CarpeDiem: an Algorithm for the Fast Evaluation of SSL Classifiers

Roberto Esposito ESPOSITO@DI.UNITO.IT
Daniele P. Radicioni RADICION@DI.UNITO.IT
Università di Torino, Dipartimento di Informatica, C.so Svizzera 185 – 10149 Torino, Italy

Abstract

In this paper we present a novel algorithm, CarpeDiem. It significantly improves on the time complexity of Viterbi algorithm, preserving the optimality of the result. This fact has consequences on Machine Learning systems that use Viterbi algorithm during learning or classification. We show how the algorithm applies to the Supervised Sequential Learning task and, in particular, to the HMPerceptron algorithm. We illustrate CarpeDiem in full details, and provide experimental results that support the proposed approach.

1. Introduction

Within the broader field of learning systems for sequential data, let us consider the *Supervised Sequential Learning* (SSL) task: in this particular problem, each observation in the sequence is associated with an individual label. The goal of SSL systems is to learn how to best predict the sequence of labels given the sequence of observations. More formally, the SSL task can be specified as follows (Dietterich, 2002):

Given: A set L of training examples of the form (X_m, Y_m), where each $X_m = (x_{m,1}, \ldots, x_{m,T_m})$ is a sequence of T_m feature vectors and each $Y_m = (y_{m,1}, \ldots, y_{m,T_m})$ is a corresponding sequence of class labels, $y \in \{1, ..., K\}$.

Find: A classifier H that, given a new sequence X of feature vectors, predicts the corresponding sequence of class labels $Y = H(X)$ accurately.

The SSL problem has been approached with many different techniques. Among others, we recall Sliding Windows (Dietterich, 2002), Hidden Markov Models (Rabiner, 1989), Maximum Entropy Markov Models (McCallum et al., 2000), Conditional Random Fields (Lafferty & Pereira, 2001), and the HMPerceptron algorithm (Collins, 2002).

In general, the labeling of an entire sequence X may show dependences among labels which extend for the whole labeling. In this case the labeling task must take into account every possible sequence of labels, resulting in a $\Theta(K^T)$ complexity. In practice, it is possible to evaluate H in polinomial time by making some simplifying assumption. In particular, assuming a *first order Markov property* (or the like, in non-probabilistic frameworks) allows evaluating H in quadratic time by means of the Viterbi algorithm (Viterbi, 1967). In point of fact, most state-of-the-art methods for dealing with the SSL task rely on the Viterbi algorithm for classifying and/or learning purposes.

In this work we introduce CarpeDiem, a novel algorithm designed for speeding up Viterbi "decoding". Provided that CarpeDiem has a plethora of possible applications, we show how it applies to a particular SSL system embedding the HMPerceptron learning algorithm. The HMPerceptron exploits the Viterbi algorithm at both learning and classification time; hence, it lends itself to clearly point out the advantages of CarpeDiem.

The paper is structured as follows: we briefly recall Viterbi and HMPerceptron algorithms. Then we illustrate CarpeDiem in full details. A complete example of its execution over a toy problem is then given and, in the end, we report the results obtained in experiments made on a real SSL task (the tonal harmony analysis problem). Finally, we discuss the results and compare with related works.

2. Background and Overview

Let us introduce few facts about the Viterbi and the HMPerceptron algorithms.

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

A) Background on Viterbi Algorithm. Given a layered and weighted graph with T layers and K nodes per layer, the algorithm evaluates the best path from the leftmost to the rightmost layer in $\Theta(TK^2)$ time. For each node of each layer, the algorithm stores the reward (the *accumulated metric*)[1] of the best path to that node. The reward for reaching a node y in the following layer is evaluated as the maximal reward that can be obtained by stepping from any node in the current layer to y. Since this is to be done for each node of the following layer, this step implies the quadratic dependence on the number of nodes per layer.

In other fields (e.g., telecommunications) there exist *ad hoc* solutions that allow one to tame the complexity by means of hardware implementations (Austin et al., 1990) and/or methods for approximating the optimum path (Fano, 1963). In learning systems, however, hardware implementations are seldom seen in practice, and suboptimal solutions are less interesting. As a consequence, the quadratic dependence on the number of labels is often a high burden.

B) HMPerceptron Algorithm and the Boolean Features Framework. The HMPerceptron uses Viterbi algorithm to find the optimum path in a graph where each *node* in a layer represents a possible *label*, and each *layer* represents a *time point* in the sequence being labelled.

The HMPerceptron has been defined in the *boolean features framework* (McCallum et al., 2000): the learnt classifier is built in terms of a linear combination of boolean features. Each feature reports about a salient aspect of the sequence to be labelled in a given time instant. More formally, given a time point t, a boolean feature is a 1/0-valued function of the whole sequence of feature vectors X, and of a restricted neighborhood of y_t. The function is meant to return 1 if the characteristics of the sequence X around time step t support the classifications given at and around y_t. The hypothesis has then the form

$$H(X) = \arg\max_{Y'} \sum_{t=1}^{T} \sum_{s} w_s \phi_s(X, y'_t, y'_{t-1}, t)$$

In this formulation, the features may take into consideration arbitrarily large neighborhoods of x_t to report about whether the labels y_t and y_{t-1} are likely to be correct.

From facts A) and B) it follows that, in the given application, Viterbi spends most computational resources to evaluate the formula:

$$\max_{y_t, y_{t-1}} \left[\omega_{y_{t-1}} + \sum_{s} w_s \phi_s(X, y_t, y_{t-1}, t) \right] \qquad (1)$$

where $\omega_{y_{t-1}}$ denotes the weight of the best path to label y_{t-1}.

In the present paper we describe an algorithm that both surmounts the mentioned complexity and eliminates the need for any graph pruning, thereby preserving the optimality of results. Under mild assumptions (see below) CarpeDiem approaches a $\Theta(TK\log(K))$ time complexity. Also, it smoothly fits to the problem at hand, so that the algorithm regresses to a $\Theta(TK^2)$ worst case complexity only when the problem at hand requires to inspect each and every transition edge to make a decision.

3. The CarpeDiem algorithm

The main idea we introduce in this paper stems from noting that in many application domains not all features really depend on both y_t, y_{t-1}. On the contrary, very often the characteristics of the example are, by themselves, very relevant for predicting y_t. Then, we identify two kind of features: the *vertical* features that do not require to know the previously predicted label –and work, thus, under a zero-order Markov assumption–, and the *horizontal* features that do need it –thereby working under a first-order Markov assumption. In the setting of an optimal path finding algorithm over a directed and weighted graph, one can think of vertical features as those allowing to assign a score to the nodes, whereas horizontal features allow assigning a score to transitions.

We will assume as given a finite set $\{\phi_1, \phi_2, \ldots, \phi_p\}$ of features suitable for the problem at hand, and partition the set $\{1, 2, \ldots p\}$ of all feature *indexes* into the two sets Φ^0 and Φ^1 corresponding to indexes of vertical and horizontal features respectively. Now, Formula 1 can be rewritten as shown by Equation 2 in Fig. 1. Equation 3 (see Fig. 1) is equivalent to Equation 2 and, in addition, it emphasizes how features in Φ^0 need to be evaluated only once per y_t label (i.e., not once for each y_t,y_{t-1} pair). It is then obvious (and in accord with the intuition) that whenever $\Phi^1 = \emptyset$ the cost of the Viterbi algorithm can be reduced to be linear in the number of labels.

Following this intuition, we propose an algorithm that exploits vertical information as much as possible, resorting to the use of horizontal features only in case this is really necessary to the classification purposes.

[1] In *shortest* path search algorithms, it is customary to use the term *cost* to refer the accumulated metric of the node. We use the term *reward* to stress the connection with the *maximization* problem.

$$\max_{y_t, y_{t-1}} \left[\omega_{y_{t-1}} + \sum_{s \in \Phi^0} w_s \phi_s(X, y_t, t) + \sum_{s \in \Phi^1} w_s \phi_s(X, y_t, y_{t-1}, t) \right] \quad (2)$$

$$\max_{y_t} \left[\sum_{s \in \Phi^0} w_s \phi_s(X, y_t, t) + \max_{y_{t-1}} \left[\omega_{y_{t-1}} + \sum_{s \in \Phi^1} w_s \phi_s(X, y_t, y_{t-1}, t) \right] \right] \quad (3)$$

Figure 1.

The algorithm can be best described as a twofold search strategy. The main *forward* search strategy scans the layers in the graph from left to right. For each layer it finds the node with the best possible reward and stops as soon as this node can be determined. In so doing, it possibly leaves a layer without having evaluated the exact reward for reaching all nodes of that layer. We will call *open* the nodes y_t for which the *true* reward $z(y_t)$ has been computed, and *closed* those ones for which a complete search has not been done and, thus, only the vertical weight is known.

The *backward* strategy is called only when necessary. Its purpose is precisely to open nodes (i.e., it evaluates $z(y_t)$ for a given y_t) opening the least possible number of previous nodes. Opening a node may imply the need to go back to nodes in previous layer(s) to gather information left unspecified during the forward step.

In the best case, the algorithm scans only the most promising node for each layer and never calls the backward strategy. In such a case, the cost of the algorithm would be $\Theta(K \log(K) T)$, where the factor $K \log(K)$ is due to the time spent for sorting the nodes in each layer (more about this later on)[2]. In the worst case (no vertical features), the algorithm has a complexity of $K^2 T$, i.e., in any case the algorithm is not asymptotically worse than Viterbi algorithm.

3.1. Algorithm details

Let us consider Equation 3 and denote with $\Sigma^0_{y_t}$ the summation over all features in Φ^0, and with $\Sigma^1_{y_t, y_{t-1}}$ the summation over Φ_1, i.e.:

$$\Sigma^0_{y_t} = \sum_{s \in \Phi^0} w_s \phi_s(X, y_t, t)$$

$$\Sigma^1_{y_t, y_{t-1}} = \sum_{s \in \Phi^1} w_s \phi_s(X, y_t, y_{t-1}, t)$$

[2]In cases where the vertical rewards range over a limited interval, one could reduce the $K \log(K)$ factor to K by using standard techniques (Cormen et al., 1990, Chap. 9). However, in the present application, this amounts to trade space for speed.

In summary, the algorithm efficiently implements the search for:

$$\max_{y_t} \left[\Sigma^0_{y_t} + \max_{y_{t-1}} \left[\omega_{y_{t-1}} + \Sigma^1_{y_t, y_{t-1}} \right] \right]$$

3.1.1. Forward search strategy.

The forward search strategy consists in *i*) sorting the nodes in the current layer according to the values of $\Sigma^0_{y_t}$, and *ii*) exploiting a *bound* β_t on $(\omega_{y_{t-1}} + \Sigma^1_{y_t, y_{t-1}})$ in order to avoid evaluating the inner maximization whenever possible.

Let us define

$$\beta_t = \omega^*_{t-1} + \Sigma^{1*},$$

where ω^*_{t-1} is the reward of the best path to any node in layer $t-1$ (including the vertical weight of the ending node), and Σ^{1*} is the sum of the weights of *horizontal* features associated to positive weights:

$$\Sigma^{1*} = \sum_{s \in \Phi^1 : w_s > 0} w_s$$

Let also $\sqsupseteq_t$ be a total ordering of nodes –based on vertical weights– at layer t, defined as

$$\sqsupseteq_t \equiv \{(y_t, y'_t) | \text{if } \Sigma^0_{y_t} \geqslant \Sigma^0_{y'_t}\}.$$

At any time t, we say that node y_t is more promising than node y'_t iff $y_t \sqsupseteq_t y'_t$.

The main routine of `CarpeDiem` (not reported here due to space reasons) simply calls the forward search strategy at each time point t. The forward strategy finds the best node for layer t stopping as soon as this node can be determined without doubts. The forward strategy is detailed in Algorithm 1.

At the beginning of the analysis of each layer all nodes are *closed*. As asserted by the **while**-loop condition, the algorithm stops as soon as the vertical weight of the following node in the $\sqsupseteq$ ordering plus β_t is lower than the current estimated best reward. Being β_t an upper-bound on the reward that can be obtained for reaching a node at layer t, exiting the **while**-loop ensures that y'_t cannot improve the reward obtained by

begin
  $y_t^* \leftarrow$ most promising node;
  $y'_t \leftarrow$ next node in the $\sqsupseteq_t$ ordering;
  Open node y_t^* {call the backward strategy};
  while $z(y_t^*) < \beta_t + \Sigma^0_{y'_t}$ **do**
    Open node y'_t {call the backward strategy};
    $y_t^* \leftarrow \arg\max_{y'' \in \{y_t^*, y'_t\}} [z(y'')]$;
    $y'_t \leftarrow$ next node in the $\sqsupseteq_t$ ordering;
  end
  return y_t^*;
end

Algorithm 1: Forward search strategy

y_t^*. Moreover, since nodes are considered in the order given by $\sqsupseteq_t$, none of the following nodes can improve y_t^*.

Informally stated, the algorithm significantly reduces the search effort when: *i*) vertical features are sufficient to discriminate well among labels (i.e., few distinguished labels have very high rewards) and/or, *ii*) horizontal features do not provide significant cues (i.e., most transition weights are close to Σ^{1*}). For instance, if all horizontal features are zero, ($\Sigma^{1*} = 0$), then all nodes of layer t share the same ancestor: the node with weight ω^*_{t-1}. Then, $z(y_t^*) = \omega^*_{t-1} + \Sigma^0_{y_t^*}$, and the test for continuing the loop can be rewritten as

$$\omega^*_{t-1} + \Sigma^0_{y_t^*} < \omega^*_{t-1} + 0 + \Sigma^0_{y'_t} \Leftrightarrow \Sigma^0_{y_t^*} < \Sigma^0_{y'_t}$$

Since y_t^* is initially set to the most promising node, we have that the requirement $\Sigma^0_{y_t^*} < \Sigma^0_{y'_t}$ is never met: the loop is exited immediately (yielding a $\Theta(TK\log(K))$ time complexity).

On the other hand, the algorithm may not be as useful when, for instance, all $\Sigma^0_{y_t} = \tau$ (τ being a constant). For the sake of simplicity, let us assume that $\Sigma^{1\blacktriangle}$, the weight of the best edge in the graph, is strictly lower than Σ^{1*}, i.e., no edge yields an horizontal weight equal to Σ^{1*}. Since $z(y_t^*)$ is at most $\omega^*_{t-1} + \Sigma^{1\blacktriangle} + \tau$ and $\beta_t + \Sigma^0_{y'_t} = \omega^*_{t-1} + \Sigma^{1*} + \tau$, it follows that $z(y_t^*) < \beta_t + \Sigma^0_{y'_t}$ is always met: the algorithm will open each and every node. In such a situation, the algorithm inspects each edge in the graph and the time complexity degrades to Viterbi's $\Theta(TK^2)$, i.e., the CarpeDiem is never asymptotically worse than Viterbi.

3.1.2. Backward search strategy

The backward search strategy opens a node y_t finding its best ancestor and setting $z(y_t)$ to the weight of the best path to the node. In much the same spirit of the forward strategy, the algorithm saves some computation *i*) by exploiting $\sqsupseteq_{t-1}$ in order to inspect first the most promising nodes, and *ii*) by taking again advantage of β_{t-1} in order to stop the search as soon as possible. The exact strategy is implemented by Algorithm 2.

Data: A node y_t to be opened
begin
  $y^*_{t-1} \leftarrow$ most promising node;
  $y'_{t-1} \leftarrow$ next node in the $\sqsupseteq_{t-1}$ ordering;
  while y'_{t-1} *is open* **do**
    $y^*_{t-1} \leftarrow \arg\max_{y'' \in \{y'_{t-1}, y^*_{t-1}\}} \left[z(y'') + \Sigma^1_{y_t, y''}\right]$;
    $y'_{t-1} \leftarrow$ next node in the $\sqsupseteq_{t-1}$ ordering;
  end
  while $\left(z(y^*_{t-1}) + \Sigma^1_{y_t, y^*_{t-1}} < \beta_{t-1} + \Sigma^0_{y'_{t-1}} + \Sigma^{1*}\right)$ **do**
    Open y'_{t-1} {call the backward strategy};
    $y^*_{t-1} \leftarrow \arg\max_{y'' \in \{y'_{t-1}, y^*_{t-1}\}} \left[z(y'') + \Sigma^1_{y_t, y''}\right]$;
    $y'_{t-1} \leftarrow$ next node in the $\sqsupseteq_{t-1}$ ordering;
  end
  $z(y_t) \leftarrow z(y^*_{t-1}) + \Sigma^1_{y_t, y^*_{t-1}} + \Sigma^0_{y_t}$;
end

Algorithm 2: Backward search strategy to open y_t.

The first loop of the algorithm finds the best ancestor among the open nodes in layer $t-1$. No shortcuts can be taken at this point, since *i*) the open nodes are still sorted according to $\sqsupseteq_{t-1}$, which is only mildly related to nodes actual weight; *ii*) in any case, in order to make a decision, the transition to the node being opened must be taken into account[3].

The second loop in the algorithm exploits a bound based on β_{t-1} in order to stop the search for the optimal ancestor as soon as possible. This amounts to exiting the loop if the estimated reward for the node currently inspected

$$\beta_{t-1} + \Sigma^0_{y'_{t-1}} + \Sigma^{1*}$$

is lower than the true reward of the current best ancestor,

$$z(y^*_{t-1}) + \Sigma^1_{y_t, y^*_{t-1}}$$

[3] Actually, we could sort the open nodes in the layer according to their weight $z(\cdot)$ and use Σ^{1*} to cut the search. However, this would be really rewarding only in case there are many open nodes in the layer (otherwise the linear search does not cost that much). By converse, a similar setting would suggest that the vertical weights are not *much* discriminating, thus making unlikely that using Σ^{1*} to cut the search provides significant improvements.

In this case, also the estimated rewards for following nodes cannot be higher.

When the loop condition is met, the procedure calls itself recursively to open the currently inspected node: its actual weight is evaluated and, if it improves y^*_{t-1}, it is set to be the best ancestor.

As our strategy cuts the search only when $\beta_t + \Sigma^0_{y'_t}$ is not larger of the weight of the current best note (see Alg.1), it becomes important to be confident that the estimate $\beta_t + \Sigma^0_{y'_t}$ cannot be lower than $\omega_{y'_t}$, the true weight of y'_t. This is indeed the case. In fact, $\omega_{y'_t}$ is the sum of the true reward to its best ancestor (y''_{t-1}), the vertical weight of y'_t ($\Sigma^0_{y'_t}$), and the horizontal weight for the transition $y''_{t-1} \rightarrow y'_t$, ($\Sigma^1_{y'_t,y''_{t-1}}$). That is:

$$\omega_{y'_t} = \omega_{y''_t} + \Sigma^0_{y'_t} + \Sigma^1_{y'_t,y''_{t-1}}.$$

By definition: $\omega^*_{t-1} \geq \omega_{y''_{t-1}}$. Moreover, since Σ^{1*} is the sum of all positive scores of horizontal features, it follows that $\Sigma^{1*} \geq \Sigma^1_{y'_t,y''_{t-1}}$. This suffices to see that our bound meets the requirement, since:

$$\beta_t + \Sigma^0_{y'_t} = \omega^*_{t-1} + \Sigma^0_{y'_t} + \Sigma^{1*} \geq \omega_{y_{t-1}} + \Sigma^0_{y'_t} + \Sigma^1_{y'_t,y''_{t-1}} = \omega_{y'_t}$$

3.2. Example

In the following we provide a description of the algorithm functioning over a toy problem.

The problem consists in labeling a sequence containing four events and two labels (named i and j). The example is reported in Fig. 2. The weight shown on the edge between labels y_{t-1} and y_t corresponds to $\Sigma^1_{y_t,y_{t-1}}$. The bound Σ^{1*} on the maximum horizontal reward is 60. Two further quantities are reported in the figure and shown graphically by means of boxes placed on nodes: within rectangular boxes, we report the vertical weight of the node; within rounded boxes, we report, depending on whether the node is open or closed, the true weight (for open ones), or the quantity $\beta_t + \Sigma^0_{y_t}$ (for closed ones).

step (a) At the beginning of the execution, all nodes are closed. The first step of the algorithm consists in opening all nodes in layer 1. Clearly, there is no reward for arriving at nodes in layer zero and no incoming transitions to take into account. The best node so far is thus the node having the maximum vertical weight. The weight of this node is the weight of the best path to layer 1, i.e., $\omega^*_1 = 100$.

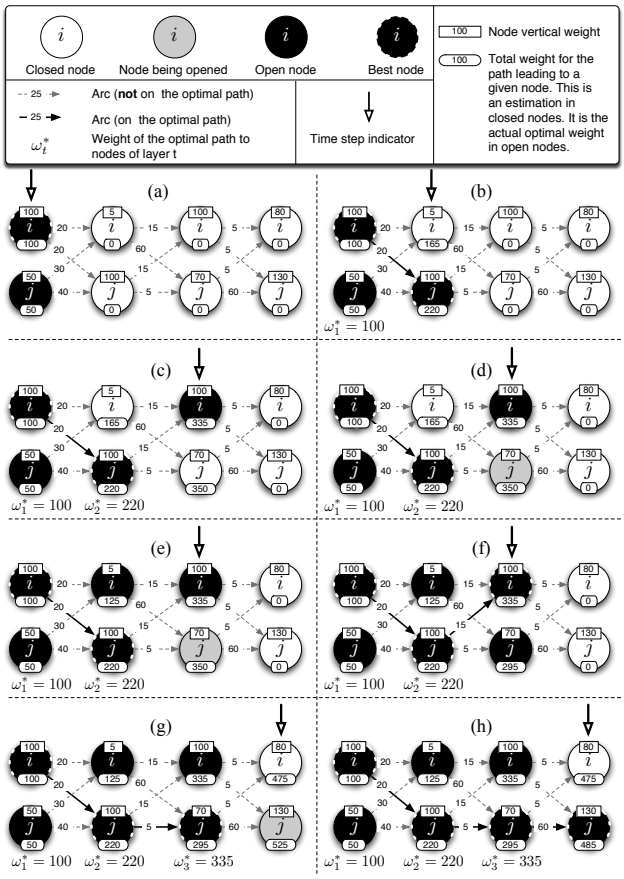

Figure 2. CarpeDiem in action on a toy problem.

step (b) The analysis of layer 2 starts by considering (i.e., opening) the most promising node in that layer (node j). Since all nodes at layer 1 are open, the backward strategy simply picks the node on the best path to node j. Once the true weight of node j is known, the algorithm compares it with the bound on the weight of the best path to node i. Since the bound $\Sigma^0_{i_2} + \beta_2 = 165$ cannot outperform the weight $z(j_2) = 220$, there is no need to open node i_2.

step (c) To open node i in layer 3, the backward strategy goes back to layer 2 and searches for the best path to that node. Again, node i_2 can be left closed since there is no chance that the best path to i_3 traverses it. In fact,

$$\beta_2 + \Sigma^0_{i_2} + \Sigma^{1*} = (100 + 60) + 5 + 60 = 225$$

cannot outperform the reward

$$z(j_2) + \Sigma^1_{i_3,j_2} = 220 + 15 = 235$$

obtained for taking the path through node j_2. The true weight of the path to i_3 is, thus, $235 + 100 =$

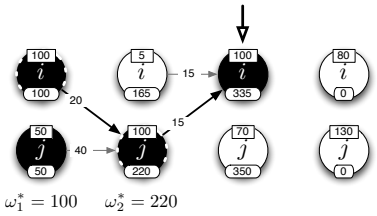

Figure 3. The graph shows only the edges that have been actually inspected at step (c) of example in Fig. 2.

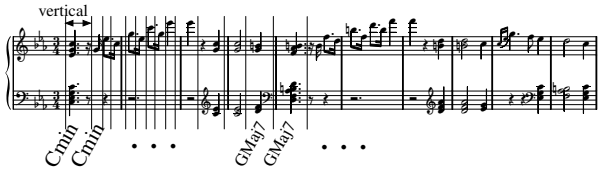

Figure 4. The tonal harmony analysis problem consists of indicating for each vertical which chord is currently sounding.

335. Unfortunately, this does not allow to make a definitive decision about whether this is the best node of layer 3, since $\beta_3 + \Sigma^0_{j_3}$ is $(220+60)+70 = 350$. Next step will thereby settle the question by opening node j_3.

step (d) The goal is, at this point, to find the best path to node j_3. Even though j_2 has a clear advantage over i_2, this does not suffice for excluding that the latter is on the optimal path to node j_3 (since $z(j_2) + \Sigma^1_{j_3,j_2} \not< \beta_2 + \Sigma^0_{i_2} + \Sigma^{1*}$): the backward strategy is then forced to recursively call itself to open node i_2.

step (e) By opening i_2, the algorithm discovers that the true reward to reach this node is 125 (through the path $i_1 \rightarrow i_2$), thus ruling it out as a candidate for being on the optimal path to node j_3.

step (f) In returning to consider the problem of evaluating the best node in layer 3, we are back to the first hypothesis, path $i_1 \rightarrow j_2 \rightarrow i_3$. To open nodes i_2 and j_3 has been wasteful, though unavoidable.

step (g) j_4 is the first node to be opened in its layer. Interestingly, the best path to j_4 is not through the best node in layer 3. In fact, while the highest reward for a three steps walk is on node i_3, it is more convenient to go through node j_3 to reach node j_4.

step (h) Since the estimated reward associated to i_4 is smaller than the true weight of j_4, we can end the algorithm leaving node i_4 closed.

In this artificial example, the algorithm does not save much computation: at the end of the execution a single node remain unopened. However, if we consider Fig. 3 (that reports only the edges evaluated at the end of step (c)), we note how CarpeDiem inspected only a reduced set of the edges of the graph. By continuing the execution, however, the advantage decreases since the example has been designed to clarify the strategy followed by the algorithm rather than showing a case where a significant time-saving is obtained. In the following Section a more realistic setting is considered, and a much larger optimization obtained.

4. Experiments

The running time of an execution of CarpeDiem heavily depends on how the weights of vertical and horizontal features compare. Then, we believe mandatory to present a situation in which weights are representative of real world classifiers. In the following, we focus on the particular application of "tonal harmony analysis": the problem of identifying the chord sounding at each time point of a music excerpt. Please refer to (Radicioni & Esposito, 2006) for a detailed description of a similar system.

4.1. Harmony Analysis

Harmony analysis is arguably one of the most sophisticated tasks that musicians deal with, and a formidable challenge for Sequential Learning, in that: *i*) it can be naturally cast to a sequential problem; *ii*) an intuitively neat separation between horizontal (that refers to the musical flow) and vertical (that pertains simultaneous sounds) features exists; *iii*) K (the number of labels) is over one hundred, thereby pointing out a typical case where Viterbi algorithm shows bad time performance.

Analyzing music harmony consists in associating a label to each *vertical* (that is, set of simultaneous notes) (Pardo & Birmingham, 2002). Such labels explain which harmony is sounding, by indicating a chord name through a fundamental note (*root*) and a *mode*, such as C minor. Given a score in MIDI format, we individuate sets of simultaneous notes (*verticals* or *events*), and associate a *label* ⟨fundamental note, mode⟩ to each vertical (Fig. 4).

Music analysis task can be naturally represented as a Machine Learning classification problem, suitable to be solved by SSL techniques. In fact, by considering

only the "vertical" aspects of musical structure, one would hardly produce reasonable analyses. Experimental evidences about human cognition reveal that in order to disambiguate unclear cases, composers and listeners refer to "horizontal" features of music as well: in these cases, context plays a fundamental role, and contextual cues can be useful to the analysis system. Moreover, harmony changes are well known to follow patterns where analysis must take into consideration the succession of chords (e.g., the case of *cadence*).

Let us go back to the SSL definition: in the case of music analysis, each X_m corresponds to a particular piece of music; $x_{m,t}$ is the information associated to the event at time t; and $y_{m,t}$ corresponds to the chord label (i.e., the chord root and mode) associated to the event sounding at time t. The problem is, thus, to learn how to predict accurately the chord labels given the information about musical events.

By definition, a feature $\phi_s(X, y_t, y_{t-1})$ is a boolean function of the entire sequence and of labels y_t and y_{t-1}. Usually, it analyzes a small neighborhood of the current event, and return 1 if there is evidence that the currently predicted label is correct. For the present application, the features have been engineered so that they take into account the prescriptions from Music Harmony Theory, a field where vertical and horizontal features naturally arise. *Vertical* features report about simultaneous sounds and their correlation with the currently predicted chord. *Horizontal* features capture metric patterns and chordal successions.

Goal of the present experimentation is to discover whether `CarpeDiem` obtains significant running time improvements w.r.t. Viterbi. To this aim, we embedded it in a SSL system implementing the HMPerceptron learning algorithm (Collins, 2002). The learning system has been trained on a data set composed of 30 chorales by J.S. Bach (1675-1750). The learnt weights have been then used to build two classifiers: one based on standard Viterbi, the other one based on `CarpeDiem`. The two classifiers have been fed with 42 testing sequences, their running time has been recorded and reported in the following.

4.2. Results

Needless to say, being equal their results, both algorithms provided the same accuracy results (on average, 79% accuracy rate).

The Viterbi-based HMPerceptron took 62,333 seconds of CPU time in order to complete learning, on the same task the `CarpeDiem`-based HMPerceptron took only 10,866 seconds, with a net saving of 82.56%.

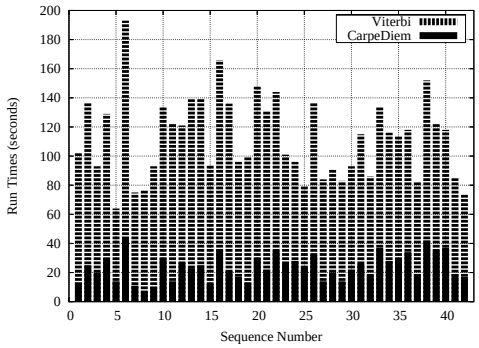

Figure 5. Times spent by Viterbi and `CarpeDiem` in analyzing the test set.

Fig. 5 reports the time spent by both algorithms to analyze each sequence of the test set. On average, 79% of the testing time has been saved. In the best case `CarpeDiem` ran in 7 seconds instead of 77, thus saving 90% of the time. In the worst case it ran in 37 seconds instead of 118, thus saving 68.64% of the time. Also, the magnitude of the improvement does not show large variations depending on the sequences, thus encouraging the generalization of these results to new musical pieces.

5. Discussion

The algorithm performs surprisingly well on this task: clearly, in this domain, much information is retrieved based on vertical features. Nonetheless, horizontal features are necessary for obtaining "good" classifications: we observed that the classification accuracy substantially drops if horizontal features are removed.

About the running behavior of the algorithm, it is interesting that there are places in the music where harmony is explicitly stated (thus being easier to analyze), and other places where it is not. In the former places, the algorithm *skims* through the score placing the labels confidently, meanwhile requiring few computational resources. Somewhere, however, the algorithm slows down requiring more analysis in order to pronounce. This observation is interesting from the computational point of view: it shows that the algorithm adapts to the problem at hand; it is also relevant from a cognitive perspective: it suggests that difficult sections of the excerpts can be automatically identified, thus providing a sort of complexity measure for excerpts of the piece being analyzed.

As earlier mentioned, most of Viterbi algorithm variants rely on hardware implementations or renounce to optimality. Recent works proposed algorithms that improve the running time with respect to Viterbi, by

assuming that the HMM transition matrix obeys certain constraints (Siddiqi & Moore, 2005; Felzenszwalb et al., 2003). In Siddiqi and Moore (2005) the authors assume that few transition weights are of interest, and that the other ones can be approximated by a constant. Then they use this fact to reduce the search to few nodes per layer. This approach works well in many conditions, but it has the downside of giving up optimality in case the transition matrix does not naturally fits this form. In Felzenszwalb et al. (2003) the authors assume that nearby labels have "similar" weights and use this structural information to improve the search for the highest ranking node. On our side, we assume that the vertical information suffices to avoid the inspection of most of the horizontal features. The success of both approaches, the one by Felzenszwalb et al. and ours, depends on how well the real world conditions match the respective assumptions.

In the previous Section we showed that the Musical Analysis task naturally fits the assumptions underlying `CarpeDiem`; we now spend some concluding remarks arguing about why these assumptions should generalize to other domains. Let us consider how vertical and horizontal features come into play in traditional sequential tasks, such as speech recognition and the OCR task. In speech recognition the phoneme sounding at a given instant provides much information about the classification. That is, the phoneme /d/ is likely to be confused with /t/, but most probably not with /tʃ/. In the OCR task, the features describing the current letter may be uncertain among few characters, but it is seldom the case that this uncertainty concerns all the letters of the alphabet.

6. Conclusion

In this paper we introduced `CarpeDiem`, a novel algorithm that improves on Viterbi time complexity. `CarpeDiem` employs a bound to use contextual information only when necessary. We provided the new algorithm with full details, experimentally verified that it brings significant time savings, and advocated that it can be generalized to further applicative domains.

Acknowledgments

We wish to thank Lorenza Saitta for her precious advices.

References

Austin, S., Peterson, P., Placeway, P., Schwartz, R., & Vandergrift, J. (1990). Toward a Real-Time Spoken Language System Using Commercial Hardware. *DARPA Speech and Natural Language Workshop* (pp. 72–77).

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. *Proc. of the Conf. on Empirical Methods in Natural Language Processing*.

Cormen, T. H., Leiserson, C., & Rivest, R. L. (1990). *Introduction to Algorithms*. Cambridge, MASS: MIT Press.

Dietterich, T. (2002). Machine Learning for Sequential Data: A Review. *Structural, Syntactic, and Statistical Pattern Recognition* (pp. 15–30). Springer-Verlag.

Fano, R. M. (1963). A heuristic discussion of probabilistic decoding. *IEEE T. Inform. Theory*, *9*, 64–73.

Felzenszwalb, P. F., Huttenlocher, D. P., & Kleinberg, J. M. (2003). Fast Algorithms for Large-State-Space HMMs with Applications to Web Usage Analysis. *Advances in Neural Information Processing System*.

Lafferty, J., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. of the Eighteenth International Conference on Machine Learning* (pp. 282–289). San Francisco, CA: Morgan Kaufmann.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. *Proc. 17th International Conf. on Machine Learning* (pp. 591–598). Morgan Kaufmann, San Francisco, CA.

Pardo, B., & Birmingham, W. P. (2002). Algorithms for Chordal Analysis. *Comput. Music J.*, *26*, 27–49.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proc. of the IEEE*, *77*, 267–296.

Radicioni, D. P., & Esposito, R. (2006). A Conditional Model for Tonal Analysis. *Foundations of Intelligent Systems* (pp. 652–661). Berlin: Springer-Verlag.

Siddiqi, S. M., & Moore, A. W. (2005). Fast Inference and Learning in Large-State-Space HMMs. *Proc. of the 22nd International Conf. on Machine Learning*.

Viterbi, A. J. (1967). Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE T. Inform. Theory* (pp. 260–269).