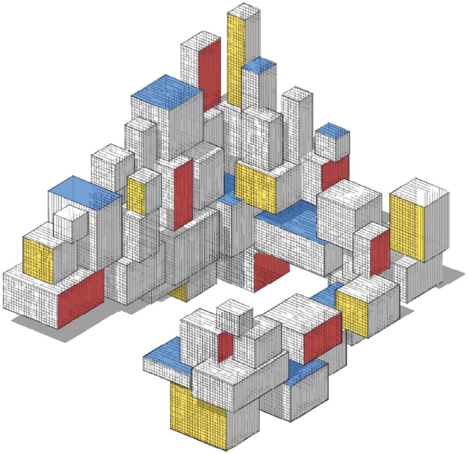
Ensemble Methods



Purpose

In this lecture we discuss two “ensemble” methods:

- Bootstrap aggregation
- Random Forests

Bootstrap Aggregation

The major idea of the *bootstrap aggregation* or bagging method is to combine prediction functions learned from multiple data sets, with a view to improving overall prediction accuracy.

Bagging is especially beneficial when dealing with predictors that tend to overfit the data, such as in decision trees, where the (unpruned) tree structure is very sensitive to small changes in the training set.

Suppose we have B iid copies $\mathcal{T}_1, \ldots, \mathcal{T}_B$ of a training set $\mathcal{T}$. Then, we can train B separate regression models (e.g., B different decision trees) using these sets, giving learners $g_{\mathcal{T}_1}, \ldots, g_{\mathcal{T}_B}$, and take their average:

$$g_{\text{avg}}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} g_{\mathcal{T}_b}(\boldsymbol{x}). \tag{1}$$

By the law of large numbers, as $B \to \infty$, the average prediction function converges to the expected prediction function $g^{\dagger} := \mathbb{E} g_{\mathcal{T}}$.

Expected Trainer

The following result shows that using the expected trainer $g^{\dagger}$ as a prediction function (if it were known) would result in an expected squared-error generalization risk that is less than or equal to the expected generalization risk for a general prediction function $g_{\mathcal{T}}$.

Theorem: Expected Squared-Error Generalization Risk

Let $\mathcal{T}$ be a random training set and let $\boldsymbol{X}, Y$ be a random feature vector and response that are independent of $\mathcal{T}$. Then,

$$\mathbb{E}\left(Y - g_{\mathcal{T}}(\boldsymbol{X})\right)^2 \geqslant \mathbb{E}\left(Y - g^{\dagger}(\boldsymbol{X})\right)^2.$$

Proof

We have

$$\mathbb{E}\left[\left(Y - g_{\mathcal{T}}(\boldsymbol{X})\right)^2 \,\middle|\, \boldsymbol{X}, Y\right] \geqslant \left(\mathbb{E}[Y \mid \boldsymbol{X}, Y] - \mathbb{E}[g_{\mathcal{T}}(\boldsymbol{X}) \mid \boldsymbol{X}, Y]\right)^2 = \left(Y - g^{\dagger}(\boldsymbol{X})\right)^2,$$

where the inequality follows from $\mathbb{E}U^2 \geqslant (\mathbb{E}U)^2$ for any (conditional) expectation.

Consequently, by the tower property,

$$\mathbb{E}\left(Y - g_{\mathcal{T}}(\boldsymbol{X})\right)^2 = \mathbb{E}\left[\mathbb{E}\left[\left(Y - g_{\mathcal{T}}(\boldsymbol{X})\right)^2 \mid \boldsymbol{X}, Y\right]\right] \geqslant \mathbb{E}\left(Y - g^{\dagger}(\boldsymbol{X})\right)^2.$$

Bootstrapped Aggregated Estimator

Unfortunately, multiple independent data sets are rarely available.

But we can substitute them by bootstrapped ones.

Specifically, instead of the $\mathcal{T}_1, \ldots, \mathcal{T}_B$ sets, we can obtain random training sets $\mathcal{T}_1^*, \ldots, \mathcal{T}_B^*$ by resampling them from a single (fixed) training set τ and use them to train B separate models.

By model averaging as in (1) we obtain the bootstrapped aggregated estimator or bagged estimator of the form:

$$g_{\text{bag}}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} g_{\mathcal{T}_b^*}(\boldsymbol{x}). \tag{2}$$

Algorithm 1: Bootstrap Aggregation Sampling

Input: Training set $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ and resample size B.

Output: Bootstrapped data sets.

```
1 for $b = 1$ to $B$ do
2     $\mathcal{T}_b^* \leftarrow \emptyset$
3     for $i = 1$ to $n$ do
4         Draw $U \sim \mathcal{U}(0, 1)$;
5         $I \leftarrow \lceil nU \rceil$          // select random index;
6         $\mathcal{T}_b^* \leftarrow \mathcal{T}_b^* \cup \{(\boldsymbol{x}_I, y_I)\}$.
7 return $\mathcal{T}_b^*, b = 1, \ldots, B$.
```

Bootstrap Aggregation for Classification Problems

Note that (2) is suitable for handling regression problems.

However, the bagging idea can be readily extended to handle classification settings as well. For example, g_{bag} can take the majority vote among $\{g_{\mathcal{T}_b^*}\}, b = 1, \ldots, B$; that is, to accept the most frequent class among B predictors.

While bagging can be applied for any statistical model, it is most effective for predictors that are sensitive to small changes in the training set.

The reason becomes clear when we decompose the expected generalization risk as

$$\mathbb{E}\,\ell(g_{\mathcal{T}}) = \ell^* + \underbrace{\mathbb{E}\left(\mathbb{E}[g_{\mathcal{T}}(\boldsymbol{X}) \mid \boldsymbol{X}] - g^*(\boldsymbol{X})\right)^2}_{\text{expected squared bias}} + \underbrace{\mathbb{E}[\mathbb{V}\mathrm{ar}[g_{\mathcal{T}}(\boldsymbol{X}) \mid \boldsymbol{X}]]}_{\text{expected variance}}. \quad (3)$$

(Un)stable Predictors

Compare this with the same decomposition for the average prediction function g_{bag} in (1).

As $\mathbb{E} g_{\text{bag}}(\boldsymbol{x}) = \mathbb{E} g_{\mathcal{T}}(\boldsymbol{x})$, we see that any possible improvement in the expected generalization risk must be due to the expected variance term.

Averaging and bagging are thus only useful for predictors with a large expected variance, relative to the other two terms.

Examples of such "unstable" predictors include decision trees, neural networks, and subset selection in linear regression.

On the other hand, "stable" predictors are insensitive to small data changes, an example being the K-nearest neighbors method.

Note that for independent training sets $\mathcal{T}_1, \ldots, \mathcal{T}_B$ a reduction of the variance by a factor B is achieved: $\mathbb{V}\text{ar}\, g_{\text{bag}}(\boldsymbol{x}) = B^{-1} \mathbb{V}\text{ar}\, g_{\mathcal{T}}(\boldsymbol{x})$.

Out-of-Bag Observations

The bagging process provides an opportunity to estimate the generalization risk of the bagged model without an additional test set.

Specifically, recall that we obtain the $\mathcal{T}_1^*, \ldots, \mathcal{T}_B^*$ sets from a single training set τ by sampling via Algorithm 1, and use them to train B separate models.

It can be shown (exercise!) that, for large sample sizes, on average about a third (more precisely, a fraction $\mathrm{e}^{-1} \approx 0.37$) of the original sample points are not included in bootstrapped set $\mathcal{T}_b^*$ for $1 \leqslant b \leqslant B$.

Therefore, these samples can be used for the loss estimation. These samples are called out-of-bag (OOB) observations.

Algorithm 2: Out-of-Bag Loss Estimation

Input: The original data set $\tau = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, the bootstrapped data sets $\{\mathcal{T}_1^*, \ldots, \mathcal{T}_B^*\}$, and the trained predictors $\{g_{\mathcal{T}_1^*}, \ldots, g_{\mathcal{T}_B^*}\}$.

Output: Out-of-bag loss for the averaged model.

```
1 for $i = 1$ to $n$ do
2     $C_i \leftarrow \emptyset$ ;   // Indices of predictors not depending on $(\boldsymbol{x}_i, y_i)$
3     for $b = 1$ to $B$ do
4         if $(\boldsymbol{x}_i, y_i) \notin \mathcal{T}_b^*$ then $C_i \leftarrow C_i \cup \{b\}$ ;
5     $Y_i' \leftarrow |C_i|^{-1} \sum_{b \in C_i} g_{\mathcal{T}_b^*}(\boldsymbol{x}_i)$
6     $L_i \leftarrow \text{Loss}\left(y_i, Y_i'\right)$
7 $L_{\text{OOB}} \leftarrow \frac{1}{n} \sum_{i=1}^{n} L_i$
8 return $L_{\text{OOB}}$.
```

Example: Bagging for a Regression Tree

Below is a basic bagging example for a regression tree, comparing the decision tree estimator with the corresponding bagged estimator. We use the R^2 metric (coefficient of determination) for comparison.

BaggingExample.py

```
import numpy as np
from sklearn.datasets import make_friedman1
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

np.random.seed(100)

# create regression problem
n_points = 1000 # points
x, y =  make_friedman1(n_samples=n_points, n_features=15,
                       noise=1.0, random_state=100)

# split to train/test set
x_train, x_test, y_train, y_test = \
        train_test_split(x, y, test_size=0.33, random_state=100)
```

```
# training
regTree = DecisionTreeRegressor(random_state=100)
regTree.fit(x_train,y_train)

# test
yhat = regTree.predict(x_test)

# Bagging construction
n_estimators=500
bag = np.empty((n_estimators), dtype=object)
bootstrap_ds_arr = np.empty((n_estimators), dtype=object)
for i in range(n_estimators):
    # sample bootstrapped data set
    ids = np.random.choice(range(0,len(x_test)),size=len(x_test),
                     replace=True)
    x_boot = x_train[ids]
    y_boot = y_train[ids]
    bootstrap_ds_arr[i] = np.unique(ids)

    bag[i] = DecisionTreeRegressor()
    bag[i].fit(x_boot,y_boot)
```

```python
# bagging prediction
yhatbag = np.zeros(len(y_test))
for i in range(n_estimators):
    yhatbag = yhatbag + bag[i].predict(x_test)

yhatbag = yhatbag/n_estimators

# out of bag loss estimation
oob_pred_arr = np.zeros(len(x_train))
for i in range(len(x_train)):
    x = x_train[i].reshape(1, -1)
    C = []
    for b in range(n_estimators):
        if(np.isin(i, bootstrap_ds_arr[b])==False):
            C.append(b)
    for pred in  bag[C]:
        oob_pred_arr[i] = oob_pred_arr[i] + (pred.predict(x)/len(C))

L_oob = r2_score(y_train, oob_pred_arr)
print("DecisionTreeRegressor R^2 score = ",r2_score(y_test, yhat),
      "\nBagging R^2 score = ", r2_score(y_test, yhatbag),
      "\nBagging OOB R^2 score = ",L_oob)
```

```
DecisionTreeRegressor R^2 score =  0.575438224929718
Bagging R^2 score =  0.7612121189201985
Bagging OOB R^2 score =  0.7758253149069059
```

Correlated Predictions

For a feature vector $\boldsymbol{x}$ let $Z_b = g_{\mathcal{T}_b}(\boldsymbol{x}), b = 1, 2, \ldots, B$ be iid prediction values, obtained from independent training sets $\mathcal{T}_1, \ldots, \mathcal{T}_B$.

Suppose that $\mathbb{V}\mathrm{ar}\, Z_b = \sigma^2$ for all $b = 1, \ldots, B$. Then the variance of the average prediction value $\overline{Z}_B$ is equal to σ^2/B.

However, if bootstrapped data sets $\{\mathcal{T}_b^*\}$ are used instead, the random variables $\{Z_b\}$ will be correlated. In particular, $Z_b = g_{\mathcal{T}_b^*}(\boldsymbol{x})$ for $b = 1, \ldots, B$ are identically distributed (but not independent) with some positive pairwise correlation ϱ. It then holds (exercise!) that

$$\mathbb{V}\mathrm{ar}\, \overline{Z}_B = \varrho\, \sigma^2 + \sigma^2 \frac{(1 - \varrho)}{B}. \tag{4}$$

While the second term of (4) goes to zero as the number of observation B increases, the first term remains constant.

Random Forests

This correlated prediction issue is particularly relevant for bagging with decision trees.

For example, consider a situation in which there exists a feature that provides a very good split of the data.

Such a feature will be selected for every $\{g_{\mathcal{T}_b^*}\}_{b=1}^{B}$ at the root level and we will consequently end up with highly correlated predictions.

In such a situation, prediction averaging will not introduce the desired improvement in the performance of the bagged predictor.

The major idea of random forests is to perform bagging in combination with a “decorrelation” of the trees by including only a subset of features during the tree construction.

For each bootstrapped training set $\mathcal{T}_b^*$ we build a decision tree using a randomly selected subset of $m \leqslant p$ features for the splitting rules.

```
Algorithm 3: Random Forest Construction
Input: Training set $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, the number of trees in the
       forest $B$, and the number $m \leqslant p$ of features to be included,
       where $p$ is the total number of features in $\boldsymbol{x}$.
Output: Ensemble of trees.
1 Generate bootstrapped training sets $\{\mathcal{T}_1^*, \ldots, \mathcal{T}_B^*\}$ via
   Algorithm 1.
2 for $b = 1$ to $B$ do
3     Randomly select $m$ out of $p$ features, without replacement.
4     Using only these features, train a decision tree $g_{\mathcal{T}_b^*}$.
5 return $\{g_{\mathcal{T}_b^*}\}_{b=1}^B$.
```

For regression problems, the output of Algorithm 3 is combined to yield the random forest prediction function: $g_{\mathrm{RF}}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^B g_{\mathcal{T}_b^*}(\boldsymbol{x})$.

In the classification setting, we take instead the majority vote from the $\{g_{\mathcal{T}_b^*}\}$.

BaggingExampleRF.py

```python
from sklearn.datasets import make_friedman1
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor

# create regression problem
n_points = 1000 # points
x, y =  make_friedman1(n_samples=n_points, n_features=15,
                       noise=1.0, random_state=100)
# split to train/test set
x_train, x_test, y_train, y_test = \
        train_test_split(x, y, test_size=0.33, random_state=100)
rf = RandomForestRegressor(n_estimators=500, oob_score = True,
     max_features=8,random_state=100)
rf.fit(x_train,y_train)
yhatrf = rf.predict(x_test)
print("RF R^2 score = ", r2_score(y_test, yhatrf),
      "\nRF OOB R^2 score = ", rf.oob_score_)
```

```
RF R^2 score =  0.8106589580845707
RF OOB R^2 score =  0.8260541058404149
```

The Optimal Number of Subset Features m

The default values for m are $\lfloor p/3 \rfloor$ and $\left\lfloor \sqrt{p} \right\rfloor$ for regression and classification setting, respectively. However, the standard practice is to treat m as a hyperparameter that requires tuning, depending on the specific problem at hand.

Note that the procedure of bagging decision trees is a special case of a random forest construction. Consequently, the OOB loss is readily available for random forests.

While the advantage of bagging in the sense of enhanced accuracy is clear, we should also consider its negative aspects and, in particular, the loss of interpretability.

Feature Importance

The feature importance intends to address the interpretability issue.

The idea is as follows. Each *internal* node v of a decision tree induces a certain decrease $\Delta_{\text{Loss}}(v)$ in the training loss.

In addition, recall that for splitting rules of the type $\mathbb{I}\{x_j \leqslant \xi\}$ $(1 \leqslant j \leqslant p)$, each node v is associated with a feature x_j that determines the split. Using the above definitions, we can define the feature importance of x_j as

$$\mathcal{I}_{\mathbb{T}}(x_j) = \sum_{v \text{ internal } \in \mathbb{T}} \Delta_{\text{Loss}}(v)\, \mathbb{I}\{x_j \text{ is associated with } v\}, \quad 1 \leqslant j \leqslant p. \tag{5}$$

The feature importance of the random forest $\{\mathbb{T}_1, \ldots, \mathbb{T}_B\}$ is the average:

$$\mathcal{I}_{\text{RF}}(x_j) = \frac{1}{B} \sum_{b=1}^{B} \mathcal{I}_{\mathbb{T}_b}(x_j), \quad 1 \leqslant j \leqslant p. \tag{6}$$

Example: Feature Importance

We consider a classification problem with 15 features. The data is specifically designed to contain only 5 informative features out of 15. In the code below, we apply the random forest procedure and calculate the corresponding feature importance measures.

VarImportance.py

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt, pylab

n_points = 1000 # create regression data with 1000 data points
x, y =  make_classification(n_samples=n_points, n_features=15,
  n_informative=5, n_redundant=0, n_repeated=0, random_state=100,
       shuffle=False)

rf = RandomForestClassifier(n_estimators=200, max_features="log2")
rf.fit(x,y)
```

```python
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]

for f in range(15):
    print("Feature %d (%f)" % (indices[f]+1, importances[indices[f]]))

std = np.std([rf.feature_importances_ for tree in rf.estimators_],
             axis=0)
f = plt.figure()
plt.bar(range(x.shape[1]), importances[indices],
       color="b", yerr=std[indices], align="center")
plt.xticks(range(x.shape[1]), indices+1)
plt.xlim([-1, x.shape[1]])
pylab.xlabel("feature index")
pylab.ylabel("importance")
plt.show()
```

Clearly, it is hard to visualize and understand the prediction process based on 200 trees. However, the figure shows that the features x_1, x_2, x_3, x_4, and x_5 were correctly identified as being important.

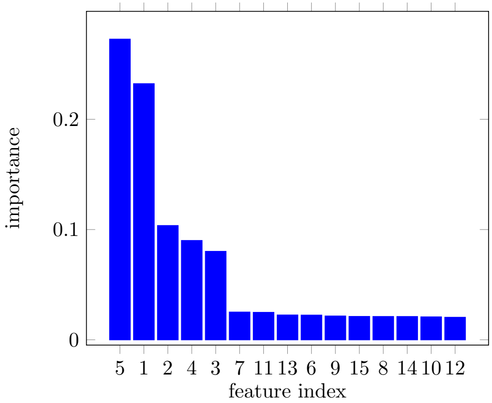


Figure: Importance measure for the 15-feature data set with only 5 informative features x_1, x_2, x_3, x_4, and x_5.