Algorithm 1. Cutting Plane Algorithm (dual) with uniform sampling

1: Input: \((x_1, y_1), \ldots, (x_n, y_n), C, \epsilon\)
2: \(S \leftarrow \emptyset; t = 0;\)
3: repeat
4: Update matrix \(H\) with a new constraint
5: \(\alpha \leftarrow\) optimize QP problem (5)
6: \(\xi = \frac{1}{C} (h^T \alpha - \frac{1}{2} \alpha^T H \alpha)\)
7: \(w = -\frac{1}{|S|} \sum_{j=1}^{|S|} \alpha_j g^{(j)}\)
8: Sample \(r\) examples from the training set /* find the most violated constraint (cutting plane) */
9: for \(i = 1\) to \(r\) do
10: \(c_i \leftarrow \begin{cases} 1 & y_i (w \cdot \phi(x_i)) \leq 1 \\ 0 & \text{otherwise} \end{cases}\)
11: end for
12: \(h^{(t)} = \frac{1}{r} \sum_{i=1}^r c_i\)
13: \(g^{(t)} = -\frac{1}{r} \sum_{i=1}^r c_i y_i \phi(x_i)\) /* add a constraint to the active set */
14: \(S \leftarrow S \cup \{(h^{(t)}, g^{(t)})\}\)
15: \(t = t + 1\)
16: until \(h^{(t)} + w \cdot g^{(t)} \leq \xi + \epsilon\)
17: return \(w, \xi\)

problem is constructed. The algorithm stops when no constraints are violated by more than \(\epsilon\), which is formalized by the criteria in line 16.

The analysis of the inner product given by (7) reveals that, since it needs to be computed for each training example, it requires the time \(O(n^2 + Tn)\) after total of \(T\) iterations. Similarly, as we add a cutting plane to \(S\) at each iteration \(t\), a new column is added to the matrix \(H\) (Algorithm 1, line 4) requiring the computation of

\[
H_{it} = g^{(t)} \cdot g^{(t)} = \frac{1}{n^2} \sum_{k=1}^n \sum_{l=1}^n c_{ki} c_{li} y_k y_l K(x_k, x_l)
\]

which takes \(O(Tn^2)\). Thus, the obtained \(O(n^2)\) scaling behavior makes cutting plane training no better than conventional decomposition methods.

To address this limitation, we employ the approach of Yu and Joachims [28] to construct approximate cuts by sampling \(r\) examples from the training set. They suggest two strategies to sample examples, namely uniform and importance sampling (the pseudocode of the algorithm using uniform sampling is presented in Algorithm 1). These two strategies derive constant-time and linear-time algorithms. The former uniformly samples \(r\) examples from the training set to approximate the cut. Thus, we approximate a subgradient \(g^{(t)}\) with only \(r\) examples, which replaces the number of expensive kernel evaluations in (7) over \(n\) by a more tractable: \(\sum_{i,j=1}^r K(x_i, x_j)\) (lines 9-13). The importance sampling acts in a more targeted way as it looks through the whole dataset to compute two cutting planes, one to be used in the optimization problem (line 5), and the
other for termination criterion (line 16). The training complexity reduces from $O(n^2)$ to $O(T^2r^2)$, when the uniform sampling algorithm is used, to $O(Tnr)$ for the importance sampling.

4 Tree Kernels

The main idea underlying tree kernels is to compute the number of common substructures between two trees $T_1$ and $T_2$ without explicitly considering the whole fragment space. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ be the set of tree fragments and $\chi_i(n)$ an indicator function equal to 1 if the target $f_i$ is rooted at node $n$ and equal to 0 otherwise. A tree kernel function over $T_1$ and $T_2$ is defined as

$$TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2),$$

where $N_{T_1}$ and $N_{T_2}$ are the sets of nodes in $T_1$ and $T_2$, respectively, and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} \chi_i(n_1)\chi_i(n_2).$$

The $\Delta$ function is equal to the number of common fragments rooted in nodes $n_1$ and $n_2$ and thus depends on the fragment type.

4.1 Fragment Types

In [17], we pointed out that there are three main categories of fragments: the subtree (ST), the subset tree (SST) and the partial tree (PT) fragments corresponding to three different kernels. STs are fragments rooted in any node of a tree along with all its descendants. The SSTs are more general structures since, given the root node of an SST, not all its descendants (with respect to the referring tree) have to be included, i.e. the SST leaves can be non-terminal symbols. PT fragments are still more general since their nodes can contain a subset of the children of the original trees, i.e. partial sequences.

For example, Figure 1 illustrates the syntactic parse tree of the sentence *Autism is a disease* on the left along with some of the possible fragments on the right of the arrow. ST kernel generates complete structures like $[D a]$ or $[NP [D a] [N disease]]$. SST kernel can generate more structures, e.g. $[NP [D] [N disease]]$ whereas PT kernel can also separate children in the fragments, e.g. $[NP [N disease]]$, and generate the individual tree nodes as features, e.g. *Autism* or *VBZ*.

[30] provided a version of SST kernel, which also generates leaves, i.e. words, as features, hereafter, SST-bow. However, such lexical features, when the data is very sparse, tend to cause overfitting. Thus, we give the definition of a variant of PTK, namely, the unlexicalized partial tree kernel (uPTK), which does not include lexicals and individual nodes in the feature space. This will promote the importance of structural information.