Algorithm-Independent Learning Issues

Selim Aksoy
Bilkent University
Department of Computer Engineering
saksoy@cs.bilkent.edu.tr

CS 551, Spring 2006

Introduction

- We have seen many learning algorithms and techniques for pattern recognition.
- Some of these algorithms may be preferred because of their lower computational complexity.
- Others may be preferred because they take into account some prior knowledge about the form of the data.
- Even though the Bayes error rate is the theoretical limit for classifier accuracy, it is rarely known in practice.

Introduction

- Given practical constraints such as finite training data, we have seen that no pattern classification method is inherently superior to any other.
- We will explore several ways to quantify and adjust the match between a learning algorithm and the problem it addresses.
- We will also study techniques for integrating the results of individual classifiers with the goal of improving the overall decision.

No Free Lunch Theorem

- Are there any reasons to prefer one classifier or learning algorithm over another?
- Can we even find an algorithm that is overall superior to random guessing?
- The *no free lunch theorem* states that the answer to these questions is "no".
- There are no context-independent or usage-independent reasons to favor one learning or classification method over another.

No Free Lunch Theorem

- If one algorithm seems to outperform another in a particular situation, it is a consequence of its fit to the particular pattern recognition problem, not the general superiority of the algorithm.
- It is the type of problem, prior distribution, and other information that determine which form of classifier should provide the best performance.
- Therefore, we should focus on important aspects such as prior information, data distribution, amount of training data, and cost functions.

Estimating and Comparing Classifiers

- To compare learning algorithms, we should use test data that are sampled independently, as in the training set, but with no overlap with the training set.
- Using the error on points not in the training set (also called the *off-training set error*) is important for evaluating the generalization ability of an algorithm.
- There are at least two reasons for requiring to know the generalization rate of a classifier on a given problem:
  - to see if the classifier performs well enough to be useful,
  - to compare its performance with that of a competing design.

Estimating and Comparing Classifiers

- Estimating the final generalization performance requires making assumptions about the classifier or the problem or both, and can fail if the assumptions are not valid.
- Occasionally, our assumptions are explicit (such as in parametric models), but more often they are implicit and difficult to identify or relate to the final estimation.
- We will study the following methods for evaluating classifiers:
  - Parametric models
  - Cross-validation
  - Jackknife and bootstrap estimation
  - Maximum likelihood model selection
  - Bayesian model selection
  - Minimum description length principle

Parametric Models

- One approach for estimating the generalization rate is to compute it from the assumed parametric model.
- Estimates for the probability of error can be computed using approximations such as the Bhattacharyya or Chernoff bounds.
- However, such estimates are often overly optimistic.
- Finally, it is often very difficult to compute the error rate exactly even if the probabilistic structure is known completely.

Cross-Validation

- In simple *cross-validation*, we randomly split the set of labeled training samples $\mathcal{D}$ into two parts.
- We use one set as the traditional training set for adjusting model parameters in the classifier, and the other set as the validation set to estimate the generalization error.
- Since our goal is to obtain low generalization error, we train the classifier until we reach a minimum of this validation error.

Cross-Validation

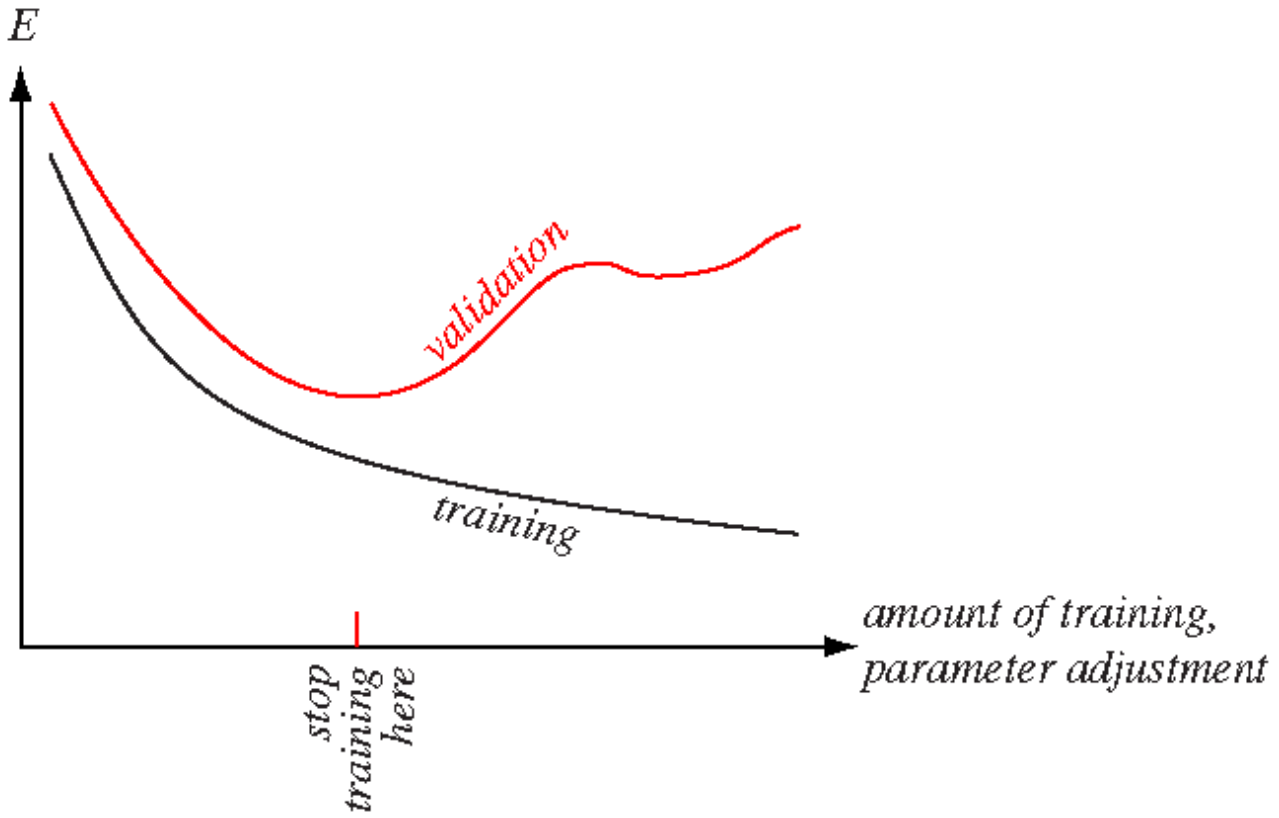


Figure 1: The data set $\mathcal{D}$ is split into two parts for validation. The first part (e.g., 90% of the patterns) is used as a standard training set for learning, the other (i.e., 10%) is used as the validation set. For most problems, the training error decreases monotonically during training. Typically, the error on the validation set decreases, but then increases, an indication that the classifier may be overfitting the training data. In validation, training is stopped at the first minimum of the validation error.

Cross-Validation

- A simple generalization of this method is *m-fold cross-validation* where the training set is randomly divided into m disjoint sets of equal size.
- Then, the classifier is trained m times, each time with a different set held out as a validation set.
- The estimated error is the average of these m errors.
- This technique can be applied to virtually every classification method, e.g., setting the number of hidden units in a neural network, finding the width of the Gaussian window in Parzen windows, choosing the value of k in the k-nearest neighbor classifier, etc.

Cross-Validation

- Once we train a classifier using cross-validation, the validation error gives an estimate of the accuracy of the final classifier on the unknown test set.
- If the true but unknown error rate of the classifier is p and if k of the n i.i.d. test samples are misclassified, then k has a binomial distribution, and the fraction of test samples misclassified is exactly the maximum likelihood estimate $\hat{p} = \frac{k}{n}$.
- The properties of this estimate for the parameter p of a binomial distribution are well known, and can be used to obtain confidence intervals as a function of the error estimate and the number of examples used.

Jackknife Estimation

- In the *jackknife* approach, we estimate the accuracy of a given algorithm by training the classifier n separate times, each time using the training set $\mathcal{D}$ for which a different single training point has been deleted.
- This is the $m = n$ limit of m-fold cross-validation, also called the *leave-one-out* estimate.
- Each resulting classifier is tested on the single deleted point, and the jackknife estimate of the accuracy is the average of these individual errors.

Bootstrap Estimation

- A *bootstrap* data set is created by randomly selecting n points from the training set $\mathcal{D}$, with replacement.
- Bootstrap estimation of classification accuracy consists of training m classifiers, each with a different bootstrap data set, and testing on other bootstrap data sets.
- The final estimate is the average of these bootstrap accuracies.

Maximum Likelihood Model Selection

- The goal of *maximum likelihood model selection* is to choose the model that best explains the training data.
- Let $\mathcal{M}_i$ represent a candidate model and let $\mathcal{D}$ represent the training data.
- The posterior probability of any given model can be computed using the Bayes rule

$$P(\mathcal{M}_i|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{M}_i)P(\mathcal{M}_i)}{p(\mathcal{D})} \propto P(\mathcal{D}|\mathcal{M}_i)P(\mathcal{M}_i)$$

where the data-dependent term $P(\mathcal{D}|\mathcal{M}_i)$ is the evidence for the particular model $\mathcal{M}_i$, and $P(\mathcal{M}_i)$ is our subjective prior over the space of all models.

Maximum Likelihood Model Selection

- In practice, the data-dependent term dominates and the prior is often neglected in the computation.
- Therefore, in maximum likelihood model selection, we find the maximum likelihood parameters for each of the candidate models, calculate the resulting likelihoods, and select the model with the largest such likelihood.

Maximum Likelihood Model Selection

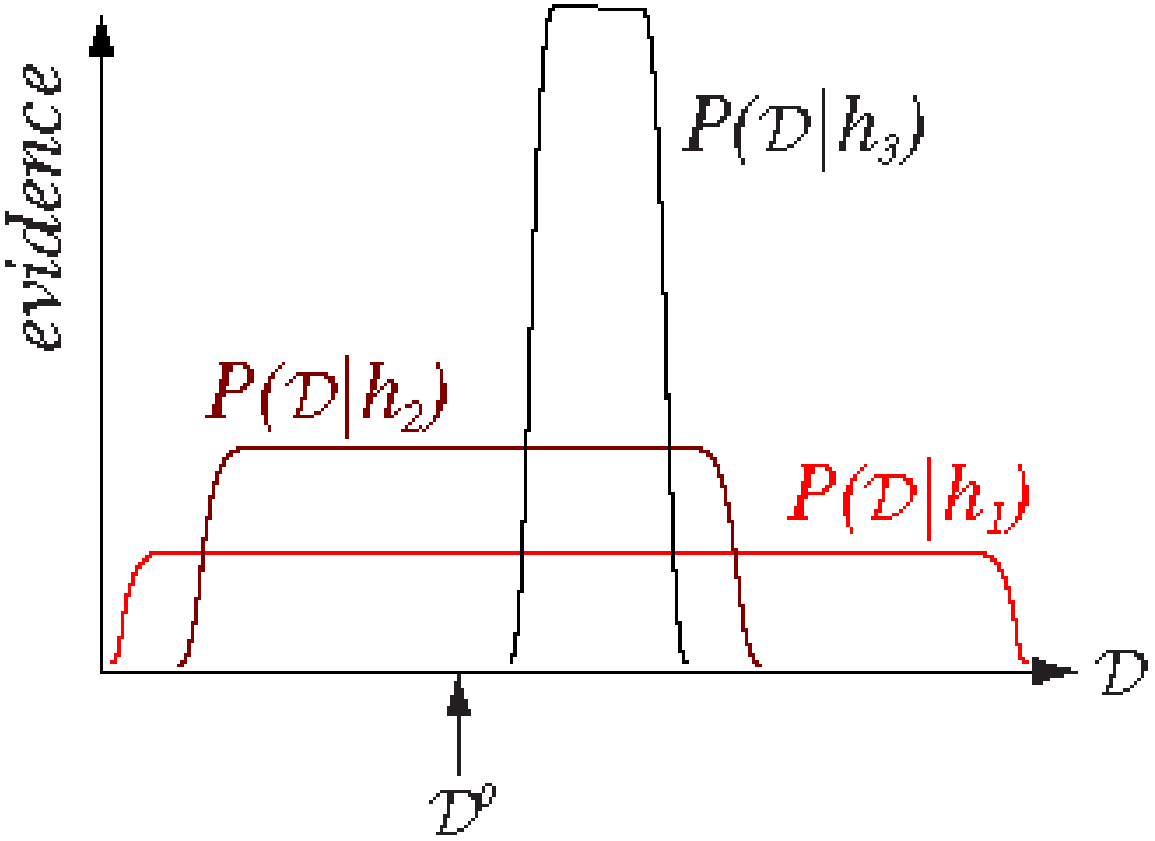


Figure 2: The evidence is shown for three models of different expressive power or complexity. Model h_1 is the most expressive and model h_3 is the most restrictive of the three. If the actual data observed is $\mathcal{D}^0$, then maximum likelihood model selection states that we should choose h_2, which has the highest evidence.

Bayesian Model Selection

- *Bayesian model selection* uses the full information over priors when computing the posterior probabilities $P(\mathcal{M}_i|\mathcal{D})$.
- In particular, the evidence for a particular model is the integral

$$P(\mathcal{D}|\mathcal{M}_i) = \int p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}_i)\, p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}_i)\, d\boldsymbol{\theta}$$

where $\boldsymbol{\theta}$ describes the parameters in the candidate model.

Bayesian Model Selection

- The posterior $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}_i)$ is often peaked at $\hat{\boldsymbol{\theta}}$ so the evidence integral can be approximated as

$$P(\mathcal{D}|\mathcal{M}_i) \simeq \underbrace{P(\mathcal{D}|\hat{\boldsymbol{\theta}}, \mathcal{M}_i)}_{\text{best-fit likelihood}} \underbrace{p(\hat{\boldsymbol{\theta}}|\mathcal{M}_i)\Delta\boldsymbol{\theta}}_{\text{Occam factor}}.$$

- The *Occam factor* is the ratio of the volume in the parameter space that corresponds to the prior range of model parameters $(\Delta^0\boldsymbol{\theta})$ and the volume that corresponds to the data $(\Delta\boldsymbol{\theta})$

$$\text{Occam factor} = p(\hat{\boldsymbol{\theta}}|\mathcal{M}_i)\Delta\boldsymbol{\theta} = \frac{\Delta\boldsymbol{\theta}}{\Delta^0\boldsymbol{\theta}}.$$

Bayesian Model Selection

- The Occam factor has magnitude less than 1, where it is the factor by which the model space collapses by the presence of data.
- The more the training data, the smaller the range of parameters that are proportionate with it, and thus the greater this collapse in the parameter space and the larger the Occam factor.
- Once the posteriors for different models have been calculated, we select the one having the highest such posterior.

Bayesian Model Selection

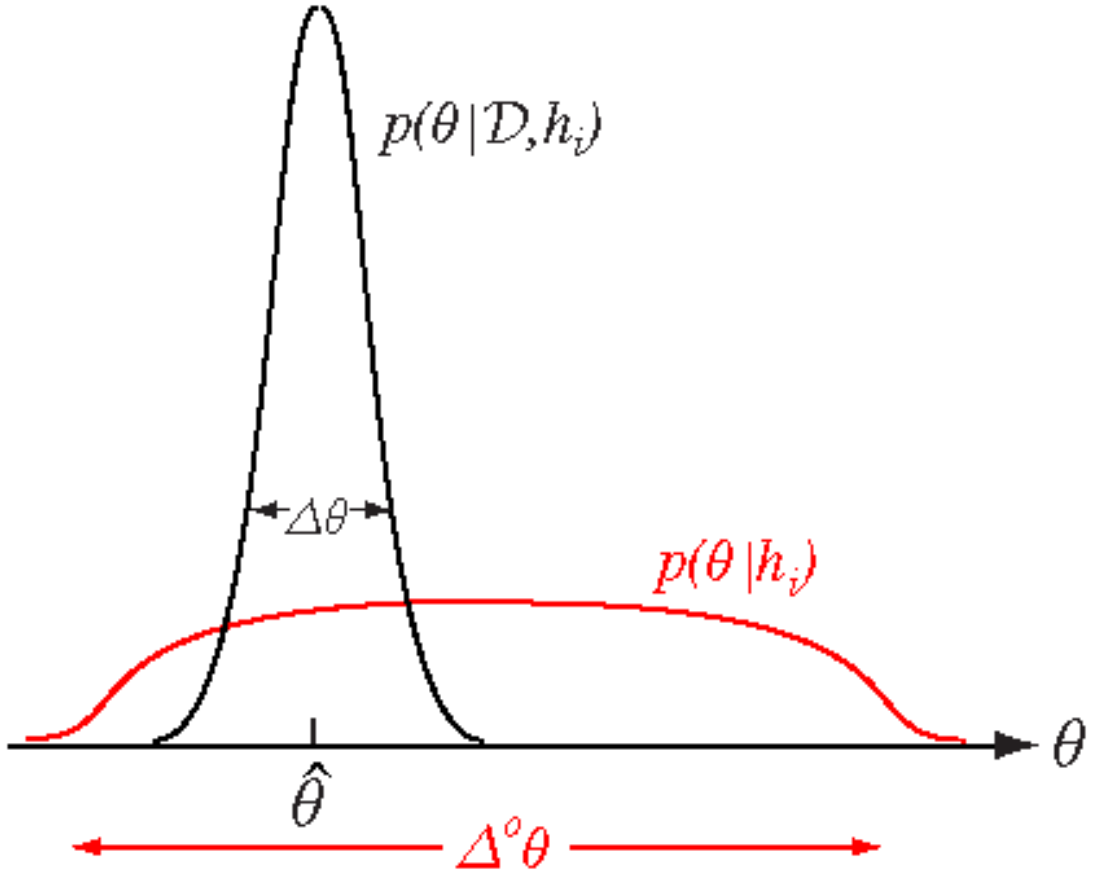


Figure 3: In the absence of training data, a particular model h_i has available a large range of possible values of its parameters, denoted $\Delta^0\boldsymbol{\theta}$. In the presence of a particular training set $\mathcal{D}$, a smaller range is available. The Occam factor, $\Delta\boldsymbol{\theta}/\Delta^0\boldsymbol{\theta}$, measures the fractional decrease in the volume of the model's parameter space due to the presence of training data $\mathcal{D}$.

Minimum Description Length Principle

- The *Minimum Description Length (MDL)* principle tries to find a compromise between the model complexity (still having a good data approximation) and the complexity of the data approximation (while using a simple model).
- Under the MDL principle, the best model is the one that minimizes the sum of the model's complexity $\mathcal{L}(\mathcal{M})$ and the efficiency of the description of the training data with respect to that model $\mathcal{L}(\mathcal{D}|\mathcal{M})$, i.e.,

$$\mathcal{L}(\mathcal{D}, \mathcal{M}) = \mathcal{L}(\mathcal{M}) + \mathcal{L}(\mathcal{D}|\mathcal{M}).$$

Minimum Description Length Principle

- According to Shannon, the shortest code-length to encode data $\mathcal{D}$ with a distribution $p(\mathcal{D}|\mathcal{M})$ under model $\mathcal{M}$ is given by

$$\mathcal{L}(\mathcal{D}|\mathcal{M}) = -\log L(\mathcal{M}|\mathcal{D}) = -\log p(\mathcal{D}|\mathcal{M})$$

  where $L(\mathcal{M}|\mathcal{D})$ is the likelihood function for model $\mathcal{M}$ given the sample $\mathcal{D}$.

- The model complexity is measured as the number of bits required to describe the model parameters.

- According to Rissanen, the code-length to encode $\kappa_{\mathcal{M}}$ real-valued parameters characterizing n data points is

$$\mathcal{L}(\mathcal{M}) = \frac{\kappa_{\mathcal{M}}}{2} \log n$$

  where $\kappa_{\mathcal{M}}$ is the number of free parameters in model $\mathcal{M}$ and n is the size of the sample used to estimate those parameters.

Minimum Description Length Principle

- Once the description lengths for different models have been calculated, we select the one having the smallest such length.
- It can be shown theoretically that classifiers designed with a minimum description length principle are guaranteed to converge to the ideal or true model in the limit of more and more data.

Minimum Description Length Principle

- As an example, let's derive the description lengths for Gaussian mixture models with m components.
- The total number of free parameters for different covariance matrix models are:

$$\boldsymbol{\Sigma_j} = \sigma^2\mathbf{I} \qquad \kappa_{\mathcal{M}} = (m-1) + md + 1$$

$$\boldsymbol{\Sigma_j} = \sigma_j^2\mathbf{I} \qquad \kappa_{\mathcal{M}} = (m-1) + md + m$$

$$\boldsymbol{\Sigma_j} = \text{diag}(\{\sigma_{jk}^2\}_{k=1}^d) \qquad \kappa_{\mathcal{M}} = (m-1) + md + md$$

$$\boldsymbol{\Sigma_j} = \boldsymbol{\Sigma} \qquad \kappa_{\mathcal{M}} = (m-1) + md + \frac{d(d+1)}{2}$$

$$\boldsymbol{\Sigma_j} = \text{arbitrary} \qquad \kappa_{\mathcal{M}} = (m-1) + md + m\frac{d(d+1)}{2}$$

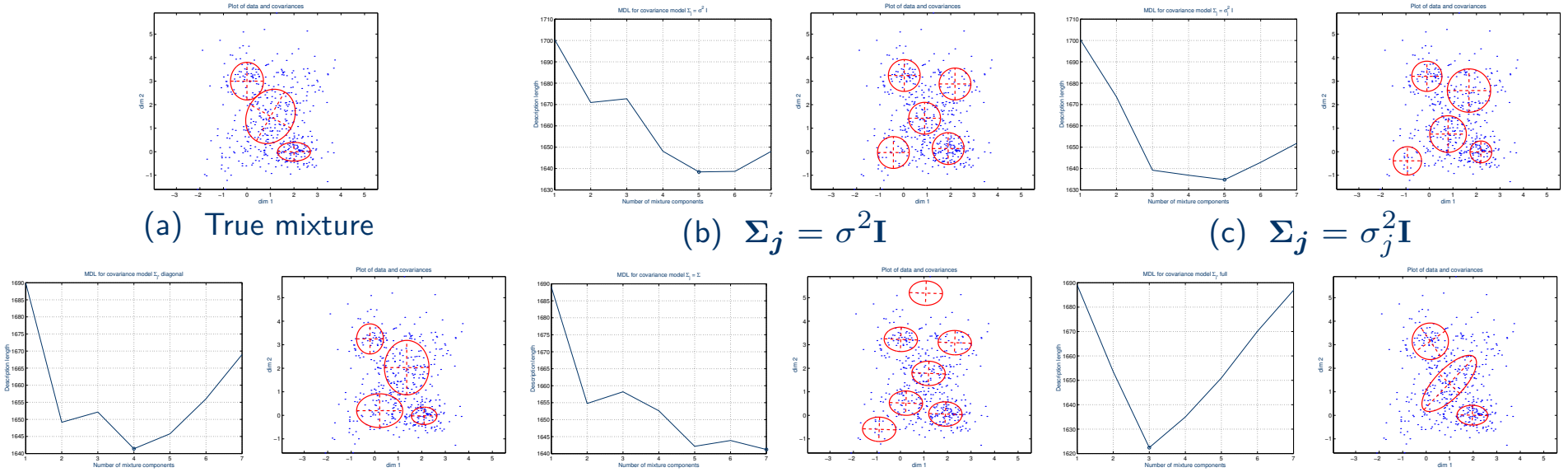
where d is the dimension of the feature vectors.

Minimum Description Length Principle

- The first term describes the mixture weights $\{\alpha_j\}_{j=1}^m$, the second term describes the means $\{\boldsymbol{\mu_j}\}_{j=1}^m$ and the third term describes the covariance matrices $\{\boldsymbol{\Sigma_j}\}_{j=1}^m$.
- Hence, the best m can be found as

$$m^* = \arg\min_m \left[\frac{\kappa_{\mathcal{M}}}{2} \log n - \sum_{i=1}^{n} \log \left(\sum_{j=1}^{m} \alpha_j p_j(\mathbf{x_i}|\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j}) \right) \right].$$

Minimum Description Length Principle



(a) True mixture (b) $\boldsymbol{\Sigma_j} = \sigma^2\mathbf{I}$ (c) $\boldsymbol{\Sigma_j} = \sigma_j^2\mathbf{I}$

(d) $\boldsymbol{\Sigma_j} = \text{diag}(\{\sigma_{jk}^2\}_{k=1}^q)$ (e) $\boldsymbol{\Sigma_j} = \boldsymbol{\Sigma}$ (f) $\boldsymbol{\Sigma_j} = \text{arbitrary}$

Figure 4: Example fits for a sample from a mixture of three bivariate Gaussians. For each covariance model, description length vs. the number of components (left) and fitted Gaussians as ellipses at one standard deviations (right) are shown. Using MDL with the arbitrary covariance matrix gave the smallest description length and also could capture the true number of components.

Combining Classifiers

- Just like different features capturing different properties of a pattern, different classifiers also capture different structures and relationships of these patterns in the feature space.
- An empirical comparison of different classifiers can help us choose one of them as the best classifier for the problem at hand.
- However, although most of the classifiers may have similar error rates, sets of patterns misclassified by different classifiers do not necessarily overlap.
- Not relying on a single decision but rather combining the advantages of different classifiers is intuitively promising to improve the overall accuracy of classification.
- Such combinations are variously called *combined classifiers*, *ensemble classifiers*, *mixture-of-expert models*, or *pooled classifiers*.

Combining Classifiers

- Some of the reasons for combining multiple classifiers to solve a given classification problem can be stated as follows:
  - Access to different classifiers, each developed in a different context and for an entirely different representation/description of the same problem.
  - Availability of multiple training sets, each collected at a different time or in a different environment, even may use different features.
  - Local performances of different classifiers where each classifier may have its own region in the feature space where it performs the best.
  - Different performances due to different initializations and randomness inherent in the training procedure.

Combining Classifiers

- In summary, we may have different feature sets, training sets, classification methods, and training sessions, all resulting in a set of classifiers whose outputs may be combined.
- Combination architectures can be grouped as:
  - *Parallel:* all classifiers are invoked independently and then their results are combined by a combiner.
  - *Serial (cascading):* individual classifiers are invoked in a linear sequence where the number of possible classes for a given pattern is gradually reduced.
  - *Hierarchical (tree):* individual classifiers are combined into a structure, which is similar to that of a decision tree, where the nodes are associated with the classifiers.

Combining Classifiers

- Selecting and training of individual classifiers:
  - Combination of classifiers is especially useful if the individual classifiers are largely independent.
  - This can be explicitly forced by using different training sets, different features and different classifiers.
- Combiner:
  - Some combiners are static, with no training required, while others are trainable.
  - Some are adaptive where the decisions of individual classifiers are evaluated (weighed) depending on the input pattern, whereas nonadaptive ones treat all input patterns the same.
  - Different combiners use different types of output from individual classifiers: confidence, rank, or abstract.

Combining Classifiers

- Examples of classifier combination schemes are:
  - Majority voting (each classifier makes a binary decision (vote) about each class and the final decision is made in favor of the class with the largest number of votes),
  - Sum, product, maximum, minimum and median of the posterior probabilities computed by individual classifiers,
  - Class ranking (each class receives m ranks from m classifiers, the highest (minimum) of these ranks is the final score for that class),
  - Borda count (sum of the number of classes ranked below a class by each classifier),
  - Weighted combination of classifiers.
- We will study different combination schemes using a Bayesian framework and resampling.

Bayesian Classifier Combination

- Given c classes $w_1, \ldots, w_c$ with prior probabilities $P(w_1), \ldots, P(w_c)$, and n feature vectors $\mathbf{x_1}, \ldots, \mathbf{x_n}$, the Bayesian classifier makes the decision using the posterior probabilities as

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} P(w_j|\mathbf{x_1}, \ldots, \mathbf{x_n})$$

$$= \arg\max_{j=1}^{c} p(\mathbf{x_1}, \ldots, \mathbf{x_n}|w_j)P(w_j).$$

- In a practical situation with limited training data, computing the joint probability density $p(\mathbf{x_1}, \ldots, \mathbf{x_n}|w_j)$ for each class will be difficult.
- However, we can train a separate classifier for each feature vector and make some assumptions to simplify the decision rule.

Bayesian Classifier Combination

- *Product rule:* Assuming that the features $\mathbf{x_1}, \ldots, \mathbf{x_n}$ are conditionally statistically independent given the class, the decision rule becomes

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \left[P(w_j) \prod_{i=1}^{n} p(\mathbf{x_i}|w_j) \right]$$

$$= \arg\max_{j=1}^{c} \left[(P(w_j))^{-(n-1)} \prod_{i=1}^{n} P(w_j|\mathbf{x_i}) \right]$$

  where $P(w_j|\mathbf{x_i})$ is the posterior probability output of the i'th classifier under class w_j.

- Under the assumption of equal priors, the rule becomes

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \prod_{i=1}^{n} P(w_j|\mathbf{x_i}).$$

Bayesian Classifier Combination

- *Sum rule:* Assuming that the posterior probabilities from individual classifiers will not deviate dramatically from the corresponding prior probabilities, they can be rewritten as

$$P(w_j|\mathbf{x_i}) = P(w_j)(1 + \epsilon_{ji})$$

  where $\epsilon_{ji} \ll 1$.

- Substituting this approximation in the product rule and neglecting any terms of second and higher order of ϵ_{ji} gives the decision rule

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \left[(1-n)P(w_j) + \sum_{i=1}^{n} P(w_j|\mathbf{x_i}) \right].$$

- Under the assumption of equal priors, the rule becomes

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \sum_{i=1}^{n} P(w_j|\mathbf{x_i}).$$

Bayesian Classifier Combination

- *Max rule:* Approximating the sum in the sum rule by the maximum of the posterior probabilities ($\frac{1}{n}\sum_{i=1}^{n} a_i \leq \max_{i=1}^{n} a_i$ for $a_i \in \mathbb{R}$) gives the decision rule

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \left[(1-n)P(w_j) + n \max_{i=1}^{n} P(w_j|\mathbf{x_i}) \right].$$

- Under the assumption of equal priors, the rule becomes

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \max_{i=1}^{n} P(w_j|\mathbf{x_i}).$$

Bayesian Classifier Combination

- *Min rule:* Approximating the product in the product rule by the minimum of the posterior probabilities ($\prod_{i=1}^{n} a_i \leq \min_{i=1}^{n} a_i$ for $0 \leq a_i \leq 1$) gives the decision rule

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \left[(P(w_j))^{-(n-1)} \min_{i=1}^{n} P(w_j|\mathbf{x_i}) \right].$$

- Under the assumption of equal priors, the rule becomes

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \min_{i=1}^{n} P(w_j|\mathbf{x_i}).$$

Bayesian Classifier Combination

- *Median rule:* Using the fact that median is a robust estimate of the mean, approximating the sum in the sum rule by the median of the posterior probabilities gives the decision rule

$$\text{assign to } w_k \text{ if } k = \arg \max_{j=1}^{c} \operatorname{median}_{i=1}^{n} P(w_j|\mathbf{x_i})$$

  under the assumption of equal priors.

- *Harmonic mean rule:* Another measure of central tendency is the harmonic mean, so replacing the sum in the sum rule by the harmonic mean gives the decision rule

$$\text{assign to } w_k \text{ if } k = \arg \max_{j=1}^{c} \left[\sum_{i=1}^{n} (P(w_j|\mathbf{x_i}))^{-1} \right]^{-1}$$

  under the assumption of equal priors. Note that this rule is valid when the posterior probabilities $P(w_j|\mathbf{x_i})$ are positive.

Bayesian Classifier Combination

- *Majority vote rule:* If we set each classifier to make a binary decision

$$\delta_{ki} = \begin{cases} 1 & \text{if } k = \arg\max_{j=1}^{c} P(w_j|\mathbf{x_i}) \\ 0 & \text{otherwise,} \end{cases}$$

approximating the sum in the sum rule by the individual binary decision outcomes gives the decision rule

$$\text{assign to } w_k \text{ if } k = \arg\max_{j=1}^{c} \sum_{i=1}^{n} \delta_{ji}$$

under the assumption of equal priors.

- The votes received by each class from individual classifiers are counted and the final decision is made in favor of the class with the largest number of votes.

Bayesian Classifier Combination

- All of these combination methods are based on the conditional independence assumption.
- Furthermore, some additional conditions need to be satisfied for the combination methods to improve the classification performance.
- For example, the individual classifiers should not be strongly correlated in their misclassification, i.e.,
  - they should not agree with each other when they misclassify a sample, or
  - at least they should not assign the same incorrect class to a particular sample.
- This requirement can be satisfied to a certain extent by using different feature vectors and different classifiers.

Bayesian Classifier Combination

- These combination algorithms require the output of individual classifiers to be probabilities.
- When the classifiers provide non-probabilistic output, these values can be normalized and used in place of the probabilities.
- *Analog output:* If the outputs of a classifier are analog values $g_j, j = 1, \ldots, c$, we can use the softmax transformation

$$P_j = \frac{e^{g_j}}{\sum_{i=1}^{c} e^{g_i}}$$

to convert them to probability values P_j.

Bayesian Classifier Combination

- *Rank order output:* If the output is a rank order list, the probability value can be assumed to be linearly proportional to the rank order of the pattern on the list (with proper normalization of the sum to 1.0).
- *One-of-c output:* If the output is a one-of-c representation, in which a single category is identified, $P_j = 1.0$ for the j corresponding to the chosen category, and 0.0 otherwise.

Resampling for Classifier Combination

- We have studied using resampling for generating training data and evaluating the accuracy of different classifiers.
- Resampling methods can also be used to build classifier ensembles.
- We will study:
  - *bagging*, where multiple classifiers are built by bootstrapping the original training set, and
  - *boosting*, where a sequence of classifiers is built by training each classifier using data sampled from a distribution derived from the empirical misclassification rate of the previous classifier.

Bagging

- *Bagging* ($b$ootstrap aggregating) uses multiple versions of the training set, each created by bootstrapping the original training data.
- Each of these bootstrap data sets is used to train a different component classifier.
- The final classification decision is based on the vote of each component classifier.
- Traditionally, the component classifiers are of the same general form (e.g., all neural networks, all decision trees, etc.) where their differences are in the final parameter values due to their different sets of training patterns.

Bagging

- A classifier/learning algorithm is informally called *unstable* if small changes in the training data lead to significantly different classifiers and relatively large changes in accuracy.
- Decision trees and neural networks are examples of unstable classifiers where a slight change in training patterns can result in radically different classifiers.
- In general, bagging improves recognition for unstable classifiers because it effectively averages over such discontinuities.

Boosting

- In *boosting*, each training pattern receives a weight that determines its probability of being selected for the training set for an individual component classifier.
- If a training pattern is accurately classified, its chance of being used again in a subsequent component classifier is reduced.
- Conversely, if the pattern is not accurately classified, its chance of being used again is increased.
- The final classification decision is based on the weighted sum of the votes of the component classifiers where the weight for each classifier is a function of its accuracy.

Boosting

- The popular *AdaBoost* (adaptive $boost$ing) algorithm allows continuous adding of classifiers until some desired low training error has been achieved.
- Let $\alpha^t(\mathbf{x}_i)$ denote the weight of pattern $\mathbf{x}_i$ at trial t, where $\alpha^1(\mathbf{x}_i) = 1/n$ for every $\mathbf{x}_i, i = 1, \ldots, n$.
- At each trial $t = 1, \ldots, T$, a classifier $\mathcal{C}^t$ is constructed from the given patterns under the distribution α^t where $\alpha^t(\mathbf{x}_i)$ of pattern $\mathbf{x}_i$ reflects its probability of occurrence.
- The error ϵ^t of this classifier is also measured with respect to the weights, and consists of the sum of the weights of the patterns that it misclassifies.

Boosting

- If ϵ^t is greater than 0.5, the trials terminate and T is set to $t-1$.
- Conversely, if $\mathcal{C}^t$ correctly classifies all patterns so that ϵ^t is zero, the trials also terminate and T becomes t.
- Otherwise, the weights α^{t+1} for the next trial are generated by multiplying the weights of patterns that $\mathcal{C}^t$ classifies correctly by the factor $\beta^t = \epsilon^t/(1-\epsilon^t)$ and then are renormalized so that $\sum_{i=1}^{n} \alpha^{t+1}(\mathbf{x}_i) = 1$.
- The boosted classifier $\mathcal{C}^*$ is obtained by summing the votes of the classifiers $\mathcal{C}^1, \ldots, \mathcal{C}^T$, where the vote for classifier $\mathcal{C}^t$ is also weighted by $\log(1/\beta^t)$.

Boosting

- Provided that ϵ^t is always less than 0.5, it was shown that the error rate of $\mathcal{C}^*$ on the given patterns under the original uniform distribution α^1 approaches zero exponentially quickly as T increases.
- A succession of *weak* classifiers $\{\mathcal{C}^t\}$ can thus be boosted to a strong classifier $\mathcal{C}^*$ that is at least as accurate as, and usually much more accurate than, the best weak classifier on the training data.
- However, note that there is no guarantee of the generalization performance of a bagged or boosted classifier on unseen patterns.

Combining Classifiers

Table 1: Classifier combination methods.

Scheme	Architecture	Trainable	Adaptive	Info-level	Comments
Voting	Parallel	No	No	Abstract	Assumes independent classifiers
Sum, mean, median	Parallel	No	No	Confidence	Robust; assumes independent confidence estimators
Product, min, max	Parallel	No	No	Confidence	Assumes independent features
Generalized ensemble	Parallel	Yes	No	Confidence	Considers error correlation
Adaptive weighting	Parallel	Yes	Yes	Confidence	Explores local expertise
Stacking	Parallel	Yes	No	Confidence	Good utilization of training data
Borda count	Parallel	Yes	No	Rank	Converts ranks into confidences
Logistic regression	Parallel	Yes	No	Rank confidence	Converts ranks into confidences
Class set reduction	Parallel cascading	Yes / No	No	Rank confidence	Efficient
Dempster-Shafer	Parallel	Yes	No	Rank confidence	Fuses non-probabilistic confidences
Fuzzy integrals	Parallel	Yes	No	Confidence	Fuses non-probabilistic confidences
Mixture of local experts (MLE)	Gated parallel	Yes	Yes	Confidence	Explores local expertise; joint optimization
Hierarchical MLE	Gated parallel hierarchical	Yes	Yes	Confidence	Same as MLE; hierarchical
Associative switch	Parallel	Yes	Yes	Abstract	Same as MLE, but no joint optimization
Bagging	Parallel	Yes	No	Confidence	Needs many comparable classifiers
Boosting	Parallel hierarchical	Yes	No	Abstract	Improves margins; unlikely to overtrain; sensitive to mislabels; needs many comparable classifiers
Random subspace	Parallel	Yes	No	Confidence	Needs many comparable classifiers
Neural tree	Hierarchical	Yes	No	Confidence	Handles large numbers of classes