Courant Institute of Mathematical Sciences

Deep Cascades with Perceptron

Ankit Vani
Srivas Venkatesh
December 20, 2015

1 Introduction

The Deep Cascades [1] algorithm is a cascade of a rich family of functions serving as the node question, leaf predictors structured as a decision tree. The algorithm exhibits data dependent guarantees that allow for use of a rich family of functions. The paper talks about using SVMs as the rich set of functions and elaborates on how the deep cascades algorithm is adapted for use with SVMs.
We look at adapting the Deep cascade algorithm to work with the Perceptron algorithm as described in [2] and it's kernelized variant [3] as the rich set of function. The perceptron algorithm learns a linear separator in an online fashion, examining one point at a time. We have however used this in a batch setting with the use of polynomial kernels.
We also show how this performs faster than an SVM based cascade while still benefiting from nature of the cascade.

2 Background on Deep Cascades, Perceptron

2.1 Deep Cascades

The paper on Deep cascades presents data dependent generalization guarantees for cascades with leaf predictors chosen from a hypothesis set H_k and node question functions selected from hypothesis sets Q_j. They present the following guarantee on the generalization error

in the simplest form:

$$R(f) \leq \widehat{R}_S(f) + \sum_{k=1}^{l} \min \left(4 \left[\sum_{j=1}^{d_k} \widehat{\mathfrak{R}}_S(Q_j) + \widehat{\mathfrak{R}}_S(H_k), \frac{m_k^+}{m} \right] \right) + \widetilde{O}\left(l\sqrt{\frac{\log pl}{m}} \right) \tag{2.1}$$

where l is the number of levels in the cascade and p is defined as the number of hypothesis sets we choose from. That is we define $\mathcal{H}$ to be the set of p hypothesis sets from which the hypothesis sets H_k are selected. Similarly each Q_k is selected out of a family of hypothesis sets $\mathcal{Q}$ of cardinality p. Since the paper considers depths of at max 4 the last term is practically insignificant and hence any cascade minimizing the first 2 terms will provide good generalization guarantees.
With these guarantees in mind the paper provides an adaptation of the algorithm for use with SVMs at each level. The topology of the cascade is as shown in Figure 2.1

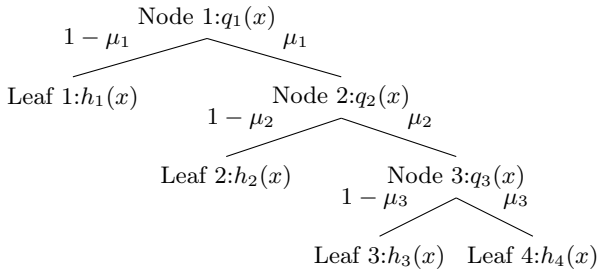

Figure 2.1: Topology of Deepcascade SVM

That is at each internal node, an SVM is trained on the incoming points. Then the closest fraction μ_k of points to the separating hyperplane are removed from the current set of points and sent to the next level for it to be trained with a new SVM with a possibly different kernel. The remainder of $1-\mu_k$ fraction of points stay classified by the same hyperplane which is treated as that particular leaf hypothesis.
The pseudocode for the DeepCascadeSVM is as follows:

Algorithm 1 Deep Cascade SVM

function DEEPCASCADESVM(L,$\mathcal{M}$,γ)
  for $l \in [1 \cdots L], (\mu_k)_{1\leq k\leq l} \subseteq \mathcal{M}, (\delta_k)_{1\leq k\leq l} \subseteq \mathcal{G}$ **do**
    $S_1 \leftarrow S$
    for $k = 1$ to l **do**
      $h_k \leftarrow \text{SVM}(\delta_k, S_k)$
      $q_k \leftarrow \arg_{q\in Q_k}\{|q^{-1}(1) \cap S_k| = \mu_k |S_k|\}$
      $S_{k+1} \leftarrow q_k^{-1}(1) \cap S_k$
    end for
    $f(\cdot) \leftarrow \sum_{k=1}^{l-1} \left(\prod_{j=1}^{k-1} q_j \right) \bar{q_k} h_k + \left(\prod_{j=1}^{l} q_j \right) h_l$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$
  end for
  $f^* \leftarrow \text{argmin}_{f\in\mathcal{F}} \widehat{R}_s(f) + \sum_{k=1}^{l} \min \left(4\gamma \left[\sum_{j=1}^{d_k} \sqrt{\frac{d_{f,j} \log \frac{em}{d_{f,j}}}{m}} + \sqrt{\frac{d_{f,k} \log \frac{em}{d_{f,k}}}{m}} \right], \frac{m_k^+}{m} \right)$

13: **end function**

The deep cascade functions generated by the algorithm are based on repeatedly using SVMs combined with polynomial kernels of different degree. The leaf hypothesis sets H_k are decision surfaces defined by polynomial kernels. The hypothesis $h_k \in H_k$ is learned via the SVM algorithm with a polynomial kernel degree δ_k on subsample S_k . The node question hypothesis set Q_k is the set of indicator functions which indicate if a point is within a distance c from the hyperplane such that a $\mu_k|S_k|$ points lie within that distance c from the hyperplane.
Hence a search is done over all depths $[1 \cdots L]$, fraction of points $\mu_k \in \mathcal{M}$ and degree of kernels $\delta_k \in \mathcal{G}$ such that the resulting cascade minimizes the generalization error. We also see that the rademacher complexities from (2.1) have been bounded by the VC dimension of the respective SVM $(d_{f,j}, d_{f,k})$ and that a factor γ has been introduced to scale the hinge loss. Thus the DeepCascadeSVM only tries to get the minimal upper bound in terms of the VC dimension rather than trying to minimize the rademacher complexities.

2.1.1 Perceptron Algorithm

The perceptron algorithm is one of a broad family of online learning algorithms and has quite a few variants. We are looking at the kernelized perceptron variant of the same. The original perceptron algorithm is an "error driven" algorithm that trains on the dataset to learn a separating linear homogeneous threshold function given as $w.x = 0$. Each time a mistake is made on point x_i, w is adjusted by $x_i y_i$.
In the case of the kernelized perceptron we actually need to learn a hyperplane in the kernel hilbert space for which we first need to identify the dual formulation. This is done by observing that w can be expressed as linear combination of the training samples. The equation being $w = \sum_{i=1}^{m} \alpha_i y_i x_i$. Then we can see that adjusting with $y_i x_i$ in the primal form is the same as adjusting the α_i by 1. Now we see that the predicted $\widehat{y_i}$ is given using $sgn(w \cdot x_i)$. Writing this with the dual we see that:

$$\begin{aligned} \widehat{y_t} &= sgn(w^T x_t) \\ &= sgn\left(\sum_{i=1}^{m} \alpha_i y_i x_i\right)^T x_t \\ &= sgn \sum_{i=1}^{m} \alpha_i y_i (x_i \cdot x_t) \end{aligned} \tag{2.2}$$

Since it has come to a dot product form we can now use kernels to evaluate the same. This is the principle behind kernelized perceptron. Also this can be adapted to a batch setting by simply running the algorithm over all the points and repeating for a few passes if needed. The psuedocode is as follows:

Algorithm 2 Batch Kernelized Perceptron

1: **function** KernelizedPerceptron(S, K) ▷ Where S is the sample set and K is the kernel
2: $\alpha \leftarrow \alpha^0$ ▷ Where α^0 is the α corresponding to the initial w,typically 0
3: **for** $p = 1$ to *passes* **do**
4: **for** $t = 1$ to T **do**

Receive($x_t \in S$)
$\widehat{y}_t \leftarrow sgn \sum_{i=1}^{m} \alpha_i y_i K(x_i, x_t)$
Receive($y_t \in S$)
if $\widehat{y}_t \neq y_t$ **then**
$\alpha_t \leftarrow \alpha_t + 1$
end if
end for
end for
return α
end function

The point to be noted with regards to perceptrons is the fact that training time for this is linear in the number of samples.

3 Algorithm

We have adapted the deep cascades to work with kernelized perceptrons serving as the leaf hypothesis and node questions. The adaptation is similar to the DeepCascadeSVM with a few subtle differences.

Algorithm 3 Deep Cascade Kernelized Perceptron

function DeepCascadePerceptron($S, L, \mathcal{M}, \mathcal{G}, \gamma$)
for $l \in [1 \cdots L], (\mu_k)_{1 \leq k \leq l} \subseteq \mathcal{M}, (\delta_k)_{1 \leq k \leq l} \subseteq \mathcal{G}$ **do**
$S_1 \leftarrow S$
for $k = 1$ to l **do**
$h_k \leftarrow$ KernelizedPerceptron($S_k, poly\_kernel(\delta_k)$)
$threshold \leftarrow get\_distance(S_k, h_k, \mu_k, poly\_kernel(\delta_k))$ ▷ Gets the distance of μ_k fraction of closest point from hyperplane
$q_k \leftarrow 1_{dist(x,h_k) \leq threshold}$
$S_{k+1} \leftarrow q_k^{-1}(1) \cap S_k$
end for
$f(\cdot) \leftarrow \sum_{k=1}^{l-1} \left(\prod_{j=1}^{k-1} q_j \right) \bar{q_k} h_k + \left(\prod_{j=1}^{l} q_j \right) h_l$
$\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$
end for
$f^* \leftarrow \text{argmin}_{f \in \mathcal{F}} \widehat{R}_s(f) + \sum_{k=1}^{l} \min \left(4\gamma \left[\sum_{j=1}^{d_k} \sqrt{\frac{d_{f,j} \log \frac{em}{d_{f,j}}}{m}} + \sqrt{\frac{d_{f,k} \log \frac{em}{d_{f,k}}}{m}} \right], \frac{m_k^+}{m} \right)$
end function

The sample set is standardized with a standard scaling before being sent to the algorithm. This is done to center the data and normalize it. The same scaling is applied to the test data as well. This centering,scaling becomes quite important in the case of perceptrons as it learns only a linear homogeneous threshold function.
The algorithm essentially consists of training perceptrons h_k at each level (total of l levels) with the incoming points using polynomial kernels of dimension δ_k. Once the perceptron at level k is trained on S_k, the distances to all points are measured as $w_k.x = \sum_{i=1}^{|S_k|} \alpha_i y_i K(x_i, x_t)$ (ignoring the factor of $\|w_k\|$ as we just want to get the μ_k fraction of closest points). With

the distances calculated we decide on a threshold as the distance from the hyperplane such that $\mu_k|S_k|$ points lie within that distance. The node question is then: if a given point lies within this threshold from the trained hyperplane. This process is then repeated for each level.
The algorithm generates multiple cascades for each of the depth $l \in [1 \cdots L]$, any sequence of fraction values $(\mu_k)_{1 \leq k \leq l} \subseteq \mathcal{M}$ and sequence of polynomial kernel degrees $(\delta_k)_{1 \leq k \leq l} \subseteq \mathcal{G}$. The node question and leaf hypothesis for these cascades are chosen as described above. Then the best cascade f^* is chosen as the one that minimizes an upper bound on the generalization error bound of (2.1) similar to what is done for the DeepCascadeSVM [1]. That is f^* is chosen to minimize the bound:

$$R(f) \leq \widehat{R}_s(f) + \sum_{k=1}^{l} \min \left(4\gamma \left[\sum_{j=1}^{d_k} \sqrt{\frac{d_{f,j} \log \frac{em}{d_{f,j}}}{m}} + \sqrt{\frac{d_{f,k} \log \frac{em}{d_{f,k}}}{m}} \right], \frac{m_k^+}{m} \right) \tag{3.1}$$

where d_k is the depth of node k and $d_{f,k}$ is the VC dimension of the perceptron trained at level k and the γ parameter is introduced to scale the loss function.
The VC dimension of a perceptron accepting samples of dimension n and working with a polynomial kernel of dimension δ is $\binom{n+\delta}{\delta}$. This can be seen by the definition of polynomial kernel: $(K(x, y) = (1 + x \cdot y)^\delta)$ and using multinomial theorem.
The γ is selected by doing a n-fold cross validation to select the best γ from a set of values.

4 Experiments

We performed some preliminary experiments on datasets from UCI machine learning repository [4].
In our experiments we used the following parameters: max cascade depth $L = 4$, the set of fraction values was chosen as $\mathcal{M} = \{0.2, 0.4, 0.6, 0.8\}$ and the degree of the kernels were chosen from $\mathcal{G} = \{1 \cdots 4\}$.
We have split our dataset into a 80% training set and 20% test set. The training set was inturn split into 3 folds and for each fold leaving that out for validation we trained on the other two. This way a 3-fold cross validation was performed to figure out the correct $\gamma \in \{0.001, 0.01, 0.1\}$.

Dataset	Number of examples	Number of features	col1	DeepCascadePerceptron	Cascade Depth

Table 4.1: Test error for DeepCascadePerceptron

5 Conclusions

References

[1] Giulia DeSalvo, Mehryar Mohri, and Umar Syed. "Learning with Deep Cascades". In: *Proceedings of the Twenty-Sixth International Conference on Algorithmic Learning Theory (ALT 2015)*. 2015.

[2] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[3] M. A. Aizerman, E. M. Braverman, and Lev I. Rozonoer. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and Remote Control* 25 (1964), p. 821.

[4] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[5] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. Adaptive computation and machine learning series. Cambridge (Mass.), London: MIT Press, 2012. ISBN: 978-0-262-01825-8. URL: `http://opac.inria.fr/record=b1134085`.