Lecture 14: Detecting Faces and Text

A.L. Yuille

March 12, 2012

1 Introduction

This lecture discusses way to detect faces and text in natural images. The strategy is to combine multiple cues to build a strong classifier – which involves selecting cues and weighting their importance. This can be formulated as a regression problem or, in machine learning terms, to AdaBoost.

An important issue here is the idea of invariance to *nuisance variables*. The image of a face depends on the identity of the person, the lighting conditions, and other factors. These can be thought of as nuisance variables. They must either be modeled explicitly or we must find cues which are invariant to them. Similarly we can treat text as "texture" and treat the identity of the letters, and the form of the font, as nuisance variables. Then we can search for cues which are invariant to these nuisances.

The nuisance variables must be modeled explicitly if we are making generative modesls of the object – i.e. models of form $P(I|W), P(W)$ that enable us to sample a configuration W of the nuisance variables (from $P(W)$) and then an image I from $P(I|W)$). But we do not need models this detailed for detection except in really extreme circumstances (there are not make objects in the world that look like faces or text). If we want to perform the harder tasks of recognizing faces (e.g., Obama or Romney) then we have to model some of these nuisance variables. Similarly is we want to read the text – does it say "Obama" or "Romney"?

The methods in this lecture are limited by this requirement – e.g., invariance to nuisance variables. Suppose you want to build a cat detector – how can you find visual cues/features of the cat which are invariant to the different poses of the cat? (Cats are extremely flexible). This is much harder, and maybe impossible, compared to finding invariant features for the face of a cat – because properties like the relative positions of eyes, mouth, nose, ears, and whiskers are fairly constant for all cats. But in general, if you are detecting objects which has large variations in their spatial relationships (e.g., cats, groups of cats, groups of humans) then it becomes very hard to get invariance to nuisance variables – and so more sophisticated models are needed, (see later lectures).

2 Regression and AdaBoost

Consider the problem of detecting an object like a face or text. This can be formulated as a binary classification problem. The data is a set of images $\{\mathbf{I}^\mu\}$ and binary-valued labels $\{y^\mu\}$ ($y^\mu \in \{-1, 1\}$). This is an example of regression, where the goal is to learn a conditional probability distribution $P(y|\mathbf{I})$.

How can this be modeled? The modeling task is to specify a class of distributions $P(y|\mathbf{I})$ and learning/estimating which of these distribution to use (based on training data). There are a range of possible distributions which can be used.

One class is non-parametric: we represent the joint distributions as $P(\mathbf{I}, y) = (1/N)\sum_{\mu=1}^{N}\delta(\mathbf{I}-\mathbf{I}^\mu)\delta(y-y^\mu)$. This can be smoothed by a kernel K (e.g., Parzen windows, like a Gaussian) to take form $P(\mathbf{I}, y) = (1/N)\sum_{\mu=1}^{N}K(\mathbf{I}-\mathbf{I}^\mu)\delta(y-y^\mu)$. From this we can calculate the conditional distribution $P(y|\mathbf{I}) = \frac{\sum_{\mu=1}^{N}K(\mathbf{I}-\mathbf{I}_\mu)\delta(y-y^\mu)}{\sum_{\mu=1}^{N}\delta(y-y^\mu)}$. Intuitively, we classify images as object $y = 1$ or non-obejext $y = -1$ based on how similar the input image $\mathbf{I}$ is to to classified examples $\{\imath^\mu, y^\mu\}$. This approach can be modified to replacing the image $\mathbf{I}$ by features $\mathbf{F}(\mathbf{I})$ extracted from the image, yielding a conditional distribution $P(y|\mathbf{I}) = \frac{\sum_{\mu=1}^{N}K(\mathbf{F}(\mathbf{I})-\mathbf{F}(\mathbf{I})_\mu)\delta(y-y^\mu)}{\sum_{\mu=1}^{N}\delta(y-y^\mu)}$.

Another approach is parametric: we represent the conditional distribution by $P(y|\mathbf{I}) = \frac{1}{Z[\lambda,\mathbf{I}]}\exp\{y\lambda \cdot \phi(\mathbf{I})\}$. The reduces learning to estimate the parameter λ by maximum likelihood (ML), or some other estimator. from the training data $\{(\mathbf{I}^\mu, y^\mu)\}$. Here $\phi(\mathbf{I})$ are statistics, or features, extracted from the images (more about them later). There are two extreme cases:

Firstly: if the number of features is fixed (this corresponds to classical regression in statistics). In this case $\phi(\mathbf{I})$ is typically low-dimensional. ML estimation selects $\lambda^* = \arg\min \sum_\mu \{-\log P(y^\mu|\mathbf{I}^\mu, \lambda)\}$ (which is a convex minimization problem – computationally tractable since $Z[\lambda, \mathbf{I}]$ is easily computable, it equals $\sum_y \exp\{y\lambda \cdot \phi(\mathbf{I})\} = \exp\{\lambda \cdot \phi(\mathbf{I})\} + \exp\{-\lambda \cdot \phi(\mathbf{I})\}$.

Secondly: suppose the number of possible features is very large. For example we can specify a large dictionary of features $\{\phi^a(\mathbf{I})\}$ and decide to select only to select a small subset of them. This can be formulated as $P(y|\mathbf{I}, \lambda) = \frac{1}{Z[\lambda,\mathbf{I}]}\exp\{\sum_a \lambda^a \cdot \phi^a(\mathbf{I})\}$. One way to impose a small subset by by using a *sparsity prior*, such as $P(\lambda) = \frac{1}{Z}\exp\{-K\sum_a |\lambda_a|\}$. This prior introduces an L^1 (sparsity) norm when we perform MAP estimation – i.e. solve

$$\lambda^* = \arg\min_\lambda \sum_\mu \{-\log P(y^\mu|\mathbf{I}^\mu, \lambda)\} + K\sum_a |\lambda^a| \tag{1}$$

This encourages sparse solutions (i.e. with $\lambda_1 = 0$ for most a). This is an alternative to the better known AdaBoost algorithm (discussed below). Various authors (e.g., Muller at al, Lebanon and Lafferty) have argued that better results are obtained using this approach (theoretically better justified, slightly more computation).

Note that this formulation can easily be generalized to multiple outputs. Mathematically, we simply replace $y\phi(\mathbf{I})$ by $\phi(y, \mathbf{I})$. This enables y to take multiple values or even be vector-valued – $\vec{y} = (y_1, ..., y_N)$. The only difference is that if $\vec{y}$ is multi-valued, then it may become computationally intractable to compute the normalization term $Z[\lambda, \mathbf{I}]$ (unless the distribution can be expressed as a probability distribution defined over a graph which does not have closed loops – similar to ML learning in earlier lectures).

3 Features

For practical applications, the choice of feature, or feature directory, is critical. In principle, we could address this problem by making the dictionary so large that it includes every possible feature. But large dictionaries require a large amount of data in order to avoid over-generalization. In addition, the way the exponential models combine cues – by weighted linear addition – is not necessarily optimal.

Suppose we have two cues $\phi_1(\mathbf{I}), \phi_2(\mathbf{I})$. The best way to combine them is by computing the joint conditional distribution $P(y|\phi_1(\mathbf{I}), \phi_2(\mathbf{I}))$. This can usually not be expressed in exponential form $P(y|\mathbf{I}) = \frac{1}{Z[\lambda,\mathbf{I}]}\exp\{\lambda_1 \cdot \phi_1(\mathbf{I}) + \lambda_2 \cdot \phi_2(\mathbf{I})\}$. Similarly, if we have more cues $\phi_1(\mathbf{I}), ..., \phi_n(\mathbf{I})$ then we should combine them by the joint conditional distribution $P(y|\phi_1(\mathbf{I}), ..., \phi_n(\mathbf{I}))$ (see Minsky and Pappert, for limitations about what can be expressed in the exponential form – note, their arguments were for perceptrons but they carry over to these cases). *The big problem, however,* is that unless we have a parametric form for $P(y|\phi_1(\mathbf{I}), ..., \phi_n(\mathbf{I}))$ then it will require an exponential amount of data in order to learn it. Hence the exponential form is used partly for practical reasons, it means that the number of parameters λ scales linearly with the number of features (instead of exponentially).

Another important issue is that the cues should be, ideally, invariant to nuisance variables. For example, the image of a face will depend on the identity of the person, the lighting conditions, and the viewpoint. We can either choose to model these variables explicitly, or try to find invariant features. It is very hard to get features which are invariant to viewpoint, so typically viewpoint is fixed.

What properties of faces are relatively invariant to identity and lighting changes? For example, (i) the intensity of the forehead is almost always brighter than the intensity of the eye/eyebrow region, and (ii) faces are symmetric and so images of faces will tend to be symmetric (unless the lighting conditions are extremely asymmetric). Properties based on the derivatives of images will tend to be more invariant (consider earlier lectures – lecture three?). Successful features (e.g., Viola and Jones) can be made by Haar basis function

$B_a(x)$, where B_1 is a vector whose elements are $\{-1, 0, 1\}$ with the constraint that $\sum_x B_a(x) = 0$ (i.e. derivative filter). See figure (1). Then we can use a polarity and a threshold to produce a binary-classifier.

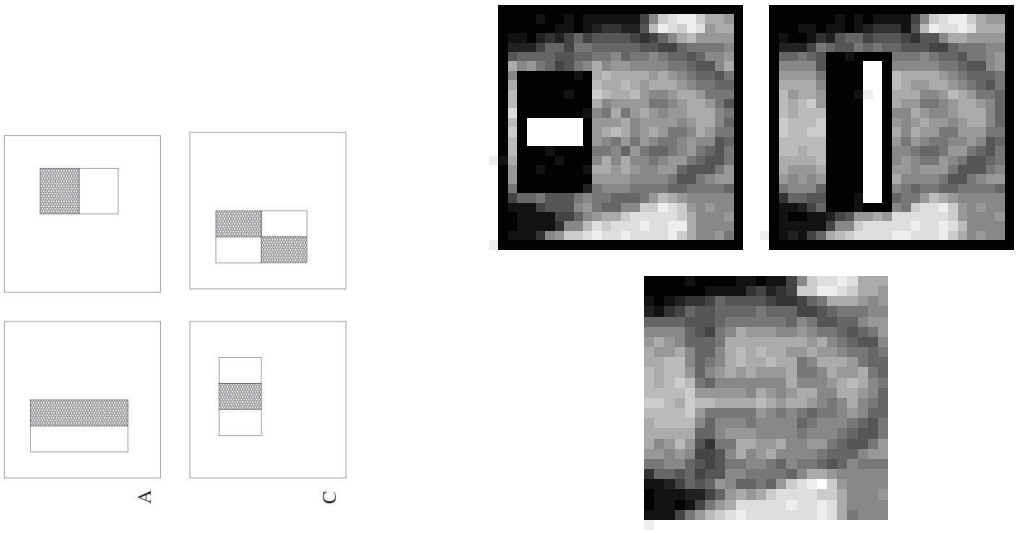

Figure 1: Left panel: The filters are rectangular binary-valued (± 1). Image window is 24×24, then a total of 49,396 features, rectangular features are used because of computational efficiency (integral images). Right panel: the top two filters respond to face symmetry and to the edge between forehead and eye region.

It is harder to find invariant features for text. The images of text vary much more – e.g., *AbC* is different than TgR. If we do principal component analysis of feces we obtain a 15 dimensional space, but we get 150 dimensions for text. Nevertheless there are invariances if we treat 'text as texture': (i) there are usually no edges in the regions directly above and below text, (ii) text regions have frequent edges in the x and y directions. This enables us to obtain features (Chen and Yuille) which are fairly invariant to different types of text. See figure (**??**). In total, In summary, we had: (i) 79 first class features including 4 intensity mean features, 12 intensity standard deviation features, 24 derivative features, (ii) 14 second class features (histograms), and (iii) 25 third class features (based on edge linking).

We mention two other issues. Firstly, in practice, cascades methods are used to speed up the algorithm (and arguably to allow it to enable it to represent combinations of features which are hard to express by an exponential additive model). The strategy (Viola and Jones) is like an attentional mechanism. It uses a small number of cues to rapidly decide that there are no faces/text in large regions of the images (surprisingly few cues can be used to do this). Each stage of the cascade can be trained by AdaBoost and constrained so that the number of false negatives is extremely small (need more details about this!). Secondly, the number of negative examples (i.e. non-faces, non-text) is much larger than the number of positives. This motivates a strategy where the negative examples are initially selected at random from all the negative examples. The AdaBoost algorithm is run to yield a classifier. This is run on the full dataset and the false positives are used as negative examples for a new run of the AdaBoost algorithm. This procedure is repeated until convergence. For example, for text the false positives are typically found in vegetation or in repetitive structure on building (which might be series of the digit 1). False positives in the first run are typically of these types. This means that the next runs of AdaBoost will select cues that deal with the examples which are hard to classify.

4 AdaBoost

AdaBoost is motivated by the idea of trying to combine together *weak classifiers*, which classifiers which are only partially successful, to make a *strong classifier* (the relationship to regression was only realized a lot

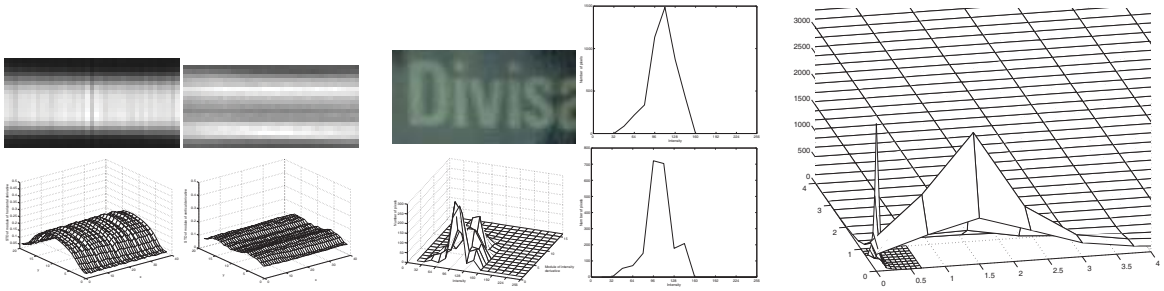

Figure 2: Left panel: The means of the modul of the x (top left0 and y (top right) derivatives have a standard pattern for text. The x-derivatives are large with large variance (bottom left). The y-derivatives are small everywhere with small variance (bottom right). The averages are different above and below the text. Center panel: the riginal image (top left) has intensity histogram (top right) with only a single peak. But the joint histograms of intensity and intensity gradient shows two peaks (bottom left) and shown in profile (bottom right). The intensity histogram is contaminated by edge pixels which have high intensity gradient and intensity values which are intermediate between the background and foreground mean intensity. The intensity gradient information helps remove this contamination. Right panel: ideally we would combine features by taking their conditional distributions and performing the log-likelihood ratio test, but this requires an amount of data which is exponential in the number of cues/features, unless we have a parametric model for the conditionals.

later). Each weak classifier can be expressed by $\phi_i(.)$, where $\phi_i(\mathbf{I}) \in \{-1, 1\}$. The error rate on the training data $\mathcal{D} = \{(\mathbf{I}_a, y_a : a = 1, .., N\}$ is given by $(1/N)\sum_{a=1}^{N} f(\phi_i(\mathbf{I}_a), y_a)$, where $f(\phi_i(\mathbf{I}_a), y_a) = 1$ if $\phi_i(\mathbf{I}_a) \neq y_a$ and $f(\phi_i(\mathbf{I}_a), y_a) = 0$ if $\phi_i(\mathbf{I}_a) = y_a$. A weak classifier has error rate ≥ 0.5. If a classifier has error rate less than 0.5 then we can convert it into a weak classifier by changing the sign of its output (i.e. if a friend's advice is always wrong then it is best to do the opposite of what he/she suggests).

A strong classifier is constructed to be of form $\sum_i \lambda_i \phi_i(\mathbf{I})$ – i.e. taking linear combinations of the weak classifiers. This is exactly like an exponential model because, after learning the λ's, the probability model will classify an input $\mathbf{I}$ by $y^* = \arg\max_y P(y|\mathbf{I})$, which corresponds to $y^* = sgn(\sum_i \lambda_i \phi_i(\mathbf{I}))$.

AdaBoost can be expressed concisely as doing coordinate descent for a function:

$$Z[\lambda] = \sum_{a=1}^{N} \exp\{-y_a \sum_i \lambda_i \phi_i(\mathbf{I}_a)\}. \tag{2}$$

This is initialized by setting $\lambda_i = 0, \ \forall i$. Then it is minimized as follows. Let $\{\lambda_i^t\}$ be the set of weights at time t. At each t, solve $\frac{\partial}{\partial \lambda_i} Z[\lambda] = 0$ to get $\lambda_i^t + \Delta_i$. For each i compute $Z_i = Z[\lambda_1^t, ..., \lambda_i^t + \Delta_i, ..., \lambda_N]$ and calculate $\hat{i} = \arg\min_i Z_i$. Then set $\lambda_{\hat{i}}^{t+1} = \lambda_{\hat{i}}^t + \Delta_{\hat{i}}$ and $\lambda_j^{t+1} = \lambda_j^t \ \forall j \neq i$. In other words, we calculate the decrease in Z due to changing each weight λ_i, and then change the weight that decreases Z most. As we will show, the form of Z means that it is possible to form the required computations directly.

It can easily be checked that $1/nZ$ is an upper bound of the error of the strong classifier $\sum_i \lambda_i \cdot \phi_i(\mathbf{I})$. This follows by comparing each term in the summation and using the fact that miss-classification occurs when $y_a \sum_i \lambda_i \cdot \phi_i(\mathbf{I}_a) < 0$ and using the fact that $\exp A > 1$ if $A > 0$, and $\exp A > 0$ if $A < 0$.

To perform the operations, we first define $D_i^+ = \{(y_a, \mathbf{I}_a) : y_a \phi_i(\mathbf{I}_a) > 0$ and $D_i^- = \{(y_a, \mathbf{I}_a) : y_a \phi_i(\mathbf{I}_a) < 0$ (i.e. D_i^+ is the set of examples which are classified correctly by weak classifier $\phi_i(.)$. Then define $w^t(a) = (1/Z_t)\exp\{-y_a \sum_i \lambda_i^t \phi_i(\mathbf{I}_a)\}$ where Z_t is chosen to normalize so that $\sum_a w^t(z) = 1$, this term is small if the data example a is well classified by the current strong classifier – i.e. $y_a \sum_i \lambda_i^t \phi_i(\mathbf{I}_a) > 0$ and is large if the data is miss-classified – i.e. $y_a \sum_i \lambda_i^t \phi_i(\mathbf{I}_a) < 0$.

The partial derivative condition $\frac{\partial}{\partial \lambda_i} Z[\lambda] = 0$ can be expressed as $\sum_a y_a \phi_j(\mathbf{I}_a) \exp\{-y_a \sum_i \lambda_i \phi_i(\mathbf{I}_a)\} = 0$, where $\lambda_i = \lambda_i^t \ \forall i \neq j$ and $\lambda_j = \lambda_i^t + \Delta_i$. Using the fact that $y_a \phi_j(\mathbf{I}_a) = 1$ if $a \in D_j^+$ and $y_a \phi_j(\mathbf{I}_a) = -1$ if $a \in$

D_j^-, we can express this as $\sum_{a\in D_j^+} \exp\{-y_a\Delta_j\phi_j(\mathbf{I}_a)\}\exp\{-y_a\sum_i \lambda_i^t\phi_i(\mathbf{I}_a)\} = \sum_{a\in D_j^-} \exp\{-y_a\Delta_j\phi_j(\mathbf{I}_a)\}\exp\{-y_a\sum_i \lambda_i^t\phi_i(\mathbf{I}$
Simply this to $\exp\{-\Delta_j\}\sum_{a\in D_j^+} w_j(a)^t = \exp\{-\Delta_j\}\sum_{a\in D_j^1} w_j(a)^t$

This gives the solution:

$$\Delta_i = \frac{1}{2}\log\frac{\sum_{a\in D_j^+} w_j^t(a)}{\sum_{a\in D_j^-} w_j^t(a)} \tag{3}$$

GIVE INTUITION!! Discuss what the weights are for. To take into account the effect of the classifiers that have already been selected. The worst classifier is right fifty percent of the time.

Similarly, we can compute the decrease in Z caused by changing λ_j^t to $\lambda_j^t+\Delta_j$. $Z = 2Z_t\{\exp\{-\Delta_i\}\sum_{a\in D_j^+} w_j(a)+ \exp\{\Delta_i\}\sum_{a\in D_j^-} w_j(a)\}$. Hence this can be computed directly so that we can estimate $\hat{i}$. It gives $Z = Z_t\{\sqrt{\sum_{a\in D_j^+} w_j(a)}\sqrt{\sum_{a\in D_j^-} w_j(a)}\}$. So we pick $\hat{j}$ to minimize $\sqrt{\sum_{a\in D_j^+} w_j(a)}\sqrt{\sum_{a\in D_j^-} w_j(a)}$.

Observe that the decrease of Z is due to the factor $2\sqrt{\sum_{a\in D_j^+} w_j(a)}\sqrt{\sum_{a\in D_j^-} w_j(a)}$. The biggest decrease, multiplying by 0, occurs if either $\sum_{a\in D_j^+} w_j(a) = 0$ or $\sum_{a\in D_j^-} w_j(a) = 0$ – or equivalently if $\sum_{a\in D_j^-} w_j(a) = 1$ or $\sum_{a\in D_j^+} w_j(a) = 1$. In other words if we can classify the training data perfectly after weighting the examples. The weighting takes into account the effect of the earlier selected classifiers. The least decrease is 1 if $\sum_{a\in D_j^+} w_j(a) = \sum_{a\in D_j^-} w_j(a) = 1/2$. In this case, the classifiers provide no information.

To summarize, the update rules of AdaBoost result from minimizing the function Z by coordinate descent (after initializing all weights to be zero). Everyting can be derived from that. The justification is that Z is an upper bound on the error of the strong classifier (if we want to minimize a different error criterion – e.g., weighting positive errors differently from negative errors – then we can derive a different bound). We do not minimize the error function directly, with respect to λ, because it is non-convex (while Z is a convex function). We can alternatively estimate the λ by ML, or MAP, as described above (previous section).