UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Maximum entropy method for multi-label classification

Dissertation presented by
Alexander GERNIERS

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor
Marco SAERENS

Readers
Sylvain COURTAIN, Pierre DUPONT, Marco SAERENS

Academic year 2017-2018

First, I would like to thank my supervisor, Prof. Marco Saerens, for his involvement in this master thesis, the help and the advice he provided throughout the elaboration of this work.

I would also like to thank my aunt Chantal for the time she spent proofreading my work.

Finally, I would like to thank my family and friends, and especially my parents, for their continued support during my studies.

Contents

Chapter 1

Introduction

Machine learning is one of the most popular fields of research in computer science today. It is used in many different contexts to face challenges related to big data analysis, personalised decisions, etc. A big part of what is done in machine learning is supervised classification, which assigns categorical labels to certain objects, based on information that was gathered for similar objects. It thus consists in predicting the class label of an object based on how its characteristics compare to those of other examples. A lot of research has been done in that area, with many different techniques being developed, as can be found in for example [1–5].

Many applications currently use supervised classification techniques, in various domains such as healthcare, finance, web applications, etc. As an example, we could be building a device to help cancer prevention. For this, we would need the (anonymised!) medical records of a set of cancer patients, as well as healthy persons. We can then train a supervised classification algorithm on this dataset, in order to obtain a model with two class labels: "at risk" and "no apparent risk". The goal is to predict as faithfully as possible, when we have a new patient, whether he is at risk of developing cancer, based on his medical record.

However, for more and more applications, predicting one single label can be seen as too restrictive. Many applications could benefit from predicting more than one label. For instance, this is the case for document classification, where different documents are sorted into different categories. Indeed, a document could be related to different subjects, so assigning only one category to it is not really relevant. Multi-label classification was therefore introduced to deal with such problems. It is a variant of traditional supervised learning, where an object is no longer classified by exclusively one label. Multiple class labels can thus be attributed to one single observation. Besides document classification, multi-label learning can be relevant, for instance, to medical or biological applications, internet or image tagging, etc.

The fact that multiple labels can be associated to one object constitutes a huge difference with respect to the traditional, single-label, classification task. Indeed, in the latter case, we only need to choose one label among a finite set of labels. The number of possible outcomes that can be given to an observation is thus simply the number of class labels present in the dataset. When the number of labels grows, the number of outputs will grow linearly. This means that single-label classification is scalable to applications with a huge number of class labels. This is not the case with multi-label classification.

Indeed, in the multi-label framework, we may choose any possible subset of the set of labels as outcome for an observation. Unfortunately, there is an exponential number of these subsets, in terms of the number of labels. This makes the classification problem much harder. For a problem with 34 class labels, which is not an excessively big number, there are already more possible outputs than the estimated number of years since the beginning of the universe (estimated at around 13 billion years). This is why it is extremely hard to develop a model that will consider in depth all the different possible outcomes. In particular, modelling the relationships between class labels is very tricky.

Maybe due to its exponential complexity, there has been less research performed in the area of multi-label classification, compared to its single-label counterpart. One approach that has been explored is called *problem transformation*. The goal of this approach is to solve multi-label problems by transforming them into a series of small single-label problems [6, 7]. Whilst these methods are widely used, they tend to not represent thoroughly the relationships between class labels. Generally, they either assume complete independence between labels, or only partially take into account relationships between them. When evaluating whether a label is present or not, these models usually predict without seeing the "full picture".

The goal of this master thesis is to develop an original multi-label classification method, based on the principle of model transformation, that explicitly models the relationships that might exist between class labels. In order to achieve this, the maximum entropy approach will be used to model 2-by-2 interactions between labels. To do this, we will use single-label classifiers to predict not only the probability of a label to appear, but also the probability of two labels to appear at the same time. This model can potentially be extended to higher order interaction, starting with 3-by-3 relations. One objective of this master thesis is to evaluate whether this new technique could constitute an improvement with respect to existing problem transformation techniques, such as *binary relevance* and *classifier chains*.

Such a maximum entropy model will require every possible outcome to be considered. Therefore, the model will have an exponential complexity. This means that it will only be usable in practice for multi-label problems that have a small number of labels. Therefore, one part of the work will be to see whether it is possible to reduce the complexity of the problem in order to be scalable to applications with large amounts of labels.

The objectives of this master thesis are therefore the following:

- Developing an original multi-label classification algorithm, based on the maximum entropy principle, and implement it.
- This new classifier should be compared to existing problem transformation algorithms in order to assess its performance. This evaluation should be done using an appropriate procedure and metric functions adapted to the multi-label setting.
- The original maximum entropy approach models 2-by-2 interactions between class labels. However, we can choose to add higher order interactions to the model. It would therefore be interesting to see whether adding 3-by-3 interactions could provide better classification.
- Since the maximum entropy has an exponential complexity, it would be useful to know whether there are ways to circumvent this. For example, we could use a Poisson regression

algorithm in order to predict in advance the number of labels that will be present in the output. If we are able to know that, we only need to model the outcomes that have this number of labels, thus removing a lot of unneeded outcomes from the model.

This master thesis will be divided in two parts. First, a theoretical part contained in chapters 2 and 3. The first one will be dedicated to the study of multi-label classification, namely what are the baseline multi-label classifiers and the metric functions that will be used in order to assess the performance of the maximum entropy classifier. The latter will be developed in the next chapter, where we will look at the maximum entropy principle, and how to adapt it to the problem of multi-label classification. The second part of this thesis, corresponding to chapters 4 and 5, will be dedicated to the empirical evaluation of the maximum entropy classifier.

Chapter 2

Theoretical basis

We will start this thesis by explaining what is multi-label classification. We will give a formal definition, and have a look at two classifiers that use problem transformation, the approach also used by the maximum entropy classifier that will be defined later. At the end of this chapter, we will see that multi-label classification requires special metric functions in order to evaluate the performance of classifiers.

2.1 What is multi-label classification?

Single-label supervised classification. First of all, let's start by defining the concept of *supervised classification* in machine learning. This subject is very widely covered in the literature, for example in [1–5]. It is the task of predicting a categorical output variable assigned to a real world object, called "example", that is described by a set of features. More formally, it consists in building a model that assigns a class label y, out of a set of possible labels $\mathcal{Y}$, to a vector of features $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the space of all possible features. We thus learn a function:

$$f : \mathcal{X} \to \mathcal{Y} : \mathbf{x} \mapsto y \tag{2.1}$$

For example, we could have a vector of features describing meteorological conditions of a certain day, e.g. temperature, air pressure, humidity, etc., and build a model to predict whether that day would be sunny, overcast, or rainy. Note that in the case where there are only two labels (in our example, it could be, for instance, predicting whether it will rain or not), we talk about *binary classification*:

$$f : \mathcal{X} \to \{0, 1\} : \mathbf{x} \mapsto y \tag{2.2}$$

where the positive label is represented by 1 (it is going to rain) and the negative label by 0 (it isn't going to rain).

In order to build such a model, we need data. Indeed, a machine learning algorithm assumes no prior knowledge about the task at hand, such as e.g. physical principles and modelling. Rather, it builds its model from the information that it is able to retrieve from the data. Therefore, the model is learned upon a set of labelled examples, that is examples for which the labels are already known. It is thus built by somehow using the information contained in the *training set*:

$$\{(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^N, y^N)\} \tag{2.3}$$

where each $(\mathbf{x}^p, y^p)$ is a labelled example, i.e. the feature vector of an observation along with its actual class label, which has been measured empirically. For the above example, we would have, for instance, the data of each day of the preceding year (temperature, pressure, etc.) in the $\mathbf{x}^p$ vectors, along with the type of weather that was observed (sunny, overcast or rainy) in the y^p outputs. Thanks to the information gathered from this training set, we should be able to build a model to predict the outcome for new, unseen, examples.

Multi-label supervised classification. Standard single-label classification assumes that the labels are mutually exclusive, and therefore only assigns one label to an input vector. This is no longer the case with multi-label classification [6, 7]. Such a model must be capable of assigning more than one label to a certain example. A simple example of a multi-label application is *document classification*, where we want to classify documents into specific categories. A multi-label framework is useful in this case as several labels could apply to one document. For example, an article titled "Parliament votes a bill on CO_2 regulation" could be assigned labels "Politics" and "Climate", rather than only one of them.

If we come back to our formal model, the output is now a set of labels $Y \subseteq \mathcal{Y}$, which is thus chosen among the power set of $\mathcal{Y}$, i.e. the set of all subsets of $\mathcal{Y}$, denoted by $2^{\mathcal{Y}}$. Hence, the learning function becomes [6]:

$$f : \mathcal{X} \to 2^{\mathcal{Y}} : \mathbf{x} \mapsto Y \tag{2.4}$$

which is trained on a set:

$$\{(\mathbf{x}^1, Y^1), (\mathbf{x}^2, Y^2), \ldots, (\mathbf{x}^N, Y^N)\} \tag{2.5}$$

It is clear, from this definition, that the number of possible outputs grows exponentially with the number of labels, which constitutes a key challenge when performing multi-label classification.

An often more convenient formulation consists in representing the output as a binary vector $\mathbf{y} \in \{0, 1\}^{|\mathcal{Y}|}$ with one corresponding element for each label. That is, element y_i indicates whether the ith label is relevant to the example or not. We thus end up with an equivalent learning function:

$$f : \mathcal{X} \to \{0, 1\}^{|\mathcal{Y}|} : \mathbf{x} \mapsto \mathbf{y} \tag{2.6}$$

with a corresponding training set:

$$\{(\mathbf{x}^1, \mathbf{y}^1), (\mathbf{x}^2, \mathbf{y}^2), \ldots, (\mathbf{x}^N, \mathbf{y}^N)\} \tag{2.7}$$

This formulation will be adopted throughout the document.

Standard classification methods only work properly for mutually exclusive labels. Other methods must therefore be developed to predict a set of non-exclusive labels. As stated before, the difficulty of developing such methods resides in the fact that there is an exponential number of possible outcomes, in terms of the labels. Under such circumstances, it is difficult to model thoroughly the correlation between the different class labels. This constitutes a big additional challenge that does not exist with standard classification, and means that we will often need to find some sort of compromise.

2.2 Baseline classifiers

In this section, we will describe some existing techniques to tackle multi-label classification problems. We focus on *problem transformation* methods, which decompose the multi-label problem in multiple single-label classification problems, and then somehow combine the results in order to get a multi-label output. Here, we are interested in methods transforming a problem into *binary* single-label problems. Two popular classifiers are *binary relevance* and *classifier chain*. Since this is also the spirit of the maximum entropy classifier that will be presented later, these methods will serve as a baseline in order to assess its performance. Note that there also exist adaptations of classic techniques, such as decision tress, k-nearest neighbours, etc., so that they can be used for multi-label classification. All these methods are covered in [6, 7].

2.2.1 Binary relevance

The most straightforward technique is *binary relevance* [6, 7]. This technique assumes independence between the class labels. Consequently, it completely ignores possible correlations that may exist between labels. It operates by considering in turn each of the labels independently, and making a prediction regarding this label alone. For each of the $C \triangleq |\mathcal{Y}|$ labels, we train a single-label binary classifier f in order to predict whether the label is relevant to the corresponding example or not. The results of the C individual classifiers will then simply be gathered in the predicted output vector, denoted by $\hat{\mathbf{y}}$.

Formally, for each label $y_i \in \mathcal{Y}$, we construct a binary training set, where we only consider this label as output, and ignore all the others:

$$\left\{(\mathbf{x}^1, y_i^1)\,,\ (\mathbf{x}^2, y_i^2)\,,\ \ldots\,,\ (\mathbf{x}^N, y_i^N)\right\} \tag{2.8}$$

We have thus a one-versus-rest approach. Then, we build a regular binary classifier:

$$f_i : \mathcal{X} \to \{0, 1\} \tag{2.9}$$

trained on the above training set. It will only predict the relevance of label y_i to example $\mathbf{x}$, i.e. if y_i should be 1 in the multi-label output $\hat{\mathbf{y}}$ corresponding to example $\mathbf{x}$, regardless of the values of the other labels.

We train such a classifier for each label, and then use them to get independent predictions of each label for the feature vector $\mathbf{x}$. Once we have these results, we put them into the output vector without any other manipulation. The output $\hat{\mathbf{y}}$ is therefore nothing else than a vector containing the results of the independent classifiers $f_1(\mathbf{x}), \ldots, f_C(\mathbf{x})$. The learning function is thus:

$$f : \mathcal{X} \to \{0, 1\}^C : \mathbf{x} \mapsto \Big(f_1(\mathbf{x})\,,\ f_2(\mathbf{x})\,,\ \ldots\,,\ f_C(\mathbf{x})\Big) \tag{2.10}$$

This method is computationally cheap, as it avoids the exponential complexity of the multi-label problem by considering that each label is independent. This is a very gross assumption, which is in general not true. Nevertheless, this technique is quite widely used for its simplicity, and is often able to give reasonably good results. However, as it does not exploit in any way the possible correlations between class labels, we might be missing a potentially significant amount of information that could improve the result of the classification.

2.2.2 Classifier chain

In order to account for the correlation between classes, the *classifier chain* was introduced in [8] as an improvement of binary relevance. It uses the same technique, i.e. building a binary classifier for each label, but puts them in a chain. This means that we plug the results of the previous classifiers into the input set of features given to the next classifier in the chain [6–8]. We will therefore perform a prediction for label y_i not only by considering the feature vector $\mathbf{x}$, but also by looking at the values of the previous labels y_1 to y_{i-1}. We will thus exploit any correlation that might exist between a label and all the preceding ones in the chain.

In practice, we begin by building a classifier f_1 to predict label y_1 of the example, just as before. Once f_1 has predicted a certain value $\hat{y}_1$ for the first label, we add this $\hat{y}_1$ as an extra feature to the feature vector $\mathbf{x}$ that will be given to the f_2 classifier, to predict the second label. The input vector given to the f_2 classifier is therefore:

$$(x_1\,,\ x_2\,,\ \ldots\,,\ x_D\,,\ \hat{y}_1) \triangleq \langle \mathbf{x}\,,\ \hat{y}_1 \rangle$$

where $\langle \cdot \rangle$ is a notation used here to signify that we extend a vector with one or more elements. For each subsequent classifier, we will add the result of the preceding classifier to the feature vector. Classifier f_3 will thus predict a value for the third label based on the features and the two first labels, etc.

Formally, we construct a training set for each label $y_i \in \mathcal{Y}$ by extending the feature vector with the binary values of the labels preceding y_i in the chain:

$$\left\{ \left(\langle \mathbf{x}^1,\, y_1^1,\, \ldots,\, y_{i-1}^1 \rangle\,,\ y_i^1 \right),\ \ldots\,,\ \left(\langle \mathbf{x}^N,\, y_1^N,\, \ldots,\, y_{i-1}^N \rangle\,,\ y_i^N \right) \right\} \tag{2.11}$$

Again, we use a binary classifier to predict the relevance of class y_i, which this time, since we extended the feature vector, is of the form:

$$f_i : \mathcal{X} \cup \{0,1\}^{i-1} \to \{0,1\} \tag{2.12}$$

As before, we put the successive results of the classifiers into the output vector, in order to obtain the following learning function:

$$f : \mathcal{X} \to \{0,1\}^C : \mathbf{x} \mapsto \Big(f_1(\mathbf{x})\,,\ f_2(\langle \mathbf{x}, y_1 \rangle)\,,\ \ldots\,,\ f_C(\langle \mathbf{x}, y_1, \ldots, y_{C-1} \rangle) \Big) \tag{2.13}$$

We thus end up with a vector that potentially contains more information than what we have with binary relevance, but with a limited additional cost, as we only extended the input set of the classifiers. Note that the results are in general dependent on the ordering of the labels. The only drawback of this method, compared to binary relevance, is that the number of features might increase significantly for classifiers at the end of the chain, if there are a lot of labels. This might lead to some problems related to the curse of dimensionality. However, if needs be, it should be possible to perform some feature selection at each point of the chain, and maybe add only a fraction of the labels in the feature vector.

While this method often constitutes an improvement over binary relevance, all the relations among class labels are not necessarily accounted for. Indeed, the chained nature of the classifier implies there is an imbalance in terms of information between all the classifiers. For the first

label, we must make a prediction without knowing anything about the other labels, so we have no information about relations among classes. The next classifier has some more information, namely the values of the first label, but it only regards relations between this label and the preceding one, and so forth. On the contrary, classifiers at the end of the chain will have all the values of the preceding labels at their disposal, and will therefore be able to access almost full knowledge about relationships between labels.

Therefore, the goal of the maximum entropy method is to have a model where all the relationships between classes are properly taken into account. In this case, no label will be predicted individually; the classifier will evaluate all the possible output vectors as a whole. This will of course mean that the computational complexity of this method will be much higher than that of the previous models.

2.3 Metric functions

An important question in machine learning is the evaluation of the performance of the classifiers. Already in the single-label framework, the choice of metric functions to assess this performance is important. However, when it comes to multi-label classification, this choice is even more complicated because of the nature of the output. Indeed, one must evaluate an output composed of several values, which can be combined in different ways, and thus yield different results. For example, there are multiple ways to compute the F_1 score, which is not the case for standard classification. The metrics that will be used in the experimental part of this work are presented in this sections, which is inspired from [6, 7].

2.3.1 Accuracy score

The first metric function that comes to mind when evaluating a classifier is the *accuracy*. It is often used in the single-label framework as it is simple, and yet often meaningful when we are in presence of a more or less well balanced set, i.e. there is more or less the same amount of examples of each class. It consists in checking whether or not the predicted output of the classifier is the same as the actual output of the example. It thus returns the ratio of correctly classified examples. For single-label classification, we can express the accuracy as:

$$accur = \frac{1}{N}\sum_{p=1}^{N}[\![\hat{y}^p = y^p]\!] \tag{2.14}$$

where the binary function $[\![\cdot]\!]$ returns the truth value of the contained predicate.

This function can be applied as is to the multi-label case, where it is referred to as *subset accuracy* [6, 7]:

$$accur = \frac{1}{N}\sum_{p=1}^{N}[\![\hat{\mathbf{y}}^p = \mathbf{y}^p]\!] \tag{2.15}$$

However, this scoring function is often poorly suited to the multi-label case, as it will assign 1 to an example only if all the labels are correctly predicted. If, for example, only one label

is wrong, it will receive a score of 0, which is often seen as too strict. If a classifier is able to correctly assign a great fraction of the labels, but not all of them, it should still receive some credit for it. Therefore, other metric functions are absolutely necessary in the multi-label setting.

2.3.2 F_1 score

The F_1 score was introduced in the field of information retrieval [9,10]. In the single-label classification context, it is used as a more robust alternative to accuracy. Indeed, the latter doesn't make any distinction between type I errors, i.e. wrongly predicting the positive class, and type II errors, i.e. wrongly predicting the negative class. This can be significant, for example, in the case of unbalanced sets, where there are only a few examples for a certain label. For instance, if a dataset contains only 5 % of positive labels, it is trivial to get an accuracy of 95 % by building a dummy classifier that always assigns the negative label, which is completely useless.

Therefore, instead of simply comparing the outputs of the examples, we can compute the precision and recall scores of a binary classifier, which are two measures of effectiveness [9,10]. These scores can be expressed in terms of:

- *true positive*, TP, i.e. the percentage of examples that were correctly labelled as positive;
- *false positive*, FP, i.e. the percentage of examples that were wrongly labelled as positive;
- *true negative*, TN, i.e. the percentage of examples that were correctly labelled as negative;
- *false negative*, FN, i.e. the percentage of examples that were wrongly labelled as negative.

The *precision* score is the number of examples that were correctly labelled positive, among all the examples that were predicted as positive:

$$precision(TP, FP) = \frac{TP}{TP + FP} \tag{2.16}$$

The *recall* score is the number of correctly labelled positive examples among all the examples that were actually positive:

$$recall(TP, FN) = \frac{TP}{TP + FN} \tag{2.17}$$

We can combine these scores into a single measure of performance, called the F score, which is the weighted harmonic mean of precision and recall [9]:

$$F_\beta = \frac{(\beta^2 + 1) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{2.18}$$

The β parameter controls the balance between precision and recall. Indeed, depending on the application at hand, one might be more important than the other. For most applications, however, $\beta = 1$ is used, which gives the same importance to both measures. This metric is called the F_1 score:

$$F_1(TP, FP, FN) = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{2.19}$$

This function is often used in single-label classification as an alternative to the accuracy score, but is not directly transposable to the multi-label case. Indeed there are several ways of combining the information contained in the outputs, which yield different metrics. First of all, we must distinguish two categories of metrics [6, 7]:

- *Label-based metrics* evaluate the performance independently for each label over the entire test set, and then average the results across all labels.
- *Example-based metrics* evaluate the performance of all the labels on each example separately, and then return the mean result for the whole test set.

Note that the subset accuracy score defined in the previous section is thus an example-based metric.

Label-based metrics. Let's start with the label-based metrics. Since we work on each label separately, and thus on all the examples together, we can reuse our definition in equation 2.19 by computing TP, FP, TN and FN for each label. They are defined as follows for each y_i:

$$TP_i = \left| \left\{ \mathbf{x}^p \;\middle|\; (\hat{y}_i^p = 1) \wedge (y_i^p = 1), 1 \leq p \leq N \right\} \right| \tag{2.20}$$

$$FP_i = \left| \left\{ \mathbf{x}^p \;\middle|\; (\hat{y}_i^p = 1) \wedge (y_i^p = 0), 1 \leq p \leq N \right\} \right| \tag{2.21}$$

$$TN_i = \left| \left\{ \mathbf{x}^p \;\middle|\; (\hat{y}_i^p = 0) \wedge (y_i^p = 0), 1 \leq p \leq N \right\} \right| \tag{2.22}$$

$$FN_i = \left| \left\{ \mathbf{x}^p \;\middle|\; (\hat{y}_i^p = 0) \wedge (y_i^p = 1), 1 \leq p \leq N \right\} \right| \tag{2.23}$$

Recall that y designates the actual value of the outcome, which has been measured empirically, whilst $\hat{y}$ is the value predicted by the classifier.

Once we have these scores for each individual class, we need to average them among all the classes. In order to obtain this, there exists two types of averaging that can be used [6, 7]:

- *Micro-averaging*, which counts the total TP, FP, TN and FN over all labels before computing the F_1 score. By using the definition of the F_1 score in equation 2.19, we obtain:
$$F_1^{\text{micro}} = F_1\left(\sum_{i=1}^{C} TP_i \,,\; \sum_{i=1}^{C} FP_i \,,\; \sum_{i=1}^{C} FN_i \right) \tag{2.24}$$
- *Macro-averaging*, which computes the F_1 score for each label before averaging. Using equation 2.19, we have:
$$F_1^{\text{macro}} = \frac{1}{C} \sum_{i=1}^{C} F_1(TP_i, FP_i, FN_i) \tag{2.25}$$

One can also obtain a *weighted* metric by modifying the macro-averaged metric to add weights to each label depending on its number of occurrences in the dataset.

Example-based metrics. On the other hand, example-based metrics aim at evaluating the performance on each example separately, before averaging the results over the entire test set. Therefore, we need to redefine the precision and recall as follows, using the set representation of the outputs for simplicity [6, 7]:

$$\textit{precision} = \frac{1}{N}\sum_{p=1}^{N}\frac{|\hat{Y}^p \cap Y^p|}{|\hat{Y}^p|} \qquad \text{and} \qquad \textit{recall} = \frac{1}{N}\sum_{p=1}^{N}\frac{|\hat{Y}^p \cap Y^p|}{|Y^p|} \tag{2.26}$$

This means that, for each example p, we divide the number of labels that were correctly classified as positive by either the number of predicted positive labels, to obtain the *precision* score, or the number of actual positive labels, to obtain the *recall* score. Once we have these quantities, we just need to plug them into equation 2.19 to get the F_1^{exam} metric.

As we can see, there are a lot of ways to compute metric scores in the multi-label case. Depending on the type of averaging, we can get different results as some classifiers could tend to optimise one metric at the expense of others [6]. When evaluating multi-label classifiers, we must therefore pay attention to what we want to evaluate for each specific application.

Chapter 3

The maximum entropy method

The principle of our application of the maximum entropy method to multi-label classification is, much like the classifiers detailed before, to combine the information of single-label classifiers. However, we will this time use probability estimates that are generated by the classifier, and combine this information into a joint probability distribution, by maximising the entropy. The goal of this procedure is to capture as much as possible the correlations that might exist between class labels.

First of all, we will recapitulate the theory of the maximum entropy method, introduced by Jaynes in [11, 12]. This subject is treated in depth in [13]. After this theoretical recap, we will see how we can formulate multi-label classification as a maximum entropy problem. We will also have a quick look at an alternative formulation, proposed in [14], that is limited to datasets with binary features. Finally, we will try to limit the complexity of the optimisation problem by developing some heuristics.

3.1 Entropy maximisation

3.1.1 Managing uncertainty

The maximum entropy principle was proposed as a method to deal with *probabilistic uncertainty* [13]. Suppose we have a situation with n possible outcomes, whose probabilities are:

$$p_1, p_2, \ldots, p_n \quad \text{where} \quad p_i \geq 0 \ \forall i \quad \text{and} \quad \sum_{i=1}^{n} p_i = 1 \tag{3.1}$$

In this situation, we have an huge uncertainty about which outcome will be realised, as we do not know the values of the probabilities yet. In order to reduce this uncertainty, we should add some information we might have about what we are trying to model. This can be achieved by adding one or more linear constraints of the form $g_1 p_1 + \cdots + g_n p_n = a$, where the g_i's and a are given constants that are relevant to the problem we are modelling. The constraints therefore represent information known in advance or deducted from empirical data.

In summary, if we have m linear constraints, we can reduce our uncertainty by adding the following set of constraints to our model:

$$\sum_{i=1}^{n} g_{ri} p_i = a_r \quad \text{for} \quad r = 1, \ldots, m \tag{3.2}$$

The more constraints we add, the more the uncertainty will drop, as the choice of distributions will be more and more restricted. At each stage, there may be an infinite number of distributions satisfying the constraints, but one of them will have maximum uncertainty, $H_{\max}$, and another minimum uncertainty, $H_{\min}$. Each time we add a constraint, $H_{\max}$ will decrease and $H_{\min}$ will increase, since the resulting set of probability distributions will be a subset of the set we had before adding the constraint [13].

However, even when the uncertainty has been reduced as much as possible, there might still be an infinity of probability distributions that are consistent with the constraints, out of which one has the maximum uncertainty $H_{\max}$. Since uncertainty can only be reduced by providing additional information, the use of any distribution other than $H_{\max}$ implies the use of some information that is not present in equation 3.2. Since we should avoid using any information that was not explicitly given, we should choose the distribution with maximum uncertainty $H_{\max}$ among all those satisfying the constraints. Hence, we seek to maximise uncertainty under constraints: we try to reduce uncertainty by adding information we know about in the form of constraints, but otherwise we should be *maximally uncertain about the things we do not know* [13]. This is the principle of *maximum entropy*.

Another way of looking at this is considering that if we use a distribution with less entropy than $H_{\max}$, we implicitly make an assumption on the distribution, and thus use some information at the expense of some other information. Inside the $H_{\max}$ distribution, on the contrary, none of this unknown information is used. It is thus the distribution which implicitly contains all the information that we do not know about, as we do not make any assumption on it. We can thus see this distribution as the one that contains the most "unbounded" information about the things we do not know.

Finally, we should note that the results of this method depend crucially on the measure of uncertainty that is used. There are different ways to measure this uncertainty, but the most common one is Shannon's entropy measure. It is this measure that has been used by Jaynes in his maximum entropy principle.

3.1.2 Shannon entropy

A quantification of uncertainty, most often called *entropy*, has been proposed by Shannon in [15]. In order to derive it, he first enunciated the different properties that his measure needed to have, and then searched for a function that would satisfy all of them. He stated that in order to have a measurement of uncertainty, the entropy must have the following properties [12, 13, 15]:

- H should depend on all the probabilities $p_1, \ldots, p_n$ and should be permutationally symmetric, i.e. H shouldn't change if $p_1, \ldots, p_n$ are permuted.

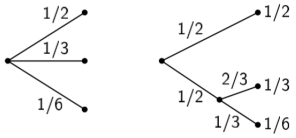
- $H(p_1, \ldots, p_n)$ should be a *continuous function* of $p_1, \ldots, p_n$. This ensures that the uncertainty only changes by small amounts for small changes of $p_1, \ldots, p_n$. Otherwise, an arbitrarily small change in the distribution could lead to a huge change of uncertainty, which would not make sense.

- $H(\frac{1}{n}, \ldots, \frac{1}{n})$ should be a *monotonically increasing* function of n. This property enforces the fact that when all events are equally likely, there is more uncertainty when the number of possible events we can chose from increases. Indeed, when there are two possible outcomes $p_1 = p_2 = \frac{1}{2}$, we are less uncertain about the outcome than in a situation with three possible outcomes $p_1 = p_2 = p_3 = \frac{1}{3}$.

- If there are different ways of computing the value of H, the result should always be the same. Specifically, if we decompose a choice into two successive choices, it should not influence the value of H. Therefore, the original H should be the *weighted sum* of the individual values of H:

$$H(p_1, \ldots, p_n) = H(p_1 + p_2, p_3, \ldots, p_n) + (p_1 + p_2) H\left(\frac{p_1}{p_1 + p_2}, \frac{p_2}{p_1 + p_2}\right) \tag{3.3}$$

  Let's illustrate this by an example, taken from [15]:



  In this situation, we have three possible outcomes, with probabilities $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{3}$ and $p_3 = \frac{1}{6}$. On the right, we decide to decompose the choice in first p_1 against $p_2 + p_3$, and then, in the latter case, p_2 against p_3. Since the final result is the same, we must ensure that the entropy of the first choice plus half the entropy of the second choice (as it only has a 0.5 probability of occurring) equals the original entropy. This is precisely what is contained in equation 3.3: $H(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}) = H(\frac{1}{2}, \frac{1}{2}) + \frac{1}{2} H(\frac{2}{3}, \frac{1}{2})$.

By following these four properties, Shannon arrived at the following measure [12, 13, 15]:

$$H(p_1, \ldots, p_n) = -k \sum_{i=1}^{n} p_i \log p_i \tag{3.4}$$

with k being an arbitrary positive constant, which can be ignored as it only amounts to a choice of unit. It can be proven that Shannon's entropy measure is the only function that satisfies this set of properties.

Along with the above properties, the entropy definition in 3.4 satisfies two very interesting properties from an optimisation point of view [13]. First of all, it is a *concave* function of $p_1, \ldots, p_n$. This means that when we find a local optimum, we can be sure that it is a maximum and that it is unique, i.e. it is the global maximum. If subject to linear constraints, an optimisation solver will always converge to this global maximum.

The second property is that when this measure is maximised under linear constraints, the *maximising probabilities p_i are all greater or equal than zero* [13]. Indeed, the domain of $p \log p$ is $[0, \infty[$ [1], so the objective function itself ensures that the probabilities cannot be negative. This is very useful, as we can satisfy the non-negativity constraints $p_i \geq 0$ of the probabilities automatically, without having to use complex optimisation techniques to enforce this.

3.1.3 Jayne's maximum entropy principle

Now, we can recapitulate everything and state the maximum entropy (MaxEnt) principle. It was proposed by Jaynes in [11], using Shannon's entropy as the measure of uncertainty. As discussed above, the model must enforce the following arguments [13]:

- Speak the truth and nothing but the truth;
- Make use of all the information that is given, and scrupulously avoid making assumptions about information that is not available.

There are, in general, an infinity of probability distributions that correspond to this description. The maximum entropy principle simply consists in choosing the distribution with maximum uncertainty, i.e. the one maximising Shannon's entropy.

Now, let's give the formal definition of Jaynes' MaxEnt principle [11–13]. For a random variable $\boldsymbol{x}$ with values $x_1, \ldots, x_n$ and corresponding probabilities $p_1, \ldots, p_n$, the MaxEnt method yields the following optimisation problem:

$$\max_{p_1,\ldots,p_n} H(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i \tag{3.5}$$

$$\sum_{i=1}^{n} p_i = 1 \tag{3.6}$$

$$\sum_{i=1}^{n} g_r(x_i)\, p_i = a_r \qquad \forall r \in \{1, \ldots, m\} \tag{3.7}$$

$$p_i \geq 0 \qquad \forall i \in \{1, \ldots, n\} \tag{3.8}$$

As stated before, the $p_i \geq 0$ inequalities are actually unnecessary since they will be automatically satisfied when maximising the objective function. We end up with a consistent probability distribution $p_1, \ldots, p_n$, and we can select the most likely outcome among $x_1, \ldots, x_n$ by choosing the one with the highest associated probability.

3.2 Application to multi-label classification

Now that we have seen the theoretical framework of the maximum entropy principle, we will apply it to the multi-label classification task. This formulation will be inspired by the methodology used in [16], where the MaxEnt principle was used to combine the outputs of different

[1] $p \log p$ is not defined in $p = 0$ because of the logarithmic function. However, $\lim_{p \to 0} p \log p = 0$, so we can define $0 \log 0 \triangleq 0$, which is necessary as it is possible to have a probability of zero for a certain outcome.

classifiers. Here, we will use binary classifiers to get estimates of the likelihood of labels and pairs of labels to be related to the example to be classified. The MaxEnt principle will then be used to blend these estimates in a meaningful probability distribution, from which we can obtain the most likely label vector for the example.

3.2.1 Optimisation problem

For the multi-label classification task, the goal is to assign a probability to each possible binary output vector $\mathbf{k} \in \{0,1\}^C$. These probabilities will represent the likelihood of the different binary vectors to be the actual label vector corresponding to the example being classified. They will thus depend on the values contained in the feature vector $\mathbf{x}$ of the example. At the end of the process, it is the label vector with the highest probability that will be chosen as the prediction output $\hat{\mathbf{y}}$.

Probability variables. Let $\boldsymbol{y}$ be the random variable representing the true, but unknown, label vector of the example to be classified. The probabilities that we seek are $P(\boldsymbol{y} = \mathbf{k}|\mathbf{x})$ for all $\mathbf{k} \in \{0,1\}^C$, i.e. the probability that the output vector $\mathbf{k}$ is the actual output of the example with feature vector $\mathbf{x}$. Since there are 2^C possible binary vectors of length C, we will have 2^C probability values to estimate. As these values represent a probability distribution, we must ensure that they sum to one, so the first constraint is:

$$\sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = 1 \tag{3.9}$$

The optimisation problem has thus variables that can be interpreted as probabilities of observing a set of labels.

Constraints. As said before, the probabilities are dependent on the feature vector of the example. We thus need a way to link the probability distribution to the information that will be extracted from the training set. In order to do this, we will estimate, from the training set, the first and second order statistics of the labels.

For the first order statistics, we need to estimate, for each label, the probability that the label is relevant to the example that is being classified. To do so, we use a binary classifier which gives the probability of the example to belong to the given class. Such a binary classifier has thus, alongside its decision function $f : \mathcal{X} \to \{0,1\}$, a function $g : \mathcal{X} \to [0,1]$ which gives the probability of an example to be of class 1. Once we have these probabilities, we can put constraints to ensure that the model complies with the label probabilities. Furthermore, in order to capture relations between labels, we use the same principle, but this time we estimate the probabilities of two classes appearing together.

Let's now see in detail how to model this. To satisfy the first order statistics, we build a classifier for each label $y_i \in \mathcal{Y}$. As for binary relevance, the training set of this classifier will be of the form:

$$\left\{(\mathbf{x}^1, y_i^1)\,,\ \ldots\,,\ (\mathbf{x}^N, y_i^N)\right\} \tag{3.10}$$

and we will use the function $g_i : \mathcal{X} \to [0,1]$ to estimate, for the new example that we are classifying, the probability that its ith label is 1. Once we have this probability, we must ensure

that the probability distribution is consistent with it, i.e. that the distribution's marginal fits the empirical marginal. For each class y_i, we thus need to enforce:

$$P(y_i = 1|\mathbf{x}) = g_i(\mathbf{x}) \tag{3.11}$$

Using the variables that we have defined for our optimisation problem, this constraint amounts to enforcing that all the probabilities corresponding to a vector where the ith class is 1 must sum to the empirical marginal. Hence, this constraint translates to:

$$\sum_{\substack{\mathbf{k}\in\{0,1\}^C \\ k_i=1}} P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = \sum_{\mathbf{k}\in\{0,1\}^C} k_i P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = g_i(\mathbf{x}) \tag{3.12}$$

Since there is one such constraint for each label, there will be C of them in total.

Now, we will model the 2-by-2 interactions between class labels. In order to comply with the second order statistics, we will build a classifier for each pair of labels. The positive class will correspond to the case where both labels are present, and the negative class to all other cases (we thus have a logical AND). For each pair y_i, y_j such that $i < j$ (in order to avoid duplicates), we build a training set:

$$\left\{(\mathbf{x}^1, y_i^1 \cdot y_j^1)\,,\ \ldots\,,\ (\mathbf{x}^N, y_i^N \cdot y_j^N)\right\} \tag{3.13}$$

and use the function $g_{ij} : \mathcal{X} \to [0, 1]$ to estimate the probability of an example of having both the ith and jth label. As before, we must ensure that both the distribution's and the empirical marginals are equal:

$$P(y_i = 1, y_j = 1|\mathbf{x}) = g_{ij}(\mathbf{x}) \tag{3.14}$$

This means that the set of probabilities corresponding to vectors with both the ith and jth class set to 1 must sum to the empirical marginal:

$$\sum_{\substack{\mathbf{k}\in\{0,1\}^C \\ k_i,k_j=1}} P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = \sum_{\mathbf{k}\in\{0,1\}^C} k_i k_j P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = g_{ij}(\mathbf{x}) \tag{3.15}$$

We need one such constraint for each pair of label. In total, there will thus be $\frac{C!}{2!(C-2)!} = \sum_{i=1}^{C-1} i \in \mathcal{O}(C^2)$ second order constraints.

Objective function. Once the model is coherent with the data at hand, i.e. it satisfies the above constraints, we must still ensure it is meaningful. In order to assign a probability mass to the set of labels, we use the maximum entropy principle that has been described earlier. The objective function is thus the entropy of the probability distribution:

$$H = -\sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y} = \mathbf{k}|\mathbf{x})\ \log P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) \tag{3.16}$$

This objective function needs to be maximised with respect to the different $P(\cdot)$ variables, in order to maximise the entropy of the probability distribution, and thus have a model with the most meaningful information possible.

Complete formulation. In the end, we thus have the following optimisation problem that needs to be solved to classify one example with feature vector $\mathbf{x}$:

$$\max_{\{P(.)\}} H = -\sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) \ \log P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) \tag{3.17}$$

$$\sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = 1 \tag{3.18}$$

$$\sum_{\mathbf{k}\in\{0,1\}^C} k_i P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = g_i(\mathbf{x}) \quad \forall i \in \{1,\dots,C\} \tag{3.19}$$

$$\sum_{\mathbf{k}\in\{0,1\}^C} k_i k_j P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = g_{ij}(\mathbf{x}) \quad \forall (i,j) \in \{1,\dots,C\}^2, \ i<j \tag{3.20}$$

Since the objective function is concave and the constraints are linear, the problem, if solvable, can only have one local optimum. We are therefore guaranteed to converge to a maximum which we know is global. Hence, we can use any known non-linear optimisation technique, such as gradient descent for instance, to find the optimal solution of this problem.

Indeed, in general, non-linear optimisation problems can have multiple local optima. Solvers might thus get stuck in a local optimum, without knowing if there are other, potentially better local optima. Solvers can for example try to go around this by performing several optimisations, with different initialisations, and then take the best result among the different runs. Even so, this constitutes an approximation as one cannot prove that the result is or is not the global optimum.

This is not the case with convex/concave optimisation, as the solvers will converge to the global optimum at once, regardless of the initialisation. Convex optimisation is thus simpler to solve, and there is a guarantee that the result is exact. For more information, see [17] as a reference for convex optimisation, and [18] for the particular case of entropy optimisation.

Now let's look at whether this problem is solvable. The total number of equality constraints is $1 + C + \sum_{i=1}^{C-1} i = \sum_{i=1}^{C} i + 1 \in \mathcal{O}(C^2)$, which is always inferior to 2^C, the number of variables. Since there are less linear equality constraints than there are dimensions, there must be a hyperplane that corresponds to the intersection of all these linear equations. The only possible exception would be if two constraints are parallel. However, to have two parallel constraints, a necessary condition is that both constraints involve exactly the same variables. This is never the case in the above optimisation problem. The latter is thus solvable in general.

Once we have the optimal solution to our problem, we simply need to pick, among all the $\mathbf{k}$ vectors, the one that has the largest probability of corresponding to the true output vector of the example. The decision function is thus:

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) \tag{3.21}$$

Analysis. In conclusion, we have a method that classifies new examples by relying on the maximum entropy principle to gather information delivered by binary classifiers. The training

part of the process is quite quick, as we only need to train the different binary classifiers over the training set. However, the prediction part of the process is a lot longer, since we need to solve one optimisation problem for each example to be classified.

Moreover, the prediction part will be quite slow because this optimisation problem has an *exponential complexity*. This means that when the number of labels grows, the memory resources that are needed and the time to solve the optimisation problem will grow exponentially. Indeed, the problem has an exponential *spacial complexity* in terms of the number of labels, as there are 2^C variables to be stored in memory. Moreover, the *temporal complexity* is also exponential, since evaluating the objective function and the constraints requires to iterate over the 2^C variables. Due to this exponential complexity, the method is expected to work for relatively small values of C, but will not be scalable to problems with a lot of class labels.

3.2.2 Reformulating the problem using the Lagrangian

The exponential spatial complexity comes from the number of variables that are present in the above formulation of the optimisation problem. By using the Lagrangian,[2] we can substitute new variables and end up with a lower spacial complexity.

The Lagrangian function of the optimisation problem is given by:

$$\begin{aligned}\mathcal{L} = &- \sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y}=\mathbf{k}|\mathbf{x})\ \log P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) \\ &+ \lambda_0 \left(\sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) - 1 \right) \\ &+ \sum_{i=1}^{C} \lambda_i \left(\sum_{\mathbf{k}\in\{0,1\}^C} k_i P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) - g_i(\mathbf{x}) \right) \\ &+ \sum_{\substack{i,j=1 \\ i<j}}^{C} \lambda_{ij} \left(\sum_{\mathbf{k}\in\{0,1\}^C} k_i k_j P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) - g_{ij}(\mathbf{x}) \right)\end{aligned} \tag{3.22}$$

where the different λs are the Lagrange multipliers.

The optimal solution is located at the point where the derivatives of the Lagrangian with respect to the different variables are equal to zero. Let us find this expression for a certain binary vector $\mathbf{k} \in \{0,1\}^C$. The derivative of the Lagrangian with respects to $P(\boldsymbol{y}=\mathbf{k}|\mathbf{x})$ must be equal to zero 0:

$$\frac{\partial \mathcal{L}}{\partial P(\boldsymbol{y}=\mathbf{k}|\mathbf{x})} = -\log P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) - 1 + \lambda_0 + \sum_{i=1}^{C} \lambda_i k_i + \sum_{\substack{i,j=1 \\ i<j}}^{C} \lambda_{ij} k_i k_j = 0 \tag{3.23}$$

[2] See for example appendix E of [1] for more information

This expression will be the same for any of the possible binary vectors $\mathbf{k} \in \{0,1\}^C$. From the above formula, we can thus retrieve an expression for each probability $P(\boldsymbol{y} = \mathbf{k}|\mathbf{x})$:

$$P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = e^{\lambda_0 - 1} \cdot \prod_{i=1}^{C} e^{\lambda_i k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} e^{\lambda_{ij} k_i k_j} \tag{3.24}$$

This expression only depends on the value of the Lagrange multipliers. In order to simplify this expression, let $G_0 = e^{\lambda_0 - 1}$, $G_i = e^{\lambda_i}$ and $G_{ij} = e^{\lambda_{ij}}$. We can rewrite the above formula:

$$P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = G_0 \cdot \prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \tag{3.25}$$

This expression can be plugged into the original optimisation problem, which leads to a substitution of variables. The new formulation becomes:

$$\max_{\{G\}} - \sum_{\mathbf{k} \in \{0,1\}^C} \left(G_0 \cdot \prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) \log \left(G_0 \cdot \prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) \tag{3.26}$$

$$\sum_{\mathbf{k} \in \{0,1\}^C} \left(G_0 \cdot \prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) = 1 \tag{3.27}$$

$$\sum_{\mathbf{k} \in \{0,1\}^C} k_\alpha \left(G_0 \cdot \prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) = g_\alpha(\mathbf{x}) \quad \forall \alpha \in \{1, \ldots, C\} \tag{3.28}$$

$$\sum_{\mathbf{k} \in \{0,1\}^C} k_\alpha k_\beta \left(G_0 \cdot \prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) = g_{\alpha\beta}(\mathbf{x}) \quad \forall (\alpha, \beta) \in \{1, \ldots, C\}^2,\ \alpha < \beta \tag{3.29}$$

In this formulation, we have the same number of variables as there are constraints, i.e. $\sum_{i=1}^{C} i + 1 \in \mathcal{O}(C^2)$. This means that the problem is still solvable, as in general, if there are not more constraints than there are dimensions, all the constraints have a common intersection.

Moreover, the *spatial complexity* has now become polynomial, as the number of variables is now proportional to C^2, and no longer to 2^C. This constitutes a great difference with the previous formulation. However, when evaluating the objective function and the constraints, we still have to iterate over the 2^C possible binary vectors $\mathbf{k}$. Therefore, the *temporal complexity* is still exponential with respect to the number of labels. In section 3.3, we will try to reduce this temporal complexity.

3.2.3 Iterative scaling procedure

As said before, the optimisation problem can be solved using different non-linear optimisation solvers, for instance gradient descent algorithms. Besides this, the solving of the Lagrangian yields another way of finding the optimal values, using an *iterative scaling procedure*. Such a procedure, proposed in [19], will iterate over the constraints in order to satisfy each of them in turn. This procedure is repeated until the values converge, i.e. there is no significant change in values between two iterations.

In order to obtain the different expressions that will be evaluated during the iterative scaling, we start from the constraints 3.27, 3.28 and 3.29 of the optimisation problem. From this set of constraints, we will isolate the different G parameters in order to have an expression for each of them.

We can find an expression for G_0 by isolating it from the probability distribution constraint (equation 3.27). Since G_0 doesn't depend on the value of $\mathbf{k}$, we can take it out of the sum so that we obtain the following expression:

$$G_0 = \left(\sum_{\mathbf{k} \in \{0,1\}^C} \left(\prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) \right)^{-1} \tag{3.30}$$

Next, the different G_α values, one for each class label, need to be estimated. We can get the expressions for these parameters by isolating each of them in their corresponding first-order constraint (equation 3.28). In the sum, only the terms where the $\mathbf{k}$ vector has $k_\alpha = 1$ will contribute to the total. All the other terms will add zero to the sum. Therefore, we can disregard k_α in the sum, which will implicitly set it to one. Consequently, G_α is now independent from $\mathbf{k}$ and can be taken out of the sum. In the end, we have for each $\alpha \in \{1, \ldots, C\}$:

$$G_\alpha = g_\alpha(\mathbf{x}) \left(G_0 \sum_{\mathbf{k} \backslash k_\alpha \in \{0,1\}^{C-1}} \left(\prod_{\substack{i=1 \\ i \neq \alpha}}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) \right)^{-1} \tag{3.31}$$

Finally, the expressions for the different $G_{\alpha\beta}$ parameters will be obtained by considering the second-order constraints (equation 3.29). Just as we did above, we can isolate $G_{\alpha\beta}$ from the sum by implicitly setting k_α and k_β to one. For $(\alpha, \beta) \in \{1, \ldots, C\}^2$, we therefore have:

$$G_{\alpha\beta} = g_{\alpha\beta}(\mathbf{x}) \left(\sum_{\mathbf{k} \backslash \{k_\alpha, k_\beta\} \in \{0,1\}^{C-2}} \left(\prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j \\ (i,j) \neq (\alpha,\beta)}}^{C} {G_{ij}}^{k_i k_j} \right) \right)^{-1} \tag{3.32}$$

These equations are quite complicated. However, they can be simplified by observing that all the expressions for G_α have a common part, as do all the expressions for $G_{\alpha\beta}$. We therefore

define the following quantities:

$$H_0 = G_0 \sum_{\mathbf{k} \in \{0,1\}^C} \left(\prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) \tag{3.33}$$

$$H_\alpha = G_0 \sum_{\mathbf{k} \in \{0,1\}^C} k_\alpha \left(\prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) \tag{3.34}$$

$$H_{\alpha\beta} = G_0 \sum_{\mathbf{k} \in \{0,1\}^C} k_\alpha k_\beta \left(\prod_{i=1}^{C} {G_i}^{k_i} \cdot \prod_{\substack{i,j=1 \\ i<j}}^{C} {G_{ij}}^{k_i k_j} \right) \tag{3.35}$$

Using these expressions, we can retrieve the expressions for the different G parameters, which yields the following iterative scaling:

$$G_0 \leftarrow \frac{G_0}{H_0} \tag{3.36}$$

$$G_\alpha \leftarrow \frac{g_\alpha(\mathbf{x}) \cdot G_\alpha}{H_\alpha} \qquad \forall \alpha \in \{1, \dots, C\} \tag{3.37}$$

$$G_{\alpha\beta} \leftarrow \frac{g_{\alpha\beta}(\mathbf{x}) \cdot G_{\alpha\beta}}{H_{\alpha\beta}} \qquad \forall (\alpha, \beta) \in \{1, \dots, C\}^2,\ \alpha < \beta \tag{3.38}$$

Each of these expressions is computed in turn to update the values of the different G parameters sequentially. Note that this involves recomputing each time the value of the H present in the expressions. This procedure is repeated until the values of the different G parameters converge. It can be shown that, thanks to the convex nature of the optimisation problem, this iterative scaling procedure converges to a unique optimum.

While this procedure has often been used for solving maximum entropy problems, research suggests that other techniques are more appropriate. In [20], the author concluded that iterative scaling methods were slower than general optimisation solvers based on gradient descent. He found that the *limited-memory variable-metric* algorithm, proposed in [21], outperformed any other solver. In [22], the authors found that *coordinate descent* algorithms were also appropriate for maximum entropy problems. Note that the choice of optimisation solver only impacts the execution time, and not the result of the optimisation.

3.2.4 Extension to higher order statistics

In the model that has been developed, we limited ourselves to the second order statistics. However, it could be interesting to examine whether higher order statistics could improve the performance of the classifier. For the third order, we would need to add the following constraint for each triplet of classes y_i, y_j, y_l with $i < j < l$:

$$P(y_i = 1, y_j = 1, y_l = 1 | \mathbf{x}) = g_{ijl}(\mathbf{x}) \tag{3.39}$$

As before, the empirical marginal $g_{ijl}(\mathbf{x})$ is obtained by using a binary classifier where the positive outcome corresponds to the case where all three labels are present. The previous equation translates to:

$$\sum_{\mathbf{k}\in\{0,1\}^C} k_i k_j k_l P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = g_{ijl}(\mathbf{x}) \tag{3.40}$$

This would add another $\frac{C!}{3!(C-3)!} \in \mathcal{O}(C^3)$ constraints to the model. During the experimental part of this work, we will see whether or not these extra constraints will improve the classification result.

3.2.5 Alternative formulation for document classification

In [14], Zhu, Ji, Xu and Gong proposed an alternative way of using a maximum entropy method for multi-label classification, for the special case of document classification. The features used in their framework is the presence or absence of certain words inside a text document. Their method is therefore limited to applications where the features are binary, contrarily to the general method developed in this work. Their formulation will be briefly explained here for information.

Optimisation problem. Instead of accounting for the relationship between features and outputs through a binary classifier, the authors took advantage of the fact that the features are binary to explicitly put them in the optimisation problem. Indeed, they proposed to add a set of constraints that corresponds to the co-occurrence statistic of each feature-label pair. For each pair of label y_i and feature x_l, they propose to compute the empirical expectation $E_{\hat{P}}[y_i x_l]$, i.e. the fraction of examples where label y_i is observed when feature x_l is 1. Then, they ensure that the expectation of the model $E_P[y_i x_l]$ corresponds to the empirical expectation:

$$E_P[y_i x_l] = E_{\hat{P}}[y_i x_l] + \varphi_{il} \tag{3.41}$$

In this expression, φ_{il} is a smoothing parameter. Indeed, the empirical expectations are computed directly from the training set, so we need a smoothing parameter to add some inductive bias, in order to avoid overfitting.[3] Since we account for the relations between $\mathbf{x}$ and $\mathbf{y}$ with the previous type of constraint, the first and second order statistics of the labels are independent from the feature vector $\mathbf{x}$ in this formulation. We simply compute the expectation of the occurrence of labels and pairs of labels from the training set. We thus have, for all classes y_i and y_j with $i < j$:

$$E_P[y_i] = E_{\hat{P}}[y_i] + \eta_i \tag{3.42}$$

$$E_P[y_i y_j] = E_{\hat{P}}[y_i y_j] + \theta_{ij} \tag{3.43}$$

where η_i and θ_{ij} are also smoothing parameters.

[3] In our formulation, the inductive bias comes from the binary classifiers that are used to compute the empirical marginals.

By using the same formalism as above, we end up with the following optimisation problem:[4]

$$\min_{\{P(.)\}} \sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) \ \log P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) \tag{3.44}$$

$$\sum_{\mathbf{k}\in\{0,1\}^C} P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = 1 \tag{3.45}$$

$$\sum_{\mathbf{k}\in\{0,1\}^C} k_i P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = E_{\hat{P}}[y_i] + \eta_i \qquad \forall i \in \{1,\dots,C\} \tag{3.46}$$

$$\sum_{\mathbf{k}\in\{0,1\}^C} k_i k_j P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = E_{\hat{P}}[y_i y_j] + \theta_{ij} \qquad \forall (i,j) \in \{1,\dots,C\}^2, \ i<j \tag{3.47}$$

$$\sum_{\mathbf{k}\in\{0,1\}^C} k_i x_l P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = E_{\hat{P}}[y_i x_l] + \varphi_{il} \qquad \forall i \in \{1,\dots,C\}, l \in \{1,\dots,D\} \tag{3.48}$$

Again, we end up with a convex optimisation problem. However, this time the are $C \cdot D$ extra constraints, which brings the total to $\sum_{i=1}^{C} i + C \cdot D + 1$.

The fact that the number of constraints depends on the number of features (D) is a bad thing, as there usually are a lot of them. Since the MaxEnt method is designed for problems with a small number of classes, it might often be the case that the number of equality constraints get larger than the number of variables (even when having performed feature selection[5] which, moreover, is difficult in this case as the features represent words in a text; selecting only a few words would make little sense). The problem would then become unsolvable. Indeed, for linear equality constraints in a space of lower dimension than their number, there is in general no point at which they all intersect. Therefore one must really pay attention to the number of features that are used in order to make the method work.

Reformulation using the Lagrangian. In the paper [14], the authors used the Lagrangian in order to derive an alternative formulation. They end up with an expression for the different probability variables:

$$P(\boldsymbol{y}=\mathbf{k}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(\mathbf{k}^{\mathrm{T}}(\mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{R}\mathbf{k})) \tag{3.49}$$

with the partition function:

$$Z(\mathbf{x}) \triangleq \sum_{\mathbf{k}\in\{0,1\}^C} \exp(\mathbf{k}^{\mathrm{T}}(\mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{R}\mathbf{k})) \tag{3.50}$$

where the vector $\mathbf{b}$ of length C, the $C \times D$ matrix $\mathbf{W}$ and the $C \times C$ strict upper triangle matrix $\mathbf{R}$ are the Lagrangian multipliers associated to the different constraints. By simplifying

[4]Formulated as a minimisation problem, since $\max\{-x\} = \min\{x\}$.

[5]For example, for a problem with $C = 6$, we have 64 variables. Since $\sum_{i=1}^{C} i + 1 = 22$, we need $C \cdot D \leq 42$, which means that we can only use 7 features for classification. For $C = 4$, we may use maximum 1 feature! The problem is always unsolvable for $C \leq 3$.

the Lagrangian and ignoring the constants, they arrive at the following expression:

$$\begin{aligned}\mathcal{L}(\mathbf{b}, \mathbf{W}, \mathbf{R}) = E_{\hat{P}}\Big[- \mathbf{k}^{\mathrm{T}}(\mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{R}\mathbf{k}) + \log Z(\mathbf{x})\Big] \\ + \frac{\lambda_{\mathbf{b}}}{2C}\|\mathbf{b}\|_2^2 + \frac{\lambda_{\mathbf{W}}}{2C}\|\mathbf{W}\|_F^2 + \frac{\lambda_{\mathbf{R}}}{2C}\|\mathbf{R}\|_F^2\end{aligned} \tag{3.51}$$

where $\lambda_{\mathbf{b}}$, $\lambda_{\mathbf{W}}$ and $\lambda_{\mathbf{W}}$ are regularisation coefficients that must be specified by the users. This function can be minimised using optimisation techniques such as gradient descent. Once we have the optimal values of $\mathbf{b}$, $\mathbf{W}$ and $\mathbf{R}$, the predicted output for feature vector $\mathbf{x}$ is:

$$\hat{\mathbf{y}} = \underset{\mathbf{k}\in\{0,1\}^C}{\operatorname{argmax}} P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) = \underset{\mathbf{k}\in\{0,1\}^C}{\operatorname{argmax}} \mathbf{k}^{\mathrm{T}}(\mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{R}\mathbf{k}) \tag{3.52}$$

In the end, we have an optimisation problem without constraints, but where the number of variables is $\sum_{i=1}^{C} i + C \cdot D$, which is more than the other formulation, when using the Lagrangian multipliers as variables (see section 3.2.2). Moreover, as discussed earlier, $C \cdot D$ can quickly take very large values, due to the often big number of features, and may exceed 2^C. Note that the temporal complexity is still $\mathcal{O}(2^C)$, as the evaluation of the expectation $E_{\hat{P}}[\cdot]$ will require to sum over all the possible $\mathbf{k} \in \{0, 1\}^C$. In the next section, we will search for ways to decrease the temporal complexity of the MaxEnt method.

3.3 Limiting the size of the problem using heuristics

One of the big drawbacks of the MaxEnt approach to multi-label classification is the exponential complexity of the problem in terms of the number of class label, which means the problem is not scalable. In this section, we will use heuristics to limit the number of values that need to be considered, in order to hopefully decrease the temporal complexity.

3.3.1 Eliminating unlikely outputs

A simple approach could be to analyse the training data and to draw a histogram of the number of labels contained in the output. Indeed, it is often the case that whilst there are many possible labels, most of the examples have only a small number of labels attributed to them, say n. A simple solution would then be to assign a probability of 0 to all $\mathbf{k}$ vectors containing more than n labels. We then no longer need to consider them (as we defined $0 \log 0 \triangleq 0$), so we only need variables corresponding to outputs having n outputs or less, i.e.:

$$P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) \quad \forall \mathbf{k} \in \{0, 1\}^C \quad \text{s.t.} \quad \mathbf{e}^{\mathrm{T}}\mathbf{k} \leq n \tag{3.53}$$

where $\mathbf{e}$ is a vector of ones. When evaluating the objective function and the constraints, we would only need to iterate over these variables, instead of the entire 2^C ones.

Whilst this method might eliminate many variables, there might still be a significant amount of them left. A more clever heuristic would be to find a way to predict the number of labels that are present in the true output, before performing the optimisation. We can then use, in

the optimisation problem, only the variables corresponding to vectors that have exactly that number of labels. In order to perform this prediction, we can use statistical models that compute *count events*, i.e. the number of occurrences of a certain event. A well-known model, often used in econometrics, is *Poisson regression*. Such a model is much more adapted to discrete outputs than a regular regression model, which is designed for continuous outputs.

3.3.2 Poisson regression

The Poisson regression model is a statistic model for count events, described for example in [23–25]. In our context, the count variable $t \in \mathbb{N}$ is the number of labels present in the output. We thus have $t = \mathbf{e}^{\mathrm{T}}\mathbf{y}$, where $\mathbf{e}$ is a vector of ones. In the statistics vocabulary, t is called the *endogenous* variable, and the elements of $\mathbf{x}$ the *exogenous* variables. Note that in a Poisson regression, there is no upper bound on the value of t, whereas, in our problem, the upper bound on the number of labels in the output is known. The model is therefore not completely accurate, however, it works well in practice: in general, no values larger than C will be predicted.[6] Searching for other statistical models is left for a further work.

Let $\boldsymbol{t}$ be the random variable that represents the real, but unknown, number of labels that characterises an example. The *Poisson probability distribution*, i.e. the probability to observe a certain value t, is given by:

$$P(\boldsymbol{t} = t) = \frac{e^{-\lambda}\lambda^t}{t!} \tag{3.54}$$

As we can see, the Poisson distribution relies on only one parameter, λ, to which the mean, variance, and all other cumulants of $\boldsymbol{t}$ are equal [23].

In order to perform a Poisson regression, we assume that the t variable follows a Poisson distribution with parameter λ. This parameter is linked to the exogenous variable $\mathbf{x}$, and can be computed from the values contained in $\mathbf{x}$ using the following relation [24, 25]:

$$\lambda = \exp(\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}) = \exp \sum_{l=1}^{D} \beta_l x_l \tag{3.55}$$

where $\boldsymbol{\beta}$ is a vector of parameters that needs to be estimated. The use of the exponential function to link the λ parameter to the exogenous variables is motivated by the fact that λ must be positive. The exponential function guarantees this, hence avoiding us to put a constraint of the form $\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} \geq 0$.

In order to estimate the $\beta_1, \ldots, \beta_N$ parameters from the observations, we will use the *maximum likelihood estimation* method. This method finds the parameters that maximise the likelihood function, or equivalently the log-likelihood function[7], given the examples [24].

Let $\{(\mathbf{x}^1, t^1), \ldots, (\mathbf{x}^N, t^N)\}$ be our training set. In this model, we assume that the different observations are independent, and that the $\mathbf{x}^p$ variables are deterministic. The log-likelihood

[6] And if it where the case, we could simply replace all results higher than C by C.

[7] Since the logarithmic function is monotonically increasing, the maximum of $\log(f)$ will be located at the same point as for f.

of the model, using the Poisson likelihood given in equation 3.54, is given by:

$$\log L = \sum_{p=1}^{N} \log P(\boldsymbol{t} = t^p) \tag{3.56}$$

$$= -\sum_{p=1}^{N} \lambda^p + \sum_{p=1}^{N} t^p \log \lambda^p - \sum_{p=1}^{N} \log(t^p!) \tag{3.57}$$

By substituting the λ's using equation 3.55, we get:

$$\log L = -\sum_{p=1}^{N} \exp(\boldsymbol{\beta}^{\mathrm{T}} \mathbf{x}^p) + \sum_{p=1}^{N} t^p \boldsymbol{\beta}^{\mathrm{T}} \mathbf{x}^p - \sum_{p=1}^{N} \log(t^p!) \tag{3.58}$$

This function is concave in terms of $\boldsymbol{\beta}$, which means that it has a unique (global) maximum [24]. The maximum is thus located at the unique point where the derivative is zero:

$$\frac{\partial \log L}{\partial \boldsymbol{\beta}} = -\sum_{p=1}^{N} \mathbf{x}^p (\exp(\boldsymbol{\beta}^{\mathrm{T}} \mathbf{x}^p) - t^p) = 0 \tag{3.59}$$

If this optimal $\boldsymbol{\beta}$ exists, we can compute λ and find the maximum value of the Poisson distribution function (equation 3.54), which will predict the number of class labels associated to the example. Note that whilst the Poisson regression model has been designed for training data with discrete outputs, the prediction will return a continuous value. Indeed, the maximum of the distribution 3.54 could be anywhere along the t axis, thus not necessarily at an integer value of t. The optimal t will need to be rounded to obtain the number of labels.

3.3.3 MaxEnt with Poisson regression

In order to reduce the number of variables in our optimisation problem, we will start by predicting the number of classes in the outputs of the different examples. For this, we will train a Poisson regression model on the training set in order to predict a value for each new example. During the optimisation, we will only use the variables $P(\mathbf{y} = \mathbf{k}|\mathbf{x})$ where $\mathbf{k}$ has the predicted number of classes. This will implicitly set all the other probabilities to 0.

If the Poisson regression predicted that the output will contain n labels, the number of binary vectors of length C containing n ones will be:

$$\binom{C}{n} = \frac{C!}{n!(C-n)!} \tag{3.60}$$

This function has a bell-shaped form peaking at $n = \frac{C}{2}$ and with value 1 at the extremities ($n = 0$ and $n = C$). At each point of the bell, we are significantly lower than 2^C, so there is quite a significant drop of the number of variables.

Depending on the values of C and n, this drop could lead to a situation where the number of variables would be smaller than the number of equality constraints. This is a problem as it is extremely unlikely that all of these equality constraints intersect at one single point in a space of lower dimension. The problem would thus be unsolvable.

In order to avoid this, we will have to relax (some of) the equality constraints. One way of doing this is to assign slack variables to the constraints, which allows the equality constraints to not be perfectly met. The constraints will still be linear, so the addition of slack variables preserves the convexity of the problem [17]. These slack variables represent a cost in the objective function, such that the optimisation will minimize the values of the slack variables so as to have an as high as possible objective function. The optimisation problem thus becomes (in case we choose to relax all of the first and second order constraints):

$$\max_{\{P(.)\}} \; - \sum_{\substack{\mathbf{k}\in\{0,1\}^C \\ \mathbf{e}^{\mathrm{T}}\mathbf{k}=n}} P(\mathbf{y}=\mathbf{k}|\mathbf{x}) \; \log P(\mathbf{y}=\mathbf{k}|\mathbf{x}) - \sum_{i=1}^{C}(\xi_i^+ + \xi_i^-) - \sum_{\substack{i,j=1 \\ i<j}}^{C}(\xi_{ij}^+ + \xi_{ij}^-) \tag{3.61}$$

$$\sum_{\substack{\mathbf{k}\in\{0,1\}^C \\ \mathbf{e}^{\mathrm{T}}\mathbf{k}=n}} P(\mathbf{y}=\mathbf{k}|\mathbf{x}) = 1 \tag{3.62}$$

$$\sum_{\substack{\mathbf{k}\in\{0,1\}^C \\ \mathbf{e}^{\mathrm{T}}\mathbf{k}=n}} k_i P(\mathbf{y}=\mathbf{k}|\mathbf{x}) + \xi_i^+ - \xi_i^- = g_i(\mathbf{x}) \qquad \forall i \in \{1,\ldots,C\} \tag{3.63}$$

$$\sum_{\substack{\mathbf{k}\in\{0,1\}^C \\ \mathbf{e}^{\mathrm{T}}\mathbf{k}=n}} k_i k_j P(\mathbf{y}=\mathbf{k}|\mathbf{x}) + \xi_{ij}^+ - \xi_{ij}^- = g_{ij}(\mathbf{x}) \qquad \forall (i,j) \in \{1,\ldots,C\}^2,\ i<j \tag{3.64}$$

$$\xi_i^+, \xi_i^- \geq 0 \qquad \forall i \in \{1,\ldots,C\} \tag{3.65}$$

$$\xi_{ij}^+, x_{ij}^- \geq 0 \qquad \forall (i,j) \in \{1,\ldots,C\}^2,\ i<j \tag{3.66}$$

Note that not necessarily all constraints must contain slack variables. What we must only ensure is that the number of variables is at least as high as the number of constraints, i.e. $\sum_{i=1}^{C} i + 1$. For certain combinations of C and n, no slack variables will be required at all. For $C \leq 6$, the number of variables will always be inferior to the number of constraints, so some slack variables will be needed. For $C > 6$, this depends on the value of n.

Let's evaluate the *temporal complexity* of the problem. When evaluating the objective function and the constraints, we will have to iterate through the different $\mathbf{k}$ vectors. Their number is given by equation 3.60. The worst case occurs when $n = \frac{C}{2}$. Figure 3.1 illustrates the evolution of the number of variables in function of C. We can clearly see that the temporal complexity is still exponential in terms of the number of labels. While there is an important saving in terms of the number of variables for relatively small values of C, the problem still isn't scalable. The formulation with Poisson regression might thus be interesting for the problems that we were already able to solve, but it does not enable us to tackle new problems, namely those with a high number of labels.

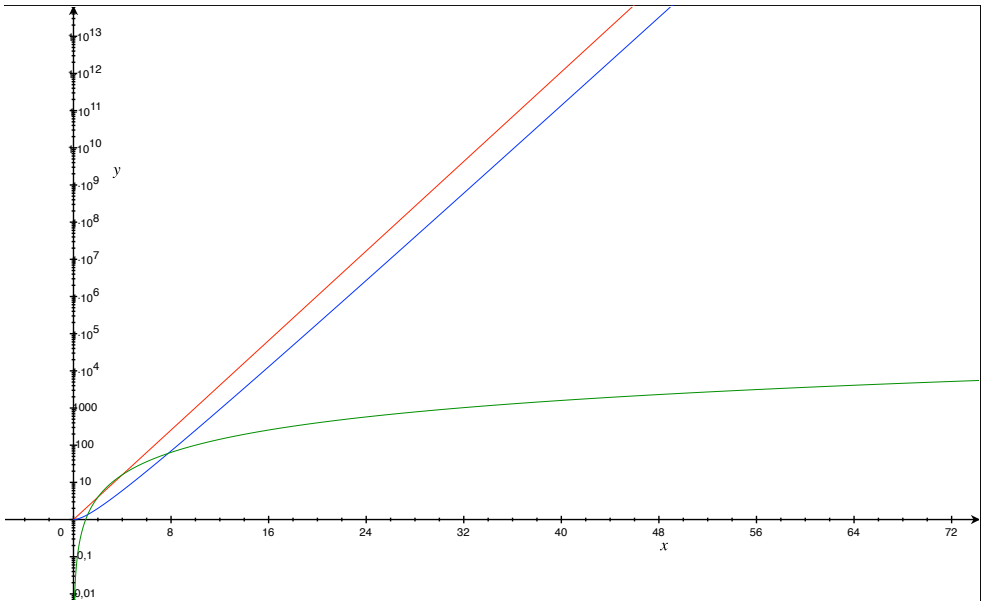


Figure 3.1: Worst-case complexity of the MaxEnt method with Poisson regression (in blue) on a lin-log scale (in red: 2^C, in green: C^2)

Chapter 4

Implementation and experimental methodology

In this chapter, we will expose the methodology that was used during the the experiments. First, we will have a quick look at how the maximum entropy method has been implemented in Python by explaining briefly the algorithm behind it. Then, we will take a look at the datasets that where used to compare the different classifiers. We will end by looking at the procedure that has been used to assess the performance of these classifiers.

4.1 Implementation

In order to test its performance, the MaxEnt multi-label classifier was implemented in Python, based on the theory that has been discussed earlier. To get an overview of what was described in the theory, we can take a look at the algorithms that describe how we can train the MaxEnt classifier, and how to predict the outputs of new examples. These algorithms will give an understanding of how the implementation in Python looks like. We will also describe the integration of the Poisson regression into the MaxEnt classifier.

4.1.1 MaxEnt classifier

Let's take a look at the implementation of the MaxEnt classifier. As said before, it consists in two parts. First, there is the training part, where the labelled examples are used to fit the model, in order to provide information about the data. For the MaxEnt classifier, this amounts to training the different binary classifiers that perform label and pair of labels predictions. This is followed by the prediction part, where the classifier will predict an outcome for each new, unlabelled, example. This is done by getting the probabilities from the different binary classifiers, and solving the optimisation problem.

Training the classifier. The training process is done in the *fit* procedure, described in algorithm 1. It takes three arguments:

- $\mathbf{X}$, the feature matrix of the training set, containing the feature vectors of the different examples $\mathbf{x}^1, \ldots, \mathbf{x}^N$ on its rows.
- $\mathbf{Y}$, the label matrix of the training set. This matrix contains the label vectors of the examples $\mathbf{y}^1, \ldots, \mathbf{y}^N$ on its rows. The columns thus correspond to the vectors containing the values of a particular label for each example. These vectors are denoted $\mathbf{y}_1, \ldots \mathbf{y}_C$. We have thus the following notation:

$$\mathbf{Y} = \begin{array}{c} \begin{matrix} \mathbf{y}_1 & \cdots & \mathbf{y}_C \\ \downarrow & & \downarrow \end{matrix} \\ \begin{pmatrix} y_1^1 & \cdots & y_C^1 \\ \vdots & \ddots & \vdots \\ y_1^N & \cdots & y_C^N \end{pmatrix} \end{array} \begin{array}{l} \leftarrow \mathbf{y}^1 \\ \\ \leftarrow \mathbf{y}^N \end{array} \tag{4.1}$$

- χ, a type of binary classifier that is able to predict the probability of an example to be of class 1. This classifier must thus have alongside its decision function $f : \mathcal{X} \to \{0, 1\}$ a probability function $g : \mathcal{X} \to [0, 1]$.

The training part simply amounts to training different instantiations of the χ classifier, in order to obtain the different probability functions $g_i(\mathbf{x})$ for each label and $g_{ij}(\mathbf{x})$ for each pair of labels. The classifiers that are used in the code come from Python's scikit-learn module.[1] It is a machine learning module that contains, among other things, the implementation of different classifiers.

First, the algorithm iterates over the different labels, and for each one creates an instance of the χ classifier. It then trains this instance by using the *fit* function of χ, with as arguments the data matrix $\mathbf{X}$, and the column vector $\mathbf{y}_i$ of the label matrix corresponding to the considered label. Once the training has been done, this classifier has a function $g_i(\mathbf{x})$ that can be used for predicting probability of an example to have label y_i.

Then, the algorithm iterates over each pair of labels, and creates for each one an instance of χ. This time, the second argument of χ's *fit* function is a vector indicating whether or not both labels appear together in the output. This corresponds to the element-wise product of the two corresponding label vectors: $\mathbf{y}_i \odot \mathbf{y}_j$. This will give a classifier with a $g_{ij}(\mathbf{x})$ function.

Predicting new examples. The prediction of new examples is done by the *predict* function, described in algorithm 2. It takes three arguments:

- $\mathbf{X}$, the feature matrix of the test set, containing the feature vectors of the different examples $\mathbf{x}^1, \ldots, \mathbf{x}^M$ on its rows.
- $\{g_\bullet\}$, the set of probability functions generated during the training process.
- ω, a non-linear optimisation solver.

[1] www.scikit-learn.org/stable/

Algorithm 1: *fit*

Input: A training set $\mathbf{X}$ whose rows contain the feature vectors $\mathbf{x}^1, \ldots, \mathbf{x}^N$

Input: The corresponding label matrix $\mathbf{Y}$ with rows $\mathbf{y}^1, \ldots, \mathbf{y}^N$ and columns $\mathbf{y}_1, \ldots, \mathbf{y}_C$

Input: A binary classifier χ whose *fit* function returns a probability function $g : \mathcal{X} \to [0, 1]$

Output: A set of functions $\{g_i \mid i = 1 \ldots C\}$ and $\{g_{ij} \mid i, j = 1 \ldots C, i < j\}$

for $i \leftarrow 1 \ldots C$ **do**
  $g_i \leftarrow \chi.\mathit{fit}(\mathbf{X}, \mathbf{y}_i)$

for $i \leftarrow 1 \ldots C$ **do**
  for $j \leftarrow i + 1 \ldots C$ **do**
    $g_{ij} \leftarrow \chi.\mathit{fit}(\mathbf{X}, \mathbf{y}_i \odot \mathbf{y}_j)$

return $\{g_\bullet\}$

Algorithm 2: *predict*

Input: A test set $\mathbf{X}$ whose rows contain the feature vectors $\mathbf{x}^1, \ldots, \mathbf{x}^M$

Input: A set of functions $\{g_i \mid i = 1 \ldots C\}$ and $\{g_{ij} \mid i, j = 1 \ldots C, i < j\}$

Input: A non-linear optimisation solver ω

Output: The predicted label matrix $\hat{\mathbf{Y}}$ with rows $\hat{\mathbf{y}}^1, \ldots, \hat{\mathbf{y}}^M$

for $p \leftarrow 1 \ldots M$ **do**

  Obj $\leftarrow$ objective function

  Ctrs $\leftarrow \emptyset$

  Add probability constraint in *Ctrs*

  for $i = 1 \ldots C$ **do**
    Add constraint involving $g_i(\mathbf{x}^p)$ in *Ctrs*

  for $i \leftarrow 1 \ldots C$ **do**
    for $j \leftarrow i + 1 \ldots C$ **do**
      Add constraint involving $g_{ij}(\mathbf{x}^p)$ in *Ctrs*

  $\left\{P(\boldsymbol{y} = \mathbf{k}|\mathbf{x}) \,\middle|\, \mathbf{k} \in \{0, 1\}^C\right\} \leftarrow \omega.\mathit{maximise}(\mathit{Obj}, \mathit{Ctrs})$

  $\hat{\mathbf{y}}^p \leftarrow \operatorname{argmax}_{\mathbf{k} \in \{0,1\}^C} P(\boldsymbol{y} = \mathbf{k}|\mathbf{x})$

return $\hat{\mathbf{Y}}$

The *predict* function will return the predicted label matrix $\hat{\mathbf{Y}}$, containing the predictions of each example on its rows. In order to get a prediction for the M examples contained in the test set, we need to perform the entropy maximisation for each example individually. The algorithm must thus iterate over the different examples, and fill the rows of $\hat{\mathbf{Y}}$ one by one.

The optimisation is performed by the optimize package of scipy.[2] This package contains a `minimize` function that will perform the optimisation. For this, we need to define functions that will allow the solver to evaluate the objective function and the constraints. The objective function will be represented by a Python function that will take the current values of the variables as arguments, and return the objective value. The three types of constraints are represented by three different functions that will return the difference between the left and right sides of the corresponding equation. The first (resp. second) order constraints take as extra arguments i and $g_i(\mathbf{x})$ (resp. i, j and $g_{ij}(\mathbf{x})$), so each constraint of the same type can be evaluated by the same function.

Finally, the algorithm uses the `minimize` function in order to get the optimal values of the probability variables. In order to optimise under constraints, the optimisation solver uses the Sequential Least SQuares Programming (SLSQP) method (see [26] for more information). At the end of the optimisation, the algorithm puts the $\mathbf{k}$ vector that corresponds to the highest probability in the row of $\hat{\mathbf{Y}}$ corresponding to the example that has been classified, and moves on to the next.

4.1.2 Poisson regression

In the second implementation, the algorithm uses the Poisson regression to determine the number of labels that each example is expected to have. In order to do that, the MaxEnt classifier must train a Poisson regression algorithm at the end of the *fit* procedure. For that, a Generalised Linear Model (GLM) algorithm, which we denote by γ, is used. GLM consists in a generalisation of linear regression, where a link function relates the linear predictor to the response variable [23]. The Poisson regression is thus a particular case of GLM, where the link function is the Poisson distribution. In the Pyhton code, the GLM package of statsmodel was used to perform the Poisson prediction.[3]

In order to train the predictor, the γ.*fit* function will take the feature matrix $\mathbf{X}$, and a vector containing the number of labels in the outputs of each example, i.e. $\mathbf{t} = \mathbf{Ye}$ (where $\mathbf{e}$ is a vector of ones). γ.*fit* will return a Poisson regression function $\pi : \mathcal{X} \to \mathbb{N}$, which predicts the number of labels that are associated to a feature vector. In the *predict* function, the algorithm will compute $\pi(\mathbf{x})$ before optimising, in order to remove all the variables that do not correspond to outputs with the predicted number of labels.

[2] `docs.scipy.org/doc/scipy/reference/optimize.html`
[3] `www.statsmodels.org/dev/glm.html`

4.2 Datasets

In order to test the MaxEnt method, we need to evaluate its performance on different datasets. Because of its exponential complexity, we can only use datasets with a small number of labels. Unfortunately there are very few such datasets available. Three such datasets have been found. To perform more experiments, two datasets with not too many labels have been chosen, and some labels were selected to be part of the classification problem whilst the others are dropped. Although this means some information is dropped, these datasets could still be useful.

4.2.1 Datasets description

emotions. This dataset was used in [27], a paper that aimed to model automated detection of emotions induced by music as a multi-label classification task. The data involves 593 songs, from which the authors extracted 72 features. These features fall into two categories: rhythmic features, derived by extracting periodic changes from a beat histogram, and timbre features, using speech recognition and music modelling techniques. The different songs were then labelled using 6 clusters of emotions, based on the Tellegen-Watson-Clark model. The labels are: amazed-surprised, happy-pleased, relaxing-calm, quiet-still, sad-lonely, and angry-fearful. A list of songs where labelled by three music experts, and those for which all three agreed on all the labels where kept.

scene. This dataset was used in a paper devoted to multi-label scene classification [28]. These semantic scenes are images of landscapes, which can contain multiple elements. For example, we could have a field scene with a mountain in the background. Therefore, these scenes can be described by multiple class labels.

flags. This is a toy dataset taken from [29]. It contains data about nations and their national flags. The multi-label classification task is to predict the different colors that compose the flags.

yeast. This dataset, taken from [30], contains information about a set of yeast cells. The task is to determine the localisation site of each cell. The dataset is formed by micro-array expression data and phylogenetic profiles of 2417 genes. Each of them is associated with a set of functional classes, which can have a potential size of more than 190. In order to simplify this, the authors used the fact that the set of functional classes is structured in a tree whose leaves are the functional categories. Given a gene, knowing which edge to take from one level to the next leads directly to a leaf, which is a functional class. With this dataset, the goal is to predict which edge to take from the root to the first level of the tree. There are 14 such edges, and several of them can be associated to an example, as one gene can have many functional classes.

medical. This last dataset comes from a paper investigating multi-label classification of clinical free text in medical reports [31]. In particular, it consisted in the assignment of ICD-9-CM codes to radiology reports. The authors collected a corpus from the Department of Radiology at Cincinnati Children's Hospital Medical Center. The different reports were disambiguated, anonymised, and manually checked to avoid the presence of any Protected Health Information.

	Nb. of examples	Nb. of features	Nb. of labels	Cardinality
emotions	593	72	6	1.869
scene	2407	294	6	1.074
flags	194	19	7	3.392
yeast	2417	103	14	4.237
medical	978	1449	45	1.245

Table 4.1: Some statistics about the datasets

The assignment of the ICD-9-CM codes is based on two components of the radiology report: the clinical history, provided by a physician before the radiology procedure, and the impression reported by a radiologist after the procedure. In order to assign the codes, a majority principle was used among the different annotators.

4.2.2 Datasets statistics

In order to have a bit more information about the different datasets, let's look at some basic statistics. Table 4.1 summarises some statistics about the datasets: the number of examples, the number of features, the number of labels, and the cardinality, which is the average number of labels present in the outputs.

As we can see, most datasets have a low cardinality with respect to their number of labels; the exception being the flags dataset. This is confirmed by looking at the bar plot of the size of the outputs, given in figure 4.1. This is good news for when we are going to limit the number of variables using Poisson regression: we will most likely avoid the worst case, where the predicted number of labels n is $\frac{C}{2}$. With a small value of n, there will be much less variables in the optimisation problem, so optimising it should be much faster.

Figure 4.2 shows the number of occurrences of each label in the dataset. Since the first three datasets have 6 or 7 labels, we can use the dataset directly with the MaxEnt method. However, for the yeast and the medical datasets, the number of labels is too high to obtain results in a reasonable time. We therefore have to select some labels, and drop the others, in order to reduce the number of labels. For the yeast dataset, the 5 most occurring labels are used. For the medical dataset, we observe that a few labels are occurring many time, and that the majority of them occur very few times. We can thus perform the classification for example on the 4 labels that occur more than 100 times in the dataset. To emphasise that limited versions of these datasets were used, we will denote them by yeast* and medical*.

4.3 Testing methodology

In this last section, we expose the testing methodology that will be used to asses the performance of the maximum entropy classifier. We will briefly recall the baseline classifiers and metric functions that are used. Then, the cross-validation procedure will be exposed.

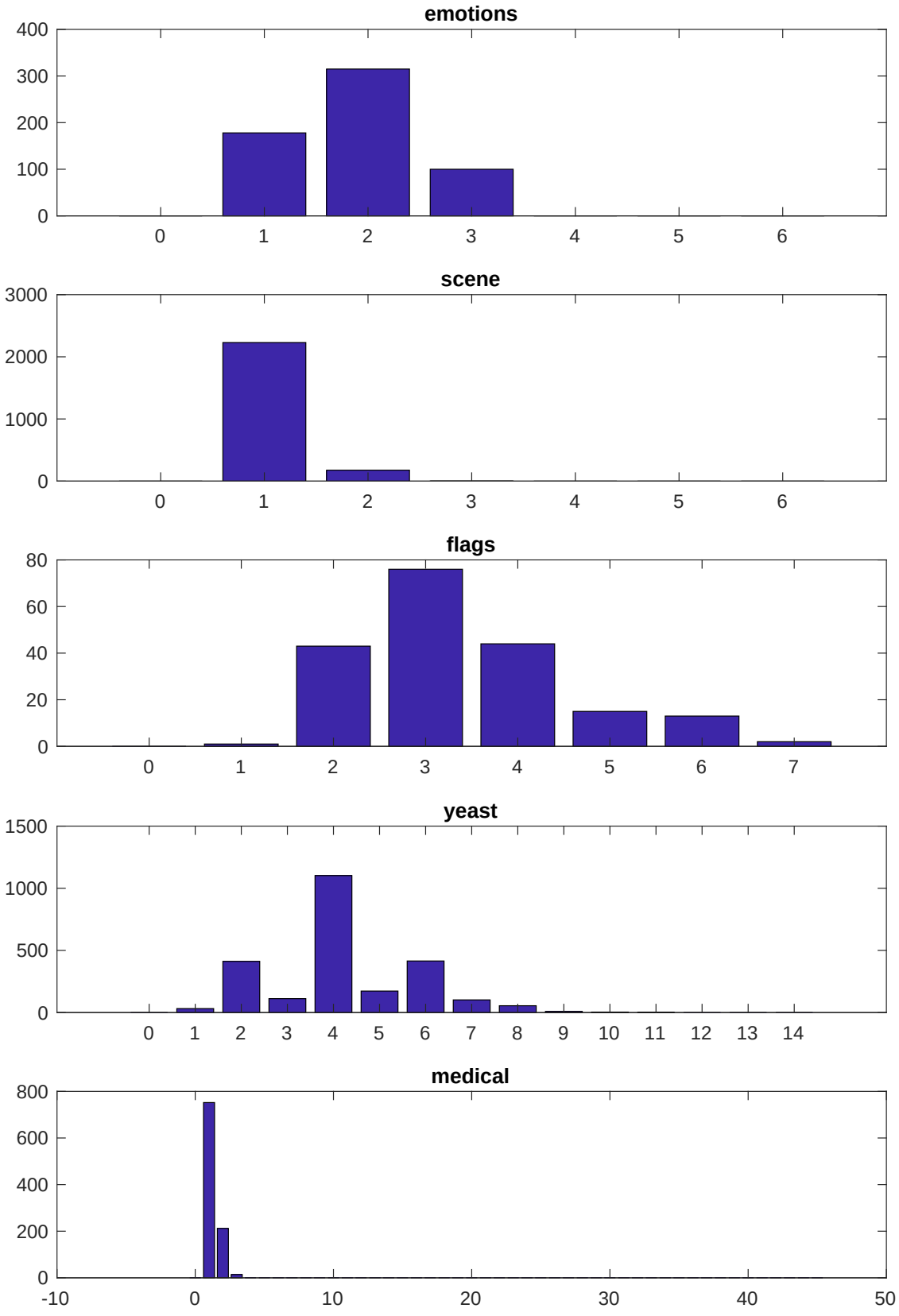


Figure 4.1: Number of outputs that have n labels

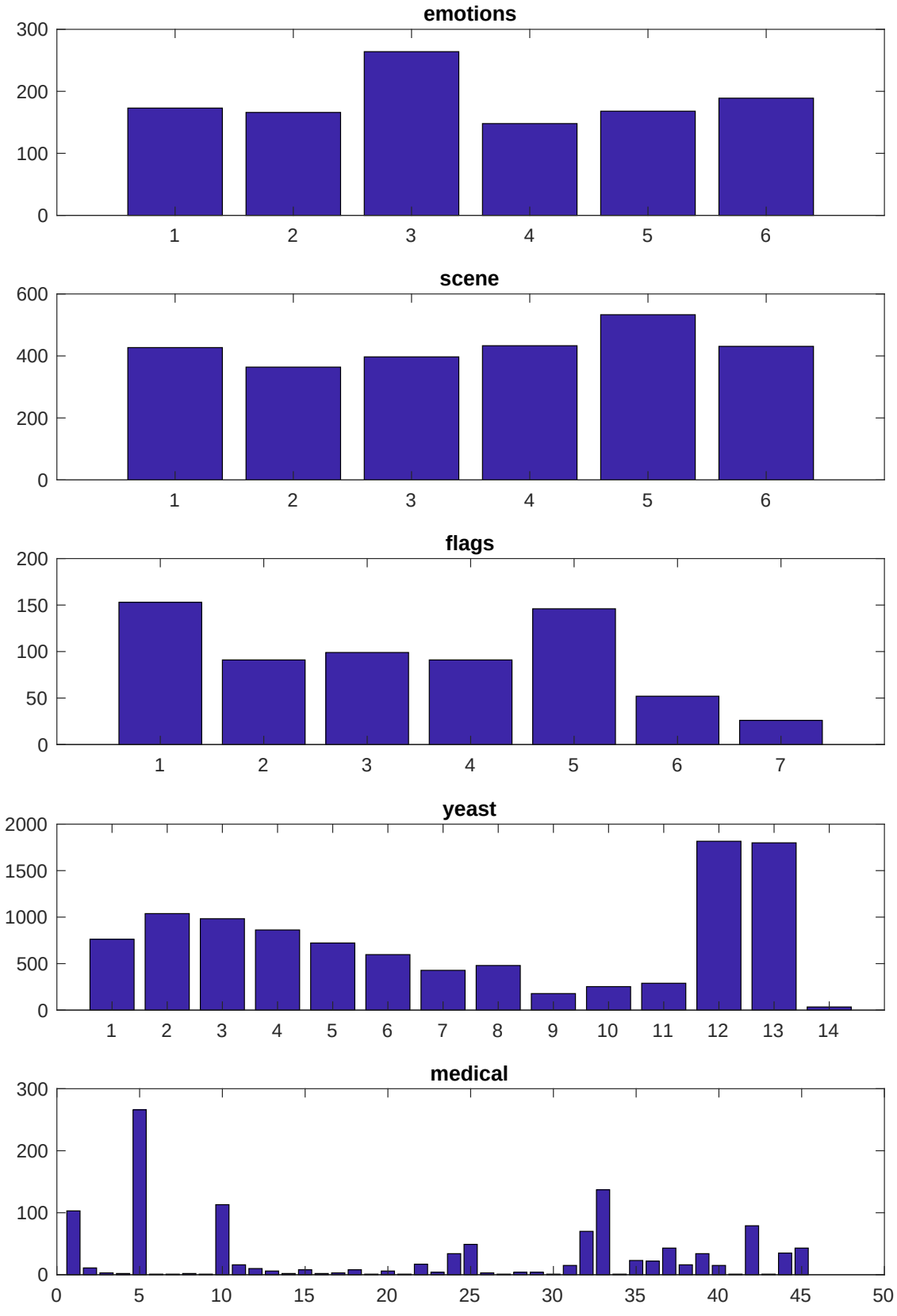


Figure 4.2: Number of occurrences of each label in the datasets

4.3.1 Classifiers

In order to evaluate the performance of the maximum entropy classifier, which we will refer to as MaxEnt, two other multi-label classifiers will be considered as baselines: the *binary relevance* (BinRel) classifier and the *classifier chain* (CChain). These two classifiers are relevant as they are problem transformation algorithms, i.e. they rely on the predictions of some binary classification algorithm, just like MaxEnt. Hence, we can assess whether MaxEnt is an improvement compared to these two existing problem transformation techniques.

The performance of these three multi-label classifier will thus be evaluated with the same internal classifier. This will ensure that the comparison is relevant, without having to tune hyper parameters. Moreover, these two classifiers are very popular in multi-label classification. This means that a good performance with respect to these two classifiers would validate the interest of MaxEnt. Note that there exist other multi-label classifiers, which were obtained using other methodologies than problem transformation, such as the adaptations of single-label classifiers. In case of a real-world multi-label application, these classifiers should also be considered during cross-validation to select the best model.

For the experiments, a *logistic regression* classifier was used to perform the binary predictions. This model is described in for example [1–3]. It does not model the output y directly, but rather models the probability that y belongs to a certain class: $P(y = 1|\mathbf{x})$. It then classifies the example based on this probability: if the probability for an example is higher than a certain threshold, generally 0.5, the example is assigned to class 1, otherwise to class 0. This model is thus perfect for this experiment, as it has a decision function $f : \mathcal{X} \to \{0, 1\}$ that can be used by BinRel and CChain, and a probability function $g : \mathcal{X} \to [0, 1]$ which is required by MaxEnt. The implementation of these classifiers in the scikit-learn package were used for the experiments.

4.3.2 Metric functions

For the performance assessment, the metrics defined in section 2.3 are used. Recall that there are two types of metrics:

- Example-based metrics, such as *accur* and F_1^{exam}, which compute scores for each example and then average over the whole dataset.
- Label-based metrics, including F_1^{micro}, F_1^{macro} and F_1^{weight}, which compute scores for each label over the whole dataset, and then average among the labels.

These metric functions are also implemented in the scikit-learn package.

4.3.3 Cross-validation

In order to asses the performance of the classifiers over the different datasets, a model selection protocol is needed. Indeed, if we only have one training and one test set, the results might be dependent on these sets. If we used different training and test sets, the results might be different. We thus need a more robust way to evaluate the performance.

Therefore, a 10-fold cross-validation protocol was used. The principle of this protocol is to divide the datasets in ten folds, and perform ten different classifications. Each fold is successively used as test set, and the other nine are grouped together to constitute the training set. This gives the advantage that each example is used exactly once for testing.

The results for each test fold are kept, and then averaged to get the average performance of the classifiers. Since we have 10 different classifications, taking the average will decrease the randomness of the result that we could observe if we took only one test set. The performance is evaluated using the 5 metrics described in section 2.3, namely *accur*, F_1^{exam}, F_1^{micro}, F_1^{macro} and F_1^{weight}. In order to verify if the difference between classifiers is significant, a *paired Student t-test* is performed on the results of the different folds.

4.3.4 Paired student t-test

In order to see if the average difference of a scoring function between two classifiers is relevant, we can use a *Student t-test*. Since the evaluation of the classifiers is performed on the same folds, we can compare the results of the same folds, and thus use a *paired test*. This statistical model is covered in for example [32, 33]. It will give us more insight about the actual difference in mean of the two classifiers. Indeed, if the same difference, even small, is observed on all the folds, the difference in mean is likely to be significant. On the contrary, if one classifier performs better on some folds and worse on others, the difference in mean might not be so relevant, even if it is large.

In order to approximate the true difference of performance between two classifiers, we have at our disposal the empirical differences on each of the K folds: $\hat{d}_1, \ldots, \hat{d}_K$. We assume that they are the values of a random sample $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_K$, where the differences follow a normal distribution of mean $\mu_{\boldsymbol{d}}$ and variance $\sigma^2_{\boldsymbol{d}}$. If the number of examples in the different folds is high enough, this is a reasonable assumption that follows from the *central limit theorem*. The sample mean is represented by $\bar{\boldsymbol{d}}$, whose value $\bar{d}$ is the mean of the different $\hat{d}_k$. We would like to know if this mean difference is significant. Therefore, we define as null hypothesis that the true mean difference in performance is zero, i.e. insignificant:

$$H_0 : \mu_{\boldsymbol{d}} = 0 \tag{4.2}$$

On the other hand, we define as alternative hypothesis that the different is non-zero, i.e. it is significant:

$$H_1 : \mu_{\boldsymbol{d}} \neq 0 \tag{4.3}$$

We would like to know whether we can reject the H_0 hypothesis, and thus conclude that the difference in performance is significant.

Let $\boldsymbol{t}$ be the test statistic. We assume that it follows a Student distribution, with $K-1$ degrees of freedom, where K is the number of folds. It is expressed by [32, 33]:

$$\boldsymbol{t} = \frac{\bar{\boldsymbol{d}} - \mu_{\boldsymbol{d}}}{\sigma_{\bar{\boldsymbol{d}}} \,/\, \sqrt{K}} \tag{4.4}$$

where the standard deviation $\sigma_{\bar{\boldsymbol{d}}}$ can be approximated by:

$$s_{\bar{\boldsymbol{d}}} = \sqrt{\frac{1}{K-1}\sum_{k=1}^{K}(\hat{d}_k - \bar{d})^2} \tag{4.5}$$

Under the null hypothesis, we have $\mu_{\boldsymbol{d}} = 0$, so we can compute the value of $\boldsymbol{t}$:

$$t = \frac{\bar{d}}{s_{\bar{\boldsymbol{d}}} \,/\, \sqrt{K}} \tag{4.6}$$

Now that we have a value t, we look at whether we can reject this H_0 hypothesis. In order to do this, we compute the rejection region of t, according to a certain level of confidence $1 - \alpha$. It is given by:

$$RR_\alpha = \,]-\infty,\, t_{\frac{\alpha}{2},K-1}] \cup [t_{\frac{\alpha}{2},K-1},\, +\infty[\tag{4.7}$$

For $K = 10$ and $\alpha = 0.05$, i.e. nine degrees of freedom and a confidence of 95 %, we have:

$$RR_{0.05} = \,]-\infty, -2.262] \cup [2.262, +\infty[\tag{4.8}$$

Whenever t falls into this region, we can reject the H_0 hypothesis, with a certain confidence, and conclude that the performance of the two classifiers are significantly different. Note that if t doesn't fall into the rejection region, we cannot conclude anything as there is no indication that the means are, or are not, significantly different.

Chapter 5

Experimental results

In this chapter, we will evaluate the performance of the MaxEnt classifier. This will be done by first comparing it to the baseline classifiers, and see if it can provide some improvement in classification on any dataset. In the second part of this chapter, we will analyse the impact of the variations of the MaxEnt classifier. We will first investigate whether adding constraints to model 3-by3 interactions is useful. Then, we will see if we can limit the number of variables present in the optimisation problem and still have the same performance.

5.1 MaxEnt versus baseline classifiers

First of all, we compare the cross-validation results of the MaxEnt classifier and the two baseline classifiers. The average results over the 10 folds, for the different datasets, are give in tables 5.1 to 5.5. In order to evaluate the significance of these results, a paired Student t-test is performed using the results of the 10 folds. If a classifier performs significantly better than MaxEnt, the result is shown in green. If it performs significantly worse, it is shown in red. Detailed results of this statistical test are presented in appendix B. Let's look at the results dataset per dataset.

emotions. We start with the emotions dataset. We can see that the MaxEnt method is clearly the best choice, regardless of the metric that is used to evaluate the performance. Indeed, the average results over the 10 folds of MaxEnt are higher than for the two other classifiers. These results are confirmed by the Student t-test:

- The difference of classification between MaxEnt and BinRel is significant for all the metrics used. MaxEnt is thus in all cases better than BinRel on the emotions dataset.
- The difference of classification between MaxEnt and CChain is significant for all the metrics but the accuracy score. When it comes to this scoring function, we cannot conclude that MaxEnt was better than CChain even though it had a better mean accuracy. For all the other metrics, we can be fairly certain that MaxEnt is indeed better than CChain.

On this dataset, MaxEnt thus offers a clear improvement of classification compared to the existing techniques.

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Maximum entropy	32.1	64.4	67.0	65.0	65.9
Binary relevance	26.3	57.8	64.1	62.0	62.9
Classifier chain	29.7	61.6	65.1	62.7	63.6

Table 5.1: Average results on the emotions dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Maximum entropy	60.4	67.6	71.2	71.9	70.9
Binary relevance	53.6	61.9	69.3	69.9	68.8
Classifier chain	66.3	71.0	71.1	72.2	71.1

Table 5.2: Average results on the scene dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Maximum entropy	18.5	65.0	69.1	59.3	67.0
Binary relevance	14.3	66.4	71.2	60.8	69.5
Classifier chain	23.0	65.8	68.9	59.2	67.6

Table 5.3: Average results on the flags dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Maximum entropy	34.1	67.6	74.2	68.5	72.6
Binary relevance	29.2	69.0	74.9	69.5	73.4
Classifier chain	32.7	67.7	73.7	68.0	72.2

Table 5.4: Average results on the yeast* dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Maximum entropy	86.9	48.4	88.5	87.4	88.5
Binary relevance	86.3	48.4	88.2	87.2	88.3
Classifier chain	87.1	48.6	88.5	87.5	88.6

Table 5.5: Average results on the medical* dataset, in %

scene. Next, we look at the scene dataset. For the label-based metrics, i.e. *accur* and F_1^{exam}, there is a clear ordering between the classifiers: CChain is the best method for this dataset, followed by MaxEnt, and BinRel is the one with the lowest performance. This ordering is confirmed by the Student t-test. This means that, considering the example-based metrics, MaxEnt is not able to compete with CChain on the scene dataset.

For the label-based metrics, MaxEnt still performs better than BinRel, as shown by the t-test. However, there is no longer a significant difference between MaxEnt and CChain. Indeed, we can see that the values of the label-based metrics lie very close to each other, and the Student t-test confirms that the differences are not significant. We can thus conclude that, from the point of view of the label-based metrics, the MaxEnt method can compete with CChain. However, it is not able to give a significant plus-value in terms of performance on the scene dataset.

Note that on this dataset, we saw a clear distinction between the two different types of metric functions. This confirms what has been said before: when classifying a multi-label dataset for a real-world application, one must really think about which metric function is the most relevant for the application at hand.

flags. We continue with the flags dataset. This dataset is actually a toy dataset, with fewer examples than the other datasets. This means that the size of one fold is around 20 examples. Recall that the Student t-test relies on the Central Limit theorem, by assuming that the differences of the folds are normally distributed. For this, there might be too few examples, so the t-test might be a bit less accurate than for the other datasets.

We can see that the mean results of the three classifiers are quite variable. When it comes to accuracy, MaxEnt seems to perform better than BinRel and worse than CChain, but according to the t-test, the results are not representative. For the other metrics, BinRel seems to slightly have the upper hand, but the t-test only considers it relevant for F_1^{micro} and F_1^{weight}. The differences between MaxEnt and CChain seem non relevant.

yeast*. Recall that for this dataset, the five most occurring labels were used by the classification algorithms. The results show that MaxEnt performs in general slightly better than CChain, but these differences are not deemed significant by the t-test.

When it comes to the comparison with BinRel, MaxEnt preforms significantly better in terms of accuracy. However, BinRel has a better score for all the other metrics. The Student t-test indicated that it is significant for F_1^{exam} and F_1^{macro}.

For the last two datasets, we saw that BinRel, a classifier that assumes complete independence between labels, has better results (except when it comes to accuracy) than two classifiers that try to model relationships between classes. This might be explained by the fact that the labels are not really correlated in these datasets, and that the independence assumption might be accurate. The less good performance of MaxEnt and CChain might be explained by the fact that they could be "looking for things that are not there".

medical*. This dataset has a very small number of labels, namely four. We can see that, for all the metrics, there is no significant difference between the three classifiers. Note that the

F_1^{exam} score is significantly lower than all the other metrics. This comes from the fact that this metric is not defined when there are no labels in the output. The default solution is to give a score of zero in this case. In this limited version of the medical dataset, where we use only four labels, zero-label outcomes occur quite a lot. Therefore, this metric function is not adapted since it will give a score of zero when the classifier correctly assigned no label to the example. This illustrates that one must be very careful about the choice of metrics to optimally evaluate the classification of a real-world application.

Conclusion. From these observations, we can conclude that once again, there is no universally better method that overpowers all the others. However, we can see that in general, the MaxEnt classifier is able to compete with the others: there is no dataset where MaxEnt is largely outperformed by another method. Moreover, it showed a significant improvement in classification for one dataset. Therefore, there might be other multi-label applications for which the MaxEnt method can have a significant advantage compared to BinRel and CChain. As usual, it is important to use some techniques, such as cross-validation, to see if the MaxEnt classifer is relevant for the task at hand.

Another way to evaluate whether MaxEnt offers some classification improvement is by counting the number of times MaxEnt performed significantly better, and significantly worse, than the other classifiers. When comparing it to BinRel, MaxEnt wins with a score of 11 against 4. Against CChain, the score is 4 against 2 in favour of MaxEnt. This confirms our earlier conclusion that MaxEnt could give some added value for the multi-label classification task.

However, a major drawback is the execution time of the MaxEnt classifier. When the number of labels begins to grow, the optimisation problem needs quite some time to execute. This was visible for the datasets with 6, and especially 7, labels. This means that classifying the entire dataset takes much more time than when using BinRel or CChain. For some real-world applications, where classification must be fast, this might be a critical factor. For other applications, it might not be such a problem.

5.2 Variations of the MaxEnt method

Now that the maximum entropy method has been compared to other classifiers, we are going to explore some variations of the method. First, we are going to see whether there is some benefit to add higher order statistics to the model. Then, we are going to see if we can effectively limit the number of variables in the optimisation problem, and how it might affect the classification performance. In particular, we're going to see whether the model with Poisson regression performs as well as the regular maximum entropy classifier.

5.2.1 Extension to higher order statistics

First, we analyse whether adding higher order statistics improves the classification of the maximum entropy method. Therefore, the experiment has been done over, on the same 10 folds, with an adapted version of MaxEnt where 3-by-3 label interactions were considered (which we

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
2nd order	32.1	64.4	67.0	65.0	65.9
3rd order	31.4	63.5	66.0	64.2	65.1

Table 5.6: Average results on the emotions dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
2nd order	60.4	67.6	71.2	71.9	70.9
3rd order	62.1	67.7	71.4	72.1	71.1

Table 5.7: Average results on the scene dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
2nd order	18.5	65.0	69.1	59.3	67.0
3rd order	18.5	65.3	69.3	58.4	65.3

Table 5.8: Average results on the flags dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
2nd order	34.1	67.6	74.2	68.5	72.6
3rd order	33.7	67.0	73.6	67.6	71.9

Table 5.9: Average results on the yeast* dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
2nd order	86.9	48.4	88.5	87.4	88.5
3rd order	86.7	48.2	88.2	87.0	88.2

Table 5.10: Average results on the medical* dataset, in %

will call MaxEnt3). In this version, the third order constraints, as defined in section 3.2.4, were added to the model. This will have as effect to slow down the optimisation solver, as there are more constraints to evaluate. However, the 3-by-3 interactions might contain some unused information that could improve classification.

The average performances of the two models are given in tables 5.7 to 5.10. It is clear from these results that there are no real differences between them. Sometimes MaxEnt3 performs sightly better, sometimes slightly worse. Even when the scores are a bit better, there is no indication that MaxEnt3 is significantly better than MaxEnt, whilst it slows down the process quite a lot. We can therefore conclude that it doesn't seem relevant to extend the maximum entropy method further than the second order statistics, at least on the investigated datasets.

5.2.2 Size limitation and Poisson prediction

Observing the data. When observing the distribution of the number of labels in the outputs (figure 4.1, page 37) for the emotions and scene datasets, we can see that there were no examples with more than 3 labels. Therefore, we can ask ourselves if we cannot force the MaxEnt method to not predict more than 3 labels. By doing this, we can remove all the variables corresponding to outputs with more than 3 labels, which constitutes quite a saving. In fact, for scene, there is only one example with 3 labels, so we might even consider limiting the model to 2 labels.

The results of these experiments are shown in tables 5.12 and 5.13. We can see that the classification performance doesn't suffer from the limitation of the variables. On the contrary, it sometimes improves the classification a bit: for some examples, MaxEnt would probably assign too many labels, which is prevented thanks to this limitation. The improvement is quite small, and thus probably not very significant, but it is still interesting. Sometimes, the improvement of the accuracy score might be significant, as is the case when we limit scene to 2 variables. One must however not limit the number of variables too much. Indeed, when limiting to 2 labels on the emotions dataset, where there are some examples with 3 labels, the performance decreases, just as should be expected.

Poisson prediction. Now, we investigate whether predicting the number of labels in the output, through a Poisson regression, and then running MaxEnt is effective. We will call this procedure MaxEntPois. In order for this to work, the Poisson prediction should be as accurate as possible. This is especially true for the accuracy score: if the prediction is wrong for an example, it will automatically get a score of zero. This is not the case for the other metrics. Indeed, if an example is for instance predicted to have 2 labels instead of 3, the MaxEntPois classifier might still assign correctly these 2 labels and get some credit for it.

The accuracies of the Poisson regression on the different datasets are given in table 5.11.[1] We can see that the Poisson prediction performs very badly on yeast*, with less that 15 % accuracy. This is reflected in the results of MaxEntPois, given in table 5.15. The results on all the metrics are extremely low, compared to what we had with the regular MaxEnt. As expected,

[1]For some reason, the Poisson regression algorithm did not converge for the medical* dataset. For this dataset, another predictor should be used.

	accur
emotions	51.3
scene	92.5
flag	83.0
yeast*	12.5

Table 5.11: Accuracy of the Poisson regression algorithm, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All labels	32.1	64.4	67.0	65.0	65.9
0 to 3 labels	32.1	64.8	67.3	65.7	66.4
0 to 2 labels	28.3	59.4	62.8	61.5	62.0
Poisson prediction	26.5	61.1	63.7	62.4	63.5

Table 5.12: Average results on the emotions dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All labels	60.4	67.6	71.2	71.9	70.9
0 to 3 labels	61.2	67.7	71.2	72.0	70.9
0 to 2 labels	63.3	67.5	71.4	72.1	71.0
Poisson prediction	64.3	68.7	68.4	69.2	68.4

Table 5.13: Average results on the scene dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All labels	18.5	65.0	69.1	59.3	67.0
Poisson prediction	23.6	66.7	69.8	59.8	67.9

Table 5.14: Average results on the flags dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All labels	34.1	67.6	74.2	68.5	72.6
Poisson prediction	4.0	31.4	38.2	37.9	38.2

Table 5.15: Average results on the yeast* dataset, in %

the results for the accuracy score are especially catastrophic. We can thus logically conclude that MaxEntPois should never be used if we cannot have a good prediction for the number of labels.

For the emotions dataset, the Poisson regression has a mitigated accuracy (table 5.12). However, MaxEntPois still performs reasonably well. The performance is lower than for MaxEnt, but it is still appreciable considering the fact that the Poisson regression is correct only half of the time. In fact, the t-test cannot affirm that the differences are significant, except of course for the accuracy score. This means that even though there is, for the other metrics, a relatively big difference in mean between MaxEnt and MaxEntPois, there is a quite high variance among the different folds. Indeed, for some folds, MaxEntPois actually performed better. Therefore, we can not tell if this difference in mean is significant.

For the last two datasets (tables 5.14 and 5.13) , the Poisson regression did a good job in predicting the number of labels. This leads to the fact that on the flags dataset, MaxEntPois did even better than the regular MaxEnt. Even though the difference is not considered significant by the Student t-test, performing as well as MaxEnt is still a nice achievement.

Paradoxically, this is not the case for the scene dataset, even though the Poisson regression had a better accuracy for this dataset than for flags. Whilst for the accuracy score, there is a significant improvement, confirmed by the t-test, the performance as measured by the label-based metrics (F_1^{micro}, F_1^{macro} and F_1^{weight}) is a bit lower for MaxEntPois. The difference is not huge, but it was nevertheless confirmed by the t-test. Indeed, this tendency is observed in the results of all the folds, so it must probably be accurate.

Conclusion. We can thus conclude that, provided a good predictor, a maximum entropy method where we predict the number of labels in the output beforehand could potentially be as good as the original method. Finding a good prediction algorithm can be done in advance, as it is completely independent from MaxEnt; the latter just requires the algorithm to have a good accuracy. We can thus easily select the best predictor beforehand, for example with a quick cross-validation. However, we have seen that the performance of MaxEntPois is not completely proportional to the accuracy of the predictor. For some datasets, a higher accuracy might be required compared to other datasets in order to have the same result as with the regular MaxEnt.

However, if we can manage to obtain this, we have a method that is much faster than the normal MaxEnt method. This is especially the case for datasets where there are few labels in the output, compared to the total number of labels. Indeed, as discussed in section 3.3.3, the number of variables follows a bell-shaped distribution. Therefore, MaxEntPois was extremely quick for the scene dataset. For the flags dataset, where the number of labels in the output varies more, the gain in speed was less important.

We should note that, as was concluded when analysing the theory, this increase in speed does not circumvent the exponential complexity of the MaxEnt method. Whilst MaxEntPois may be used to speed up the classification of problems that were already solvable, it cannot really be used to classify new problems, with larger amounts of labels.

5.2.3 Perfect predictor

As we just saw, the accuracy of the Poisson regression algorithm varies largely from one dataset to another. When the latter obtained a good accuracy, which was the case for two datasets, the MaxEntPois algorithm perform as well, or almost as well, as the complete MaxEnt algorithm. However, since all the datasets were not on an equal footing, it is difficult to have a definitive judgement on the power of this alternative maximum entropy method.

In consequence, it seems interesting to know just how far MaxEnt can go when we predict the number of variables. Therefore, we could define an imaginary "perfect predictor", which we will denote by MaxEntPerfP, to assess the limit of the model. This would be some sort of oracle that had a perfect knowledge about the number of labels assigned to an example. This theoretical evaluation can be performed by simply giving to MaxEnt the actual number of labels of each example, as indicated in the dataset, when predicting an example from the test set. Hence, we are assured to have the same set up for all datasets.

Of course, this cannot be used for predicting unseen examples, but we will be able to have an idea of the best possible performance of the MaxEnt version with prediction of the number of variables. If it performs well on each dataset, we will be able to conclude that this version of MaxEnt is as powerful as the complete MaxEnt model. In this case, when classifying a new dataset, the user will only need to find a good predictor to have a model as performing as MaxEnt, but that runs faster. Moreover, checking for a good predictor can be done relatively easily, for instance with a cross-validation, as it is independent of MaxEnt: we just need to have the best accuracy possible.

Results. The results of this experiment are represented in figures 5.16 to 5.20. As we can see, MaxEntPerfP has at least an equivalent performance to MaxEnt, and often outperforms it. Therefore, we can conclude that the version of MaxEnt with prediction of the number of variables should be as good as the complete version of MaxEnt, if we have a predictor with an accuracy that tends to the perfect case.

This assertion is especially true for the accuracy score. Indeed, we can see that MaxEntPerfP significantly outperforms MaxEnt on every dataset, with a large difference in mean. This is logical, since the accuracy gives a score of 1 if all the labels of an example are correctly predicted, and 0 in all other cases. When the correct number of labels is predicted before running MaxEnt, which is always the case in this experience, there are far less possible outcomes than when all the variables are present. Therefore, even if we pick the outcome randomly, we have a greater probability of finding the right one with MaxEntPerfP. And since the maximum entropy method is there to guide the prediction, it is not surprising we obtain a good accuracy score. Predicting the number of labels in advance is therefore certainly a good idea when the accuracy is crucial for the task at hand.

For the other metrics, most of the time there is a significant improvement in classification when using MaxEntPerfP rather than MaxEnt. In those cases, we can be quite sure that when we have a relatively good predictor, the classifier will be as good as the complete MaxEnt classifier. This was visible with the Poisson regression on flags, where a predictor accuracy of 80 % was sufficient to have a classifier that performed as good as the complete MaxEnt.

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All variables	32.1	64.4	67.0	65.0	65.9
Perfect prediction	46.7	61.3	67.3	66.4	67.1

Table 5.16: Average results on the emotions dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All variables	60.4	67.6	71.2	71.9	70.9
Perfect prediction	68.4	69.8	70.2	70.9	70.3

Table 5.17: Average results on the scene dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All variables	18.5	65.0	69.1	59.3	67.0
Perfect prediction	27.3	67.8	71.5	61.9	69.7

Table 5.18: Average results on the flags dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All variables	34.1	67.6	74.2	68.5	72.6
Perfect prediction	57.3	69.7	82.1	78.7	81.9

Table 5.19: Average results on the yeast* dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
All variables	86.9	48.4	88.5	87.4	88.5
Perfect prediction	98.0	52.2	96.7	96.0	96.7

Table 5.20: Average results on the medical* dataset, in %

For some datasets, things seem a bit more complicated. For instance, on the scene dataset, the results of the label-based metrics (F_1^{micro}, F_1^{macro} and F_1^{weight}) of MaxEntPerfP are not able to outperform the results of MaxEnt; both are considered equivalent, whilst we have a perfect prediction. Therefore, on this type of datasets, we will need a very good predictor to have an equivalent performance to MaxEnt. Indeed, the Poisson regression of scene had an accuracy of 92 %, but it wasn't sufficient to match with MaxEnt on those labels. This indicates that while the accuracy of the predictor is the principal factor that influences the performance of MaxEnt with prediction of the number of labels, there is still some influence from MaxEnt itself, i.e. how it reacts to the limitation of the number of variables.

In conclusion, it seems that, theoretically, a version of MaxEnt with prediction of the number of variables should be at least equivalent to the complete version of MaxEnt, when it comes to classification performance. Moreover, the former would run much faster, since there would be a lot less variables. Another advantage is that when the prediction is accurate, it will increase the chances of MaxEnt to find the correct outcome. However, it appears that all datasets are not on an equal footing, so that for some datasets, having an extremely good performance of the predictor is more crucial than for other datasets.

Note that this is only a theoretical evaluation. In order for this to work in practice, we have to find a good predictor for the number of labels in the output. Whether we are able to find one must be evaluated for each dataset, probably by comparing different methods. The Poisson regression was able to obtain a good accuracy on two datasets. This indicates that predicting accurately the number of labels in the outputs must be feasible for at least some datasets, while for others it might be unfeasible. In any case, when classifying a new dataset, we can easily evaluate whether we can find a good predictor using, for instance, a cross-validation. Then, we can decide whether limiting the number of variables in MaxEnt using this predictor seems relevant or not.

5.3 Combining maximum entropy and classifier chain

As an additional experiment, we can search for a way to use the maximum entropy in a classifier that is scalable to problems with larger amounts of labels. A way to obtain this could be to define hybrid methods, combining the maximum entropy method with some other classifier. In particular, a combination of maximum entropy and classifier chain comes to mind.

Indeed, for the latter, there is an imbalance between the classifiers at the beginning or at the end of the chain. When predicting the last labels in the ordering, the classifiers know the values of all the preceding labels, and can thus deduce a lot of extra information regarding relationships between classes. On the contrary, the classifiers of the first labels in the chain must operate with few or no knowledge about other labels.

In order to compensate this, we could use the MaxEnt classifier to start the algorithm by predicting the values of a few number of labels. This will mean that the prediction of these first labels will be done using more information than if we used a classifier chain. Once we have these values, we can predict the rest of the labels using a classifier chain. The advantage is

	Nb. of examples	Nb. of features	Nb. of labels	Cardinality
yeast	2417	103	14	4.237
medical	978	1449	45	1.245
enron	1702	1001	53	3.378
mediamill	43907	120	101	4.376

Table 5.21: Some statistics about the datasets

that even the first classifiers in the chain will have access to some information using the values given by MaxEnt. Of course, this will not make the imbalance vanish completely, but it might certainly help equilibrate things. We have thus found a way of using the maximum entropy method in a classifier that is scalable, which we will denote by MECC.

In order to quickly test this method, we can run it on the complete yeast and medical datasets, along with enron and mediamill, taken from [34,35]. Note that some labels were left out because they appeared in very few examples, so that it could happen that the training set contains no example with this label. BinRel, CChain and MECC use a binary classifier to make predictions for a label, and it is impossible to build a model if there are no positive examples. Therefore, these labels were left out. Table 5.21 shows some statistics about these datasets.

The results are shown in tables 5.22 to 5.25. In this experiment, we arbitrarily set the number of labels predicted by MaxEnt to five. As always, the results are different from one dataset to another. For medical, enron and mediamill, we cannot see any real difference between MECC and CChain. There seems to be no improvement on these three datasets when using MaxEnt at the beginning of the chain, rather than the normal chain.

For yeast, however, MECC seems to improve the classification compared to CChain, and this for each metric that is used to evaluate the performance. This difference is confirmed by the Student t-test. It therefore seems that this hybrid strategy could improve the classification performance of a chained strategy for some datasets.

These results constitute a first test on this subject, that could be analysed more deeply in a further work. In particular, the number of labels that are predicted using MaxEnt is an important hyperparameter. It would therefore be interesting to analyse the evolution of the classification performance in function of this parameter. It might be so that in order to have a clear benefit, a certain fraction of the labels must be predicted using MaxEnt. Indeed, the best results were observed on the dataset with the smallest number of labels, where 5 labels out of 14 were predicted by MaxEnt. On the contrary, predicting for instance 5 labels out of 101 using MaxEnt might not have a very big impact. Therefore, it is possible that if we increased the number of labels predicted by MaxEnt, we could improve the classification of the other datasets. This aspect could be evaluated properly during a further work, in order to have a better understanding of what this hybrid strategy can achieve.

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
MaxEnt + chain	23.1	61.2	64.0	40.3	59.8
Binary relevance	14.9	61.1	63.5	35.0	56.3
Classifier chain	19.8	58.4	62.0	39.3	58.0

Table 5.22: Average results on the yeast dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
MaxEnt + chain	72.4	76.4	82.6	61.7	79.7
Binary relevance	70.2	75.2	82.6	61.3	79.5
Classifier chain	72.4	76.4	82.7	62.0	79.8

Table 5.23: Average results on the medical dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
MaxEnt + chain	15.1	55.1	56.0	25.2	52.8
Binary relevance	13.7	54.7	56.4	24.9	53.0
Classifier chain	15.1	55.2	56.0	25.1	52.8

Table 5.24: Average results on the enron dataset, in %

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
MaxEnt + chain	9.8	49.2	51.6	4.7	41.4
Binary relevance	8.0	52.8	54.2	4.8	42.9
Classifier chain	9.7	49.2	51.6	4.7	41.4

Table 5.25: Average results on the mediamill dataset, in %

Chapter 6

Conclusion and further work

At the end of this thesis, we have been able to deepen our understanding of multi-label classification, the challenges that come with it, and different approaches to solving this problem. In particular, we have seen how to develop a method that explicitly models relationships between class labels, using the maximum entropy approach.

In order to achieve this, we used the predictions made by single-label classifiers regarding the probability of occurrence of a label, and co-occurrence of pairs of labels. These different probabilities are blended together in one consistent probability distribution, from which we can retrieve the most probable multi-label output. To conclude this work, we recapitulate some key characteristics of the maximum entropy classifier.

Exponential complexity. As said earlier, when modelling relationships explicitly, we need to consider every possible output. Hence, this model implies the solving of a convex optimisation problem, whose number of variables grows exponentially in function of the number of class labels that are present in the data. This means that the maximum entropy method is not scalable to applications with a large number of labels. Hence, this method is limited to problems having relatively few variables. This is a huge limitation compared to other methods in multi-label learning, which easily scale to larger amounts of labels. However, these methods usually put assumptions on the label distributions, and tend to not represent thoroughly the relationships between labels. Trying to do the latter, as is the case of the MaxEnt method, comes at a price.

It appeared that trying to circumvent this exponential complexity was not possible for the MaxEnt method. Indeed, heuristics where we predicted the number of labels before the optimisation, in order to remove a large part of the variables, did not reduce the problem to a polynomial complexity. Indeed, whilst it could in a lot of cases remove a huge amount of variables, the worst case still had an exponential complexity. Reducing the complexity of the MaxEnt method, and rendering it scalable, seems thus impossible.

However, hybrid methods combining the MaxEnt method and some other classifier could be an interesting strategy. In particular, a combination of maximum entropy and classifier chain comes to mind. Indeed, in the latter, the prediction of the last labels is done knowing the values of the preceding labels, but the first classifiers in the chain must operate with few or no

knowledge. Therefore, we could use the MaxEnt classifier to predict the values of a few labels, and then predict the rest of the labels using a classifier chain, that will use the information given by MaxEnt from the start. This could be a way of using the MaxEnt method and obtaining a scalable classifier.

Performance. When testing the maximum entropy method, it appeared that it has a comparable performance to two related multi-label classifiers, binary relevance and classifier chain, which also aggregate predictions of binary classifiers. Moreover, it was able to significantly improve the classification on one of the datasets. This leads us to believe that the maximum entropy method has some benefits to offer for certain datasets, and could thus be seen as classifier providing added value compared to existing problem transformation methods.

Of course, it is not a universally better method, and there are probably a lot of datasets that will not benefit from this method. Since it has quite a slow execution compared to other classifiers, this method should not be used if there doesn't seem to be a significant gain. However, for some datasets, there might be strong relationships between classes, that other models might not be able to represent. For these datasets, the maximum entropy classifier could provide a boost of classification performance. As usual, a careful evaluation with a good model selection procedure is required.

A drawback of the maximum entropy method is that it needs to solve one optimisation problem for each example that is being classified. This implies that the time taken to classify an example is much longer than for another classifier. By how much depends on the number of labels in the output, and it quickly becomes significant due to the exponential complexity of the method. While the problem is solvable in a reasonable time for problems with few labels, it still takes quite some time compared to other classifiers.

Whether this execution time is a problem depends on the application at hand. Indeed, if we need a multi-label classifier that operates on the stock markets, quick predictions are paramount. In that case, we will probably prefer a classifier that predicts quickly, even if it has a somewhat lower accuracy. On the contrary, a medical application, where there are few examples to classify, could afford to use a slower algorithm. When a patient consults his physician, waiting even a couple of minutes to get a result doesn't constitute a problem. What's important is that the result is accurate.

In conclusion, the usefulness of the maximum entropy classifier is strongly dependent on the application at hand. Due to its computational weight, it might be unuseful or unpractical for many applications. However, for others it might have some added value compared to existing methods.

Further work. The clearest limitation of the MaxEnt method is its exponential complexity, and thus the fact that it only can be used with a small number of labels. However, it could be interesting to search whether we can combine it with other algorithms in order to have a hybrid method, that is scalable to problems with large amounts of labels. Here, we performed a quick introductory experiment combining maximum entropy and classifier chains, but the idea might be worth exploring more in depth.

Another possible improvement regards the prediction of the number of labels. Here, we used a Poisson regression algorithm to perform this prediction. It proved effective on some datasets, but performed poorly on others. Therefore, one could search for a set of predictors as alternative to the Poisson regression. Indeed, the latter is a linear method, so it would be interesting to investigate whether there are non-linear methods that could do the trick. It would be useful to have a certain number of them, such that the user could select the one that performs best on his dataset.

Finally, this thesis focussed on machine learning, and specifically on the classification performance. During the experiments, one working implementation was used. However, this implementation is probably not the most effective, from an optimisation performance point of view. Indeed, one could compare different optimisation techniques and solvers in order to identify the one that can solve the optimisation problem in the least time, based on the results presented in [20, 22].

Notations

u	Italic: one-dimensional values
$\mathbf{u}$	Bold: vectors
$\mathbf{U}$	Capital bold: matrices
U	Capital italic: sets (unless otherwise specified)
$\boldsymbol{u}$	Bold italic: random variables
$P(\cdot)$	Probability
$E[\cdot]$	Mathematical expectation
$[\![\cdot]\!]$	Evaluation of a predicate (returns either 1 or 0)
$\langle\cdot\rangle$	Extension of a vector: $\langle \mathbf{u}, v\rangle \triangleq (u_1, \ldots, u_n, v)^{\mathrm{T}}$
C	The number of labels
D	The number of features
N	The number of examples
$\mathcal{X}$	The D-dimensional feature space
$\mathcal{Y}$	The label space with C possible labels
$\mathbf{x}$	The D-dimensional feature vector corresponding to an example
y	The empirically measured label of an example (single-label cl.)
$\hat{y}$	The predicted label of an example (single-label cl.)
$\mathbf{y}$	The empirical C-dimensional binary label vector of an example (multi-label cl.)
$\hat{\mathbf{y}}$	The predicted C-dimensional binary label vector of an example (multi-label cl.)
$\boldsymbol{y}$	Random variable representing the true, unknown, output associated to an example
$u^1, \ldots, u^p, \ldots, u^N$	The use of a superscript p denotes the number of the example in the dataset. All other superscripts denote the usual power operator.

Bibliography

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[2] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*. Springer, 2013.

[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag New York, second edition, 2009.

[4] E. Alpaydin. *Introduction to Machine Learning*. MIT press, second edition, 2009.

[5] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, fourth edition, 2008.

[6] M.-L. Zhang and Z.-H. Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, August 2014.

[7] F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesus. *Multilabel Classification: Problem Analysis, Metrics and Techniques*. Springer International Publishing, 2016.

[8] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, June 2011.

[9] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[10] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, second edition, 1979.

[11] E. T. Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, September 1982.

[12] E. T. Jaynes. *Probability theory: the logic of science*. Cambridge University Press, 2003.

[13] J. N. Kapur and H. K. Kesavan. *Entropy optimization principles with applications*. Academic Press, 1992.

[14] S. Zhu, X. Ji, W. Xu, and Y. Gong. Multi-labelled classification using maximum entropy method. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 274–281, New York, NY, USA, 2005. ACM.

[15] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3, 4):379–423, 623–656, July, October 1948.

[16] M. Saerens and F. Fouss. Yet another method for combining classifiers outputs: A maximum entropy approach. In F. Roli, J. Kittler, and T. Windeatt, editors, *Multiple Classifier Systems*, pages 82–91, Belin, Heidelberg, 2004. Springer Belin Heidelberg.

[17] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[18] S.-C. Fang, J. Rajasekera, and H.-S. Tsao. *Entropy optimization and mathematical programming*, volume 8. Springer Science & Business Media, 2012.

[19] H. Ku and S. Kullback. Approximating discrete probability distributions. *IEEE Transactions on Information Theory*, 15(4):444–447, July 1969.

[20] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[21] S. J. Benson and J. Moré. A limited-memory variable-metric algorithm for bound-constrained minimization. Technical Report ANL/MCS-P909-0901, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

[22] H.-F. Yu, F.-L. Huang, and C.-J. Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1):41–75, October 2011.

[23] P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman and Hall/CRC, second edition, 1989.

[24] C. Gourieroux. *Econométrie des variables qualitatives*. Economica Paris, 1984.

[25] W. H. Green. *Econometric analysis*. Prentice Hall, 2007.

[26] D. Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, Köln, 1988.

[27] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas. Multilabel classification of music into emotions. In *Proceedings og the 9th International Conference on Music Information Retrieval*, ISMIR '08, pages 325–330, Philadelphia, PA, USA, 2008.

[28] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

[29] E. Corrêa Gonçalves, A. Plastino, and A. A. Freitas. A genetic algorithm for optimizing the label ordering in multi-label classifier chains. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '13, pages 469–476, Washington D.C., USA, November 2013.

[30] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems 14*, pages 681–687. MIT Press, 2001.

[31] J. P. Pestian, C. Brew, P. Matykiewicz, D. J. Hovermale, N. Johnson, K. Bretonnel Cohen, and W. Duch. A shared task involving multi-label classification of clinical free text. In *Proceedings of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing*, BioNLP '07, pages 97–104, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[32] D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. Wiley, third edition, 2002.

[33] R. E. Walpole, R. H. Myers, S. L. Myers, and K. E. Ye. *Probability & Statistics for Engineers & Scientists*. Pearson, ninth edition, 2016.

[34] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*, pages 217–226. Springer, 2004.

[35] C. G. Snoek, M. Worring, J. C. van Gemert, J.-M. Geusebroek, and A. W. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th ACM International Conference on Multimedia*, MM '06, pages 421–430, New York, NY, USA, 2006. ACM.

Appendices

A Source code of the classifiers

The implementation of the MaxEnt and MECC classifier can be downloaded from:

`www.github.com/alex7994/EPL-TFE_MaxEntMLC`

The classifiers will require the installation of the following Python modules:

- NumPy: `www.numpy.org`
- ScyPy: `www.scipy.org`
- scikit-learn: `www.scikit-learn.org/stable/`
- StatsModels: `www.statsmodels.org/stable/`

A small documentation of the content of the code is provided below.

A.1 MaxEnt classifier

The MaxEnt classifier is composed of the following scripts, contained in the max_entropy package:

- multi_label_max_ent.py. This is the main script, that contains the definition of the `MultiLabelMaxEnt` class.
- optimisation.py. This script contains the definition of the optimisation problem.
- optimisation_limited.py. This script contains a variation of the optimisation problem, where bounds are imposed on the number of labels that can be present in the output.
- optimisation_sclack.py. This script contains the optimisation problem with bounds and slack variables in all the constraints (except the probability constraint $\sum P = 1$).
- poisson.py. This script contains the functions of the Poisson regression.

The maximum entropy classifier is defined by the class:

`max_entropy.multi_label_max_ent.MultiLabelMaxEnt`

It takes the following optional arguments:

- `bin_cl`, a binary classifier that must contain `fit(X, Y)` and `predict_proba(X)` methods.

  Default: `LogisticRegression()`

- `var_limits`, either `None` or an array containing 2 integers `min` and `max`, with $0 \leq$ `min` $\leq$ `max` $\leq C$. In the latter case, the integers define the minimum and maximum number of labels that may be present in the output.

  Default: `None`

- `poisson_prediction`, a boolean indicating whether the number of labels in the output of each example must be predicted before the optimisation using a Poisson regression.

  Default: `false`

- `order`, an integer. Either 2, for the standard MaxEnt model, or 3, if we want to add the third order constraints to the model.

  Default: 2

- `max_iter`, the maximum number of iterations the optimisation solver may effectuate.

  Default: 100

The `MultiLabelMaxEnt` class has the following methods:

- `fit(X, Y)`, where `X` is a feature matrix and `Y` is a label matrix. This function fits the data to the model.

- `predict(X)`, which returns the predicted label matrix associated to feature matrix `X`, using the maximum entropy method.

A.2 Hybrid classifier

The hybrid MECC classifier is implemented in max_ent_cl_chain.py, which defines the class:

`max_entropy.max_ent_cl_chain.MaxEntClChain`

The constructor takes the following optional arguments:

- `bin_cl`, a binary classifier that must contain `fit(X, Y)`, `predict_proba(X)` and `predict(X)` methods.

  Default: `LogisticRegression()`

- `maxent_nr` an integer indicating how many labels must be predicted using the maximum entropy method.

  Default: 5

- `label_order`, an array of integers defining the order in which the labels must appear in the chain. If no order is provided, the chain will use the order of the columns in the label matrix.

  Default: `None`

- `max_iter`, the maximum number of iterations the optimisation solver may effectuate. Default: 100

As for the preceding one, this class has `fit(X, Y)` and `predict(X)` methods.

B Results of the Student t-tests

In this appendix, the results of the Student t-tests of the 10-fold cross-validation are presented. Recall that the null hypothesis H_0 was that the true performance of the compared classifiers is equal. The alternative hypothesis H_1 is that there is a difference in classification between the two. A 95 % confidence was used, so H_0 is rejected whenever t falls in the rejection region $]-\infty, -2.262] \cup [2.262, +\infty[$.

B.1 MaxEnt versus baseline classifiers

Here are the results of the comparison of MaxEnt and the baseline classifiers. Note that the mean difference is always given from the point of view of MaxEnt. A positive value indicates that MaxEnt has a higher score, a negative value indicates that the baseline classifier has a higher score.

emotions:

(a) Student t-test: MaxEnt versus BinRel

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	5.8	6.5	2.8	3.0	3.0
t	4.213	5.375	3.671	3.482	3.706
Conclusion	reject H_0	reject H_0	reject H_0	reject H_0	reject H_0

(b) Student t-test: MaxEnt versus CChain

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	2.4	2.8	1.9	2.3	2.3
t	1.781	3.345	2.369	3.039	3.094
Conclusion	cannot rej.	reject H_0	reject H_0	reject H_0	reject H_0

scene:

(c) Student t-test: MaxEnt versus BinRel

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	6.8	5.7	1.9	2.0	2.1
t	7.859	7.334	2.798	2.956	2.957
Conclusion	Reject H_0	Reject H_0	Reject H_0	Reject H_0	Reject H_0

(d) Student t-test: MaxEnt versus CChain

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	−5.9	−3.4	0.1	−0.3	−0.2
t	−5.465	−4.409	0.107	−0.343	−0.314
Conclusion	Reject H_0	Reject H_0	Cannot rej.	Cannot rej.	Cannot rej.

flags:

(e) Student t-test: MaxEnt versus BinRel

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	4.2	−1.4	−2.1	−1.5	−2.5
t	1.947	−1.803	−3.787	−1.298	−3.847
Conclusion	Cannot rej.	Cannot rej.	Reject H_0	Cannot rej.	Reject H_0

(f) Student t-test: MaxEnt versus CChain

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	−4.6	−0.8	0.2	0.1	−0.6
t	−2.220	−0.580	0.139	0.069	−0.512
Conclusion	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.

yeast*:

(g) Student t-test: MaxEnt versus BinRel

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	5.0	−1.4	−0.6	−1.0	−0.8
t	6.906	−3.396	−1.779	−2.465	−2.171
Conclusion	Reject H_0	Reject H_0	Cannot rej.	Reject H_0	Cannot rej.

(h) Student t-test: MaxEnt versus CChain

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	1.4	−0.1	0.5	0.5	0.4
t	1.460	−0.108	1.223	0.893	0.884
Conclusion	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.

medical*:

(i) Student t-test: MaxEnt versus BinRel

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	0.6	0.0	0.3	0.2	0.2
t	1.964	−0.001	0.911	0.420	0.676
Conclusion	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.

(j) Student t-test: MaxEnt versus CChain

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	−0.2	−0.2	0.0	−0.1	−0.1
t	−0.802	−0.978	−0.100	−0.520	−0.477
Conclusion	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.

B.2 MaxEnt with Poisson regression

Here, the results are given from the point of view of MaxEnt with Poisson regression.

emotions:

(k) Student t-test: MaxEntPois versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	−5.6	−3.3	−3.3	−2.6	−2.4
t	−2.301	−1.735	−1.8733	−1.364	−1.403
Conclusion	Reject H_0	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.

scene:

(l) Student t-test: MaxEntPois versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	3.9	1.1	−2.8	−2.6	−2.5
t	4.970	1.441	−4.602	−4.932	−4.457
Conclusion	Reject H_0	Cannot rej.	Reject H_0	Reject H_0	Reject H_0

flags:

(m) Student t-test: MaxEntPois versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	5.1	1.7	0.8	0.5	0.9
t	1.760	1.079	0.579	0.276	0.698
Conclusion	Cannot rej.	Cannot rej.	Cannot ref.	Cannot rej.	Cannot rej.

B.3 MaxEnt with perfect predictor

Here, the results are given from the point of view of MaxEnt with perfect predictor.

emotions:

(n) Student t-test: MaxEntPoisP versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	14.6	−3.1	0.3	1.4	1.2
t	6.413	−1.990	0.258	0.968	0.991
Conclusion	Reject H_0	Cannot rej.	Cannot rej.	Cannot rej.	Cannot rej.

scene:

(o) Student t-test: MaxEntPerfP versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	8.0	2.3	−1.0	−1.0	−0.6
t	11.010	3.142	−1.836	−1.707	−1.162
Conclusion	Reject H_0	Reject H_0	Cannot rej.	Cannot rej.	Cannot rej.

flags:

(p) Student t-test: MaxEntPerfP versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	8.8	2.8	2.4	2.6	2.7
t	2.995	2.364	2.315	1.412	2.202
Conclusion	Reject H_0	Reject H_0	Reject H_0	Cannot rej.	Cannot rej.

yeast*:

(q) Student t-test: MaxEntPerfP versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	23.1	2.1	7.9	10.3	9.3
t	18.633	2.352	10.730	12.519	12.635
Conclusion	Reject H_0	Reject H_0	Reject H_0	Reject H_0	Reject H_0

medical*:

(r) Student t-test: MaxEntPerfP versus MaxEnt

	accur	F_1^{exam}	F_1^{micro}	F_1^{macro}	F_1^{weight}
Mean of the diff. (%)	11.1	3.8	8.3	8.6	6.2
t	10.624	5.644	8.422	7.788	8.616
Conclusion	Reject H_0	Reject H_0	Reject h_0	Reject H_0	Reject H_0

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**