Lecture topics:

- margin and generalization
  - linear classifiers
  - ensembles
- mixture models

Margin and generalization: linear classifiers

As we increase the number of data points, any set of classifiers we are considering may no longer be able to label the points in all possible ways. Such emerging constraints are critical to be able to predict labels for new points. This motivates a key measure of complexity of the set of classifiers, the *Vapnik-Chervonenkis dimension*. The VC-dimension is defined as the maximum number of points that a classifier can shatter. The VC-dimension of linear classifiers on the plane is three (see previous lecture). Note that the definition involves a maximum over the possible points. In other words, we may find less than d_{VC} points that the set of classifiers cannot shatter (e.g., linear classifiers with points exactly on a line in $2-d$) but there cannot be any set of more than d_{VC} points that the classifier can shatter.

The VC-dimension of the set of linear classifiers in $d-$dimensions is $d+1$, i.e., the number of parameters. This relation to the number of parameters is typical albeit certainly not always true (e.g., the VC-dimension may be infinite for a classifier with a single real parameter!).

The VC-dimension immediately generalizes our previous results for bounding the expected error from a finite number of classifiers. There are a number of technical steps involved that we won't get into, however. Loosely speaking, d_{VC} takes the place of the logarithm of the number of classifiers in our set. In other words, we are counting the number of classifiers on the basis of how they can label points, not based on their identities in the set. More precisely, we have for any set of classifiers $\mathcal{F}$: with probability at least $1-\delta$ over the choice of the training set,

$$R(f) \leq R_n(f) + \epsilon(n, d_{VC}, \delta), \quad \text{uniformly for all } f \in \mathcal{F} \tag{1}$$

where the complexity penalty is now a function of $d_{VC} = d_{VC}(\mathcal{F})$:

$$\epsilon(n, d_{VC}, \delta) = \sqrt{\frac{d_{VC}(\log(2n/d_{VC}) + 1) + \log(1/(4\delta))}{n}} \tag{2}$$

The result is problematic for kernel methods. For example, the VC-dimension of kernel classifiers with the radial basis kernel is ∞. We can, however, incorporate the notion of margin in the classifier "dimension". One such definition is V_γ dimension that measures the VC-dimension with the constraint that distinct labelings have to be obtained with a fixed margin γ. Suppose all the examples fall within an enclosing sphere of radius R. Then, as we increase γ, there will be very few examples we can classify in all possible ways with this constraint (especially when $\gamma \to R$; cf. Figure 1). Put another way, the VC-dimension of a set of linear classifiers required to attain a prescribed margin can be much lower (decreasing as a function of the margin). In fact, V_γ dimension for linear classifiers is bounded by R^2/γ^2, i.e., inversely proportional to the squared margin. Note that this result is *independent of the dimension of input examples*, and is exactly the mistake bound for the perceptron algorithm!

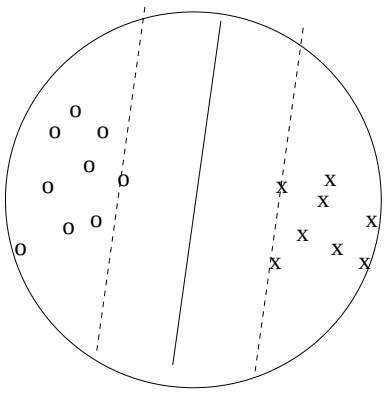

Figure 1: The set of linear classifiers required to obtain a specific geometric margin has a lower VC-dimension when the examples remain within an enclosing sphere.

The previous generalization guarantees can be used with V_γ dimension as well so long as we replace the training error with margin violations, i.e., we count the fraction of examples that cannot be separated with margin at least γ.

Margin and generalization: ensembles

An ensemble classifier can be written as a convex combination of simpler base classifiers

$$h_m(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j h(\mathbf{x}; \theta_j) \tag{3}$$

where $\alpha_j \geq 0$ and $\sum_{j=1}^m \alpha_j = 1$. Boosting generates such ensembles but does not normalized the coefficients α_j to sum to one. The normalization can be easily performed after the fact.

We are interested in understanding the complexity of such ensembles and how generalization guarantees depends on the voting margin achieved, e.g., through boosting algorithm. Note that our discussion here will not refer to how such ensembles are generated.

Let's start by defining what the ensembles are not. They are not decision trees. A decision (classification) tree is a method of recursively partitioning the set of examples into regions such that within each region the examples would have as uniform labels as possible. The partitioning in a decision tree could be based on the same type of decision stumps as we have used for the ensemble. In the ensemble, however, the domain for all the stumps is the whole space. In other words, you cannot restrict the application of the stump within a specific partition. In the ensemble, each stump contributes to the classification of all the examples.

How powerful are ensembles based on the decision stumps? To understand this further let's show how we can shatter any n points with $2n$ stumps even in one dimensions. It suffices to show that we can find an ensemble with $2n$ stumps that reproduces any specific labeling $y_1, \ldots, y_n$ of n points $x_1, \ldots, x_n$ (now real numbers). To do so, we will construct an ensemble of two stumps to reproduce the label y_t for x_t but without affecting the classification of other training examples. If ϵ is less than the smallest distance between any two training examples, then

$$h_{pair}(x; x_t, y_t) = \frac{1}{2}\text{sign}\left(y_t(x - x_t + \epsilon)\right) + \frac{1}{2}\text{sign}\left(-y_t(x - x_t - \epsilon)\right) \tag{4}$$

has value y_t within interval $[x_t - \epsilon, x_t + \epsilon]$ is zero everywhere else. Thus, setting $\alpha_t = 1/n$,

$$h_{2n}(x) = \sum_{t=1} \alpha_t h_{pair}(x; x_t, y_t) \tag{5}$$

has the correct sign for all the training examples. The ensemble of $2n$ components therefore has VC-dimension at least n. Ensembles are powerful as classifiers in this sense and their VC-dimension poorly explains their success in practice.

Each example in the above construction only has a very low voting margin $1/n$, however. Perhaps we can similarly refine the analysis to incorporate the voting margin as we did above with linear classifiers and the geometric margin. The key idea is to reduce an ensemble with many components to a coarse ensemble with few components but one that nevertheless classifies the examples in the same way. When the original ensemble achieves a large voting margin this is indeed possible, and the size of the coarse approximation that

we need decreases with increasing voting margin. In other words, if we achieve a large voting margin, we could have solved the same classification problem with much smaller ensemble insofar as we only pay attention to the classification error.

Based on this and other more technical ideas we can show that with probability at least $1 - \delta$ over the choice of the training data,

$$R_n(\hat{h}) \leq R_n(\hat{h}; \rho) + \tilde{\mathcal{O}}\left(\sqrt{\frac{d_{VC}/\rho^2}{n}}\right), \tag{6}$$

where the $\tilde{\mathcal{O}}(\cdot)$ notation hides constants and logarithmic terms, $R_n(\hat{h}; \rho)$ counts the number of training examples with voting margin less than ρ, and d_{VC} is the VC-dimension of the base classifiers. Note that the result does not depend on the number of base classifiers in the ensemble $\hat{h}$. Note also that the effective dimension d_{VC}/ρ^2 that the number of training examples is compared to has a similar form as before, decreasing with the margin ρ. See Schapire et al. (1998) for details. The paper is available from the course website as optional reading.

Mixture models

There are many problems in machine learning that are not simple classification problems but rather modeling problems (e.g., clustering, diagnosis, combining multiple information sources for sequence annotation, and so on). Moreover, even within classification problems, we often have unobserved variables that would make a difference in terms of classification. For example, if we are interested in classifying tissue samples into specific categories (e.g., tumor type), it would be useful to know the composition of the tissue sample in terms of cells that are present and in what proportions. While such variables are not typically observed, we can still model them and make use of their presence in prediction.

Mixture models are simple probability models that try to capture ambiguities in the available data. They are simple, widely used and useful. As the name suggests, a mixture model "mixes" different predictions on the probability scale. The mixing is based on alternative ways of *generating* the observed data. Let $\mathbf{x}$ be a vector of observations. A mixture model over vectors $\mathbf{x}$ is defined as follows. We assume each $\mathbf{x}$ could be of m possible types. If we knew the type, j, we would model $\mathbf{x}$ with a conditional distribution $P(\mathbf{x}|j)$ (e.g., Gaussian with a specific mean). If the overall frequency of type j in the data is $P(j)$, then the

"mixture distribution" over $\mathbf{x}$ is given by

$$P(\mathbf{x}) = \sum_{j=1}^{m} P(\mathbf{x}|j)P(j) \tag{7}$$

In other words, $\mathbf{x}$ could be generated in m possible ways. We imagine the generative process to be as follows: sample j from the frequencies $P(j)$, then $\mathbf{x}$ from the corresponding conditional distribution $P(\mathbf{x}|j)$. Since we do not observe the particular way the example was generated (assuming the model is correct), we sum over the possibilities, weighted by the overall frequencies.

We have already encountered mixture models. Take, for example, the Naive Bayes model $P(\mathbf{x}|y)P(y)$ over the feature vector $\mathbf{x}$ and label y. If we pool together examples labeled $+1$ and those labeled -1, and throw away the label information, then the Naive Bayes model predicts feature vectors $\mathbf{x}$ according to

$$P(\mathbf{x}) = \sum_{y=\pm 1} P(\mathbf{x}|y)P(y) = \sum_{y=\pm 1} \left[\prod_{i=1}^{d} P(x_i|y) \right] P(y) \tag{8}$$

In other words, the distribution $P(\mathbf{x})$ assumes that the examples come in two different varieties corresponding to their label. This type of unobserved label information is precisely what the mixtures aim to capture.

Let's start with a simple two component mixture of Gaussians model (in two dimensions):

$$P(\mathbf{x}|\theta) = P(1)N(\mathbf{x}; \mu_1, \sigma_1^2 I) + P(2)N(\mathbf{x}; \mu_2, \sigma_2^2 I) \tag{9}$$

The parameters θ defining the mixture model are $P(1)$, $P(2)$, μ_1, μ_2, σ_1^2, and σ_2^2. Figure 2 shows data generated from such a model. Note that the frequencies $P(j)$ (a.k.a. *mixing proportions*) control the size of the resulting clusters in the data in terms of how many examples they involve, μ_j's specify the location of cluster centers, and σ_j^2's control how spread out the clusters are. Note that each example in the figure could in principle have been generated in two possible ways (which mixture component it was sampled from).

There are many ways of using mixtures. Consider, for example, the problem of predicting final exam score vectors for students in machine learning. Each observation is a vector $\mathbf{x}$ with components that specify the points the student received in a particular question. We would expect that different types of students succeed in different types of questions. This "student type" information is not available in the exam score vectors, however, but we can model it. Suppose there are n students taking the course so that we have n vector

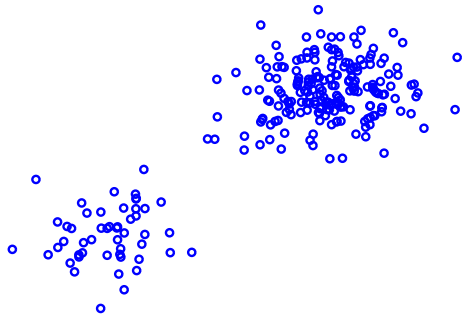

Figure 2: Samples from a mixture of two Gaussians

observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Suppose there are m underlying types of students (the number of types can be inferred from the data; this is a model selection problem). We also don't know how many students taking the course are of particular type, i.e., we have to estimate the mixing proportions $P(j)$ as well. The mixture distribution over a single example score vector is now given by

$$P(\mathbf{x}|\theta) = \sum_{j=1}^{m} P(\mathbf{x}|j)P(j) \tag{10}$$

We won't concern ourselves at this point with the problem of deciding how to parameterize the conditional distributions $P(\mathbf{x}|j)$. Suffice it to say that it wouldn't be unreasonable to assume that $P(\mathbf{x}|j) = \prod_{i=1}^{d} P(x_i|j)$ as in the Naive Bayes model but each x_i would take values in the range of scores for the corresponding exam question. Now, our mixture model assumes that each student is of particular type. If someone gave us this information, i.e., j_t for $\mathbf{x}_t$, then we would model the observations with the conditional distribution

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_n|j_1, \ldots, j_n, \theta) = \prod_{t=1}^{n} P(\mathbf{x}_t|j_t) \tag{11}$$

assuming each student obtains their score independently from others. But the type information is not present in the data so we will have to sum over the possible values of j_t for each student, weighted by the prior probabilities of types, $P(j_t)$ (same for all students):

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_n|\theta) = \prod_{t=1}^{n} \left[\sum_{j_t=1}^{m} P(\mathbf{x}_t|j_t)P(j_t) \right] = \prod_{t=1}^{n} \left[\sum_{j=1}^{m} P(\mathbf{x}_t|j)P(j) \right] \tag{12}$$

This is the likelihood of the observed data according to our mixture model. It is important to understand that the model would be very different if we exchanged the product and the sum in the above expression, i.e., define the model as

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_n|\theta) = \sum_{j=1}^{m} \left[\prod_{t=1}^{n} P(\mathbf{x}_t|j) \right] P(j) \tag{13}$$

This is also a mixture model but one that assumes that all students in the class are of specific single type, we just don't know which one, and are summing over the m possibilities (in the previous model there were m^n possible assignments of types over n students).