Professional Thesis

# Product data-quality improvement through attribute prediction

**Léonard Binet**
Telecom ParisTech
Post-Master’s Degree
BGD

-

Company supervisor: Pierre Arbelet
Academic supervisor: Slim Essid

January 01, 2018

## Abstract

This paper aims at explaining how we tackled data-quality challenges at Alkemics, with the use of some machine learning algorithms. But rather than solely describing models and algorithms, a strong focus is put on implementation and business added value.

Confronted with challenges concerning product data quality, Alkemics decided to explore new ways to provide value proposition to its clients.

# Contents

# Chapter 1

# Introduction

Alkemics is a start-up that was founded in 2011. Back then, they created e-merchandising tools to improve user experience on e-commerce websites. It then evolved to become a platform that enables the FMCG (Fast-Moving Consumer Goods) ecosystem to share all their data, from product to transactional information, in order to improve pricing, supply, marketing.

Alkemics develops products that enable collaboration between Brands and Retailers, and tackle the challenges of omni-channel commerce and smart customer interaction.

## 1.1 Alkemics

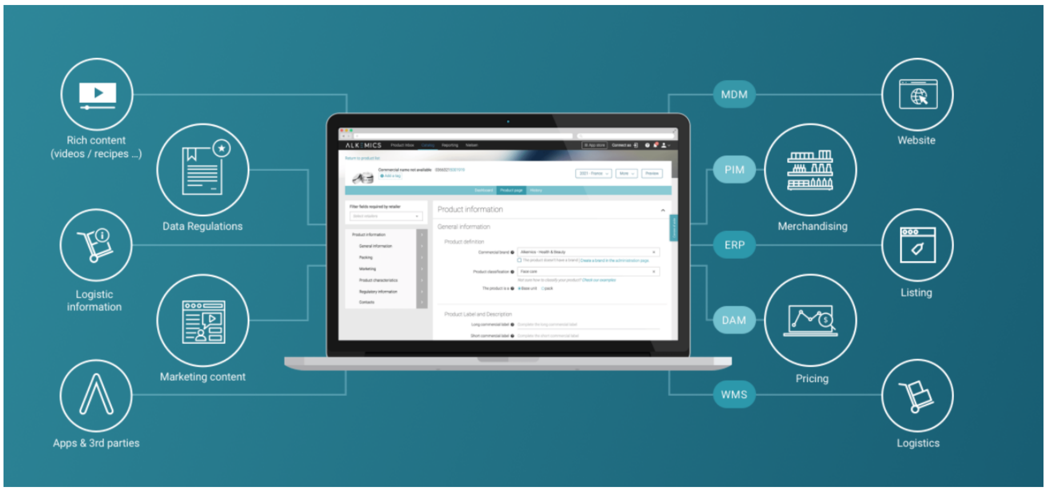


Figure 1.1: Alkemics

### 1.1.1 Product Data Empowerment

Because of increasing regulations, new customer demands, omni-channel communications, there is a growing need for richer and more exhaustive product information as well as an exponentially growing number of products. Some products change more than 10 times per year (promotion, seasonality, new recipes, new packagings...), and overall, 40% of products are renewed every year.

This cascades in a series of problems that handicap manufacturers, retailers and consumers: logistics struggle to maintain stock, accounting systems fail to track how different products are actually the same consumption unit, pricing teams struggle to maintain price coherency in this dynamic environment, marketing teams are inefficient to share richer and richer content about the product, BI teams feel that the consumption of the consumer are changing while they actually don't.

This problem is called product chaining. It describes the ability to connect products that share a given set of attributes in order to understand how they relate.

To overcome these problems, Alkemics built a product taxonomy reinforced by an ontology. PDE [1] team is fully dedicated to enhance the classification task to automatically classify the category of products, as well as the brand and other attributes such as the packaging or the price range. The classification task mainly uses Natural Language Processing techniques and distances in semantic spaces. PDE is also in charge of the management of the product data model.

### 1.1.2 Product Stream

Product Stream is a web application that enables:

**Makers** to:

- Collect product information from all the stakeholders of the organization: the *supply team* who knows the size and weight of the product, the *marketing team* who owns the pictures, videos as well as the brand content, the *quality team* who masters the composition, nutritional values, etc.
- Store product information in a centralized way, so everyone in the organization can have access to it (DAM [2]).
- Distribute product information so every partner has the same up-to-date information. This includes retailers, but also marketing agencies, trade marketing agencies, ... (EDI [3] / PIM [4])
- Collaborate with 3rd parties to collect user reviews, receive data quality reports, print coupons, use buy-it-now solutions.

**Retailers** to:

[1] Product Data Empowerment
[2] Digital Asset Management
[3] Electronic Data Interchange
[4] Product information management

- Have access to best-in-class product information to power their tools and feed their digital supports (EDI)
- Collaborate with manufacturers to reference new products & innovations, run call-for-proposals for promotional formats, etc. . .
- Connect products that share a given set of attributes in order to understand how they relate (Product Chaining).

### 1.1.3 The technological stack

**Micro-services architecture** The stack in Alkemics is composed of set of dozens of micro services communicating with synchronous calls (HTTPS) and both synchronous and asynchronous messaging (TCP enabled by Rabbit MQ). Each service has a domain specific task: data ingestion, data classification, APIs for merchandising, etc. A subset of the services are exposed to third parties and clients.

**User interface** User Interface dashboards also call the APIs to make functionalities available to the users (makers and retailers) trough a web browser. The front part of the website is coded with React framework.

**APIs and SDKs** A set of SDKs are also provided to retailers to allow them to use merchandising functionalities directly embedded in their websites.

**Storage vs indexation** All Alkemics crucial data is stored in relational databases (MySQL). Yet if relationnal databases provide very useful functionalities to guaranty integrity of data, its indexing capabilities remain quite poor in comparison to non-relational databases.

That’s we also create ElasticSearch indexes, containing data that is already stored in relational databases, but indexed in a format allowing quick and performant queries.

## 1.2 Professional thesis objective

This professional thesis aims at implementing machine learning techniques to improve products data quality.

# Chapter 2

# Data quality challenge

This chapter details one of the main challenges Alkemics faces, and how we tackled it. After presenting what is data-quality and why it matters, we will present how we built a suggestion workflow that helps our users to detect errors, and provides potential corrections. Finally we will detail what tools we built to monitor and control quality.

The detail of machine learning algorithms used to provide predictions will be detailed in next chapter.

## 2.1 Stakes

Reliable, up-to-date, accessible, tracable, products data is one of the core value proposition of Alkemics services for retailers.

To provide this service, Alkemics teams have put a lot of effort in building:

- bridges between data-models: Mondelez data-model is not the same as Pepsi, or Auchan's one. Alkemics data-model aims at generalizing all others and have the ability to translate any product data in other data-models.
- bridge between data-stores: APIs and interfaces to easily import or export data in several formats.
- bridges between people: ability to communicate easily about products through the platform

Ultimately, data is owned and provided by manufacturers, and then shared with retailers.

### 2.1.1 What is data-quality

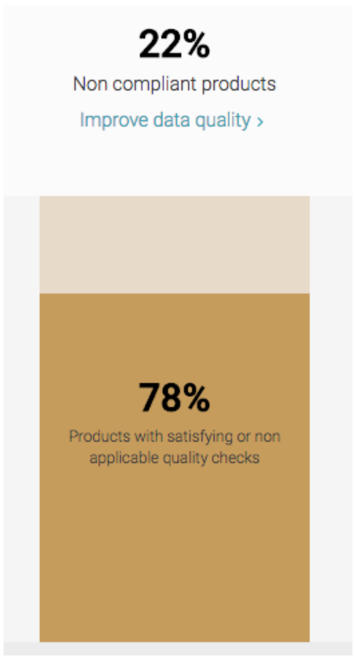


Figure 2.1: Example of data-quality measure.

Data quality can be evaluated through:

- completeness: the product page has sufficient details about the product.
  - ⋆ regulatory fields: for instance if your product contains alcohol you must provide the alcohol concentration.
  - ⋆ retailer specific fields: some retailer set their own requirements to sell on their platforms
- accuracy: the provided data is accurate, there is no error.

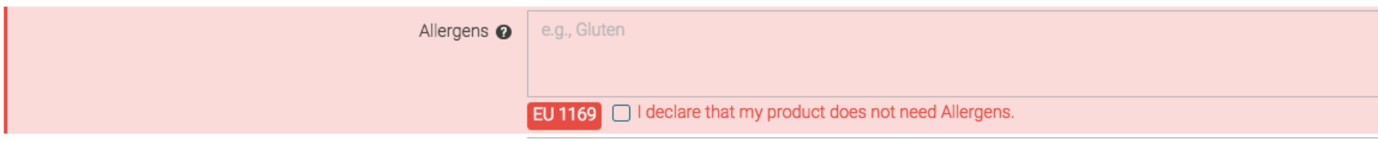


Figure 2.2: Example of a product that doesn't specify required fields.

### 2.1.2 Why it matters

**Data quality is a major issue to retailers**

- Regulatory: internet retailers have the obligation to provide some specific informations on the products. If they don't they can be heavily fined. Fields as

- Marketing: display detailed information about products to customers. Would you buy products with no description?
- Search: to correctly index products you need data, for instance how will your customer find strawberry icecream is flavors fields are not well filled. Well indexed products is key to performant search engine on web plateforms.
- Logistics: weight, size, number of units etc. These fields allow retailers to correctly handle logistics.

The growing part of online groceries. The rise of Amazon is a direct threat to all retailers. One of Amazon's advantages is its huge ability to handle data-flows.

### 2.1.3 Why is the data of poor quality

Some of the reasons are:

- many actors have to communicate: +50k manufacturers, dozens of retailers. Each manufacturer has to send its data to each retailer sending its products => hundreds of thousands of connections
- no fully accepted data-model standard. Even though GDSN standard is dominant, not all companies are compliant with it.
- actors interested in data-quality (primarly retailers), rely on actors for which it is less crucial (makers)
- products data identification numbers (EAN) frequently change

### 2.1.4 What can we do about data-quality

Given a product, we want to predict several of its attributes given some basic fields that are nearly always filled. The predicted fields for now are:

- hazard pictograms (inflammable, corrosive etc ...)
- labels (organic, made in France, eco-packaging etc...)
- category in Alkemics data-model (Processed meat, Cereal, baked product etc...)
- allergens (walnut, lactose, sulfite etc ...)

After these predictions are made, we want to evaluate it against products attributes to warn the manufacturers if we think that some of its data is incomplete or innacurate.

### 2.1.5 What are our constraints

- we cannot unilaterally change manufacturers data: they own their data
- we must garanty a high degree of confidence on suggestions we make, especially at the begining, otherwise manufacturers won't trust and use our suggestions
- predictions must be made regularly (product data continuously evolve)
- predictions must adapt to data-model changes
- our predictive models hade to perform well on datasets of poor data-quality (some fields are nearly never filled)
- we must be able to monitor closely how well our models perform, and how well perceived are our suggestions

## 2.2 Suggestion Workflow to improve data-quality

This part aims at presenting the workflow we use to provide attribute suggestions on product-pages to our users.

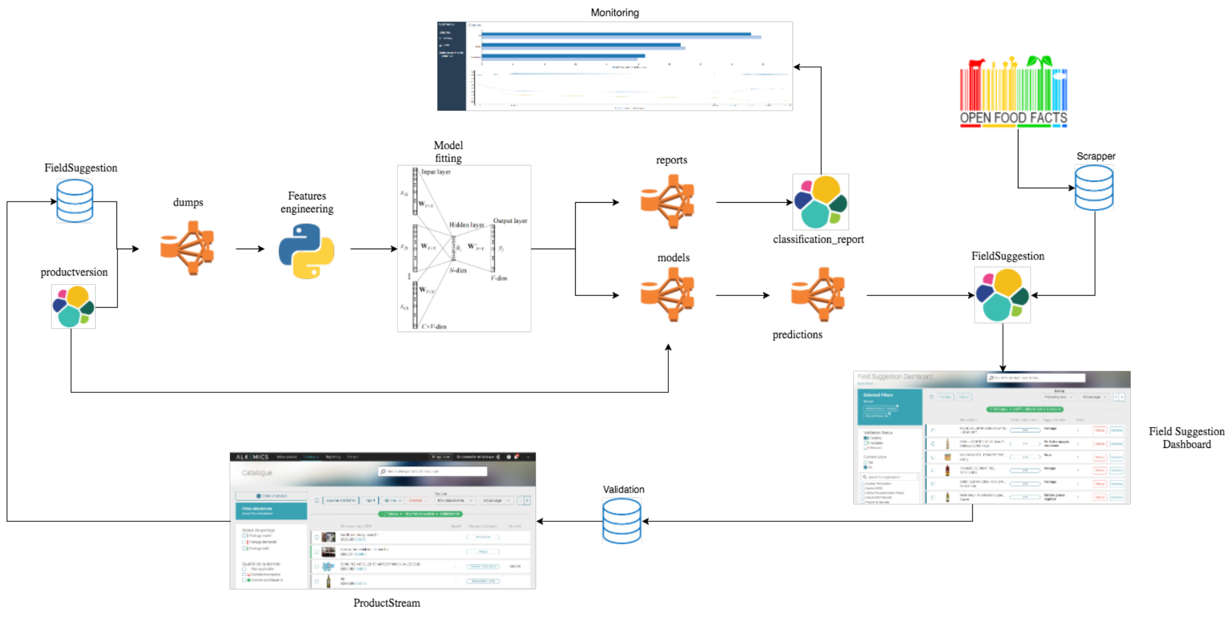


Figure 2.3: Suggestion workflow overview.

### Workflow steps

1. extraction of some attributes (name, description, composition) from products
2. feature engineering (NLP preprocessing)
3. training on labeled products (fasttext model) and report
4. prediction (on all products)
5. indexation as suggestion
    - from machine learning suggestions
    - from scrapped websites (major one: OpenFoodFacts)
6. suggestion validation by Alkemics
7. display of validated suggestions to users in product pages
8. suggestion acceptation by users

### 2.2.1 Feature and labels extraction

From product database. From validation database: suggestion validated by Alkemics, not yet accepted/refused by client. -> ability to bootstrap and improve models.

In our data-model, a product can consist of more than one hundred of fields, yet only some of them can be sufficient. The idea is to find a compromise between gathering enough information from products (-> more fields), but on fields with sufficient completion (-> less fields).

Let's take a cereal bar of brand Grany:

Figure 2.4: Product for which we will try to improve data-quality.

Here are the extracted attributes that will be processed to provide our features.

```
{
    "brand": "Grany",
    "namePublicLong": "Grany Coeur Fondant Gout Chocolat Noisettes
       120g",
    "nameLegal": "BARRE CEREALIERE FOURREE AU CHOCOLAT ET A LA
       NOISETTE.",
    "composition": "éIngrdients: ééCrales 31,5% (farine de RIZ 9,1%,
       farine de ÉBL 5,6%, grains de ïmas 4,7%, flocons de ÉBL 4,5%,
       flocons d'AVOINE 4,5%, flocons d'ORGE 2,5%, farine de ÉBL
       émalt 0,6%), sirop de glucose-fructose, sucre, huile de
       coprah, LAIT ééécrm en poudre, huile de palme, chocolat 5%
       (sucre, âpte de cacao), âpte de NOISETTES 5%, cacao maigre en
       poudre, humectant (églycrol), sel, gluten (de ÉBL), dextrose,
       émulsifiant (élcithine de tournesol), malt d'ORGE, ôarme
       (NOISETTES), ARACHIDE. PEUT CONTENIR SOJA, AUTRES FRUITS À
       COQUE.',
    "description": "Grany au œcur Chocolat au Lait et aux Noisettes,
       le plaisir brut des éécrales éassocies au fondant du chocolat
       au lait !",
    "advices": "",
    "healthAllegations": ""
}
```

### 2.2.2 Feature engineering

Transforming these features in a relevant input format for our models. The models we use will be detailed in next chapter. Theses models accept similar input as famous NLP models such skip-gram or c-bow.

This is classic NLP preprocessing.

1. concatenated fields as string
2. normalize text (encoding -> ascii)
3. tokenize
4. for each token:
   - lower
   - remove eventual html tags
   - filter stopwords and punctuation
   - remove digits
   - stem
5. concatenate tokens

Our previous object is now a list of preprocessed tokens that take the following form.

```
"grany cur chocolat lait noiset plais brut cereal associe fond
   chocolat lait grany barr cerealier fourre chocolat noiset grany
   coeur fond gout chocolat noiset ingredient cereal farin riz farin
   ble grain flocon ble flocon avoin flocon orge farin ble malt
   sirop glucos fructos sucr huil coprah lait ecrem poudr huil palm
   chocolat sucr pat cacao pat noiset cacao maigr poudr humect
   glycerol sel gluten ble dextros emulsifi lecithin tournesol malt
   orge arom noiset arachid conten soj autr fruit coqu"
```

### 2.2.3 Training stage

Training stage includes:

- labeled-set preprocessing
- split train/test:
- train on train, predict on test
- compute scores reports on both train-set and test-set

- compute global/local precision curves
- save model for further use
- index reports for monitoring

Multiclass labeled-set preprocessing: every sample has one and only one label - filter labeled items - remove classes with not enough occurences

Multilabel labeled-set preprocessing: samples can have 0->n labels - a sample can be considered even though it has no label - remove classes with not enough occurences from label lists

### 2.2.4 Predict stage

Predict stage includes:

- extracting all product features (label AND non-labeled)
- compute prediction scores
- enrich prediction with recall/precision estimations based on prediction score and local precision/recall curves computed during training-stage
- save all predictions with scores > 0.05

### 2.2.5 Machine learning suggestion indexation with scrapped information

Suggestions from machine learning, are merged with suggestions issued from web-scrapping. Those suggestions are indexed together to provide a global score (number of concordant sources) to suggestions.

Annexe:

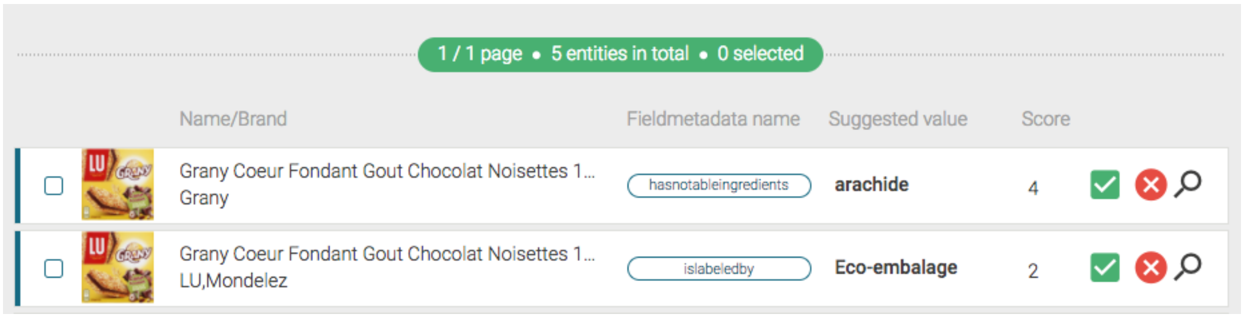
```
{
    "_type": "fieldsuggestion",
    "extended_attributes": {
        "precision_group": 0.91489,
        "features": "grany cur chocolat lait noiset plais brut
           cereal...",
        "probability": 0.87393,
        "recall_group": 0.55128
    },
    "entity_type": "carrefour_drive",
    "field": {
        "status": 0,
        "fieldmetadata_name": "hasnotableingredients",
        "fieldmetadata_id": 107,
        "is_current_value": 0,
        "field_id": 59,
        "field_name": "arachide"
    },
    "id": "1198365_107_59",
    "metadata": {
        "global_score": 4,
        "product_id": 142292,
        "media": "https://smedia.alkemics.com/product/142292/...",
        "brand": "Grany",
        "contentowner": {
            "id": 356,
            "name": "Mondelez International"
        },
        "sources": [
            "carrefour_drive",
            "intermarche",
            "openfoodfacts",
            "ml"
        ],
        "gtin": "07622300788063",
        "productversion_id": 1198365,
        "name": "Grany Coeur Fondant Gout Chocolat Noisettes 120g"
    }
}
```

### 2.2.6 Validation (or refusal) from Alkemics

Since we must reach a high degree of confidence in our predictions, and since we do not yet have history about how well we perform, each suggestion is manually validated.

For all predictions that are not values already stored in products' data (if the prediction match an already existing product data, everything's fine and we don't need to change its value), we will validate or refuse the suggestion.

A part of this task is semi-automated: for instance, suggestions issued from both scrapping and machine learning, with an estimated precision of 90% are batch validated.



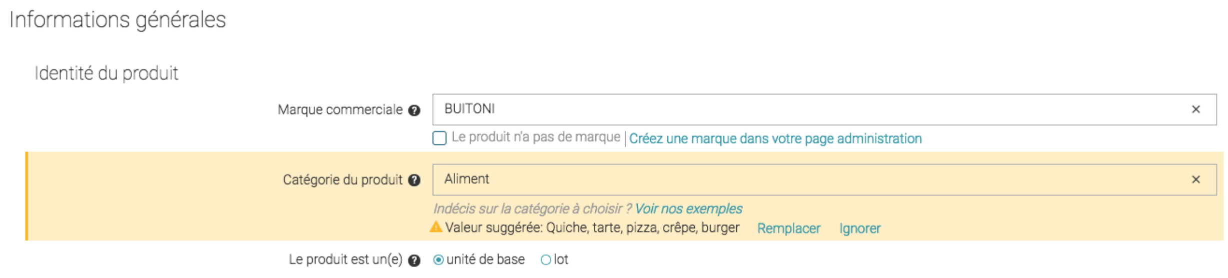
1 / 1 page • 5 entities in total • 0 selected

| Name/Brand | Fieldmetadata name | Suggested value | Score |
|---|---|---|---|
| Grany Coeur Fondant Gout Chocolat Noisettes 1… Grany | hasnotableingredients | arachide | 4 |
| Grany Coeur Fondant Gout Chocolat Noisettes 1… LU,Mondelez | islabeledby | Eco-embalage | 2 |

Figure 2.5: Suggestion validation by Alkemics.

### 2.2.7 Suggestion display to users in product pages, and acceptation (or dismissal) from manufacturer

Now that the suggestion is validated, we notify the manufacturer in charge of this product, and include a control in his product page allowing him to easily accept of dismiss the suggestion.



Informations générales

Identité du produit

Marque commerciale: BUITONI
☐ Le produit n'a pas de marque | Créez une marque dans votre page administration

Catégorie du produit: Aliment
*Indécis sur la catégorie à choisir ? Voir nos exemples*
⚠ Valeur suggérée: Quiche, tarte, pizza, crêpe, burger Remplacer Ignorer

Le produit est un(e): [✓ unité de base] lot

Figure 2.6: Suggestion acceptation by user.

In case of acceptation, the product's data is changed in Alkemics product database. Else, in case of dismissal, we store the fact that the suggestion is 'wrong' so we don't notify the manufacturer if we repredict the same value the day after.

## 2.3 Regular maintenance and improvement tasks

Now that the workflow is deployed, work isn't over.

### 2.3.1 Monitoring

Given the complexity of the workflow, and quality expected by clients, we have to closely monitor some metrics:

- did the workflow worked, if not where did it fail
- is there an evolution of classes distribution
- how our scores behave at training stage
- whether our predictions get rejected at validation stage
- do our validated suggestion get dismissed by user at acceptation stage

**Check workflow completion**

Our workflow consists of lots of interdependent tasks. Some of them have to be completed in a precise order. Some can be parallelized, some not. To ensure that our workflow runs as expected we use a workflow management tool: Luigi.

By defining each task dependencies (a task can depend on multiple other tasks), Luigi will draw an acyclic directed dependency graph.

Luigi library comes with a useful web interface, showing wich tasks are pending/runing/completed or have failed:

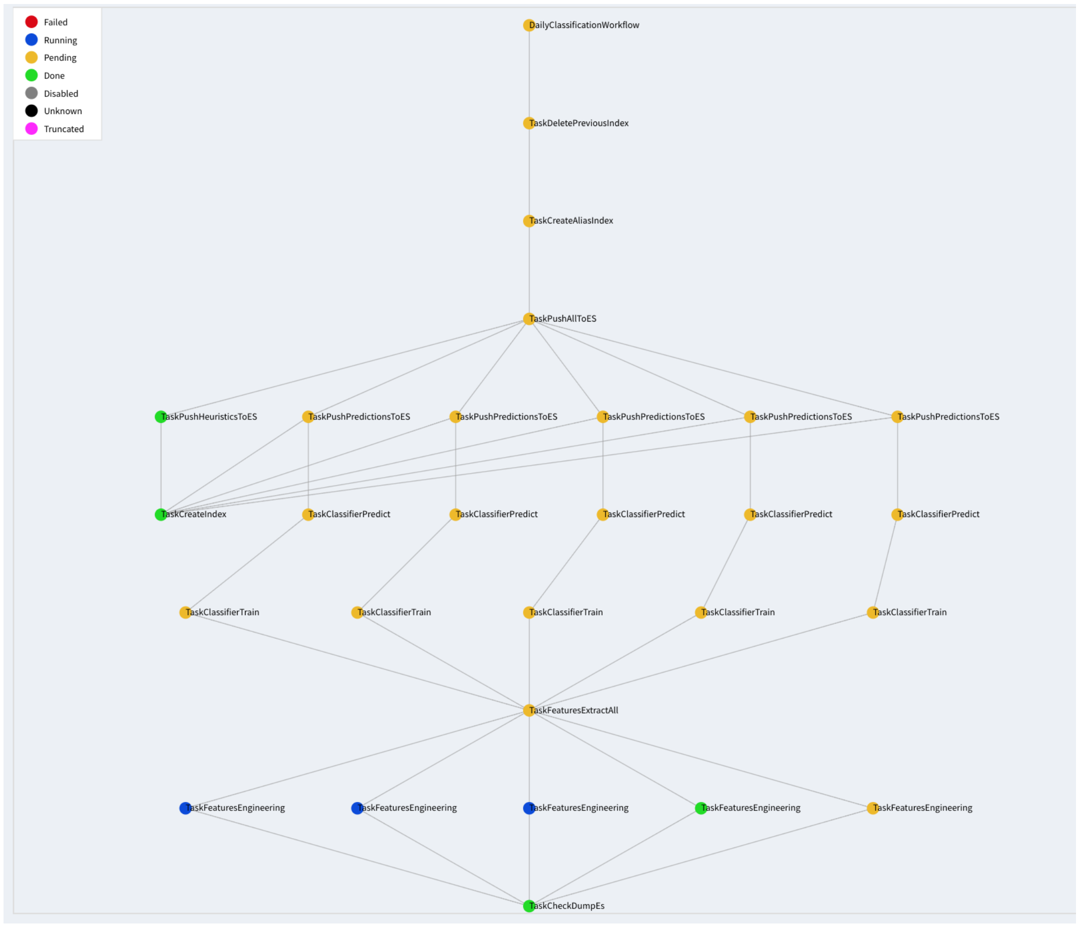


Figure 2.7: Classification workflow dependency graph.

### See and understand scores evolution

Knowing how our models perform is crucial to know how confident we can be into our models.

To do so I built a monitoring interface drawing the scores evolution for different fields we try to enrich.

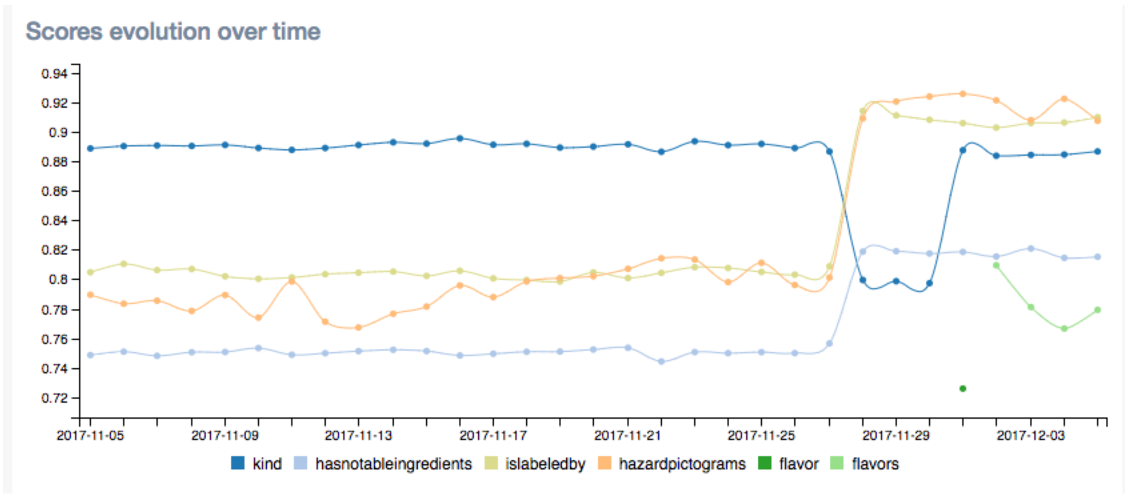


Figure 2.8: Classification micro-average global scores evolution.

Among the different reasons why our models scores can evolve, one is the change of classes distribution, so some graphs can help check if something is wrong: in this case, the following graph helped us find why a field predictions dropped: some wrongly labeled data was introduced.

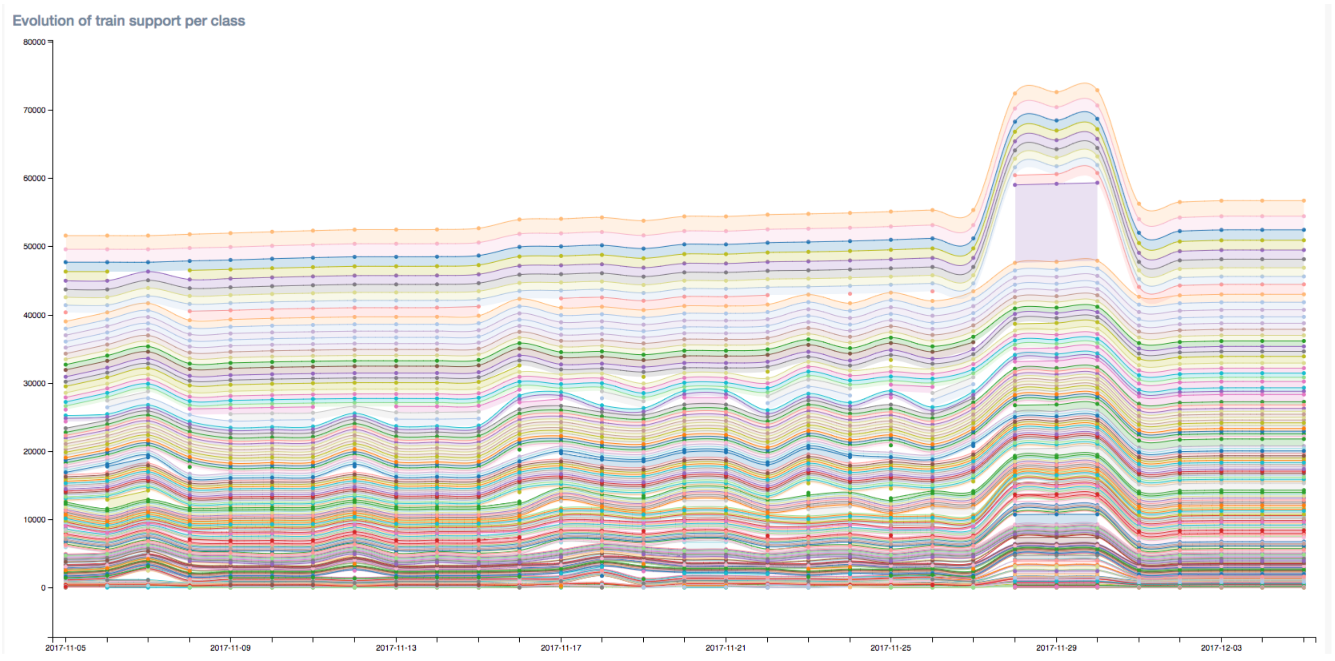


Figure 2.9: Local classes distribution evolution. We can detect the creation of an invalid class that led to dropping scores.

### Check validation / Acceptation rates

In machine learning, the more frequent way to estimate a model performance is to predict on a test-set and compare predictions to true labels of the test-set.

In our case, we can also check the validation/refusal ratio at validation stage:

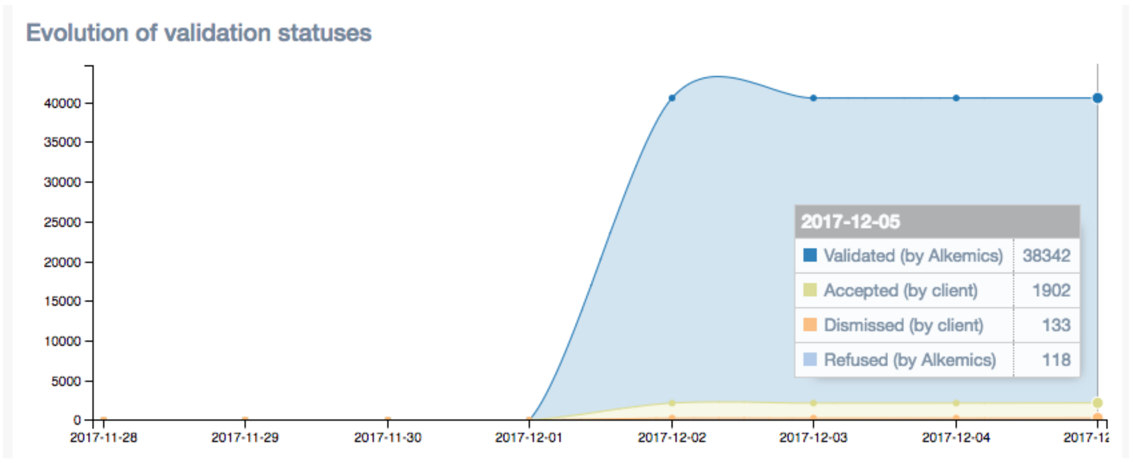


Figure 2.10: Validation/Acceptation rates.

**Evaluate business impact**

### 2.3.2 Deliver suggestion validation

By a mix of automation and manual validation. A big effort has been made to build an internal suggestion validation dashboard.

Even though it is not 'sexy', it is definitely necessary until we have enough feedback and experience.

### 2.3.3 Tune fasttext hyperparameters

Each predicted field data-set has different charateristics:

- number of different classes: from 5 to 300
- classes disparity: some fields have very distinct classes, whereas some others have very related classes.
- label density (for multilabel): do your products have on average 1 label or 5 different labels ?

Then, each of these tasks might require different model hyper-parameters (those parameters will be detailed in the next chapter). That's why we build a service on a dedicated machine whose goal is to find the best parameters.

### 2.3.4 Bootstrap new classes/fields

Everyday at Alkemics, clients use our plateform, and the data changes, lives. Thus, unlike in kaggle competition, our datasets evolves. For some reasons we might want not to take into account some deprecated classes. For instance the kind field.

Explain ...

# Chapter 3

# Machine learning challenges

## 3.1 Natural Language Processing

When dealing with natural language, one of most important common denominator is how we represent words as input to any of our models. How do we transform sentences/words into numerical vectors.

In this section, we will quickly describe Mikolov's Word2Vec models (Continuous Bag of Word, and Skip-Gram), that proposed a new way to build word embeddings based on words coocurrence in a context window.

Then we will explain in more detail recent work from Facebook, that propose new models based on Mikolov's work, allowing the use of sub-word information, and proposing a novel way to compute classification tasks while learning embeddings.

### 3.1.1 Word2Vec models

Word2vec can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words.

#### Continuous Bag of Words

#### Skip-Gram

### 3.1.2 Fasttext

Word2Vec treats each word in corpus like an atomic entity and generates a vector for each word. It treats words as the smallest unit to train on.

Fasttext (which is essentially an extension of word2vec model), treats each word as composed of character ngrams. So the vector for a word is made of the sum of this character n grams. For example the word vector "apple" is a sum of the vectors of the n-grams "<ap", "app", "appl", "apple", "apple>", "ppl", "pple", "pple>", "ple", "ple>", "le>" (assuming hyperparameters for smallest ngram[minn] is 3 and largest ngram[maxn] is 6).

This difference manifests as follows.

- Generate better word embeddings for rare words ( even if words are rare their character n grams are still shared with other words - hence the embeddings can still be good). This is simply because, in word2vec a rare word (e.g. 10 occurrences) has fewer neighbors to be tugged by, in comparison to a word that occurs 100 times - the latter has more neighbor context words and hence is tugged more often resulting in better word vectors.
- Out of vocabulary words - they can construct the vector for a word from its character n grams even if word doesn't appear in training corpus. Both Word2vec and Glove can't. From a practical usage standpoint, the choice of hyperparamters for generating fasttext embeddings becomes key since the training is at character n-gram level, it takes longer to generate fasttext embeddings compared to word2vec - the choice of hyper parameters controlling the minimum and maximum n-gram sizes has a direct bearing on this time.
- Simulatenous embedding and classification learning: in Word2Vec, we first build our embeddings, and then use it to translate words in features of low dimensionality for further classification. In fasttext, embedding and classification are different layers of a neural network that are trained simultaneously.

[2, Enriching Word Vectors with Subword Information] [3, Bag of Tricks for Efficient Text Classification]

**Model**

Suppose a previous layer $h$ (taking a vector x of given size in ouput space of size d) as follow:

$$\begin{aligned} h &= \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{dim} \end{bmatrix} && (3.1) \\ &= A^\top x && (3.2) \end{aligned}$$

$$h_j = A^{(j)\top} x \tag{3.3}$$

With $A$ being the matrix of weights from input to hidden layer, of size $(dim, V)$, ie dimension of vector embeddings $\times$ vocabulary size.

$$A = \begin{bmatrix} | & | & | & | \\ A^{(1)} & A^{(2)} & \cdots & A^{(dim)} \\ | & | & | & | \end{bmatrix} \tag{3.4}$$

$$A = \begin{bmatrix} -- & A_{(1)} & -- \\ -- & A_{(2)} & -- \\ & \vdots & \\ -- & A_{(V)} & -- \end{bmatrix} \tag{3.5}$$

Let’s denote:

$$s_k = B^{(k)\top} h = \sum_{j=1}^{dim} B_j^{(k)\top} h_j \tag{3.6}$$

we can express $f$ as, $\forall k \in [1, K]$:

$$f_k = \sigma(s_k) = \sigma(\sum_{j=1}^{d} B_j^{(k)\top} h_j) \tag{3.7}$$

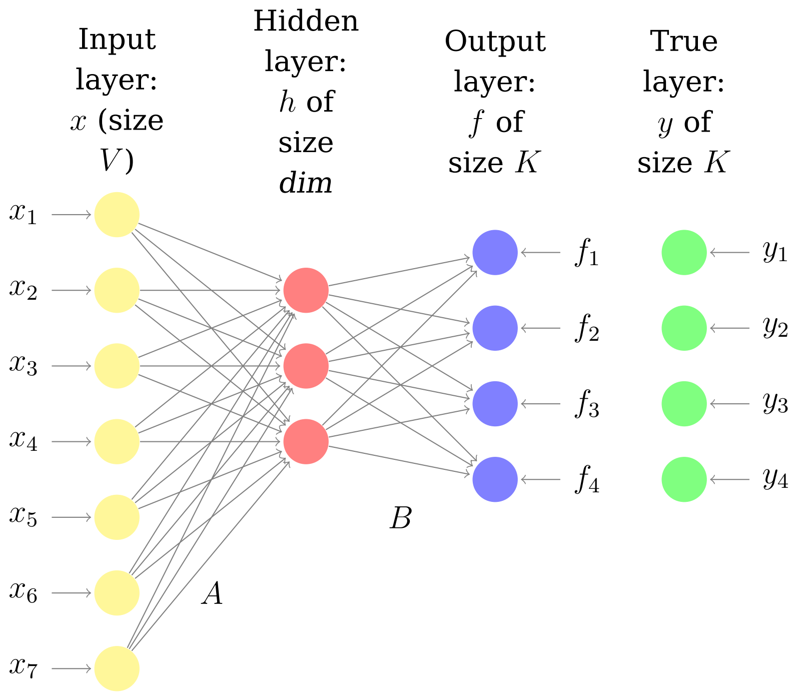
Lets consider the following error on one sample $(x, y)$:

$$E = -\sum_{k=1}^{K} \begin{cases} \log(f_k) & \text{if } y_k = 1 \\ \log(1 - f_k) & \text{if } y_k = 0 \end{cases} \tag{3.8}$$

This error is relevant since the minimization of this error is equivalent to maximizing the maximum of (log-)likelihood. Plus, it is interpretable: if the $k^{th}$ component $y_k$ is:

- positive ($y_k = 1$): then the loss increase if $f_k$ is near 0, and decrease if $f_k$ is near 1.
- negative ($y_k = 0$): then the loss decrease if $f_k$ is near 0, and increase if $f_k$ is near 1.

We can represent it as the following multilayer network:


Input layer: $x$ (size $V$)
Hidden layer: $h$ of size *dim*
Output layer: $f$ of size $K$
True layer: $y$ of size $K$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$x_6$
$x_7$
$A$
$B$
$f_1$
$f_2$
$f_3$
$f_4$
$y_1$
$y_2$
$y_3$
$y_4$

**Gradient retropropagation**

$$\frac{\partial E}{\partial f_k} = \begin{cases} -\frac{1}{f_k} & \text{if } y_k = 1 \\ \frac{1}{1-f_k} & \text{if } y_k = 0 \end{cases} \tag{3.9}$$

$$\frac{\partial E}{\partial s_k} = \frac{\partial E}{\partial f_k} \cdot \frac{\partial f_k}{\partial s_k} = \begin{cases} -\frac{1}{f_k} \cdot f_k(1-f_k) & \text{if } y_k = 1 \\ \frac{1}{1-f_k} \cdot f_k(1-f_k) & \text{if } y_k = 0 \end{cases} \tag{3.10}$$

$$= \begin{cases} f_k - 1 & \text{if } y_k = 1 \\ f_k & \text{if } y_k = 0 \end{cases} \tag{3.11}$$

$$= f_k - y_k \tag{3.12}$$

$$\frac{\partial E}{\partial B_i^{(k)}} = \frac{\partial E}{\partial s_k} \cdot \frac{\partial s_k}{\partial B_i^{(k)}} = h_i(f_k - y_k) \tag{3.13}$$

Trick: derivate $E$ in regard to $s_k$:

$$\frac{\partial E}{\partial h_j} = \sum_{k=1}^{K} \frac{\partial E}{\partial s_k} \cdot \frac{\partial s_k}{\partial h_j} \tag{3.14}$$

$$= \sum_{k=1}^{K} B_j^{(k)}(f_k - y_k) \tag{3.15}$$

$$\frac{\partial E}{\partial A_i^{(j)}} = \frac{\partial E}{\partial h_j} \cdot \frac{\partial h_j}{\partial A_i^{(j)}} \tag{3.16}$$

$$= x_i \sum_{k=1}^{K} B_j^{(k)}(f_k - y_k) \tag{3.17}$$

**Gradient descent**

At each sample $(x, y)$, given a learning rate $\mu$, we update weight as follow:
$B$ weights:

$$B_i^{(k)\text{new}} \leftarrow B_i^{(k)\text{old}} - \mu(h_i(f_k - y_k)) \tag{3.18}$$

$A$ weights:

$$A_i^{(j)\text{new}} \leftarrow A_i^{(j)\text{old}} - \mu\left(x_i \sum_{k=1}^{K} B_j^{(k)\text{new}}(f_k - y_k)\right) \tag{3.19}$$

### 3.1.3 Fasttext parameters implementation

List of all parameters:

- *epoch*
- *lr*
- *lrUpdateRate*
- *dim*
- *ws*
- *loss*
- *neg*

- *minCountLabel*
- *minCount*
- *minn*
- *maxn*
- *bucket*
- *t*

## Dictionnary

## Learning rate

Fasttext implementation lets you choose two parameters to control $\mu$ over time:

- *lr*: sets $\mu$ at initialization
- *lrUpdateRate*: how continuous *vs* per steps *lr* decays

The learning rate $\mu$ decrease linearly, from initial given parameter $lr$ to zero.

## 3.2 Multilabel classification

### 3.2.1 Review of major algorithms

**Problem Transformation Approaches**

**Adaptative Approaches**

**Neural network**

### 3.2.2 Multiclass/multilabel scoring metrics

**Regular binary metrics**

Evaluation metrics should take into account the class imbalance between churners and non-churners. A greater cost should be associated with false negatives than with false positives: misidentify a potential churner costs far more to a company than identifying a non-churner as churner. Moreover, we are more interested in perfectly identifying those customers who are most likely to churn than perfectly identifying *all* the potential churners.

**Binary classification metrics**

As we are in a binary classification problem, several metrics can be used to assess the performance of the models.

Lets define some naming/metrics used in the following:

| | |
|---|---|
| $TP/FP$ | $true/false\ positives$ |
| $TN/FN$ | $true/false\ negatives$ |
| $P$ | $TP + FN$ |
| $N$ | $TN + FP$ |
| $recall$ | $TP/P$ |
| $precision$ | $TP/(TP + FP)$ |

Precision, recall and accuracy are often used to measure the classification quality of binary classifiers.

**F-score**

The $F_1$ is a measure of a classifier accuracy, computed as the harmonic mean of precision and recall:

$$F_1 = 2.\frac{precision.recall}{precision + recall}$$

This definition assign the same weight to precision and recall but depending on the problem, one can be more valuable than the other.

To handle these constraints, the $F_\beta - score$ is adapted: in consists in the *weighted* harmonic mean of recall and precision, assigning $\beta$ times as much importance to recall as precision:

$$F_\beta = (1+\beta^2)\frac{precision.recall}{\beta^2.precision + recall}$$

**Calibration plot**

When dealing with probabilistic classifiers, one of the signs that a suitable classification model has been found is also that predicted probabilities (scores) are well calibrated, that is that a fraction of about $p$ of events with predicted probability $p$ actually occurs. Calibration plot is a method that shows us how well the classifier is calibrated [1]:

$$x = true\, probability,\ \ y = predicted\, probability$$

True probabilities are calculated for (sub)sets of examples with the same predicted score $P(y=1|X) \in [c-\delta, c+\delta]$:

$$p^c_{true} = \frac{P^c_{sub}}{P^c_{sub} + N^c_{sub}}$$

with $P^c_{sub}, N^c_{sub}$ being the proportion on positives and negative examples in a given subset with predicted probabilities $[c-\delta, c+\delta]$. The calibration plot of a perfectly calibrated classifier will be a diagonal.

**Adaptation to multiclass-multilabel: global / local metrics**

## 3.3 Calibration

### 3.3.1 Predictions precision/recall estimation

## 3.4 Invalid/sparse data

### 3.4.1 Strategies to handle sparse data

Consider recall rather than precision. The presence of a lot a FN will impact precision, not recall.

### 3.4.2 Iterative approach to bootstrap new classes

Manually label some products with new classes, then focus validation on these products. The validated suggestions will be used in next classification workflow.

After some iterations, there might be enough occurences to ensure sufficient scores.

## 3.5 Misc

### 3.5.1 Dive into kind tree-structure

### 3.5.2 Enforce better calibration

### 3.5.3 Tuning strategies

# Conclusion

During this project, we transformed a workflow dedicated to one multiclass field, to a much flexible and adaptable workflow.

Machine learning hype drives attention mostly to models and algorithms. Yet, this project convinced me that 90% of a project success depends on pragmatism and implementation.

# Acknowledgment

# Appendix A

# Logistic Regression: from binary, to multiclass, to multilabel

## Common notations

### Notations for single-labeled samples

#### Probabilistic notations

- X random vector, $\mathcal{X} \subset \mathbb{R}^d$

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_d \end{bmatrix} \tag{A.1}$$

- Y discrete random variable, $\mathcal{Y}$, with $|\mathcal{Y}|$ finite. In practice, Y is encoded as an integer, ie $\mathcal{Y} \subset \mathbb{N}$ and $Y \in \mathbb{N}$
- P joint probability of X and Y $(X, Y)$ (unknown).

#### Empirical notations

- $x$, a vector sample, which is a realization of X (the empirical counterpart of X) $x \in \mathbb{R}^d$:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \tag{A.2}$$

- $y$, a sample, which is a realization of Y (the empirical counterpart of Y): $y \in \mathcal{Y} \subset \mathbb{N}$

- $\mathcal{D}_m = \{(x^{(i)}, y^{(i)}), i \in 1, m\} \in (\mathcal{X}, \mathcal{Y})^m$ a learning set containing $m$ samples $iid$ and from P.

Note: For the $i^{th}$ sample, $x^{(i)}$ $y^{(i)}$, which are vectors, we'll respectively denote the $j^{th}$ component as follow: $x_j^{(i)}$ and $y_j^{(i)}$

### Notations for multi-labeled samples

Everything is the same except that labels are no longer scalars but vectors, with $K = |\mathcal{Y}|$:

- Y random vector, $\mathcal{Y} \subset \mathbb{N}^d$

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_K \end{bmatrix} \tag{A.3}$$

- Y discrete random variable, $\mathcal{Y}$, with $|\mathcal{Y}|$ finite. In practice, Y is encoded as an integer, ie $\mathcal{Y} \subset \mathbb{N}$ and $Y \in \mathbb{N}$
- P joint probability of X and Y $(X, Y)$ (unknown).

## A.1 Binary case: Logistic regression

We want to predict the value of $y^{(i)}$ for the example $x^{(i)}$ using a function $y = h_\theta(x)$ with binary-valued labels $\left(y^{(i)} \in \{0, 1\}\right)$. We want to predict the probability that a given example belongs to the "1" class versus the probability that it belongs to the "0" class. Specifically, we will try to learn a function of the form:

### A.1.1 Model

$$\begin{aligned} P(Y = 1|X = x) &= f_\theta(x) && \text{(A.4)} \\ &= \frac{1}{1 + \exp(-\theta^\top x)} && \text{(A.5)} \\ &= \sigma(\theta^\top x), && \text{(A.6)} \\ P(Y = 0|X = x) &= 1 - f_\theta(x) && \text{(A.7)} \\ &= \sigma(-\theta^\top x) && \text{(A.8)} \end{aligned}$$

With:

$$\sigma(z) \equiv \frac{1}{1+\exp(-z)} \tag{A.9}$$

the "logistic" function. It is an S-shaped function that "squashes" the value of $\theta^\top x$ into the range [0, 1] so that we may interpret $f_\theta(x)$ as a probability.

Our goal is to search for a value of $\theta$ so that the probability $P(Y=1|x) = f_\theta(x)$ is large when x belongs to the "1" class and small when x belongs to the "0" class (so that $P(Y=0|x)$ is large).

For further use, note than for $k \in \{0,1\}$:

$$P(Y=k|x) = (f_\theta(x))^k \times (1-f_\theta(x))^{(1-k)} \tag{A.10}$$

### A.1.2 Objective

For a set of training examples with binary labels $\{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$, assuming that the $m$ training examples were generated independently, we can then write down the likelihood of the parameters as:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} P(Y=y^{(i)}|x^{(i)},\theta) && \text{(A.11)} \\ &= \prod_{i=1}^{m} (f_\theta(x^{(i)}))^{y^{(i)}} \times (1-f_\theta(x^{(i)})^{(1-y^{(i)})}) && \text{(A.12)} \end{aligned}$$

It will be easier to maximize the log likelihood:

$$\begin{aligned} l(\theta) &= \log L(\theta) && \text{(A.13)} \\ &= \log\left(\prod_{i=1}^{m} P(Y=y^{(i)}|x^{(i)})\right) && \text{(A.14)} \\ &= \sum_{i=1}^{m} \left(y^{(i)} \log f_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-f_\theta(x^{(i)}))\right) && \text{(A.15)} \end{aligned}$$

It is the same as minimizing the following (cross entropy loss):

$$E(\theta) = -\sum_i \left(y^{(i)} \log(f_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-f_\theta(x^{(i)}))\right). \tag{A.16}$$

Note that only one of the two terms in the summation is non-zero for each training example (depending on whether the label $y^{(i)}$ is 0 or 1). When $y^{(i)}=1$ minimizing the cost function means we need to make $f_\theta(x^{(i)})$ large, and when $y^{(i)}=0$ we want to make $1-f_\theta$ large as explained above.

We now have a cost function that measures how well a given hypothesis $f_\theta$ fits our training data. We can learn to classify our training data by minimizing $E(\theta)$ to find the best choice of $\theta$. Once we have done so, we can classify a new test point as “1” or “0” by checking which of these two class labels is most probable: if $P(y=1|x) > P(y=0|x)$ then we label the example as a “1”, and “0” otherwise. This is the same as checking whether $f_\theta(x) > 0.5$.

### A.1.3 Optimization

We want to minimize $E(\theta)$. The derivative of $E(\theta)$ as given above with respect to $\theta_j$ is:

$$\frac{\partial E(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)}(f_\theta(x^{(i)}) - y^{(i)}) \tag{A.17}$$

Demonstration available here.

Written in its vector form, the entire gradient can be expressed as ($x^{(i)}$ being a d dimensional vector, with d being the dimension of each sample feature):

$$\nabla_\theta E(\theta) = \begin{bmatrix} \frac{\partial E(\theta)}{\partial \theta_1} \\ \frac{\partial E(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial E(\theta)}{\partial \theta_d} \end{bmatrix} = \sum_i x^{(i)}(f_\theta(x^{(i)}) - y^{(i)}) \tag{A.18}$$

We can then apply gradient descent to find global minimum since $E(\theta)$ is convex, with learning rate $\alpha$:

$$\theta \leftarrow \theta - \alpha \nabla_\theta E(\theta) \tag{A.19}$$

### A.1.4 Vectorial representations

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{bmatrix} \tag{A.20}$$

$$\mathcal{D}_m = \begin{bmatrix} -x^{(1)}- \\ -x^{(2)}- \\ \vdots \\ -x^{(m)}- \end{bmatrix} \tag{A.21}$$

## A.2 Multiclass case: Softmax regression

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes. In logistic regression we assumed that the labels were binary: $y^{(i)} \in \{0, 1\}$. Softmax regression allows us to handle $y^{(i)} \in \{1, \ldots, K\}$ where K is the number of classes.

In the softmax regression setting, we are interested in multi-class classification (as opposed to only binary classification), and so the label y can take on K different values, rather than only two. Thus, in our training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, we now have that $y^{(i)} \in \{1, 2, \ldots, K\}$. (Note that our convention will be to index the classes starting from 1, rather than from 0.)

### A.2.1 Model

Given a test input x, we want our hypothesis to estimate the probability $P(Y = k|x)$ for each value of $k = 1, \ldots,$ K. i.e. we want to estimate the class probabilities of the K different possible labels. Thus, our hypothesis function will output multinomial distribution: a K-dimensional vector (whose elements sum to 1) giving us our K estimated probabilities. Concretely, our hypothesis $f_\theta(x)$ takes the form:

$$f_\theta(x) = \begin{bmatrix} P(Y = 1|x;\theta) \\ P(Y = 2|x;\theta) \\ \vdots \\ P(Y = K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix} \tag{A.22}$$

Here $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)} \in \mathbb{R}^d$ are the parameters of our model. Notice that the term $\frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)}$ normalizes the distribution, so that it sums to one.

For convenience, we will also write $\theta$ to denote all the parameters of our model. When you implement softmax regression, it is usually convenient to represent $\theta$ as a n-by-K matrix obtained by concatenating $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ into columns, so that:

$$\theta = \left[ \begin{array}{cccc} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \cdots & \theta^{(K)} \\ | & | & | & | \end{array} \right] \tag{A.23}$$

With each $\theta^{(i)}$ being a vector $\in \mathbb{R}^d$:

$$\theta^{(i)} = \begin{bmatrix} \theta_1^{(i)} \\ \theta_2^{(i)} \\ \vdots \\ \theta_d^{(i)} \end{bmatrix} \tag{A.24}$$

### A.2.2 Objective

We now describe the cost function that we'll use for softmax regression.

For a set of training examples with multiclass labels $\{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$, assuming that the $m$ training examples were generated independently, we can then write down the likelihood of the parameters as:

$$L(\theta) = \prod_{i=1}^{m} P(Y = y^{(i)} | x^{(i)}) \tag{A.25}$$

$$= \prod_{i=1}^{m} (f_\theta(x^{(i)}))^{y^{(i)}} \times (1 - f_\theta(x^{(i)})^{(1-y^{(i)})}) \tag{A.26}$$

It will be easier to maximize the log likelihood:

$$l(\theta) = \log L(\theta) \tag{A.27}$$

$$= \log \left( \prod_{i=1}^{m} P(Y = y^{(i)} | x^{(i)}) \right) \tag{A.28}$$

$$= \sum_{i=1}^{m} \left( y^{(i)} \log f_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - f_\theta(x^{(i)})) \right) \tag{A.29}$$

It is the same as minimizing the following (cross entropy loss):

$$E(\theta) = - \left[ \sum_{i=1}^{m} \sum_{k=1}^{K} \mathbb{1} \left\{ y^{(i)} = k \right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} \right] \tag{A.30}$$

Notice that this generalizes the logistic regression cost function, which could also have been written:

$$E(\theta) = - \left[ \sum_{i=1}^{m} (1 - y^{(i)}) \log(1 - f_\theta(x^{(i)})) + y^{(i)} \log f_\theta(x^{(i)}) \right] \tag{A.31}$$

$$= - \left[ \sum_{i=1}^{m} \sum_{k=0}^{1} \mathbb{1} \left\{ y^{(i)} = k \right\} \log P(y^{(i)} = k | x^{(i)}; \theta) \right] \tag{A.32}$$

The softmax cost function is similar, except that we now sum over the K different possible values of the class label. Note also that in softmax regression, we have that

$$P(y^{(i)} = k|x^{(i)};\theta) = \frac{\exp(\theta^{(k)\top}x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x^{(i)})} \tag{A.33}$$

### A.2.3 Optimization

We cannot solve for the minimum of $E(\theta)$ analytically, and thus as usual we'll resort to an iterative optimization algorithm. Taking derivatives, one can show that the gradient is:

$$\nabla_{\theta^{(k)}}E(\theta) = -\sum_{i=1}^{m}\left[x^{(i)}\left(\mathbb{1}\{y^{(i)} = k\} - P(Y^{(i)} = k|x^{(i)};\theta)\right)\right] \tag{A.34}$$

In particular, $\nabla_{\theta^{(k)}}E(\theta)$ is itself a vector, so that its j-th element is $\frac{\partial E(\theta)}{\partial \theta_{lk}}$ the partial derivative of $E(\theta)$ with respect to the j-th element of $\theta^{(k)}$.

Armed with this formula for the derivative, one can then plug it into a standard optimization package and have it minimize $E(\theta)$.

### A.2.4 Properties of softmax regression parameterization

Softmax regression has an unusual property that it has a "redundant" set of parameters. To explain what this means, suppose we take each of our parameter vectors $\theta^{(j)}$, and subtract some fixed vector $\psi$ from it, so that every $\theta^{(j)}$ is now replaced with $\theta^{(j)} - \psi$ (for every j=1, . . ., k). Our hypothesis now estimates the class label probabilities as

$$\begin{aligned} P(y^{(i)} = k|x^{(i)};\theta) &= \frac{\exp((\theta^{(k)} - \psi)^\top x^{(i)})}{\sum_{j=1}^{K}\exp((\theta^{(j)} - \psi)^\top x^{(i)})} && \text{(A.35)} \\ &= \frac{\exp(\theta^{(k)\top}x^{(i)})\exp(-\psi^\top x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x^{(i)})\exp(-\psi^\top x^{(i)})} && \text{(A.36)} \\ &= \frac{\exp(\theta^{(k)\top}x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x^{(i)})}. && \text{(A.37)} \end{aligned}$$

In other words, subtracting $\psi$ from every $\theta^{(j)}$ does not affect our hypothesis' predictions at all! This shows that softmax regression's parameters are "redundant." More formally, we say that our softmax model is "'overparameterized,"' meaning that for any hypothesis we might fit to the data, there are multiple parameter settings that give rise to exactly the same hypothesis function $h_\theta$ mapping from inputs x to the predictions.

Further, if the cost function $E(\theta)$ is minimized by some setting of the parameters $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(k)})$, then it is also minimized by $(\theta^{(1)} - \psi, \theta^{(2)} - \psi, \ldots, \theta^{(k)} - \psi)$ for any value of $\psi$. Thus, the minimizer of $E(\theta)$ is not unique. (Interestingly, $E(\theta)$ is still convex, and thus gradient descent will not run into local optima problems. But the Hessian is singular/non-invertible, which causes a straightforward implementation of Newton's method to run into numerical problems.)

Notice also that by setting $\psi = \theta^{(K)}$, one can always replace $\theta^{(K)}$ with $\theta^{(K)} - \psi = \vec{0}$ (the vector of all 0's), without affecting the hypothesis. Thus, one could "eliminate" the vector of parameters $\theta^{(K)}$ (or any other $\theta^{(k)}$, for any single value of k), without harming the representational power of our hypothesis. Indeed, rather than optimizing over the $K\cdot$ n parameters $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}) (where \theta^{(k)} \in \Re^n)$, one can instead set $\theta^{(K)} = \vec{0}$ and optimize only with respect to the $K\cdot$ n remaining parameters.

### A.2.5 Relationship to Logistic Regression

In the special case where K = 2, one can show that softmax regression reduces to logistic regression. This shows that softmax regression is a generalization of logistic regression. Concretely, when K=2, the softmax regression hypothesis outputs

$$f_\theta(x) = \frac{1}{\exp(\theta^{(1)\top} x) + \exp(\theta^{(2)\top} x^{(i)})} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \end{bmatrix} \tag{A.38}$$

Taking advantage of the fact that this hypothesis is overparameterized and setting $\psi = \theta^{(2)}$, we can subtract $\theta^{(2)}$ from each of the two parameters, giving us

$$f(x) = \frac{1}{\exp((\theta^{(1)} - \theta^{(2)})^\top x^{(i)}) + \exp(\vec{0}^\top x)} \begin{bmatrix} \exp((\theta^{(1)} - \theta^{(2)})^\top x) \exp(\vec{0}^\top x) \end{bmatrix} \tag{A.39}$$

$$= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ \frac{\exp((\theta^{(1)}-\theta^{(2)})^\top x)}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \tag{A.40}$$

$$= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ 1 - \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \tag{A.41}$$

Thus, replacing $\theta^{(2)} - \theta^{(1)}$ with a single parameter vector $\theta'$, we find that softmax regression predicts the probability of one of the classes as $\frac{1}{1+\exp(-(\theta')^\top x^{(i)})}$, and that of the other class as $1 - \frac{1}{1+\exp(-(\theta')^\top x^{(i)})}$, same as logistic regression.

### A.2.6 Vectorial representations

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \dots & \theta^{(K)} \\ | & | & | & | \end{bmatrix} \tag{A.42}$$

## A.3 Multilabel case: ’multi-binary’ regression

In this case, suppose you want to predict which pictograms are present on a product, like ’organic food’, ’product of the year’ or ’made in France’. Each product can have multiple labels.

One possibility is to treat each class as an independent binary problem. Our sample label is no longer a scalar, but a vector of $d$ dimensions (number of possible classes).

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_K \end{bmatrix} \tag{A.43}$$

With $Y_i \in \{0,1\}$

Our sample then is not a single value but an indicator vector:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} \tag{A.44}$$

With $y_i \in \{0,1\}$

$$y_i = \begin{cases} 1 & \text{if the } i^{th} \text{ class is positive} \\ 0 & \text{else.} \end{cases} \tag{A.45}$$

### A.3.1 Model

Given a test input x, we want our hypothesis to independently estimate the probability that $P(Y = y|x)$. More specifically, for each possible class, we want to estimate the probability that the class is positive. Thus, our hypothesis will output a K-dimensional vector (whose elements don’t sum to 1! in opposition to softmax regression) giving us our K estimated probabilities. Concretely, our model $F_\theta(x)$ takes the form:

$$F_\theta(x) = \begin{bmatrix} f_{\theta^{(1)}}(x) \\ f_{\theta^{(2)}}(x) \\ \vdots \\ f_{\theta^{(K)}}(x) \end{bmatrix} \tag{A.46}$$

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \dots & \theta^{(K)} \\ | & | & | & | \end{bmatrix} \tag{A.47}$$

With each $\theta^{(i)}$ being a vector $\in \mathbb{R}^d$. For $i \in [1, K]$:

$$\theta^{(i)} = \begin{bmatrix} \theta_1^{(i)} \\ \theta_2^{(i)} \\ \vdots \\ \theta_d^{(i)} \end{bmatrix} \tag{A.48}$$

$$f_{\theta^{(i)}}(x) = P(Y_i = 1 | x; \theta^{(i)}(x)) \tag{A.49}$$
$$= \frac{1}{1 + \exp(-\theta^{(i)\top} x)} \tag{A.50}$$
$$= \sigma(\theta^{(i)\top} x) \tag{A.51}$$

### A.3.2 Objective

For a set of training examples with multilabel samples $\{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$, assuming that the $m$ training examples were generated independently, we can then write down the likelihood of the parameters as:

$$L(\theta) = \prod_{i=1}^{m} P(Y = y^{(i)} | x^{(i)}) \tag{A.52}$$
$$\tag{A.53}$$

Making the assumptions that all classes are independent (which is in pratice usually inaccurate but theorically necessary):

$$L(\theta) = \prod_{i=1}^{m} \prod_{j=1}^{K} P(Y_j = y_j^{(i)} | x^{(i)}) \tag{A.54}$$

It will be easier to minimize the negative log likelihood:

$$E(\theta) = -\log L(\theta) \tag{A.55}$$

$$= -\sum_{i=1}^{m}\sum_{j=1}^{K} \log P(Y_j = y_j^{(i)} | x^{(i)}) \tag{A.56}$$

$$= -\sum_{i=1}^{m}\sum_{j=1}^{K} \begin{cases} \log(\sigma(\theta^{(j)\top}x^{(i)})) & \text{if } y_j^{(i)} = 1 \\ \log(1-\sigma(\theta^{(j)\top}x^{(i)})) & \text{if } y_j^{(i)} = 0 \end{cases} \tag{A.57}$$

$$\tag{A.58}$$

## Optimization

We want to minimize $E(\theta)$. The derivative of $E(\theta)$ as given above with respect to $\theta_j^{(k)}$, $j \in [1, d]$ is:

$$\frac{\partial E(\theta)}{\partial \theta_k^{(j)}} = \sum_{i=1}^{m} x_k^{(i)}(\sigma(\theta^{(j)\top}x^{(i)}) - y_j^{(i)}) \tag{A.59}$$

$$= \sum_{i=1}^{m} x_k^{(i)}(f_{\theta^{(j)}}(x^{(i)}) - y_j^{(i)}) \tag{A.60}$$

Hints:

$$\frac{\partial(\theta^{(j)\top}x^{(i)})}{\partial \theta_k^{(j)}} = x_k^{(i)} \tag{A.61}$$

$$(log \circ \sigma)'(t) = \sigma(-t) \tag{A.62}$$

The entire gradient can be expressed as a $d \times K$ dimensional matrix (with d being the dimension of each sample feature and K the number of classes):

$$\nabla_\theta E(\theta) = \begin{bmatrix} \frac{\partial E(\theta)}{\partial \theta_1^{(1)}} & \cdots & \frac{\partial E(\theta)}{\partial \theta_1^{(K)}} \\ \cdots & \frac{\partial E(\theta)}{\partial \theta_i^{(j)}} & \cdots \\ \frac{\partial E(\theta)}{\partial \theta_d^{(1)}} & \cdots & \frac{\partial E(\theta)}{\partial \theta_d^{(K)}} \end{bmatrix} \tag{A.63}$$

We can then apply gradient descent to find global minimum since $E(\theta)$ is convex, with learning rate $\alpha$:

$$\theta \leftarrow \theta - \alpha \nabla_\theta E(\theta) \tag{A.64}$$

## Context

This method is nearly equivalent to applying binary relevance (problem transformation method) to logistic regression.

Yet:

- it can be considered as an adaptative method (in opposition to problem transformation method): the algorithm is adapted to fit the data
- an advantage of this algorithm is to be able to compute gradient for each sample evaluated during training. This allows to retropropagate gradient to previous layers in the context of a multi-layer neural network. In comparison, using binary relevance method with any classifier does not allow this.

## Gradient retro-propagation in multilayer neural network

### Network definition

Suppose a previous layer $h$ (taking a vector x of given size in ouput space of size d) as follow:

$$h = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_d \end{bmatrix} \tag{A.65}$$

With $g$ being a function parametrized by w so that:

$$h_j = g_{W^{(j)}}(x) \tag{A.66}$$

With $W$ being the matrix of weights from input to hidden layer.

Let's denote:

$$s_k = \theta^{(k)\top} h = \sum_{j=1}^{d} \theta_j^{(k)\top} h_j \tag{A.67}$$

we can express $f$ as, $\forall k \in [1, K]$:

$$f_k = \sigma(s_k) = \sigma(\sum_{j=1}^{d} \theta_j^{(k)\top} h_j) \tag{A.68}$$

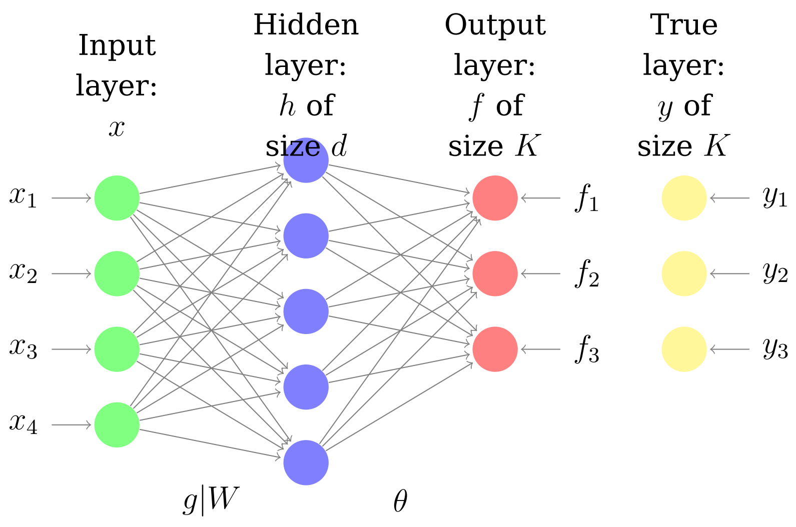
Lets consider the following error on one sample $(x, y)$:

$$E = -\sum_{k=1}^{K} \begin{cases} \log(f_k) & \text{if } y_k = 1 \\ \log(1 - f_k) & \text{if } y_k = 0 \end{cases} \tag{A.69}$$

This error is relevant since the minimization of this error is equivalent to maximizing the maximum of (log-)likelihood. Plus, it is interpretable: if the $k^{th}$ component $y_k$ is:

- positive ($y_k = 1$): then the loss increase if $f_k$ is near 0, and decrease if $f_k$ is near 1.
- negative ($y_k = 0$): then the loss decrease if $f_k$ is near 0, and increase if $f_k$ is near 1.

We can represent it as the following multilayer network:


Input layer: $x$
Hidden layer: $h$ of size $d$
Output layer: $f$ of size $K$
True layer: $y$ of size $K$
$x_1$
$x_2$
$x_3$
$x_4$
$f_1$
$f_2$
$f_3$
$y_1$
$y_2$
$y_3$
$g|W$
$\theta$

**Gradient retropropagation**

$$\frac{\partial E}{\partial f_k} = \begin{cases} -\frac{1}{f_k} & \text{if } y_k = 1 \\ \frac{1}{1-f_k} & \text{if } y_k = 0 \end{cases} \tag{A.70}$$

$$\frac{\partial E}{\partial s_k} = \frac{\partial E}{\partial f_k} \cdot \frac{\partial f_k}{\partial s_k} = \begin{cases} -\frac{1}{f_k} \cdot f_k(1-f_k) & \text{if } y_k = 1 \\ \frac{1}{1-f_k} \cdot f_k(1-f_k) & \text{if } y_k = 0 \end{cases} \tag{A.71}$$

$$= \begin{cases} f_k - 1 & \text{if } y_k = 1 \\ f_k & \text{if } y_k = 0 \end{cases} \tag{A.72}$$

$$= f_k - y_k \tag{A.73}$$

$$\frac{\partial E}{\partial \theta_i^{(k)}} = \frac{\partial E}{\partial s_k} \cdot \frac{\partial s_k}{\partial \theta_i^{(k)}} = h_i(f_k - y_k) \tag{A.74}$$

Trick: derivate $E$ in regard to $s_k$:

$$\frac{\partial E}{\partial h_i} = \sum_{k=1}^{K} \frac{\partial E}{\partial s_k} \cdot \frac{\partial s_k}{\partial h_i} \tag{A.75}$$

$$= \sum_{k=1}^{K} \theta_i^{(k)}(f_k - y_k) \tag{A.76}$$

We now have the gradient of E in regard to the output of the hidden layer. The equation to update $W$ will depend of which $g$ is chosen.

# Bibliography

[1] R. Caruana. A. Niculscu-Mizil. Predicting good probabilities with supervised learning. *Proceeding: ICML ’05 Proceedings of the 22nd international conference on Machine learning*, 22:625 – 632, 2005.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[3] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.