Supervised Learning

Sparsity Methods

December, 2016

John Shawe-Taylor

Today's plan

- Weak learners
- revisit Adaboost
- Properties of boosting
- L_1 sparsity and Linear Programming Boosting
- Greedy sparsity and the set cover machine
- Matching pursuit

Spam email classification problem

Consider the problem of classifying an email you receive as “good email” or “spam email”

- Different features can be used to represent an email

  e.g. the bag of words representation (we may also use additional features to code the text in the subject section of the email etc.)

- *Key observation:* it is easy to find “*rules of thumb*” that are often correct (say 60% of the time) e.g. “IF ‘business’ occurs in email THEN predict as spam”

- Hard to find highly accurate prediction rules

Weak learners and boosting

Weak learner: an algorithm which can consistently find classifiers (“rules of thumb”) at least slightly better than random guessing, say better than 55%

Boosting: a general method of converting rough rules of thumb into a highly accurate prediction rule (classifier)

Boosting algorithm

- Devise a computer program for deriving rough rules of thumb
- Choose rules of thumb to fit a subset of example
- Repeat T times
- Combine the classifiers by weighted majority vote

key steps are:

- how do we choose the subset of examples at each round? concentrate on **hardest examples** (those most often misclassified by previous classifiers)
- how do we combine the weak learners? by **weighted majority**

Adaboost algorithm

Given training data $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$
Initialize $D_1(i) = \frac{1}{m}$

For $t = 1, \ldots, T$:

- fit a classifier $h_t : \mathbf{R}^d \to \mathbf{R}$ using distribution D_t
- choose $\alpha_t \in \mathbf{R}$
- update

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t} \qquad (*)$$

where Z_t is a normalization factor (so as to ensure that D_{t+1} is a distribution)

Output the final classifier $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$

Some remarks

- The basic idea in Adaboost is to maintain a distribution D on the training set and iteratively train a weak learner on it

- At each round larger weights are assigned to hard examples, hence the weak learner will focus mostly on those examples

Let ϵ_t be the weighted training error of classifier t:

$$\epsilon_t = \sum_{i=1}^{m} D_t(i) I\{h_t(\mathbf{x}) \neq y_i\}$$

We will justify later the following choice for α_t

$$\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \qquad (1)$$

Weight by majority

$$\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$$

The final classifier is a weighted majority vote of the T base classifiers where α_t is the weight assigned to h_t

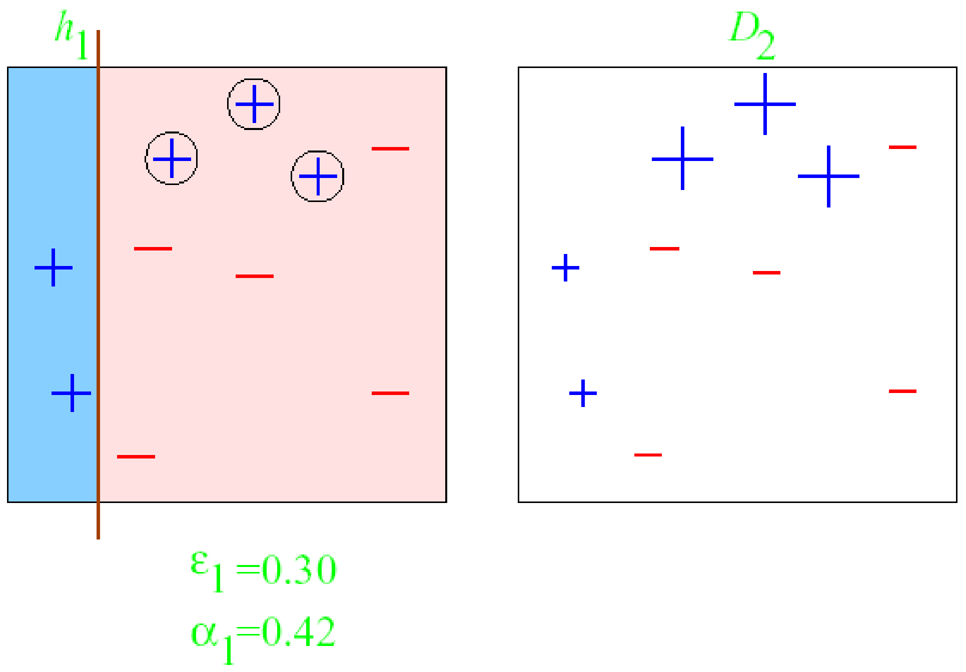
$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t$$

Typically $\epsilon_t \leq 0.5$ hence $\alpha_t \geq 0$ (this is always the case if h_t and $-h_t$ are both in the set of weak learners, e.g. for decision stumps)

Thus, f is essentially a convex combination of the h_t with weights controlled by the training error

Example (round 1)

Let's discuss a simple example where the weak learners are decision stumps (vertical or horizontal lines, i.e. trees with only two leaves)
the 2nd plot highlights the difficult examples



(Thanks to Rob Schapire for providing the figures for this example)

Analysis of the training error (I)

The training error of the boosting algorithm is bounded as

$$\frac{1}{m}\sum_{i=1}^{m} I\{H(\mathbf{x}_i) \neq y_i\} \leq \frac{1}{m}\sum_{i=1}^{m} e^{-y_i f(\mathbf{x}_i)} = \prod_{t=1}^{T} Z_t$$

where we have defined $f = \sum_t \alpha_t h_t$ (so that $H(\mathbf{x}) = \mathrm{sign}(f(\mathbf{x}))$)

The inequality follows from:

$$H(\mathbf{x}_i) \neq y_i \;\Rightarrow\; e^{-y_i f(\mathbf{x}_i)} \geq 1$$

The equality follows from the recursive definition of D_t (see also page 19)

Analysis of the training error (II)

The previous bound suggests that if at each iteration we choose α_t and h_t by minimizing Z_t the final training error of H will be reduced most rapidly

Generally we choose $h_t : \mathbf{R}^d \rightarrow \mathbf{R}$ (margin classifiers). However, if we constrain h_t to have range $\{-1, 1\}$ (so these are binary classifiers), Z_t is minimized by the choice in equation (1) above

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

We show this next

Analysis of the training error (III)

From formula (*) on page 6 we have

$$Z_t = \sum_i D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$

Using the fact that h_t returns binary values we also have that

$$\begin{aligned} Z_t &= e^{\alpha_t} \sum_{i:y_i \neq h_t(\mathbf{x}_i)} D_t(i) + e^{-\alpha_t} \sum_{i:y_i = h_t(\mathbf{x}_i)} D_t(i) \\ &= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} \end{aligned}$$

Equation (1) now follows by solving $\frac{dZ_t}{d\alpha_t} = 0$

Analysis of the training error (IV)

Placing $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$ in the above formula for Z_t we obtain

$$Z_t = \epsilon_t e^{\alpha_t} + (1-\epsilon_t)e^{-\alpha_t} = 2\sqrt{\epsilon_t(1-\epsilon_t)} = \sqrt{1-4\gamma_t^2}$$

where $\gamma_t = 1/2 - \epsilon_t$. Hence we have the following bound for the training error

$$Error \leq \prod_t Z_t = \prod_t \sqrt{1-4\gamma_t^2} \leq e^{-2\sum_t \gamma_t^2}$$

Thus, if each weak classifier is slightly better than random guessing (if $\gamma_t \geq \gamma > 0$) the training error drops **exponentially fast**

$$Error \leq e^{-2T\gamma^2}$$

Additive models and boosting

Boosting can be seen as a greedy way to solve the problem

$$\min_{\mathbf{w}} \sum_{i=1}^{m} V(y_i, \mathbf{w}^\top \phi(\mathbf{x}_i))$$

where the feature vector $\phi = (\phi_1, \phi_2, \ldots, \phi_N)$ is formed only by weak learners

At each iteration a new basis function is added to the current basis expansion $f^{(t-1)} = \sum_{s=1}^{t-1} \alpha_s h_s$

$$(\alpha_t, n(t)) = \mathrm{argmin}_{\alpha,n} \sum_{i=1}^{m} V(y_i, f^{(t-1)}(\mathbf{x}_i) + \alpha\phi_n(\mathbf{x}_i))$$

and $h_t = \phi_{n(t)}$. This is unlike in CART and MARS, where at each iteration previous basis function coefficients are re-adjusted

Additive models and boosting

$$\min_{\alpha_t, n} \sum_{i=1}^{m} V(y_i, f^{(t-1)}(\mathbf{x}_i) + \alpha_t \phi_n(\mathbf{x}_i))$$

In the statistical literature, this type of algorithm is called *forward stagewise additive model*

It is also possible to establish a relation between boosting and L^1 norm regularization

$$\min_{\mathbf{w}} \sum_{i=1}^{m} V(y_i, \mathbf{w}^\top \phi(\mathbf{x}_i)) + \lambda \sum_{n=1}^{N} |w_n|$$

which says that essentially boosting maximizes the L^1 margin $(\sum_{n=1}^{N} |w_n|)^{-1}$

Why the exponential loss?

When V is the exponential loss using formula (*) on page 6 recursively it is easy to see that

$$D_t(i) = \frac{\prod_{s=1}^{t-1} e^{-y_i \alpha_s h_s(\mathbf{x}_i)}}{m \prod_{s=1}^{t-1} Z_s} = \frac{e^{-y_i f^{(t-1)}(\mathbf{x}_i)}}{m \prod_{s=1}^{t-1} Z_s}$$

from which, summing over i, it follows that

$$D_t(i) = \frac{e^{-y_i f^{(t-1)}(\mathbf{x}_i)}}{\sum_{j=1}^{m} e^{-y_j f^{(t-1)}(\mathbf{x}_j)}}$$

Why the exponential loss? (cont.)

The formula

$$e^{-y_i\left(f^{(t-1)}(\mathbf{x}_i)+\alpha_t h_t(\mathbf{x}_i)\right)} \propto D_t(i)e^{-y_i\alpha_t h_t(\mathbf{x}_i)}$$

implies that

$$\begin{aligned}\sum_{i=1}^{m} e^{-y_i\left(f^{(t-1)}(\mathbf{x}_i)+\alpha_t h_t(\mathbf{x}_i)\right)} &\propto \sum_{i=1}^{m} D_t(i)e^{-y_i\alpha_t h_t(\mathbf{x}_i)} \\ &= (e^{\alpha_t} - e^{-\alpha_t})\epsilon_t(h_t) + e^{-\alpha_t}\end{aligned}$$

where as before $\epsilon_t(h_t) = \sum_{i=1}^{m} D_t(i)I\{h_t(\mathbf{x}_i) \neq y_i\}$

Hence, for a fixed value of $\alpha_t > 0$, minimizing the exponential loss w.r.t. h_t is the same as minimizing w.r.t. the weighted misclassification error!

Summarizing

In summary, Adaboost can be interpreted as a greedy way to minimize the exponential loss criterion via the forward-stagewise additive model approach

Alternatively, let $\{h_n\}_{n=1}^N$ the set of all weak learners (assume they are finite in number). Adaboost minimizes

$$\sum_{i=1}^{m} e^{-y_i \sum_{n=1}^{N} w_n \phi_n(\mathbf{x}_i)}$$

by coordinate descent, at each iteration t choosing coordinate $h_t = \phi_{n(t)}$ which produces the biggest decrease in error and updating $w_{n(t)} = \alpha_t$

Boosting and logistic regression

The expected error w.r.t. the exponential loss

$$\mathcal{E}_{\mathbf{x},y}\left[e^{-yf(\mathbf{x})}\right] =$$
$$= \mathcal{E}_{\mathbf{x}}\left[P(y=1|\mathbf{x})e^{-f(\mathbf{x})} + P(y=-1|\mathbf{x})e^{f(\mathbf{x})}\right]$$

is minimized (exercise) for

$$f^*(x) = \frac{1}{2}\log\frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})} \quad \Rightarrow$$
$$P(y=1|\mathbf{x}) = \frac{1}{1+e^{-2f^*(\mathbf{x})}}$$

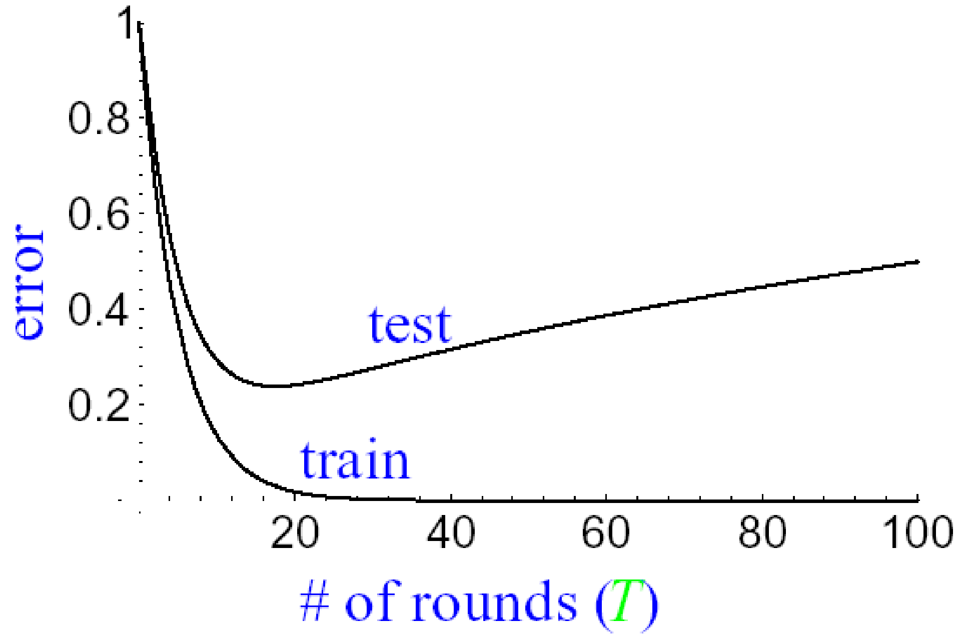
The last formula can be used to convert the output of Adaboost into a probability estimate

Generalization error (I)

How do we expect training and test error of boosting to depend on T?

We'd expect that if T is too large overfitting kicks in

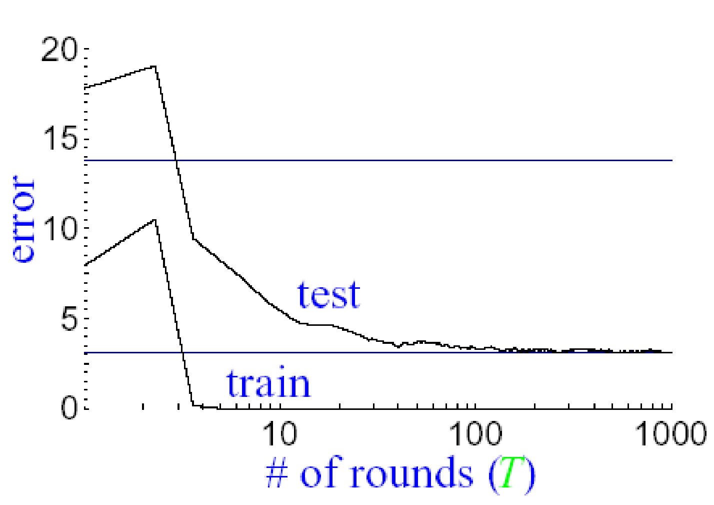
However, the plot shown here is not typically the case...



(Figures from Schapire *et. al*)

Generalization error (II)

Typically, the test error keeps decreasing even when the training error is zero! (eventually it will increase but may take many more iterations to do so)



However the **margin distribution** keeps decreasing as well which explains why test error does so

Generalization error (III)

The margin of point i is simply the quantity $\text{margin}_i = y_i f(\mathbf{x}_i)$ where we assume that $\sum_t \alpha_{t=1}^{T} = 1$ (just normalize the weights after the last iteration of boosting).

The margin distribution is the empirical distribution of the margin

One can show that with high probability (say 99%)

$$\text{generalization error} \leq \text{Pr}_{\text{emp}}(\text{margin} \leq \theta) + O\left(\frac{\sqrt{d/m}}{\theta}\right)$$

where m is the number of training points, d the VC-dimension of the set of weak learners ($\log N$ in our case) and Pr_{emp} denotes empirical probability of the margin (like the training error this converges exponentially fast to zero)

L_1 sparsity

- Rademacher complexity gives an alternative measure of function class complexity:

$$R_m(\mathcal{H}) = \mathbb{E}_S \mathbb{E}_{\sigma \in \{-1,+1\}^m} \left[\frac{2}{m} \sup_{h \in \mathcal{H}} \sum_{i=1}^{m} \sigma_i h(x_i) \right]$$

where we assume the class $\mathcal{H}$ is closed under negation.

- Rademacher complexity is not increased by taking the convex closure of $\mathcal{H}$:

$$R_m(B\mathcal{C}(\mathcal{H})) \leq B R_m(\mathcal{H}) \quad \text{for}$$

$$\mathcal{C}(\mathcal{H}) = \left\{ \sum_i \alpha_i h_i : h_i \in \mathcal{H},\ \|\alpha\|_1 = 1 \right\}$$

Generalization error for linear classifiers

Using this definition we can bound the generalisation in terms of the margin distribution as with SVMs

$$\text{generalization error} \leq \sum_{i=1}^{m} \xi_i + B R_m(\mathcal{H}) + 2\sqrt{\frac{\log(1/\delta)}{2m}}$$

where $\mathcal{H}$ is the class of weak learners with range $[-1, 1]$ and $B = \sum_{i=1}^{T} \alpha_i$.

Note the ξ_i are the margin slack variables computed as

$$\xi_i = \left(1 - y_i \sum_{j=1}^{N} \alpha_j h_j(x_i)\right)_+$$

Linear programming machine

- Note that Rademacher complexity of N feature indicators is bounded by $1/m + 4\ln(Nm)/\sqrt{m}$
- The bound suggests an optimisation similar to that of SVMs (aka 1-norm SVM).
- seeks linear function in a feature space defined explicitly.
- For example using the 1-norm it seeks $\mathbf{w}$ to solve

$$\begin{array}{ll} \min_{\mathbf{w},b,\xi} & \|\mathbf{w}\|_1 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ & i = 1, \ldots, m. \end{array}$$

Linear programming boosting

- Can explicitly optimise margin with 1-norm fixed:

$$\begin{array}{ll} \text{max}_{\rho,\mathbf{a},\xi} & \rho - D\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\mathbf{H}_i\mathbf{a} \geq \rho - \xi_i,\ \xi_i \geq 0, a_j \geq 0 \\ & \sum_{j=1}^{N} a_j = 1. \end{array}$$

- Dual has the following form:

$$\begin{array}{ll} \text{min}_{\beta,\mathbf{u}} & \beta \\ \text{subject to} & \sum_{i=1}^{m} u_i y_i \mathbf{H}_{ij} \leq \beta,\ j = 1,\ldots,N, \\ & \sum_{i=1}^{m} u_i = 1,\ 0 \leq u_i \leq D. \end{array}$$

(Demiriz, Bennett and S-T, 2001)

Column generation

Can solve the dual linear programme using an iterative method:

```
1   initialise $u_i = 1/m, i = 1, \ldots, m$, $\beta = \infty$, $J = \emptyset$
2   choose $j^\star$ that maximises $f(j) = \sum_{i=1}^m u_i y_i \mathbf{H}_{ij}$
3   if $f(j^\star) \leq \beta$ solve primal restricted to $J$ and exit
4   $J = J \cup \{j^\star\}$
5   Solve dual LP restricted to set $J$ to give $u_i, \beta$
6   Go to 2
```

- Note that u_i is a distribution on the examples
- Each j added acts like an additional weak learner
- $f(j)$ is simply the weighted classification accuracy of weak learner j
- Hence gives ‘boosting’ algorithm – optimising error bound

Multiple kernel learning

- MKL puts a 1-norm constraint on a linear combination of kernels:

$$\left\{\kappa(\mathbf{x},\mathbf{x}') = \sum_{t=1}^{N} z_t \kappa_t(\mathbf{x},\mathbf{x}') : z_t \geq 0, \sum_{t=1}^{N} z_t = 1\right\}$$

  and trains an SVM while optimizing z_t

- It is somewhat surprising that this remains a convex problem

- It would, however, appear to lead to a significant danger of overfitting if more than a handful of kernels were considered

- Question of how performance scales with N?

Empirical Rademacher complexity

- The empirical Rademacher complexity provides a way of measuring the complexity of a function class $\mathcal{F}$ by testing how well on average it can align the training data with random noise:

$$\hat{R}_m(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \frac{2}{m} \sum_{i=1}^{m} \sigma_i f\left(\mathbf{x}_i\right) \right].$$

  is known as the empirical Rademacher complexity of the function class $\mathcal{F}$.

Rademacher complexity bound for MKL

- Learning over the convex hull of the union of the individual kernel spaces

$$\mathrm{conv}\left(\bigcup_{t=1}^{N} \mathcal{F}_t\right)$$

where $\mathcal{F}_t = \{\mathbf{x} \to \langle \mathbf{w}, \phi_t(\mathbf{x})\rangle : \|\mathbf{w}\| \leq 1\}$ is t-th kernel class.

- With Rademacher analysis obtain corresponding bound (using convex hull bound for Rademacher complexity):

$$P(y \neq \mathrm{sgn}(g(\mathbf{x}))) \\ \leq \frac{1}{m\gamma}\sum_{i=1}^{m} \xi_i + \frac{1}{\gamma}\hat{R}_m\left(\bigcup_{t=1}^{N} \mathcal{F}_t\right) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}$$

Bounding MKL

- Need a bound on

$$\hat{R}_m\left(\mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t\right)$$

- It is possible to show that with probability $1 - \delta_0$ of a random selection of σ^*:

$$\hat{R}_m(\mathcal{F}) \leq \frac{2}{m} \sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) + 4\sqrt{\frac{\ln(1/\delta_t)}{2m}}$$

and

$$\frac{2}{m} \sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) \leq \hat{R}_m(\mathcal{F}_t) + 4\sqrt{\frac{\ln(1/\delta_t)}{2m}}$$

with probability $1 - \delta_t$

Bounding MKL

- Hence taking $\delta_t = \delta/2(N+1)$ for $t = 0, \ldots, N$

$$\begin{aligned}
\hat{R}_m\left(\mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t\right) & \\
&\leq \frac{2}{m} \sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) + 4\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} \\
&\leq \frac{2}{m} \max_{1 \leq t \leq N} \sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) + 4\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} \\
&\leq \frac{2}{m} \max_{1 \leq t \leq N} \hat{R}_m(\mathcal{F}_t) + 8\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}}
\end{aligned}$$

with probability $1 - \delta/2$.

Bounding MKL

- This gives an overall bound on the generalisation of MKL of

$$P(y \neq \mathrm{sgn}(g(\mathbf{x}))) \leq \frac{1}{m\gamma}\sum_{i=1}^{m}\xi_i + \frac{2}{\gamma m}\max_{1\leq t\leq N}\sqrt{\mathrm{trace}(\mathbf{K}_t)} + \frac{8}{\gamma}\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} + 3\sqrt{\frac{\ln(4/\delta)}{2m}}$$

  where $\mathbf{K}_t$ is the t-th kernel matrix.

- Bound gives only a logarithmic dependence on the number of kernels.

Experimental results with large-scale MKL

- Vedaldi et al. have applied to the PASCAL Visual Objects Challenge (VOC 2007) data and
  - improvements over the winners of the challenge in 17 out of the 20 categories
  - in more than half of the categories the increase in average precision was over 25%
  - have also scaled effectively to millions of kernels

$\star$ A. Vedaldi, V. Gulshan, M. Varma and A. Zisserman. Multiple kernels for object detection. In Proceedings CVPR, Kyoto, Japan, September 2009.

Linear Programming MKL

- Column generation gives efficient MKL if we can pick the best weak learner in each $\mathcal{F}_t$ efficiently:

$$\begin{aligned}
\sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} u_i y_i f(\mathbf{x}_i) &= \sup_{\mathbf{w}:\|\mathbf{w}\| \leq 1} \sum_{i=1}^{m} u_i y_i \langle \mathbf{w}, \phi_t(\mathbf{x}_i) \rangle \\
&= \sup_{\mathbf{w}:\|\mathbf{w}\| \leq 1} \left\langle \mathbf{w}, \sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i) \right\rangle \\
&= \left\| \sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i) \right\| \\
&= \sqrt{\mathbf{u}' \mathbf{Y} \mathbf{K}_t \mathbf{Y} \mathbf{u}} =: N_t
\end{aligned}$$

easily computable from the kernel matrices (note that $\mathbf{u}$ is sparse after first iteration and can also be chosen sparse at the start).

MKL Algorithmics

- The optimal weak learner from $\mathcal{F}_t$ is realised by the weight vector that achieves the supremum

$$\mathbf{w} = \frac{\sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i)}{\left\| \sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i) \right\|}$$

  which has dual representation: $\alpha_i = \frac{1}{N_t} u_i y_i$

- Hence, can use the linear programming boosting approach to implement multiple kernel learning.

- More generally can view the $\mathbf{u}$ vector as a signal to refine other representations

L_1 regression: LASSO

- We can apply the same regularisation with least squares regression
- This also leads to a sparse set of features being included
- The technique is known as LASSO
- Software available to compute the complete regularisation path: e.g. LARS

Sparsity Compression bounds

- Basic 'folk' theorem: given a compression scheme:

$$\Lambda : S \longmapsto \Lambda(S) \subset S$$

such that can reconstruct the classifier output for a learning algorithm $\mathcal{A}$:

$$\mathcal{A}(S) = \Phi(\Lambda(S))$$

then can bound generalisation in terms of $d = |\Lambda(S)|$ and $k = \hat{\text{err}}(\mathcal{A}(S))$ the empirical error:

$$\text{err}(\mathcal{A}(S)) \leq \frac{1}{m-k-d}\left(\ln\binom{m}{k+d}\binom{k+d}{d} + 2\ln\frac{m}{\delta}\right)$$

(Littlestone and Warmuth, 1986; Floyd and Warmuth, 1995)

Sparsity Compression bounds

- Proof relies on counting ways you could choose the classifier by specifying the set of examples and then using the rest of the examples as a ‘test set’ to verify the quality of the classifier.

- Example is number of support vectors for an SVM.

- Can consider adding in additional side information needed to specify the classifier.

- Inspires optimisation of the sparsity.

Perceptron algorithm

- If we consider the set of points on which the perceptron algorithm makes updates, this forms a compression set

- By Novikoff's theorem this implies a bound on the generalisation of the perceptron algorithm in terms of the margin: R^2/γ^2

- Similar to SVM bounds – perceptron performance can match the SVM if we update more aggressively.

Relation to the VC dimension

- The VC dimension was shown to characterise learnability in a distribution free sense

- Question of the relation of VC dimension to compression was raised in the sense that does a class of VC dimension d always exhibit a compression scheme of order d.

- True for maximal VC classes – but open in general.

Set cover machine

- Set cover machine makes use of the fact that we can use information in the chosen examples in the reconstruction – gives data dependent function classes.

- SCM greedily chooses a pair of points one negative and one positive, so that the sphere centred at the negative point with radius equal to the distance between them covers the most negative examples and no positives.

- The covered points are removed and the process is iterated.

- This creates a conjunction of complements of spheres that exclude the set of negative examples – this defines the positive region.

Performance of SCM

- The SCM is not a simple compression scheme as we need the information of which pairs of examples define the spheres

- This amounts to a modest amount of side information as some inference can be used to work out which examples are matched.

- If $B_k(m_p, m_n)$ is the number of ways we can choose k balls from m_p positive and m_n negative examples then generalisation can be bounded by a standard expression in which $\log(B_k(m_p, m_n))$ plays the role of the VC dimension.

Greedy approximation

- guaranteed to find good approximation if small cover exists: if r sized cover exists, then $r\ln(|S|)$ cover will be found.
  - suppose k examples remain at some stage. Since r spheres can cover these there exists a sphere that will cover k/r of them. Hence, greedy choice will reduce the remaining examples by a factor of at least
$$1-\frac{1}{r}$$
  - Hence, t iterations reduces the remaining set by a factor of
$$\left(1-\frac{1}{r}\right)^t \leq \exp\left(-\frac{t}{r}\right)$$
  - Setting the rhs equal to $1/|S|$ gives the result.

Extensions of the SCM

- I have presented the simplest version of the SCM. There are many variants/extensions:
  - use disjunctions instead of conjunctions
  - create decision lists rather than conjunctions
  - use hyperplanes defined by two or three examples rather than spheres
  - etc.
- the proof technique we have outlined for bounding the error extends naturally to all of these cases.

Set cover machine

- Results on some UCI datasets:

Data Set	Type	SVM error		SCM		
		$C = \infty$	Finite C	size	error	bound
BreastW	c	27	19	1	23	139
Votes	c	5	3	1	6	23
Pima	d	243	203	20	208	626
Haberman	c	111	71	1	76	202
Bupa	d	121	107	46	118	320
Glass	d	42	34	11	36	104
Credit	d	205	190	79	217	608

- Will see similar ideas used again for matching pursuit.

Matching Pursuit

- Matching pursuit greedily chooses training examples that determine directions in feature space that are well-suited to some task and then deflates
- Analysis combining sparse reconstruction with generalisation error bounds gives first bounds on performance in learnt subspace
- Allows different criteria for selection to be implemented in one framework, eg sparse PCA, classification, regression, canonical correlation analysis, etc. and all come with bounds

Kernel matching pursuit

- Not a compression scheme as uses additional information in the sparsely selected subspace to compute the regressor

- Can combine sample compression bound with standard VC bound to obtain bound on the probability of an error being greater than a given threshold ϵ:

$$\mathcal{E}[\ell(\mathcal{A}(S))] \leq \frac{2}{m-d-t}\left[(d+1)\log\left(\frac{4e(m-d-t)}{d+1}\right) + d\log\left(\frac{em}{d}\right) + t\log\left(\frac{e(m-d)}{t}\right) + \log\left(\frac{2m^2}{\delta}\right)\right].$$

with d the dimension of the selected space and t the empirical rate for errors greater than ϵ.

Kernel matching pursuit

```
Require: kernel $\mathbf{K}$, sparsity parameter $d > 0$.
1: initialise $\mathbf{i} = [\,]$
2: for $i = 1$ to $d$ do
3:     set $\mathbf{i}_i$ to the index of $\max \left| \frac{(\mathbf{K}\mathbf{y})}{\sqrt{(\mathbf{K}^{\cdot 2})'\mathbf{1}}} \right|$
4:     set $\boldsymbol{\tau} = \mathbf{K}[:, \mathbf{i}_i]$, then deflate
           $$\mathbf{K} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}\boldsymbol{\tau}'}{\boldsymbol{\tau}'\boldsymbol{\tau}}\right)\mathbf{K}$$
5: end for
Ensure: index vector $\mathbf{i}$ and regression solution in the space spanned by
    the points with these indices
```