SUPPORT VECTOR MACHINE AND MAXIMUM LIKELIHOOD APPROACHES TO F-MEASURE OPTIMIZATION

Trevor Rose

Supervisor: A/ Prof. Spiridon Penev

School of Mathematics,
The University of New South Wales.

June 2014

Submitted in partial fulfillment of the requirements of the degree of
Bachelor of Science with Honours

Acknowledgements

Thanks most of all must go to my supervisor A/ Prof. Spiridon Penev, for consistently devoting his time and expertise in abundance to support my work over the past year. A huge thank you also to Dr. Georgi Dimitroff for providing his direct supervision and philosophy in the maximum likelihood components of this thesis. Indeed, I raise my glass to Borislav Popov, Dr. Georgi Georgiev and the rest of the Semantic Annotation and Search group at Ontotext for welcoming me and my work into the team. Additional thanks to Dr. Guoyin Li and Prof. V. Jeyakumar for their advice on optimization problems.

My immense gratitude goes out to my family and friends, who provided distraction and motivation in precisely the optimal ratio during the writing of this thesis. And thanks to all the poor souls who asked me about my thesis at parties and were met with an answer more detailed and convoluted than they could have possibly imagined. I hope that this document will remedy all of their confusion.

Trevor Rose, 6 June 2014.

Abstract

Most standard methods of classifying data are designed to optimize accuracy. In this thesis, we consider classifiers that are adjusted instead to optimize the F-measure, allowing a more flexible approach to classification, and in particular, providing a more reliable assessment in the presence of class imbalance. We formulate both the F-measure optimizing support vector machine with a new method of implementation, and the F-measure optimizing maximum entropy classifier, as well as adapting the latter to a new framework of uncertainty, where classes are not necessarily deterministic but observed with some noise given by a probability distribution. Our theoretical findings are put into practice on some synthetic and real-world datasets.

Contents

Chapter 1

Introduction

A linear binary classifier is a linear function "trained" to classify objects in $\mathbb{R}^n$ into one of two categories (which we refer to arbitrarily as "positive" and "negative") on the basis of some given instructive data. Once a classifier is defined, the natural question arises of how best to evaluate its performance. Usually, we consider evaluating performance both with regard to the training data and on an unseen test set, in order to have information about the classifier's ability to explain observed patterns and to predict new ones. In either case, there are a number of measures of performance that are useful to consider.

Standard classifiers tend to approximately optimize the measure of accuracy, which is the proportion of correctly classified training examples. Clearly, high accuracy is preferable. However, it is not generally the sole aim of classification to obtain optimal accuracy, particularly in the presence of unbalanced data. Whether one class contains more training examples than the other, or whether we have a preference about the performance of the classifier on one class over the other, and at the same time realizing that there could be many possible notions of performance, accuracy maximization is unlikely to be sufficient to return an informative classifier. In the former case, we may give the example in [20] of 1% of training examples belonging to the positive class and 99% of training examples belonging to the negative class. In situations where such extreme class imbalance exists, accuracy maximization is likely to yield a trivial classifier that erroneously predicts all data as belonging exclusively to the negative class. Scenarios of heavy class imbalance are not uncommon in practice, with one such instance given in [27], the problem of detecting underwater mines and distinguishing them from other mine-like objects. Even in the presence of a less extreme imbalance, accuracy is not as informative a measure of performance as we would ideally consult.

In this thesis, we focus on optimization of the F-measure, which is one of the main alternatives to accuracy. F-measure is a type of weighted average of two other important measures of fit, precision and recall. Roughly speaking, precision measures the correctness of classification into the positive class, and recall measures the effectiveness of the classifier in retrieving as many as possible out of the total correct instances into that class. There is a trade-off between classifying very precisely and with high recall, which is captured by a parameter β in the definition of the F-measure. This flexibility of the F-measure between emphasis on correctness and retrieval ability of predictions is another of its strengths over more rudimentary measures of performance such as accuracy.

There are as many applications of F-measure optimization as there are applications of classification techniques. In particular, the F-measure is heavily used as

a measure of performance in the natural language processing domain. As noted in [8], high precision may be required in automatic summarization or machine translation problems and high recall may be required in information retrieval systems, e.g. search engines. We will not focus too much on such applications, but instead we will devise and test methods of F-measure optimization to be applied practically at some later stage. In particular, we explore F-measure optimization of two important and widespread types of classifier – support vector machines and maximum entropy classifiers. Similarly to the F-measure itself, these two classifiers are applicable to an incredibly diverse range of classification problems. Support vector machines have the advantage that they are easily generalizable to nonlinear classification, and maximum entropy classifiers have the advantage that they are more naturally extended to the multi-class case (involving more than two classes). However, in general, we cannot consider one method to be superior to the other, and theoretical justification is provided for this by the No Free Lunch theorem, discussed for example in [9, p. 454]. Thus, it is also not our focus to study comparatively the classifiers we present.

Support vector machines make predictions by defining a linear hyperplane that maximizes the margin (distance) between the clouds of training data that belong to different classes. This is typically formulated in an optimization problem that in addition allows some classification errors to occur (i.e. does not assume that the data is strictly linearly separable) while penalizing them at the user's discretion. Usually, misclassification distance is minimized, but by modifying this error metric, we may formulate an F-measure maximizing version of the support vector machine as in [20], which has the exploitable property of equivalence to another support vector machine variant that minimizes the approximate count of misclassifications, as opposed to their distances. We explore the possible advantages and disadvantages of this approach.

Maximum entropy classifiers aim to form a probabilistic classification model on the basis of as few assumptions as possible, and are fitted via maximizing the likelihood of the training sample having been observed. It is shown in [8] that there exists a maximum entropy classifier which is optimal with regard to F-measure, and a simple adaptive algorithm is devised to obtain it. We propose to generalize this work into a new framework of uncertainty, where classes are not considered to be deterministically assigned to training examples but instead follow probability distributions, such as the Bernoulli distributions typically returned by a maximum entropy classifier. The attempt to reproduce the randomness inherent in a training sample in the performance of a classifier will lead to us considering a new measure of performance that itself generalizes the F-measure to the uncertain setting. We then formulate a maximum entropy classifier that performs optimally with regard to this "noisy" F-measure, and we demonstrate and discuss this optimality and its implications.

The structure of our thesis is as follows. We start out with support vector machines in Part I. In Chapter 2, we introduce theoretically the support vector machine and the F-measure. Then in Chapter 3, we state and develop a few modifications to the standard support vector machine, exploring their properties along the way, with the goal of F-measure optimization. We move on to maximum entropy classification in Part II. In Chapter 4 we introduce the maximum likelihood-based maximum

entropy classifier and the corresponding F-measure maximizing algorithm. Then in Chapter 5, we introduce the uncertain likelihood, a counterpart to the standard likelihood for classes which are uncertain with a given distribution. In Chapter 6, we formulate the noisy F-measure, a more informative alternative to the F-measure in the presence of class randomness. We also give a proof that noisy F-measure maximization is equivalent to weighted uncertain likelihood maximization for a certain choice of weights. These weights can be chosen similarly to the deterministic setting via an adaptive algorithm, which we define and discuss in Chapter 7. The last part, Part III, containing Chapters 8 and 9, is devoted to implementing and demonstrating performance of the F-measure maximizing support vector machine and the noisy F-measure maximizing maximum entropy classifier respectively on some sample data.

Part I

Support Vector Machines

Chapter 2

Introduction to Support Vector Machines

In this chapter, we introduce the modern support vector machine classifier as first historically outlined in [26], and we discuss ways of measuring its performance, or indeed the performance of any classifier. We begin in Section 2.1 with some basic definitions and results that apply to support vector machines. For a good alternative introductory treatment of support vector machine theory, see [13, pp. 455–469]. Section 2.2 then introduces the notion of a measure of performance, and in particular we define here the F-measure.

2.1 Definitions and Properties

We set up the general problem of linear binary classification mathematically in the following way. Suppose we have a training set $\{(x_i, y_i) : i = 1, \ldots, m\}$, with $x_i \in \mathbb{R}^n$ being training examples, and $y_i = \pm 1$ being their corresponding classes. The aim of linear classification is to provide a linear function that assigns classes to any point x on the basis of the observed data. We consider, then, decision functions of the form:

$$f(x) = \text{sign}(x^T w + b).$$

That is, depending on which side of the hyperplane $x^T w + b = 0$ the point x lies on, it is classified either into the positive or negative class by the sign function.

The primary objective of support vector machine methodology is to choose this separating hyperplane parameterized by (w, b) such that geometrically the distance, or margin, between the clouds of observed classes is as large as possible. This involves maximizing the distance of the hyperplane from both positive and negative classes, denoted d_+ and d_- respectively. Indeed, defining so-called canonical hyperplanes $x^T w + b = \pm 1$ to represent the boundary of the margin, we have that $d_+ = d_- = 1/\|w\|$, and our objective becomes to maximize

$$d_+ + d_- = \frac{2}{\|w\|},$$

in other words simply to minimize $\|w\|$, or in practice, its square.

Let us first assume that the training data are strictly linearly separable, that is, there is a linear classifier which is able to perfectly separate points belonging to the positive and negative class. Then we have the condition that all observations must lie on either side of the margin, i.e.

$$x^T w + b \geq +1, \text{ for } y_i = +1$$
$$x^T w + b \leq -1, \text{ for } y_i = -1,$$

which is more conveniently expressed as:

$$y_i(x_i^T w + b) \geq 1, \quad \forall i. \tag{2.1}$$

The points that lie on the boundary of the margin, i.e. have equality in (2.1), are termed support vectors. Indeed, the margin is defined uniquely by support vectors.

In practice, the assumption that the training data are strictly linearly separable tends to be too rigid. Thus, the version of support vector machine classification we employ today does not make such an assumption. It is known as the standard linear or "soft margin" support vector machine, and was introduced in [26]. The idea is to allow violations of strict separation at a cost C to be determined by the user. These violations are represented by misclassification distances $\xi \in \mathbb{R}^m$, the 1-norm of which we tend to minimize (although as we will see in Chapter 3, there are many choices for an error metric).

The standard linear support vector machine is formulated as follows, incorporating margin maximization and misclassification distance minimization in the objective function, as well as (2.1) and the condition that misclassification distance is nonnegative as constraints.

Definition 2.1.1. The standard linear support vector machine is the following optimization problem:

$$\begin{aligned} \min_{(w,b,\xi)\in\mathbb{R}^{n+1+m}} \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y_i(x_i^T w + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{2.2}$$

To provide a simpler way to solve this optimization problem in practice, we note that it involves minimizing a convex function over a convex set, and is thus a convex optimization problem. There are a well-known set of first-order optimality conditions defined in [12], the Karush-Kuhn-Tucker (KKT) conditions, which, under simple regularity conditions on the constraints which are here satisfied, are both necessary and sufficient for the solution of a convex optimization problem. To transform (2.2) into a "dual" formulation which exploits this fact, we first introduce Langrange multipliers and form the relevant Lagrange function.

The Lagrange function for the "primal" problem is:

$$\mathcal{L}_P(w,b,\xi) = \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\{y_i(x_i^T w + b) - 1 + \xi_i\} - \sum_{i=1}^{m}\mu_i\xi_i,$$

where $\alpha_i \geq 0$ and $\mu_i \geq 0$ are Lagrange multipliers. Minimization of this function is equivalent to solving (2.2).

The KKT optimality conditions are:

$$\frac{\partial \mathcal{L}_P}{\partial w} = 0 : w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0 \tag{2.3}$$

$$\frac{\partial \mathcal{L}_P}{\partial b} = 0 : \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.4}$$

$$\frac{\partial \mathcal{L}_P}{\partial \xi_i} = 0 : C - \alpha_i - \mu_i = 0 \tag{2.5}$$

with conditions on the Lagrange multipliers:

$$\alpha_i \geq 0 \tag{2.6}$$

$$\mu_i \geq 0 \tag{2.7}$$

$$\alpha_i\{y_i(x_i^T w + b) - 1 + \xi_i\} = 0 \tag{2.8}$$

$$\mu_i \xi_i = 0 \tag{2.9}$$

Performing some algebra on these conditions and the Lagrangian itself, we reformulate the primal problem (2.2) as the following dual problem:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^m} \quad & \mathcal{L}_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{2.10}$$

The form of (2.10) makes it easier to solve and more informative than (2.2) for the following reasons. In general, w is high-dimensional, so it is significant that now we are optimizing over $\alpha \in \mathbb{R}^m$ as opposed to $(w, b, \xi) \in \mathbb{R}^{n+1+m}$. Indeed, whenever support vector machines are used for nonlinear classification, we will see shortly that w is possibly so high-dimensional that we prefer not to, or cannot, calculate it explicitly. We finally note that most α_i's are 0, since we only have $\alpha_i > 0$ when the point x_i is either misclassified or lies on the boundary of the margin. This property provides computational benefits for the dual formulation (2.10) and an informative solution which explicitly gives these "support vectors" that define the resulting linear classifier.

Once the optimal α^* has been found, we can use the KKT condition (2.3) to obtain

$$w^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i,$$

and (2.8) to derive one possible expression for b^*,

$$b^* = y_i - \sum_{j=1}^{m} \alpha_j^* y_j x_i^T x_j, \quad \forall i \text{ such that } 0 < \alpha_i < C.$$

As found for example in [3], for numerical purposes we tend to take b^* to be the average over all training points that satisfy this condition. Then (w^*, b^*) defines uniquely the linear separating hyperplane $f^*(x) = \text{sign}(x^T w^* + b^*)$.

Support vector machines are easily, though subtly, generalized to the case of nonlinear classification, where it is not necessarily a linear classification rule we wish to devise, but instead potentially a much more complicated and nonlinear one, depending on the way that the data to be classified is distributed. Though nonlinear classification via support vector machines is not our focus, we outline briefly this important extension. Since support vector machines are naturally linear classifiers, we consider performing the transformation $\Psi : \mathbb{R}^n \to \mathbb{H}$ on the training set which maps it into some usually high-dimensional space $\mathbb{H}$ where linear classification is possible. In the form of the dual problem (2.10), this simply amounts to replacing examples with the output of this transformation, namely modifying the objective: $\mathcal{L}_D(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \Psi(x_i)^T \Psi(x_j)$. We can simplify this expression. If a kernel function can be defined such that $K(x_i, x_j) = \Psi(x_i)^T \Psi(x_j)$, then we may replace accordingly the scalar product in $\mathcal{L}_D$ with its kernel representation. Mercer's condition, defined in [17], gives necessary and sufficient conditions for a symmetric function $K(x_i, x_j)$ to be such a kernel, namely that we must have positive definiteness, or:

$$\sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j K(x_i, x_j) \geq 0$$

for any $\lambda \in \mathbb{R}^m$.

Replacing a scalar product in $\mathbb{H}$ with its kernel representation allows us to save the computational cost of calculating a potentially very large-dimensional scalar product, and even more, alleviates us of the need to know the transformation Ψ explicitly. This is known as the "kernel trick". One example out of the many possible kernel functions to consider is the isotropic Gaussian kernel,

$$K(x_i, x_j; \sigma) = e^{-\|x_i - x_j\|/2\sigma^2}.$$

We typically introduce an extra parameter in the kernel, such as σ in the above, in order to provide generality and thereby the ability to reproduce as faithfully as possible the scalar product in $\mathbb{H}$. This parameter, as with all parameters in the support vector machine problem, must usually be optimized in practice via the computational technique of cross-validation, which will be discussed in Chapter 8. Once we have solved the nonlinear dual problem, we form the nonlinear decision function

$$f^*(x) = \text{sign}\left(\sum_{j=1}^m \alpha_j^* y_j K(x, x_j) + b^*\right).$$

2.2 The F-measure

After defining and fitting a classifier, the question remains of how to evaluate its performance. Most measures of performance are calculated from a "confusion matrix", which is a table of true and false classification decisions made by the model. An ideal model, of course, will make many more correct choices than wrong ones, and this idea is captured in different ways by different measures of performance.

The following confusion matrix assigns examples as TP, FN, FP or TN depending on the class they actually belong to and the class they are assigned by a given classifier. Specifically, the entries in the matrix are the tallies of true positives, true negatives, false positives and true negatives over all classified examples.

		Actual class	
		+1	**-1**
Predicted class	**+1**	True Positive (TP)	False Positive (FP)
	-1	False Negative (FN)	True Negative (TN)
	Total	m_+	m_-

Using this notation, accuracy is given by the formula

$$acc = \frac{TP + TN}{m},$$

where $m = m_+ + m_-$ is again the number of training examples provided. We can see from this formula that if one class, say the negative class, is vastly over-represented, then for a trivial classifier that assigns all possible examples to the negative class, the lack of true positives will contribute proportionally very little to acc, resulting in a high accuracy value for an uninformative classifier. Even if the over-representation is not vast, imbalance is simply not taken into account. This is one of the main reasons why accuracy is often insufficient to assess the performance of a classifier.

The standard support vector machine (2.2) optimizes approximate test set accuracy by minimizing the number of misclassified points in the training set, via minimizing misclassification distances ξ. While high accuracy is clearly desirable, it is not indicative of a "good" classifier in general, which motivates defining alternative measures of performance that overcome its deficiencies.

An important type of measures of classification performance are those which are functions of precision and recall. Precision and recall are defined in the following way:

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

Precision is the fraction of examples classified as positive which are truly positive out of all examples classified as positive. It is a measure of how well our classifier is assigning the positive class to positive examples. Recall is the fraction of examples classified as positive which are truly positive out of all examples which are truly positive. It is a measure of how well our classifier is retrieving positive instances

from the entire training sample. The precision/recall tradeoff is captured in the F-measure, from [25] and defined as follows.

Definition 2.2.1. The F-measure is the weighted harmonic mean of precision and recall,

$$F_\beta = \left(\frac{\beta}{P} + \frac{1-\beta}{R}\right)^{-1},$$

with parameter $\beta \in [0, 1]$.

In [25] and much of the literature, the parameter β is instead called α. We go against this convention both in keeping with notation in [8] and [23], and to avoid confusion with support vector machine coefficients α. Using our notation, the most common F-measure is the unweighted harmonic mean of P and R,

$$F_{0.5} = \frac{2PR}{P+R}.$$

There exist other averages between precision and recall, most notably the geometric mean, or G-measure, $G = \sqrt{PR}$. The primary reason to favour the F-measure is that it allows for a customizable tradeoff between precision and recall, based on the parameter β. We may choose an equal weighting of precision and recall, or instead favour one over the other, depending on the application that interests us.

Chapter 3

Support Vector Machine Variants for F-measure Optimization

The purpose of this chapter is to introduce the reader to the problem of F-measure optimization in the support vector machine context, and to build on the groundwork of Chapter 2 by varying traditional methodology to suit this need. This chapter begins with Section 3.1 and the concept of a support vector machine that minimizes a count of misclassifications instead of the sum of their distances. Section 3.2 introduces the F-measure maximizing support vector machine and provides a result linking this formulation to the misclassification counting support vector machine. Finally in Section 3.3 we discuss further the properties of the F-measure maximizing support vector machine, in particular providing a dual problem.

3.1 The Misclassification Counting Support Vector Machine

A natural extension of the standard support vector machine is its weighted version, which allows us to weight more heavily the error contribution from the rarer class. In keeping with our observations that standard support vector machines approximately optimize accuracy, and that this may not provide an informative classifier, we are able via weighting to compensate for class imbalance.

Definition 3.1.1. The weighted standard linear support vector machine is the following optimization problem for a vector of penalty weights c:

$$\begin{aligned} \min_{(w,b,\xi)\in\mathbb{R}^{n+1+m}} \quad & \frac{1}{2}w^T w + \sum_{i=1}^{m} c_i \xi_i \\ \text{s.t.} \quad & y_i(x_i^T w + b) \geq 1 - \xi_i \\ & \xi_i \geq 0. \end{aligned} \tag{3.1}$$

The general formulation (3.1) can be further specified for our purposes. We only require class-wise weights, and do not ordinarily have any need to weight differently across the same class. So we restrict $c \in \mathbb{R}^m$ in the following way.

$$c_i = \begin{cases} C_+ & \text{if } y_i = +1 \\ C_- & \text{if } y_i = -1 \end{cases} \tag{3.2}$$

This gives us the objective function

$$f(w, b, \xi) = \frac{1}{2} w^T w + C_+ \sum_{i:y_i=+1} \xi_i + C_- \sum_{i:y_i=-1} \xi_i.$$

We can see from this expression that if for example the positive class is rare, we should choose $C_+ > C_-$ in order to compensate for the class imbalance, and this is precisely what practitioners tend to do. Indeed, a common heuristic to use in the weighted standard formulation is to balance C_+ and C_- so that the errors for both classes contribute equally. That is, choose C_+ and C_- such that:

$$\frac{C_+}{C_-} = \frac{m_-}{m_+}. \tag{3.3}$$

There is no adequate theory for the choice of C in the standard support vector machine, let alone the specific choice of C_+ and C_- in the weighted version.

The dual problem for the weighted standard support vector machine is derived via the KKT conditions in a similar manner to the unweighted case. It is as follows.

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^m} \quad & \mathcal{L}_D(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq c_i \\ & \sum_{i=1}^m \alpha_i y_i = 0. \end{aligned}$$

As always, the scalar product $x_i^T x_j$ can be replaced with the kernel $K(x_i, x_j)$ for the purpose of nonlinear classification.

We turn our attention now to a further extension of standard support vector machines, given first in [20]. Minimization of the sum of misclassification distances ξ is performed in the standard support vector machine only as a proxy to minimization of the number of misclassified points, as noted in [26]. That is, instead of minimizing $\sum_{i=1}^m \xi_i$, in an ideal world we would prefer to minimize $\sum_{i=1}^m (\xi_i)_*$. Here we have denoted the step function:

$$(\xi_i)_* = \begin{cases} 1 & \text{if } \xi_i > 0 \\ 0 & \text{otherwise.} \end{cases}$$

However, this variety of optimization problem is difficult to solve, in particular due to the non-differentiability introduced. We therefore consider applying a smooth step function approximation s to the misclassifications, and minimizing instead $\sum_{i=1}^m s(\xi_i)$. When we additionally apply the strategy of weighting from (3.1), we arrive at the following.

Definition 3.1.2. The weighted misclassification counting support vector machine is the following optimization problem:

$$\begin{aligned} \min_{(w,b,\xi)\in\mathbb{R}^{n+1+m}} \quad & \frac{1}{2}w^T w + \sum_{i=1}^{m} c_i s(\xi_i) \\ \text{s.t.} \quad & y_i(x_i^T w + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \end{aligned} \tag{3.4}$$

where s is a differentiable approximation to the step function.

There are a few choices for the step function approximation s. Note that if we let $s(\xi_i) = \xi_i$, then we have again the standard linear support vector machine. However, there exist better approximations. We suggest a transformation based on the sigmoid function.

$$s(\xi_i; a) = 2\left(\frac{1}{1+e^{-a\xi_i}} - \frac{1}{2}\right),$$

for $a > 0$. Note that $s(0) = 0$ and $\lim_{\xi_i\to\infty} s(\xi_i) = 1$. The larger the parameter a, the more closely s approximates the step function.

3.2 The F-measure Maximizing Support Vector Machine

We now return to the F-measure in an attempt to formulate another support vector machine variant that optimizes it. Focusing on the tallies in the confusion matrix, we can express F_β in the following way:

$$F_\beta = \left(\frac{\beta}{P} + \frac{1-\beta}{R}\right)^{-1} = \frac{TP^2}{TP^2 + \beta TP \cdot FP + (1-\beta)TP \cdot FN}$$

Simplifying this expression and using $TP = m_+ - FN$, we arrive at:

$$F_\beta = \frac{1}{1 + \frac{\beta FP + (1-\beta)FN}{m_+ - FN}}.$$

Maximizing F_β is thus equivalent to minimizing

$$\frac{\beta FP + (1-\beta)FN}{m_+ - FN}, \tag{3.5}$$

which is a function purely of misclassification counts, or $(\xi_i)_*$'s in support vector machine notation. This suggests that instead of attempting to minimize misclassification distances or even misclassification counts within the support vector machine program, we should attempt to minimize (3.5) if our interest is F-measure optimization.

A simple extension of the F-measure maximizing support vector machine in [20] (where only the $F_{0.5}$ maximizing support vector machine is defined) is therefore given by the following definition. Again it is necessary to approximate the step function with a differentiable function s.

Definition 3.2.1. The F-measure maximizing linear support vector machine is the following optimization problem:

$$\begin{aligned} \min_{(w,b,\xi)\in\mathbb{R}^{n+1+m}} \quad & \frac{1}{2}w^T w + C\frac{\beta\sum_{i:y_i=-1} s(\xi_i) + (1-\beta)\sum_{i:y_i=+1} s(\xi_i)}{m_+ - \sum_{i:y_i=+1} s(\xi_i)} \\ \text{s.t.} \quad & y_i(x_i^T w + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \end{aligned} \tag{3.6}$$

This formulation, though not a nonsmooth optimization problem, is highly nonlinear and difficult to solve. The following theorem from [20], in which the proof is provided without loss of generality for $F_{0.5}$, greatly simplifies the problem of F-measure optimization. To prove the result, equivalence is drawn between the KKT optimality conditions for (3.6) and (3.4).

Theorem 3.2.2. *Given a parameter C, an optimal separating surface found by the F-measure maximizing support vector machine* (3.6) *is also an optimal separating surface for the misclassification counting support vector machine* (3.4) *for an appropriate choice of parameters C_+ and C_- (contained in c, the vector of penalty weights defined in* (3.2)*).*

It is therefore unnecessary to solve the F-measure optimizing support vector machine (3.6) directly in order to obtain it. Instead, we may solve the misclassification counting formulation (3.4) for the correct choice of C_+ and C_-. Note that Theorem 3.2.2 still does not give us information about how to obtain these parameters, instead expressing them as functions of ξ. We must rely on numerical techniques or manual tweaking to provide a satisfying outcome.

A consequence of Theorem 3.2.2 is that the weighted standard support vector machine (3.1) is equivalent to the F-measure optimizing support vector machine (3.6) where a linear error metric is used. We can see this by allowing $s(\xi_i) = \xi_i$. This indicates that the weighted standard support vector machine does approximately optimize F-measure for the right choice of parameters, despite the possible existence of better approximations not using the linear error metric. The advantage of the weighted standard support vector machine over the more complicated formulations involving a step function approximation to count errors is that it is a convex quadratic optimization problem for which a dual problem is easily derived and standard methods exist of solving it. Indeed, this is why in [20], the weighted standard approach is favoured for F-measure optimization. We pose the question of whether there would be a measurable gain in classification performance by using the step function approximation s, which we will attempt to answer experimentally in Chapter 8. This will provide either a justification for the conclusions of [20], or motivation to attempt to push beyond this view.

3.3 Further Properties of the F-measure Maximizing Support Vector Machine

We now investigate the solution of the F-measure maximizing support vector machine (3.6) by exploring some of its further properties and stating a dual problem which solves the misclassification support vector machine (3.4) under a certain

condition, which itself is difficult to check. If this condition could be shown to be satisfied, the implications for F-measure maximization with support vector machines would be significant, and in particular it would allow us the same benefits we enjoy from deriving the dual problem in the standard case, namely: dimensionality reduction, other important computational benefits and an extension to nonlinear classification.

We first note that the misclassification counting support vector machine (3.4) is not a convex optimization problem. This is because the objective function $f(w, b, \xi) = \frac{1}{2}w^T w + \sum_{i=1}^{m} c_i s(\xi_i)$ is the sum of a convex function and m functions which are concave on the feasible region, that is, the set defined by the constraints. There are several generalizations of convexity which we may appeal to in order to retain such desirable properties as sufficiency of KKT conditions, duality, and even simply that local minimization implies global minimization. We first mention casually the types of generalized convexity that our problem (3.4) does not exhibit: quasiconvexity and pseudoconvexity. While for our purposes their definitions are not necessary, we defer the curious reader to [1] for the full story. Indeed, we note simply that convexity implies pseudoconvexity, which implies quasiconvexity. This is illustrated helpfully in [2]. It is shown in [5] that for a sum of functions to be quasiconvex, it is necessary that all but (at most) one of these functions be convex, thus f is not quasiconvex. We know then that it is not pseudoconvex either, so we need not consider these notions further.

However, there is another generalization of convex functions to "invexity", which we now define. A thorough treatment of invexity in optimization is to be found in [18]. We observe the following definition.

Definition 3.3.1. Assume $X \subseteq \mathbb{R}^n$ is an open set. The differentiable function $f : X \to \mathbb{R}$ is invex if there exists a vector function $\eta : X \times X \to \mathbb{R}^n$ such that:

$$f(x) - f(y) \geq \eta(x, y)^T \nabla f(y), \quad \forall x, y \in X.$$

We also have the following two results from [18, p. 16], where the proofs may additionally be found.

Theorem 3.3.2. *Let $f : X \to \mathbb{R}$, $g : X \to \mathbb{R}$ be invex. A common η, with respect to which both f and g are invex, exists if and only if $\forall x, y \in X$ either:*

1. *$\nabla f(y) \neq \lambda \nabla g(y)$ for any $\lambda > 0$, or*
2. *$\nabla f(y) = -\lambda \nabla g(y)$ for some $\lambda > 0$ and $f(x) - f(y) \geq -\lambda[g(x) - g(y)]$.*

Theorem 3.3.3. *Let $f : X \to \mathbb{R}$, $g_1, \ldots, g_m : X \to \mathbb{R}$ be invex. A common η, with respect to which $f, g_1, \ldots, g_m$ are invex, exists if and only if $f + \lambda_1 g_1 + \cdots + \lambda_m g_m$ is invex for all $\lambda_1, \ldots, \lambda_m > 0$.*

In [18, p. 74], the sufficiency of KKT optimality conditions, which we used in Chapter 2 to derive the dual problem for the standard support vector machine, is extended to hold for invex optimization problems, which are optimization problems where the objective function and constraints are all invex with respect to the same η. We now assert the following.

Proposition 3.3.4. *The objective and constraints of the weighted misclassification counting support vector machine* (3.4)*, with*

$$s(\xi_i; a) = 2\left(\frac{1}{1+e^{-a\xi_i}} - \frac{1}{2}\right)$$

being a differentiable approximation to the step function, are all invex functions.

Proof. We write the objective function and constraints of (3.4) in standard form as follows:

$$\begin{aligned} f(w,b,\xi) &= \frac{1}{2}w^T w + \sum_{i=1}^{m} c_i s(\xi_i) \\ g_{i1}(w,b,\xi) &= 1 - \xi_i - y_i(x_i^T w + b) \\ g_{i2}(w,b,\xi) &= -\xi_i \end{aligned}$$

We know already that g_{i1} and g_{i2} are convex functions, as they are the standard support vector machine constraints. Since invexity is simply a generalization of convexity, we have that g_{i1} and g_{i2} are invex.

To see that f is invex too, we first decompose $f = f_1 + f_2$, where $f_1(w,b) = \frac{1}{2}w^T w$ and $f_2(\xi) = \sum_{i=1}^{m} c_i s(\xi_i)$. Clearly f_1 is convex, and therefore invex. For f_2, we have:

$$f_2(x) - f_2(y) = \sum_{i=1}^{m} c_i(s(x_i) - s(y_i)),$$

and

$$\nabla f_2(y) = (c_1 s'(y_1), \ldots, c_m s'(y_m))^T.$$

We may define

$$\eta(x,y) = \left(\frac{s(x_1) - s(y_1)}{s'(y_1)}, \ldots, \frac{s(x_m) - s(y_m)}{s'(y_m)}\right)^T,$$

since the derivative of s is always positive. Then

$$\eta(x,y)^T \nabla f_2(y) = f_2(x) - f_2(y) \quad \forall x, y \in \mathbb{R}^m,$$

thus f_2 satisfies the definition of invexity.

We now show that there exists a common η with respect to which f_1 and f_2 are both invex by Theorem 3.3.2. It then follows directly from Theorem 3.3.3 that f is invex. To apply Theorem 3.3.2, we take gradients of f_1 and f_2:

$$\begin{aligned} \nabla f_1(w,b,\xi) &= (w_1, \ldots, w_n, 0, 0, \ldots, 0)^T \\ \nabla f_2(w,b,\xi) &= (0, \ldots, 0, 0, c_1 s'(\xi_1), \ldots, s'(\xi_m))^T. \end{aligned}$$

Clearly $\nabla f_1(w,b,\xi) \neq \lambda \nabla f_2(w,b,\xi)$ for any $\lambda > 0$. Thus there exists a common η with respect to which both f_1 and f_2 are invex, and consequently f is itself invex. □

Note that we have only shown that the objective function and constraints of (3.4) are invex, and not necessarily that they are invex with respect to the same η. This condition is difficult to check, and it is not even known whether it holds. However, given that the objective and constraints are invex, there is some hope that we do have an invex optimization problem, and if this is the case, we have sufficient and KKT conditions. [18, p. 90] observes that duality results follow such that we could reformulate (3.4) in an equivalent dual problem. This dual problem is the following.

Theorem 3.3.5. *The weighted misclassification counting support vector machine* (3.4)*, with*

$$s(\xi_i; a) = 2\left(\frac{1}{1+e^{-a\xi_i}} - \frac{1}{2}\right)$$

being a differentiable approximation to the step function, admits the dual problem under the assumption of invexity:

$$\begin{aligned}
\max_{(\alpha,\mu)\in\mathbb{R}^{2m}} \quad & \mathcal{L}_D(\alpha,\mu) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j + \frac{1}{a}\sum_{i=1}^{m}\left\{\alpha_i\left[1+e^{a\left[1-y_i(x_i^T w(\alpha)+b(\alpha,\mu))\right]}\right]+2\mu_i\right\} \\
\text{s.t.} \quad & \alpha_i, \mu_i \geq 0 \\
& \sum_{i=1}^{m}\alpha_i y_i = 0 \\
& \alpha_i\left\{\frac{2ac_i e^{-a\left[1-y_i(x_i^T w(\alpha)+b(\alpha,\mu))\right]}}{\left(1+e^{-a\left[1-y_i(x_i^T w(\alpha)+b(\alpha,\mu))\right]}\right)^2} - \alpha_i - \mu_i\right\} = 0,
\end{aligned}$$

where

$$w(\alpha) = \sum_{i=1}^{m}\alpha_i y_i x_i$$

and

$$b(\alpha,\mu) = \frac{\sum_{i=1}^{m}\alpha_i\mu_i y_i - \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\mu_i y_j x_i^T x_j}{\sum_{i=1}^{m}\alpha_i\mu_i}.$$

Proof. Let

$$s(\xi_i; a) = 2\left(\frac{1}{1+e^{-a\xi_i}} - \frac{1}{2}\right).$$

With this differentiable approximation to the step function, the weighted approximate misclassification counting support vector machine becomes:

$$\begin{aligned}
\min_{(w,b,\xi)\in\mathbb{R}^{n+1+m}} \quad & \frac{1}{2}w^T w + \sum_{i=1}^{m}\frac{2c_i}{1+e^{-a\xi_i}} \\
\text{s.t.} \quad & y_i(x_i^T w + b) \geq 1 - \xi_i \\
& \xi_i \geq 0.
\end{aligned}$$

The Lagrange function to minimize is therefore:

$$\mathcal{L}_P(w,b,\xi) = \frac{1}{2}w^T w + \sum_{i=1}^{m} \frac{2c_i}{1+e^{-a\xi_i}} - \sum_{i=1}^{m} \alpha_i\{y_i(x_i^T w + b) - 1 + \xi_i\} - \sum_{i=1}^{m} \mu_i \xi_i,$$

where $\alpha_i, \mu_i \geq 0$ are Lagrange multipliers.

The KKT conditions are as follows:

$$\frac{\partial \mathcal{L}_P}{\partial w} = 0 : w = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{3.7}$$

$$\frac{\partial \mathcal{L}_P}{\partial b} = 0 : \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{3.8}$$

$$\frac{\partial \mathcal{L}_P}{\partial \xi_i} = 0 : \frac{2ac_i e^{-a\xi_i}}{(1+e^{-a\xi_i})^2} - \alpha_i - \mu_i = 0 \tag{3.9}$$

$$\alpha_i, \mu_i \geq 0 \tag{3.10}$$

$$\alpha_i\{y_i(x_i^T w + b) - 1 + \xi_i\} = 0 \tag{3.11}$$

$$\mu_i \xi_i = 0 \tag{3.12}$$

Under the assumption of invexity, the KKT conditions are necessary and sufficient for an optimal solution. We can now find the Lagrangian for the dual problem. Similarly to the case of the standard support vector machine, we use the KKT conditions to derive:

$$\begin{aligned}\mathcal{L}(w,b,\xi,\alpha,\mu) &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{m} \left\{ \frac{2c_i}{1+e^{-a\xi_i}} - \alpha_i \xi_i - \mu_i \xi_i \right\} \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j + 2\sum_{i=1}^{m} \frac{c_i}{1+e^{-a\xi_i}} - \sum_{i=1}^{m} \alpha_i \xi_i \quad \text{(by (3.12))}.\end{aligned}$$

Here we have $\mathcal{L}$ as a function of α and ξ. To formulate a dual problem that maximizes a function only of Lagrange multipliers α and μ, we must eliminate ξ in the above by expressing all terms involving ξ in the Lagrange multipliers instead. Let us first examine the last sum.

Rearranging (3.11) and taking the sum over all i, we have:

$$\sum_{i=1}^{m} \alpha_i \xi_i = \sum_{i=1}^{m} \alpha_i \left[1 - y_i(x_i^T w + b)\right],$$

where (3.7) gives us directly an appropriate expression for w,

$$w(\alpha) = \sum_{i=1}^{m} \alpha_i y_i x_i,$$

and we derive a similar expression for b in terms of the Lagrange multipliers in the following way.

Multiplying (3.11) by μ_i, we have

$$\alpha_i \mu_i \left[y_i (x_i^T w + b) - 1 + \xi_i \right] = 0.$$

Using (3.12) and rearranging, we obtain

$$\alpha_i \mu_i y_i b = \alpha_i \mu_i (1 - y_i x_i^T w).$$

Multiplying both sides by y_i and using the fact that $y_i^2 = 1$,

$$\alpha_i \mu_i b = \alpha_i \mu_i (y_i - x_i^T w).$$

Now taking sums, rearranging and using (3.7), we obtain:

$$b(\alpha, \mu) = \frac{\sum_{i=1}^m \alpha_i \mu_i y_i - \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j \mu_i y_j x_i^T x_j}{\sum_{i=1}^m \alpha_i \mu_i}.$$

This allows us to express the final sum in $\mathcal{L}$ solely in terms of Lagrange multipliers:

$$\sum_{i=1}^m \alpha_i \xi_i = \sum_{i=1}^m \alpha_i \left[1 - y_i \left(x_i^T w(\alpha) + b(\alpha, \mu) \right) \right].$$

Now let us examine the third sum in $\mathcal{L}$. Rearranging (3.9), we obtain

$$\frac{c_i}{1 + e^{-a\xi_i}} = \frac{1}{a} \left(1 + e^{a\xi_i} \right) (\alpha_i + \mu_i).$$

Then taking sum over all i, we find that

$$\sum_{i=1}^m \frac{c_i}{1 + e^{-a\xi_i}} = \frac{1}{a} \left\{ \sum_{i=1}^m \alpha_i e^{a\xi_i} + \sum_{i=1}^m \mu_i e^{a\xi_i} + \sum_{i=1}^m (\alpha_i + \mu_i) \right\}.$$

To eliminate ξ from this expression, let us examine (3.11) and (3.12). By (3.12), if $\alpha_i = 0$ then $\xi_i = 0$. This is because, by (3.9), whenever $\alpha_i = 0$, $\mu_i > 0$. Also by (3.11), if $\alpha_i > 0$ then $\xi_i = 1 - y_i(x_i^T w + b)$. This allows us to state that:

$$\xi_i = \begin{cases} 0 & \text{if } \alpha_i = 0 \\ 1 - y_i \left(x_i^T w(\alpha) + b(\alpha, \mu) \right) & \text{if } \alpha_i > 0. \end{cases} \tag{3.13}$$

Then, applying an exponential and multiplying by α_i, we obtain:

$$\alpha_i e^{a\xi_i} = \begin{cases} 0 & \text{if } \alpha_i = 0 \\ \alpha_i e^{a\left[1 - y_i \left(x_i^T w(\alpha) + b(\alpha, \mu) \right)\right]} & \text{if } \alpha_i > 0. \end{cases}$$

Taking the sum over all i, we obtain:

$$\sum_{i=1}^m \alpha_i e^{a\xi_i} = \sum_{i=1}^m \alpha_i e^{a\left[1 - y_i \left(x_i^T w(\alpha) + b(\alpha, \mu) \right)\right]}.$$

In a similar way from (3.13), and using (3.12), we also obtain:

$$\sum_{i=1}^{m} \mu_i e^{a\xi_i} = \sum_{i=1}^{m} \mu_i.$$

This gives us:

$$\sum_{i=1}^{m} \frac{c_i}{1+e^{-a\xi_i}} = \frac{1}{a} \sum_{i=1}^{m} \left\{ \alpha_i \left[1 + e^{a\left[1+y_i\left(x_i^T w(\alpha)+b(\alpha,\mu)\right)\right]} \right] + 2\mu_i \right\}.$$

Substituting what we have derived thus far into $\mathcal{L}$ and using (3.8),

$$\mathcal{L}_D(\alpha,\mu) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j + \frac{1}{a} \sum_{i=1}^{m} \left\{ \alpha_i \left[1 + e^{a\left[1+y_i\left(x_i^T w(\alpha)+b(\alpha,\mu)\right)\right]} \right] + 2\mu_i \right\}.$$

The dual problem is defined as $\max_{(\alpha,\mu)\in\mathbb{R}^{2m}} \mathcal{L}_D(\alpha,\mu)$ subject to KKT constraints, and gives the solution of the primal problem. It remains only to formulate constraints on (α,μ).

Our first constraint is (3.8), that is,

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$

Now we look for constraints which impose the cost c_i on the coefficients, equivalent to the box constraints in the standard support vector machine. We start with (3.9). By (3.13), we can express (3.9) in the following way:

$$\frac{2ac_i e^{-a\xi_i}}{(1+e^{-a\xi_i})^2} - \alpha_i - \mu_i = \begin{cases} \frac{1}{2}ac_i - \alpha_i - \mu_i & \text{if } \alpha_i = 0 \\ \dfrac{2ac_i e^{-a\left[1-y_i\left(x_i^T w(\alpha)+b(\alpha,\mu)\right)\right]}}{\left(1+e^{-a\left[1-y_i\left(x_i^T w(\alpha)+b(\alpha,\mu)\right)\right]}\right)^2} - \alpha_i - \mu_i & \text{if } \alpha_i > 0. \end{cases}$$

Multiplying by α_i, we find our set of box-like constraints to be:

$$\alpha_i \left\{ \frac{2ac_i e^{-a\left[1-y_i\left(x_i^T w(\alpha)+b(\alpha,\mu)\right)\right]}}{\left(1+e^{-a\left[1-y_i\left(x_i^T w(\alpha)+b(\alpha,\mu)\right)\right]}\right)^2} - \alpha_i - \mu_i \right\} = 0.$$

□

If invexity holds, it is now possible to solve the dual problem and thus find a numerical solution via a nonlinear optimization solver for the misclassification counting support vector machine in general, and, by Theorem 3.2.2, therefore for the F-measure maximizing support vector machine. It requires much further investigation to show that invexity holds, or if it does not, to either similarly derive a dual problem under different assumptions, to provide a different choice of step function approximation s such that convexity or a generalized convexity does hold,

in which case a dual problem would be derived in the above manner, or to focus on devising methods of solving the primal problem directly.

Part II

Maximum Entropy Classification

Chapter 4

Introduction to Maximum Entropy Classifiers

This chapter introduces the maximum entropy model of classification, based on maximum likelihood methods, and we give a summary of the work that has thus far been done on F-measure maximization in this context, which can primarily be found in [8]. We begin in Section 4.1 with some definitions and properties relating to the maximum entropy model. We then move onto F-measure optimization in Section 4.2, giving some key results and an algorithm to perform this task in practice.

4.1 Definitions and Properties

The maximum entropy modelling framework, similar to the framework we used for support vector machines, involves a training set $\{(x_i, y_i) : i = 1, \ldots, m\}$, where x_i are training examples or attributes and $y_i \in \mathcal{Y} = \{0, 1\}$ are their corresponding classes. We refer arbitrarily to the class with label 1 as the positive class and the class with label 0 as the negative class.

We present the general "feature" representation of the maximum entropy model, where each observation x_i is characterized by a set of N features $\{f_j(x_i, y_i) : j = 1, \ldots, N\}$. Features represent properties of generic observed objects x_i and in applications are often categorical, such as "male" or "female". These functions of the data are used to fit a probabilistic classification model that follows from the maximum entropy principle, which gives the form of model conditional probabilities $p(y|\, x, \lambda)$ for an example with attributes x to be of class y:

$$p(y|\, x, \lambda) = \frac{1}{Z_\lambda(x)} \exp(\lambda \cdot f(x, y)),$$

where $\lambda \in \mathbb{R}^N$ are model parameters and $Z_\lambda(x)$ is the corresponding partition function given by:

$$Z_\lambda(x) = \sum_{y \in \mathcal{Y}} \exp(\lambda \cdot f(x, y)).$$

Maximum entropy modelling is equivalent to logistic regression, as derived for example in [19]. The calibration of the model amounts to maximizing the likelihood, or log-likelihood, of the observations:

$$l(\lambda : x, y) = \sum_{i=1}^{m} \log p(y_i|\, x_i, \lambda).$$

We generalize this to a weighted log-likelihood, with a vector of weights $w \in \mathbb{R}^m$,

$$l^W(\lambda : w, x, y) = \sum_{i=1}^{m} w_i \log p(y_i|\, x_i, \lambda),$$

which we will make use of for the purpose of F-measure optimization in the next section.

4.2 F-measure Optimization

The natural aim of maximum entropy classification is to achieve high accuracy, similarly to support vector machines. Again we will often wish to aim for a high F-measure instead. To do this, we establish the equivalence between "expected" F-measure and weighted likelihood maximization, for a specific choice of weights.

Since we expect the model to "do well", and return true classification probabilities close to 1 and false classification probabilities close to 0, we may build up a standard notion of expected F-measure (though it is not actually an expectation) from an expected confusion matrix based on model probabilities. We here denote the positive class by c_1 and the negative class by c_0.

$$\begin{aligned}
\tilde{TP} &= \sum_{i:y_i=1} p(c_1|\, x_i, \lambda) \\
\tilde{FN} &= \sum_{i:y_i=1} p(c_0|\, x_i, \lambda) = \sum_{i:y_i=1} (1 - p(c_1|\, x_i, \lambda)) \\
\tilde{FP} &= \sum_{i:y_i=0} p(c_1|\, x_i, \lambda) \\
\tilde{TN} &= \sum_{i:y_i=0} p(c_0|\, x_i, \lambda) = \sum_{i:y_i=0} (1 - p(c_1|\, x_i, \lambda))
\end{aligned}$$

Then we have natural plug-in estimators of precision and recall:

$$\tilde{P} = \frac{\tilde{TP}}{\tilde{TP} + \tilde{FP}}$$

$$\tilde{R} = \frac{\tilde{TP}}{\tilde{TP} + \tilde{FN}}.$$

Definition 4.2.1. The expected F-measure is the plug-in approximation of F_β using the expected confusion matrix:

$$\tilde{F}_\beta = \left(\frac{\beta}{\tilde{P}} + \frac{1-\beta}{\tilde{R}} \right)^{-1}$$

The following definition of Pareto optimality will prove useful in maximizing the expected F-measure.

Definition 4.2.2. For a multicriteria optimization problem (MOP)

$$\begin{aligned} \max_{x} \quad & \{f_1(x), \ldots, f_k(x)\} \\ \text{s.t.} \quad & x \in \mathcal{X}, \end{aligned}$$

the point $x_0 \in \mathcal{X}$ is said to dominate $x_1 \in \mathcal{X}$ if it holds for all $i = 1, \ldots, k$ that $f_i(x_0) \geq f_i(x_1)$ and for at least one $j = 1, \ldots, k$, $f_j(x_0) > f_j(x_1)$. A feasible point $x_0 \in \mathcal{X}$ is called Pareto optimal if there is no point $x \in \mathcal{X}$ which dominates x_0. The Pareto optimal set is the set of all Pareto optimal points.

The notion of Pareto optimality for a point x_0 means simply that by moving to no other point are we able to increase the value of all objective functions simultaneously, and instead, by selecting a different point we must decrease the value of at least one of the objectives.

We introduce now the main result for F-measure optimization via weighted maximum likelihood. The result and its proof, of which we provide an outline, may be found in [8].

Theorem 4.2.3. *Let $\hat{\lambda}_\beta$ be the maximizer of the expected F-measure $\tilde{F}_\beta$. Then there exists a vector of weights $w(\beta) \in \mathbb{R}^m$ such that $\hat{\lambda}_\beta$ coincides with the weighted maximum likelihood estimator,*

$$\hat{\lambda}_{ML}^{w(\beta)} = \arg\max l^W(\lambda : w(\beta), x, y).$$

That is, we have

$$\hat{\lambda}_\beta = \hat{\lambda}_{ML}^{w(\beta)}.$$

The idea of the proof of Theorem 4.2.3 is the following. Weighted likelihood maximization involves maximizing simultaneously the conditional probabilities $p(y_i|\, x_i, \lambda)$ while setting trade-offs between them. We may set these trade-offs in such a way that different kinds of performance can be achieved, and in particular it can be shown that both optimization problems of log-likelihood maximization and $\tilde{F}_\beta$ maximization are particular solutions of the MOP

$$\max_{\lambda} \{p(y_1|\, x_1, \lambda), \ldots, p(y_m|\, x_m, \lambda)\}. \tag{4.1}$$

All solutions of (4.1) can be obtained by maximizing nonnegative linear combinations of the objectives, as shown generally in [10], and in our case this is a weighted maximum likelihood. We are then able to draw equivalence between $\tilde{F}_\beta$ maximization and weighted likelihood maximization for a certain choice of weights. (We expore more thoroughly such lines of argumentation in our proof of the main result in Chapter 6.) During the proof, the optimal weights are constructed to be:

$$w(\beta)_i = \begin{cases} \frac{1-\beta\tilde{F}_\beta(\hat{\lambda}_\beta)}{\beta\tilde{F}_\beta(\hat{\lambda}_\beta)} p(y_i|\, x_i, \hat{\lambda}_\beta) & \text{if } y_i = 1 \\ p(y_i|\, x_i, \hat{\lambda}_\beta) & \text{if } y_i = 0. \end{cases} \tag{4.2}$$

The expression (4.2) involves $\hat{\lambda}_\beta$, the optimal parameter to be estimated in the first place. However, roughly along the lines of the expectation-maximization algorithm (see [6]), we can determine them in an adaptive manner via the following algorithm, which has been shown to converge to at least a local maximum of $\tilde{F}_\beta$.

Algorithm 4.1 $\tilde{F}_\beta$ Maximizer

1: $n = 1$.
2: Initialize model parameters and calculate initial $\hat{F}_\beta(1)$ from initialized model.
3: Set weights

$$(w_n)_i = \begin{cases} \frac{1-\beta\hat{F}_\beta(n)}{\beta\hat{F}_\beta(n)} p(y_i|\, x_i, \hat{\lambda}_n) & \text{if } y_i = 1 \\ p(y_i|\, x_i, \hat{\lambda}_n) & \text{if } y_i = 0. \end{cases}$$

4: Do one full-batch update of weighted log-likelihood maximization with weights w_n. $n := n + 1$.
5: Calculate model $\hat{F}_\beta(n)$ and model conditional probabilities $p(y_i|\, x_i, \hat{\lambda}_n)$.
6: If convergence criteria not met (no significant improvement of target $\tilde{F}_\beta$ in last k steps) $\rightarrow$ Step 3.

Chapter 5

The Uncertain Maximum Entropy

In this chapter, we introduce the uncertain maximum entropy model, an extension of the standard one presented in Chapter 4. We begin in Section 5.1 with an introduction to the concept of uncertain classification. We continue in Section 5.2 with some definitions and properties relating to the uncertain maximum entropy. In Section 5.3, we then provide a revealing and intuitive link between the uncertain likelihood and the Kullback-Leibler divergence, a measure of difference between probability distributions. Finally in Section 5.4, we explore where we might require uncertain likelihood maximization over standard likelihood maximization in practice, and how we might apply it. Note that the material in Chapters 5, 6 and 7 will be adapted in the forthcoming [23].

5.1 Introduction and Motivation

It is in some situations presumptuous to think that we should have exclusively deterministic information about a given sample when training a classifier. While traditional classification procedures tend to learn only from a deterministically labelled training set, it may be the case that some of the training examples fall into each class with given probabilities, and thus in the binary classification setting we are concerned with, follow a Bernoulli distribution. The intuition that guides such an uncertain framework is that when humans make subjective judgements, whether deciding on a particular course of action or simply giving their opinion about a popular film or a political party, it is rarely a process of deductions from verified knowledge, but instead a process that involves uncertainty on multiple levels. Whether or not we speak from direct knowledge, it is usually belief, as in [7], which informs realistic decision-making that comprehensively exploits the sample available to us.

The presence of uncertainty in a training set may be due to a few possible reasons, such as the following:

1. If the classification problem involves a high degree of subjectivity, we must contend with the bias of individuals, specifically when we have employed annotators to manually classify a sample for the purpose of training. A sentiment analysis application may be considered here, where for example if the sentiment of a social media post x_i on a particular issue is viewed as negative by 70% of polled readers and positive by 30% of them, it would make sense to assign a corresponding Bernoulli distribution to the random class $Y_i \sim B(1, 0.3)$. Since the state of objects in this problem is intrinsically random, we should not consider the output of a classification model to be deterministic either.

2. We consider the case where there is now an objective truth but with data having again been manually annotated. We must here deal with the aspect of human error, as in [4], in order to output the best possible deterministic model classes. To do this we may consider the manually labelled examples to have noisy classifications, with the entropy of a given class governing the emphasis we place on the corresponding point in the training set when fitting our model. In this case, the less certain we are of an example's class, the less we should let it influence our future decisions.
3. Finally, we consider the case of semi-supervised learning, as in [16], where we have a great deal of data available to us, most of which has not been assigned a class. We may wish to use as much of this data as possible by providing (as rough or as precise as we like) noisy estimates of their classes. For example, we might automatically generate classes by a possibly noisy and potentially quite simple (e.g. hard-coded) rule operating on each instance. We may then use this uncertain data in the training phase.

Thus, we are motivated to define an uncertain classifier, as well as a measure of performance which is able to take into account the randomness given to us by the training sample and the randomness that the model predicts, in a manner analogous to a confusion matrix. This will give us a corresponding extension of the F-measure in Chapter 6, which we will often desire to maximize given this framework. We will consider doing so during the training phase, similarly to [14] and [8] in the deterministic framework.

5.2 Definitions and Interpretations

It is straightforward to make sense of the likelihood function for a training set containing examples that are not deterministically classified but rather have random labels. To stress the fact that the observed classes are random we will denote them with capital letters Y_i. We give the corresponding row of probability weights as $q_i = (q_{i1}, \ldots, q_{iM})$, where $M = |\mathcal{Y}|$. These q_i's form the matrix $q \in \mathbb{R}^{m \times M}$. Of course, in our setting of binary classification we consider only $|\mathcal{Y}| = 2$, but we include the general form of the likelihood in order to give a flavour of the multiclass extension.

Definition 5.2.1. For a set of training examples with random labels corresponding to classes $c_j, j = 1, \ldots, M$, we define the uncertain likelihood to be:

$$L^U(\lambda : x, Y) = \prod_{i=1}^{m} \prod_{j=1}^{M} p(c_j | \, x_i, \lambda)^{q_{ij}}.$$

Correspondingly, the uncertain log-likelihood is:

$$l^U(\lambda : x, Y) = \sum_{i=1}^{m} \sum_{j=1}^{M} q_{ij} \log p(c_j | \, x_i, \lambda).$$

To understand why the random observations Y_i lead to a likelihood function L^U, we can "emulate" the random observations by creating many copies of the observations, cloning the attributes x but assigning different classes in the frequencies given by the observed distributions. If our random observation is (x_i, Y_i) with Y_i having the distribution given by the weights $(q_{i1}, \ldots, q_{iM})$, we create a large number K of observations $(x_i, y_{i1}), \ldots, (x_i, y_{iK})$, where the attributes x_i are the same and we have approximately $q_{ij} \cdot K$ copies with class c_j. If we do this for all training examples using the same large number of copies K, the corresponding likelihood function after renormalization is precisely (if all probability weights of the observations are rationals and can be written as Q/K) the uncertain likelihood as $K \to \infty$. From this representation we know that the uncertain likelihood inherits the highly desirable properties of the standard deterministic likelihood. For example, the log-likelihood is a concave function which can be maximized via standard algorithms, e.g. gradient ascent, with:

$$\nabla l^U = \sum_{i=1}^{m} \sum_{j=1}^{M} \frac{q_{ij}}{p(c_j|\, x_i, \lambda)} \nabla p(c_j|\, x_i, \lambda). \tag{5.1}$$

We observe also that there is an obvious extension of the uncertain log-likelihood to its weighted counterpart:

$$l^U(\lambda : w, x, Y) = \sum_{i=1}^{m} w_i \left[\sum_{j=1}^{M} q_{ij} \log p(c_j|\, x_i, \lambda) \right] = \sum_{i=1}^{m} \sum_{j=1}^{M} w_i q_{ij} \log p(c_j|\, x_i, \lambda).$$

As usual, this expression corresponds to modifying the training set by adding new examples having the same attributes and uncertain classes (x_i, Y_i) with intensity w_i.

The primary interpretation of the uncertain maximum entropy we will follow is the "panel of annotators" interpretation, where we think of the randomization of classes as being generated by a panel of annotators who have voted on the class for each example. Our job is to train a model given their votes, and since they do not always agree we end up having non-degenerate distributions over the classes, defined by their proportional votes, instead of deterministic examples.

5.3 A Link to the Kullback-Leibler Divergence

To develop some more intuition about what it means to maximize the uncertain likelihood, we first define the Kullback-Leibler divergence, a measure of difference between two probability distributions (though it is not a distance, since it is not symmetric).

Definition 5.3.1. The Kullback-Leibler divergence of discrete probability distribution p from q is:

$$\mathcal{D}_{KL}(q||p) = \sum_j q_j \log \frac{q_j}{p_j}.$$

The following theorem states that maximizing the uncertain likelihood is equivalent to minimizing Kullback-Leibler divergences between obtained model probabilities $p_i(\lambda) = p(\cdot|\, x_i, \lambda)$ and the class distributions of training examples q_i.

Theorem 5.3.2. *The uncertain log-likelihood admits the following decomposition:*

$$l^U(\lambda : x, Y) = -\sum_{i=1}^{m} \left[H(q_i) + \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda))\right] \tag{5.2}$$

where $H(q_i) = \sum_{j=1}^{M} q_{ij} \log q_{ij}$ is the entropy of the distribution of Y_i and $\mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda))$ is the Kullback-Leibler divergence of the estimated class distribution $p(\cdot|\, x_i, \lambda)$ from the observed distribution of Y_i.

In particular, let $\hat{\lambda}$ be the uncertain likelihood maximizer. Then $\hat{\lambda}$ minimizes the average Kullback-Leibler divergence between experimental and model distributions, that is:

$$\hat{\lambda} = \arg\min \frac{1}{m} \sum_{1}^{m} \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda)).$$

Proof. The statement follows immediately from the definition of the Kullback-Leibler divergence.

$$\begin{aligned}
\sum_{i=1}^{m} \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda)) &= \sum_{i=1}^{m}\sum_{j=1}^{M} q_{ij} \log \frac{q_{ij}}{p(c_j|\, x_i, \lambda)} \\
&= -\sum_{i=1}^{m}\sum_{j=1}^{M} q_{ij} \log p(c_j|\, x_i, \lambda) + \sum_{i=1}^{m}\sum_{j=1}^{M} q_{ij} \log q_{ij} \\
&= -l^U(\lambda : x, Y) - \sum_{i=1}^{m} H(q_i),
\end{aligned}$$

where $H(q_i)$ is the entropy of the distribution q_i. The term $\sum_{i=1}^{m} H(q_i)$ does not depend on the model parameters and hence maximizing the the log-likelihood is equivalent to minimizing the Kullback-Leibler divergence term. □

It can be shown in the same way that maximizing the weighted uncertain likelihood is equivalent to minimizing the weighted average of Kullback-Leibler divergences between experimental and model distributions. The decomposition (5.2) is extremely intuitive, with the uncertain likelihood maximization problem being now interpreted as a minimization of discrepancies between observed and model distributions. The uncertain maximum entropy model attempts to reproduce as faithfully as possible the randomness inherent in the observed sample, in contrast to standard (deterministic) likelihood maximization, which attempts to maximize conditional model probabilities corresponding to the classes y_i which were observed. We find a similar result to (5.2) in [21], which hints at a link to a relaxed expectation-maximization type algorithm that considers the weights w_i to be expectations of

unobserved random variables that should be updated incrementally at each iteration after the "maximization" step. We will explore such an algorithm in Chapter 7, although we will not stress this interpretation, instead showing convergence directly.

5.4 Methodology

Depending on what kind of data is available to us, we advise to use the uncertain likelihood in slightly different ways in order to train efficiently. We provide guidelines to devising simple algorithms of this nature as follows, in cases where we have available a training set of labelled (deterministically and/or noisily) data along with some prior information on some unlabelled data, a training set of labelled and unlabelled data in the absence of prior information, or a training set with only unlabelled data:

1. Labelled data with additional prior information on some unlabelled data.
   (i) Find optimal parameters by incorporating the given labels on the labelled data and the prior information on the unlabelled data in the uncertain likelihood.
   (ii) Use the estimated parameters to infer the distribution of the labels of the unlabelled data.
   (iii) Build the uncertain likelihood with this information and then optimize it.
   (iv) Repeat until some convergence.
2. Labelled and unlabelled data without prior information.
   (i) Find optimal parameters on the labelled data (standard).
   (ii) Use the estimated parameters to infer the distribution of the labels of the unlabelled data.
   (iii) Build the uncertain likelihood with this information and then optimize it.
   (iv) Repeat until some convergence.
3. Only unlabelled data.
   (i) Use hard-coded rules or annotators to build a training set with either uncertain or deterministic classes, or a mix of both.
   (ii) Find optimal parameters by incorporating the assigned labels on the labelled data and/or the assigned distributions on the unlabelled data in the uncertain likelihood.
   (iii) Use the estimated parameters to infer the distribution of the labels of the unlabelled data.
   (iv) Build the uncertain likelihood with this information and then optimize it.
   (v) Repeat until some convergence.

Chapter 6

Noisy F-measure Optimization

In this chapter, we introduce and discuss a generalization of the F-measure to the uncertain classification setting, and investigate the problem of maximizing it. In Section 6.1, we define this "noisy" F-measure, and in Section 6.2, we present equivalence between maximization of the noisy F-measure we defined and maximization of the weighted uncertain likelihood, for a certain choice of weights.

6.1 The Noisy F-measure

In Chapter 4, we outlined the result of expected F-measure maximization via weighted maximum likelihood from [8]. We will extend this result to data with random labels, but first we must define one or more variants of the F-measure appropriate for this framework.

In the presence of random labels, we have the underlying class distribution of training examples given by the matrix q. When we maximize the uncertain likelihood function, we get model distributions $p(\cdot|\, x_i, \lambda)$, which we will sometimes express for convenience as $p_i(\lambda) = (p_{i1}(\lambda), \ldots, p_{iM}(\lambda))$, reducing to $(p_{i1}(\lambda), p_{i0}(\lambda))$ in the binary classification case, where as usual we give the positive class the label 1 and the negative class 0.

Define the random variable $X_i = (Y_i, Z_i)$ where Y_i follows the Bernoulli distribution q_i and Z_i represents the class returned by the model, i.e. is a random variable with distribution $p_i(\lambda)$. Noting that X_i can take four possible values, we are able to define equivalents to TP, FN, FP and TN in the following way:

$$\begin{aligned}
\tilde{TP}_i^U &= P(X_i = (1,1)) \\
\tilde{FN}_i^U &= P(X_i = (1,0)) \\
\tilde{FP}_i^U &= P(X_i = (0,1)) \\
\tilde{TN}_i^U &= P(X_i = (0,0))
\end{aligned}$$

The way that we calculate these quantities relies on what we consider to be the nature of the dependence between the true and modelled classes. If we consider them to be independent, we get products of the probabilities, for example $\tilde{TP}_i^U = P(X_i = (1,1)) = q_{i1} p_{i1}(\lambda) = q_{i1} p(c_1|\, x_i, \lambda)$. On the other hand, we can model (the

strongest possible) dependence in the following way:

$$\begin{aligned}
\tilde{TP}_i^U &= P(X_i = (1,1)) = \min(q_{i1}, p_{i1}(\lambda)) \\
\tilde{FN}_i^U &= P(X_i = (1,0)) = \max(0, q_{i1} - p_{i1}(\lambda)) = \max(0, p_{i0}(\lambda) - q_{i0}) \\
\tilde{FP}_i^U &= P(X_i = (0,1)) = \max(0, p_{i1}(\lambda) - q_{i1}) \\
\tilde{TN}_i^U &= P(X_i = (0,0)) = \min(q_{i0}, p_{i0}(\lambda))
\end{aligned} \tag{6.1}$$

Note that taking the minima of probabilities in the true classification case allows us to take only the probability weight common to the true and model distributions, whereas we only get weight for false classification if we predict there to be more, or less, probability weight in the given class than actually exists. This property is highly desirable, and so we opt for the formulation (6.1). We will exploit later another of its useful properties to prove the main result of this chapter. We do note however, that though we have given two extreme cases of representing the dependence between true and model class distributions, there are in theory other ways to do so.

The definitions (6.1), in both uncorrelated and correlated cases, are confusion contributions for the i'th training example. The uncertain equivalents to the deterministic TP, FN, FP and TN confusion counts are the sums of our newly defined quantities over i. In the case of true and modeled classes being considered uncorrelated, we define:

$$\begin{aligned}
\tilde{TP}^U &= \sum_{i=1}^{m} \min(q_{i1}, p(c_1 | \, x_i, \lambda)) \\
\tilde{FN}^U &= \sum_{i=1}^{m} \max(0, p(c_0 | \, x_i, \lambda) - q_{i0}) \\
\tilde{FP}^U &= \sum_{i=1}^{m} \max(0, p(c_1 | \, x_i, \lambda) - q_{i1}) \\
\tilde{TN}^U &= \sum_{i=1}^{m} \min(q_{i0}, p(c_0 | \, x_i, \lambda))
\end{aligned}$$

Using these definitions, we derive noisy precision and noisy recall:

$$\tilde{P}^U = \frac{\tilde{TP}^U}{\tilde{TP}^U + \tilde{FP}^U}, \quad \tilde{R}^U = \frac{\tilde{TP}^U}{\tilde{TP}^U + \tilde{FN}^U}$$

and the noisy F–measure follows.

Definition 6.1.1. The noisy F-measure is the weighted harmonic mean of noisy precision and noisy recall,

$$\tilde{F}_\beta^U = \left(\frac{\beta}{\tilde{P}^U} + \frac{1-\beta}{\tilde{R}^U} \right)^{-1},$$

with parameter $\beta \in [0, 1]$.

We wish ideally to directly maximize the noisy F-measure. An apparent disadvantage of the correlated formulation is that it is not everywhere differentiable. However, the advantage, due to its definitional properties, is that maximization of correlated F-measure corresponds roughly also to a minimization of discrepancies between q_i and $p_i(\lambda)$. We will see this in Section 6.2. Indeed, on a related tangent, maximizing noisy accuracy, that is,

$$acc^U = \frac{\tilde{TP}^U + \tilde{TN}^U}{m},$$

is equivalent to minimizing the average L^1 distance between q_i and $p_i(\lambda)$. To see this, we use the fact that $\min(x, y) = \frac{1}{2}[x + y - |x - y|]$ to express acc^U in the following way:

$$acc^U = 1 - \frac{1}{m}\sum_{i=1}^{m}|q_{i1} - p_{i1}(\lambda)|.$$

6.2 Noisy F-measure Optimization via Weighted Maximum Uncertain Likelihood

We present now the main theorem of this chapter, which directly links noisy F-measure maximization and uncertain likelihood maximization.

Theorem 6.2.1. *Let $\hat{\lambda}_\beta$ be the maximizer of the noisy F-measure $\tilde{F}^U_\beta$. Then there exists a matrix of weights $w(\beta) \in \mathbb{R}^m$ such that $\hat{\lambda}_\beta$ coincides with the weighted maximum uncertain likelihood estimator*

$$\hat{\lambda}^{w(\beta)}_{ML} = \arg\max_\lambda l^U(\lambda : w(\beta), x, Y).$$

That is, we have

$$\hat{\lambda}_\beta = \hat{\lambda}^{w(\beta)}_{ML}.$$

Proof. Let $\hat{\lambda}_\beta$ be the $\tilde{F}^U_\beta$ maximizer, i.e.

$$\hat{\lambda}_\beta = \arg\max_\lambda \tilde{F}^U_\beta.$$

When we give the following form of the noisy F-measure:

$$\tilde{F}^U_\beta = \frac{\tilde{TP}^U}{\beta(\tilde{TP}^U - \tilde{TN}^U) + \beta\sum_{i=1}^m q_{i0} + (1-\beta)\sum_{i=1}^m q_{i1}},$$

we realize that the maximizer $\hat{\lambda}_\beta$ of $\tilde{F}^U_\beta$ is an element of the Pareto optimal set of the multi-criteria optimization problem (MOP):

$$\max_\lambda \left\{\tilde{TP}^U, \tilde{TN}^U\right\},$$

or:

$$\max_{\lambda} \left\{ \sum_{i=1}^{m} \min(q_{i1}, p(c_1 | x_i, \lambda)), \sum_{i=1}^{m} \min(q_{i0}, p(c_0 | x_i, \lambda)) \right\}. \tag{6.2}$$

This is because the denominator of $\tilde{F}_{\beta}^{U}$ is always positive and the function

$$f(x, y) = \frac{x}{\beta(x - y) + \beta \sum_{i=1}^{m} q_{i0} + (1 - \beta) \sum_{i=1}^{m} q_{i1}}$$

is increasing in x and y. If we assume that $\hat{\lambda}_{\beta}$ is not Pareto efficient for the MOP (6.2), then we can find another set of parameters λ_0 such that the pair $(\tilde{TP}^{U}(\lambda_0), \tilde{TN}^{U}(\lambda_0))$ dominates $(\tilde{TP}^{U}(\hat{\lambda}_{\beta}), \tilde{TN}^{U}(\hat{\lambda}_{\beta}))$, that is, at least one of the objectives is improved in λ_0 when compared with $\hat{\lambda}_{\beta}$ with the other one having not decreased. However, this would mean that $\tilde{F}_{\beta}^{U}(\lambda_0) > \tilde{F}_{\beta}^{U}(\hat{\lambda}_{\beta})$, which contradicts the assumption that $\hat{\lambda}_{\beta}$ maximizes the noisy F-measure.

Passing to the finer granularity MOP, we observe also that $\hat{\lambda}_{\beta}$ must be an element of the Pareto optimal set of:

$$\max_{\lambda} \{ \min(q_{i1}, p(c_1 | x_i, \lambda)), \min(q_{i0}, p(c_0 | x_i, \lambda)) : \forall i \}. \tag{6.3}$$

Indeed, the following argument shows that (6.3) has a Pareto optimal set which contains that of (6.2). If we assume that λ is Pareto optimal for (6.2) but not for (6.3), then we can find λ_0 such that some of the objectives (6.3) are improved and none of them is decreased. But this would mean that the pair $(\tilde{TP}^{U}(\lambda_0), \tilde{TN}^{U}(\lambda_0))$ dominates $(\tilde{TP}^{U}(\lambda), \tilde{TN}^{U}(\lambda)$, which directly contradicts the assumption of Pareto optimality of λ for (6.2). Therefore Pareto optimality for (6.2) implies Pareto optimality for (6.3).

See that the Pareto optimal set of (6.3) is the same as the Pareto optimal set of:

$$\begin{aligned} \max_{\lambda} \{ & [q_{i1} \log p(c_1 | x_i, \lambda) + q_{i0} \log p(c_0 | x_i, \lambda)] \, \mathbb{1}\{p(c_1 | x_i, \lambda) \leq q_{i1}\}, \\ & [q_{i1} \log p(c_1 | x_i, \lambda) + q_{i0} \log p(c_0 | x_i, \lambda)] \, \mathbb{1}\{p(c_1 | x_i, \lambda) \geq q_{i1}\} : \forall i \} \end{aligned} \tag{6.4}$$

which is equivalent to:

$$\begin{aligned} \max_{\lambda} \{ & - [H(q_i) + \mathcal{D}_{KL}(q_i || p(\cdot | x_i, \lambda))] \, \mathbb{1}\{p(c_1 | x_i, \lambda) \leq q_{i1}\}, \\ & - [H(q_i) + \mathcal{D}_{KL}(q_i || p(\cdot | x_i, \lambda))] \, \mathbb{1}\{p(c_1 | x_i, \lambda) \geq q_{i1}\} : \forall i \}. \end{aligned} \tag{6.5}$$

We are able to claim that the Pareto optimal set of (6.3) is the same as that of (6.5) by the following argument. Take λ which is Pareto optimal for (6.3) but not for (6.5). Then in (6.5) there is a λ_0 such that for at least one i, we have one of the given objectives improving on λ without the other one decreasing. By the form of the objectives (negative Kullback-Leibler divergences), this means that $|q_{i1} - p(c_1 | x_i, \lambda_0)| < |q_{i1} - p(c_1 | x_i, \lambda)|$. But then in (6.3) we have by the form of the min function that λ_0 improves upon the i'th objectives in λ in the same way, i.e. λ

is not Pareto optimal for (6.3). We conclude by this contradiction that $\hat{\lambda}_\beta$ is in the Pareto optimal set of (6.5).

Since the objectives in (6.5) are concave, every point in its Pareto optimal set can be represented as a maximizer of:

$$G(\lambda : u, v, x, Y) = -\sum_{i=1}^{m} \{u_i \left[H(q_i) + \mathcal{D}_{KL}(q_i || p(\cdot | x_i, \lambda))\right] \mathbb{1}\{p(c_1 | x_i, \lambda) \leq q_{i1}\} + v_i \left[H(q_i) + \mathcal{D}_{KL}(q_i || p(\cdot | x_i, \lambda))\right] \mathbb{1}\{p(c_1 | x_i, \lambda) \geq q_{i1}\}\} \quad (6.6)$$

for some nonnegative weights $u, v \in \mathbb{R}^m$.

So $\hat{\lambda}_\beta$ can be represented as a maximizer of G for some u, v. G can be viewed as a weighted uncertain likelihood in the following way:

$$G(\lambda : w, x, Y) = -\sum_{i=1}^{m} w_i \left[H(q_i) + \mathcal{D}_{KL}(q_i || p(\cdot | x_i, \lambda))\right],$$

where

$$w_i = u_i \mathbb{1}\{p(c_1 | x_i, \lambda) \leq q_{i1}\} + v_i \mathbb{1}\{p(c_1 | x_i, \lambda) \geq q_{i1}\}.$$

We calculate these weights w_i as in the deterministic case by matching the coefficients of gradients.

We know that $\nabla \tilde{F}_\beta^U(\hat{\lambda}_\beta) = 0$. Now:

$$\nabla \tilde{F}_\beta^U = \sum_{i=1}^{m} \frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TP}^U} \cdot \frac{\partial \tilde{TP}^U}{\partial p(c_1 | x_i, \lambda)} \nabla p(c_1 | x_i, \lambda) + \sum_{i=1}^{m} \frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TN}^U} \cdot \frac{\partial \tilde{TN}^U}{\partial p(c_1 | x_i, \lambda)} \nabla p(c_1 | x_i, \lambda).$$

The derivatives of $\tilde{TP}^U$ and $\tilde{TN}^U$ are generalized derivatives, since $\tilde{TP}^U$ and $\tilde{TN}^U$ are continuous when taken as functions of $p(c_1 | x_i, \lambda)$ which are not smooth only at one point ($p(c_1 | x_i, \lambda) = q_{i1}$). We get:

$$\frac{\partial \tilde{TP}^U}{\partial p(c_1 | x_i, \lambda)} = \frac{\partial \min(q_{i1}, p(c_1 | x_i, \lambda))}{\partial p(c_1 | x_i, \lambda)} = \mathbb{1}\{p(c_1 | x_i, \lambda) \leq q_{i1}\}$$
$$\frac{\partial \tilde{TN}^U}{\partial p(c_1 | x_i, \lambda)} = \frac{\partial \min(1 - q_{i1}, 1 - p(c_1 | x_i, \lambda))}{\partial p(c_1 | x_i, \lambda)} = -\mathbb{1}\{p(c_1 | x_i, \lambda) \geq q_{i1}\}.$$

and

$$\nabla \tilde{F}_\beta^U = \sum_{i=1}^{m} \left[\frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TP}^U} \mathbb{1}\{p(c_1 | x_i, \lambda) \leq q_{i1}\} - \frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TN}^U} \mathbb{1}\{p(c_1 | x_i, \lambda) \geq q_{i1}\} \right] \nabla p(c_1 | x_i, \lambda). \quad (6.7)$$

We also have the alternate representation of $\hat{\lambda}_\beta$ as a maximizer of G. This means that $\nabla G(\hat{\lambda}_\beta) = 0$ for some correct choice of $u(\beta), v(\beta)$ to be determined.

$$\nabla G = -\sum_{i=1}^{m} \left\{ u(\beta)_i \frac{\partial \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda))}{\partial p(c_1|\, x_i, \lambda)} \mathbb{1}\{p(c_1|\, x_i, \lambda) \leq q_{i1}\} \right.$$
$$\left. + v(\beta)_i \frac{\partial \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda))}{\partial p(c_1|\, x_i, \lambda)} \mathbb{1}\{p(c_1|\, x_i, \lambda) \geq q_{i1}\} \right\} \nabla p(c_1|\, x_i, \lambda) \quad (6.8)$$

Comparing gradients, we find appropriate weights, to be evaluated at $\lambda = \hat{\lambda}_\beta$:

$$u(\beta)_i = -\frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TP}^U} \Bigg/ \frac{\partial \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda))}{\partial p(c_1|\, x_i, \lambda)}$$
$$v(\beta)_i = \frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TN}^U} \Bigg/ \frac{\partial \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda))}{\partial p(c_1|\, x_i, \lambda)}.$$

Observe that by letting

$$w(\beta)_i = u(\beta)_i \mathbb{1}\{p(c_1|\, x_i, \hat{\lambda}_\beta) \leq q_{i1}\} + v(\beta)_i \mathbb{1}\{p(c_1|\, x_i, \hat{\lambda}_\beta) \geq q_{i1}\}$$

we arrive at the uncertain likelihood

$$G(\lambda : w(\beta), x, Y) = l^U(\lambda : w(\beta), x, Y) = -\sum_{i=1}^{m} w(\beta)_i \left[H(q_i) + \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda)) \right].$$

Note that this choice of $w(\beta)$ is always positive.

After some simple algebra, we find that:

$$\frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TP}^U} = \frac{\tilde{F}_\beta^U}{\tilde{TP}^U} \left[1 - \beta \tilde{F}_\beta^U \right]$$
$$\frac{\partial \tilde{F}_\beta^U}{\partial \tilde{TN}^U} = \frac{\beta (\tilde{F}_\beta^U)^2}{\tilde{TP}^U},$$

and

$$\frac{\partial \mathcal{D}_{KL}(q_i||p(\cdot|\, x_i, \lambda))}{\partial p(c_1|\, x_i, \lambda)} = \frac{p(c_1|\, x_i, \lambda) - q_{i1}}{p(c_1|\, x_i, \lambda) \cdot p(c_0|\, x_i, \lambda)}.$$

Our final expression for the weights $w(\beta)$ is:

$$w(\beta)_i = \frac{p(c_1|\, x_i, \hat{\lambda}_\beta) \cdot p(c_0|\, x_i, \hat{\lambda}_\beta)}{p(c_1|\, x_i, \hat{\lambda}_\beta) - q_{i1}} \cdot \frac{\tilde{F}_\beta^U(\hat{\lambda}_\beta)}{\tilde{TP}^U(\hat{\lambda}_\beta)} \Big[\beta \tilde{F}_\beta^U(\hat{\lambda}_\beta) \mathbb{1}\{p(c_1|\, x_i, \hat{\lambda}_\beta) \geq q_{i1}\}$$
$$- (1 - \beta \tilde{F}_\beta^U(\hat{\lambda}_\beta)) \mathbb{1}\{p(c_1|\, x_i, \hat{\lambda}_\beta) \leq q_{i1}\} \Big] \quad (6.9)$$

Each $\tilde{F}_\beta^U$ maximizer can be realized as a maximizer of the weighted uncertain likelihood, for these weights $w(\beta)$. $\square$

Chapter 7

Algorithm for Noisy F-measure Maximization

This chapter is dedicated to the application of our main result of equivalence between noisy F-measure maximization and uncertain likelihood maximization. First in Section 7.1, we introduce an algorithm that allows this. Then in Section 7.2 we provide some insights into the form of the weights, and why it necessitates us placing a bound on them in practice. Finally in Section 7.3, we give some justification for our algorithm's validity by showing convergence to at least a local maximum of noisy F-measure.

7.1 The Algorithm

Since the uncertain likelihood inherits nice properties from the standard likelihood, and in particular concavity, it is simple to maximize. When we turn our attention to the form of the weights (6.9), we realize that as in the deterministic likelihood case, they depend on the parameter $\hat{\lambda}_\beta$ we wish to obtain, which is at first sight unfortunate. However, we may determine them adaptively via the following algorithm. Note that in (6.9), the denominator includes a factor of $p(c_1|\, x_i, \hat{\lambda}_\beta) - q_{i1}$ and so, when theoretical and model distributions are close (which ideally they are), the weights may explode in size. To control this behaviour in practice we place computationally sensible bounds on $w(\beta)$.

Algorithm 7.1 $\tilde{F}_\beta^U$ maximizer

1: $n = 1$.
2: Initialize model parameters and calculate initial $\hat{F}_\beta^U(1)$ and $\hat{TP^U}(1)$ from initialized model.
3: Set weights (with some imposed bound)

$$(w_n)_i = \frac{p(c_1|\, x_i, \hat{\lambda}_n) \cdot p(c_0|\, x_i, \hat{\lambda}_n)}{p(c_1|\, x_i, \hat{\lambda}_n) - q_{i1}} \cdot \frac{\hat{F}_\beta(n)}{\hat{TP^U}(n)} \Big[\beta \hat{F}_\beta(n) \mathbb{1}\{p(c_1|\, x_i, \hat{\lambda}_n) \geq q_{i1}\} - (1 - \beta \hat{F}_\beta(n)) \mathbb{1}\{p(c_1|\, x_i, \hat{\lambda}_n) \leq q_{i1}\}\Big].$$

4: Do one full-batch update of weighted uncertain log-likelihood maximization with weights w_n. $n := n + 1$.
5: Calculate model $\hat{F}_\beta^U(n)$, $\hat{TP^U}(n)$ and model conditional probabilities $p(y_i \,|\, x_i, \hat{\lambda}_n)$.
6: If convergence criteria not met (no significant improvement of target $\tilde{F}_\beta^U$ in the last k steps) $\rightarrow$ Step 3.

7.2 Bounding the Weights

We justify bounding the weights theoretically in the following way. The idea is to smooth out the confusion counts in order to avoid the use of indicator functions in (6.9), replacing them with expressions which are linear in $p(c_1|\, x_i, \hat{\lambda}_\beta) - q_{i1}$ close to the singularity. When these terms cancel with the problematic denominator, we are left with bounded weights.

Consider smoothing $\tilde{TP}^U$:

$$\begin{aligned}\tilde{TP}^U &= \sum_{i=1}^{m} \min(q_{i1}, p(c_1|\, x_i, \lambda)) \\ &= \sum_{i=1}^{m} \left[p(c_1|\, x_i, \lambda) - \max(0, p(c_1|\, x_i, \lambda) - q_{i1})\right] \\ &= \sum_{i=1}^{m} \left[p(c_1|\, x_i, \lambda) - g_1(p(c_1|\, x_i, \lambda) - q_{i1})\right].\end{aligned}$$

And similarly,

$$\tilde{TN}^U = \sum_{i=1}^{m} \left[q_{i0} - g_2(p(c_1|\, x_i, \lambda) - q_{i1})\right],$$

for some choice of smooth functions g_1 and g_2 that approximate $\max(0, x)$.

Then we have:

$$\begin{aligned}\frac{\partial \tilde{TP}^U}{\partial p(c_1|\, x_i, \lambda)} &= 1 - g_1'(p(c_1|\, x_i, \lambda) - q_{i1}) \\ \frac{\partial \tilde{TN}^U}{\partial p(c_1|\, x_i, \lambda)} &= -g_2'(p(c_1|\, x_i, \lambda) - q_{i1}).\end{aligned}$$

And so,

$$\begin{aligned}w(\beta)_i = \frac{p(c_1|\, x_i, \hat{\lambda}_\beta) \cdot p(c_0|\, x_i, \hat{\lambda}_\beta)}{p(c_1|\, x_i, \hat{\lambda}_\beta) - q_{i1}} \cdot \frac{\tilde{F}_\beta^U(\hat{\lambda}_\beta)}{\tilde{TP}^U(\hat{\lambda}_\beta)} \Big[\beta \tilde{F}_\beta^U(\hat{\lambda}_\beta) g_2' \left(p(c_1|\, x_i, \hat{\lambda}_\beta) - q_{i1}\right) \\ -(1 - \beta \tilde{F}_\beta^U(\hat{\lambda}_\beta))(1 - g_1' \left(p(c_1|\, x_i, \hat{\lambda}_\beta) - q_{i1}\right)\Big]. \quad (7.1)\end{aligned}$$

One appropriate choice of functions g_1 and g_2 is the following. Suppose $\epsilon > 0$ and let:

$$\begin{aligned}f_1(x) &= \frac{(x+\epsilon)^2}{2\epsilon} \\ f_2(x) &= \frac{x^2}{2\epsilon}.\end{aligned}$$

Now set:

$$g_1(x) = \begin{cases} 0, & \text{if } x \leq -\epsilon. \\ f_1(x), & \text{if } -\epsilon < x \leq 0. \\ x + \frac{\epsilon}{2}, & \text{if } x > 0. \end{cases} \tag{7.2}$$

$$g_2(x) = \begin{cases} 0, & \text{if } x \leq 0. \\ f_2(x), & \text{if } 0 < x < \epsilon. \\ x - \frac{\epsilon}{2}, & \text{if } x \geq \epsilon. \end{cases} \tag{7.3}$$

Then we find that for $p(c_1 | \, x_i, \lambda) - q_{i1} \in [-\epsilon, 0]$,

$$w(\beta)_i = \frac{p(c_1 | \, x_i, \hat{\lambda}_\beta) \cdot p(c_0 | \, x_i, \hat{\lambda}_\beta) \cdot \tilde{F}^U_\beta(\hat{\lambda}_\beta)(1 - \beta \tilde{F}^U_\beta(\hat{\lambda}_\beta))}{\epsilon \tilde{TP}^U(\hat{\lambda}_\beta)}.$$

Similarly for $p(c_1 | \, x_i, \lambda) - q_{i1} \in [0, \epsilon]$,

$$w(\beta)_i = \frac{p(c_1 | \, x_i, \hat{\lambda}_\beta) \cdot p(c_0 | \, x_i, \hat{\lambda}_\beta) \cdot \beta \tilde{(F^U_\beta)^2}(\hat{\lambda}_\beta)}{\epsilon \tilde{TP}^U(\hat{\lambda}_\beta)}.$$

These cases taken together show that as $p(c_1 | \, x_i, \lambda) - q_{i1} \to 0$, the weights are bounded in inverse proportionality to ϵ.

7.3 Convergence of the Algorithm

We now show that, given that the learning rate for the full-batch gradient ascent is small enough, each step of Algorithm 7.1 improves the attained value of the noisy F-measure. Hence, if the learning rate is decreasing appropriately, the algorithm will converge.

At the n'th step of the algorithm, the following update of the parameters λ is performed:

$$\hat{\lambda}_{n+1} := \hat{\lambda}_{n+1} + \epsilon_n \cdot \nabla l^U(\lambda_n : w_n, x, y) \tag{7.4}$$

where w_n are the weights described in the algorithm, and ϵ_n is the learning rate. The update (7.4) describes a theoretically small change in λ_{n+1} in the direction of the gradient at the previous λ_n.

For the gradient ∇l^U, from (5.1) we have:

$$\nabla l^U(\lambda : w_n, x, Y) = \sum_{i=1}^{m} (w_n)_i \frac{q_{i1} - p(c_1 | \, x_i, \lambda)}{p(c_1 | \, x_i, \lambda) \cdot p(c_0 | \, x_i, \lambda)} \nabla p(c_1 | \, x_i, \lambda)$$

Then using the definition of w_n,

$$\begin{aligned} \nabla l^U(\lambda : w_n, x, Y) = \sum_{i=1}^{m} & \frac{p(c_1 | \, x_i, \hat{\lambda}_n) \cdot p(c_0 | \, x_i, \hat{\lambda}_n)}{p(c_1 | \, x_i, \lambda) \cdot p(c_0 | \, x_i, \lambda)} \cdot \frac{p(c_1 | \, x_i, \lambda) - q_{i1}}{p(c_1 | \, x_i, \hat{\lambda}_n) - q_{i1}} \\ & \times \left[\frac{\partial \tilde{F}^U_\beta}{\partial \tilde{TP}^U} \mathbb{1}\{p(c_1 | \, x_i, \lambda) \leq q_{i1}\} - \frac{\partial \tilde{F}^U_\beta}{\partial \tilde{TN}^U} \mathbb{1}\{p(c_1 | \, x_i, \lambda) \geq q_{i1}\} \right] \nabla p(c_1 | \, x_i, \lambda). \end{aligned}$$

Now from (6.7), evaluating the above gradient at the current parameter state $\hat{\lambda}_n$ obviously yields:

$$\nabla l^U(\hat{\lambda}_n : w_n, x, Y) = \nabla \tilde{F}^U_\beta(\hat{\lambda}_n).$$

Therefore, the update (7.4) always results in an improvement of the objective $\tilde{F}^U_\beta$. This proves that the algorithm will eventually converge to a local maximum of noisy F-measure. The immediate question arises of finding the global maximum of noisy F-measure, and whether at least a better local maximum can be found using Algorithm 7.1 by performing at each iteration more than one full-batch update of uncertain likelihood maximization (i.e. more than one step of gradient ascent). We provide further motivation for exploring this direction of research in Chapter 9, where it is noted that both convergence and performance tend to improve when considering more than one step of gradient ascent. It might be that this revised approach allows us to escape regions with suboptimal local maxima of $\tilde{F}^U_\beta$. If this is the case, the question remains of how to choose the number of steps in the gradient ascent performed at each iteration of our algorithm.

Part III

Experiments and Conclusions

Chapter 8

Support Vector Machine Experiments

In this chapter, we test the performance of F-measure optimization via support vector machines, aiming to demonstrate the superior performance of the F-measure maximizing support vector machine as previously formulated in Chapter 3 over the more standard approach. In Section 8.1, we outline the data we use and the general framework of the experiments. Then in Section 8.2, we provide details of the design of the experiments to perform, as well as of the implementation. Finally, in Section 8.3, we give results of the performance and summarize the implications of these results.

8.1 Data

We implement the F-measure maximizing support vector machine on classification tasks for four datasets, aiming to show that both for balanced data and in the presence of class imbalance, there is a measurable gain in F-measure both during the training and testing phase from minimizing approximate misclassification counts instead of distances. In a similar setup to [8], we define the following datasets.

Balanced data – A_1: We simulate a dataset of 450 samples (x_i, y_i) with two classes, $\bar{u}$ (negative class) and u (positive class), and two features. Each class contains 225 examples, distributed as spherical Gaussians in the space of features. The examples from class $\bar{u}$ are distributed as $\mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = (0.5, 1)$ and $\Sigma_0 = (0.3, 0.3)^T I_2$. Class u is generated by $\mathcal{N}(\mu_1, \Sigma_1)$, with $\mu_1 = (1, 0.8)$ and $\Sigma_1 = (0.3, 0.3)^T I_2$. We use balanced sets of 400 examples for training and 50 for testing.

Unbalanced data – A_2: We simulate an unbalanced dataset of 450 samples (x_i, y_i) with two classes, $\bar{u}$ (negative class) and u (positive class), and two features. Class $\bar{u}$ contains 405 examples and class u contains 45 examples, distributed as spherical Gaussians in the space of features. The examples from class $\bar{u}$ are distributed as $\mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = (0.5, 1)$ and $\Sigma_0 = (0.3, 0.3)^T I_2$. Class u is generated by $\mathcal{N}(\mu_1, \Sigma_1)$, with $\mu_1 = (1, 0.8)$ and $\Sigma_1 = (0.3, 0.3)^T I_2$. We use an unbalanced set of 400 examples for training (380 from $\bar{u}$ and 20 from u) and a balanced set of 50 for testing.

Balanced data – B_1: We simulate a dataset of 450 samples (x_i, y_i) with two classes, $\bar{u}$ (negative class) and u (positive class), and two features. Each class contains 225 examples, distributed as elliptical Gaussians in the space of features. The examples from class $\bar{u}$ are distributed as $\mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = (2, 1)$ and $\Sigma_0 = (1, 0.3)^T I_2$. Class u is generated by $\mathcal{N}(\mu_1, \Sigma_1)$, with $\mu_1 = (1, 2)$ and $\Sigma_1 = (0.3, 1)^T I_2$. We use balanced sets of 400 examples for training and 50 for testing.

Unbalanced data – B_2: We simulate an unbalanced dataset of 450 samples (x_i, y_i) with two classes, $\bar{u}$ (negative class) and u (positive class), and two features. Class $\bar{u}$ contains 405 examples and class u contains 45 examples, distributed as spherical Gaussians in the space of features. The examples from class $\bar{u}$ are distributed as $\mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = (2, 1)$ and $\Sigma_0 = (1, 0.3)^T I_2$. Class u is generated by $\mathcal{N}(\mu_1, \Sigma_1)$, with $\mu_1 = (1, 2)$ and $\Sigma_1 = (0.3, 1)^T I_2$. We use an unbalanced set of 400 examples for training (380 from $\bar{u}$ and 20 from u) and a balanced set of 50 for testing.

8.2 Experimental Setup

In order to compare both explanatory and predictive performance of standard weighted and F-measure maximizing support vector machines, we examine their performance on the training and test set. We fit three different classifiers: a standard weighted support vector machine cross-validated to optimize predictive F-measure, an F-measure maximizing support vector machine cross-validated to optimize predictive F-measure, and a standard weighted support vector machine cross-validated to optimize predictive accuracy. The details are as follows, and we perform each of the below for a range of $\beta \in [0, 1]$.

First, we fit a standard weighted support vector machine model, with parameters (C_+, C_-) chosen by a 2-fold cross-validation. We choose parameters this way in order to maintain a good fit while minimizing overfitting to the data at hand. The criterion we use for cross-validation is average predictive F_β. The process of cross-validation involves first splitting the training set randomly into two subsets or "folds". For each combination of parameters (C_+, C_-) considered on a grid of values (we consider the lower triangular grid of values such that $C_+ \geq C_-$ since in our examples we consider the positive class to be the rare one, with $m_+ \leq m_-$), we test a support vector machine with these parameters on each fold, or "validation" subset, measuring the predictive performance of the model trained on the remaining subset. We actually measure predictive performance in terms of confusion counts TP, FP and FN, resulting in us having two such values for each confusion count. We then take cross-validation averages. That is, we define (we are in the simplest case of $v = 2$):

$$TP^{(CV)} = \frac{1}{v} \sum_{i=1}^{v} TP^{(i)}$$

$$FP^{(CV)} = \frac{1}{v} \sum_{i=1}^{v} FP^{(i)}$$

$$FN^{(CV)} = \frac{1}{v} \sum_{i=1}^{v} FN^{(i)},$$

where $TP^{(i)}$, $FP^{(i)}$ and $FN^{(i)}$ are the predictive true positive, false positive and false negative rates for the support vector machine tested on the i'th fold.

We take the cross-validation criterion to be the F-measure with plug-in cross-validated confusion counts, that is:

$$F_\beta^{(CV)} = \left(\frac{\beta}{P^{(CV)}} + \frac{1-\beta}{R^{(CV)}} \right)^{-1},$$

where $P^{(CV)}$ and $R^{(CV)}$ are plug-in precision and recall defined on the basis of the cross-validation averaged confusion counts. Once we have performed the above and recorded values of $F_\beta^{(CV)}$ for a sufficiently large number of combinations of C_+ and C_-, we declare the parameters with the highest such average to be optimal.

An interesting discussion of cross-validation on F-measure is found in [11], where it is explained that the above "$F_{tp,fp}$" method of averaging F-measure for the purposes of cross-validation is by far the most unbiased one, particularly in the presence of class imbalance. This is in contrast to what many practitioners still tend to do, which is to compute predictive F-measure on each fold and average these values instead.

In general, v-fold cross-validation can be considered, where we split a dataset randomly into v folds, and for a given combination of parameters and for $k = 1, \ldots, v$, we train our model on the training set with the k'th fold removed, then "validating" or testing it on the k'th fold. Then, once we have v predictive scores, we take the average of them in order to calculate the cross-validation criterion. After repeating this procedure for all parameter values we wish to consider, we take the optimal parameter values to be those giving the highest cross-validation criterion (in our case, we take the highest, but often a cross-validation criterion will take the form of an error rate, in which case we would wish to take the lowest score). However, we consider at this stage only 2-fold cross-validation in order to balance explanatory capacity of our experiments with the high computational cost.

Secondly, we fit the F-measure maximizing support vector machine by using a nonlinear optimization solver. More specifically, we use the package `nloptr` in R. This requires us first to define an initial guess for the support vector machine solution (w, b, ξ). We find experimentally that a reasonable choice of initial guess, at least for our data, is given by the solution of the standard weighted support vector machine (which is of course easy to solve, since it is a quadratic programming problem) with parameters $(C_+, C_-) = (m_-, m_+)/10$. This is in keeping with (3.3), that a good, though rough, heuristic, is to choose parameters such that the ratio $C_+/C_- = m_-/m_+$. We then set the parameter C in the formulation of the F-measure maximizing support vector machine and a in the step function approximation s. We find often that in the nonlinear optimization program it is reasonable to set uniformly $C = m_+$ and $a = 1$ (we are not even altering these between values of β), however, as usual in support vector machine theory, these parameters are difficult to optimize, and selecting them wrongly leaves the classifier very prone to underfitting or more likely, overfitting. We therefore perform cross-validation only over a and on a very small grid of values, noting simply the often superior performance of the F-measure maximizing support vector machine even in this case. For practical purposes it is not computationally feasible to extend it any further than this.

Another thing we need to take into account when using a nonlinear optimization solver is the choice of algorithm. We use the `COBYLA` algorithm included in the `nloptr` package, which performs constrained optimization by linear approximations as described in [22]. An advantage for implementation is that it does not require manual calculation of the derivative of the objective function or the Jacobian of the constraints. We do note that since the problem we are considering is in principle quite difficult to solve, and is not convex, there is no solid guarantee that we find a global solution. This can indeed explain the at times idiosyncratic behaviour of the `nloptr` program and consequently our F_β optimizer. There are parameters which we find it necessary to tweak manually across sample realizations, not only C but also the maximum number of iterations performed by `nloptr`. Our results were even quite sensitive to variations in the maximum number of iterations performed by the quadratic programming solver that provides the initial guess for `nloptr`. We provide our experimental results as not only a matter of practical importance, but more importantly also as motivation for further investigation of F-measure optimization via support vector machines. Perhaps, for example, the step function approximation s could be altered or further approximations explored in order to guarantee global minimization whenever local minimization occurs.

Thirdly, we fit another standard weighted support vector machine model, with parameters (C_+, C_-) chosen by a 2-fold cross validation that maximizes the criterion of average predictive accuracy instead of F_β. This corresponds to minimization of error rate, and since this kind of cross-validation is much more commonly practised than F-measure maximization, we provide it as a baseline, with the intent of reinforcing experimentally our initial statement that accuracy maximization is grossly insufficient for any kind of F-measure optimization, and in particular, tends to return notoriously useless classifiers when confronted with unbalanced data.

8.3 Results

We now compare the three classifiers fitted above on each dataset. Figure 8.1 shows comparative F_β (for a range of β) of the two standard weighted support vector machines and the F-measure maximizing support vector machine on data A_1 (balanced). We give similar plots for data A_2 (unbalanced), B_1 (balanced) and B_2 (unbalanced) in Figures 8.2, 8.3 and 8.4 respectively.

Clearly, both in the balanced and unbalanced settings, the direct F-measure optimizer is at least holding its own against the cross-validated standard weighted support vector machine. Particularly in the presence of class imbalance, we observe the consistently superior performance of both of these classifiers over the cross-validated accuracy maximizing standard weighted support vector machine.

Indeed, often we notice a significant improvement in performance from direct F_β optimization over its standard weighted cross-validation counterpart, indicating that there is a substantial gain shown in using a step function approximation to minimize approximate misclassification counts instead of the standard linear error metric that minimizes misclassification distances. Though we expect this improvement to be entirely universal in theory, there are a few important reasons why it is not. Namely, the F-measure maximizing support vector machine must be implemented via a generic nonlinear optimizer as described in Section 8.2, and since the

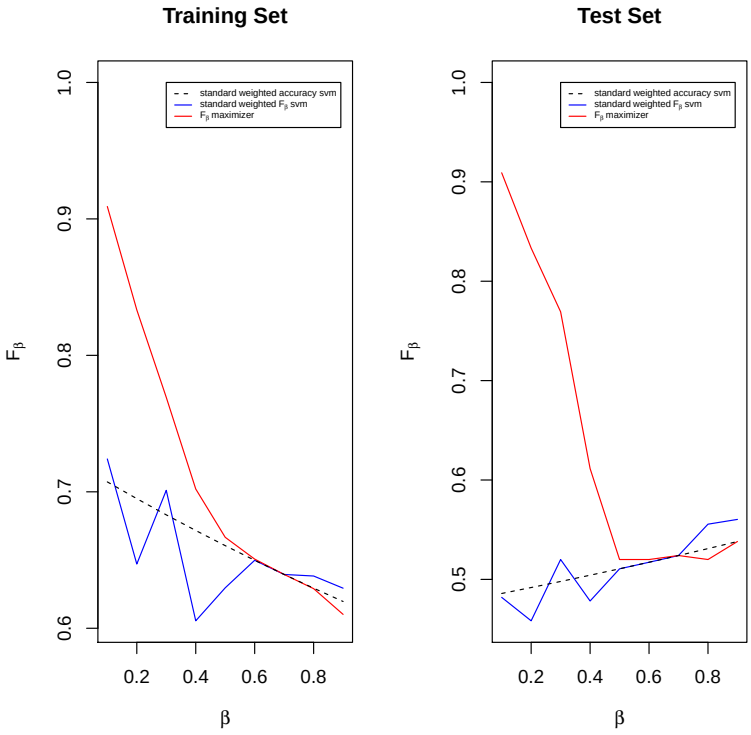

Figure 8.1: F_β on training and test set for synthetic balanced data A_1.

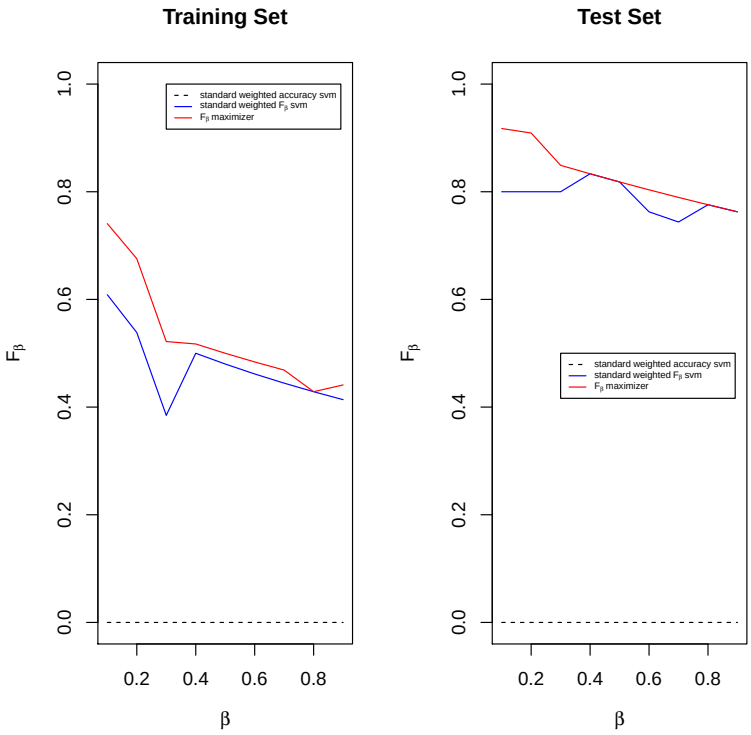

Figure 8.2: F_β on training and test set for synthetic unbalanced data A_2.

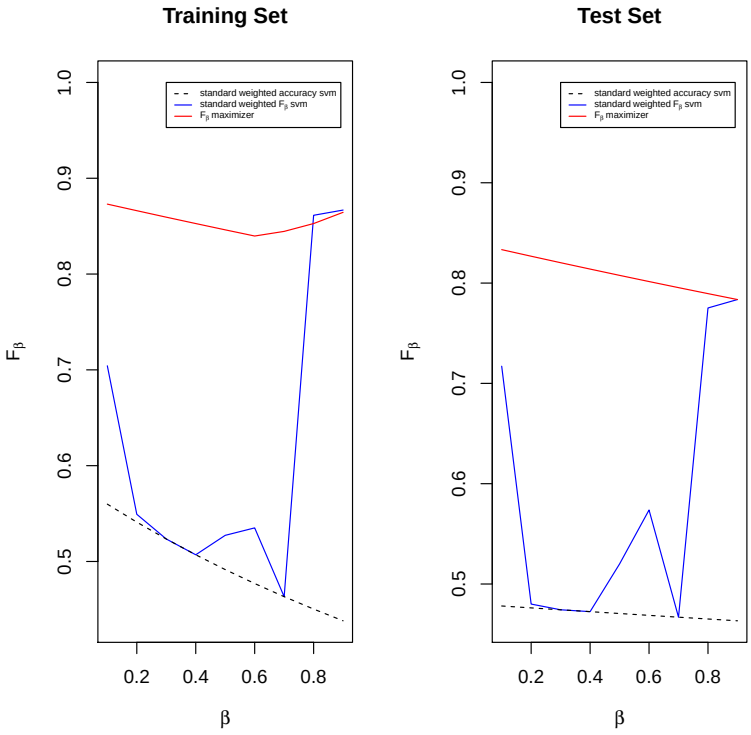

Figure 8.3: F_β on training and test set for synthetic balanced data B_1.

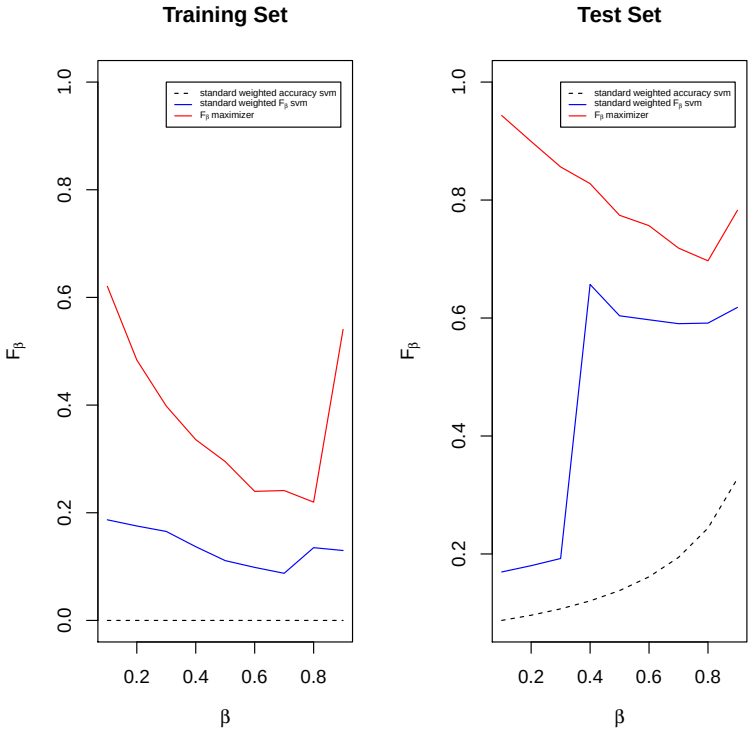

Figure 8.4: F_β on training and test set for synthetic unbalanced data B_2.

problem is highly nonlinear and even non-convex, there is not always a guarantee of local or global maximization. We also have that the F-measure maximizing support vector machine is cross-validated over a very small grid in only one of its parameters a (out of computational necessity, since nonlinear optimization is performed via an intensive algorithm), which does not allow the exhaustiveness of the search we perform in the standard case. Finally, the 2-fold cross-validation approach would normally be extended to at least a 5-fold cross-validation to further reduce the variance of the criterion, but computational demands necessitated reducing v, which, while still providing a compelling illustration of the performance of the F-measure maximizing support vector machine, may subject us to a higher variance than we hope for. These limitations of our F-measure maximizing approach indicate directions for immediate and future research, which we discuss in Chapter 10.

Chapter 9

Maximum Entropy Experiments

In this chapter, we test the performance of noisy F-measure optimization via uncertain maximum entropy, aiming to demonstrate in a number of ways the optimal performance of our algorithm as previously formulated in Chapter 7. In Section 9.1, we outline the data we use and the general framework of the experiments. Then in Section 9.2, we provide details of the design of the experiments to perform, as well as of the implementation. Finally, in Section 9.3, we give results of the performance and summarize the implications of these results.

9.1 Data

Our aim is to show that uncertain likelihood maximization works successfully, and that we can improve its performance in terms of noisy F-measure via Algorithm 7.1. We start by generating or loading the datasets below, and in Section 9.2 we will describe the addition of class noise to the data, representing the uncertainty we wish to model. We test our algorithm on three different datasets, which are as follows.

Synthetic data – A: We simulate a dataset of 1100 samples (x_i, y_i) with two classes, u (positive class) and $\bar{u}$ (negative class), and two features. Each class contains 550 samples, distributed as spherical Gaussians in the space of features. The samples from class $\bar{u}$ are distributed as $\mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = (0.5, 1)$ and $\Sigma_0 = (0.3, 0.3)^T I_2$. Class u is generated by $\mathcal{N}(\mu_1, \Sigma_1)$, with $\mu_1 = (1, 0.8)$ and $\Sigma_1 = (0.3, 0.3)^T I_2$. We use 1000 examples for training and 100 for testing.

Synthetic data – B: We simulate a dataset of 1100 samples (x_i, y_i) with two classes, u and $\bar{u}$, and two features. Each class contains 550 samples, distributed as elliptical Gaussians in the space of features. The samples from class $\bar{u}$ are distributed as $\mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = (2, 1)$ and $\Sigma_0 = (1, 0.3)^T I_2$. Class u is generated by $\mathcal{N}(\mu_1, \Sigma_1)$, with $\mu_1 = (1, 2)$ and $\Sigma_1 = (0.3, 1)^T I_2$. We use 1000 examples for training and 100 for testing.

Titanic data: We consider the famous "Titanic" dataset (taken from [24]) with 1309 samples, two classes and five features. There are 500 samples in the positive class and 809 samples in the negative class. We use 1000 examples for training and 309 for testing. The data itself concerns the survival of passengers on the Titanic. The positive class contains passengers who survived and the negative class contains passengers who did not survive. The features we use are passenger class, age, sex and number of siblings/spouses aboard, as well as an interaction effect between passenger class and sex, since it is well-known that in the sinking of the Titanic, passenger class showed a larger difference in survival rate amongst women compared to men (that is, women of a higher passenger class were much more likely

to survive than those of a lower class, whereas this division was more marginal amongst men). It should be noted that we present this dataset only as an example of data with some non-trivial features to demonstrate the "natural" performance of our algorithm. That is, we could give the uncertain interpretation of the Titanic dataset to represent a short-term after the fact measure of certainty of whether a given passenger is currently alive or dead, but we mostly eschew such interpretation, focusing instead directly on performance, which itself gives an indication of the potential of our method to be used for much more practical applications in the future.

9.2 Experimental Setup

We now elaborate on the precise nature of our uncertain maximum entropy experiments. We take a set of data with theoretically deterministic classes as above. We then define new positive and negative classes c_1 and c_0, and the class random variables $Y_i \sim q_i$ corresponding to each training example and for some given matrix of class Bernoulli distributions q. Actually, we generate this prior probability matrix q ourselves in the following way. We consider class-flipping probabilities (positive u to negative c_0 or negative $\bar{u}$ to positive c_1) given by arbitrary β distributions. That is, we flip points x_i which theoretically belong to class u into class c_0 with probabilities $P(Y_i = c_0 | \, y_i = u) = q_{i0} \sim \beta(1, 5)$, and we flip points x_i which theoretically belong to class $\bar{u}$ into class c_1 with probabilities $P(Y_i = c_1 | \, y_i = \bar{u}) = q_{i1} \sim \beta(2, 5)$. Now we have the uncertain training set (x_i, Y_i) with given prior class probability matrix q.

We visualize the most likely deterministic realization of the above on samples from datasets A and B in Figure 9.1, using 100 data points. That is, for the purposes of visualization, we fix a deterministic class for each example x_i corresponding to the higher value of $q_i = (q_{i1}, q_{i0})$

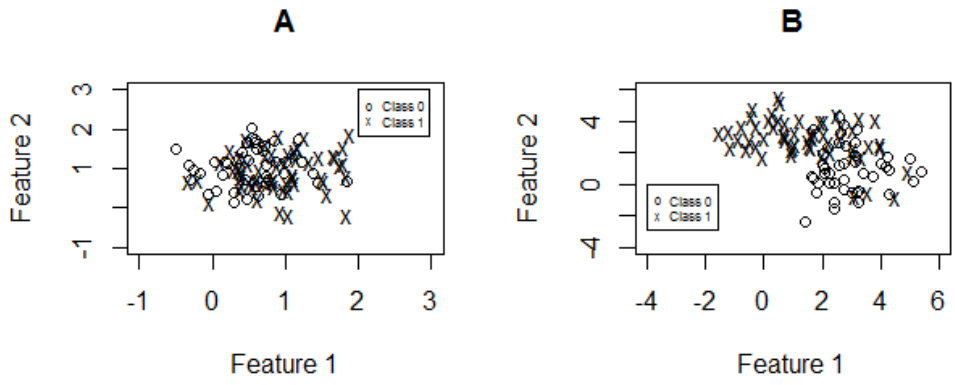

Figure 9.1: Most likely realization of samples in the space of features for synthetic datasets A and B.

As expounded in Chapter 5, the main interpretation we give to class uncertainty is that we have a panel of annotators, some of whom thought that the point x_i belonged to the class c_1 and others who thought that x_i belonged to c_0. The

proportion of opinions is then given by row q_i. Within this realistic framework, we consider there to be no necessary "ground truth" (even though originally we contrived the data as having belonged to certain true distributions).

In the absence of the results of uncertain classification presented in this thesis, we would ordinarily perform deterministic maximum entropy classification, setting for example:

$$y_i = \begin{cases} 1 & \text{if } q_{i1} \geq 0.5 \\ 0 & \text{if } q_{i0} > 0.5. \end{cases} \tag{9.1}$$

(We could alternatively apply some threshold other than 0.5.) We might even think naturally to fit a weighted maximum entropy based on the certainty of our observations, i.e. assigning weights $w_i = \max(q_{i1}, q_{i0})$, or some similar setup that gives more weight to data points with lower class entropy and less weight to data points with higher class entropy. There are a few possible ways of going about this, which we need not really consider, since our interest is in maximizing the noisy F-measure and we have an algorithm to do it. The details are as follows, and we perform each of the below for a range of $\beta \in [0, 1]$.

We fit four models – deterministic maximum entropy (via thresholding as in (9.1)), F-measure maximizing deterministic maximum entropy (again via thresholding), uncertain maximum entropy and noisy F-measure maximizing uncertain maximum entropy. We consider the performance of each of these models on the training and test set, with respect to both F-measure and noisy F-measure. We expect that the deterministic models will perform well with respect to deterministic F-measure and the uncertain models will perform well with respect to noisy F-measure, since the deterministic models attempt to maximize model probabilities corresponding to deterministic classes, and the uncertain models attempt to minimize the discrepancy between model class distributions and given experimental class distributions.

Additionally, we sample $B = 1000$ times from the class distributions q and test the average deterministic F-measure of the fitted classifiers over all resamples. The intuition here is that an uncertain classifier, trained with the knowledge of the randomness inherent in the training sample, will give a generally better long-term deterministic F-measure on test data drawn from these distributions. If we return again to the interpretation of the uncertain likelihood as a deterministic likelihood of an infinite number of samples from the Bernoulli distributions q_i of random classes Y_i, the F-measure of this deterministic likelihood maximizer should approximate the noisy F-measure of the uncertain likelihood maximizer. This means indeed that noisy F-measure maximization should give optimal deterministic F-measure on a large number of samples from the distributions q.

Implementations are again carried out in R, with all maximum entropy classifiers being fitted via logistic regression in the `glm` function.

9.3 Results

We now provide the results of the experiments described. In Figure 9.2, we compare the performance (F-measure and noisy F-measure) of standard deterministic

and uncertain classifiers with the performance of the F-measure maximizing deterministic classifier and the noisy F-measure maximizing uncertain classifier on data A.

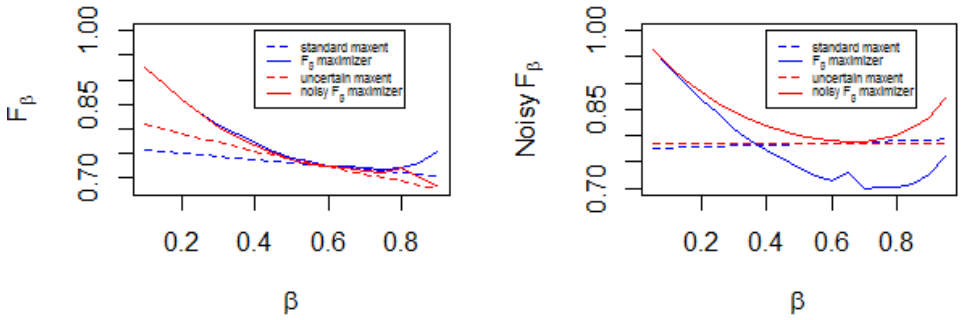

Figure 9.2: F_β and $\tilde{F}_\beta^U$ on the training set for synthetic data A.

The noisy F-measure maximizing uncertain maximum entropy is giving a significantly higher noisy F-measure on the training set than any of the other classifiers. While its performance in terms of F-measure is comparable with that of the deterministic F-measure maximizer for some values of β, it is clearly not performing to the same standard, as expected.

In Figure 9.3, we examine the predictive performance of the noisy F-measure maximizing algorithm, comparing it with the baseline unweighted uncertain maximum entropy model.

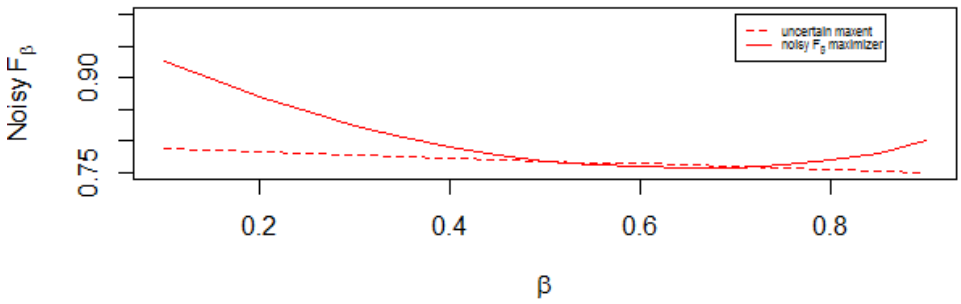

Figure 9.3: $\tilde{F}_\beta^U$ on the test set for synthetic data A.

We see that indeed the $\tilde{F}_\beta^U$ maximizer is giving improved performance on the test set over the baseline, and is able to reproduce the class randomness inherent in the training distributions in a very useful manner for predictive purposes. We observe that both on the training and test set, there is a point around $\beta = 0.6$

where the performance of our algorithm approximately meets that of its baseline. Additionally, while we did not expect overfitting to occur, it may be possible to be further minimized by introducing a regularization term into the uncertain likelihood. We leave this extension for the future.

In Figure 9.4, we test the average performance of our classifiers on $B = 1000$ datasets resampled via the distributions q. We have here fitted the uncertain likelihood classifier with weights calculated by plugging deterministic confusion counts and F-measure in the place of their uncertain counterparts. The improvement in using the noisy F-measure maximizing algorithm instead of the standard uncertain likelihood is seen particularly emphatically when we do this, evident for most values of β. The intuition behind the plug-in algorithm is simply that we are tailoring it towards the deterministic measure while approximately reproducing the observed class randomness.

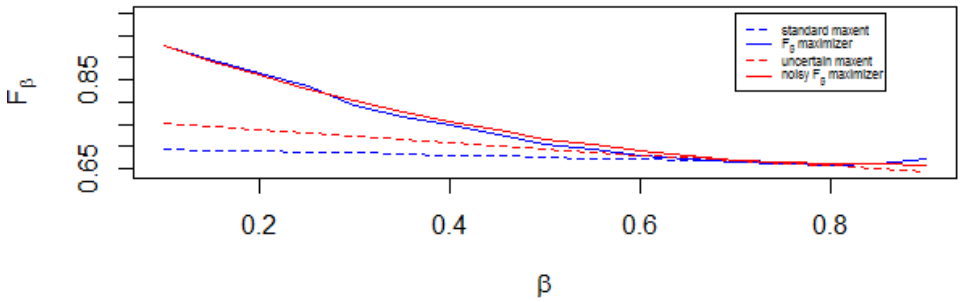

Figure 9.4: Average F_β on 1000 resamples from q on synthetic data A, with $\tilde{F}_\beta^U$ maximizer fitted with weights chosen via deterministic F_β plug-in algorithm.

In Figure 9.5, we compare the performance (F-measure and noisy F-measure) of standard deterministic and uncertain classifiers with the performance of the F-measure maximizing deterministic classifier and the noisy F-measure maximizing uncertain classifier on data B. In Figure 9.6, we examine the predictive performance of the noisy F-measure maximizing algorithm, comparing it with the baseline unweighted uncertain maximum entropy model. In Figure 9.7, we test the average performance of our classifiers on $B = 1000$ datasets resampled via the distributions q. We have again fitted the uncertain likelihood classifier with weights calculated by plugging deterministic confusion counts and F-measure in the place of their uncertain counterparts. The improvement in using the noisy F-measure maximizing algorithm instead of the standard uncertain likelihood is seen particularly emphatically when we do this, evident for most values of β.

The conclusions for these results in the case of data B are basically identical to those we made for data A. Note that our algorithm's test set performance compared to the baseline is even better on data B.

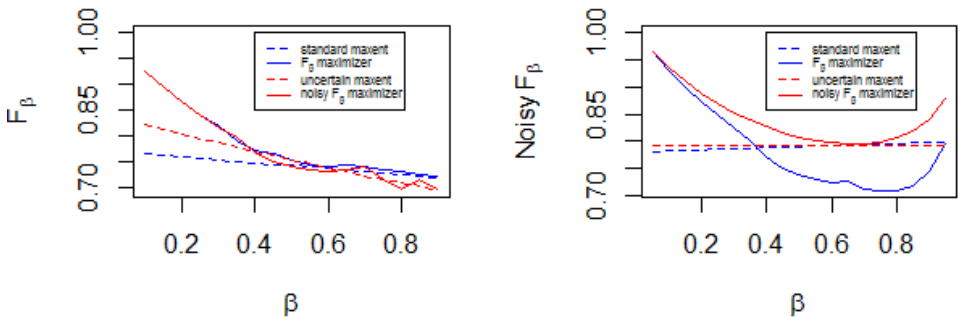

Figure 9.5: F_β and $\tilde{F}_\beta^U$ on the training set for synthetic data B.

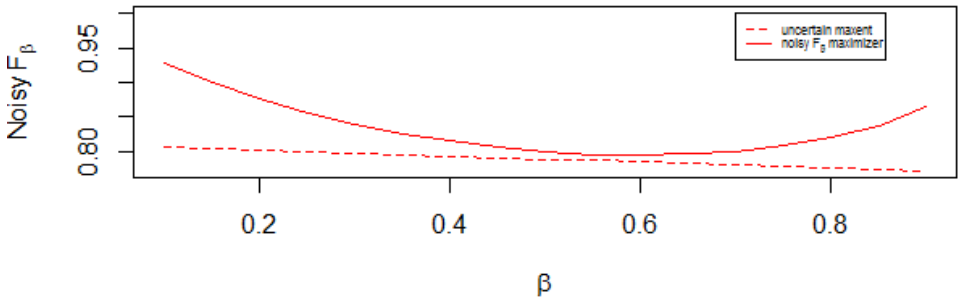

Figure 9.6: $\tilde{F}_\beta^U$ on the test set for synthetic data B.

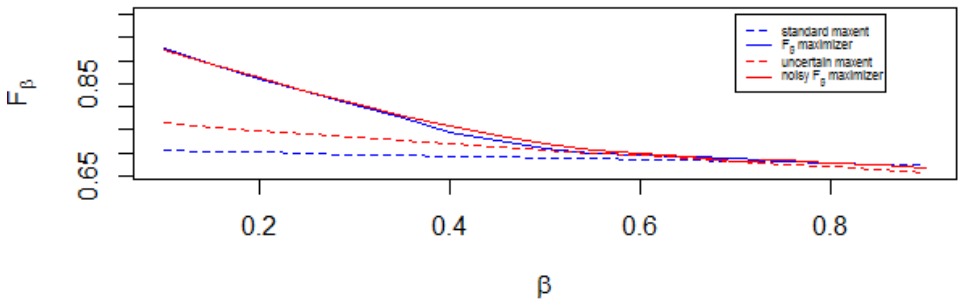

Figure 9.7: Average F_β on 1000 resamples from q on synthetic data B, with $\tilde{F}_\beta^U$ maximizer fitted with weights chosen via deterministic F_β plug-in algorithm.

In Figure 9.8, we give a typical learning curve for the $\tilde{F}_\beta^U$ maximizing algorithm. It demonstrates that the maximum $\tilde{F}_\beta^U$ is reached quite quickly, typically for our examples in about 20 iterations.

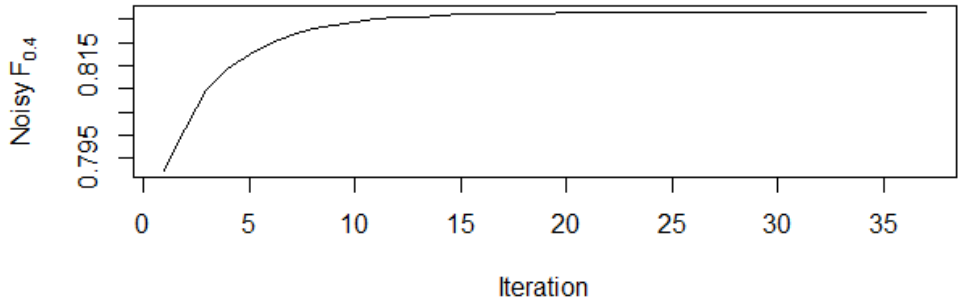

Figure 9.8: Typical learning curve for $\tilde{F}_\beta^U$ maximization, here given for $\beta = 0.4$ on the dataset B.

In Figure 9.9, we give also a plot comparing the $\tilde{F}_\beta^U$ curve for two different values of the `maxit` parameter in the logistic regression implemented in R. This parameter controls the maximum number of iterations in the likelihood maximization algorithm employed by R. The plot demonstrates that using only one step in their algorithm, which our algorithm guarantees to find a local maximum, does well, but not as well as using a much larger number of maximum iterations (`maxit=25`). That is, by doing a complete likelihood maximization at every step of our algorithm, we are ending up with not only a local maximum, but indeed a better local maximum than we were guaranteed to find by doing only one step, and perhaps a global maximum.

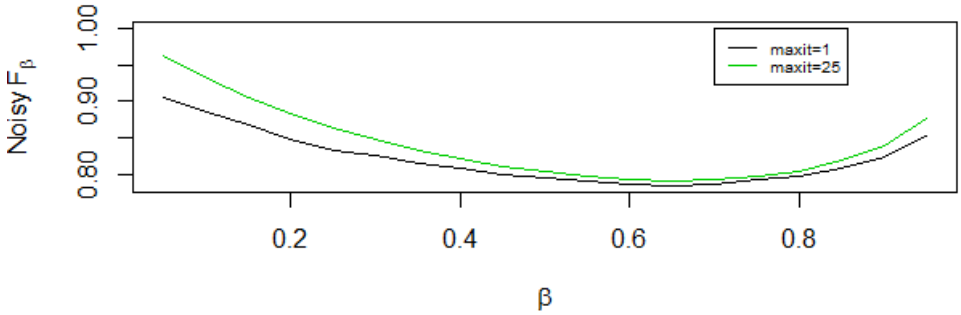

Figure 9.9: Comparative $\tilde{F}_\beta^U$ curves for 1 and 25 steps of uncertain likelihood maximization at each iteration of the $\tilde{F}_\beta^U$ maximizing algorithm, here given on the dataset B.

In Figure 9.10, we compare the performance (F-measure and noisy F-measure) of standard deterministic and uncertain classifiers with the performance of the F-measure maximizing deterministic classifier and the noisy F-measure maximizing uncertain classifier on the Titanic data.

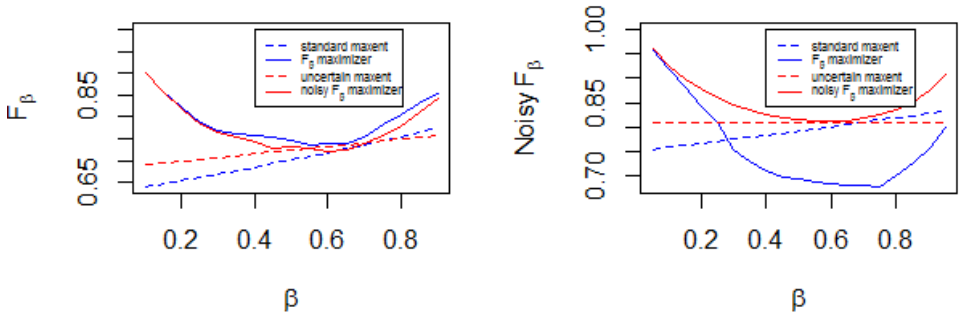

Figure 9.10: F_β and $\tilde{F}_\beta^U$ on the training set for Titanic data.

In Figure 9.11, we examine the predictive performance of the noisy F-measure maximizing algorithm, comparing it with the baseline unweighted uncertain entropy model.

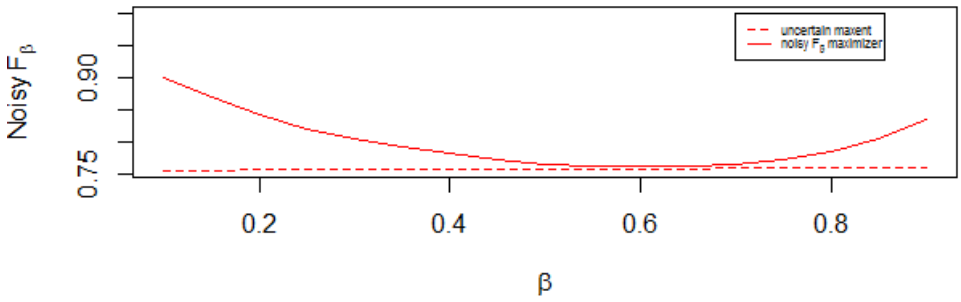

Figure 9.11: $\tilde{F}_\beta^U$ on the test set for Titanic data.

The conclusions for our results in the case of the Titanic data are basically identical to those we made for data A and B, indicating that our algorithm is ready to be employed in solving real-world problems.

Chapter 10

Summary and Final Comments

In this thesis, we have seen that the importance of considering the F-measure in classification problems is that it is able to define a flexible trade-off between precision and recall, controlled by the parameter β to be adjusted to the scenario we are in on a practical level. While improvements in F-measure necessarily improve the accuracy of a classifier, a classifier with high accuracy could nevertheless have grossly suboptimal F-measure. This problem becomes particularly apparent when classes are unbalanced. Support vector machines and maximum entropy models are both very popular classifiers, thus the question of how to perform F-measure optimization for each is a natural one to consider. Much work has been done in these areas, but much is left to be done.

The main importance and implications of our work in support vector machines are as follows. We took the F-measure maximizing support vector machine defined in [20], and generalized it to choice of β, implementing it using a nonlinear optimization solver and deriving a dual problem for the equivalent weighted misclassification count support vector machine. This dual problem theoretically allows for nonlinear classification via kernel methods, if a generalized convexity holds. Our favourable results for a primitive implementation motivate the use of the F-measure maximizing support vector machine, as well as necessitating further investigation into its theoretical properties and easier methods of solution, so that for example we could guarantee reaching a global minimum in (w, b, ξ), even though it is not a convex problem.

We summarize the importance and implications of our work in maximum entropy classification as follows. We took the foundation introduced by [8], which offers a direct F-measure maximizing algorithm via a weighted maximum likelihood. We extended this framework to the uncertain setting, where classes are not provided deterministically, but instead there exist probabilities that an example belongs to each class. Given such a framework, we devised a noisy F-measure, which is a measure of performance analogous to the F-measure in the uncertain context. Optimizing the noisy F-measure involves minimizing average Kullback-Leibler divergences between true and model distributions, which we showed can be performed via a weighted maximum uncertain likelihood, and we provided experimental evidence for the resulting algorithm's success.

There is much scope for future work. In the support vector machine framework, we may be interested in defining the problem of uncertain classification, perhaps along the lines of [15], in order to consider noisy F-measure maximization here as well. We also would like to investigate further the choice of parameters (C, a) in the F-measure maximizing support vector machine. We realize that choice of

parameter C even for the standard support vector machine is a huge problem, and no adequate theory exists for choosing it to fit optimally without using cross-validation, a very computationally costly technique that carries with it additional limitations. Perhaps it would then be more fruitful to focus on devising a different choice of step function approximation s, one which might provide the guarantee of global minimization and for which methods from convex optimization could be directly applied. This of course would be highly preferable.

In the maximum entropy framework, there is the immediate question of how to choose the number of steps in the uncertain likelihood maximization performed at each iteration of our noisy F-measure maximizing algorithm, in order ideally to reach a global maximum. We also in an upcoming work will consider a generalization of the deterministic F-measure maximizing algorithm in [8] to the multiclass case, and this may lead to a similar natural extension of the noisy F-measure maximizing algorithm. We additionally note that in setting up the noisy F-measure we have chosen to parameterize the dependence between true and modelled classes in a very specific way. However, as we mentioned, there does exist an "uncorrelated" noisy F-measure, as well as probably other ways of representing the structure of dependence between experimental and model class distributions. This warrants further investigation, in particular to answer the question of whether there is ever a "best" such formulation.

Appendix A

R Code

A.1 R Implementation: Support Vector Machines

The following code generates data (with an easy way of modifying the balance of classes), then finds the standard weighted support vector machine optimal with respect to each F_β via cross-validation, as well as the F-measure maximizing support vector machine for a particular choice of C and cross-validation over a small grid for a. It plots F_β across values of β with an additional F_β curve for the accuracy-maximizing support vector machine with parameters chosen via cross-validation.

```
rm(list = ls())

library(nloptr)
library(kernlab)
library(MASS)

#GENERATE DATA

#let the positive class be the rare one (i.e. mminus>=mplus)
mplus<-200
mminus<-200
m<-mminus+mplus

#data A

mu1<-c(1,0.8)
sigma1<-diag(c(0.3,0.3))
x1<-mvrnorm(mplus,mu1,sigma1)
y1<-rep(1,mplus)

mu2<-c(0.5,1)
sigma2<-diag(c(0.3,0.3))
x2<-mvrnorm(mminus,mu2,sigma2)
y2<-rep(-1,mminus)

#data B

#mu1<-c(1,2)
#sigma1<-diag(c(0.3,1))
```

```
#x1<-mvrnorm(mplus,mu1,sigma1)
#y1<-rep(1,mplus)

#mu2<-c(2,1)
#sigma2<-diag(c(1,0.3))
#x2<-mvrnorm(mminus,mu2,sigma2)
#y2<-rep(-1,mminus)

x<-rbind(x1,x2)
y<-c(y1,y2)

#plot with classes
plot.new()
plot.window(xlim=c(-1,3),ylim=c(-1,3))
for (i in 1:m) {
  if (y[i]==1) {
    points(matrix(x[i,],nrow=1),pch="x")
  } else {
    points(matrix(x[i,],nrow=1))
  }
}
axis(1)
axis(2)
box()

#test set
mplustestset<-25
mminustestset<-25
mtestset<-mplustestset+mminustestset
x1<-mvrnorm(mplustestset,mu1,sigma1)
y1<-rep(1,mplustestset)
x2<-mvrnorm(mminustestset,mu2,sigma2)
y2<-rep(-1,mminustestset)
xtestset<-rbind(x1,x2)
ytestset<-c(y1,y2)

#FUNCTIONS

#calculate confusion matrix given modelled and true classes
confusionCalc<-function(modelClass,trueClass) {
  #confusion counts
  tp<-sum(modelClass==1 & trueClass==1)
  fp<-sum(modelClass==1 & trueClass==-1)
  fn<-sum(modelClass==-1 & trueClass==1)
  tn<-sum(modelClass==-1 & trueClass==-1)
  #confusion matrix
  c("TP"=tp,"FP"=fp,"FN"=fn,"TN"=tn)
```

```
}

#calculate f-measure from confusion counts
fConfusionCalc<-function(confusions,beta) {
  #confusion counts
  tp<-confusions[1]
  fp<-confusions[2]
  fn<-confusions[3]
  tn<-confusions[4]
  #precision and recall
  prec<-tp/(tp+fp)
  rec<-tp/(tp+fn)
  #f-measure
  1/(beta/prec + (1-beta)/rec)
}

#calculate f-measure given modelled and true classes
fBetaCalc<-function(modelClass,trueClass,beta) {
  #confusion counts
  tp<-sum(modelClass==1 & trueClass==1)
  fp<-sum(modelClass==1 & trueClass==-1)
  fn<-sum(modelClass==-1 & trueClass==1)
  tn<-sum(modelClass==-1 & trueClass==-1)
  #precision and recall
  prec<-tp/(tp+fp)
  rec<-tp/(tp+fn)
  #f-measure
  1/(beta/prec + (1-beta)/rec)
}

#svm decisions
decision<-function(x,data,y,alpha,bstar) {
  terms<-c()
  for (i in 1:length(y)) {
    terms[i]<-y[i]*alpha[i]*sum(x*data[i,])
  }
  f<-sum(terms)+bstar
  sign(f)
}

#svm decision values
decisionvalue<-function(x,data,y,alpha,bstar) {
  terms<-c()
  for (i in 1:length(y)) {
    terms[i]<-y[i]*alpha[i]*sum(x*data[i,])
  }
  f<-sum(terms)+bstar
```

```
  f
}

#svm decisions given primal solution (w,b)
directdecision<-function(x,w,bstar) {
  f<-sum(w*x) + bstar
  sign(f)
}

#standard weighted svm solver
svmFunc<-function(x,y,Cplus,Cminus,maxIt) {

  m<-length(y)

  #input parameters in ipop
  b<-0
  r<-0
  e<-matrix(rep(1,times=m))
  c<--e
  l<-matrix(rep(0,times=m))
  A<-t(y)

  H<-matrix(data=NA,nrow=m,ncol=m)
  for (i in 1:m) {
    for (j in 1:m) {
      H[i,j]<-y[i]*y[j]*sum(x[i,]*x[j,])
    }
  }

  u<-c()
  for (i in 1:m) {
    if (y[i]==1) {
      u[i]<-Cplus
    } else if (y[i]==-1) {
      u[i]<-Cminus
    }
  }
  u<-matrix(u)

  #optimization to find required coeffs
  alphacoeffs<-primal(ipop(c,H,A,b,l,u,r,maxiter=maxIt))
  for (i in 1:m) {
    if (alphacoeffs[i]<0) {
      alphacoeffs[i]<-0
    } else if (alphacoeffs[i]>u[i]) {
      alphacoeffs[i]<-u[i]
    }
```

```
  }

  #finding offset b
  mucoeffs<-u-alphacoeffs
  H2<-matrix(data=NA,nrow=m,ncol=m)
  for (i in 1:m) {
    for (j in 1:m) {
      H2[i,j]<-mucoeffs[i]*y[j]*sum(x[i,]*x[j,])
    }
  }
  bstar<-((sum(alphacoeffs*mucoeffs*y)
           - (t(alphacoeffs) %*% H2 %*% alphacoeffs))
          /sum(alphacoeffs*mucoeffs))

  #w
  w1<-sum(alphacoeffs*y*x[,1])
  w2<-sum(alphacoeffs*y*x[,2])
  w<-c(w1,w2)

  #decision values
  decisionvaltrain<-c()
  for (i in 1:m) {
    decisionvaltrain[i]<-decisionvalue(x[i,],x,y,alphacoeffs,bstar)
  }

  #classification errors
  xi<-pmax(0,1-y*decisionvaltrain)

  list("alpha"=alphacoeffs,"w"=w,"bstar"=bstar,"xi"=xi)

}

#f-measure maximizing svm objective
obj<-function(x,data,y,cost,a,beta) {

  stepapprox<-function(x,a) {
    2*(1/(1+exp(-a*x)) - 0.5)
  }

  m<-length(y)
  mplus<-sum(y==1)
  n<-ncol(data)

  w<-x[1:n]
  b<-x[n+1]
  xi<-x[n+2:(m+1)]
```

```
  (0.5*sum(w*w)
   + cost*(beta*sum((y==-1)*(stepapprox(xi,a)))
           + (1-beta)*sum((y==1)*(stepapprox(xi,a))))
   /(mplus-sum((y==1)*stepapprox(xi,a))))

}

#f-measure maximizing svm inequality constraints
ineq<-function(x,data,y,cost,a,beta) {

  m<-length(y)
  n<-ncol(data)

  w<-x[1:n]
  b<-x[n+1]
  xi<-x[n+2:(m+1)]

  ineq1<-c()
  for (i in 1:m) {
    ineq1[i]<-(1-xi[i]) - y[i]*(sum(data[i,]*w)+b)
  }

  ineq1

}

svmOptFunc<-function(w,bstar,xi,x,y,cost,a,beta) {

  n<-ncol(x)
  m<-length(y)
  wbxi<-c(w,bstar,xi)

  #bounds
  lb<-c(rep(-max(ceiling(abs(w))),n),-ceiling(abs(bstar)),rep(0,m))
  ub<-c(rep(ceiling(max(abs(w))),n),ceiling(abs(bstar)),
        rep(ceiling(max(xi)),m))

  #optimization to find the required coeffs
  wbxinew<-nloptr(x0=wbxi,
                  eval_f=obj,
                  eval_g_ineq=ineq,
                  lb=lb,
                  ub=ub,
                  opts=opts,
                  data=x,
                  y=y,
                  cost=cost,
```

```
                  a=a,
                  beta=beta)

  wnew<-wbxinew$solution[1:n]
  bnew<-wbxinew$solution[n+1]
  xinew<-wbxinew$solution[n+2:(m+1)]

  list("w"=wnew,"bstar"=bnew,"xi"=xinew)

}

#DEFINITIONS

n<-ncol(x)

#beta parameter in f-measure
betaSeq<-seq(0.1,0.9,by=0.1)

#options for nloptr
opts<-list("algorithm"="NLOPT_LN_COBYLA",
           "ftol_rel"=1.0e-4,
           "maxeval"=500)

#cost in nloptr
Cnew<-mplus

#cross-validation subsetting
istar<-sample(1:m)
xstar<-x[istar,]
ystar<-y[istar]
#number of folds on which to perform cv
v<-2
#size of cv validation set
mtest<-m/v
#size of cv training set
mfold<-m-mtest

#parameters Cplus, Cminus for grid search
Cplusgrid<-c()
Cminusgrid<-c()
ncost<-20
for (ci in 1:ncost) {
  Cplusgrid[ci]<-2^(-4+ci/2)
  Cminusgrid[ci]<-2^(-4+ci/2)
}

#parameter a for grid search
```

```
agrid<-c(0.1,1,5,10,30)

#f-measure definitions
fTrainStdCV<-c()
fTestStdCV<-c()
fTrainOpt<-c()
fTestOpt<-c()
fTrainAccCV<-c()
fTestAccCV<-c()

accscore<-matrix(data=NA,nrow=ncost,ncol=ncost)

#MAIN LOOP

for (betaK in 1:length(betaSeq)) {

  beta<-betaSeq[betaK]

  #STANDARD WEIGHTED SVM CV

  cvscore<-matrix(data=NA,nrow=ncost,ncol=ncost)

  #2 loops for 2-dimensional grid search over Cplus, Cminus
  for (ci in 1:ncost) {
    for (cj in 1:ncost) {

      #only checking parameter values where Cplus >= Cminus
      if (ci>=cj) {

        Cplus<-Cplusgrid[ci]
        Cminus<-Cminusgrid[cj]

        #confusion counts and accuracy on each fold
        conf<-matrix(data=NA,nrow=v,ncol=4)
        accfold<-c()

        #iterating to perform cv
        for (cvi in 1:v) {

          #training data to test on cvi'th fold
          xfold<-xstar[-((cvi-1)*mtest + 1:mtest),]
          yfold<-ystar[-((cvi-1)*mtest + 1:mtest)]
          mplusfold<-sum(yfold==1)
          mminusfold<-mfold-mplusfold

          #testing data
          xtest<-xstar[(cvi-1)*mtest + 1:mtest,]
```

```
      ytest<-ystar[(cvi-1)*mtest + 1:mtest]

      model.svm<-svmFunc(xfold,yfold,Cplus,Cminus,1)
      alphacoeffs<-model.svm$alpha
      bstar<-model.svm$bstar

      #predictions
      decisiontest<-c()
      for (i in 1:mtest) {
        decisiontest[i]<-decision(xtest[i,],xfold,yfold,alphacoeffs,
                                  bstar)
      }

      #confusion quantities
      conf[cvi,]<-confusionCalc(decisiontest,ytest)

      #saving accuracy
      if (betaK==1) {
        accfold[cvi]<-(conf[cvi,][1]+conf[cvi,][4])/mfold
        accfold[cvi]<-replace(accfold[cvi],is.na(accfold[cvi]),0)
      }

    }

    #cv scores
    confCV<-colMeans(conf)

    fmeas<-fConfusionCalc(confCV,beta)
    fmeas<-replace(fmeas,is.na(fmeas),0)
    cvscore[ci,cj]<-fmeas

    #acc score
    if (betaK==1) {
      accscore[ci,cj]<-mean(accfold)
    }

  } else {

    cvscore[ci,cj]<-0
    if (betaK==1) {
      accscore[ci,cj]<-0
    }

  }

 }
}
```

```
#optimal Cplus, Cminus
cvmax<-max(cvscore)
coptvals<-which(cvscore==cvmax,arr.ind=T)[1,]
Cplusopt<-Cplusgrid[coptvals[1]]
Cminusopt<-Cminusgrid[coptvals[2]]

model.svm<-svmFunc(x,y,Cplusopt,Cminusopt,1)
alphacoeffs<-model.svm$alpha
bstar<-model.svm$bstar

#performance on training set
decisiontrain<-c()
for (i in 1:m) {
  decisiontrain[i]<-decision(x[i,],x,y,alphacoeffs,bstar)
}

fTrainStdCV[betaK]<-fBetaCalc(decisiontrain,y,beta)

#performance on test set
decisiontest<-c()
for (i in 1:mtestset) {
  decisiontest[i]<-decision(xtestset[i,],x,y,alphacoeffs,bstar)
}

fTestStdCV[betaK]<-fBetaCalc(decisiontest,ytestset,beta)

#STANDARD WEIGHTED SVM GUESS

Cplus<-mminus/10
Cminus<-mplus/10

model.svm<-svmFunc(x,y,Cplus,Cminus,1)
bstar<-model.svm$bstar
xi<-model.svm$xi
w<-model.svm$w

#F-MEASURE MAXIMIZING SVM

wbxifold<-matrix(data=NA,nrow=n+1+mfold,ncol=v)

#first guesses for cv
for (cvi in 1:v) {

  #training data to test cvi'th fold
  xfold<-xstar[-((cvi-1)*mtest + 1:mtest),]
  yfold<-ystar[-((cvi-1)*mtest + 1:mtest)]
```

```
  model.svm<-svmFunc(xfold,yfold,Cplus,Cminus,1)
  bstarfold<-model.svm$bstar
  xifold<-model.svm$xi
  wfold<-model.svm$w

  wbxifold[,cvi]<-c(wfold,bstarfold,xifold)

}

cvscore<-c()

for (aK in 1:length(agrid)) {

  #smoothing parameter in step function approximation
  a<-agrid[aK]

  conf<-matrix(data=NA,nrow=v,ncol=4)

  for (cvi in 1:v) {

    #training data to test cvi'th fold
    xfold<-xstar[-((cvi-1)*mtest + 1:mtest),]
    yfold<-ystar[-((cvi-1)*mtest + 1:mtest)]

    #validation data
    xtest<-xstar[(cvi-1)*mtest + 1:mtest,]
    ytest<-ystar[(cvi-1)*mtest + 1:mtest]

    wbxiCV<-wbxifold[,cvi]
    wCV<-wbxiCV[1:n]
    bCV<-wbxiCV[n+1]
    xiCV<-wbxiCV[n+2:(mfold+1)]

    #f-measure maximizing svm with nloptr
    optCV.svm<-svmOptFunc(wCV,bCV,xiCV,xfold,yfold,Cnew,a,beta)
    wnew<-optCV.svm$w
    bnew<-optCV.svm$bstar
    xinew<-optCV.svm$bstar

    #decisions on validation data
    decisiontest<-c()
    for (i in 1:mtest) {
      decisiontest[i]<-directdecision(xtest[i,],wnew,bnew)
    }

    #defining confusion quantities and computing f-measure on test set
```

```
      conf[cvi,]<-confusionCalc(decisiontest,ytest)

    }

    confCV<-colMeans(conf)

    fmeas<-fConfusionCalc(confCV,beta)
    fmeas<-replace(fmeas,is.na(fmeas),0)

    cvscore[aK]<-fmeas

  }

  aOpt<-agrid[which.max(cvscore)]

  opt.svm<-svmOptFunc(w,bstar,xi,x,y,Cnew,aOpt,beta)
  wnew<-opt.svm$w
  bnew<-opt.svm$bstar
  xinew<-opt.svm$xi

  #performance on training set
  decisiontrain<-c()
  for (i in 1:m) {
    decisiontrain[i]<-directdecision(x[i,],wnew,bnew)
  }

  fTrainOpt[betaK]<-fBetaCalc(decisiontrain,y,beta)

  #performance on test set
  decisiontest<-c()
  for (i in 1:mtestset) {
    decisiontest[i]<-directdecision(xtestset[i,],wnew,bnew)
  }

  fTestOpt[betaK]<-fBetaCalc(decisiontest,ytestset,beta)

}

#STANDARD WEIGHTED SVM WITH CV ON ACCURACY

accmax<-max(accscore)
coptvals<-which(accscore==accmax,arr.ind=T)[1,]
Cplusopt<-Cplusgrid[coptvals[1]]
Cminusopt<-Cminusgrid[coptvals[2]]

model.svm<-svmFunc(x,y,Cplusopt,Cminusopt,1)
alphacoeffs<-model.svm$alpha
```

```
bstar<-model.svm$bstar

#performance on training set
decisiontrain<-c()
for (i in 1:m) {
  decisiontrain[i]<-decision(x[i,],x,y,alphacoeffs,bstar)
}

for (betaK in 1:length(betaSeq)) {
  beta<-betaSeq[betaK]
  fTrainAccCV[betaK]<-fBetaCalc(decisiontrain,y,beta)
}

#performance on test set
decisiontest<-c()
for (i in 1:mtestset) {
  decisiontest[i]<-decision(xtestset[i,],x,y,alphacoeffs,bstar)
}

for (betaK in 1:length(betaSeq)) {
  beta<-betaSeq[betaK]
  fTestAccCV[betaK]<-fBetaCalc(decisiontest,ytestset,beta)
}

fTrainStdCV<-replace(fTrainStdCV,is.na(fTrainStdCV),0)
fTestStdCV<-replace(fTestStdCV,is.na(fTestStdCV),0)
fTrainOpt<-replace(fTrainOpt,is.na(fTrainOpt),0)
fTestOpt<-replace(fTestOpt,is.na(fTestOpt),0)
fTrainAccCV<-replace(fTrainAccCV,is.na(fTrainAccCV),0)
fTestAccCV<-replace(fTestAccCV,is.na(fTestAccCV),0)

#PLOTS

yLowerTrain<-min(min(fTrainStdCV),min(fTrainOpt),min(fTrainAccCV))
yLowerTest<-min(min(fTestStdCV),min(fTestOpt),min(fTestAccCV))

par(mfrow=c(1,2))

plot(betaSeq,fTrainStdCV,type="l",col="blue",ylim=c(yLowerTrain,1),
     xlab=expression(beta),ylab=expression(F[beta]),main="Training Set")
lines(betaSeq,fTrainOpt,col="red")
lines(betaSeq,fTrainAccCV,lty=2)
#legend(0.3,1,c("standard weighted accuracy svm",
#               expression(standard~weighted~F[beta]~svm),
#               expression(F[beta]~maximizer)),
#       lty=c(2,1,1),col=c("black","blue","red"),cex=0.5)
```

```
plot(betaSeq,fTestStdCV,type="l",col="blue",ylim=c(yLowerTest,1),
     xlab=expression(beta),ylab=expression(F[beta]),main="Test Set")
lines(betaSeq,fTestOpt,col="red")
lines(betaSeq,fTestAccCV,lty=2)
#legend(0.3,1,c("standard weighted accuracy svm",
#               expression(standard~weighted~F[beta]~svm),
#               expression(F[beta]~maximizer)),
#       lty=c(2,1,1),col=c("black","blue","red"),cex=0.5)
```

A.2 R Implementation: Maximum Entropy Classification

The following code generates data (with an easy way of modifying the balance of classes) as well as the prior probability matrix q that defines the class distributions. Over a range of β, it performs unweighted and F-measure maximizing deterministic maximum entropy classification on the most-likely classes given by q, as well as unweighted and $\tilde{F}_\beta^U$ maximizing uncertain maximum entropy classification. We compare performance on training and test sets, and average performance on resampled test sets.

```
#supervised uncertain classification

#generate data with classes given by distribution q. train deterministic and
#uncertain classifiers on this data. compare performance on training and test
#set, as well as long-term behaviour on resamples.

rm(list = ls())

library(MASS)

#FUNCTIONS

#calculate deterministic f-measure given modelled and true classes
fBetaCalc<-function(modelClass,trueClass,beta) {

  #confusion counts
  tp<-sum(modelClass*trueClass)
  fp<-sum(modelClass*(1-trueClass))
  fn<-sum((1-modelClass)*trueClass)
  tn<-sum((1-modelClass)*(1-trueClass))

  #precision and recall
  prec<-tp/(tp+fp)
  rec<-tp/(tp+fn)

  #f-measure
  1/(beta/prec + (1-beta)/rec)
}
```

```
#GENERATING SYNTHETIC DATA

mplus<-500
mminus<-500
m<-mplus+mminus

#data A

#mu1<-c(1,0.8)
#sigma1<-diag(c(0.3,0.3))
#x1<-mvrnorm(mplus,mu1,sigma1)
#y1<-rep(1,mplus)

#mu2<-c(0.5,1)
#sigma2<-diag(c(0.3,0.3))
#x2<-mvrnorm(mminus,mu2,sigma2)
#y2<-rep(0,mminus)

#data B

mu1<-c(1,2)
sigma1<-diag(c(0.3,1))
sigma1[1,1] = 2;
x1<-mvrnorm(mplus,mu1,sigma1)
y1<-rep(1,mplus)

mu2<-c(2,1)
sigma2<-diag(c(1,0.3))
sigma2[2,2] = 2;
x2<-mvrnorm(mminus,mu2,sigma2)
y2<-rep(0,mminus)

x<-rbind(x1,x2)
y<-c(y1,y2)

#flip class 1 to 0 with probability errorClass1
#flip class 0 to 1 with probability 1-errorClass2
errorClass1 <- rbeta(mplus,1,5)
errorClass2 <- rbeta(mminus,5,2)

#prior probability matrix
#q_{i1} = q[i,1] and q_{i0} = q[i,2]
q<-matrix(data=NA,nrow=m,ncol=2)
for (i in 1:mplus) {
  q[i,1]<-1-errorClass1[i]
  q[i,2]<-errorClass1[i]
}
```

```
for (i in (mplus+1:mminus)) {
  q[i,1]<-1-errorClass2[i-mplus]
  q[i,2]<-errorClass2[i-mplus]
}

#vector of most likely classes
ydet<-c()
for (i in 1:m) {
  if (q[i,1]>q[i,2]) {
    ydet[i]<-1
  } else {
    ydet[i]<-0
  }
}

#test set

mplustest<-50
mminustest<-50
mtest<-mplustest+mminustest

x1<-mvrnorm(mplustest,mu1,sigma1)
y1<-rep(1,mplustest)
x2<-mvrnorm(mminustest,mu2,sigma2)
y2<-rep(0,mminustest)

xtest<-rbind(x1,x2)
ytest<-c(y1,y2)

#flip class 1 to 0 with probability prob1 (errorClass1)
#flip class 0 to 1 with probability prob2 (errorClass2)
errorClass1 <- rbeta(mplustest,1,5)
errorClass2 <- rbeta(mminustest,5,2)

#prior probability matrix
#q_{i1} = q[i,1] and q_{i0} = q[i,2]
qtest<-matrix(data=NA,nrow=mtest,ncol=2)
for (i in 1:mplustest) {
  qtest[i,1]<-1-errorClass1[i]
  qtest[i,2]<-errorClass1[i]
}
for (i in (mplustest+1:mminustest)) {
  qtest[i,1]<-1-errorClass2[i-mplustest]
  qtest[i,2]<-errorClass2[i-mplustest]
}

#resampling
```

```
B<-1000
ystar<-matrix(data=NA,nrow=m,ncol=B)
for (b in 1:B) {
  #sampling from q
  flip1<-rbinom(mplus,1,errorClass1)
  y1star<-y1-flip1
  flip2<-rbinom(mminus,1,1-errorClass2)
  y2star<-y2+flip2
  ystar[,b]<-c(y1star,y2star)
}

#FITTING MODELS FOR A RANGE OF BETA

betaSeq<-c()
for (betaK in 1:19) {
  betaSeq[betaK]<-betaK/20
}

#definitions

#training set f-measure and noisy f-measure for standard maxent and
#f-measure maximizing maxent algorithm
fDetBaseTrain<-c()
fDetTrain<-c()
noisyfDetBaseTrain<-c()
noisyfDetTrain<-c()

#training set f-measure and noisy f-measure for uncertain maxent and noisy
#f-measure maximizing uncertain maxent algorithm
fUncBaseTrain<-c()
fUncTrain<-c()
noisyfUncBaseTrain<-c()
noisyfUncTrain<-c()

#resampled f-measure for standard maxent, f-measure maximizing maxent,
#uncertain maxent and noisy f-measure maximizing uncertain maxent algorithm
fDetBaseResample<-c()
fDetResample<-c()
fUncBaseResample<-c()
fUncResample<-c()

#test set noisy f-measure for uncertain maxent and noisy f-measure maximizing
#uncertain maxent algorithm
noisyfUncBaseTest<-c()
noisyfUncTest<-c()

for (betaK in 1:19) {
```

```
#beta parameter in f-measure
beta<-betaSeq[betaK]

#logistic regression for deterministic max likelihood
#with f-measure maximizing algorithm

fmeasl<-c()
noisyfL<-c()
its<-100

for (k in 1:its) {

  if (k==1) {
    #initialize with standard (non-weighted) logistic regression
    w<-rep(1,m)
  } else {
    #update weights according to f-measure maximizing algorithm
    w<-pmax(0.01, pmin(10,((1-beta*fmeasl[k-1])
                           /(beta*fmeasl[k-1]))*lvals*(ydet==1)
                       + lvals*(ydet==0)))
  }

  glm.det<-glm(ydet~x,family=binomial(logit),weights=w)

  #success probabilities
  lvalspos<-glm.det$fitted.values

  #probability of point x[i,] belonging to class ydet[i]
  lvals<-c()
  for (i in 1:m) {
    if (ydet[i]==1) {
      lvals[i]<-lvalspos[i]
    } else {
      lvals[i]<-1-lvalspos[i]
    }
  }

  #fitted model
  lfmaxfit<-(lvalspos>0.5)*1

  #save first iteration (this is the non-weighted logistic regression fit)
  if (k==1) {
    lfit<-lfmaxfit
  }

  #f-measure at k'th iteration
```

```
  fmeasl[k]<-fBetaCalc(lfmaxfit,ydet,beta)

  #noisy f-measure at k'th iteration
  noisytp<-sum(pmin(q[,1],lvalspos))
  noisyfL[k]<-noisytp/(beta*sum(lvalspos)+(1-beta)*sum(q[,1]))

  #break if no significant improvement of f-measure
  if (k>1) {
    if (abs(fmeasl[k]-fmeasl[k-1])<1e-8) {
      break
    }
  }

}

fDetBaseTrain[betaK]<-fmeasl[1]
fDetTrain[betaK]<-fmeasl[k]
noisyfDetBaseTrain[betaK]<-noisyfL[1]
noisyfDetTrain[betaK]<-noisyfL[k]

#logistic regression for uncertain max likelihood with noisy f-measure
#maximizing algorithm

#create 2 copies of data x
xunc<-rbind(x,x)
#create 2 copies of classes y
yflip<-1-y
yunc<-c(y,yflip)
#prior probability weights for input into glm
qweights<-c(q[,1][1:mplus],q[,2][mplus+1:mminus],q[,2][1:mplus],
            q[,1][mplus+1:mminus])

fmeaslu<-c()
noisyfLU<-c()
its<-100

for (k in 1:its) {

  if (k==1) {
    #initialize with standard (non-weighted) logistic regression
    w<-rep(1,m)
  } else {
    #update weights according to f-measure maximizing algorithm
    w<-pmax(1e-6, pmin(10,(luvals[,1]*luvals[,2]/(luvals[,1]-q[,1]))
                       *(noisyfLU[k-1]/noisytp)*(beta*noisyfLU[k-1]
                                                 *(luvals[,1]>=q[,1])
                                                 - (1-beta*noisyfLU[k-1])
```

```
                                                    *(luvals[,1]<=q[,1])))

    #deterministic plug-in weights
    #w<-pmax(1e-6, pmin(10,(luvals[,1]*luvals[,2]/(luvals[,1]-q[,1]))
    #                   *(fmeaslu[k-1]/tp)*(beta*fmeaslu[k-1]
    #                                       *(luvals[,1]>=q[,1])
    #                                       - (1-beta*fmeaslu[k-1])
    #                                       *(luvals[,1]<=q[,1]))))
  }

  x1<-xunc[,1]
  x2<-xunc[,2]
  uncFrame<-data.frame(x1=x1,x2=x2,yunc=yunc)

  glm.unc<-glm(yunc~x1+x2,data=uncFrame,family=binomial(logit),
               weights=rep(w,2)*qweights)

  #matrix of probabilities of point x[i,] belonging to class 1 or 0
  luvals<-matrix(c(glm.unc$fitted.values[1:m],
                   1-glm.unc$fitted.values[1:m]),ncol=2)
  #fitted model
  lufmaxfit<-(luvals[,1]>0.5)*1

  #save first iteration (this is the non-weighted logistic regression fit)
  if (k==1) {
    glm.unc0<-glm.unc
    lufit<-lufmaxfit
  }

  #f-measure at k'th iteration
  tp<-sum(lufmaxfit*ydet)
  fmeaslu[k]<-fBetaCalc(lufmaxfit,ydet,beta)

  #noisy f-measure at k'th iteration
  noisytp<-sum(pmin(q[,1],luvals[,1]))
  noisyfLU[k]<-noisytp/(beta*sum(luvals[,1]) + (1-beta)*sum(q[,1]))

  #break if no significant improvement of noisy f-measure
  if (k>1) {
    if (abs(noisyfLU[k]-noisyfLU[k-1])<1e-8) {
      break
    }
  }

}

fUncBaseTrain[betaK]<-fmeaslu[1]
```

```
fUncTrain[betaK]<-fmeaslu[k]
noisyfUncBaseTrain[betaK]<-noisyfLU[1]
noisyfUncTrain[betaK]<-noisyfLU[k]

#performance on test set

#noisy f-measure maximizing algorithm
LUfitTest<-predict(glm.unc,newdata=data.frame(x1=xtest[,1],x2=xtest[,2]),
                   type="response")

#noisy f-measure
noisytp<-sum(pmin(qtest[,1],LUfitTest))
noisyfUncTest[betaK]<-noisytp/(beta*sum(LUfitTest)
                               + (1-beta)*sum(qtest[,1]))

#baseline max uncertain likelihood
LUfitBaseTest<-predict(glm.unc0,
                       newdata=data.frame(x1=xtest[,1],x2=xtest[,2]),
                       type="response")

#noisy f-measure
noisytp<-sum(pmin(qtest[,1],LUfitBaseTest))
noisyfUncBaseTest[betaK]<-noisytp/(beta*sum(LUfitBaseTest)
                                   + (1-beta)*sum(qtest[,1]))

#performance on resamples

fdetstar<-c()
fdetfmaxstar<-c()
funcstar<-c()
funcfmaxstar<-c()

for (b in 1:B) {

  #f-measure for standard maxent
  fdetstar[b]<-fBetaCalc(lfit,ystar[,b],beta)

  #f-measure for f-measure maximizing maxent
  fdetfmaxstar[b]<-fBetaCalc(lfmaxfit,ystar[,b],beta)

  #f-measure for uncertain maxent
  funcstar[b]<-fBetaCalc(lufit,ystar[,b],beta)

  #f-measure for noisy f-measure maximizing uncertain maxent
  funcfmaxstar[b]<-fBetaCalc(lufmaxfit,ystar[,b],beta)

}
```

```
  fDetBaseResample[betaK] <- mean(fdetstar)
  fDetResample[betaK] <- mean(fdetfmaxstar)
  fUncBaseResample[betaK] <- mean(funcstar)
  fUncResample[betaK] <- mean(funcfmaxstar)
}

#PLOTS

#plots for training data

par(mfrow=c(1,2))

fLimLower<-min(c(fDetBaseTrain[2:18],fDetTrain[2:18],fUncBaseTrain[2:18],
                 fUncTrain[2:18]))

plot(betaSeq[2:18],fDetBaseTrain[2:18],type="l",lty=2,col="blue",
     xlab=expression(beta),ylab=expression(F[beta]),ylim=c(fLimLower,1))
lines(betaSeq[2:18],fDetTrain[2:18],col="blue")
lines(betaSeq[2:18],fUncBaseTrain[2:18],col="red",lty=2)
lines(betaSeq[2:18],fUncTrain[2:18],col="red")
legend(0.4,1,c("standard maxent",expression(F[beta]~maximizer),
                  "uncertain maxent",expression(noisy~F[beta]~maximizer)),
       lty=c(2,1,2,1),col=c("blue","blue","red","red"),cex=0.5)

noisyfLimLower<-min(c(noisyfDetBaseTrain,noisyfDetTrain,noisyfUncBaseTrain,
                      noisyfUncTrain))

plot(betaSeq,noisyfDetBaseTrain,type="l",lty=2,col="blue",
     xlab=expression(beta),ylab=expression(Noisy~tilde(F)[beta]),
     ylim=c(noisyfLimLower,1))
lines(betaSeq,noisyfDetTrain,col="blue")
lines(betaSeq,noisyfUncBaseTrain,col="red",lty=2)
lines(betaSeq,noisyfUncTrain,col="red")
legend(0.4,1,c("standard maxent",expression(F[beta]~maximizer),
                   "uncertain maxent",expression(noisy~F[beta]~maximizer)),
       lty=c(2,1,2,1),col=c("blue","blue","red","red"),cex=0.5)

#plots for test set

par(mfrow=c(1,1))

noisyfTestLimLower<-min(c(noisyfUncBaseTest[2:18],noisyfUncTest[2:18]))

plot(betaSeq[2:18],noisyfUncBaseTest[2:18],type="l",lty=2,col="red",
     xlab=expression(beta),ylab=expression(Noisy~F[beta]),
     ylim=c(noisyfTestLimLower,1))
```

```
lines(betaSeq[2:18],noisyfUncTest[2:18],col="red")
legend(0.7,1,c("uncertain maxent",expression(noisy~F[beta]~maximizer)),
       lty=c(2,1),col=c("red","red"),cex=0.5)

#plots for resamples

par(mfrow=c(1,1))

fResampleLimLower<-min(c(fDetBaseResample[2:18],fDetResample[2:18],
                         fUncBaseResample[2:18],fUncResample[2:18]))

plot(betaSeq[2:18],fDetBaseResample[2:18],type="l",lty=2,col="blue",
     xlab=expression(beta),ylab=expression(F[beta]),
     ylim=c(fResampleLimLower,1))
lines(betaSeq[2:18],fDetResample[2:18],col="blue")
lines(betaSeq[2:18],fUncBaseResample[2:18],col="red",lty=2)
lines(betaSeq[2:18],fUncResample[2:18],col="red")
legend(0.7,1,c("standard maxent",expression(F[beta]~maximizer),
                  "uncertain maxent",expression(noisy~F[beta]~maximizer)),
       lty=c(2,1,2,1),col=c("blue","blue","red","red"),cex=0.5)

#plot of typical learning curve

plot(noisyfLU[1:37],xlab="Iteration",ylab=expression(Noisy~F[beta]),type="l")
```

References

[1] Bazaraa, M.S., Sherali, H.D. and Shetty, C.M. (2006), *Nonlinear Programming: Theory and Algorithms*, 3rd Ed., John Wiley and Sons, New Jersey.

[2] Ben-Israel, A. and Mond, B. (1986), What is invexity? *J. Austral. Math. Soc. Ser. B* **28**, 1–9.

[3] Burges, C.J.C. (1998), A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery*, **2**(2), 121–167.

[4] Cohn, T. and Specia, L. (2013), Modelling annotator bias with multi-task Gaussian processes: an application to machine translation quality estimation, *Association for Computational Linguistics Conference*.

[5] Debreu, G. and Koopmans, T.C. (1982), Additively decomposed quasiconvex functions, *Mathematical Programming*, **24**(1), 1–38.

[6] Dempster, A.P., Laird, N.M. and Rubin, D.B. (1977), Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society. Series B (Methodological)*, **39**(1), 1–38.

[7] Denoeux, T. (2013), Maximum likelihood estimation from uncertain data in the belief function framework, *IEEE Transactions on Knowledge and Data Engineering* **25**(1), 119–130.

[8] Dimitroff, G., Tolosi, L., Popov, B. and Georgiev, G. (2013), Weighted maximum likelihood as a convenient shortcut to optimize the F-measure of maximum entropy classifiers, *Proceedings of the International Conference RANLP*.

[9] Duda, R.O., Hart, P.E. and Stork, D.G. (1997), *Pattern Classification*, 2nd Ed., John Wiley and Sons, Chichester.

[10] Ehrgott, M. (2005), *Multicriteria Optimization*, Springer, New Jersey.

[11] Forman, G. and Scholz, M. (2009), Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement, *ACM SIGKDD Explorations*, **12**(1): 49–57.

[12] Gale, D., Kuhn, H.W. and Tucker, A.W. (1951), Linear programming and the theory of games, in T.C. Koopmans (ed.), *Activity Analysis of Production and Allocation*, John Wiley and Sons, New York.

[13] Hardle, W.K. and Simar, L. (2012), *Applied Multivariate Statistical Analysis*, 3rd Ed., Springer-Verlag, Berlin.

[14] Jansche, M. (2005), Maximum expected F-measure training of logistic regression models, *HLT*, 692–699.

[15] Lin, C.-F. and Wang, S.-D. (2002), Fuzzy support vector machines, *IEEE Transactions on Neural Networks*, **13**(2), 464–471.

[16] Mann, G.S. and McCallum, A. (2010), Generalized expectation criteria for semi-supervised learning with weakly labeled data, *Journal of Machine Learning Research* **11**, 955–984.

[17] Mercer, J. (1909), Functions of positive and negative type and their connection with the theory of integral equations, *Philosophical Transactions of the Royal Society of London* **209**, 415–446.

[18] Mishra, S.K. and Giorgi, G. (2008), *Invexity and Optimization*, Springer, New York.

[19] Mount, J. (2011), The equivalence of logistic regression and maximum entropy models, URL: `http://www.win-vector.com/dfiles/LogisticRegressionMaxEnt.pdf`

[20] Musicant, D.R., Kumar, V. and Ozgur, A. (2003), Optimizing F-measure with support vector machines, *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, 356–360.

[21] Neal, R.M. and Hinton, G.E. (1999), A view of the EM algorithm that justifies incremental, sparse and other variants, *Learning in Graphical Models*.

[22] Powell, M.J.D. (1994), A direct search optimization method that models the objective and constraint functions by linear interpolation, *Advances and Optimization and Numerical Analysis*, 51–67.

[23] Rose, T., Dimitroff, G., Tolosi, L., Popov, B. and Georgiev, G. (2014), Likelihood and F-measure maximization under uncertainty, Working paper, University of New South Wales.

[24] *Vanderbilt Biostatistics Datasets*, (cited Jun 2014), URL: `http://biostat.mc.vanderbilt.edu/wiki/Main/DataSets`

[25] van Rijsbergen, C. (1979), *Information Retrieval*, Butterworths, London.

[26] Vapnik, V. and Cortes, C. (1995), Support-vector networks, *Machine Learning*, **20**, 273–297.

[27] Wang, X., Shao, H., Matwin, S., Liu, X., Japkowicz, N., Bourque, A. and Nguyen, B. (2012), Using SVM with adaptively asymmetric misclassification costs for mine-like objects detection, *11th International Conference on Machine Learning and Applications*.

[28] Zhu, X. and Davidson, I. (2007), *Knowledge Discovery and Data Mining*, IGI Global, London.