**Weighted Majority**

- Assume $K$ experts $f_1, f_2, \ldots, f_K$ (base learners)
- Each expert makes a decision $f_k(x) = \pm 1$
- Improve predictions by making the experts “vote” according to how good they are:
  \[
  F(x) = \sum_k \alpha_k f_k(x)
  \]
  \[
  0 \leq \alpha_k \leq 1
  \]
  \[
  \sum_k \alpha_k = 1
  \]
- Decision: sign[$F(x)$]

**Mixture of Experts/Committee**

\[
F(x) = \sum_k \alpha_k f_k(x)
\]
**Simple Examples**

- **Kernel methods:**
  - Each example is an “expert”
    \[ f_i(x) = y_i k(x, x_i) \]
- **Decision stumps:**
  - Each variable is an “expert”
    \[ f_i(x) = x_i \quad \text{("orient" variables s.t. } x_1, x_2, \ldots, x_m, y > 0) \]
  - Each feature is an “expert”
    \[ f_j(x) = \phi_j(x) \]

---

**Bias-variance tradeoff**


- **For the square loss:**
  \[
  E_D[f(x,D)-y]^2 = [E_D[f(x,D)-y]]^2 + E_D[f(x,D)-E_D[f(x,D)]]^2
  \]

**Bias:** \[ E_D[f(x,D)-y]^2 \]

- \( E_D[f(x)] \) : your “ideal” committee machine.
- **Bias**: what your ideal committee can’t learn (from m training examples)
- \( E_D[f(x)] \) has the **same bias** as \( E_D[f(x)] \) but **no variance**.
- Note: Each committee member was trained on a different set on m examples…

**Variance:** \[ E_D[f(x,D)-E_D[f(x,D)]]^2 \]

- \( E_D[f(x)] \) : your “ideal” committee machine.
- **Variance**: how far apart on average your solution \( f(x,D) \) is from your “ideal” committee machine.
- If the **variance is high** but the **bias is low**: there is hope that a committee can improve performance.
- Note: Subsampling introduces extra bias …
Feature Selection

1) Mixture of decision stumps ⇒ feature selection
2) Merging expert feature rankings:
   - Average ranking index: \( C_j = \sum_k \alpha_k C_{jk} \)
   - Average rank: \( C_j = \sum_k \alpha_k (R_{\text{max}} - R_{jk}) \)
3) Merging feature sets selected by experts:
   - Ranking index: \( C_j = \sum_k \alpha_k \delta_{jk} \) (\( \delta_{jk} = 1 \) if feat \( j \) selected by expert \( k \), 0 otherwise)
   - \( S^* = \arg\max_k \min_{k'} |S_k \cap S_{k'}| \) (most 'stable' subset)
   - \( R^* = \arg\min_k \max_{k'} \text{dist}(R_k, R_{k'}) \)
   - [http://people.revoledu.com/kardi/tutorial/Similarity/OrdinalVariables.html](http://people.revoledu.com/kardi/tutorial/Similarity/OrdinalVariables.html)
4) Sensitivity-based (special for bagging)

Bayesian Approach

Simple Justification

\[
F(x) = \sum_k \alpha_k f_k(x), \quad 0 \leq \alpha_k \leq 1, \quad \sum_k \alpha_k = 1
\]

- \( P(y|x,D) = \sum_l P(l|D) P(y|x,f_l,D) \)
- Individual "expert" decisions: \( P(y|x,f_l,D) \)
- Weights: \( P(l|D) \)

Risk=negative log posterior: \( P(l|D) \propto 1 - R_{\text{emp}}[l] \)

\[ P(l|D) \propto \exp(-R_{\text{emp}}[l]/T) \]

Success rate:

\[ P(l|D) \propto \exp(-R_{\text{emp}}[l]/T) \exp(-\lambda[l]/T) \]

Bayesian Methods

- \( P(y|x,D) = \sum_l P(l|D) P(y|x,f_l,D) \)
- \( P(y|x,D) = \int f P(f|D) P(y|x,f,D) \) df

\[ P(f^*|D) \]

MAP approximation

\[ P(y|x,D) = P(y|x,f^*,D) \]

\[ P(f^*|D) \Delta f = 1 \]
**Difficulties**

- **Continuous case:**
  Infinitely many experts, we can’t try them all!
- **Idea:**
  Let’s take a sample…
- **How?**
  Grid, heuristic search, stochastic search
- **Important:**
  Avoid sampling “poor” experts or “redundant” experts.

**Iterative Sampling**

**MCMC**

- $P(f|D) \propto \exp(-R[f]/T)$
- Simulated annealing:
  - Make a random step
  - Accept with proba $\exp(-R[f]/T)$
  - Progressively decrease $T$
  
  (Metropolis-Hasting, 1953-1970)

- Gibbs sampling:
  - Investigate a bunch of nearby solutions
  - Sample according to $\text{local}_\text{sum} \exp(-R[f]/T)$
  - Start over from new point
  
  (Geman-Geman, 1984)

**Variable-dimension MCMC**

- Some steps include removal or addition of a feature
- We obtain $P(\text{model}, \text{feature-subset}|D)$ for some samples of models and feature subsets
- Subset relevance can be computed by marginalization (averaging over the functions using the same subset)
- Feature relevance can also be computed by marginalization (averaging over all subsets containing that feature)
**Performance Gain?**

- If we draw $M$ classifiers $f_k$ according to $P(f|D)$, we can approximate $P(y|x,D) = \int P(f|D) P(y|x,f,D) \, df$ by $P(y|x,D) \sim \sum_{k=1}^{M} P(y|x,f_k,D)$
- Relative error difference with optimum Bayes classifier decays with $O(1/M)$ (Ng, Jordan, 2001)

---

**Non-Bayesian Approaches**

- Parallel ensembles: bagging
- Serial ensembles: boosting

---

**Bagging**

*Breiman, 1996*

- Bootstrap Aggregation:
  - Draw with replacement $m$ samples from the original training set of size $m$
  - Train a learning machine
  - Repeat many times
  - On average, each example appears in the training set $(1-1/m)^m \approx 1 - e^{-1/2} \approx 0.632$ times

---

**Random Forests**

*Breiman, 2001*

1. A number $n$ is specified much smaller than the total number $N$ of variables (typically $n \approx \sqrt{N}$)
2. Each tree of maximum depth is grown on a bootstrap sample of the training set
3. At each node, $n$ variables are selected at random out of the $N$
4. The split used is the best split on these $n$ variables
Tree Classifiers

CART (Breiman, 1984) or C4.5 (Quinlan, 1993)

At each step, choose the feature that "reduces entropy" most. Work towards "node purity".

Information Gain

Choose \( f_2 \)

\[
H_{\text{before}} = -\frac{11}{19} \log_2 \left( \frac{11}{19} \right) - \frac{8}{19} \log_2 \left( \frac{8}{19} \right) = 0.98
\]

\[
H_{\text{left}} = -\frac{4}{11} \log_2 \left( \frac{4}{11} \right) - \frac{7}{11} \log_2 \left( \frac{7}{11} \right) = 0.94
\]

\[
IG = -\frac{11}{19} H_{\text{left}} + \frac{8}{19} H_{\text{right}} = 0.98 - 0.78 = 0.2
\]

Embedding Variable Scoring

- \( IG_t(f) = \) Information gain due to splitting with feature \( f \) at node \( t \)
- Ranking index: \( R(f) = \sum_t IG_t(f) \)
- Surrogate variables (detect masking)
- Use of M trees:
  \[
  R(f) = \sum_T \sum_{t \in T} IG_t(f)
  \]

Iris Data
Sensitivity-based Scoring

Breiman, 2001

• Classify the OOB cases and count the number of votes cast for the correct class in every
tree grown in the forest
• Randomly permute the values of feature \( f \) in the OOB cases and classify these cases
down the trees
• Subtract the number of votes for the correct class in the feature-\( f \) permuted OOB data
  from the untouched OOB data
• Average this number over all trees in the forest to obtain the importance score \( R(f) \)

Cross-validated Committee

Parmanto et al., 1996

• Any learning machine
• Any method of splitting the (training) data many times into training set and validation set (vset)
• Perturb feature \( f \) randomly in vset (pvset)
• \( R(f) = \text{mean} [ \text{num-correct-class(vset)} - \text{num-correct-class(pvset)}] \)
• \( Z\text{score} = R(f)/\text{stderr} \)

Boosting

• Adaboost (Freund and Schapire, 1996): At every step add a new base learner that is forced (by re-weighting the training data) to concentrate on misclassified examples.
• Forward stagewise boosting (Breiman, 1997, Friedman et al., 2000)
  1. Initialize \( F(x)=0 \)
  2. For \( k=1 \) to \( M \)
      \( F(x) \leftarrow F(x) + \alpha f(x) \)
      \( (\alpha_k, f_k) = \text{argmin}_{\alpha, f} \sum_i \exp(-y_i F(x_i)) \)
  3. Output \( F(x) = \sum_{k=1:M} \alpha_k f_k(x) \)

Loss Functions

- 0/1 loss
- Square loss \( (1-z)^2 \)
- SVC loss, \( \beta=1 \)
  \( \max(0, 1-z) \)
- Logistic loss
  \( \log(1+e^{-z}) \)
- Adaboost loss
  \( e^{-z} \)
- Perceptron loss
  \( \max(0, -z) \)
- SVC loss, \( \beta=2 \)
  \( \max(0, (1-z)^2) \)
- Square loss
  \( (1-z)^2 \)
**Conclusion**

- Ensemble methods help reducing the "variance"
- They benefit most to "low bias" base learners
- One should not confuse feature set variability and variance in predictions
- CV committees allow to rank features according to sensitivity ans compute zscores.

**Exercise Class**

Arcene
Boosting

---

**Arcene**

Best performances: 11.9% ± 1.2 (use training set + validation set in training) with 11% of the features (1100 features)

Baseline model: 14.7% ±1.4 1100 features

```matlab
my_svc=svc({'coef0=1', 'degree=3', 'gamma=0', 'shrinkage=0.1'});
my_model=chain({standardize, s2n('f_max=1100'), normalize, my_svc})
```

**Forward Stagewise Boosting**

1. Initialize \( F(x) = 0 \)
2. For \( k=1 \) to \( M \)
   \[ F(x) \leftarrow F(x) + \alpha f(x) \]
   \( (\alpha_k, f_k) = \text{argmin}_{\alpha, f} \sum_i \exp[-y_i (F_{t-1}(x_i) + \alpha f(x_i))] \)
3. Output \( F(x) = \sum_{k=1}^M \alpha_k f_k(x) \)

At step \( t \):
\[
(\alpha_k, f_k) = \text{argmin}_{\alpha, f} \sum_i \exp[-y_i (F_t(x_i) + \alpha f(x_i))] 
\]
Compute \( \alpha_k, f_k \) for decision stumps
\[ \alpha = \frac{1}{2} \log \left( \frac{1 - E(f)}{E(f)} \right) \]

**Adaboost**

- Initialize \( \alpha_i = \frac{1}{2} \) for all \( i = 1, \ldots, N \).
- For every round \( r = 1, \ldots, T \) do:
  - For every instance \( (x_i, y_i) \) do:
    - Minimize the weighted error of the current model \( f_r(x_i) \):
      \[ E_r = \sum_{i=1}^{N} w_i \mathbb{I}[f_r(x_i) \neq y_i] \]
      \[ w_i \text{ is the weight of instance } x_i \]
      \[ \text{end for} \]
  - Compute the weight update \( \Delta w_i \):
    \[ \Delta w_i = \frac{w_i}{\sum_{i=1}^{N} w_i} \exp \left( -2 \alpha_r y_i f_r(x_i) \right) \]
    \[ \text{end for} \]
  - Update the weights:
    \[ w_i^{r+1} = w_i^r \exp \left( \Delta w_i \right) \]
    \[ \text{end for} \]
- Output the final model:
  \[ f(x) = \text{sign} \left( \sum_{r=1}^{T} \alpha_r f_r(x) \right) \]