UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and Computer Science

Bachelor degree in
Computer Science

FINAL ELABORATE

MATCHING ITALIAN COMPANIES WITH FACEBOOK PAGES AND USERS

Machine Learning Classifier: a Random Forest Approach

Yannis Velegrakis
Supervisor

Michele Sordo
Graduand

Academic year 2015/2016

Acknowledgements

Contents

Summary

Nowadays companies are getting more and more connected with potential customers with the help of social networks. These instruments are not only supposed to help people getting linked and chat together but also to give public figures and companies a way to show themselves and to get promoted. Social Networks directly provide services for advertisement: Facebook Ads for Facebook, Twitter Ads for Twitter etc. These services are developed to help people and organizations promoting themselves on that particular platform and usually aren't very useful as a platforms itselves to manage an entire lead generation campaign. Companies are supported only in the brand and products promotion, meaning that the ones that want to find potential customers are generally left alone. The lead generation process, the generation of consumer interest or inquiry into products or services of a business , requires company to have a list of potential customers and, most important, a way to connect with them.

The aim of this project is to show a practical solution to connect real companies with their social pages. Such solution is meant to be implemented in a bigger platform for lead generation that will help companies in the path of finding other businesses that may be interested in buying specific products. The solution shown connects Italian companies with their Facebook page in order to enrich companies' description on Atoka[1], a leading tool developed by SpazioDati[12] that collects structured data and information of all the 6 million companies currently opened in Italy.

The first problem faced was to determine if companies really structure their Facebook data in pages or if they use Facebook users as pages. From a legal point of view it's against the Facebook Terms to use your personal account to represent something other than yourself (ex: your business), and you could permanently lose access to your account if you don't convert it to a Page. Taking care of the fact that Atoka aims to give the more information it can about a company, team decided to do a first exploratory analysis to determine if Italian companies really used only Facebook pages. The results show that a lot of businesses still use Facebook users as pages. The main reason found is that small companies can't afford to pay a single person to work on public relation, this cause the fact that the bigger companies are the more structured their Facebook data was. [3] The only way to get the data of a Facebook page or user is to query Facebook GraphAPI[8] by possible names of it. The project was developed by querying Facebook with a lot of Batch Requests in order to create a wide index of Facebook data locally available and structured to be compared with the existing company index. One of the greatest problem was to structure the data in a reasonable way: most of Facebook pages don't have the data in the proper place: as said before the creation of the page is usually done by non-expert people and this lack of expertise lead to have everything in the main section, without the division in the proper fields provided by Facebook. An example of the description field that collects everything without the division in fields can be:

Hi! We are NAMEOFCOMPANY, you can find us in ADDRESS. Our telephone number is: NUMBER.

The main techniques used in this project are described in the implementation section. The classification algorithm that best fits in this analysis is Random Forest and the paper tries to explain why it's the best choice available. This project had an important effect on Spaziodati companies' index, finding Facebook pages for more than 800% of the previous values indexed. The Graduand worked on this project for the entire duration of the internship at Spaziodati (plus more months at the end

Struttura Ateco 2007	
Codice Ateco 2007	**Descrizione**
56	**ATTIVITÀ DEI SERVIZI DI RISTORAZIONE**
56.1	**RISTORANTI E ATTIVITÀ DI RISTORAZIONE MOBILE**
56.10	**Ristoranti e attività di ristorazione mobile**
56.10.1	Ristorazione con somministrazione; ristorazione connessa alle aziende agricole
56.10.11	Ristorazione con somministrazione
56.10.12	Attività di ristorazione connesse alle aziende agricole
56.10.2	Ristorazione senza somministrazione con preparazione di cibi da asporto
56.10.20	Ristorazione senza somministrazione con preparazione di cibi da asporto
56.10.3	Gelaterie e pasticcerie
56.10.30	Gelaterie e pasticcerie
56.10.4	Ristorazione ambulante e gelaterie ambulanti
56.10.41	Gelaterie e pasticcerie ambulanti
56.10.42	Ristorazione ambulante
56.10.5	Ristorazione su treni e navi
56.10.50	Ristorazione su treni e navi
56.2	**FORNITURA DI PASTI PREPARATI (CATERING) E ALTRI SERVIZI DI RISTORAZIONE**
56.21	**Fornitura di pasti preparati (catering per eventi)**
56.21.0	Catering per eventi, banqueting
56.21.00	Catering per eventi, banqueting
56.29	**Mense e catering continuativo su base contrattuale**
56.29.1	Mense
56.29.10	Mense
56.29.2	Catering continuativo su base contrattuale
56.29.20	Catering continuativo su base contrattuale
56.3	**BAR E ALTRI ESERCIZI SIMILI SENZA CUCINA**
56.30	**Bar e altri esercizi simili senza cucina**
56.30.0	Bar e altri esercizi simili senza cucina
56.30.00	Bar e altri esercizi simili senza cucina

Figure 1.1: Subdivision of ATECO 56.

of it), from the exploratory analysis at the beginning to the integration in the previous workflow.

1   Introduction

One important part of companies' public relations is being connected with potential customers and make it easy to be found in the huge quantity of data available by potential clients. In a connected world, where many companies have a Facebook page (and many a website too), it's difficult to find somebody interested in buying your product or solution. As introduced before, social networks provide advertising services that are vertical to the particular platform (ex. Facebook, Twitter, etc.) but a marketing platform for lead generation service needs to link social networks' data with existing companies legally registered in the real world.

The paper analyze the process of connecting Italian companies with their Facebook pages or user profile. The aim of the project is to create a model that can automatically find if a Facebook page is the official (and real) page of a determined company. This job is very simple if it's done by a human but it can't be done by people because of time lacking: nobody could find the right match for more than 6 million companies in a reasonable amount of time. The only way to overcome human computational time and cost is to develop a prediction algorithm that can learn by itself to recognize an official page (or user) from a set of potential pages (and users).

The paper analyze only the subset of companies that has ATECO code n.56. ATECO n.56 is the code that collect companies that operate in the food sector[11]. This limitation is caused by time reason: the classification time that the VPS will spend analyzing all the 6 million companies will probably be more than 20 days. Apart from time causes the process needs to be integrated with
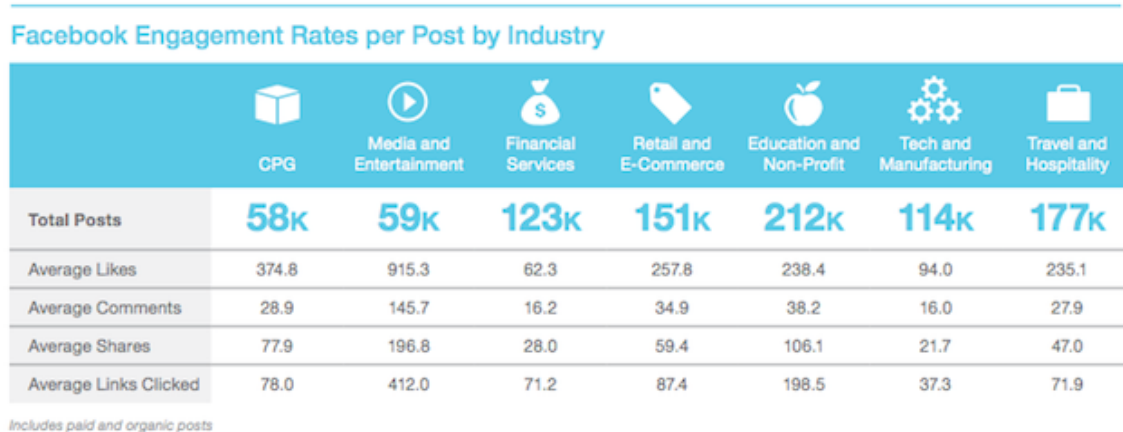
Facebook Engagement Rates per Post by Industry

	CPG	Media and Entertainment	Financial Services	Retail and E-Commerce	Education and Non-Profit	Tech and Manufacturing	Travel and Hospitality
Total Posts	58K	59K	123K	151K	212K	114K	177K
Average Likes	374.8	915.3	62.3	257.8	238.4	94.0	235.1
Average Comments	28.9	145.7	16.2	34.9	38.2	16.0	27.9
Average Shares	77.9	196.8	28.0	59.4	106.1	21.7	47.0
Average Links Clicked	78.0	412.0	71.2	87.4	198.5	37.3	71.9

Includes paid and organic posts

Figure 1.2: Facebook Engagement Industry

Spaziodati's information retrieval pipeline and the automation process made with Azkaban Flow is not fully described in this paper mainly for time, space and copyright reasons. Some data contained in the paper will also be more qualitative than quantitative in order to respect the non disclosure agreement signed with Cerved Group SPA for the usage of the companies' data index.

This problem was an important step in upgrading company information available on Atoka and gained priority at SpazioDati, making the company invest money and time to develop this project. This approach might also be extended to different information and usages in similar problems. The solution developed was born after discussion with SpazioDati's developers, that have big experience in comparable problems. The most similar problem is the one developed by Spaziodati to connect companies' data with websites and the solution section describes how we it helped us and how it saved us time. This solution was developed directly with a random forest implementation, taking care of the previous problem solution and hurdles found in the development of the Website part.

The final solution is a machine learning algorithm application that runs after a score generating application that compute how similar is a page/user data collected to the company data already suited in Atoka Index. The machine learning algorithm chosen to classify the data in this project is Random Forest Classifier and the solution will explain why, in this situation, a random forest approach is better than any other classification algorithms. This paper is written from a scientific point of view but the approach used in the development was a corporate one. For copyright reasons and time given to spend on the development of the project some measurements given in the experimental chapter may be more qualitative than quantitative.

2 Motivation

When it comes to lead generation, one of the main important aspects is to give to the service users the more way it can get to connect with potential customers. Most of the companies collected in the Atoka index have, apart from their Name, a complete field of legal address, sometimes the phone number (of the legal location) but they didn't have a lot of "modern communication" fields like email, website and, most of all, social media. Comparing all the social media used by companies, Facebook instantaneously looked the best because it's the more used and, particularly in Italy, it's the social that provides more connection between customers and businesses.

These are some of the last statistics about the impact of Facebook[13]

- Worldwide, there are over 1.71 billion monthly active Facebook users (Facebook MAUs) which is a 15 percent increase year over year. (Source: Facebook as of 7/27/16) What this means for you: In case you had any lingering doubts, statistically, Facebook is too big to ignore.
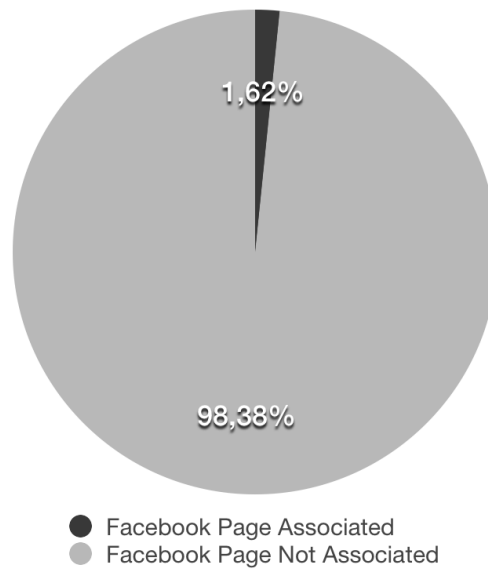
Facebook Pages in Atoka Index

Figure 2.1: Only 96310 over 5952128 companies in Atoka had a Facebook Page or User associated before this project.

- 4.5 billion likes generated daily as of May 2013 which is a 67 percent increase from August 2012 (Source: Facebook)

- 1.13 billion people log onto Facebook daily active users (Facebook DAU) for June 2016, which represents a 17% increase year over year (Source: Facebook as 7/27/16) The Implication: A huge and vastly growing number of Facebook users are active and consistent in their visits to the site, making them a promising audience for your marketing efforts.

- There are 1.57 billion mobile active users (Mobile Facebook MAU) for June 2016 (Source: Facebook as of 7/27/16) an increase of 20 percent year-over-year. There are 1.03 billion Mobile Daily Active Users (Facebook DAU) for June 2016 which is an increase of 22% year-over-year.

- On average, the Like and Share Buttons are viewed across almost 10 million websites daily. (Source: Facebook as of 10/2/2014)

- In Europe, over 307 million people are on Facebook. (Source: Search Engine Journal) The Takeaway: This isn't just a U.S. phenomenon - a worldwide market is available via Facebook.

Before the project, only 1.62% of Italian companies had a Facebook Page/User associated in Atoka Index. Having the company's Facebook page linked to their account on Atoka became soon a priority for Spaziodati, that decided to invest in this project. The previous knowledge about Facebook pages/user was collected analyzing the content of companies' official websites but was clearly not enough for a business product that aims to connect companies to new business customers. However, the mining of the website data was one of the most influent project over this research and the team adopted an approach very similar to the one used before.

As introduced in the previous chapter the subset chosen is the ATECO 56. This subset of companies is interesting to study because many restaurant, pubs and companies involved in the food sector:

- have a Facebook Page, or user created erroneously instead of a business page (now Facebook has built a tool that migrates a business user account to a formal Facebook page but not all the companies have used it so far).

- still don't have a website: this is a really fundamental task to improve contacts in Atoka and a great reason to find how many of them really have a Facebook Page.

Another interesting part of the problem is how to sample a training dataset for the classifier. In fact, as said before, there are only few companies with ATECO n.56 connected to a website and this implies that less of them have a Facebook page/user connected. The number of matched Facebook pages is 6,355, over a dataset of 362,154 companies with ATECO n.56, less than 2% of them. This is one of the main reasons of why the approach that have been choses was Random Forest.

The result that this project will produce were used to upgrade Atoka index data and, in consequence, data available on the online Atoka lead generation platform.

The problem was hard to solve for different reasons:

- Not all companies have a Facebook page, for most of them it's just looking for nothing

- Companies, instead of using the fields that Facebook provide them for optimizing their own indicization, use to put everything in the description field and so every potential field needs to be searched and normalized

- It's impossible to cluster the data because a restaurant have the same information of a trading agency or a freelance carpenter.

- A developer can't legally analyze post content without the page administrator permission so everything could be based only on the basic informations of the page.

3 Background

This chapter introduces some concepts that are useful for the comprehension of the next chapters.

3.1 ElasticSearch Index

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License.[5]

The basic index structure of Elasticsearch is the inverted index, a very versatile data structure. At the same time it's also easy to use and understand. That said, Lucene's implementation is a highly optimized, impressive feat of engineering. We will not venture into Lucene's implementation details, but rather stick to how the inverted index is used and built. That is what influences how we can search and index.[6]

Having introduced the inverted index as the "bottom" of the abstraction levels, we'll look into:

- How simple searches are performed.

- What types of searches can (and cannot) effectively be done, and why, with an inverted index, we transform problems until they look like string-prefix problems.

- Why text processing is important.

- How indexes are built in "segments" and how that affects searching and updating.

- What constitutes a Lucene-index.

- The Elasticsearch shard and index.

At that point, we'll know a lot about what happens inside a single Elasticsearch node when searching as well as indexing.

Figure 3.1: Sample documents and resulting inverted index

Inverted Indexes and Index Terms

Sample documents and resulting inverted index Let's say we have these three simple documents: "Winter is coming.", "Ours is the fury." and "The choice is yours.". After some simple text processing (lowercasing, removing punctuation and splitting words), we can construct the "inverted index" shown in the figure.

The inverted index maps terms to documents (and possibly positions in the documents) containing the term. Since the terms in the dictionary are sorted, we can quickly find a term, and subsequently its occurrences in the postings-structure. This is contrary to a "forward index", which lists terms related to a specific document.

A simple search with multiple terms is then done by looking up all the terms and their occurrences, and take the intersection (for AND searches) or the union (for OR searches) of the sets of occurrences to get the resulting list of documents. More complex types of queries are obviously more elaborate, but the approach is the same: first, operate on the dictionary to find candidate terms, then on the corresponding occurrences, positions, etc.

Consequently, an index term is the unit of search. The terms we generate dictate what types of searches we can (and cannot) efficiently do.

Building Indexes

When building inverted indexes, there's a few things we need to prioritize: search speed, index compactness, indexing speed and the time it takes for new changes to become visible.

Search speed and index compactness are related: when searching over a smaller index, less data needs to be processed, and more of it will fit in memory. Both, particularly compactness, come at the cost of indexing speed, as we'll see.

To minimize index sizes, various compression techniques are used. For example, when storing the postings (which can get quite large), Lucene does tricks like delta-encoding (e.g., [42, 100, 666] is stored as [42, 58, 566]), using variable number of bytes (so small numbers can be saved with a single byte), and so on.

Keeping the data structures small and compact means sacrificing the possibility to efficiently update them. In fact, Lucene does not update them at all: the index files Lucene write are immutable, i.e. they are never updated. This is quite different to B-trees, for instance, which can be updated and often lets you specify a fill factor to indicate how much updating you expect.

The exception is deletions. When you delete a document from an index, the document is marked as such in a special deletion file, which is actually just a bitmap which is cheap to update. The index structures themselves are not updated.

Consequently, updating a previously indexed document is a delete followed by a re-insertion of the document. Note that this means that updating a document is even more expensive than adding it in

the first place. Thus, storing things like rapidly changing counters in a Lucene index is usually not a good idea - there is no in-place update of values.

When new documents are added (perhaps via an update), the index changes are first buffered in memory. Eventually, the index files in their entirety, are flushed to disk. Note that this is the Lucene-meaning of "flush". Elasticsearch's flush operation involves a Lucene commit and more, covered in the transaction log-section.

When to flush can depend on various factors: how quickly changes must be visible, the memory available for buffering, I/O saturation, etc. Generally, for indexing speed, larger buffers are better, as long as they are small enough that your I/O can keep up2. We go a bit more into detail in the next section.

The written files make up an index segment.

Index Segments

A Lucene index is made up of one or more immutable index segments, which essentially is a "mini-index". When you do a search, Lucene does the search on every segment, filters out any deletions, and merges the results from all the segments. Obviously, this gets more and more tedious as the number of segments grows. To keep the number of segments manageable, Lucene occasionally merges segments according to some merge policy as new segments are added. Lucene-hacker Michael McCandless has a great post explaining and visualizing segment merging.3 When segments are merged, documents marked as deleted are finally discarded. This is why adding more documents can actually result in a smaller index size: it can trigger a merge.

Elasticsearch and Lucene generally do a good job of handling when to merge segments. Elasticsearch's policies can be tweaked by configuring merge settings. You can also use the optimize API to force merges.

Before segments are flushed to disk, changes are buffered in memory. In the old days (Lucene ¡2.3), every added document actually existed as its own tiny segment4, and all were merged on flush. Nowadays, there is a DocumentsWriter, which can make larger in-memory segments from a batch of documents. With Lucene 4, there can now be one of these per thread, increasing indexing performance by allowing for concurrent flushing. (Earlier, indexing would have to wait for a flush to complete.)

As new segments are created (either due to a flush or a merge), they also cause certain caches to be invalidated, which can negatively impact search performance. Caches like the field and filter caches are per segment. Elasticsearch has a warmer-API5, so the necessary caches can be "warmed" before the new segment is made available for search.

The most common cause for flushes with Elasticsearch is probably the continuous index refreshing, which by default happens once every second. As new segments are flushed, they become available for searching, enabling (near) real-time search. While a flush is not as expensive as a commit (as it does not need to wait for a confirmed write), it does cause a new segment to be created, invalidating some caches, and possibly triggering a merge.

When indexing throughput is important, e.g. when batch (re-)indexing, it is not very productive to spend a lot of time flushing and merging small segments. Therefore, in these cases it is usually a good idea to temporarily increase the refresh_interval-setting, or even disable automatic refreshing altogether. One can always refresh manually, and/or when indexing is done.

Elasticsearch Indexes

An Elasticsearch index is made up of one or more shards, which can have zero or more replicas. These are all individual Lucene indexes. That is, an Elasticsearch index is made up of many Lucene indexes, which in turn is made up of index segments. When you search an Elasticsearch index, the search is executed on all the shards - and in turn, all the segments - and merged. The same is true when you search multiple Elasticsearch indexes. Actually, searching two Elasticsearch indexes with one shard each is pretty much the same as searching one index with two shards. In both cases, two underlying Lucene indexes are searched.

From this point onwards in this article, when we refer to an "index" by itself, we mean an Elasticsearch index.

A "shard" is the basic scaling unit for Elasticsearch. As documents are added to the index, it

is routed to a shard. By default, this is done in a round-robin fashion, based on the hash of the document's id. In the second part of this series, we will look more into how shards are moved around. It is important to know, however, that the number of shards is specified at index creation time, and cannot be changed later on. An early presentation on Elasticsearch by Shay has excellent coverage of why a shard is actually a complete Lucene index, and its various benefits and tradeoffs compared to other methods.

Which Elasticsearch indexes, and what shards (and replicas) search requests are sent to, can be customized in many ways. By combining index patterns, index aliases, and document and search routing, lots of different partitioning and data flow strategies can be implemented. We will not go into them here, but we can recommend Zachary Tong's article on customizing document routing and Shay Banon's presentation on big data, search and analytics. Just to give you some ideas, here are some examples:

Lots of data is time based, e.g. logs, tweets, etc. By creating an index per day (or week, month, ...), we can efficiently limit searches to certain time ranges - and expunge old data. Remember, we cannot efficiently delete from an existing index, but deleting an entire index is cheap. When searches must be limited to a certain user (e.g. "search your messages"), it can be useful to route all the documents for that user to the same shard, to reduce the number of indexes that must be searched.

Summary

To summarize, these are the important properties to be aware of when it comes to how Lucene builds, updates and searches indexes on a single node:

- How we process the text we index dictates how we can search. Proper text analysis is important.

- Indexes are built first in-memory, then occasionally flushed in segments to disk.

- Index segments are immutable. Deleted documents are marked as such.

- An index is made up of multiple segments. A search is done on every segment, with the results merged.

- Segments are occasionally merged.

- Field and filter caches are per segment.

- Elasticsearch does not have transactions.

3.2 Random Forests

The random forest (Breiman, 2001) is an ensemble approach that can also be thought of as a form of nearest neighbor predictor. Ensembles are a divide-and-conquer approach used to improve performance. The main principle behind ensemble methods is that a group of "weak learners" can come together to form a "strong learner". The figure 3.2 provides an example. Each classifier, individually, is a "weak learner", while all the classifiers taken together are a "strong learner". The data to be modeled are the blue circles. We assume that they represent some underlying function plus noise. Each individual learner is shown as a gray curve. Each gray curve (a weak learner) is a fair approximation to the underlying data. The red curve (the ensemble "strong learner") can be seen to be a much better approximation to the underlying data.

Trees and Forests

The random forest starts with a standard machine learning technique called a "decision tree" which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets. In this example, the tree advises us, based upon weather conditions, whether to play ball. For example, if the outlook is sunny and the humidity is less than or equal to 70, then it is probably OK to play.

The random forest (see Figure 3.3) takes this notion to the next level by combining trees with the notion of an ensemble. Thus, in ensemble terms, the trees are weak learners and the random forest is a strong learner.
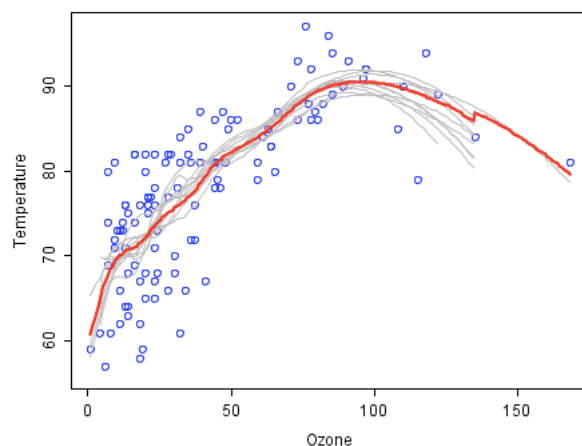
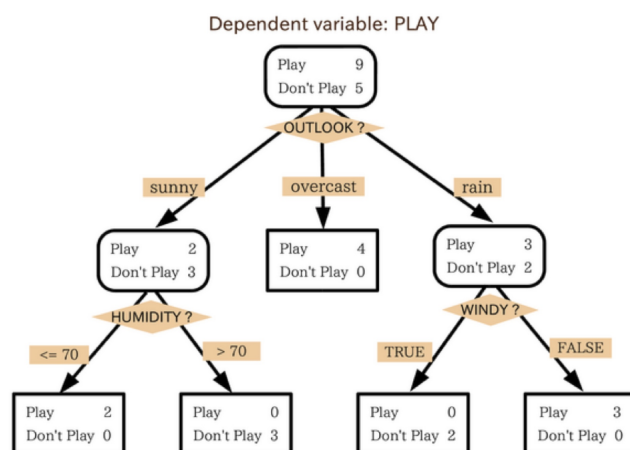Figure 3.2: Analysis on the relationship between ozone and temperature

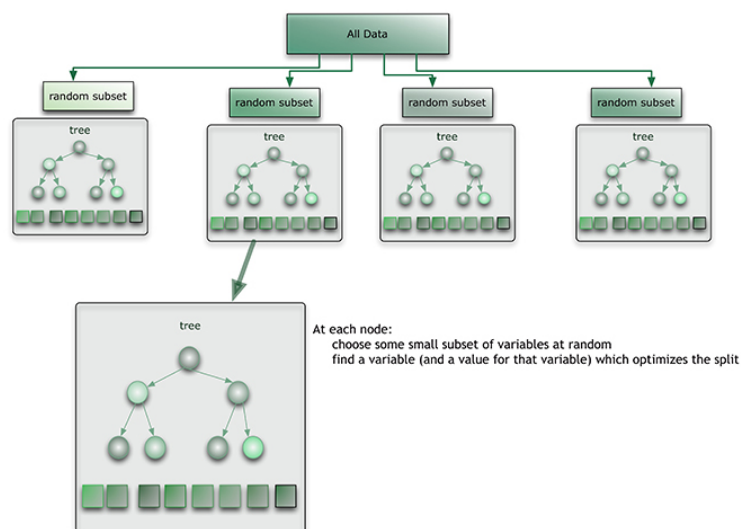Figure 3.3: Example of a single Tree of a Random Forest

Figure 3.4: Example of how a Random Forest is composed

Here is how such a system is trained; for some number of trees T:

1. Sample N cases at random with replacement to create a subset of the data (see top layer of figure above). The subset should be about 66% of the total set.

2. At each node:

   (a) For some number m (see below), m predictor variables are selected at random from all the predictor variables.

   (b) The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.

   (c) At the next node, choose another m variables at random from all predictor variables and do the same.

Depending upon the value of m, there are three slightly different systems:

- Random splitter selection: m =1

- Breiman's bagger: m = total number of predictor variables

- Random forest: m $\ll$ number of predictor variables. Brieman suggests three possible values for m: $1/2\sqrt{m}$, $\sqrt{m}$, and $2\sqrt{m}$

Running a Random Forest

When a new input is entered into the system, it is run down all of the trees. The result may either be an average or weighted average of all of the terminal nodes that are reached, or, in the case of categorical variables, a voting majority. Note that:

- With a large number of predictors, the eligible predictor set will be quite different from node to node.

- The greater the inter-tree correlation, the greater the random forest error rate, so one pressure on the model is to have the trees as uncorrelated as possible.

- As m goes down, both inter-tree correlation and the strength of individual trees go down. So some optimal value of m must be discovered.

Strengths and weaknesses

Random forest runtimes are quite fast, and they are able to deal with unbalanced and missing data. Random Forest weaknesses are that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy. Of course, the best test of any algorithm is how well it works upon your own data set.

3.3 Testing a Classifier

To understand how to test a classifier, several concepts have to be explained:

- cross-validation

- thresholds

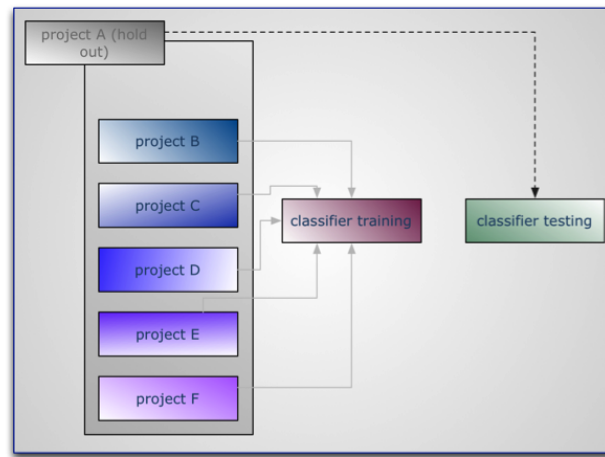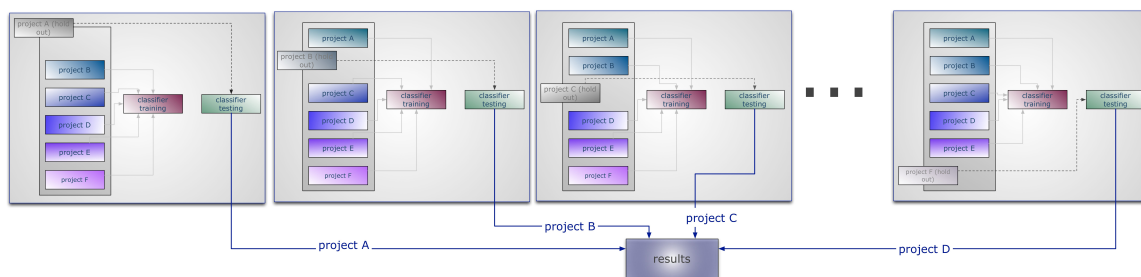- mean precision

- precision above chance

Figure 3.5:

Figure 3.6:

Cross-Validation

Our classifier test should be ecologically valid. It should prove our classifier under conditions as close as possible to the production environment.

In production the classifier is trained on many projects where we know the CTRs, but really we would like to know how the classifier predicts CTRs for a new project it has never seen before.

We test our classifier using a technique called "cross-validation": train the classifier on all projects except for one. Of course, we also know the correct CTRs for this held-out project, so we can see how well the classifier does on it, without cheating by training the classifier on the hold-out. We do this in turn with each project. See Figure 3.5.

Let's say we have projects A through F. We train a classifier on projects A through E, and test on F. Then we train on A and C through F, and test on B. We do this in turn, holding out each project, until we train on A through E and test on F. Each project is a subject in our experiment, with subjects A through F (see Figure 3.6).

Finally, we collect the results from each cross-validation run for statistical analysis. The experimental question we want to answer: does the classifier give us any more information about the CTR than just guessing?

Thresholds

The classifier predicts whether the input vector would result in a low or high CTR. The classifier's output is a number from zero to one, give or take. A zero predicts low CTR, a one predicts high CTR. But what do we do if the classifier outputs a value in between, say a 0.4 or a 0.6? We need to choose a threshold. Above this value we treat the classifier output as predicting high CTR. Below this value we treat the classifier output as predicting low CTR.

The classifier isn't perfect, so sometimes it will make mistakes. Sometimes the classifier will incorrectly predict that a high CTR item is low (a "miss"), and sometimes the classifier will incorrectly predict that a low CTR item is high (a "false alarm").
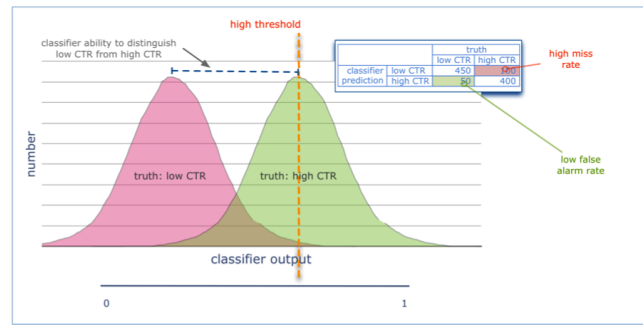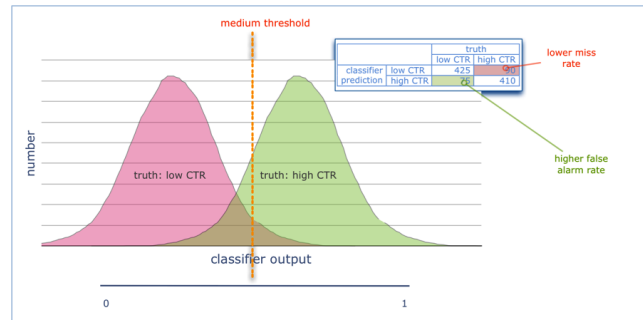
Figure 3.7:

Figure 3.8:

The following figures show a frequency distribution for the classifier output for both low and high CTR items. High CTR items, in green, tend to result in a classifier output towards 1. Low CTR items, in red, tend to result in classifier output towards 0. Note that in all three figures the distance between the two curves remains unchanged. All that changes is the threshold.

Assume we have a total of 1,000 data points, evenly split between low and high CTR. In the above figure, our threshold is rather high (say, 0.85). This results in a relatively low false alarm rate (only 50 items in total are false alarms), and a rather high miss rate (100 items that are in truth high CTR are missed).

In the next figure, below, moving the threshold to the left (say, to 0.75) means a higher false alarm rate (75) and a lower miss rate (90).

In the figure below, at the lowest threshold (say, 0.40), our false alarm rate goes up to 100, and our miss rate goes down to 75. So as we lower our threshold, our false alarm rate goes up from 50 to 75 to 100, while our miss rate goes down from 100 to 90 to 75.

We draw the threshold based upon which type of error we want to avoid more, misses or false alarms. A higher threshold means more misses and fewer false alarms. A lower threshold means the opposite. Changing the threshold does not change the underlying accuracy of the classifier, it just moves the error around.
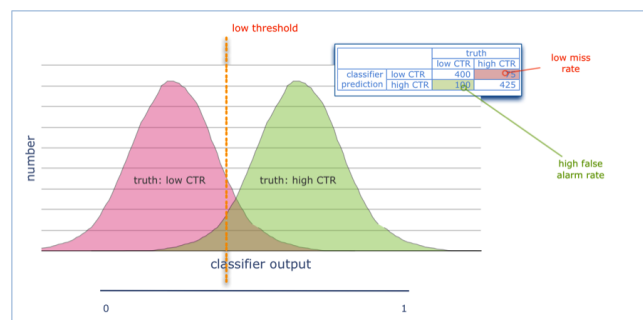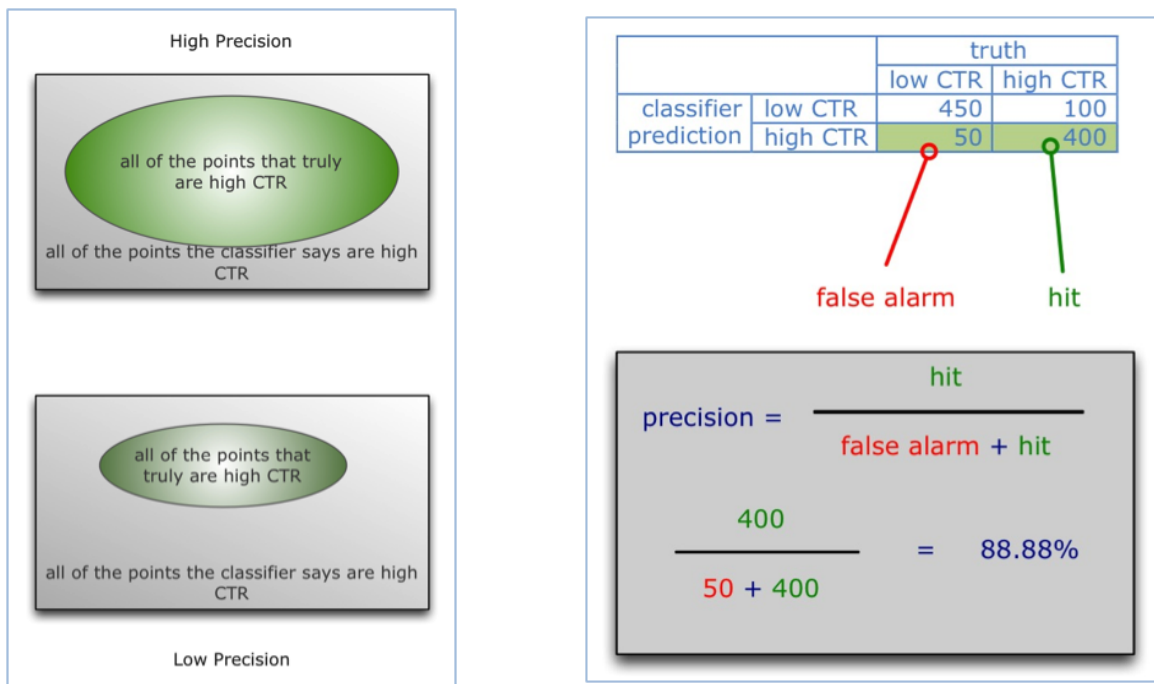

Figure 3.9:

High Precision

all of the points that truly
are high CTR

all of the points the classifier says are high
CTR

Low Precision

all of the points that
truly are high CTR

all of the points the classifier says are high
CTR

		truth	
		low CTR	high CTR
classifier	low CTR	450	100
prediction	high CTR	50	400

false alarm   hit

$$precision = \frac{hit}{false\ alarm + hit}$$

$$\frac{400}{50 + 400} = 88.88\%$$

Mean Precision

The effectiveness of the classifier is the distance between the two means, which does not vary as threshold changes. One way of measuring the effectiveness of the classifier is the "precision". Precision is the number of truly correct items ("hits") divided by the number of items that the classifier says are correct (hits + false alarms).

In our example above, we have 1,000 data points, of which 500 are high CTR. Our classifier correctly catches 400 high CTRs. The classifier indicates an additional 50 points are high CTRs, when in fact they are low CTRs. This means we have 50 false alarms. These results in a precision of $400/450 = 88.88\%$ items that the classifier says are correct (hits + false alarms).

"Mean precision" takes into account the issues with choosing a threshold, noted above, by performing this calculation at a range of thresholds and taking the mean.

Improvement above chance

Unfortunately, in reality it is not as easy as this: projects vary greatly in the number of high CTR keywords. Some projects truly contain only 15% high CTR, while others are as high as 80% or 90%. The finance sector, as an example, will always have a lower average CTR than a popular Facebook game. We need more than mean precision to measure classifier performance.

Imagine our classifier has a mean precision of 95%. Although this sounds good, if the project is 95% high CTR, it's less impressive. In this case, if we always guessed that something was high CTR, we would do as well as the classifier! Thus, we must report the mean precision after subtracting out the true percentage of high CTR items for that project. We call this "mean precision above chance". In our reports, a mean precision of 10% means that the classifier has a 10-point higher precision than guessing alone.

This section and the previous contain an explanation about how a random forest works and how to test a classifier that is gently taken from an article on citizennet.com[10]

3.4 Random Search for Hyper-Parameter Optimization

This section includes the conclusions of the illuminating paper "Random Search for Hyper-Parameter Optimization", that inspired us toward the decision of using a Random Search in our classifier.[15]

Grid search experiments are common in the literature of empirical machine learning, where they are used to optimize the hyper-parameters of learning algorithms. It is also common to perform multi-stage, multi-resolution grid experiments that are more or less automated, because a grid experiment with a fine-enough resolution for optimization would be prohibitively expensive. We have shown that
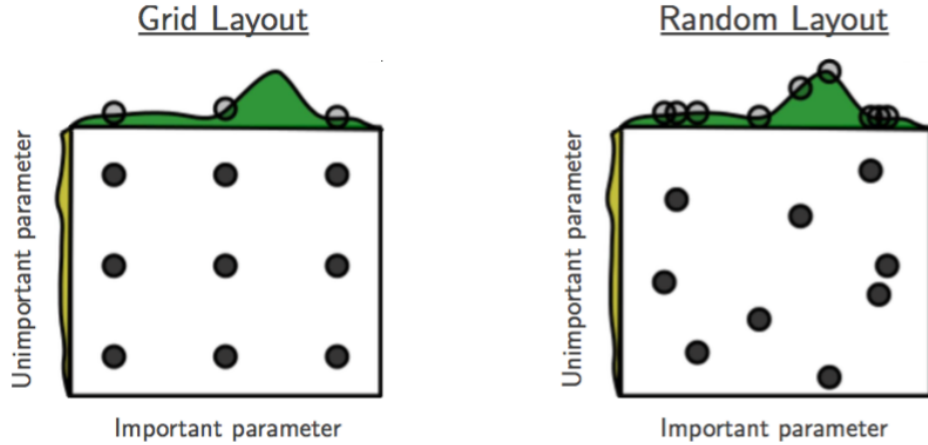
Figure 3.10: Differences between Grid Layout and Random Layout

random experiments are more efficient than grid experiments for hyper-parameter optimization in the case of several learning algorithms on several data sets. Our analysis of the hyper-parameter response surface (Ψ) suggests that random experiments are more efficient because not all hyper- parameters are equally important to tune. Grid search experiments allocate too many trials to the exploration of dimensions that do not matter and suffer from poor coverage in dimensions that are important. Compared with the grid search experiments of Larochelle et al. (2007), random search found better models in most cases and required less computational time. Random experiments are also easier to carry out than grid experiments for practical reasons related to the statistical independence of every trial.

- The experiment can be stopped any time and the trials form a complete experiment.

- If extra computers become available, new trials can be added to an experiment without having to adjust the grid and commit to a much larger experiment.

- Every trial can be carried out asynchronously.

- If the computer carrying out a trial fails for any reason, its trial can be either abandoned or restarted without jeopardizing the experiment.

Random search is not incompatible with a controlled experiment. To investigate the effect of one hyper-parameter of interest X, we recommend random search (instead of grid search) for optimizing over other hyper-parameters. Choose one set of random values for these remaining hyper-parameters and use that same set for each value of X. Random experiments with large numbers of trials also bring attention to the question of how to measure test error of an experiment when many trials have some claim to being best. When using a relatively small validation set, the uncertainty involved in selecting the best model by cross- validation can be larger than the uncertainty in measuring the test set performance of any one model. It is important to take both of these sources of uncertainty into account when reporting the uncertainty around the best model found by a model search algorithm. This technique is useful to all experiments (including both random and grid) in which multiple models achieve approximately the best validation set performance. Low-discrepancy sequences developed for QMC integration are also good alternatives to grid- based experiments. In low dimensions (e.g., 1-5) our simulated results suggest that they can hold some advantage over pseudo-random experiments in terms of search efficiency. However, the trials of a low-discrepancy experiment are not i.i.d. which makes it inappropriate to analyze performance with the random efficiency curve. It is also more difficult in practice to conduct a quasi-random experiment because like a grid experiment, the omission of a single point can be more severe. Finally, when there are many hyper-parameter dimensions relative

to the computational budget for the experiment, a low-discrepancy trial set is not expected to behave very differently from a pseudo- random one.

Finally, the hyper-parameter optimization strategies considered here are non-adaptive: they do not vary the course of the experiment by considering any results that are already available. Random search was not generally as good as the sequential combination of manual and grid search from an expert (Larochelle et al., 2007) in the case of the 32-dimensional search problem of DBN optimization, because the efficiency of sequential optimization overcame the inefficiency of the grid search employed at each step of the procedure. Future work should consider sequential, adaptive search/optimization algorithms in settings where many hyper-parameters of an expensive function must be optimized jointly and the effective dimensionality is high. We hope that future work in that direction will consider random search of the form studied here as a baseline for performance, rather than grid search.

4   Problem Statement

Atoka Index stores information about Italian companies, including contacts like emails, official website and official Facebook page. Only the 1.62% of the companies had a linked Facebook Page or user before the project. The aim is to fill more companies with Facebook social data. This topic was chosen because in today's world it has become more and more important to have social information about companies if you want to find potential customers through a lead generation technique.

4.1   Data Sources

The data sources used in the project are mainly two: Atoka Index and the results give by Facebook's GraphAPI. Data is analyzed and collected in a single index, used by the classifier to predict if a Facebook page belongs or not to a particular company.

4.1.1   Atoka

Atoka is a collection of data coming from Cerved Group S.P.A and crawlers made at SpazioDati, stored in a Elasticsearch index. This elasticsearch index is the heart of atoka.io, the lead generation tool in which the results of this project will be shown, added to data previously available.

The most important fields of Atoka index used in the project are:

- ateco: ateco code of the company, representing the classification of economical activities

- description: a description of the company, used as a source for the research of keywords

- emails: list of email of the company

- entities: entities extracted with Dandelion API

- location: list of addresses of the company (companies may have more than a single office)

- social: social media link of the company (facebook, twitter, linkedin, etc.)

- website: official website of the company

It's important to underline the fact that not all the companies have a social and a website stored in Atoka index: the purpose, in fact, is to fill the missing values in social field.

4.1.2   GraphAPI

The Graph API is the primary way to get data in and out of Facebook's social graph[8]. It's a low-level HTTP-based API that you can use to query data, post new stories, upload photos and a variety of other tasks that an app might need to do.

The endpoint used for retrieving information is:

```
GET graph.facebook.com
  /search?
    q={your-query}&
    [type={object-type}](#searchtypes)
```

After being cleaned by the system developed, the company name is put in the parameter q as q={your-query}, for each company pages and user are searched inside Facebook graph (because many companies still use user instead of pages). For making less number of GraphAPI calls and reducing the HTTP request time we used a batch request, collecting multiple calls in only one. The limit imposed by Facebook is a maximum of 50 call per batch request. In this project, every batch request was made with 50 request.

The Batch endpoint is:

```
curl \
    -F 'access_token={TOKEN}' \
    -F 'batch=[{ "method":"POST", \
               "relative_url":"search", \
               "body": {query_1} }, \
             { "method":"POST", \
               "relative_url":"search", \
               "body": {query_2}]' \
    https://graph.facebook.com
```

4.2   Problem Definition

Achieving the required functionality seems to be a three-step process.

First is to take companies' names from Atoka index, clean them, query facebook for every set of name of the company and create a new elasticsearch index with the Facebook responses. Facebook responses are remodeled and, one time they are stored in the index, they should contain the same information types of Atoka in order to compare them and predict which page should be the original one.

After having stored Facebook's information, the second step is to query the new index and find which local Facebook page may match with the single company. This step in very important because it shifts the problem from a "one-vs-all" classification to a "one-vs-few" classification. The query selects a small subset that may match with the single company and makes possible to compare the company with only 20-30 pages/users instead of all the dataset. Selected the company's pages/users, a metrics script has to define some feature of the pages, giving a list of n-uples representing how similar the page/user data is to the company one.

The last step is to train a classifier with the features computed in the previous step and then classify all the pages. At the end of the classification each company with a Facebook page should have the page linked by the classifier, with a percentage of how similar it is to that company's data. The classifier algorithm chosen is random forest. The choice was made for time reasons, for avoiding overfitting problems and because random forest is easier to be configured in an affordable way and amount of time.

One big problem is that some company names are not equal or contained in their Facebook page because the society name is only used for legal purpose and the real name is actually different. In that case it's impossible to find the page with Facebook's GraphAPI and the result cannot be contained in the resulting set of the classifier.

5   Solution

The following paragraph contains a description of how the solution to the problem was developed and why certain choices were made.
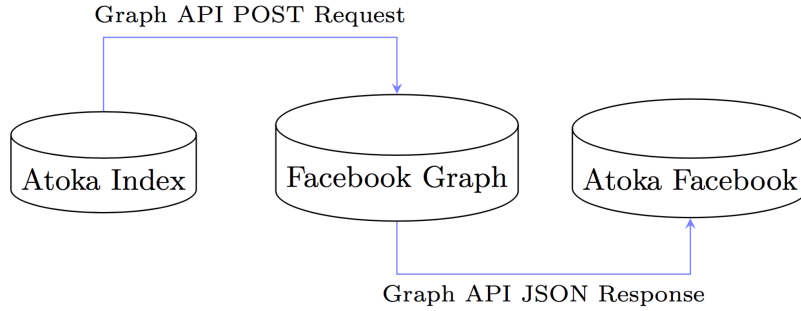
Figure 5.1: Process of data acquisition from Facebook GraphAPI.

5.1 Getting Possible Facebook Pages

The first module of the project has the task to get company names contained in the Atoka index and query Facebook Graph via Facebook GraphAPI in order to get a list of possible compatible pages.

For each company contained in Atoka index, the module generate a list of different possible alternatives to the legal name. This approach extends the time spent making multiple queries but makes sure to have the most number of possible Facebook Pages/Users per company.

The first correction is to add the cleaned legal name. The module cleans the name from company's legal form so as to make a reasonable request to GraphAPI that returns a bunch of pages that match with the given query.

An example of a cleaned name is:

"*Spaziodati S.R.L*" ⇒ "*Spaziodati*"

A second cleaned name to add is the one with patterns associated to company legal forms removed:

"*Bar Roma di Rossi Carlo e figli*" ⇒ "*Bar Roma*"

Other names added are the ones with all possible legal forms denomination alternatives. Atoka index in fact contains companies' name with their own legal form sign. The project used a dedicated function of Dandelion in order to generate all the options.

An example of one a new alternative is:

"*ENI S.P.A*" ⇒ "*ENI Societa per Azioni*"

or can simply be:

"*Enel S.P.A*" ⇒ "*Enel SPA*"

Another step is to change the legal name with the many possible alternatives names contained in Atoka index.

A plausible example is:

"*CervedGroup S.P.A*" ⇒ "*CG SPA*"

Even combinations of the alternatives are made:

"*CervedGroup S.P.A*" ⇒ "*CG Societa per Azioni*"

An important thing made achievable by Atoka data is to add to the list signs' names. Signs' names are names contained in the signs outside the companies and in many cases represent the real name of the Facebook Page/User. This fact happen because it's not unusual to have different local stores with different names that belong to a single holding company that will have nothing to do with the single marketing strategies and social activities.

That cases are usual initials, in the form of:

"*ABC S.N.C*" ⇒ "*Ristorante da Luigi*"

As briefly explained before, every name is added to a list and every company will have a query associated at each name in that list. This makes sure that every possible page name is searched inside Facebook Graph and stored in the new elasticsearch index. We found that is far better to call 10 queries for each company and find a reasonable amout of pages that may match rather than populate the index in less time but having less possibility to find a matching page.

Type	Attributes
Page	id, name, link, about, email, location, likes members, description, category, contact_address, general_info, general_manager, mission, phone, website, cover
User	id, name, link, about, email, bio, work, location, description, address, hometown, website, cover

Figure 5.2: List of attributes exposed by Facebook for Pages and Users through Graph API.

For using GraphAPI, Facebook requires a Developer account, verified with an individual mobile telephone number. Each account can have different applications running at the same time. The applications are identified with a text token and each of them can make at most 100 million API calls per day[7]. Even if the documentation shows that limit, experimentation found that the real limit is around 600 calls per 600 seconds, per token & per IP[14]. This bottleneck is overtaken creating 5 different accounts with 5 applications each, with a total of 25 applications. The module is structured in multiprocessing with one different process for each token, running simultaneously on a private server. A second trick used not to be banned from Facebook is making requests with a customized version of facepy[9] library that changes proxy continuously and simulates request all around the world. Every request is made after a random delay timeout, chosen to simulate real world's typical application calls.

A batch request makes possible to query the API 50 times at one, optimizing the latency created by http's connection opening and closing. Each query is composed by two different groups of graph Api subqueries, one group for user profiles and one group for Facebook pages. All the two groups of requests put together the names generated in the previous step. Each request return at most 30 entities, the limit is chosen not to have a too large amount of useless data: most of the time the right company page/user comes in the first 20-30 results.

5.2 Cleaning and Normalizing Data

Before saving all the pages and user results in the new elasticsearch index, the data is cleaned and normalized. This step is obligatory because pages and users have different fields and because very frequently data exists but can't be found in the right place. This happens because people seldom set up their company page/user profile without taking care of adding the right thing in the right place. For example, in a lot of different pages, the field "phone" or "email" is empty but you can find one or both of them in the "about" field, that was originally thought to contain a brief description of the page and what it is about.

To make sure to have all the data cleaned and normalized the module combines the fields in a single structure. After having collected everything in one place, the module checks with a proper regex if possible domains and emails are contained in the structure and annotates the structure with ontologies extracted with Dandelion. Dandelion extract persons, telephone numbers, places, addresses and entities (like "data mining", "agriculture", "automobiles"). Telephone numbers and addresses are automatically normalized by Dandelion and can be directly checked with the same attributes contained in the Atoka index.

The only field that is not included in the structure is the location information, that is analyzed on its own to avoid problem of ambiguity finding, for example, people inside the structure.

Ex: "[..] $Via\ Marco\ Polo,\ 13\ [...]\ \Rightarrow \{\}$

After having cleaned and normalized the information, data is stored in the new elasticsearch index. The primary key chosen to identify the tuples is the real Facebook ID: this choice becomes handy because it make sure that one page/user is store only once and that the index is redundancy free and consistent.

5.3 Generating Similarity Scores

After the data is normalized and stored we need to generate the similarity score between the company and the possible Facebook pages. This is a list of how the scores are gives for each property:

- **WEBSITE** Check if the sites are present: 0.5 pts if not. Matching of the normalized sites: 1pt if positive, 0 if negative. Sites normalization works removing from the string all the unnecessary parts, such as "hhtp://", "https://", "www.", we then match the two strings fully

- **EMAIL** The results are in form of tuple containing the resulting scores. First we check the presence of the email addresses: (0.5, 0.5) pts if not present. Next we match the various components of the input email with the company registry. The email is split in it's three main components: username, domain and ext, and for the companies we use data such as complete name, city and province code. For every input we produce the rankings with the company registry, using username first and domain in another run. Each run we clean up the input data, we fetch the province's full name and create a token list that are in common with the email and the processed data. Next pass is to check if every input token exceeds a similarity threshold, set at 75% of the total length. Then the suitable ones are divided into two categories, common matches and uncommon matches. The token total score is calculated, adding up all the 'common matches'. Same for the 'uncommon matches'. Finally we produce the final score, choosing the highest couple

$$\left(\frac{scoreCommon}{length(username)}, \frac{scoreUncommon}{length(username)} \right)$$

  The same thing is done for domain:

$$\left(\frac{scoreCommon}{length(domain)}, \frac{scoreUncommon}{length(domain)} \right)$$

- **PHONE** Check if the phone numbers are present: 0.5 pts if not. Matching of the inputs filtered from the empty ones: 1 pt if positive, 0 if negative.

- **NAME** Check for the presence of a description or a name in the facebook input: 0.5 pt if not present. We firstly checked in the description if there was at least one of the alternative names of the company: 1 pt if at least a match is present, 0 otherwise. We decided to optimize the compare in order to avoid to give a too high rate for too usual name. In fact it happens frequently to have situation where different companies have a very similar name:

  $Pizza\ Vesuvio\ di\ Nino\ Coppola \simeq Pizza\ Vesuvio$

  the formula chosen to overtake this problem is:

  $$\frac{\#tokenAtoka \cap \#tokenFacebook}{\#tokenAtoka \cup \#tokenFacebook}$$

- **ADDRESS** For this section we calculate multiple scores for each part that the Facebook input address is consisting of.

  - **STREET** Check for the street presence: 0.5 pts if not. Matching using the subdivision of the input streets in tokens, and then analyzing their set intersection using the formula:
    $$\frac{\#tokenAtoka \cap \#tokenFacebook}{\#tokenAtoka \cup \#tokenFacebook}$$

  - **HOUSE NUMBER** Exact match between the input numbers: 1 pt if positive, 0 pts if negative or The function can also calculate a score determined by the number distance of the 2 inputs, using the formula:
    $$1.5^{-|house\_number\_1 - house\_number\_2|}$$

  - **CITY** Check for the city presence: 0.5 pts if not Exact match between the two cities: 1 pt if positive, 0 if negative.

  - **PROVINCE** Check the presence of the province: 0.5 pts if not Exact match between the provinces 1 pt if positive, 0 otherwise

  - **ZIP CODE** Check the presence of the zip code: 0.5 pts if not. Exact match between the two zip codes: 1 pt if positive, 0 otherwise

  - **TOTAL** Results are in form of tuples having 5 scores each. Firstly we control the presence of all the above scores, and put a 0.5 if they are absent (or 0 in case of the house number) Each tuple is of the form:

($ScoreStreet$, $ScoreHouseNumber$, $ScoreCity$,
$ScoreProvince$, $ScoreZipCode$)

If no score is present, the resulting tuple is:

(0.5, 0, 0.5, 0.5, 0.5)

Otherwise, we choose the best scoring tuple, from all the comparison of all the Facebook addresses with the Atoka input.

- ENTITIES Score given by the intersection of the input data entities using the following formula:

$$\frac{EntitiesAtoka \cap EntitiesFacebook}{EntitiesAtoka}$$

5.4 Training Random Forest Classifier

After the process of score assignment a subset of results is used to to train the classifier. The algorithm chosen for the machine learning classification is random forest.

Random Forest is generally faster than the other types of classification algorithms, and this is one of the reasons because it's the best algorithm found for this problem[16]. In a slightly different problem, the classification of websites instead of Facebook pages/users, SVM was implemented but the running time was tens of times longer than Random Forest and the results were even worse. This shows the second reason of why Random Forest seems to be better than SVM or other approaches: it's easier and faster to set, to regulate and to train. Other important facts that showed Random Forest Classification Algorithm as the best for this job are that it's very good avoiding overfitting, that it's solid with outliers and that it works fine with partial and dirty data.

The comparable Website Problem

When the project started the first thing that has been taken in consideration was a similar project previously developed at Spaziodati. The aim of that project was to match website crawled from Italian net with the same company index that was used for this project.

The first step in order to get the official website classification was building a crawler for Italian websites network that could populate an index. That bot crawled all the websites looking for the same informations found in the company index (descriptions, emails, entities, location, social media and obviously the website URL).

After having created and populated the index, information were cleaned and normalized with Dandelion, in a really similar way of what was done in the Facebook project. Once the data was made comparable, as done here, a proprietary algorithm generated a similarity score for each of the websites' fields.

One time the scores were given, the team that was working on the project faced our same problem of choosing the best algorithm for the classification. Unfortunately, instead of having a similar project as we had, they had nothing to compare the project with and to choose the algorithm on the high level so they tried different solutions and took the best one.

Many algorithms were tried but the main two were:

- Random Forest

- Support Vector Machine (SVM).

After the tests, the team decided to choose Random Forest mainly for performance of precision and recall over that dataset (obviously this information can't be generalized all over the data available in the world).

In that case Random Forest was the more convenient because it returned values of features that were really useful in order to evaluate the data gathered and to decide how deep the crawler needed to look in the process of data extraction fro the web. A second important fact that drove the team in the algorithm decision was that the training time was far less than any other method, a fundamental consideration taking care of the fact that the index need to be continuously updated. Furthermore, Random Forest gives the developer the possibility to work with categorical features without too many manipulations. Last but not least, Random Forest in the best algorithm that an analyst can choose in order to avoid the overfitting problem.

As revealed before, when we had to choose the best algorithm to classify Facebook pages data, instead of taking care of testing the different algorithms - a process that really makes a company waste a lot of time and money - we decided to compare the data available in the Facebook index with the data available in the website's. The result was that the informations were really similar - as said before the two indexes were cleaned and normalized with really alike methods - so we directly decided to use Random Forest for our classification. Before taking the final decision we also decided to try SVM too because we wanted to be 100% sure about the fact that it was the best also in our case but in less that a week we faced that, apart from the fact that was really harder to configure it right, it was really too slow for our needs and we firmly opted for Random Forest.

Random Forest in the Facebook Problem

The first step when somebody is working with a machine learning algorithm is finding a useful training subset of the data. The main training subset is composed by 60% of the data, splitted randomly from the main dataset. Using the Random Forest Algorithm it is not necessary to define a custom cross validation system because it already have its own method to estimate the out-of-bag error. Even if the algorithm comes with this "simplification" we chose not to use it and to implement a cross validation. This choice was driven by the fact that the dataset is really skewed, with only few positive examples and few association between companies contained in Atoka index and theirs Facebook pages. The approach used is stratified three fold cross validation: the training set is sampled in the three fold, making sure to maintain the same positive/negative percentage composition, and for three time the random forest is trained with one fold and tested on the other two. At the end, the trained classifier is tested with the remaining 40% of the data.

In the training step, instead of giving parameters as possible values, the approach used was to give them as distribution with the Randomize Grid Search method. Randomize Grid Search randomly samples the distribution, then tests the different parameters and takes the better ones. The method uses 100 samples for the training dataset. This method works fine with this problem because it gives decent results if the distribution is uniform (as it is) and works in a reasonable amount of time.

The results with the first training set were not very good. This problem was caused by the fact that ATECO n.56's companies only had 6,000 pages/users classified and because many of that pages were false positive and many other pages that were missing in the initial index were found right in the training set. To avoid this huge problem we opened a story on CrowdFlower[4], asking people to verify if the pages associated by the classifier were right, wrong. This process help us redesigning the training set without false positive and true negative. This may seems a conscious modification of the training dataset to have "better final result" but, in reality, it is not overfitting because it only fix error that should not be in the first dataset. This solution was the only adoptable because the only platform that surely knows if a page is really about a company is Facebook itself.

6   Implementation

6.1   GraphAPI

The GraphAPI can be queried with a single POST request. Here's the code to compute the batch request:

```
[{'method': 'GET',
    'relative_url': 'search?q={0}
    &type=page&locale=it_IT&limit=50
    &fields=id,name,link,about,email,
    location,description,category,
    contact_address,general_info,
    general_manager,mission,phone,
    website,members,cover,likes'
```

```
        .format(
            quote(unicode.encode(company_name,
            encoding='utf-8')))
} for company_name in company_list
] + [
{'method': 'GET',
    'relative_url':'search?q={0}
    &type=user&locale=it_IT&limit=50
    &fields=id,name,link,about,email,
    bio,work,location,description,
    address,hometown,website,cover'
    .format(
        quote(unicode.encode(company_name,
        encoding='utf-8')))
} for company_name in company_list
]
```

6.2   Dandelion API

Dandelion API is a Semantic Text Analytics as a service product developed by Spaziodati. Dandelion is a product that implement context intelligence: it extracts meaning from an unstructured text and put it in context, obtaining actionable data. Instead of classic NLP technologies, Dandelion API leverages its underlying Knowledge Graph, without relying on traditional NLP pipelines. This makes it faster, more scalable, easier to customize and natively language independent. Dandelion API works well even on short and malformed texts in English, French, German, Italian and Portuguese. The main APIs available on Dandelion are:

- ENTITY EXTRACTION: find mentions of places, persons, brands and events in documents and social media. Easily get additional data about the entities.

- TEXT & CONTENT CLASSIFICATION: classify multilingual text into standard, pre-defined taxonomies or build your own custom classification scheme in minutes.

- SENTIMENT ANALYSIS: identify whether the expressed opinion in short texts (like product reviews) is positive, negative, or neutral.

- KEYWORDS / CONCEPTS EXTRACTION: automatically identify important, contextually relevant, concepts and key-phrases in articles and social media posts.

- SEMANTIC SIMILARITY: compare two texts and compute their syntactic and semantic similarity. Understand when two texts are about the same subject.

- ARTICLE EXTRACTION: extract clean text article from newspapers, blogs and other websites. Remove boilerplate and advertising and get the article full text and images.

6.3   Atoka

Atoka is a lead generation business-to-business tool that helps companies to find new customers, receive news and updates from them and to generate prospect lists with few clicks. Thanks to the partership with Cerved Group, an Italian rating agency, and the usage of semantic analysis algorithms, Atoka offers affordable information about more than 6 millions Italian companies and economic subjects. Atoka provide addresses, contacts, websites, social networks links, management team composition, corporate structure, economic and financial data. Atoka also enables new search options that overtake tradictional filters (like ATECO code), enabling the customer to make more precise and articulated queries.
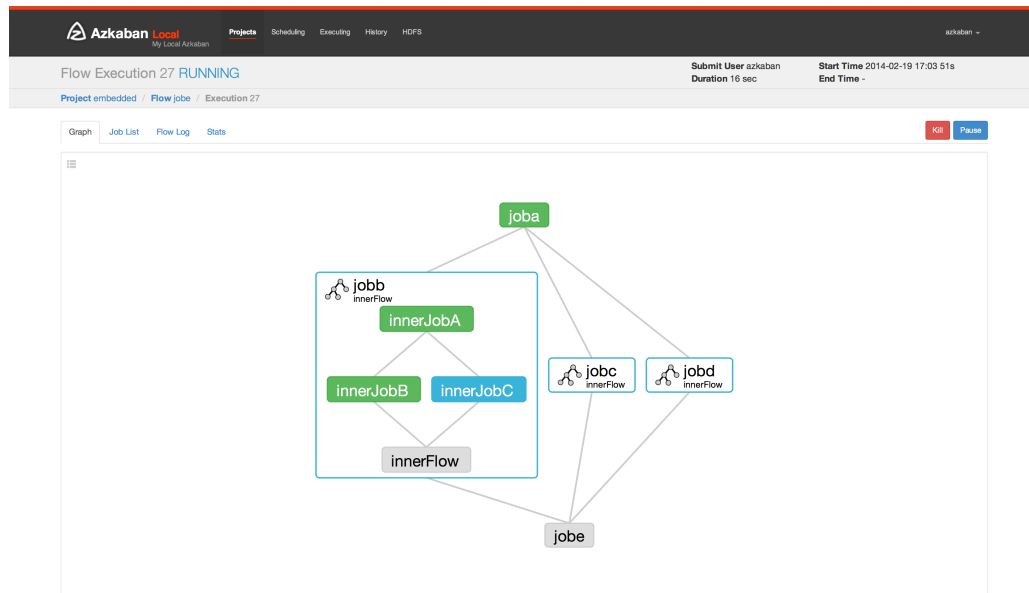
Figure 6.1: Interface of Azkaban

6.4 Scikit-learn

Scikit-learn (formerly scikits.learn) is an open source machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

6.5 Azkaban Flow

The automation of the entire project was implemented with Azkaban.

Azkaban is a batch workflow job scheduler created at LinkedIn to run Hadoop jobs. Azkaban resolves the ordering through job dependencies and provides an easy to use web user interface to maintain and track your workflows.[2] It presents many features:

- Compatible with any version of Hadoop
- Easy to use web UI
- Simple web and http workflow uploads
- Project workspaces
- Scheduling of workflows
- Modular and pluginable
- Authentication and Authorization
- Tracking of user actions
- Email alerts on failure and successes
- SLA alerting and auto killing
- Retrying of failed jobs

Azkaban was designed primarily with usability in mind. It has been running at LinkedIn for several years, and drives many of their Hadoop and data warehouse processes.
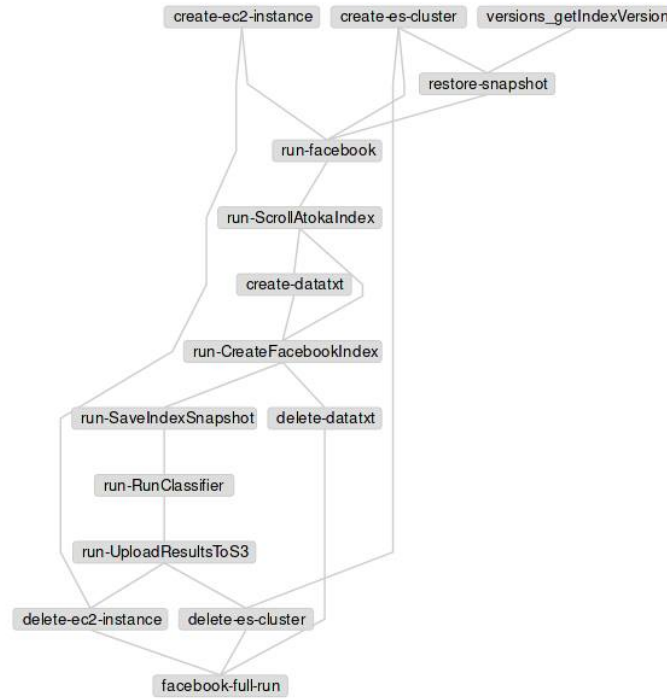
Figure 6.2: Spaziodati's Facebook Azkaban Flow

At Spaziodati Azkaban is scheduled once a month and the results are loaded on an Amazon S3 machine. The flow starts reading the Atoka Index and getting the anagraphic data used for computing the features and creating a new elastic search index in order not to modify or kill the production one (that keeps Atoka online). For finding information and normalizing data the flow was developed also to create a new instance of DataTXT, as before, not to touch in any ways the production one.

7 Experiment

This final chapter is meant to give some explanation of why and how some technical decision were taken.

7.1 GraphAPI Requests, Names, DNS

GraphAPI are the proprietary Facebook REST API that developers need to use if they want to access Facebook's graph.

When we started to use the APIs we immediately found some problems:

- Facebook isn't really clear about how many requests a single developer user/app can ask per second, minute, hour or day.

- Facebook isn't clear about how many request from the same IP can be asked in a single application.

- Facebook isn't clear at all about how many applications can run with similar requests at the same time

- Facebook is really foggy in the explanation of what you can ask through the query

Our Solution

As explained in the solution section about GraphAPI, Facebook isn't clear about how many requests a single developer user/app can make. When we started to develop the GraphAPI part the script used to give error in random moment because Facebook stopped the request without a standard and controllable rule. Besides, Facebook doesn't mention at all how it process different application for each developer or the place of the requests tracked down from the IP address. In order to avoid every possible problem, as explained in the solution chapter, we decided to use with different users, with different application, with requests made with multiple IP thanks to a DNS. Even if we hadn't any information we solved the problem and we never got banned again (banning stands for a few minutes without the possibility to ask we requests).

7.2 Why blocking isn't useful

When we were analyzing the problem from a top-down perspective we tried to understand if we could use a blocking technique to classify the Facebook pages in a more affordable and faster way. A blocking technique is a technique where data is clustered before being classified so the classification can be between fewer options in the subset. In our problem it could be useful to cluster the data in blocks that could contain types of companies: banks with banks, grocery stores with grocery stores etc. With a cluster for each type of company of the Facebook pages, in the step of finding the possible pages to compare with the company selected, we could have fewer pages to assign scores and to compute with random forest and it could save time. The main problem of using a blocking technique is that you have to find a way to cluster the data and, in our dataset, it wasn't possible because:

- the "category" field of the Facebook page is really tricky and most of the pages use a wrong one, a too specific one or a too general one. With that kind of information it's impossibile to cluster in a reasonable way

- the kind of information in all the other fields aren't different between different possible clusters: banks and grocery stores have a telephone number, maybe a website but these information aren't useful for dividing pages in different subsets.

- we could try to cluster by analyzing the description field but the information we analyzed showed that companies don't use the filed in a correct mode and instead of putting a clear and specific description of what the company does, they put random information that aren't useful for a clusterization. Most of the times we couldn't extract a single entity from the description.

- even if we could find a way to cluster the data, most of the companies wouldn't fit in any cluster so we would have to find the possible pages non only in the specific cluster but even in the "no clusterizable" cluster too, without having a performance improvement.

8 Conclusions

This paper analyzes the problem of finding the official Facebook page (or user) for the Italian companies. The process shows:

- how to get data from Facebook

- how to clean and normalize the data

- how to get a subset of possible Facebook pages/users

- the criteria that can be used to compute the similarity scores between company data and Facebook data
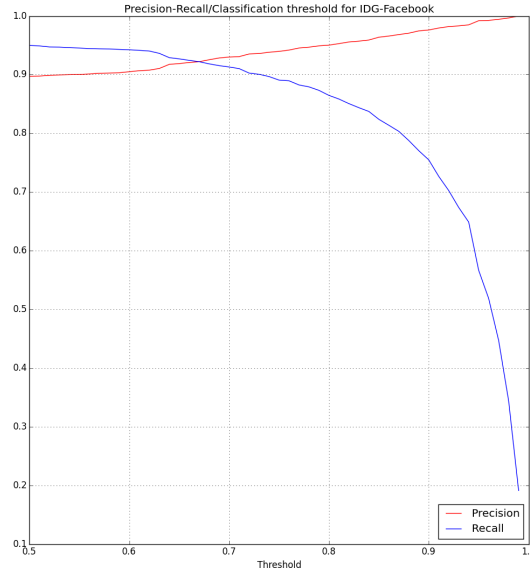
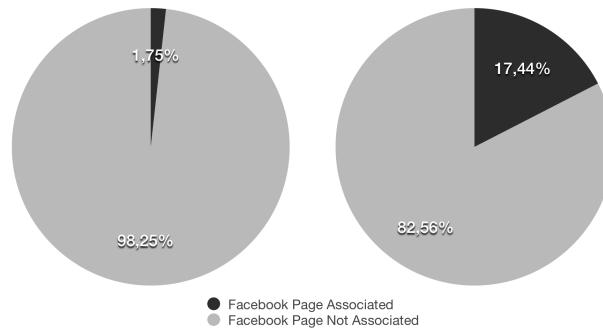Figure 8.1: Precision-Recall/Classification Threshold Plot

Figure 8.2: Percentage of Facebook Pages matched before and after the project

- how to use random forest to find the most probable Facebook page/user for the company

The final data presents a precision that start from 0.9 and grow to 1 while the recall, starting above at 0.95, after a threshold of 0.75 drastically fall down, getting to zero very fast.
The threshold chosen is 0.73, that assures a precision of 95% and a recall of 90%. This means that we give a wrong page one time over twenty and that we loose a good page one time over ten.

We chose a really high precision rather than a higher recall because in a business product like Atoka it is far more important not to give wrong information instead of losing a result that sometimes could be available.

We started with 362154 companies with ATECO n.56 collected in Atoka index. The Facebook pages connected to that companies, before the project was only 6355. The number of pages collected in the first Facebook index was 1,321,634.

After this work, with the threshold chosen at 0.73, the Facebook pages/users matched with Atoka companies are 63172. At a lower precision with a lot higher recall, founded with a threshold of 0.5, the number of matched pages amount at 79210.

This project help Atoka to classify Facebook pages/user for a total of 10 times more than before the project. The percentage of companies with ATECO n.56 that now are matched with a Facebook has grown to 17.44%.

Bibliography

[1] Atoka - lead generation. https://atoka.io/.

[2] Azkaban - official website. https://azkaban.github.io.

[3] Converting your profile into a facebook page. https://www.facebook.com/help/175644189234902.

[4] Crowdflower - official website. http://www.crowdflower.com/.

[5] Elasticsearch - wikipedia. https://en.wikipedia.org/wiki/Elasticsearch.

[6] Elasticsearch from the bottom up, part 1. https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up.

[7] Facebook api polices. https://developers.facebook.com/policy#thingstoknow.

[8] Facebook graph api. https://developers.facebook.com/docs/graph-api.

[9] Facepy project - github. https://github.com/jgorset/facepy.

[10] A gentle introduction to random forests, ensembles, and performance metrics in a commercial system. https://citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics/.

[11] Istat. http://www3.istat.it/strumenti/definizioni/ateco/ateco.html?versione=2007.3&codice=I-56.

[12] Spaziodati. http://spaziodati.eu/en/.

[13] The top 20 valuable facebook statistics - updated july 2016. https://zephoria.com/top-15-valuable-facebook-statistics/.

[14] What is the real facebook api limit? http://stackoverflow.com/questions/8713241/whats-the-facebooks-graph-api-call-limit.

[15] Yoshua Bengio James Bergstra. Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13 (2012) 281-305*, 2012.

[16] Senen Barro Manuel Fernandez-Delgado, Eva Cernadas. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research 15 (2014) 3133-3181*, 2014.