LECTURE NOTE #NEW 7

PROF. ALAN YUILLE

1. Introduction to AdaBoost

AdaBoost is a method for combining many weak classifiers to make a strong classifier.

Input: set of weak classifiers $\{\phi_\mu(x) : \mu = 1, ..., M\}$. Labelled data $\mathcal{X} = \{(x^t, y^t) : t = 1, ..., N\}$, with $y_t\{\pm 1\}$.

Output: strong classifier:

$$S(x) = sign(\sum_{\mu=1}^{M} \lambda_\mu \phi_\mu(x),)$$

where the $\{\lambda_\mu\}$ are weights (to be learnt).

We generally want most of the $\lambda_\mu = 0$, this means that the corresponding weak classifier $\phi_\mu(.)$ is *not selected*.

Note: the strong classifier is a plane in feature space, see figure (1).

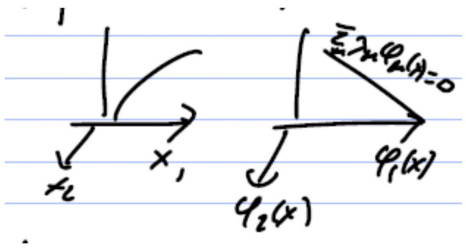

Figure 1. AdaBoost learns a strong classifier which is a plane in feature space..

2. AdaBoost Motivation

The task of AdaBoost is to select weights $\{\lambda_\mu\}$ to make a strong classifier whose performance is as good as possible.

The motivation is that it is often possible to specify weak classifiers for a problem – e.g., to find a "weak classifier" that is effective sixty percent of the time. Or, more generally to specify a large dictionary of weak classifiers from which AdaBoost can select a limited number to build a strong classifier which is correct ninety nine point nine percent of the time.

Why combine the weak classifier by linear weights? Because there is an efficient algorithm to do this. And because this algorithm appears to give a classifier which generalizes well even if there is only limited amounts of data (as always, cross-validation is used to check that AdaBoost does not over-generalize).

Note that weak classifiers which are correct only forty percent of the time are still useful. By changing their sign, we can make a weak classifier that is right fifty five percent of the time. (Maybe you have a friend whose advice is usually bad – if so, ask for their advice, but then do the opposite),

The "best" way to combine weak classifiers if we have enough data is to learn the distribution $P(y|\{\phi_\mu(x)\})$ – the distribution of y conditioned on the data. But this requires much too much data to be practical (most of the time).

3. Example: Face Detection

The training set are a set of images I (we don't use x because that is used to denote position in the image) which are labelled as face or non-face – $y = 1$ means face, $y = -1$ means not-face, see figure (1).

Viola and Jones calculated image features $f(I)$. For example, they computed the average intensity within different subregions of the images and subtracted these averages to obtain the feature $f(I)$. They obtain a weak classifier by thresholding the value of the feature (e.g., set $y = 1$ if $f(I) > T$. For example, the forehead region is typically a lot brighter than the region directly below – the eyebrows and eyes – which gives us a weak classifier by putting a threshold on the magnitude of this difference. Similarly faces are often symmetric, so the average intensity on one side of the face is uaually equal to the intensity on the other side – so the difference between these averages is often smaller than a threshold.

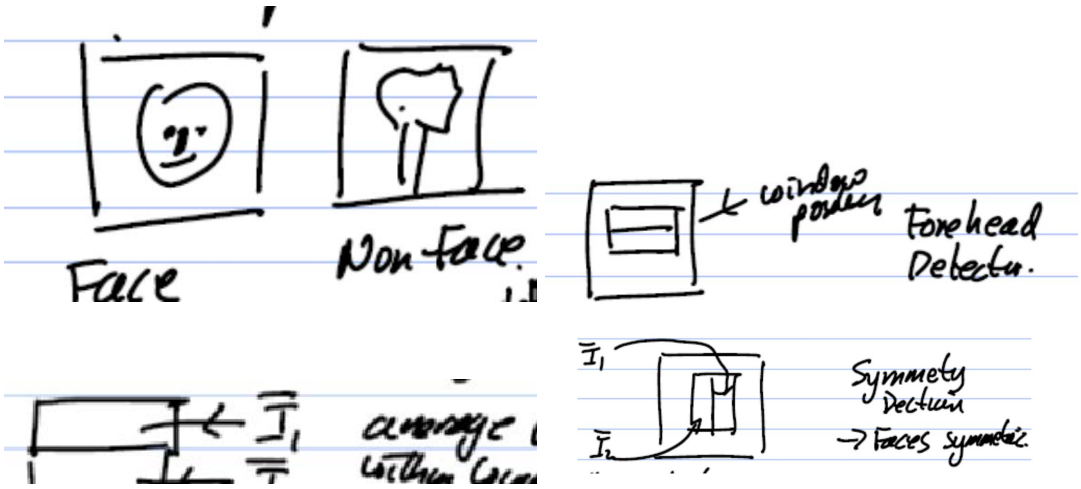

Figure 2. The face detection problem (left panel). Weak classifiers for detecting faces (remaining panels).

Viola and Jones specified a large dictionary of weak classfiers of this type ($M = 20,000$).

Note: AdaBoost will only give you good results if you have a good dictionary of weak classifiers.

4. AdaBoost: The convex upper bound Z of the empirical risk

Let

$$Z[\lambda_1, ..., \lambda_M] = \sum_{t=1}^{N} \exp\{-y^t \sum_{\mu=1}^{M} \lambda_\mu(x)\}.$$

This is a *convex* upper bound of the empirical risk of the strong classifier $S(.)$. Convexity can be shown by calculating the Hessian and showing it is positive definite (Cauchy-Schwartz inequality).

To prove it is an upper bound of the empirical risk (penalizing all errors the same), recall that the empirical risk of a strong classifier $S(.)$ is:

$$R_{emp}(\mathcal{X}) = \sum_{t=1}^{N} \{1 - I(S(x^t) = y^t)\},$$

where $I(.)$ is the indicator variable.

To prove the result, we compare equations (4,4) term by term. Terms in the empirical risk take value 1, if $S(x^t) \neq y^t$ (if the strong classifier is wrong) and value 0 if $S(x^t) = y^t$ (if the strong classifier gets the right answer).

Mathematical Fact: $y^t S(x)^t = 1$ if the strong classifier is correct and $y^t S(x^t) = -1$ is the strong classifier is wrong. This mathematical fact is useful when we discuss AdaBoost and also Max-Margin methods (e.g., Support Vector Machines) in the next lecture.

Now consider the terms $\exp\{\sum_{\mu=1}^{M} \lambda_\mu y^t \phi_\mu(x^t)\}$ in $Z[\lambda_1, ..., \lambda_M]$. If the strong classifier is right – $S(x^t) = y^t$ – then y^t and $\sum_{\mu=1}^{M} \lambda_\mu \phi_\mu(x)$ must have the same sign. This implies that $y^t sum_{\mu=1}^{M} \lambda_\mu \phi_\mu(x) > 0$, hence $\exp\{\sum_{\mu=1}^{M} \lambda_\mu y^t \phi_\mu(x^t)\} > e > 1$. If the strong classier is wrong, then the term take value between 0 and e, which is greater than 0. Hence each term in Z is bigger than the corresponding term in the empirical risk.

Note: this is a standard technique in machine learning. The empirical risk is a non-convex function of the parameters λ of the strong classifier. But we can bound it by a convex function Z. This allows us to specify an algorithm that is guaranteed to converge to the global minimum of Z, which hence gives an upper bound on the empirical risk. (This doesn't mean that we have minimized the empirical risk – but, it does mean that we know that is below the minimum of Z).

5. AdaBoost: Coordinate Descent

The AdaBoost algorithm can be expressed as coordinate descent in Z (this was not how it was originally described), see figure (3).

Initialize $\lambda_\mu = 0$, for $\mu = 1, ..., M$.

At time step t, suppose the weights are $\{\lambda_\mu^t : \mu = 1, ..., M\}$,

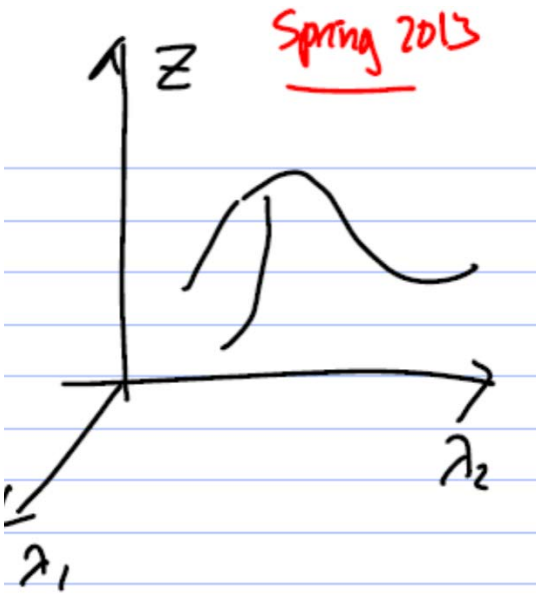

FIGURE 3. AdaBoost performs coordinate descent in Z. At each time step it calculate the best direction to maximally minimize Z – i.e. by selecting $\hat{\mu}$ – and then moves in that direction.

For each μ, minimize Z with respect to λ_μ keeping the remaining λ's fixed. I.e. solve $\frac{\partial Z}{\partial \lambda_\mu} = 0$ (because Z is convex there will only be one solution) to obtain $\hat{\lambda}_\mu$. Then compute $Z[\lambda_1^t, ..., \hat{\lambda}_\mu, ..., \lambda_M^t]$. This gives the decrease in Z due to minimizing it with respect to the μ^{th} λ. (See next section for how to do this).

Then compute:

$$\hat{\mu} = \arg\min Z[\lambda_1^t, ..., \hat{\lambda}_\mu, ..., \lambda_M^t].$$

(I.e. find the μ for which we can maximal decrease in Z.

Then update the weights:

$$\lambda_{\hat{\lambda}}^{t+1} = \hat{\lambda}_{\hat{\mu}}, \quad \lambda_\mu^{t+1} = \lambda_\mu^t, \text{ for all } \mu \neq \hat{\mu}.$$

Repeat until Z stops decreasing.

Note: this may over-learn the data. In practice, stop earlier and check by cross-validation.

Intuition. At each time step calculate how much you can decrease Z by changing only one of the λ's, and chose the λ which gives the biggest decrease. Each step of the algorithm decreases Z and so the algorithm converges to the (unique) global minimum of Z.

Note: this algorithm is only practical because we can solve for $\hat{\lambda}_\mu$ and for $\hat{\mu}$ efficiently, see next section.

6. Details of AdaBoost

For each weak classifier $\phi_\mu(.)$ divide the data into two sets: (i) the set $W_\mu^+ = \{t : y^t\phi_\mu(x^t) = 1\}$, and (ii) $W_\mu^- = \{t : y^t\phi_\mu(x^t) = 1\}$, I.e. W_μ^+ is the set of the data which $\phi_\mu(.)$ classifies correctly, and W_μ^- is the set which it gets wrong,

Then, at each time step l, define a set of "weights" for the training data:

$$D_t^l = \frac{e^{-y^t \sum_{\mu=1}^M \lambda_\mu^l \phi_\mu(x^t)}}{\sum_t e^{-y^t \sum_{\mu=1}^M \lambda_\mu^l \phi_\mu(x^t)}}.$$

These weights are all positive and sum to 1 – i.e. $\sum_t D_t^l = 1$. At $l = 0$ all the weights take value $1/N$. Otherwise, the weights are largest for the data which is incorrectly classified by the classifier $sign(\sum_{\mu=1}^M \lambda_\mu^l \phi_\mu(x^t))$ (our current estimate of the strong classifier) and smallest for those which are correctly classified (to see this, look at the sign of the exponent). *These weights are a way to take into account the weak classifiers we have already selected – and the weights we have assigned them – when we try to add a new classifier.*

Now we describe the AdaBoost algorithm, explicity showing how to compute the steps which were summarized in the previous section.

Initialize $\lambda_\mu = 0$, for $\mu = 1, ..., M$.

At time step l, let the weights be $\{\lambda_\mu^l : \mu = 1, ..., M\}$. Then, for each μ compute:

$$\Delta_\mu^l = \frac{1}{2} \log \frac{\sum_{t \in W_\mu^+} D_t^l}{\sum_{t \in W_\mu^-} D_t^l}.$$

(this corresponds to solving $\partial Z/\partial \lambda_\mu = 0$, see next section).

Then solve for:

$$\hat{\mu} = \arg\min \sqrt{\sum_{t \in W_\mu^+} D_t^l} \sqrt{\sum_{t \in W_\mu^-} D_t^l}.$$

(This computes the change in Z, and selects the μ for which this change is biggest). Then set:

$$\lambda_{\hat{\mu}}^{l+1} = \lambda_{\hat{\mu}}^l + \Delta_{\hat{\mu}}, \quad \lambda_\mu^{l+1} = \lambda_\mu^l, \text{ for all } \mu \neq \hat{\mu}.$$

Repeat until convergence.

Convergence occurs when $\sqrt{\sum_{t \in W_\mu^+} D_t^l} \sqrt{\sum_{t \in W_\mu^-} D_t^l}$ takes its maximal value of $1/2$ for all μ. (To see this is the maximal possible value, observe that $\sum_{t \in W_\mu^+} D_t^l + \sum_{t \in W_\mu^-} D_t^l = 1$.) This maximal values occurs when the weak classifier gets exactly half the data correct (allowing for the data weighs) – i.e. when $\sum_{t \in W_\mu^+} D_t^l = \sum_{t \in W_\mu^-} D_t^l = 1/2$. (Note, if the weights take vale $1/N$ – at the start of the algorithm – then this is the condition that the weak classifier is fifty percent correct).

7. AdaBoost algorithm and the mathematics

$$\frac{\partial Z}{\partial \lambda_\mu} = \sum_{t=1}^{N} (-y^t \phi_\mu(x^t)) e^{-y^t \sum_{\mu=1}^{M} \lambda_\mu \phi_\mu(x^t)}.$$

Solve for $\lambda_\mu + \Delta_\mu$. $\frac{\partial Z}{\partial \lambda_\mu} = 0$ implies:

$$\sum_{t=1}^{N} (y^t \phi_\mu(x^t)) e^{-y^t \sum_{\mu=1}^{M} \lambda_\mu \phi_\mu(x^t)} e^{-y^t \Delta_\mu \phi_\mu(x^t)} = 0$$

Which implies that (using the definition of D_l);

$$\sum_{t=1}^{N} (y^t \phi_\mu(x^t)) D_t e^{-y^t \Delta_\mu \phi_\mu(x^t)} = 0$$

This can be expressed as (using definitions of W^+ and W^-):

$$\sum_{t \in W_\mu^+} D_t e^{-\Delta_\mu} - \sum_{t \in W_\mu^-} D_t e^{\Delta_\mu} = 0$$

. This can be solved to give the result:

$$\Delta_\mu = \frac{1}{2} \log \frac{\sum_{t \in W_\mu^+} D_t}{\sum_{t \in W_\mu^-} D_t}.$$

Now compute $Z[\lambda_1, ..., \lambda_\mu + \Delta_\mu, ..., \lambda_M]$ by:

$$Z[\lambda_1, ..., \lambda_\mu + \Delta_\mu, ..., \lambda_M] = \sum_{t=1}^{N} e^{-y^t \{\sum_{\mu=1}^{M} \lambda_\mu \phi_\mu(x^t) + \Delta_\mu(x^t)\}}$$

$$= K \sum_{t=1}^{N} D_l e^{\Delta_\mu(x^t)}$$

where $K = \sum_{t=1}^{N} D_t e^{-y^t \Delta_\mu \phi_\mu(x^t)}$ is independent of μ. Hence:

$$Z[\lambda_1, ..., \lambda_\mu + \Delta_\mu, ..., \lambda_M] = K\{ \sum_{t \in W_\mu^+} D_t e^{-\Delta_i} + \sum_{t \in W_\mu^-} e^{\Delta_\mu} \}$$

$$= 2K \sqrt{\sum_{t \in W_\mu^+} D_t} \sqrt{\sum_{t \in W_\mu^-} D_t}$$

.

This shows the equivalence between the mathematical and the algorithmic descriptions of AdaBoost.

8. AdaBoost and Regression

It has been shown (Friedman, Hastie, Tishbirani) that AdaBoost converges asymptotically to a solution to the regression problem:

$$P(y|x) = \frac{e^{y\sum_\mu \lambda_\mu \phi_\mu(x)}}{e^{\sum_\mu \lambda_\mu \phi_\mu(x)} + e^{-\sum_\mu \lambda_\mu \phi_\mu(x)}}.$$

This holds only in the limit as the amount of data tends to infinity (plus other technical conditions).

It has been argued (Miller, Lebanon, and others) that doing regression gives better results that AdaBoost. It does require more computation. It also requires enforcing a sparsity constrain that most of the λ's are zero (see lecture on sparsity).