4: Probability distribution of two instances



This could be done by looking at the variance of the model. Looking at the plot (figure 2.4) we could say that the blue colored classification is not really confident whereas the orange colored classification seems to be sure assigning the `c30` category. In this particular case we could assign the blue instance to the `Other` category, because it is more or less equally unsure of its category.

2.0.4 Implementation in the Framework

Although decision trees and Naive Bayes performed well during the rapidminer tests, they have some issues inherent to their specification. Decision trees are accurate, but their performance is not suitable for an almost real time high demanding environment. Naive Bayes is better at performance but it makes independence assumptions that could cause some classification problems.
There is an algorithm though that is both efficient and doesn't make any assumption. It's called Maximum Entropy (or Logistic Regression).

The Maximum Entropy

Why maximum entropy? Maximize entropy = Minimize commitment

Model all that is known and assume nothing about what is unknown. Model all that is known: satisfy a set of constraints that must hold respecting uniformity.

Assume nothing about what is unknown: choose the most "uniform" distribution choose the one with maximum entropy

One nice aspect of maximum entropy is that it does not suffer from any independence assumptions.
For example, consider a document containing the same number of occurrences of `rainy` and `social network`.
A Naive Bayes classifier will double count the occurrences of `social network` - it will add in the weight for `social` and the weight for `network`. Since `rainy` and `social network` occur equally, a single occurrence of `social network` will contribute twice the weight as an occurrence of `rainy`. This will cause Naive Bayes to prefer one class incorrectly. Maximum entropy, on the other hand, will discount the λ_i for each of these features such that their weight towards classification is appropriately reduced by half. This is because the constraints work over expectations of the counts. One implication for this freedom from independence assumptions is that bigrams and phrases can be easily added as features by maximum entropy, without worry that the features are overlapping.

MaxEnt Algorithm

- Compute $d_j, j = 1, ..., k+1$
- Initialize $\lambda_j^{(1)}$ to zero
- Repeat until converge
- For each j
  - Compute $E_{p^{(n)}} f_j = \sum_{x \in \varepsilon} p^{(n)}(x) f_j(x)$ where $p^{(n)}(x) = \frac{e^{\sum_{j=1}^{k+1} \lambda_j^{(n)} f_j(x)}}{Z}$
  - Update $\lambda_j^{(n+1)} = \lambda_j^{(n)} + \frac{1}{C}(log \frac{d_i}{E_{p^{(n)}} f_j})$

Feature selection

After observing the feature selection performance in the previous chapter, maybe the best choice for a fast vector creation is counting the term frequency. It usually performed better, and it always was faster.

Performance

Compared to other classification algorithms like Naive Bayes, the Maximum Entropy algorithm has a higher computational requirement. The platform where the MaxEnt classification algorithm is running is highly scalable and follows the map/reduce paradigm. Taking advantage of the paradigm, the MaxEnt algorithm is able to perform notably faster.

Figure 2.5: 2D Performance graph

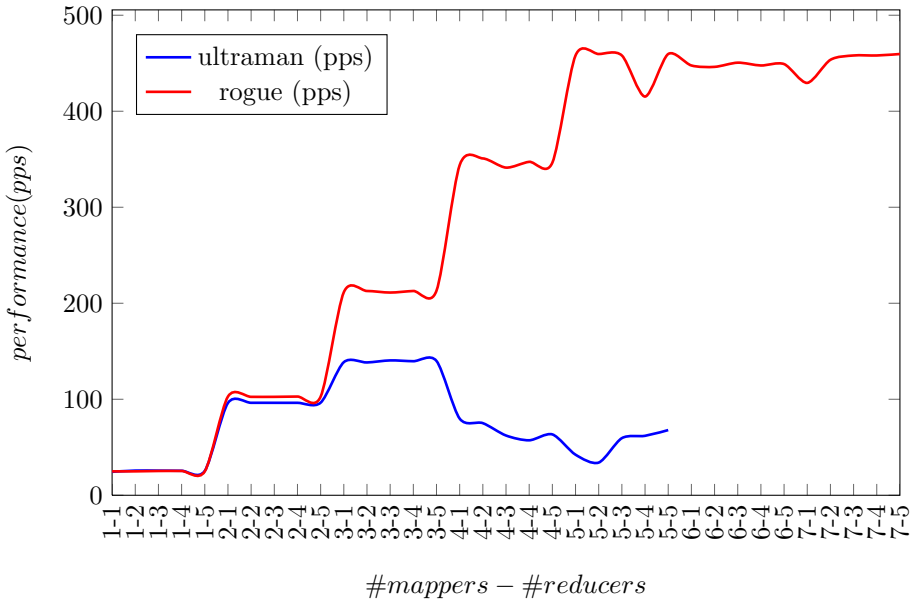
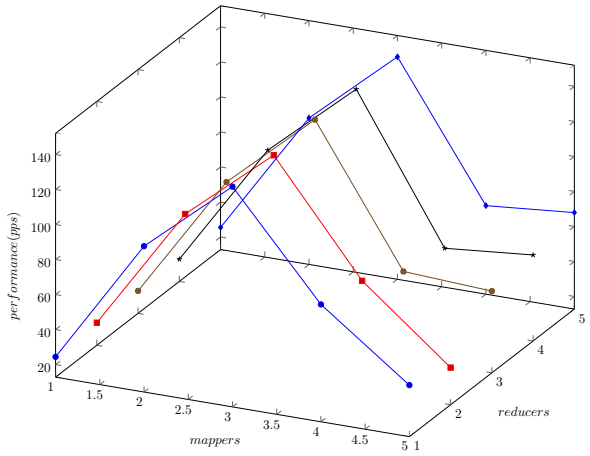


Figure 2.5 represents the execution time of the same algorithm with different number of mappers and reducers, in two different machines: `ultraman (Intel Core 2 Duo CPU 2.00GHz 4GB RAM)` is the machine where I'm writing most of this document, and `rogue (Intel Core 2 Quad CPU Q9300 @ 2.5GHz 8GB RAM)` is a more powerful machine used for testing purposes. The `x` axis represents the different mapper and reducer combinations I tested while the `y` axis represents the number of categorized pages per second. As we can see both machines perform similary when using one to two mappers, but the most powerful machine outperforms the other one when using more than two mappers. We can also observe how `ultraman`'s performance quickly stalls for more than thre mappers, while `rogue` is able to hold a rising performance until six mappers are used.

Figure 2.6 is a three dimensional representation of `ultraman`'s performance. We can quiclky observe a repeating pattern for each number of mappers.

Figure 2.6: 3D Performance graph



Chapter 3

Language drawbacks

There are several difficulties in the classification task that can be optimized by some refination. The first difficulty is that some servers returned their content in the language of the country they detected the request was coming from. Changing the header in the requests and using a proxy solved this problem. Another difficulty in this classification task is the fact that the content of some of the URIs that are being classified is written in languages different from english. As in this specific problem the classifier is trained to work in English, this situation will potentially cause wrong classifications. We could just filter those URIs from foreign domains (`.cz, .jp, .cn`) and automatically discard them, but some of those webpages could be written in english, and we would still have the same problem with the generic `.com` domain. A better approach would be identifying the language itself and discarding the webcontents which are not in english.

3.1 Language Identification

Language identification is the process of determining which natural language given content is in. There are several computational/statistical approaches to solve this problem which is split into two different stages: modeling and classification

3.1.1 Modeling techniques

Before being able to classify languages we need to generate models that will represent each language. This can be done using different techniques:

- **Common words technique**: the most common words in a corpus are sorted by frequency in order to get a probability distribution.
- **N-Gram technique**: similar to common words technique but instead of sorting the most common words, the most common successions of N characters are sorted.

The disadvantage of the **common words technique** is that although common words may occur in large amounts of text, they might not occur in shorter input examples. With the **N-Gram** technique we avoid that disadvantage by not using just the words but also all the N-partitions in the text.

3.1.2 Classification techniques

Once we have a model for each language we want to identify, we have to classify them.

- **Relative entropy**: compares the compressibility of the text we want to identify to the compressibility of the previously generated models.
- **Rank order statistics**: generates a probability by comparing how far an N-Gram is from the position of the same N-Gram in the model.

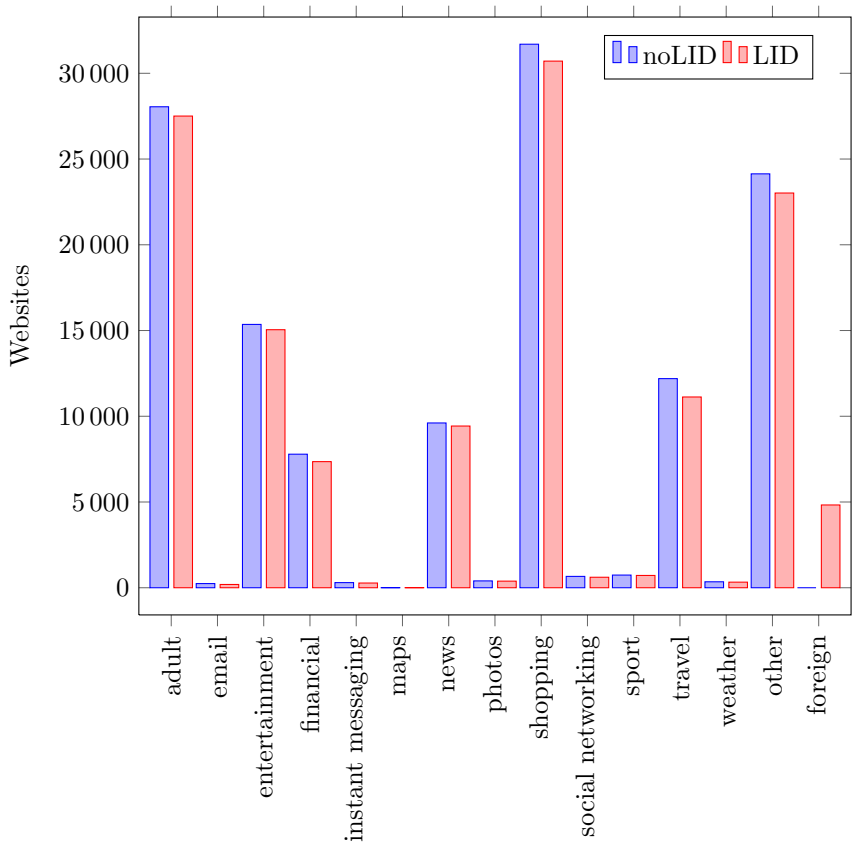
3.1.3 Application

For the scope of this project I chose the **3-Gram technique** to generate the language models and the **Rank order statistics** to identify the languages. I used a Python implementation by Damir Cavar[3] consisting of a model generator and a classifier. For generating the language models, I used the Wikipedia as a text source by automatically accessing random articles and crawling their content. After generating the models a simple Python algorithm rewrote the output file with the language information.

```
import lid
import os
import csv

myLid = lid.Lid()
output_file = "language_categorized.csv"
input_file = "categorized.csv"

seed_write = csv.writer(open(output_file, 'wb'), delimiter=',',
    quotechar='|', quoting=csv.QUOTE_MINIMAL)
seed_read = csv.reader(open(input_file, 'rU'), delimiter=',')

for row in seed_read:
  try:
    # This call returns the language of the file content
    language = myLid.checkText(open("unknown-data/http:"+row[0]+".out
        ").read())
    if (language != "English"):
      seed_write.writerow([row[0],"foreign_language"])
    else:
      seed_write.writerow([row[0],row[1]])
  except Exception, e:
    print "error identifying language of "+row[0]
    print e
    seed_write.writerow([row[0],"other"])
```

Before applying the LID algorithm, websites in other languages were being categorized as random categories, adding unuseful noise to the results. After applying the algorithm, those websites were categorized as `foreign_language` and were easily identified. We can see how the foreign language web pages were wrongly classified both in absolute values and percentage.

Figure 3.1: Applying the LID algorithm



Chapter 4

Further issues

After applying all sorts of refining to the classificaction there are still some sites that are not being classified correctly. A clear example is the url "youtube.com". Whereas it is supposed to be a social network or entertainment website, the content and text in it is generated by the users themselves and just a small percentage of the content in the website talks about what it really is. This causes the website to be classified differently depending on the "social trends" of whenever the content was crawled.

Chapter 5

Conclusions

Bibliography

[1] Ronald L. Graham, Donald E. Knuth and Oren Patashnik,*Concrete mathematics* Addison-Wesley, Reading, MA, 1995.

[2] alias-i, "Logistic Regression Tutorial", *Lingpipe Home*, 23 Nov. 2011, http://alias-i.com/lingpipe/demos/tutorial/logistic-regression/read-me.html

[3] Damir Cavar, "Language Identification LID Examples", *Personal Site*, 9 Jan. 2012, http://www.cavar.me/damir/LID

[4] Arjen Poutsm, "Applying Monte Carlo Techniques to Language Identification", *Personal Site*, 9 Jan. 2012, http://ajwp.home.xs4all.nl/langident.pdf

[5] Rapid-I, "Rapidminer", *Rapidminer site*, 9 Nov. 2011, http://rapid-i.com/content/view/181/190

[6] Yahoo! Directory, "Yahoo", *Yahoo*, 10 oct. 2011, http://dir.yahoo.com/

[7] Jason D. M. Rennie, Lawrence Shih, Jaime Teevan, David R. Karger, "Tackling the Poor Assumptions of Naive Bayes Text Classifiers", *MIT*, 19 Jan. 2012, http://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf

[8] Dan Klein, Chris Manning, "Maxent Models, Conditional Estimation, and Optimization", *Stanford University*, 19 Jan. 2012, http://www.cs.berkeley.edu/ klein/papers/maxent-tutorial-slides.pdf

[9] Joshua Goodman, "Exponential Priors for Maximum Entropy Models", *Microsoft*, 19 Jan. 2012, http://acl.ldc.upenn.edu/N/N04/N04-1039.pdf

[10] Kamal Nigam, John Lafferty, Andrew McCallum, "Using Maximum Entropy for Text Classification", *Carnegie Mellon University*, 19 Jan. 2012, http://www.kamalnigam.com/papers/maxent-ijcaiws99.pdf

[11] Alec Go, Richa Bhayani, Lei Huang, "Twitter Sentiment Classification using Distant Supervision", *Stanford University*, 19 Jan. 2012, http://www.stanford.edu/∼alecmgo/papers/TwitterDistantSupervision09.pdf

Chapter 6

Annex: Diary

This annex contains the diary notes I took during the project development. It is written in catalan and it may contain partial or disconnected information. Reader discretion is advised.

- Rapidminer per testejar diferents solucions, naive bayes rapid, maxent lent pero + accuracy
- Categoritzacio URL per contingut -¿ MaxEnt
- Classificador basat en categories de Yahoo Directory (training)
- Testing inicial amb petites dades
- Testing posterior amb 140.000 URLs d'Irlanda O2
- Descartar idiomes estrangers -¿ Trigraphs
- Fitxers amb trigrams d'idiomes generats via randomly crawling wikipedia
- Crawling a traves de proxy amb headers tema idioma
- Afegida categoria adult
- Afegit categoria other, massa generica
- Desmembrat categoria other en altres per posterior mapeig a other
- Generat script per calcular mitja de certesa de classficacio i distancia mitja amb segona opcio
- Millorats fitxers de train en base al punt anterior

El primer approach per la classificació va ser fent servir les categories que demanava O2 irlanda. La categoria entertainment englobava molts temes, movies, tv, music, literature..., i es requeria a més una categoria "Other". El classificador sempre assigna una categoria, la que té la màxima probabilitat de ser, i per tant s'havia de generar la categoria other a partir de categories que no tinguessin res a veure. El problema de crear categoria generica other o entertainment, es que passaven a tenir molt ambigüitat, i moltes coses passaven a considerar-se other i entertainment. La solucio va ser crear subcategories sense tenir en compte entertainment i other, i fer un post tractament del ficher resultant de la classificació assignant other o entertainment a allò que realment ho era. D'aquesta manera es té més granularitat, tot i que l'accuracy del classificador baixa al tenir més categories. Un problema comú és que algunes urls no es deixen crawlejar. Detecten que no ets un navegador corrent i et donen un contingut que no és significatiu. A l'hora de classificar aquestes urls acaben sent categoritzades

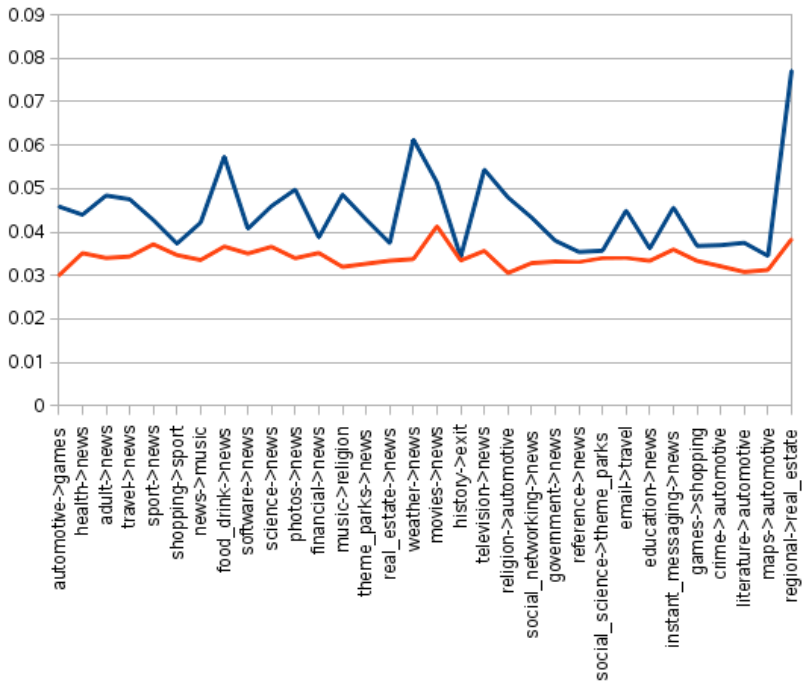
a categories que no tenen res a veure. Per exemple wikipedia retorna simplement una llista de països, que fa que la categoria passi a ser "Adult". Coses similars passen amb facebook. A youtube el problema és que el text crawlejat són els títols i descripcions de videos que la gent puja, per tant depenent de quan es produeixi el crawling, la classificacio de Youtube pot canviar.

29-12-2011

En afegir categories concretes per a ser englobades posteriorment per Entertainment, l'accuracy del classificador ha baixat fins al 73.33 adult education food_drink health literature music real_estate religion social_networking sport travel automotive email games history maps news reference science social_science television weather crime financial government instant_messaging movies photos regional shopping software theme_parks

Si la confusió és entre categories que seran englobades per "Entertainment" no hi ha problema. Observem que la distància disminueix lògicament perquè ara hi ha més categories.

Figure 6.1: Gràfic de blabla



05-01-2012

Proves a jabato