ENSEMBLE METHODS

Model averaging, bagging, and Random Forest
IDEA OF ENSEMBLE LEARNING

• In principle we take a bunch of simple methods and by combining them we can
  • improve their performance
  • approach more complex problems
  • retain most of their advantages
IDEA OF ENSEMBLE LEARNING II

• Traditional methods of machine learning define a class of functions $F$ and then select the optimal function $f \in F$ to be the classifier.

• Let the class $F$ be finite - for example a set of different classifiers trained on the data.

• The idea of the ensemble methods is to build a new predictor as a linear combination of the functions in $F$

\[
\hat{f}_{ensemble}(x) = \sum_{f_i \in F} w_i f_i(x)
\]
DEFINING AN ENSEMBLE METHOD

• The general form of ensemble methods

\[ f_{ensemble}(x) = \sum_{f_i \in F} w_i f_i(x) \]

• Define \( F \)
  • Ideally \( F \) is composed of independent functions that describe different aspects of the data

• Find weights \( w_i \)
  • One has to minimize

\[ \arg \max_w L(y_i, \sum_{f_i \in F} w_i f_i(x)) \]
**BAYESIAN MODEL AVERAGING**

- We can using conditional probabilities we can write for any event $\zeta$

$$\Pr(\zeta|Z) = \sum_{m=1}^{M} \Pr(\zeta|M_m, Z)Pr(M_m|Z)$$

- And

$$E(\zeta|Z) = \sum_{m=1}^{M} E(\zeta|M_m, Z)Pr(M_m|Z)$$

- This means it is reasonable to use posterior probabilities of the models as weights. Assuming uniform priors for the models, we can approximate the posteriors using model likelihoods.
FREQUENTIST APPROACH

- We can also approach this task as an optimization problem and find a solution \( w \) such that

\[
\hat{w} = \operatorname{argmin}_w E_P \left[ Y - \sum_{m=1}^{M} w_m \hat{f}_m(x) \right]^2
\]

- This is in principle a standard linear regression task and the solution for this is

\[
\hat{w} = E_P [\hat{F}(x) \hat{F}(x)^T]^{-1} E_P [\hat{F}(x) Y]
\]

- Note that this formula requires knowledge about the underlying distribution, which we usually do not know

- To apply this in practice we could use training data to approximate the expectations.
The problem with the previous method is its reliance on training errors. Therefore, it tends to add more weight to the methods that overfit the data. To overcome this, we can use cross validated predictions instead of training sample predictions. This method is called stacking. Stacked generalization, or stacking, is a technique to improve prediction performance by combining models. The final prediction is the weighted sum of predictions from different models. This method is often used in machine learning to improve the accuracy of predictions.
• Stacking was heavily used in Netflix competition
• First the teams built models that captured specific aspects of the huge dataset, such as
  • movie biases,
  • user preferences,
  • ...
• In the end the models were combined using stacking
Another ensemble method is “bagging” - bootstrap aggregating (Breiman, 1994)

First let's fix the class of models we use, like: trees, linear regression, splines, ...

Now let's generate $B$ bootstrap samples from our training dataset $Z$.

For every bootstrap sample $Z^b$ fit the model $\hat{f}^b(x)$

The bagging estimate is defined by

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$
WHEN DOES BAGGING MAKE SENSE?

• If the class of models is linear on the data then the bagged estimate $\hat{f}_{bag}(x)$ converges to the whole data estimate $f(x)$. Therefore, bagging does not make sense on
  • linear regression or classification methods

• Classification and regression trees, however, are
  • highly nonlinear
  • sensitive to input data

• Therefore, good candidate for bagging
# COMPARISON OF METHODS

**TABLE 10.1.** Some characteristics of different learning methods. Key: ▲ = good, ◆ = fair, and ▼ = poor.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Neural Nets</th>
<th>SVM</th>
<th>Trees</th>
<th>MARS</th>
<th>k-NN, Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural handling of data of “mixed” type</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
</tr>
<tr>
<td>Handling of missing values</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
</tr>
<tr>
<td>Robustness to outliers in input space</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▼</td>
<td>▲</td>
</tr>
<tr>
<td>Insensitive to monotone transformations of inputs</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Computational scalability (large $N$)</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
</tr>
<tr>
<td>Ability to deal with irrelevant inputs</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
</tr>
<tr>
<td>Ability to extract linear combinations of features</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Interpretability</td>
<td>▼</td>
<td>▼</td>
<td>◆</td>
<td>▲</td>
<td>▼</td>
</tr>
<tr>
<td>Predictive power</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
<td>◆</td>
</tr>
</tbody>
</table>

Hastie, Tibshirani, Friedman *Elements of Statistical Learning*
EXAMPLE

Data:
- 5 features (pairwise correlations 0.95)
- Sample size 30

\[ Pr(Y = 1 | x_i < 0.5) = 0.2 \]
\[ Pr(Y = 1 | x_i \geq 0.5) = 0.8 \]
EXAMPLE

FIGURE 8.10. Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.

Bagging helps under squared-error loss, in short because averaging reduces variance and leaves bias unchanged.

Assume our training observations \((x_i, y_i), i=1, \ldots, N\) are independently drawn from a distribution \(P\), and consider the idea of aggregating an estimator \(\hat{f}^*\) from \(x\). Here \(x\) is fixed and the bootstrap dataset \(Z^*\) consists of observations \(x^*_i, y^*_i, i=1, 2, \ldots, N\) sampled from \(P\). Note that \(\hat{f}^*(x)\) is a bagging estimate, drawing bootstrap samples from the actual population \(P\) rather than the data. It is not an estimate that we can use in practice, but is convenient for analysis. We can write

\[
E_P \left[ Y - \hat{f}^* (x) \right]^2 = E_P \left[ Y - \hat{f} (x) + \hat{f} (x) - \hat{f}^* (x) \right]^2 \\
\geq E_P \left[ Y - \hat{f} (x) \right]^2.
\]

(8.52)

The extra error on the right-hand side comes from the variance of \(\hat{f}^* (x)\) around its mean \(\hat{f} (x)\). Therefore true population aggregation never increases mean squared error. This suggests that bagging—drawing samples from the training data—will often decrease mean-squared error.

The above argument does not hold for classification under 0-1 loss, because of the nonadditivity of bias and variance. In that setting, bagging, a
WHY BAGGING WORKS

- Let the training sample be drawn from a distribution $P$.
- Consider the ideal aggregate estimator $f_{ag}(x) = E_P \hat{f}^*(x)$, where $\hat{f}^*(x)$ is a model based on bootstrap sample from $P$.
- Then
  \[
  E_P[Y - \hat{f}^*(x)]^2 = E_P[Y - f_{ag}(x) + f_{ag}(x) - \hat{f}^*(x)]^2 \\
  = E_P[Y - f_{ag}(x)]^2 + E_P[\hat{f}^*(x) - f_{ag}(x)]^2 \\
  \geq E_P[Y - f_{ag}(x)]^2.
  \]
- Therefore bagging works the best on situations where the models we use have high variance.
EXAMPLE WHERE BAGGING DOESN’T WORK

Bagged Decision Rule

[Diagram showing decision boundaries from bagging and boosting]
EFFECTIVENESS OF BAGGING

- The error of bagging can be divided into two components: bias and variation

\[
\begin{align*}
E_P[Y - \hat{f}^*(x)]^2 &= E_P[Y - f_{ag}(x) + f_{ag}(x) - \hat{f}^*(x)]^2 \\
&= E_P[Y - f_{ag}(x)]^2 + E_P[\hat{f}^*(x) - f_{ag}(x)]^2 \\
&\geq E_P[Y - f_{ag}(x)]^2.
\end{align*}
\]

- Bagging works by minimizing the variation component
- Bagging guarantees that the trained models come from the same distribution, but not their independence
- In case of positively correlated identically distributed random variables the variance is given by

\[
\rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2
\]
To maximize the performance of bagging one has to generate the classifiers with as little correlation as possible.

At the same time not letting the bias become too large.

Random Forest (Breiman, 2001) algorithm tries achieve this by including randomness to the tree building step.

In principle, each level of every tree is built using a random subset of variables.
Algorithm 15.1 Random Forest for Regression or Classification.

1. For $b = 1$ to $B$:
   
   (a) Draw a bootstrap sample $Z^*$ of size $N$ from the training data.
   
   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
      
      i. Select $m$ variables at random from the $p$ variables.
      
      ii. Pick the best variable/split-point among the $m$.
      
      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point $x$:

Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the $b$th random-forest tree. Then $\hat{C}_{rf}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$. 
PERFORMANCE

Typically values for \( m \) are \( \sqrt{p} \) or even as low as 1. After \( B \) such trees \( \{T(x; \Theta_b)\} \) are grown, the random forest (regression) predictor is

\[
\hat{f}_{rf}(x) = \frac{1}{B} \sum_{b=1}^{B} T(x; \Theta_b).
\]

(15.2)

As in Section 10.9 (page 356), \( \Theta_b \) characterizes the \( b \)th random forest tree in terms of split variables, cutpoints at each node, and terminal-node values. Intuitively, reducing \( m \) will reduce the correlation between any pair of trees in the ensemble, and hence by (15.1) reduce the variance of the average.

![Graph](image)

FIGURE 15.1. Bagging, random forest, and gradient boosting, applied to the spam data. For boosting, 5-node trees were used, and the number of trees were chosen by 10-fold cross-validation (2500 trees). Each "step" in the figure corresponds to a change in a single misclassification (in a test set of 1536).

Not all estimators can be improved by shaking up the data like this. It seems that highly nonlinear estimators, such as trees, benefit the most. For bootstrapped trees, \( \rho \) is typically small (0.05 or lower is typical; see Figure 15.9), while \( \sigma^2 \) is not much larger than the variance for the original tree. On the other hand, bagging does not change linear estimates, such as the sample mean (hence its variance either); the pairwise correlation between bootstrapped means is about 50% (Exercise 15.4).
PARAMETER M

- The parameter $m$ controls how big subset of variables is selected at each tree node.
- In principle one can control the bias-variance tradeoff with this parameter.
  - Small values of $m$ produce independent trees with poor performance and therefore high bias.
  - Large values of $m$ produce trees with better performance but high correlation.
- For default values for $m$ are
  - $\lceil \sqrt{p} \rceil$ for classification.
  - $\lfloor p/3 \rfloor$ for regression.
EXAMPLE OF INFLUENCE OF $M$

Random Forest Ensemble

Mean Squared Error and Squared Bias

Variance

$0.65 \quad 0.70 \quad 0.75 \quad 0.80 \quad 0.85$

$0.0 \quad 0.05 \quad 0.10 \quad 0.15 \quad 0.20$

$m$

Mean Squared Error
Squared Bias
Variance

$0 \quad 10 \quad 20 \quad 30 \quad 40 \quad 50$

$\text{Mean Squared Error}$
$\text{Squared Bias}$
$\text{Variance}$
OUT OF BAG ERRORS

• In each bootstrap sample, some of the observations get left out
• To estimate the error rate it is possible use these observation
• An out-of-bag (OOB) error rate is constructed by, creating separate random forest estimate for each observation, based on trees that did not include this variable in their training samples.
• The OOB is almost identical to N-fold cross-validation error
• The OOB error is calculated on every step of the algorithm
• Stabilization of OOB error is used as a stopping criterion for Random Forest
VARIABLE IMPORTANCE

- Variable importance are calculated in two ways
  - Using the risk improvements on tree building
    $$\mathcal{I}_\ell^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 I(v(t) = \ell)$$
    $$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T_m)$$
  - Using the risk improvement on OOB samples
    - On each node:
      - calculate the risk improvement with OOB samples
      - calculate the risk improvement on permuted OOB samples
      - these effects are summed up for a tree and averaged over trees
VARIABLE IMPORTANCES ON SPAM DATASET

FIGURE 15.5. Variable importance plots for a classification random forest grown on the spam data. The left plot bases the importance on the Gini splitting index, as in gradient boosting. The rankings compare well with the rankings produced by gradient boosting (Figure 10.6 on page 354). The right plot uses oob randomization to compute variable importances, and tends to spread the importances more uniformly.
RESISTANCE TO NOISE

- Random forests are surprisingly resistant to noise
- Say we have 6 variables with signal and 100 with pure noise, then setting \( m = \sqrt{6 + 100} \approx 10 \) gives still probability 0.46 that a relevant variable is included in any split.

![Graph showing test misclassification error and Bayes error for Random Forest and Gradient Boosting](image)

**FIGURE 15.7.** A comparison of random forests and gradient boosting on problems with increasing numbers of noise variables. In each case the true decision boundary depends on two variables, and an increasing number of noise variables are included. Random forests uses its default value \( m = \sqrt{p} \). At the top of each pair is the probability that one of the relevant variables is chosen at any split. The results are based on 50 simulations for each pair, with a training sample of 300 and a test sample of 500.
RANDOM FOREST AND OVERFITTING

• For the bagged classifiers do not overfit in a classical sense
• Adding more trees does not necessarily improve training error, but makes the function converge to some limit

\[ \hat{f}_{rf}(x) = \mathbb{E}_{\Theta} T(x; \Theta) = \lim_{B \to \infty} \hat{f}(x)^B_{rf} \]

• Overfitting can happen only if this limit overfits the data i.e the individual models are too complex
• It has been shown that pruning Random Forest trees can improve its performance somewhat. However, the gains are usually small and using this introduces another parameter to be optimized.
Random Forest is one of the most popular and useful classifiers available. With very little tuning they give rather good performance. Additionally, they:

- can handle missing values
- can handle various data types
- do not need data normalization
- are robust
- ...
INTRODUCTION TO BOOSTING

• Conceptually, bagging is a rather dull strategy: we learn the same model over and over again and finally average the results.

• We do not reward the models that work better with higher share in the final prediction.

• Finally, it really only works with trees, with most other possible classifiers we do not get any gains

• Boosting is an alternative strategy that can take almost any simple classifier - a “weak learner” - and build an ensemble of them that predicts the outcome much more accurately than the original
THE IDEA OF BOOSTING

• Boosting algorithms train the members of ensemble one by one
• At each step the training sample is re-weighted, with more weight given to observations that were misclassified by the current classifiers
• Therefore, the additional classifiers do not try to learn the same model again as in bagging, but rather concentrate on the observations that cannot be reliably classified by current ensemble
• The most well known boosting method is called “AdaBoost” (Freund and Schapire 1997)
Algorithm 10.1 AdaBoost.M1.

1. Initialize the observation weights \( w_i = 1/N, \quad i = 1, 2, \ldots, N \).

2. For \( m = 1 \) to \( M \):
   
   (a) Fit a classifier \( G_m(x) \) to the training data using weights \( w_i \).
   
   (b) Compute
   \[
   \text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.
   \]

   (c) Compute \( \alpha_m = \log((1 - \text{err}_m)/\text{err}_m) \).

   (d) Set \( w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))] \), \( i = 1, 2, \ldots, N \).

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

Figure 14.2 Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number \( m \) of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the \( m = 1 \) base learner are given greater weight when training the \( m = 2 \) base learner.

Instead of doing a global error function minimization, however, we shall suppose that the base classifiers \( y_1(x), \ldots, y_{m-1}(x) \) are fixed, as are their coefficients \( \alpha_1, \ldots, \alpha_{m-1} \), and so we are minimizing only with respect to \( \alpha_m \) and \( y_m(x) \). Separating off the contribution from base classifier \( y_m(x) \), we can then write the error function in the form

\[
E = \sum_{n=1}^{N} \exp\left\{-t_n f_{m-1}(x_n) - \alpha_m y_m(x_n)\right\}
\]

(14.22)

where the coefficients \( w_{(m)}^n = \exp\left\{-t_n f_{m-1}(x_n)\right\} \) can be viewed as constants because we are optimizing only \( \alpha_m \) and \( y_m(x) \). If we denote by \( T_m \) the set of data points that are correctly classified by \( y_m(x) \), and if we denote the remaining misclassified points by \( M_m \), then we can in turn rewrite the error function in the form

\[
E = \sum_{n \in T_m} w_{(m)}^n + \sum_{n \in M_m} w_{(m)}^n \exp\left\{-\alpha_m y_m(x_n)\right\}
\]
FORWARD STAGEWISE ADDITIVE MODELING

Algorithm 10.2 Forward Stagewise Additive Modeling.

1. Initialize $f_0(x) = 0$.

2. For $m = 1$ to $M$:
   
   (a) Compute
   
   $$(\beta_m, \gamma_m) = \arg\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$. 
FSAM INTERPRETATION IN REGRESSION

• Let us consider the the squared error loss function

\[
L(y, f(x)) = (y - f(x))^2.
\]

• For step \( m \) on can write down the loss as

\[
L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2
\]

\[
= (r_{im} - \beta b(x_i; \gamma))^2,
\]

• Therefore, for this loss function we fit the models on the residuals of previous models. So this is kind of a boosting algorithm.
EXPONENTIAL LOSS AND ADABOOST

• It turns out that AdaBoost actually follows the Forward Stagewise Additive Modeling algorithm as given before. Only instead of some ordinary loss function it follows exponential loss, that is defined as

\[ L(y, f(x)) = \exp(-y f(x)). \]

• The main attraction of exponential loss is computational, since it allows to construct simple method such as AdaBoost to optimize it.
PROPERTIES OF EXPONENTIAL LOSS

• Exponential loss is ideally minimized by

\[ f^*(x) = \arg \min_{f(x)} \mathbb{E}_y |x (e^{-y f(x)}) \]

\[ = \frac{1}{2} \log \frac{\Pr(Y = 1|x)}{\Pr(Y = -1|x)} \]

• This also justifies the sign as a classification rule
For classification (-1, 1 response) the loss functions are monotone decreasing in the margin \( yf(x) \).

Positive margins show correctly classified observations.

Therefore loss functions should penalize negative margins more than positives.

The differences between these loss functions are in degree. The penalty associated with binomial deviance increases linearly for large increasingly negative margins, whereas the exponential criterion increases the influence of such observations exponentially.
ROBUSTNESS OF ADABOOST

• For negative margins the exponential loss gets really big
• If the data contains some observations that are hard to classify, then AdaBoost concentrates all the energy on these values
• Therefore, AdaBoost is rather sensitive to noise in the data
• Robustness can be improved, if we would use a less sensitive loss function instead of exponential loss
• Unfortunately, plugging in other loss functions do not yield such simple optimization schemes
In principle one can use boosting with many classifiers.

The key is to be able to provide a training scheme for the model that can optimize the desired loss function.

Most of the boosting implementations still use trees.

Gradient boosting methodology uses ideas from numerical optimization to plug in arbitrary loss functions to the boosting algorithm with classification and regression trees.

There are several R packages that implement this approach:

- ada
- gbm
- mboost
SUMMARY

• Some of the most successful learning methods are ensemble methods
  • Random Forest is a good first algorithm to try on a new dataset
    • it can handle wide variety of data, requires little tuning and performs rather well
  • Gradient boosting can offer some performance improvements over RF (as seen in the examples from “Elements of Statistical learning”) with the expense of little bit more tuning
• The ideas behind ensemble methods like model averaging, stacking, boosting and bagging can be used for developing new methods for specific tasks