A Maximum Expected Utility Framework for Binary Sequence Labeling

Martin Jansche*
jansche@acm.org

Abstract

We consider the problem of predictive inference for probabilistic binary sequence labeling models under F-score as utility. For a simple class of models, we show that the number of hypotheses whose expected F-score needs to be evaluated is linear in the sequence length and present a framework for efficiently evaluating the expectation of many common loss/utility functions, including the F-score. This framework includes both exact and faster inexact calculation methods.

1 Introduction

1.1 Motivation and Scope

The weighted F-score (van Rijsbergen, 1974) plays an important role in the evaluation of binary classifiers, as it neatly summarizes a classifier's ability to identify the positive class. A variety of methods exists for training classifiers that optimize the F-score, or some similar trade-off between false positives and false negatives, precision and recall, sensitivity and specificity, type I error and type II error rate, etc. Among the most general methods are those of Mozer et al. (2001), whose constrained optimization technique is similar to those in (Gao et al., 2006; Jansche, 2005). More specialized methods also exist, for example for support vector machines (Musicant et al., 2003) and for conditional random fields (Gross et al., 2007; Suzuki et al., 2006).

All of these methods are about classifier training. In this paper we focus primarily on the related, but orthogonal, issue of predictive inference with a fully trained probabilistic classifier. Using the weighted F-score as our utility function, predictive inference amounts to choosing an optimal hypothesis which maximizes the expected utility. We refer to this as the prediction or decoding task. In general, decoding can be a hard computational problem (Casacuberta and de la Higuera, 2000; Knight, 1999). In this paper we show that the maximum expected F-score decoding problem can be solved in polynomial time under certain assumptions about the underlying probability model. One key ingredient in our solution is a very general framework for evaluating the expected F-score, and indeed many other utility functions, of a fixed hypothesis.[1] This framework can also be applied to discriminative classifier training.

1.2 Background and Notation

We formulate our approach in terms of sequence labeling, although it has applications beyond that. This is motivated by the fact that our framework for evaluating expected utility is indeed applicable to general sequence labeling tasks, while our decoding method is more restricted. Another reason is that the F-score is only meaningful for comparing two (multi)sets or two binary sequences, but the notation for multisets is slightly more awkward.

All tasks considered here involve strings of binary labels. We write the length of a given string $y \in \{0,1\}^n$ as $|y| = n$. It is convenient to view such strings as real vectors – whose components happen to be 0 or 1 – with the dot product defined as usual. Then $y \cdot y$ is the number of ones that occur in the string y. For two strings x, y of the same length $|x| = |y|$ the number of ones that occur at corresponding indices is $x \cdot y$.

Given a hypothesis z and a gold standard label sequence y, we define the following quantities:

1. $T = y \cdot y$, the genuine positives;
2. $P = z \cdot z$, the predicted positives;
3. $A = z \cdot y$, the true positives (predicted positives that are genuinely positive);

*Current affiliation: Google Inc. Former affiliation: Center of Computational Learning Systems, Columbia University.

[1] A proof-of-concept implementation is available at `http://purl.org/net/jansche/meu_framework/`.

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 736–743,
Prague, Czech Republic, June 2007.

4. $Recl = A/T$, recall (a.k.a. sensitivity or power);
5. $Prec = A/P$, precision.

The β-weighted F-score is then defined as the weighted harmonic mean of recall and precision. This simplifies to

$$F_\beta = \frac{(\beta+1)A}{P+\beta T} \qquad (\beta > 0) \tag{1}$$

where we assume for convenience that $0/0 \overset{\text{def}}{=} 1$ to avoid explicitly dealing with the special case of the denominator being zero. We will write the weighted F-score from now on as $F(z,y)$ to emphasize that it is a function of z and y.

1.3 Expected *F*-Score

In Section 3 we will develop a method for evaluating the expectation of the F-score, which can also be used as a smooth approximation of the raw F-score during classifier training: in that task (which we will not discuss further in this paper), z are the supervised labels, y is the classifier output, and the challenge is that $F(z,y)$ does not depend smoothly on the parameters of the classifier. Gradient-based optimization techniques are not applicable unless some of the quantities defined above are replaced by approximations that depend smoothly on the classifier's parameters. For example, the constrained optimization method of (Mozer et al., 2001) relies on approximations of sensitivity (which they call *CA*) and specificity[2] (their *CR*); related techniques (Gao et al., 2006; Jansche, 2005) rely on approximations of true positives, false positives, and false negatives, and, indirectly, recall and precision. Unlike these methods we compute the expected F-score exactly, without relying on *ad hoc* approximations of the true positives, etc.

Being able to efficiently compute the expected F-score is a prerequisite for maximizing it during decoding. More precisely, we compute the expectation of the function

$$y \mapsto F(z,y), \tag{2}$$

which is a unary function obtained by holding the first argument of the binary function F fixed. It will henceforth be abbreviated as $F(z,\cdot)$, and we will denote its expected value by

$$\mathrm{E}\left[F(z,\cdot)\right] = \sum_{y \in \{0,1\}^{|z|}} F(z,y)\ \Pr(y). \tag{3}$$

[2]Defined as $[(\vec{1}-z)\cdot(\vec{1}-y)]/[(\vec{1}-y)\cdot(\vec{1}-y)]$.

This expectation is taken with respect to a probability model over binary label sequences, written as $\Pr(y)$ for simplicity. This probability model may be conditional, that is, in general it will depend on covariates x and parameters θ. We have suppressed both in our notation, since x is fixed during training and decoding, and we assume that the model is fully identified during decoding. This is for clarity only and does not limit the class of models, though we will introduce additional, limiting assumptions shortly. We are now ready to tackle the inference task formally.

2 Maximum Expected *F*-Score Inference

2.1 Problem Statement

Optimal predictive inference under F-score utility requires us to find an hypothesis $\hat{z}$ of length n which maximizes the expected F-score relative to a given probabilistic sequence labeling model:

$$\hat{z} = \operatorname*{argmax}_{z \in \{0,1\}^n} \mathrm{E}\left[F(z,\cdot)\right] = \operatorname*{argmax}_{z \in \{0,1\}^n} \sum_y F(z,y)\ \Pr(y). \tag{4}$$

We require the probability model to factor into independent Bernoulli components (Markov order zero):

$$\Pr(y = (y_1,\ldots,y_n)) = \prod_{i=1}^{n} p_i^{y_i}\,(1-p_i)^{1-y_i}. \tag{5}$$

In practical applications we might choose the overall probability distribution to be the product of independent logistic regression models, for example. Ordinary classification arises as a special case when the y_i are i.i.d., that is, a single probabilistic classifier is used to find $\Pr(y_i = 1 \mid x_i)$. For our present purposes it is sufficient to assume that the inference algorithm takes as its input the vector $(p_1,\ldots,p_n)$, where p_i is the probability that $y_i = 1$.

The discrete maximization problem (4) cannot be solved naively, since the number of hypotheses that would need to be evaluated in a brute-force search for an optimal hypothesis $\hat{z}$ is exponential in the sequence length n. We show below that in fact only a few hypotheses ($n+1$ instead of 2^n) need to be examined in order to find an optimal one.

The inference algorithm is the intuitive one, analogous to the following simple observation: Start with the hypothesis $z = 00\ldots0$ and evaluate its raw F-score $F(z,y)$ relative to a fixed but unknown binary string y. Then z will have perfect precision (no positive labels means no chance to make mistakes), and zero recall (unless $y = z$). Switch on any bit of z that is currently off. Then precision will decrease or remain equal, while recall will increase or remain equal. Repeat until $z = 11\ldots1$ is reached, in which case recall will be perfect and precision at its minimum. The inference algorithm for expected F-score follows the same strategy, and in particular it switches on the bits of z in order of non-increasing probability: start with $00\ldots0$, then switch on the bit $i_1 = \mathrm{argmax}_i\, p_i$, etc. until $11\ldots1$ is reached. We now show that this intuitive strategy is indeed admissible.

2.2 Outer and Inner Maximization

In general, maximization can be carried out piecewise, since

$$\operatorname*{argmax}_{x \in X} f(x) = \operatorname*{argmax}_{x \in \{\mathrm{argmax}_{y \in Y} f(y) \mid Y \in \pi(X)\}} f(x),$$

where $\pi(X)$ is any family $(Y_1, Y_2, \ldots)$ of nonempty subsets of X whose union $\bigcup_i Y_i$ is equal to X. (Recursive application would lead to a divide-and-conquer algorithm.) Duplication of effort is avoided if $\pi(X)$ is a partition of X.

Here we partition the set $\{0,1\}^n$ into equivalence classes based on the number of ones in a string (viewed as a real vector). Define S_m to be the set

$$S_m = \{s \in \{0,1\}^n \mid s \cdot s = m\}$$

consisting of all binary strings of fixed length n that contain exactly m ones. Then the maximization problem (4) can be transformed into an inner maximization

$$\hat{s}^{(m)} = \operatorname*{argmax}_{s \in S_m} \mathrm{E}\left[F(s, \cdot)\right], \tag{6}$$

followed by an outer maximization

$$\hat{z} = \operatorname*{argmax}_{z \in \{\hat{s}^{(0)}, \ldots, \hat{s}^{(n)}\}} \mathrm{E}\left[F(z, \cdot)\right]. \tag{7}$$

2.3 Closed-Form Inner Maximization

The key insight is that the inner maximization problem (6) can be solved analytically. Given a vector $p = (p_1, \ldots, p_n)$ of probabilities, define $z^{(m)}$ to be the binary label sequence with exactly m ones and $n - m$ zeroes where for all indices i, k we have

$$\left[z_i^{(m)} = 1 \wedge z_k^{(m)} = 0\right] \rightarrow p_i \geq p_k.$$

Algorithm 1 Maximizing the Expected F-Score.

```
1: Input: probabilities p = (p_1, ..., p_n)
2: I ← indices of p sorted by non-increasing probability
3: z ← 0...0
4: a ← 0
5: v ← expectF(z, p)
6: for j ← 1 to n do
7:     i ← I[j]
8:     z[i] ← 1        // switch on the ith bit
9:     u ← expectF(z, p)
10:    if u > v then
11:        a ← j
12:        v ← u
13: for j ← a + 1 to n do
14:    z[I[j]] ← 0
15: return (z, v)
```

In other words, the most probable m bits (according to p) in $z^{(m)}$ are set and the least probable $n - m$ bits are off. We rely on the following result, whose proof is deferred to Appendix A:

Theorem 1. $(\forall s \in S_m)\ \mathrm{E}\left[F(z^{(m)}, \cdot)\right] \geq \mathrm{E}\left[F(s, \cdot)\right]$.

Because $z^{(m)}$ is maximal in S_m, we may equate $z^{(m)} = \mathrm{argmax}_{s \in S_m} \mathrm{E}\left[F(s, \cdot)\right] = \hat{s}^{(m)}$ (modulo ties, which can always arise with argmax).

2.4 Pedestrian Outer Maximization

With the inner maximization (6) thus solved, the outer maximization (7) can be carried out naively, since only $n + 1$ hypotheses need to be evaluated. This is precisely what Algorithm 1 does, which keeps track of the maximum value in v. On termination $z = \mathrm{argmax}_s \mathrm{E}\left[F(s, \cdot)\right]$. Correctness follows directly from our results in this section.

Algorithm 1 runs in time $\mathrm{O}(n \log n + n\, f(n))$. A total of $\mathrm{O}(n \log n)$ time is required for accessing the vector p in sorted order (line 2). This dominates the $\mathrm{O}(n)$ time required to explicitly generate the optimal hypothesis (lines 13–14). The algorithm invokes a subroutine *expectF*(z, p) a total of $n + 1$ times. This subroutine, which is the topic of the next section, evaluates, in time $f(n)$, the expected F-score (with respect to p) of a given hypothesis z of length n.

3 Computing the Expected F-Score

3.1 Problem Statement

We now turn to the problem of computing the expected value (3) of the F-score for a given hypothesis z relative to a fully identified probability model. The method presented here does not strictly require the zeroth-order Markov assumption (5) instated earlier (a higher-order Markov assumption will suffice), but it shall remain in effect for simplicity.

As with the maximization problem (4), the sum in (3) is over exponentially many terms and cannot be computed naively. But observe that the F-score (1) is a (rational) function of integer counts which are bounded, so it can take on only a finite, and indeed small, number of distinct values. We shall see shortly that the function (2) whose expectation we wish to compute has a domain whose cardinality is exponential in n, but the cardinality of its range is polynomial in n. The latter is sufficient to ensure that its expectation can be computed in polynomial time. The method we are about to develop is in fact very general and applies to many other loss and utility functions besides the F-score.

3.2 Expected F-Score as an Integral

A few notions from real analysis are helpful because they highlight the importance of thinking about functions in terms of their range, level sets, and the equivalence classes they induce on their domain (the kernel of the function). A function $g : \Omega \to \mathbb{R}$ is said to be *simple* if it can be expressed as a linear combination of indicator functions (characteristic functions):

$$g(x) = \sum_{k \in K} a_k \, \chi_{B_k}(x),$$

where K is a finite index set, $a_k \in \mathbb{R}$, and $B_k \subseteq \Omega$. ($\chi_S : S \to \{0,1\}$ is the characteristic function of set S.)

Let Ω be a countable set and $\mathscr{P}$ be a probability measure on Ω. Then the expectation of g is given by the Lebesgue integral of g. In the case of a simple function g as defined above, the integral, and hence the expectation, is defined as

$$\mathrm{E}[g] = \int_{\Omega} g \, \mathrm{d}\mathscr{P} = \sum_{k \in K} a_k \, \mathscr{P}(B_k). \qquad (8)$$

This gives us a general recipe for evaluating $\mathrm{E}[g]$ when Ω is much larger than the range of g. Instead of computing the sum $\sum_{y \in \Omega} g(y) \, \mathscr{P}(\{y\})$ we can compute the sum in (8) above. This directly yields an efficient algorithm whenever K is sufficiently small and $\mathscr{P}(B_k)$ can be evaluated efficiently.

The expected F-score is thus the Lebesgue integral of the function (2). Looking at the definition of the F-score in (1) we see that the only expressions which depend on y are $A = z \cdot y$ and $T = y \cdot y$ ($P = z \cdot z$ is fixed because z is). But $0 \le z \cdot y \le y \cdot y \le n = |z|$. Therefore $F(z, \cdot)$ takes on at most $(n+1)(n+2)/2$, i.e. quadratically many, distinct values. It is a simple function with

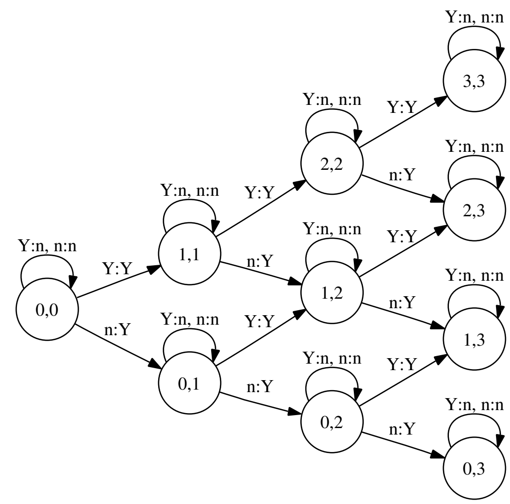

Figure 1: Finite State Classifier h'.

$$K = \{(A,T) \in \mathbb{N}_0 \times \mathbb{N}_0 \mid A \le T \le |z|, \, A \le z \cdot z\}$$

$$a_{(A,T)} = \frac{(\beta+1)A}{z \cdot z + \beta\, T} \qquad \text{where } 0/0 \stackrel{\text{def}}{=} 1$$

$$B_{(A,T)} = \{y \mid z \cdot y = A, \; y \cdot y = T\}.$$

3.3 Computing Membership in B_k

Observe that the family of sets $\left(B_{(A,T)}\right)_{(A,T) \in K}$ is a partition (namely the kernel of $F(z, \cdot)$) of the set $\Omega = \{0,1\}^n$ of all label sequences of length n. In turn it gives rise to a function $h : \Omega \to K$ where $h(y) = k$ iff $y \in B_k$. The function h can be computed by a deterministic finite automaton, viewed as a sequence classifier: rather than assigning binary accept/reject labels, it assigns arbitrary labels from a finite set, in this case the index set K. For simplicity we show the initial portion of a slightly more general two-tape automaton h' in Figure 1. It reads the two sequences z and y on its two input tapes and counts the number of matching positive labels (represented as Y) as well as the number of positive labels on the second tape. Its behavior is therefore $h'(z, y) = (z \cdot y, \; y \cdot y)$. The function h is obtained as a special case when z (the first tape) is fixed.

Note that this only applies to the special case when

Algorithm 2 Simple Function Instance for F-Score.

def *start*():
  return $(0,0)$

def *transition*(k,z,i,y_i):
  $(A,T) \leftarrow k$
  if $y_i = 1$ **then**
    $T \leftarrow T+1$
    if $z[i] = 1$ **then**
      $A \leftarrow A+1$
  return (A,T)

def $a(k,z)$:
  $(A,T) \leftarrow k$
  $F \leftarrow \dfrac{(\beta+1)A}{z \cdot z + \beta T}$ // where $0/0 \overset{\text{def}}{=} 1$
  return F

Algorithm 3 Value of a Simple Function.

Input: instance g of the simple function interface, strings z and y of length n
$k \leftarrow g.start()$
for $i \leftarrow 1$ **to** n **do**
  $k \leftarrow g.transition(k,z,i,y[i])$
return $g.a(k,z)$

the family $B = (B_k)_{k \in K}$ is a partition of Ω. It is always possible to express any simple function in this way, but in general there may be an exponential increase in the size of K when the family B is required to be a partition. However for the special cases we consider here this problem does not arise.

3.4 The Simple Function Trick

In general, what we will call the *simple function trick* amounts to representing the simple function g whose expectation we want to compute by:

1. a finite index set K (perhaps implicit),
2. a deterministic finite state classifier $h : \Omega \to K$,
3. and a vector of coefficients $(a_k)_{k \in K}$.

In practice, this means instantiating an interface with three methods: the start and transition function of the transducer which computes h' (and from which h can be derived), and an accessor method for the coefficients a. Algorithm 2 shows the F-score instance.

Any simple function g expressed as an instance of this interface can then be evaluated very simply as $g(x) = a_{h(x)}$. This is shown in Algorithm 3.

Evaluating $\mathrm{E}[g]$ is also straightforward: Compose the DFA h with the probability model p and use an algebraic path algorithm to compute the total probability mass $\mathscr{P}(B_k)$ for each final state k of the resulting automaton. If p factors into independent components as required by (5), the composition is greatly simplified. If p incorporates label history (higher-order Markov assumption), nothing changes in principle, though the following algorithm assumes for simplicity that the stronger assumption is in effect.

Algorithm 4 Expectation of a Simple Function.

1: **Input:** instance g of the simple function interface, string z and probability vector p of length n
2: $M \leftarrow Map()$
3: $M[g.start()] \leftarrow 1$
4: **for** $i \leftarrow 1$ **to** n **do**
5:   $N \leftarrow Map()$
6:   **for** $(k,P) \in M$ **do**
7:     // transition on $y_i = 0$
8:     $k_0 \leftarrow g.transition(k,z,i,0)$
9:     **if** $k_0 \notin N$ **then**
10:       $N[k_0] \leftarrow 0$
11:     $N[k_0] \leftarrow N[k_0] + P \times (1-p[i])$
12:     // transition on $y_i = 1$
13:     $k_1 \leftarrow g.transition(k,z,i,1)$
14:     **if** $k_1 \notin N$ **then**
15:       $N[k_1] \leftarrow 0$
16:     $N[k_1] \leftarrow N[k_1] + P \times p[i]$
17:   $M \leftarrow N$
18: $E \leftarrow 0$
19: **for** $(k,P) \in M$ **do**
20:   $E \leftarrow E + g.a(k,z) \times P$
21: **return** E

Algorithm 4 expands the following composed automaton, represented implicitly: the finite-state transducer h' specified as part of the simple function object g is composed on the left with the string z (yielding h) and on the right with the probability model p. The outer loop variable i is an index into z and hence a state in the automaton that accepts z; the variable k keeps track of the states of the automaton implemented by g; and the probability model has a single state by assumption, which does not need to be represented explicitly. Exploring the states in order of increasing i puts them in topological order, which means that the algebraic path problem can be solved in time linear in the size of the composed automaton. The maps M and N keep track of the algebraic distance from the start state to each intermediate state. On termination of the first outer loop (lines 4–17), the map M contains the final states together with their distances. The algebraic distance of a final state k is now equal to $\mathscr{P}(B_k)$, so the expected value E can be computed in the second loop (lines 18–20) as suggested by (8).

When the utility function interface g is instantiated as in Algorithm 2 to represent the F-score, the runtime of Algorithm 4 is cubic in n, with very small constants.[3] The first main loop iterates over n. The inner loop iterates over the states expanded at iteration i, of which there are $\mathrm{O}(i^2)$ many when dealing with the F-score. The second main loop iterates over the final states, whose number is quadratic in n in this case. The overall cubic runtime of the first loop dominates the computation.

3.5 Other Utility Functions

With other functions g the runtime of Algorithm 4 will depend on the asymptotic size of the index set K. If there are asymptotically as many intermediate states at any point as there are final states, then the general asymptotic runtime is $\mathrm{O}(n\,|K|)$.

Many loss/utility functions are subsumed by the present framework. Zero–one loss is trivial: the automaton has two states (success, failure); it starts and remains in the success state as long as the symbols read on both tapes match; on the first mismatch it transitions to, and remains in, the failure state.

Hamming (1950) distance is similar to zero–one loss, but counts the number of mismatches (bounded by n), whereas zero–one loss only counts up to a threshold of one.

A more interesting case is given by the P_k-score (Beeferman et al., 1999) and its generalizations, which moves a sliding window of size k over a pair of label sequences (z,y) and counts the number of windows which contain a segment boundary on one of the sequences but not the other. To compute its expectation in our framework, all we have to do is express the sliding window mechanism as an automaton, which can be done very naturally (see the proof-of-concept implementation for further details).

4 Faster Inexact Computations

Because the exact computation of the expected F-score by Algorithm 4 requires cubic time, the overall runtime of Algorithm 1 (the decoder) is quartic.[4]

Faster decoding can be achieved by modifying Algorithm 4 to compute an approximation (in fact, a lower bound) of the expected F-score.[5] This is done by introducing an additional parameter L which limits the number of intermediate states that get expanded. Instead of iterating over all states and their associated probabilities (inner loop starting at line 6), one iterates over the top L states only. We require that $L \geq 1$ for this to be meaningful. Before entering the inner loop the entries of the map M are expanded and, using the linear time selection algorithm, the top L entries are selected. Because each state that gets expanded in the inner loop has out-degree 2, the new state map N will contain at most $2L$ states. This means that we have an additional loop invariant: the size of M is always less than or equal to $2L$. Therefore the selection algorithm runs in time $\mathrm{O}(L)$, and so does the abridged inner loop, as well as the second outer loop. The overall runtime of this modified algorithm is therefore $\mathrm{O}(nL)$.

If L is a constant function, the inexact computation of the expected F-score runs in linear time and the overall decoding algorithm in quadratic time. In particular if $L = 1$ the approximate expected F-score is equal to the F-score of the MAP hypothesis, and the modified inference algorithm reduces to a variant of Viterbi decoding. If L is a linear function of n, the overall decoding algorithm runs in cubic time.

We experimentally compared the exact quartic-time decoding algorithm with the approximate decoding algorithm for $L = 2n$ and for $L = 1$. We computed the absolute difference between the expected F-score of the optimal hypothesis (as found by the exact algorithm) and the expected F-score of the winning hypothesis found by the approximate decoding algorithm. For different sequence lengths $n \in \{1,\dots,50\}$ we performed 10 runs of the different decoding algorithms on randomly generated probability vectors p, where each p_i was randomly drawn from a continuous uniform distribution on $(0,1)$, or, in a second experiment, from a $\mathrm{Beta}(1/2,1/2)$ distribution (to simulate an over-trained classifier).

For $L = 1$ there is a substantial difference of about

[3] A tight upper bound on the total number of states of the composed automaton in the worst case is $\left\lfloor \frac{1}{12}n^3 + \frac{5}{8}n^2 + \frac{17}{12}n + 1 \right\rfloor$.

[4] It is possible to speed up the decoding algorithm in absolute terms, though not asymptotically, by exploiting the fact that it explores very similar hypotheses in sequence. Algorithm 4 can be modified to store and return all of its intermediate map data-structures. This modified algorithm then requires cubic space instead of quadratic space. This additional storage cost pays off when the algorithm is called a second time, with its formal parameter z bound to a string that differs from the one of the preceding run in just one position. This means that the map data-structures only need to be recomputed from that position forward. However, this does not lead to an asymptotically faster algorithm in the worst case.

[5] For error bounds, see the proof-of-concept implementation.

0.6 between the expected F-scores of the winning hypothesis computed by the exact algorithm and by the approximate algorithm. Nevertheless the approximate decoding algorithm found the optimal hypothesis more than 99% of the time. This is presumably due to the additional regularization inherent in the discrete maximization of the decoder proper: even though the computed expected F-scores may be far from their exact values, this does not necessarily affect the behavior of the decoder very much, since it only needs to find the maximum among a small number of such scores. The error introduced by the approximation would have to be large enough to disturb the order of the hypotheses examined by the decoder in such a way that the true maximum is reordered. This generally does not seem to happen.

For $L = 2n$ the computed approximate expected F-scores were indistinguishable from their exact values. Consequently the approximate decoder found the true maximum every time.

5 Conclusion and Related Work

We have presented efficient algorithms for maximum expected F-score decoding. Our exact algorithm runs in quartic time, but an approximate cubic-time variant is indistinguishable in practice. A quadratic-time approximation makes very few mistakes and remains practically useful.

We have further described a general framework for computing the expectations of certain loss/utility functions. Our method relies on the fact that many functions are sparse, in the sense of having a finite range that is much smaller than their codomain. To evaluate their expectations, we can use the simple function trick and concentrate on their level sets: it suffices to evaluate the probability of those sets/ events. The fact that the commonly used utility functions like the F-score have only polynomially many level sets is sufficient (but not necessary) to ensure that our method is efficient. Because the coefficients a_k can be arbitrary (in fact, they can be generalized to be elements of a vector space over the reals), we can deal with functions that go beyond simple counts.

Like the methods developed by Allauzen et al. (2003) and Cortes et al. (2003) our technique incorporates finite automata, but uses a direct threshold-counting technique, rather than a nondeterministic counting technique which relies on path multiplicities. This makes it easy to formulate the simultaneous counting of two distinct quantities, such as our A and T, and to reason about the resulting automata.

The method described here is similar in spirit to those of Gao et al. (2006) and Jansche (2005), who discuss maximum expected F-score training of decision trees and logistic regression models. However, the present work is considerably more general in two ways: (1) the expected utility computations presented here are not tied in any way to particular classifiers, but can be used with large classes of probabilistic models; and (2) our framework extends beyond the computation of F-scores, which fall out as a special case, to other loss and utility functions, including the P_k score. More importantly, expected F-score computation as presented here can be exact, if desired, whereas the cited works always use an approximation to the quantities we have called A and T.

Acknowledgements

Most of this research was conducted while I was affilated with the Center for Computational Learning Systems, Columbia University. I would like to thank my colleagues at Google, in particular Ryan McDonald, as well as two anonymous reviewers for valuable feedback.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*.

Doug Beeferman, Adam Berger, and John Lafferty. 1999. Statistical models for text segmentation. *Machine Learning*, 34(1–3):177–210.

Francisco Casacuberta and Colin de la Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. In *5th International Colloquium on Grammatical Inference*.

Corinna Cortes, Patrick Haffner, and Mehryar Mohri. 2003. Rational kernels. In *Advances in Neural Information Processing Systems*, volume 15.

Sheng Gao, Wen Wu, Chin-Hui Lee, and Tai-Seng Chua. 2006. A maximal figure-of-merit (MFoM)-learning approach to robust classifier design for text categorization. *ACM Transactions on Information Systems*, 24(2):190–218. Also in ICML 2004.

Samuel S. Gross, Olga Russakovsky, Chuong B. Do, and Serafim Batzoglou. 2007. Training conditional random fields for maximum labelwise accuracy. In *Advances in Neural Information Processing Systems*, volume 19.

R. W. Hamming. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal*, 26(2):147–160.

Martin Jansche. 2005. Maximum expected F-measure training of logistic regression models. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.

Michael C. Mozer, Robert Dodier, Michael D. Colagrosso, César Guerra-Salcedo, and Richard Wolniewicz. 2001. Prodding the ROC curve: Constrained optimization of classifier performance. In *Advances in Neural Information Processing Systems*, volume 14.

David R. Musicant, Vipin Kumar, and Aysel Ozgur. 2003. Optimizing F-measure with support vector machines. In *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference*.

Jun Suzuki, Erik McDermott, and Hideki Isozaki. 2006. Training conditional random fields with multivariate evaluation measures. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*.

C. J. van Rijsbergen. 1974. Foundation of evaluation. *Journal of Documentation*, 30(4):365–373.

Appendix A Proof of Theorem 1

The proof of Theorem 1 employs the following lemma:

Theorem 2. *For fixed n and p, let $s,t \in S_m$ for some m with $1 \leq m < n$. Further assume that s and t differ only in two bits, i and k, in such a way that $s_i = 1$, $s_k = 0$; $t_i = 0$, $t_k = 1$; and $p_i \geq p_k$. Then* $\mathrm{E}[F(s,\cdot)] \geq \mathrm{E}[F(t,\cdot)]$.

Proof. Express the expected F-score $\mathrm{E}[F(s,\cdot)]$ as a sum and split the summation into two parts:

$$\sum_y F(s,y)\,\Pr(y) = \sum_{\substack{y\\ y_i=y_k}} F(s,y)\,\Pr(y) + \sum_{\substack{y\\ y_i\neq y_k}} F(s,y)\,\Pr(y).$$

If $y_i = y_k$ then $F(s,y) = F(t,y)$, for three reasons: the number of ones in s and t is the same (namely m) by assumption; y is constant; and the number of true positives is the same, that is $s \cdot y = t \cdot y$. The latter holds because s and y agree everywhere except on i and k; if $y_i = y_k = 0$, then there are no true positives at i and k; and if $y_i = y_k = 1$ then s_i is a true positive but s_k is not, and conversely t_k is but t_i is not. Therefore

$$\sum_{\substack{y\\ y_i=y_k}} F(s,y)\,\Pr(y) = \sum_{\substack{y\\ y_i=y_k}} F(t,y)\,\Pr(y). \qquad (9)$$

Focus on those summands where $y_i \neq y_k$. Specifically group them into pairs (y,z) where y and z are identical except that $y_i = 1$ and $y_k = 0$, but $z_i = 0$ and $z_k = 1$. In other words, the two summations on the right-hand side of the following equality are carried out in parallel:

$$\sum_{\substack{y\\ y_i\neq y_k}} F(s,y)\,\Pr(y) = \sum_{\substack{y\\ y_i=1\\ y_k=0}} F(s,y)\,\Pr(y) + \sum_{\substack{z\\ z_i=0\\ z_k=1}} F(s,z)\,\Pr(z).$$

Then, focusing on s first:

$$\begin{aligned}
&F(s,y)\,\Pr(y) + F(s,z)\,\Pr(z)\\
&= \frac{(\beta+1)(A+1)}{m+\beta T}\,\Pr(y) + \frac{(\beta+1)A}{m+\beta T}\,\Pr(z)\\
&= [(A+1)p_i\,(1-p_k) + A\,(1-p_i)p_k]\ \frac{(\beta+1)}{m+\beta T}\,C\\
&= [p_i + (p_i+p_k-2p_ip_k)A - p_ip_k]\ \frac{(\beta+1)}{m+\beta T}\,C\\
&= [p_i + C_0]\,C_1,
\end{aligned}$$

where $A = s \cdot z$ is the number of true positives between s and z (s and y have an additional true positive at i by construction); $T = y \cdot y = z \cdot z$ is the number of positive labels in y and z (identical by assumption); and

$$C = \frac{\Pr(y)}{p_i\,(1-p_k)} = \frac{\Pr(z)}{(1-p_i)\,p_k}$$

is the probability of y and z evaluated on all positions except for i and k. This equality holds because of the zeroth-order Markov assumption (5) imposed on $\Pr(y)$. C_0 and C_1 are constants that allow us to focus on the essential aspects.

The situation for t is similar, except for the true positives:

$$\begin{aligned}
&F(t,y)\,\Pr(y) + F(t,z)\,\Pr(z)\\
&= \frac{(\beta+1)A}{m+\beta T}\,\Pr(y) + \frac{(\beta+1)(A+1)}{m+\beta T}\,\Pr(z)\\
&= [A\,p_i\,(1-p_k) + (A+1)\,(1-p_i)p_k]\ \frac{(\beta+1)}{m+\beta T}\,C\\
&= [p_k + (p_i+p_k-2p_ip_k)A - p_ip_k]\ \frac{(\beta+1)}{m+\beta T}\,C\\
&= [p_k + C_0]\,C_1
\end{aligned}$$

where all constants have the same values as above. But $p_i \geq p_k$ by assumption, $p_k + C_0 \geq 0$, and $C_1 \geq 0$, so we have

$$\begin{aligned}
F(s,y)\,\Pr(y) + F(s,z)\,\Pr(z) &= [p_i + C_0]\,C_1\\
\geq F(t,y)\,\Pr(y) + F(t,z)\,\Pr(z) &= [p_k + C_0]\,C_1,
\end{aligned}$$

and therefore

$$\sum_{\substack{y\\ y_i\neq y_k}} F(s,y)\,\Pr(y) \geq \sum_{\substack{y\\ y_i\neq y_k}} F(t,y)\,\Pr(y). \qquad (10)$$

The theorem follows from equality (9) and inequality (10). □

Proof of Theorem 1: $(\forall s \in S_m)\ \mathrm{E}[F(z^{(m)},\cdot)] \geq \mathrm{E}[F(s,\cdot)]$.

Observe that $z^{(m)} \in S_m$ by definition (see Section 2.3). For $m = 0$ and $m = n$ the theorem holds trivially because S_m is a singleton set. In the nontrivial cases, Theorem 2 is applied repeatedly. The string $z^{(m)}$ can be transformed into any other string $s \in S_m$ by repeatedly clearing a more likely set bit and setting a less likely unset bit.

In particular this can be done as follows: First, find the indices where $z^{(m)}$ and s disagree. By construction there must be an even number of such indices; indeed there are equinumerous sets

$$\left\{ i \mid z_i^{(m)} = 1 \wedge s_i = 0 \right\} \approx \left\{ j \mid z_j^{(m)} = 0 \wedge s_j = 1 \right\}.$$

This holds because the total number of ones is fixed and identical in $z^{(m)}$ and s, and so is the total number of zeroes. Next, sort those indices by non-increasing probability and represent them as $i_1, \ldots, i_k$ and $j_1, \ldots, j_k$. Let $s_0 = z^{(m)}$. Then let s_1 be identical to s_0 except that $s_{i_1} = 0$ and $s_{j_1} = 1$. Form $s_2, \ldots, s_k$ along the same lines and observe that $s_k = s$ by construction. By definition of $z^{(m)}$ it must be the case that $p_{i_r} \geq p_{j_r}$ for all $r \in \{1, \ldots, k\}$. Therefore Theorem 2 applies at every step along the way from $z^{(m)} = s_0$ to $s_k = s$, and so the expected utility is non-increasing along that path. □