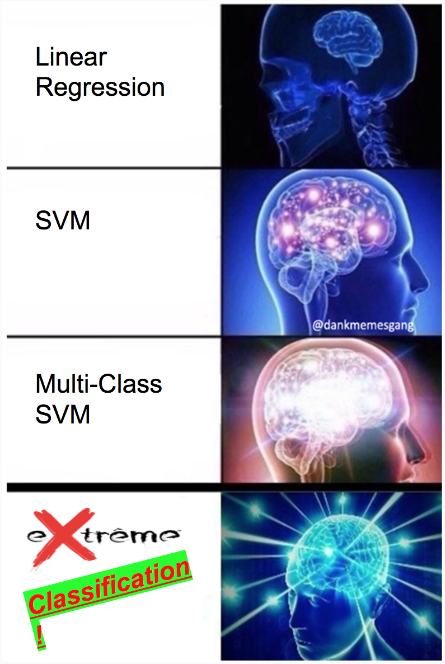
eXtrême Classification

Efficient Applications to Tree-Based Multi-class Models

Anish Thilagar
Ashwin Balakrishna

May 2, 2017

CS 159 - Caltech


Linear Regression
SVM
@dankmemesgang
Multi-Class SVM
eXtrême
Classification
!

Motivation

Classification

In the standard classification problem, we have items that each belong to one of k categories. To classify, calculate score for each category and pick the best one.

What happens when this framework doesn't work?

- Some items in multiple categories

Classification

In the standard classification problem, we have items that each belong to one of k categories. To classify, calculate score for each category and pick the best one.

What happens when this framework doesn't work?

- Some items in multiple categories
- Some types of misclassifications extremely costly

Classification

In the standard classification problem, we have items that each belong to one of k categories. To classify, calculate score for each category and pick the best one.

What happens when this framework doesn't work?

- Some items in multiple categories
- Some types of misclassifications extremely costly
- n extremely large

Classification

In the standard classification problem, we have items that each belong to one of k categories. To classify, calculate score for each category and pick the best one.

What happens when this framework doesn't work?

- Some items in multiple categories
- Some types of misclassifications extremely costly
- n extremely large

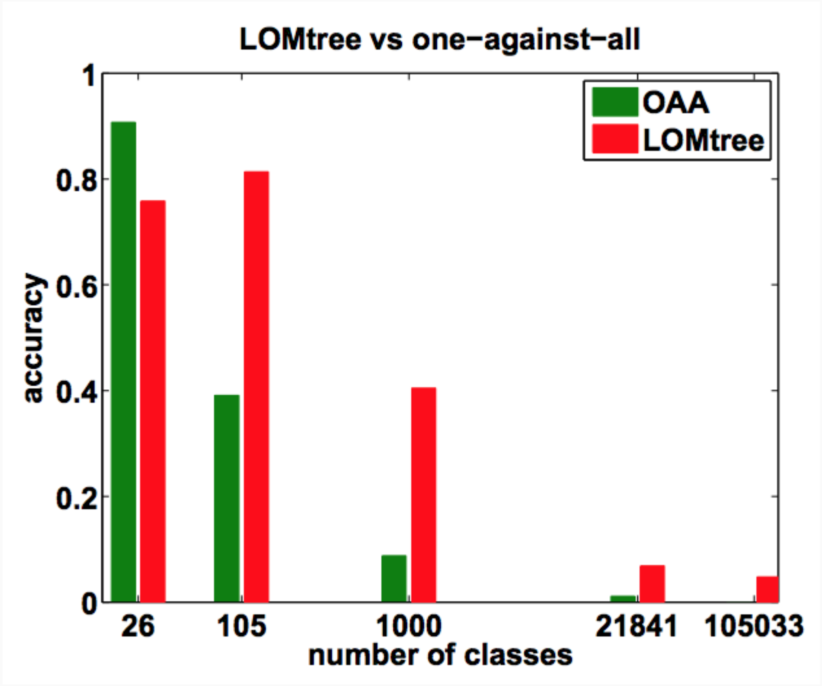
Classification

In the standard classification problem, we have items that each belong to one of k categories. To classify, calculate score for each category and pick the best one.

What happens when this framework doesn't work?

- Some items in multiple categories
- Some types of misclassifications extremely costly
- n extremely large

We turn to extreme classification.


LOMtree vs one−against−all
accuracy
1
0.8
0.6
0.4
0.2
0
26
105
1000
21841
105033
number of classes
OAA
LOMtree

Examples

- Classifying animals along phylogenetic tree

Examples

- Classifying animals along phylogenetic tree
- Classifying text by genre

Examples

- Classifying animals along phylogenetic tree
- Classifying text by genre
- Classifying patent documents by categories

Examples

- Classifying animals along phylogenetic tree
- Classifying text by genre
- Classifying patent documents by categories
- Gene/biological classification

Goal

Given a trained model, normal classification is $O(k)$. This becomes bad when k is large.

Given k labels, what's the best classification time we can hope for?

Goal

Given a trained model, normal classification is $O(k)$. This becomes bad when k is large.

Given k labels, what's the best classification time we can hope for?

From Kraft's inequality, we know the minimum time to just compute the bit representation of a label X is $O(H(X))$, the Shannon entropy, regardless of our bit representation. In the worst case, this becomes $\Omega(\log k)$. This gives us a strict lower bound on how we can even represent a category, because we need at least $\log k$ bits.

Goal

Given a trained model, normal classification is $O(k)$. This becomes bad when k is large.

Given k labels, what's the best classification time we can hope for?

From Kraft's inequality, we know the minimum time to just compute the bit representation of a label X is $O(H(X))$, the Shannon entropy, regardless of our bit representation. In the worst case, this becomes $\Omega(\log k)$. This gives us a strict lower bound on how we can even represent a category, because we need at least $\log k$ bits.

Recall, the Shannon entropy of a random variable $\mathcal{X} \in \{x_1, ..., x_n\}$ is just $H(\mathcal{X}) = -\sum_{i=0}^{n} p(x_i) \log p(x_i) = -\mathbb{E}(\log p(x_i))$.

Goal

Given a trained model, normal classification is $O(k)$. This becomes bad when k is large.

Given k labels, what's the best classification time we can hope for?

From Kraft's inequality, we know the minimum time to just compute the bit representation of a label X is $O(H(X))$, the Shannon entropy, regardless of our bit representation. In the worst case, this becomes $\Omega(\log k)$. This gives us a strict lower bound on how we can even represent a category, because we need at least $\log k$ bits.

Recall, the Shannon entropy of a random variable $\mathcal{X} \in \{x_1, ..., x_n\}$ is just $H(\mathcal{X}) = -\sum_{i=0}^{n} p(x_i) \log p(x_i) = -\mathbb{E}(\log p(x_i))$.

If we can come close to this, we can't really do much better.

Structural Requirements

So we want:

- $O(\log k)$ classification time

Structural Requirements

So we want:

- $O(\log k)$ classification time
- supports an extremely large number of categories

Structural Requirements

So we want:

- $O(\log k)$ classification time
- supports an extremely large number of categories
- some types of misclassifications are worse than others (there must be some notion of distance between categories)

Structural Requirements

So we want:

- $O(\log k)$ classification time
- supports an extremely large number of categories
- some types of misclassifications are worse than others (there must be some notion of distance between categories)

Structural Requirements

So we want:

- $O(\log k)$ classification time
- supports an extremely large number of categories
- some types of misclassifications are worse than others (there must be some notion of distance between categories)

What kind of structure should we use?

Structural Requirements

So we want:

- $O(\log k)$ classification time
- supports an extremely large number of categories
- some types of misclassifications are worse than others (there must be some notion of distance between categories)

What kind of structure should we use?

(Binary?) Trees

Framework

Overview

We will employ a binary classification tree. Assign each category (intelligently) to a leaf node. At each node, have a function which classifies items to either child node.

Overview

We will employ a binary classification tree. Assign each category (intelligently) to a leaf node. At each node, have a function which classifies items to either child node.

To classify an item, start at the root, keep calling the classifier at the current node and traveling that way until we reach a leaf. Run some kind of standard classification with the categories assigned to each leaf.

Overview

We will employ a binary classification tree. Assign each category (intelligently) to a leaf node. At each node, have a function which classifies items to either child node.

To classify an item, start at the root, keep calling the classifier at the current node and traveling that way until we reach a leaf. Run some kind of standard classification with the categories assigned to each leaf.

For each non-leaf node n, we train a binary classifier $h_n : X \to \{-1, 1\}$, such that $h_n(x) = 1$ if x is sent to the right subtree, and -1 for the left.

Overview

We will employ a binary classification tree. Assign each category (intelligently) to a leaf node. At each node, have a function which classifies items to either child node.

To classify an item, start at the root, keep calling the classifier at the current node and traveling that way until we reach a leaf. Run some kind of standard classification with the categories assigned to each leaf.

For each non-leaf node n, we train a binary classifier $h_n : X \to \{-1, 1\}$, such that $h_n(x) = 1$ if x is sent to the right subtree, and -1 for the left.

When we get to a leaf, take the label with highest training frequency at that leaf.

Objective

Want to maximize 2 quantities

1. Purity: At each node, almost all instances of a given class should be sent to the same side.
2. Balance: To keep desired classification time, need tree to be relatively balanced.

Objective

Want to maximize 2 quantities

1. Purity: At each node, almost all instances of a given class should be sent to the same side.
2. Balance: To keep desired classification time, need tree to be relatively balanced.

We want an objective function that is maximized or minimized when our tree is both pure and balanced. Shannon entropy does this...

Objective

However, we probably want our model to be able to train online, which we cannot do with entropy, so we pick something simpler.

Objective

However, we probably want our model to be able to train online, which we cannot do with entropy, so we pick something simpler.

We want to minimize the probability that at a given node, examples are split between the left and right side. We can see this reflected in the quantity

$$|P(h(x) > 0) - P(h(x) > 0|i)|$$

where $P(h(x) > 0|i)$ is the conditional probability over instances x of class i.

Objective

However, we probably want our model to be able to train online, which we cannot do with entropy, so we pick something simpler.

We want to minimize the probability that at a given node, examples are split between the left and right side. We can see this reflected in the quantity

$$0 \leq |P(h(x) > 0) - P(h(x) > 0|i)| \leq \frac{1}{2}$$

where $P(h(x) > 0|i)$ is the conditional probability over instances x of class i.

Objective

For some node n, define

$$\pi_i = \frac{\text{number of examples at node } n \text{ with label } i}{\text{number of examples at node } n}$$

Objective

For some node n, define

$$\pi_i = \frac{\text{number of examples at node } n \text{ with label } i}{\text{number of examples at node } n}$$

We can then define our objective function as

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left|P(h_n(x) > 0) - P(h_n(x) > 0|i)\right| \tag{1}$$

Example

Recall the objective function

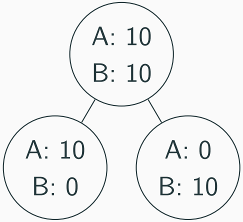
$$J(h) = 2 \sum_{i=1}^{k} \pi_i \, |P(h_n(x) > 0) - P(h_n(x) > 0|i)|$$

Example

Recall the objective function

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left|P(h_n(x) > 0) - P(h_n(x) > 0|i)\right|$$

Pure and Balanced

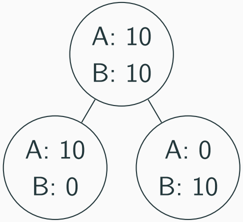


Example

Recall the objective function

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left|P(h_n(x) > 0) - P(h_n(x) > 0|i)\right|$$

Pure and Balanced



$$\pi_A = \pi_B = \tfrac{1}{2}$$

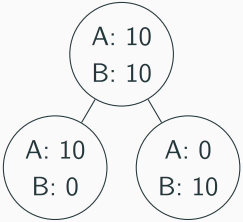
$$J(h) = 2(\tfrac{1}{2}|\tfrac{1}{2} - 0| + \tfrac{1}{2}|\tfrac{1}{2} - 1|) = 1$$

Example

Recall the objective function

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left|P(h_n(x) > 0) - P(h_n(x) > 0|i)\right|$$

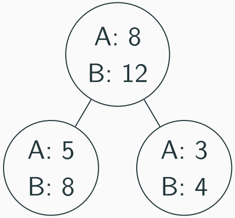
Pure and Balanced



$\pi_A = \pi_B = \frac{1}{2}$

$J(h) = 2(\frac{1}{2}|\frac{1}{2} - 0| + \frac{1}{2}|\frac{1}{2} - 1|) = 1$

Less Pure and Less Balanced

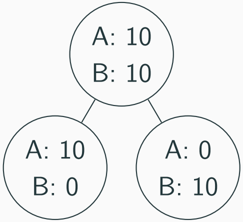


Example

Recall the objective function

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left|P(h_n(x) > 0) - P(h_n(x) > 0|i)\right|$$

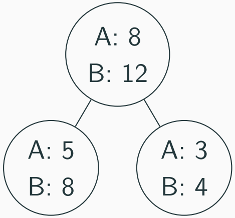
Pure and Balanced



$$\pi_A = \pi_B = \tfrac{1}{2}$$

$$J(h) = 2(\tfrac{1}{2}|\tfrac{1}{2} - 0| + \tfrac{1}{2}|\tfrac{1}{2} - 1|) = 1$$

Less Pure and Less Balanced



$$\pi_A = \tfrac{2}{5},\ \pi_B = \tfrac{3}{5}$$

$$J(h) = 2(\tfrac{2}{5}|\tfrac{13}{20} - \tfrac{5}{8}| + \tfrac{1}{2}|\tfrac{2}{5} - \tfrac{2}{3}|) = \tfrac{43}{150}$$

Metrics

Define the following quantities

$$\alpha := \sum_{i=1}^{k} \pi_i \min(P(h(x) > 0|i), P(h(x) < 0|i))$$

We say the partition function h is δ-pure if $\alpha \leq \delta$. We say the partition function h is maximally-pure if $\alpha = 0$.

Metrics

Define the following quantities

$$\alpha := \sum_{i=1}^{k} \pi_i \min(P(h(x) > 0|i), P(h(x) < 0|i))$$

We say the partition function h is δ-pure if $\alpha \leq \delta$. We say the partition function h is maximally-pure if $\alpha = 0$.

$$\beta := P(h(x) > 0)$$

We say the partition function h is c-balanced if $c \leq \beta \leq 1 - c$. We say the partition function h is maximally-balanced if $\beta = \frac{1}{2}$.

Intermediate Results

Lemma (1)

For any training set $\{(x, y)\}$ *and partition function* h

$$\alpha \leq \min\left(\frac{2 - J(h)}{4\beta} - \beta, \frac{1}{2}\right)$$

Intermediate Results

Lemma (1)

For any training set $\{(x, y)\}$ *and partition function* h

$$\alpha \leq \min\left(\frac{2 - J(h)}{4\beta} - \beta, \frac{1}{2}\right)$$

Lemma (2)

For any partition function h

$$0 \leq J(h) \leq 1$$

Specifically, $J(h) = 1$ *when* h *is maximally-pure and maximally-balanced.*

Global Properties

We now have some tools to measure the quality of the tree at each node, but we still need to train globally. First need to define our tree $\mathcal{T}$.

- $\mathcal{L}$ - the set of leaf nodes of $\mathcal{T}$
- t - number of internal (non-leaf) nodes of $\mathcal{T}$
- $\mathcal{X}$ - the input space
- $\mathcal{P}$ - the true distribution over $\mathcal{X}$ from which our training examples x are drawn
- $\pi_{i,l}$ - the conditional probability over $x \sim \mathcal{P}$ in class i reaching leaf l
- w_l - the weight of leaf l, the probability over $x \sim \mathcal{P}$ of reaching leaf l. $\sum_{l \in \mathcal{L}} w_l = 1$

Quality and Entropy

- $\pi_{i,l}$ - the conditional probability over $x \sim \mathcal{P}$ in class i reaching leaf l
- w_l - the weight of leaf l, the probability over $x \sim \mathcal{P}$ of reaching leaf l. $\sum_{l \in \mathcal{L}} w_l = 1$

We use the *quality* of the tree as a measure of the overall balance and purity of the nodes. We define it as the sum of the local entropies at each node. The quality of the tree $\mathcal{T}$ is

$$G_t := \sum_{l \in \mathcal{L}} w_l \sum_{i=1}^{k} \pi_{l,i} \ln \left(\frac{1}{\pi_{l,i}} \right)$$

LOMTree Algorithm

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left| P(h_n(x) > 0) - P(h_n(x) > 0|i) \right|$$

We can reformulate our objective function as an expectation value

$$J(h) = 2\mathbb{E}_i \left[\left| \mathbb{E}_x[\mathbb{1}(h(x) > 0)] - \mathbb{E}_x[\mathbb{1}(h(x) > 0|i)] \right| \right]$$

Now, to update our model online, we update the expectation value of each node as we encounter new training examples.

As the number of examples reaching each leaf nodes grows above some threshold, we add children to the node and assign the examples to these children. We stop when the number of internal nodes reaches some threshold T.

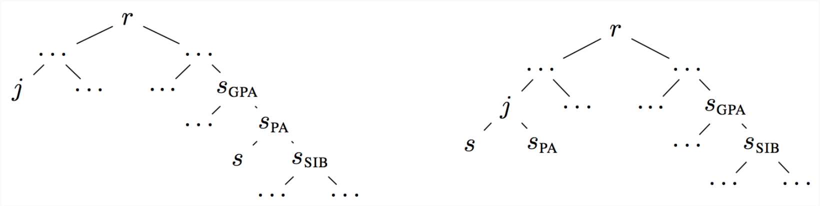
Swapping

As the tree gets unbalanced, we are likely to lose the desired runtime. However, we can perform swaps to nodes that meet the condition

$$C_j - \max_{0 \le i \le k} l_j(i) > R_S(C_r + 1)$$

where C_j is the size of the smallest leaf in the subtree rooted at j, $l_j(i)$ is the number of points of class i reaching j, and R_S is the "swap resistance" we choose. This ensures nodes with high purity are not split in an attempt to attain more balance. .

We then retrain those two branches of the tree.



Correctness

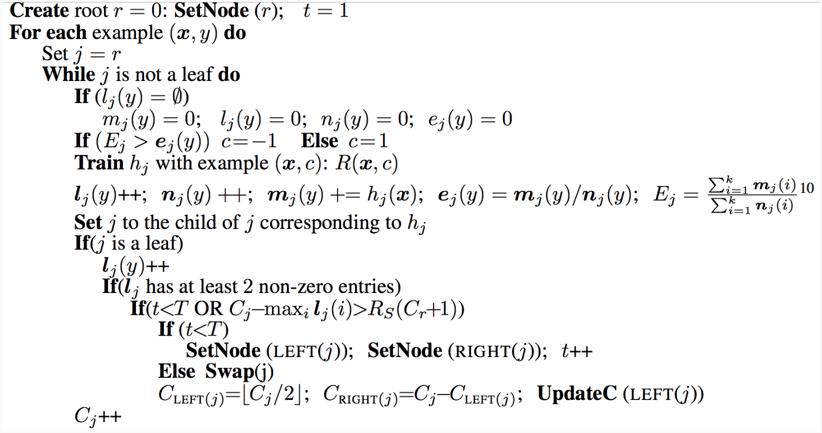
Convergence and correctness aren't necessarily guaranteed, unless we make some "Weak Hypothesis Assumption" about our data and our tree. Is basically a formalization of the condition that for any distribution $\mathcal{P}$ over $\mathcal{X}$ at each node m, there is some h_m that brings our objective function above some threshold.

Correctness

Convergence and correctness aren't necessarily guaranteed, unless we make some "Weak Hypothesis Assumption" about our data and our tree. Is basically a formalization of the condition that for any distribution $\mathcal{P}$ over $\mathcal{X}$ at each node m, there is some h_m that brings our objective function above some threshold.

Under this assumption, we can use our "Weak Learning Framework", and ensure that our tree will satisfy both "weak purity" and be "weakly balanced". These proofs are extremely technical and not instructive, but can be found in the supplemental section of the paper on LOMTree linked in the references.

Pseudocode



```
Create root $r = 0$: SetNode $(r)$;   $t = 1$
For each example $(\boldsymbol{x}, y)$ do
    Set $j = r$
    While $j$ is not a leaf do
        If $(l_j(y) = \emptyset)$
            $m_j(y) = 0$;  $l_j(y) = 0$;  $n_j(y) = 0$;  $e_j(y) = 0$
        If $(E_j > \boldsymbol{e}_j(y))$  $c{=}{-1}$   Else  $c{=}1$
        Train $h_j$ with example $(\boldsymbol{x}, c)$: $R(\boldsymbol{x}, c)$
        $\boldsymbol{l}_j(y)$++;  $\boldsymbol{n}_j(y)$ ++;  $\boldsymbol{m}_j(y) \mathrel{+}= h_j(\boldsymbol{x})$;  $\boldsymbol{e}_j(y) = \boldsymbol{m}_j(y)/\boldsymbol{n}_j(y)$;  $E_j = \frac{\sum_{i=1}^{k} \boldsymbol{m}_j(i)}{\sum_{i=1}^{k} \boldsymbol{n}_j(i)}$ 10
        Set $j$ to the child of $j$ corresponding to $h_j$
        If($j$ is a leaf)
            $\boldsymbol{l}_j(y)$++
            If($\boldsymbol{l}_j$ has at least 2 non-zero entries)
                If($t{<}T$ OR $C_j{-}\max_i \boldsymbol{l}_j(i){>}R_S(C_r{+}1)$)
                    If $(t{<}T)$
                        SetNode (LEFT($j$));  SetNode (RIGHT($j$));  $t$++
                    Else  Swap(j)
                    $C_{\text{LEFT}(j)}{=}\lfloor C_j/2 \rfloor$;  $C_{\text{RIGHT}(j)}{=}C_j{-}C_{\text{LEFT}(j)}$;  UpdateC (LEFT($j$))
        $C_j$++
```

This works under the weak learning hypothesis that we can always find a good partition function at each node.

Hierarchical Document Categorization with Support Vector Machines

Motivation

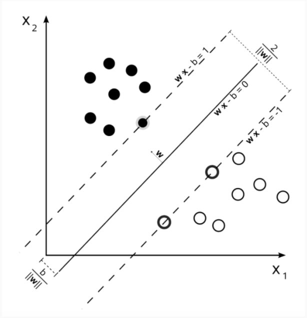
Previous algorithm looked at a binary tree structure for laying out classes. What if we want to represent classes in some arbitrary latice?

We need to encode some sort of relationship between classes.

SVMs for Classification

SVMs are robust classifiers.

Standard SVM is a binary linear classifier that maximizes minimum distance from a separating boundary.

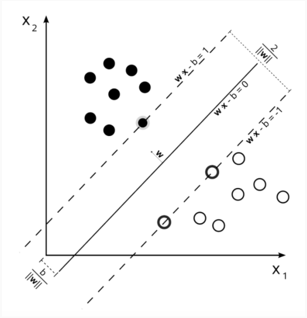


x_2
x_1
$\mathbf{w}\cdot\mathbf{x} - b = 1$
$\mathbf{w}\cdot\mathbf{x} - b = 0$
$\mathbf{w}\cdot\mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

SVMs for Classification

SVMs are robust classifiers.

Standard SVM is a binary linear classifier that maximizes minimum distance from a separating boundary.



x_2
x_1
$\mathbf{w}\cdot\mathbf{x} - b = 1$
$\mathbf{w}\cdot\mathbf{x} - b = 0$
$\mathbf{w}\cdot\mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Can be hard margin or soft margin

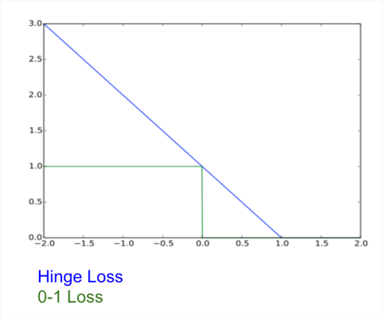
Use different kernels by mapping inputs into a high dimensional space (kernel trick)

Hinge Loss

For soft margin SVM, use hinge loss function:

$$\mathbf{max}(0, 1 - y_i(\mathbf{wx_i} - b))$$

Consider labels y_i that are ± 1. Impose a small penalty for correct classification with small margin ($\mathbf{wx_i} - b < 1$), larger penalty for misclassification.



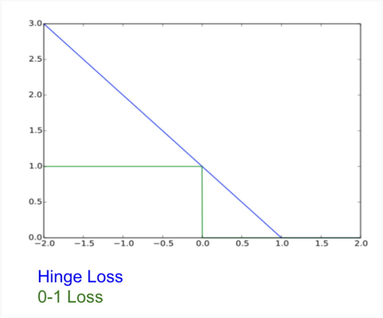
Optimization Problem

Training objective: minimize hinge loss over training set

Minimize

$$\left[\frac{1}{n}\sum_{i=1}^{n}\mathbf{max}(0, 1 - y_i(\mathbf{w}\mathbf{x_i} - b))\right] + \lambda||\mathbf{w}||^2$$

Regularization term allows trade off between increasing margin size and ensuring accurate classification



Optimization Formulation

For each $i \in 1, ..., n$, let $\gamma_i = \mathbf{max}(0, 1 - y_i(\mathbf{wx_i} - b))$.

Note that γ_i is smallest non-negative number such that

$$y_i(\mathbf{wx_i} - b) > 1 - \gamma_i$$

Thus, we can formulate this optimization as the following primal problem: minimize

$$\frac{1}{n}\sum_{i=1}^{n}\gamma_i + \lambda||\mathbf{w}||^2$$

subject to $y_i(\mathbf{wx_i} - b)) \geq 1 - \gamma_i$
and $\gamma_i \geq 0$ for all i.

We can also convert this into its Lagrangian Dual, that can be solved efficiently using quadratic programming.

Multiclass Classification

We extend Binary SVMs to discriminate between more than one class.
Assign some scalar label to each feature.

Multiclass Classification

We extend Binary SVMs to discriminate between more than one class. Assign some scalar label to each feature.

We use a flat category structure, and define a set of binary classifiers that individually distinguish each class from all the others.

Multi-class SVM

We use a separate weight vector for each of the q possible categories.

Multi-class SVM

We use a separate weight vector for each of the q possible categories.

For classification, we want to find the category y that maximizes the inner product of $\langle \mathbf{w}_y, \mathbf{x}_i \rangle$ for each training instance $(\mathbf{x_i}, y)$ that was assigned class y in our training set.

Multi-class SVM

We use a separate weight vector for each of the q possible categories.

For classification, we want to find the category y that maximizes the inner product of $\langle \mathbf{w}_y, \mathbf{x}_i \rangle$ for each training instance $(\mathbf{x_i}, y)$ that was assigned class y in our training set.

Our goal is to classify any data point $(\mathbf{x_i}, y_i)_{i=1}^{n}$ with the maximum margin as one of the q classes.

Multi-class SVM Formulation

Combine the $\mathbf{w}_y$ into stacked weight vector $\mathbf{w} = (\mathbf{w_1}, ...\mathbf{w_q})$

We then define the linear discriminant function $F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w_y}, \mathbf{x} \rangle$ and the classification function: $f(\mathbf{x}; \mathbf{w}) = \text{argmax}_{y \in Y} F(\mathbf{x}, y; \mathbf{w})$.

Winner take all: each example is assigned the class that scores the highest under the discriminant function

Multi-class SVM Formulation

The multi-class margin is given by $\gamma_i(\mathbf{w}) = F(\mathbf{x_i}, y_i) - max_{y \neq y_i} F(\mathbf{x_i}, y)$ with respect to a given weight vector $\mathbf{w}$.

We need a positive margin for correct classification, so we apply the max-margin principle for the optimal weight vector to maximize the margin defined above.

$$\mathbf{w}^* = \underset{\mathbf{w}:||\mathbf{w}||=1}{\operatorname{argmax}} \left(\min_{1 \leq i \leq n} \gamma_i(\mathbf{w}) \right)$$

Multi-class SVM Formulation

$$\min_{w,\zeta} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\zeta_i$$

such that $\gamma_i(\mathbf{w}) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$ where

$$\gamma_i(\mathbf{w}) = F(\mathbf{x_i}, y_i) - \max_{y \neq y_i} F(\mathbf{x_i}, y)$$

and $F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w_y}, \mathbf{x} \rangle$

Multi-class SVM Formulation

$$\min_{w,\zeta} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\zeta_i$$

such that $\gamma_i(\mathbf{w}) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$ where

$$\gamma_i(\mathbf{w}) = F(\mathbf{x_i}, y_i) - \max_{y \neq y_i} F(\mathbf{x_i}, y)$$

and $F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w_y}, \mathbf{x} \rangle$

Nonlinear constraints can be decomposed to $q-1$ linear constraints as follows for $y \neq y_i$:

$$\langle \mathbf{w}_{y_i} - \mathbf{w}_y, \mathbf{x_i} \rangle \geq 1 - \zeta_i$$

Convex quadratic program with nq constraints.

Representing Class Structure

Using multiclass SVM formulation now we can now consider problems with many possible classes

We still cannot encode relationship between classes...

- Each class so far is just a scalar value

Representing Class Structure

Using multiclass SVM formulation now we can now consider problems with many possible classes

We still cannot encode relationship between classes...

- Each class so far is just a scalar value
- Need to add some notion of **structure** to indicate class relationships

Representing Class Structure

Using multiclass SVM formulation now we can now consider problems with many possible classes

We still cannot encode relationship between classes...

- Each class so far is just a scalar value
- Need to add some notion of **structure** to indicate class relationships

We arrange our classes in an s node lattice, where each node can represent a class or a relationship between classes.

Class Representations

What if we are trying to classify something within a taxonomic structure? We first need to represent the relationships between classes.

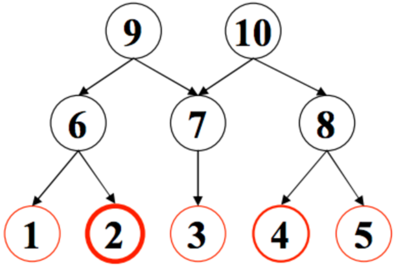
Represent each class with attribute vector $\Lambda(y)$

We can then define more general discriminant function:

$$F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w_y}, \mathbf{x} \rangle \;\rightarrow\; F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w}, \phi(\mathbf{x}, \mathbf{y}) \rangle$$

where $\phi(x, y) = \Lambda(y) \otimes \mathbf{x}$.

Structured Example



$$\Lambda(2) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ \mathbf{x} \\ 0 \\ 0 \\ 0 \\ \mathbf{x} \\ 0 \\ 0 \\ \mathbf{x} \\ 0 \end{pmatrix} = \Phi(\mathbf{x}, 2)$$

$$\langle \mathbf{w}, \Phi(\mathbf{x}, 2) \rangle = \langle \mathbf{w}_2, \mathbf{x} \rangle + \langle \mathbf{w}_6, \mathbf{x} \rangle + \langle \mathbf{w}_9, \mathbf{x} \rangle$$

Encoding Class Relationships

We can write out $\phi(x, y)$ for $F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w}, \phi(\mathbf{x}, \mathbf{y}) \rangle$ for s attribute vectors as:

$$\phi(\mathbf{x}, y) = \begin{bmatrix} \lambda_1 \cdot \mathbf{x} \\ \lambda_2 \cdot \mathbf{x} \\ \vdots \\ \lambda_s \cdot \mathbf{x} \end{bmatrix}$$

Encoding Class Relationships

We can write out $\phi(x, y)$ for $F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w}, \phi(\mathbf{x}, \mathbf{y}) \rangle$ for s attribute vectors as:

$$\phi(\mathbf{x}, y) = \begin{bmatrix} \lambda_1 \cdot \mathbf{x} \\ \lambda_2 \cdot \mathbf{x} \\ \vdots \\ \lambda_s \cdot \mathbf{x} \end{bmatrix}$$

Reduces to multiclass SVM formulation if $\lambda_r(y) = \delta_{ry}$ because then you just get

$$\langle \mathbf{w}, \phi(\mathbf{x}, \mathbf{y}) \rangle \ = \langle \mathbf{w}_{\mathbf{y}}, \mathbf{x} \rangle$$

$$\phi(\mathbf{x}, y) = \begin{bmatrix} \vdots \\ 0 \\ \mathbf{x} \\ 0 \\ \vdots \end{bmatrix}$$

New Quadratic Program Formulation

Can rewrite $F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w}, \phi(\mathbf{x}, \mathbf{y}) \rangle$ as $F(\mathbf{x}, y; \mathbf{w}) = \sum_{r=1}^{s} \lambda_r(y) \langle \mathbf{w_r}, \mathbf{x} \rangle$ by linearity over our s nodes.

Thus, we can formulate quadratic program as before with linear constraints:

$$\langle \delta\phi_i(y), \mathbf{w} \rangle \geq 1 - \zeta_i \quad (\forall i, y \neq y_i)$$
$$\zeta_i \geq 0 \quad (\forall i)$$
$$\delta\phi_i(y) \equiv \phi(\mathbf{x_i}, \mathbf{y_i}) - \phi(\mathbf{x_i}, y)$$

Dual Problem

This yields a dual problem given as

$$\alpha^* \equiv \underset{\alpha}{\operatorname{argmax}}\, \Theta(\alpha)$$

$$\text{s.t. } \alpha_{iy} \geq 0, \sum_{y \neq y_i} \alpha_{iy} < C$$

which can be solved with quadratic programming.

Defining Class Relationships via Taxonomy

Define taxonomy as some tree with leaves corresponding to categories. (Interior nodes can also represent categories by adding a leaf to them.)

Denote nodes by $z \in Z = \{z_1, ...z_p\}$ with $p \geq q$ where $y_k = z_k$ for $k = 1, ...q$ for q total categories.

Define class attributes for non-negative weights $v_z \geq 0$ as follows:

$$\lambda_z(y) = \begin{cases} v_z & \text{if } z \prec y \\ 0 & \text{otherwise} \end{cases}$$

where $z \prec y$ means that y is a child of z.

Defining Class Relationships via Taxonomy

We could set $v_z = 1$ to make λ_z an indicator function. We could also set v_z to be constant for any nodes at same depth in the tree.

Defining class attributes through common predecessors leads to a decomposition of the discriminant function into contributions from the nodes along the path from the root to a specific leaf.

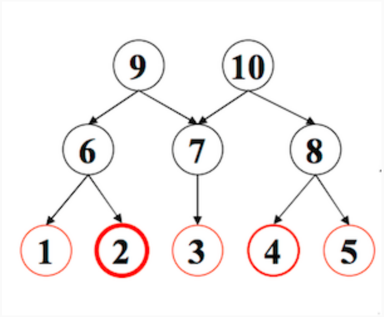
This sort of taxonomy based class structure gives the following discriminant function:

$$F(\mathbf{x}, y; \mathbf{w}) = \sum_{z : z \prec y} \lambda_z(y) \langle \mathbf{w_z}, \mathbf{x} \rangle$$

Defining The Loss Function

One issue with the current approach is it is still based on the standard SVM hinge loss. Hinge loss provides an upper bound on the empirical misclassification rate, but it treats all classification swaps equivalently.

But, when classifying in a taxonomy structure, not all mistakes are the same...



Defining The Loss Function

We need some loss function that penalizes local swaps less than swaps with far away nodes.

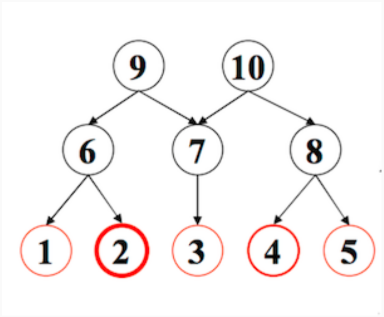
Therefore, we aim to define some function $\Delta(y, \hat{y})$ to denote the loss between the true class y and the predicted class $\hat{y}$.

Defining Loss Function

Now we need to...

- **Define a meaningful loss function between categories in a taxonomy structure**
- Modify SVM formulation to directly minimize desired loss.

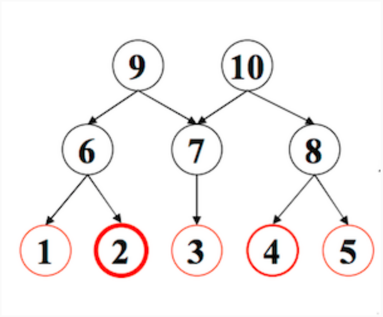
Loss Function for Document Filtering



Consider a scenario where a document is forwarded to users based on its position in the tree. Users could "subscribe" to a topic by specifying some node z of interest.

Let f_z be the "subscription load" at node z, given by the number of users that access a document categorized at or below node z in the tree.

Loss Function for Document Filtering



Total loss for node z is the sum of:

- c_z: the cost of classifying document under node z when it **SHOULD NOT** be
- $\bar{c}_z$: the cost of not classifying document under node z when it **SHOULD** be

Loss Function for Document Filtering

Now we can weight each of the two costs for each node z by the subscription load f_z for that node to get a total loss as follows:

$$\Delta(y, \hat{y}) = \sum_{z : z \prec y, z \not\prec \hat{y}} f_z c_z + \sum_{z : z \not\prec y, z \prec \hat{y}} f_z \bar{c}_z$$

Thus, we see that for a tree, the loss involves the costs for nodes on the path to the first common predecessor in the tree.

Defining Loss Function

Now we need to...

- Define a meaningful loss function between categories in a taxonomy structure
- **Modify SVM formulation to directly minimize desired loss.**

Adapted SVM Formulation

We want to generalize the vanilla SVM solution to accommodate the changed function $\Delta(y, \hat{y})$.

We can just scale penalties for margin violations proportional to loss and use the same formulation

Standard Multi-class

$$\min_{w,\zeta} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\zeta_i$$

$$\text{s.t. } \gamma_i(\mathbf{w}) \geq 1 - \zeta_i, \quad \zeta_i \geq 0, \forall i$$

Adapted Multi-class

$$\min_{w,\zeta} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\zeta_i$$

$$\text{s.t. } \langle \mathbf{w}, \delta\psi_i(y)\rangle \geq 1 - \frac{\zeta_i}{\Delta(y_i, y)}, (\forall i, y \neq y_i)$$

$$\zeta_i \geq 0, (\forall i)$$

Algorithm

We can similarly modify the dual Quadratic Problem to accommodate our new weights to get

Standard Multi-class

$$\alpha^* \equiv \underset{\alpha}{\operatorname{argmax}}\, \Theta(\alpha)$$

$$\text{s.t. } \alpha_{iy} \geq 0, \sum_{y \neq y_i} \alpha_{iy} < C$$

Adapted Multi-class

$$\alpha^* \equiv \underset{\alpha}{\operatorname{argmax}}\, \Theta(\alpha)$$

$$\text{s.t. } \sum_{y \neq y_i} \frac{\alpha_{iy}}{\Delta(y_i, y)} \leq C, (\forall i, y \neq y_i)$$

$$\alpha_{iy} \geq 0, (\forall i)$$

However, the dual problem grows as $O(nq)$ now, where q is the total number of classes, instead of as $O(n)$ as in the standard formulation. We can alleviate some of the extra training time by noting that some constraints factor and by exploiting sparsity.

Independence Condition

Note that for α_{iy} and $\alpha_{jy'}$ from different training instances i and j are not coupled at all in our optimization. Therefore, we can hold most dual variables α constant and perform subspace optimization. This gives us a linear number of subproblems, each of which we can solve sublinearly compared the the original large QP.

Variable Selection

We can also use the fact that most α_{iy} will be 0, because α will be sparse since we expect a very small amount of active constraints. We can exploit this fact by only choosing certain variables to use in our solution. The authors use a variable selection approach that capitalizes on this fact, but we skip the derivations because they provide little insight or clarity.

LOMtree Experimental Results

LOMtree Algorithm Experimental Results

Hypotheses on LOMTree Algorithm:

1. Achieves logarithmic time computation in practice
2. Competitive with or better than all other logarithmic train/test time algorithms for multi-class classification
3. LOMTree algorithm has statistical performance close to standard $O(k)$ approaches

Address these hypotheses by testing on benchmark multiclass datasets.

LOMtree Algorithm Experimental Results

All sets divided into 90 percent training and 10 percent testing. Also, 10 percent of training set used as validation set.

Dataset information shown below:

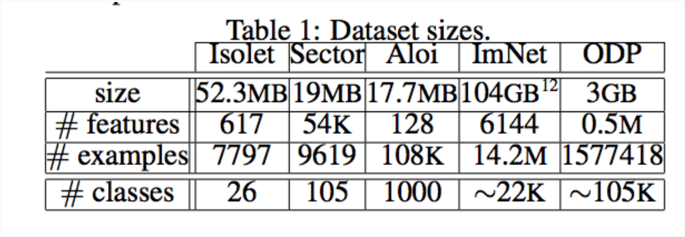


Table 1: Dataset sizes.

	Isolet	Sector	Aloi	ImNet	ODP
size	52.3MB	19MB	17.7MB	104GB[12]	3GB
# features	617	54K	128	6144	0.5M
# examples	7797	9619	108K	14.2M	1577418
# classes	26	105	1000	∼22K	∼105K

LOMtree Algorithm Experimental Results

Compared LOMTree with:

- Balanced random tree of logarithmic depth (Rtree) ($O(logk)$)
- Filter Tree ($O(logk)$)
- One Against All Classifier ($O(k)$)

All methods were implemented in some fixed learning system and trained by online gradient descent with a variety of step sizes.

For each method, ran training with up to 20 passes through the data and selected the best step size/number of passes pair for each model.

LOMtree Algorithm Experimental Results

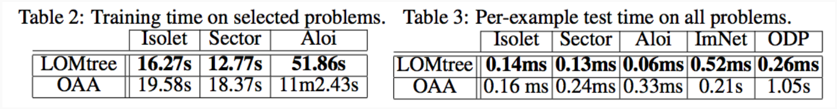


Table 2: Training time on selected problems.

	Isolet	Sector	Aloi
LOMtree	**16.27s**	**12.77s**	**51.86s**
OAA	19.58s	18.37s	11m2.43s

Table 3: Per-example test time on all problems.

	Isolet	Sector	Aloi	ImNet	ODP
LOMtree	**0.14ms**	**0.13ms**	**0.06ms**	**0.52ms**	**0.26ms**
OAA	0.16 ms	0.24ms	0.33ms	0.21s	1.05s

We see LOMTree does indeed achieve logarithmic time computation in practice (Hypothesis 1) since LOMTree performs much better than OAA since LOMTree only builds close to logarithmic depth trees

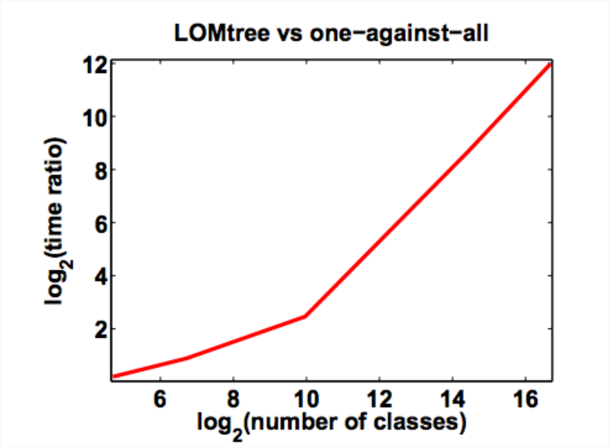
Improvement in training time only increases with increase in number of classes in problem as seen above

Improvement in test time also increases with increase in number of classes in problem.

Per example test time for Aloi, ImageNet, and ODP are respectively 5.5, 403.8, and 4038.5 times faster than OAA.

LOMtree Algorithm Experimental Results

Test time of LOMTree over OAA shown again below. Note that because of the log scale its actually an exponential speedup, as expected.



LOMtree Algorithm Experimental Results

Test error shown for logarithmic train/test time algorithms.

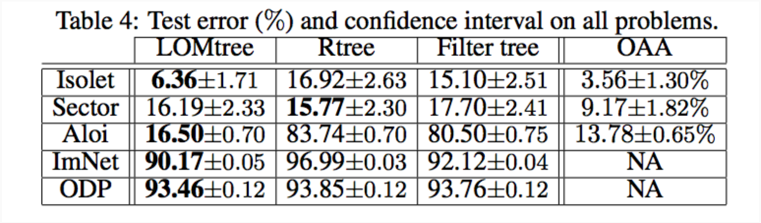


Table 4: Test error (%) and confidence interval on all problems.

	LOMtree	Rtree	Filter tree	OAA
Isolet	**6.36**±1.71	16.92±2.63	15.10±2.51	3.56±1.30%
Sector	16.19±2.33	**15.77**±2.30	17.70±2.41	9.17±1.82%
Aloi	**16.50**±0.70	83.74±0.70	80.50±0.75	13.78±0.65%
ImNet	**90.17**±0.05	96.99±0.03	92.12±0.04	NA
ODP	**93.46**±0.12	93.85±0.12	93.76±0.12	NA

Clearly the LOMtree algorithm is generally competitive with or better than all other logarithmic train/test time algorithms for multiclass classification (Hypothesis 2)

LOMtree Algorithm Experimental Results

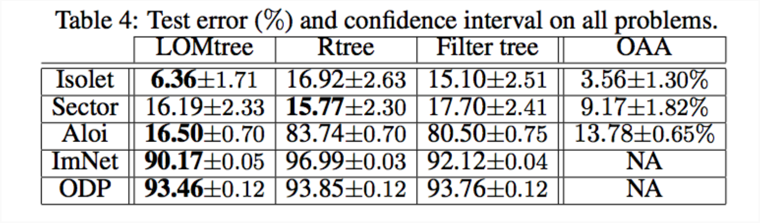


Table 4: Test error (%) and confidence interval on all problems.

	LOMtree	Rtree	Filter tree	OAA
Isolet	**6.36**±1.71	16.92±2.63	15.10±2.51	3.56±1.30%
Sector	16.19±2.33	**15.77**±2.30	17.70±2.41	9.17±1.82%
Aloi	**16.50**±0.70	83.74±0.70	80.50±0.75	13.78±0.65%
ImNet	**90.17**±0.05	96.99±0.03	92.12±0.04	NA
ODP	**93.46**±0.12	93.85±0.12	93.76±0.12	NA

RTree imposes random label partition so its error is worse than LOMTree (learns partitions from data)

LOMtree at least slightly better, and sometimes much better than all other logarithmic time algorithms on all sets other than Sector

Unclear if LOMTree has statistic performance comparable to $O(k)$ approaches since still good amount higher error than OAA

Hierarchical Document Categorization Application

The Problem

Extremely large database of documents with a specific categorization

Features generated from each document by using title and header information + tokenizing body of document

Want to correctly classify each document to a specific categorization...getting categories wrong closer to root in the taxonomy tree is expensive

Data

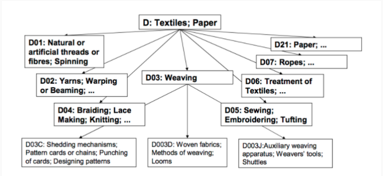
We have 2 data sets: Synthetic and WIPO-alpha

Synthetic:

- Generated tree structure with constant depth, chose fixed number of features
- Random weight vector generated for each node, following iid multinomial distributions
- Variance fixed across levels, decreases with increasing depth

WIPO-alpha

- patent documents with a 4 level hierarchy
- Sections, classes, subclasses, groups
- Eg:


D: Textiles; Paper
D01: Natural or artificial threads or fibres; Spinning
D21: Paper; ...
D07: Ropes; ...
D02: Yarns; Warping or Beaming; ...
D03: Weaving
D06: Treatment of Textiles; ...
D04: Braiding; Lace Making; Knitting; ...
D05: Sewing; Embroidering; Tufting
D03C: Shedding mechanisms; Pattern cards or chains; Punching of cards; Designing patterns
D003D: Woven fabrics; Methods of weaving; Looms
D003J:Auxiliary weaving apparatus; Weavers' tools; Shuttles

Approach

Choose loss function

$$\Delta(y, \hat{y}) = \left(\sum_{z : z \prec y, z \not\prec \hat{y}} \frac{1}{2} \right) + \left(\sum_{z : z \not\prec y, z \prec \hat{y}} \frac{1}{2} \right)$$

For computational convenience, and easy comparision to standard SVM, set $v_z = \sqrt{\frac{1}{\text{depth}}} = \text{const}$ so that $\langle \Lambda(y), \Lambda(y) \rangle = 1$ and therefore: $\langle \psi(\mathbf{x}, y), \psi(\mathbf{x}, y) \rangle = \langle \Lambda(y), \Lambda(y) \rangle \, \langle \mathbf{x}, \mathbf{x} \rangle = \langle \mathbf{x}, \mathbf{x} \rangle$

Linear kernel used

All data normalized in pre-processing

Testing accuracy determined using cross-validation and averaging

Evaluation Measures

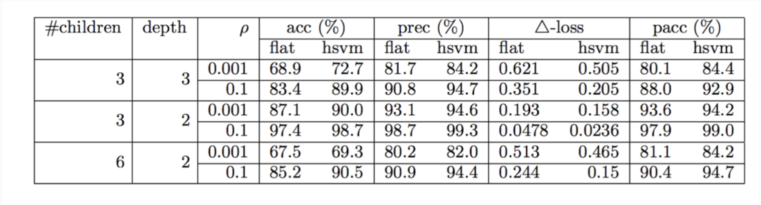
- **Accuracy:** fraction of documents classified perfectly
- **Precision:**

$$prec(f) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{|\{y : F(\mathbf{x_i}, y) \geq F(\mathbf{x_i}, y_i)\}|}$$

- **Taxonomy Loss:** $\Delta\text{-loss(f)} = \frac{1}{n}\sum_{i=1}^{n} \Delta(y_i, f(\mathbf{x_i}))$
- **Parent accuracy:** likelihood of assigning a sibling of the correct lowest classification.

Performance - Synthetic Data

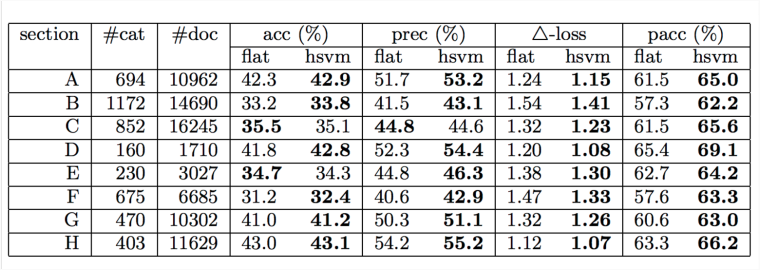
- 100 training examples, 20 features



#children	depth	ρ	acc (%)		prec (%)		$\triangle$-loss		pacc (%)	
			flat	hsvm	flat	hsvm	flat	hsvm	flat	hsvm
3	3	0.001	68.9	72.7	81.7	84.2	0.621	0.505	80.1	84.4
		0.1	83.4	89.9	90.8	94.7	0.351	0.205	88.0	92.9
3	2	0.001	87.1	90.0	93.1	94.6	0.193	0.158	93.6	94.2
		0.1	97.4	98.7	98.7	99.3	0.0478	0.0236	97.9	99.0
6	2	0.001	67.5	69.3	80.2	82.0	0.513	0.465	81.1	84.2
		0.1	85.2	90.5	90.9	94.4	0.244	0.15	90.4	94.7

- Doesn't feel that extreme though...

Performance - WIPO



section	#cat	#doc	acc (%)		prec (%)		Δ-loss		pacc (%)	
			flat	hsvm	flat	hsvm	flat	hsvm	flat	hsvm
A	694	10962	42.3	**42.9**	51.7	**53.2**	1.24	**1.15**	61.5	**65.0**
B	1172	14690	33.2	**33.8**	41.5	**43.1**	1.54	**1.41**	57.3	**62.2**
C	852	16245	**35.5**	35.1	**44.8**	44.6	1.32	**1.23**	61.5	**65.6**
D	160	1710	41.8	**42.8**	52.3	**54.4**	1.20	**1.08**	65.4	**69.1**
E	230	3027	**34.7**	34.3	44.8	**46.3**	1.38	**1.30**	62.7	**64.2**
F	675	6685	31.2	**32.4**	40.6	**42.9**	1.47	**1.33**	57.6	**63.3**
G	470	10302	41.0	**41.2**	50.3	**51.1**	1.32	**1.26**	60.6	**63.0**
H	403	11629	43.0	**43.1**	54.2	**55.2**	1.12	**1.07**	63.3	**66.2**

Conclusions

- Hierarchical SVM outperforms flat SVM in general
- Especially true when number of examples low
- Learned solutions tend to be sparse, less than 1 percent of alpha variables are nonzero

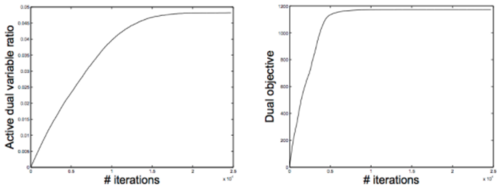


Figure 4: Optimization process of the hierarchical SVM on D section. The objective of the dual problem is defined in Eq. (16)**.**

References

1. https://info.cis.uab.edu/zhang/Spam-mining-papers/Hierarchical.Data.Classification.with.Support.Vector.Machines.pdf
2. http://www.cs.utexas.edu/~inderjit/public_papers/pdsparse_icml16.pdf
3. http://papers.nips.cc/paper/5937-logarithmic-time-online-multiclass-prediction.pdf

Questions?