Classifier Performance

Assessment and Improvement
Error Rates

Define the Error Rate function

\[ Q(\hat{\omega}, \omega) = \delta(\hat{\omega} - \omega) = \begin{cases} 1 & \text{if } \hat{\omega} - \omega = 0 \\ 0 & \text{otherwise} \end{cases} \]

- When training a classifier, the Apparent error rate (or Test Error) is:

\[ E_{test} = \frac{1}{N} \sum_{j=1:N} Q(\hat{\omega}_j(\{x_i, \omega_i\}, x_j), \omega_j) \]

where \( x_i \) are training features, and \( \omega_i \) training labels

- We want a general way to assess the Generalization error (or True error rate)

\[ E_{true} = \sum_{\omega} \int p(x, \omega) Q(\hat{\omega}(x, \omega), \omega) \, dx \]
Learning as Empirical Risk Minimization

Optimal decisions can be formulated as minimizing the theoretical Risk
\[
R(\hat{\omega}) = \int \sum_{j=1:C} L(\hat{\omega}(x) | \omega_j) p(\omega_j, x) \, dx
\]

But now our decision function is determined by parameters \( \alpha \)
\[
\hat{\omega}(x) = g(x; \alpha) \quad [\text{e.g. } \alpha \text{ is the weight vector for Perceptron}]
\]

This induces a loss function on \( \alpha \)
\[
R(\alpha) = \int \sum_{j=1:C} L(\hat{\omega}(x; \alpha) | \omega_j) P(\omega_j | x) p(x) \, dx
\]
\[
\hat{\omega}(x) = g(x; \alpha) \quad [\text{e.g. } \alpha \text{ is the weight vector for Perceptron}]
\]

Two problems:

1. **TRUE** \( P(\omega_j, x) \) is unknown
2. The estimate \( \alpha(D) \) is typically a complicated random var. across test data sets, thus the Risk is too.
Learning as Empirical Risk Minimization

Goal: \[ R(\alpha) = \int \sum_{j=1:C} L(\omega(x;\alpha) | \omega_j) p(\omega_j, x) \, dx \]

Problem: What we are faced with is an estimate from data:
\[ R_{est}(\hat{\alpha}_D) = \int \sum_{j=1:C} L(\omega(x;\hat{\alpha}_D) | \omega_j) \hat{p}_D(\omega_j, x) \, dx \]

Where \( \hat{p}_D(\omega_j, x) \) is an estimate of the true distribution based on data \( D \), and \( \hat{\alpha}_D \) is an estimate of the model parameters. 

\[ R_{est}(\hat{\alpha}_D) \] will be termed the **Estimated Risk** (not standard terminology)
Interpreting Minimizing Training Error

\[ R(\alpha) = \int \sum_{j=1:C} L(\hat{\omega}(x;\alpha) \mid \omega_j) p(\omega_j,x) \, dx \]

Derive

Replace the probability dist. with the sample:

\[ p(\omega_j,x) = \sum_{i=1:N} \delta(x - x_i) \delta(\omega_j - \omega_j^i) \]

Observed Risk

Substituting in above (after integrating and summing):

\[ R_{emp}(\alpha) = R_{obs}(\alpha) = \frac{1}{N} \sum_{i=1:N} L(\hat{\omega}(x_i;\alpha) \mid \omega_j^i) \]

When the loss is making an error:

\[ L(\hat{\omega}(x_i;\alpha) \mid \omega_j) = \delta(\hat{\omega}(x_i;\alpha) - \omega_j) = \begin{cases} 1 \text{ if } \hat{\omega}(x_i;\alpha) = \omega_j^i \\ 0 \text{ else} \end{cases} \]

Observed Risk for 0-1 Loss is Error Rate

\[ R_{obs}(\alpha) = \frac{1}{N} \sum_{i=1:N} \delta(\hat{\omega}(x_i;\alpha) - \omega_j) = \# \text{ correct} / \text{Total} \]
**Implication:** Minimizing the training error is equivalent to modeling $p(x, \omega)$ *as a sum of delta functions*

Data is sampled from some unknown distribution

Samples form empirical distribution

$$F(x) = \sum_{i=1:N} \delta(x - x_i)$$
Need better estimator for the Empirical Risk

Improvement comes from the way we use the data:

1. Parameter Estimate
   - Approximation vs. estimation error
   - Depends implicitly on:

2. Probability Density Estimate
   - Resampling statistics: Methods to provide better estimates of the actual Risk by resampling from data.
     - Bootstrap
     - Jackknife
     - Cross-validation
   - Theoretical bounds: Based on better approximations to the true data distribution
     - Support vector machines
Improving Parameter Estimates

**Goal:** Parameter estimates with low

- **Estimation error (VARIANCE):** How far is the estimate from minimizing the Empirical Risk

- **Approximation error (BIAS):** How does the model constrain the best estimate from optimizing the true Risk
FIGURE 1.5. Overly complex models for the fish will lead to decision boundaries that are complicated. While such a decision may lead to perfect classification of our training samples, it would lead to poor performance on future patterns. The novel test point marked ? is evidently most likely a salmon, whereas the complex decision boundary shown leads it to be classified as a sea bass. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.
Bias-Variance Trade off

• For Regression (fitting continuous functions to data), there is a well known trade off in fit quality.

• **Bias**: For models with few parameters, there is a large *approximation error*. The true function is less likely to be in the set achievable via the model.

• **Variance**: For models with many parameters, there is a large *estimation error*. Test error variance across test samples will be finite.
Trade-off:

Complex $\mathcal{F}$  overfitting
Simple $\mathcal{F}$  underfitting

Objective:  Find ‘best balance’ between the two
Split error into Bias + Variance
Unbiased estimator: $\text{bias} = 0$.

<table>
<thead>
<tr>
<th>complex $\mathcal{F}$</th>
<th>low bias, high variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple $\mathcal{F}$</td>
<td>high bias, low variance</td>
</tr>
</tbody>
</table>

Objective: minimize bias and variance simultaneously - usually impossible.

Rule of thumb: The variance is the main problem for small training sets and complex $\mathcal{F}$. 
Bias-Variance for Classifiers

\[ \Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & \sigma_{i12} \\ \sigma_{i21} & \sigma_{i2}^2 \end{pmatrix} \quad \Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & 0 \\ 0 & \sigma_{i2}^2 \end{pmatrix} \quad \Sigma_i = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \]
Bias-Variance Decomp. for Classifiers

Dichotomy not as clear for classifiers as regression. Topic of current research. Recent result:

For training sets $D = \{D_1, \ldots, D_M\}$

Main Estimate:

$$\hat{\omega}_m(D) = \arg\min_\omega E_D[L(\hat{\omega}, \omega)]$$

Est. based on many Data sets

‘Bias’

$$B(x_i) = L(\hat{\omega}_m(x_i), \omega_{\text{best}}(x_i))$$

Error from to best estimate

‘Variance’

$$V(x_i) = E_D[L(\hat{\omega}_m(x_i), \hat{\omega}(x_i))]$$

Error from single Data set est. to main

‘Noise’

$$N(x_i) = E_\omega[L(\omega_{\text{best}}(x_i), \omega_{\text{true}})]$$

Errors you can’t avoid

Error Decomp:

$$\mathbb{E}[Error(x_i)] = c_1 N(x_i) + B(x_i) + c_2 V(x_i)$$

$$\omega_{\text{best}}(x_i) = \text{Optimal Prediction}$$

Schematic of bias and variance.

The model space is the set of all possible predictions from the model, with the "closest fit" labeled with a black dot.

The model bias from the truth is shown, along with the variance, indicated by the large yellow circle centered at the black dot labeled "closest fit in population". A shrunken or regularized fit is also shown, having more est. bias, but smaller prediction error due to its decreased variance.
Solutions

Given enough data, split it:

- test
- validation
- training

If not enough data, estimate generalization performance
- analytically (Akaike information criterion, etc.)
- by reusing samples:
  - cross-validation
  - bootstrap
Roadmap

• Methods to estimate generalization error from training sample.

• Methods for improving classifier performance:
  – Choosing minimal gen. Error rate classifier
  – Committee decisions (voting)
  – Averaging classifiers
  – Bagging
  – Boosting
Cross-Validation

simple and powerful

- Partition data into $K$ pieces. Predict class membership in one piece by training on all others. Repeat.

- Typically, $K = 5\ldots10$.

- For $K = N$ (# training feature vectors), leave-one-out (LOO) cross-validation results.

- Choosing $K \ll N$ is faster and has lower variance but more bias$, especially for small $N$. 
Bootstrap

- Sample from training set with replacement. Train a classifier on that set, and calculate “leave-one-out prediction error” on samples that are not in the set.
- Repeat 50-100 times, and estimate parameters

**The .632 estimator for prediction error**

Compute

\[(1 - 0.632) \times \text{(training error)} + 0.632 \times \text{(leave-one-out prediction error)}\]

- balances optimism (from inclusion of training error) with pessimism (because not all of the data is used in training)
- works well as long as there is no overfitting
Bootstrap

**Basic idea:** Generate artificial data in place of unknown distribution.

1. Generate $K$ samples, each of size $n$, drawn
   *with replacement* from the $n$ training observations. Set
   
   $$
   D_{b}^{(k)} = \{(X_1^{(*k)}, Y_1^{(*k)}), \ldots, (X_n^{(*k)}, Y_n^{(*k)})\}
   $$

2. For each bootstrap sample $k$, let $D_{b,v}^{(k)}$ include
   the training samples which are *not* included in $D_{b}^{(k)}$.

3. Compute the decision rule $\hat{\alpha}^{(k)}$ for each
   bootstrap sample $D_{b}^{(k)}$. 
4. Evaluate the validation-error for sample $k$:

$$\tilde{L}_b^{(k)} = \frac{1}{n_v^{(k)}} \sum_{\ell \in D_v^{(k)}} \lambda(\hat{\alpha}^{(k)}(X_{\ell}) | Y_{\ell}).$$

5. Average the results over all $K$ bootstrap samples, namely

$$\tilde{L}_b(\hat{\alpha}_n) = \frac{1}{K} \sum_{k=1}^{K} \tilde{L}_b^{(k)}(\hat{\alpha}^{(k)}).$$
1. Choose BEST
2. Committee vote
3. Average
Examples of bagging unstable and stable classifier (Breiman, 1996)

Single and Bagged Decision Trees (50 Bootstrap Replicates)
Test Set Average Misclassification Rates over 100 Runs

Single and Bagged k-NN (100 Bootstrap Replicates)
Test Set Average Misclassification Rates over 100 Runs
Combining Classifiers

• **Goal:** generate a set of simple “weak” classification methods and combine them into a single “strong” method

• **Solution:** Average multiple classifiers by estimate of their reliability.

Combine the discriminant functions additively so that the final classifier is the sign of

\[
\hat{g}_m(x) = \alpha_1 h_1(x; \hat{w}_1) + \cdots + \alpha_m h_m(x; \hat{w}_m)
\]

where the “votes” \(\alpha_i\) emphasize component classifiers that make more reliable predictions than others
Adaboost

**Final Classifier**

\[ G(x) = \text{sign} \left[ \sum_{m=1}^{M} \alpha_m G_m(x) \right] \]

- **Weighted Sample** \( G_M(x) \)
- **Weighted Sample** \( G_3(x) \)
- **Weighted Sample** \( G_2(x) \)
- **Training Sample** \( G_1(x) \)

**FIGURE 10.1.** Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.
General Algorithm

1: initialize weights $w_i^t = 1/n$ on each sample $x_i$
2: for each round $t$ do
3: select classifier $f_t$ that minimizes the weighted empirical loss on the reweighted training set
4: compute relative ensemble weight $\alpha_t$ for this classifier
5: update distribution $w_i^{t+1}$ by increasing/decreasing weights of incorrectly/correctly classified examples
6: end for
7: output classifier $\text{sign} \sum_{t=1}^{T} \alpha_t f_t(\cdot)$
Algorithm 1 (AdaBoost)

1. begin initialize \( D = \{x^1, y_1, x^2, y_2, \ldots, x^n, y_n\} \), \( k_{max} \), \( W_1(i) = 1/n, i = 1, \ldots, n \)
2. \( k \leftarrow 0 \)
3. \( \textbf{do} \) \( k \leftarrow k + 1 \)
4. Train weak learner \( C_k \) using \( D \) sampled according to distribution \( W_k(i) \)
5. \( E_k \leftarrow \text{Training error of } C_k \text{ measured on } D \text{ using } W_k(i) \)
6. \( \alpha_k \leftarrow \frac{1}{2} \ln[(1 - E_k)/E_k] \)
7. \( W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} \times \left\{ \begin{array}{ll} e^{-\alpha_k} & \text{if } h_k(x^i) = y_i \text{ (correctly classified)} \\ e^{\alpha_k} & \text{if } h_k(x^i) \neq y_i \text{ (incorrectly classified)} \end{array} \right. \)
8. \( \textbf{until} \ k = k_{max} \)
9. return \( C_k \) and \( \alpha_k \) for \( k = 1 \) to \( k_{max} \) (ensemble of classifiers with weights)
10. end
Adaboost Example

- The simple classifiers in our case are decision stumps:

  \[ h(x; \theta) = \text{sign}(w_1 x_k - w_0) \]

  where \( \theta = \{k, w_1, w_0\} \).

  Each decision stump pays attention to only a single component of the input vector.
Boosting: example
FIGURE 8.11. Data with two features and two classes, separated by a linear boundary. Left panel: decision boundary estimated from bagging the decision rule from a single split, axis-oriented classifier. Right panel: decision boundary from boosting the decision rule of the same classifier. The test error rates are 0.166, and 0.065 respectively. Boosting is described in Chapter 10.
Adaboost Summary

- Basic AdaBoost: Combine weak classifiers to make a strong classifier.
- Dynamically weight the data, so that misclassified data weighs more (like SVM pay more attention to hard-to-classify data).
- Exponential convergence to empirical risk (weak conditions).