Machine learning overview

Erik Pärlstrand

November 2018

1 Machine learning concepts

1.1 Bias variance trade-off

The bias variance trade-off is characterized by trying to minimize two sources of errors, bias and variance, that prevent a supervised learner to generalize beyond its training set.

The bias in the model comes from erroneous assumptions in the learner. Large bias can arise when relevant relationships between the features and the response are missed i.e. underfitting; the model is not complex enough to fit the data. Underfitted models do not perform well on the training, nor the test set.

The variance in the model arises from sensitivity to small fluctuations in the training set. High variance can be caused by overfitting; modeling the random noise in the training set rather than the intended response and therefore usually generalize poorly outside of the training set.

Figure 1 is an illustration of the training and test set errors as the model complexity increases:

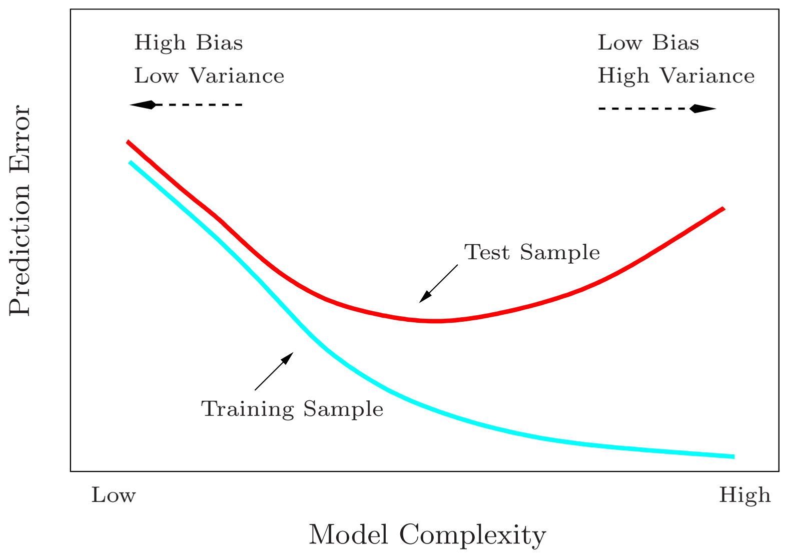


Figure 1: A visualization of the bias variance trade-off.

From figure 1 one can see that high bias, low variance and low bias, high variance classifiers has large out of sample error. However, by balancing the variance and bias, out of sample performance can be optimized.

1.2 Cross-validation

The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting and to give an insight on how the model will generalize to an independent dataset (e.g. the validation or test set).

For time-series, the order of the data is important and therefore cross-validation might be problematic for. A more appropriate approach might be split the data in "time order".

K-fold cross-validation

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. A common value for k is 10.

Holdout method

In the holdout method, we randomly assign data points to two sets $\mathcal{D}_0$ and $\mathcal{D}_1$, usually called the training set and the test set, respectively. We then train (build a model) on $\mathcal{D}_0$ and test (evaluate its performance) on $\mathcal{D}_0$. In typical cross-validation, results of multiple runs of model-testing are averaged together; in contrast, the holdout method, involves a single run.

K-fold cross-validation is to prefer if the time to train the model is low. Otherwise, the holdout method is to prefer (e.g. for large deep learning models).

2 Tree based models

2.1 Classification trees

Let us define what a tree is:

Definition 1. A tree is a graph $\mathcal{G}$ in which any two nodes are connected by exactly one path.

Definition 2. A rooted tree is a tree in which one of the nodes has been assigned as the root. Furthermore, assume that the rooted tree is a directed graph, i.e. all edges are directed away from the root.

Definition 3. If there is an edge from η_1 to η_2 then node η_1 is said to be the parent of node η_2 while node η_2 is said to be a child of node η_1.

Definition 4. In a rooted tree, a node is said to be internal if it has one or more children and terminal if it has no children.

Definition 5. A binary tree is a rooted tree where all internal nodes have exactly two children.

Classification trees are rooted binary trees. Figure 2 is an example of a rooted binary tree:

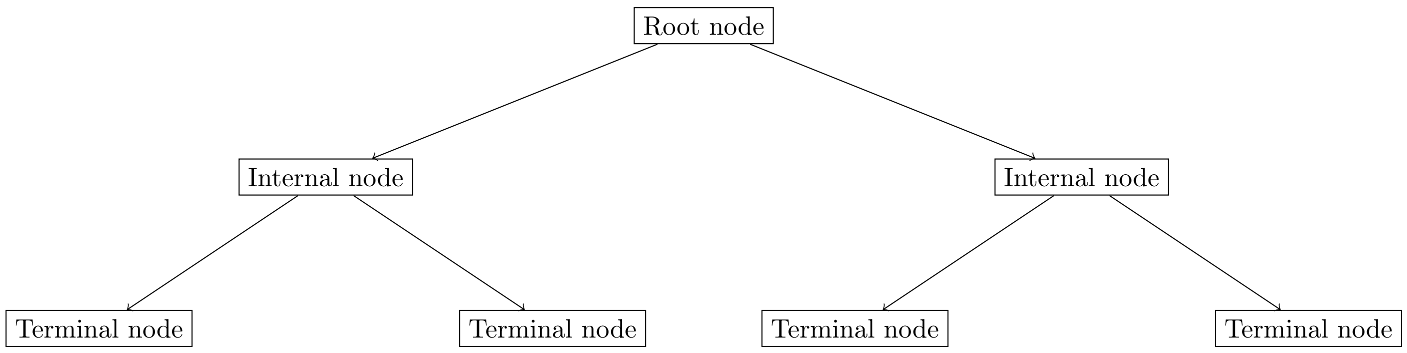


Figure 2: Example of a rooted binary tree structure.

A classification tree can be defined as a model $f : \mathcal{X} \to \mathcal{Y}$ represented by a rooted tree, where any node η represents a subspace $\mathcal{X}_\eta \subseteq \mathcal{X}$ of the feature space, with the root node η_0 corresponding to $\mathcal{X}$ itself. Internal nodes η are labeled with a split s_η, which divides the space $\mathcal{X}_\eta$ that node η represents into disjoint subspaces.

Let us define an impurity measure $i(\eta)$ which evaluates the goodness of any node η. Let us assume that the smaller $i(\eta)$, the purer the node and the better the predictions $\hat{y}_i$ is, for all $\boldsymbol{x}_i \in \mathcal{D}_\eta$, where $\mathcal{D}_\eta$ is the subset of training samples such that $\boldsymbol{x}_i \in \mathcal{X}_\eta$, where $\{\boldsymbol{x}_i, y_i\} \in \mathcal{D}$.

Starting from a single node representing the whole training set $\mathcal{D}$, near-optimal decision trees can then be grown greedily by iteratively dividing nodes into purer nodes. That is, by iteratively dividing $\mathcal{D}$ into smaller subsets, until a stopping criteria is meet, for example the impurity decrease is not large enough in a given node. The greedy assumption is to divide each node η using the split s that locally maximizes the decrease of impurity of the resulting child nodes. Formally, the decrease of impurity of a binary split s is defined as follows:

Definition 6. The impurity decrease of a binary split s dividing node η in a left node η_L and a right node η_R is:

$$\Delta i(s, \eta) = i(\eta) - \frac{\bar{\eta}_L}{\bar{\eta}} i(\eta_L) - \frac{\bar{\eta}_R}{\bar{\eta}} i(\eta_R)$$

where $\bar{\eta}_* = |\mathcal{D}_{\eta_*}|$, where $*$ indicates the node index.

The Gini impurity can be used as the impurity measure. The Gini impurity measures the inequality among classes of a frequency distribution. If the Gini impurity is close to one it indicates almost perfect equality (i.e. the class frequency is roughly the same). If the Gini impurity is close to zero it has maximal inequality among classes (i.e. one class has a frequency of 100% while the others have 0%). The Gini impurity is defined as:

$$i_G(\eta) = 1 - \sum_{k=1}^{K} \hat{\nu}_k^2 \tag{1}$$

where $\hat{\nu}_k^2$ is the frequency of class k amongst the population in node η and K is the total number of classes. $\hat{\nu}_k$ can be calculated in node η as:

$$\hat{\nu}_k = \frac{1}{|\mathcal{D}_\eta|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_\eta} \mathbb{1}\big(y_i = k\big) \tag{2}$$

In order to build the tree one has to be able to determine where the splits should be made. The splits are binary splits defined on a single feature X_j and results in non-empty subsets, i.e. $\mathcal{D}_{\eta_L} \neq \emptyset \wedge \mathcal{D}_{\eta_R} \neq \emptyset$.

Now, if X_j is a categorical feature then one can simply compare the different possible splits (assuming p is of reasonable size). For example, assume that the feature X_j is a discrete feature with three outcomes, $\{-1, 0, 1\}$. Then the splits to consider would be $X_j \geq 0$ and $X_j \geq 1$. Now, one would choose the split that yields the highest Gini gain according to equation (1).

However, determining the split for a continuous feature X_j is a bit harder. The most common approach for determining the split for a continuous feature is to group the data set into bins. The following algorithm can be used to find the best splitting point on a continuous feature X_j:

Algorithm 1: Finding the best split s_j on feature X_j in node η.

```
1 Assume that $\{x_i^j\} \in \mathcal{D}_\eta$
2 Let $Q$ be the number of bins.
3 Create $Q$ bins from $\{x_i^j\}$, $\mathcal{B}_1, ..., \mathcal{B}_Q$
4 for $q = 1, ..., Q-1$ do
5     Create the split $s_j^q$ according to the largest value in bin $q$, i.e. $s_j^q = \max\left(x_i^j | x_i^j \in \mathcal{B}_q\right)$
6     Compute the Gini gain for the split $s_j^q$, i.e. $\Delta i_G(s_j^q, \eta)$
7 end
8 Choose the split $s_j$ that maximizes the Gini gain, i.e. $s_j = \arg\max_q \Delta i_G(s_j^q, \eta)$
```

Algorithm 1 can also be used for categorical features when p is large, with some modifications.

Now, the following algorithm can be employed to determine the best split amongst all features:

Algorithm 2: Finding the best split s_η in node η.

```
1 $\Delta = -\infty$
2 for $j = 1, ..., p$ do
3     Find the best binary split $s_j$ defined on $X_j$ according to algorithm 1
4     if $\Delta i_G(s_j, \eta) > \Delta$ then
5         $\Delta = \Delta i_G(s_j, \eta)$
6         $s_\eta = s_j$
7     end
8 end
```

Now, equipped with the methods described earlier, one can finally define the process of building a classification tree:

Algorithm 3: The algorithm for building a classification tree T_b.

```
1 for each internal node $\eta$, recursively do
2     Find the best split $s_\eta$ according to algorithm 2
3     Split the node $\eta$ into two child nodes $(\eta_L, \eta_R)$ according to $s_\eta$
4     Partition the data $\mathcal{D}_\eta$ according to $s_\eta$, i.e.
5     $\mathcal{D}_{\eta_L} = \{\boldsymbol{x_i}, y_i\} \big| \{\boldsymbol{x}_i, y_i\} \in \mathcal{D}_\eta \wedge x_i^j \leq s_\eta$
6     $\mathcal{D}_{\eta_R} = \mathcal{D}_\eta \setminus \mathcal{D}_{\eta_L}$
7 end
```

Figure 3 is an example of a classification tree which partition the feature space according to a tree model into four disjoint regions:

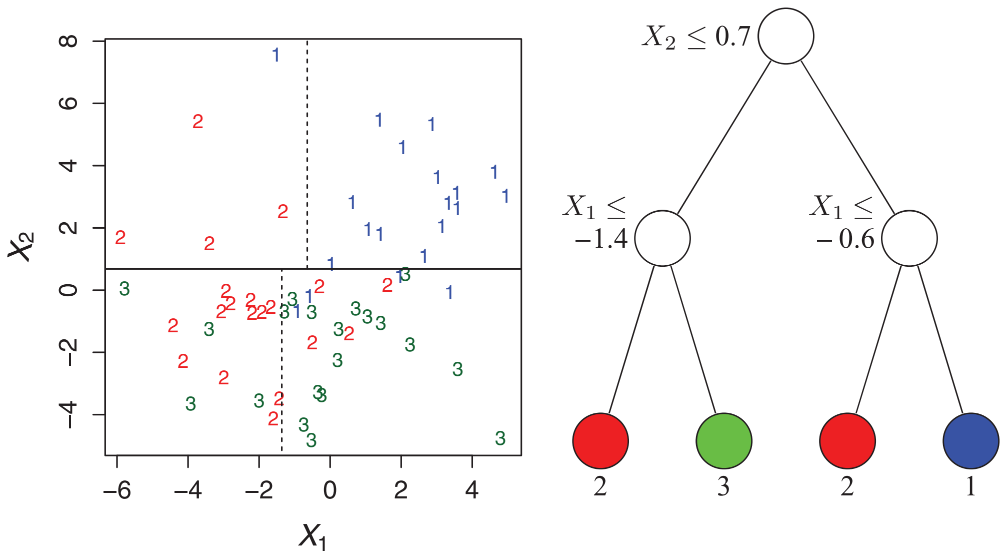


Figure 3: An example of a classification tree and the corresponding feature space partitions.

Assuming that the classification tree is built, one can predict a new point $\boldsymbol{x}_*$ by:

Algorithm 4: Predicting $\hat{y}_*$ from a classification tree.

```
1 $\eta = \eta_0$
2 while $\eta$ is not a terminal node do
3     $\eta$ = the child node $\eta'$ of $\eta$ such that $\boldsymbol{x}_* \in \mathcal{X}_{\eta'}$
4 end
5 $\hat{y}_* = \arg\max_k \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_\eta} \mathbb{1}(y_i = k)$
```

Unfortunately, classification trees tend to create over-complex trees that do not generalize well out of sample (i.e. overfitting). In terms of the variance-bias trade off, they usually produce low bias, high variance classifiers. This can be solved with an ensemble learner.

3 Ensemble learning

An ensemble learner contains a set of learners which are usually called base learners. The generalization ability of an ensemble is usually much larger than the base learners. Actually, ensemble learning is appealing because they are able to boost weak learners which are just slightly better than random guesses to strong learners which can make very accurate predictions.

The base learners f_i are generated from a training set by a base learning algorithm which can be decision trees. Bagging of trees and random forest are two examples of ensemble methods based on decision trees.

Typically, an ensemble is constructed in two steps. First, a number of base learners are produced, which can be generated in a parallel style or in a sequential style where the generation of base learners has influence on the generation of subsequent learners. Then, the base learners are combined, usually by majority voting for classification:

$$\hat{F}(\boldsymbol{x}) = \arg\max_k \sum_i \mathbb{1}\big(\hat{f}_i(\boldsymbol{x}) = k\big) \tag{3}$$

3.1 Bagging

Bagging (also known as bootstrap aggregating) is an ensemble learner designed to lower the variance and thereby increase prediction accuracy by combining multiple classification trees.

Given a training set $\mathcal{D}$ of size n, the bagging method generates B new training sets $\{\mathcal{D}'_b\}_{b=1}^{B}$ with size n' by sampling from $\mathcal{D}$ through bootstrap cases. These new training sets $\mathcal{D}'_b$ are being used to train the new models $\hat{f}_b$ and combined by majority vote to produce a prediction:

$$\hat{F}_{bag}(\boldsymbol{x}) = \arg\max_k \sum_{b=1}^{B} \mathbb{1}\big(\hat{f}_b(\boldsymbol{x}) = k\big) \tag{4}$$

A critical factor to whether bagging will improve accuracy is the stability of the procedure for constructing $\hat{f}$, i.e. the base learner. If changes in $\mathcal{D}$, i.e. a replicate $\mathcal{D}$, produces small changes in $\hat{f}$ then $\hat{F}$ will be close to $\hat{f}$. Improvement will occur for unstable procedures where a small change in $\mathcal{D}$ can result in large changes in $\hat{f}$. Therefore, bagging unstable classifiers usually improves them. Bagging stable classifiers is not a good idea.

In terms of bias variance trade-off, the bagging procedure leads to a decrease in variance and a small increase in bias. The prediction of a single tree is highly sensitive to noise in the training set. The average of many trees are not sensitive to noise, as long as the tress are not correlated. Training multiple trees on the same data set would lead to highly correlated trees, and choosing bootstrap samples is a way to de-correlate the tress by showing them different training sets.

The number of trees B is a free parameter that is chosen by the user. The algorithm for bagging follows:

Algorithm 5: The bagging algorithm.

```
1  for $b = 1$ to $B$ do
2      Select $n'$ observations from $\mathcal{D}$ with replacement to form the bootstrap sample $\mathcal{D}'_b$
3      Grow a classification tree $T_b$ on the bootstrap sample $\mathcal{D}'_b$ according to algorithm 3,
         which represent the function $\hat{f}_b$
4  end
5  Majority vote the output from all $B$ trees, i.e. $\hat{F}_{\text{bag}}(\boldsymbol{x}) = \arg\max_k \sum_{b=1}^{B} \mathbb{1}\big(\hat{f}_b(\boldsymbol{x}) = k\big)$
```

3.2 Random forest

Random forest is an extension of the bagging algorithm. It uses the same bootstrap procedure as bagging but, at each split, a random subset of features with size $p_{try} \leq p$ is selected to build the trees. The reason for doing this is to de-correlate the trees even more. When one uses ordinary bootstrap samples, some of the features can be strong predictors for the response and they will be selected in many of the B trees which will cause them to become correlated. By choosing a random subset one might break up this dependency and make them more de-correlated.

Usually, in a classification setting with p features, one chooses $p_{try} = \sqrt{p}$. However, this parameter can be determined by the user according to the problem specification. Furthermore, random forest have more hyper-parameters that can be tuned, for example, number of trees that are grown B, depth of the tree and more.

The algorithm for random forest follows:

Algorithm 6: The random forest algorithm.

```
1  for $b = 1$ to $B$ do
2      Select $n'$ observations from $\mathcal{D}$ with replacement to form the bootstrap sample $\mathcal{D}'_b$
3      Grow a classification tree $T_b$ on the bootstrap sample by recursively
4      for each internal node $\eta$ do
5          Randomly select $p_{try}$ features
6          Select the best split $s_\eta$ among the $p_{try}$ features according to algorithm 2 in node $\eta$
7          Split the node $\eta$ into two child nodes according to the best split $s_\eta$
8      end
9      This tree $T_b$ represent the function $\hat{f}_b$
10 end
11 Majority vote the output from all $B$ trees, i.e. $\hat{F}_{\text{rf}}(\boldsymbol{x}) = \arg\max_k \sum_{b=1}^{B} \mathbb{1}\big(\hat{f}_b(\boldsymbol{x}) = k\big)$
```

4 Performance metrics

4.1 Classification metrics

4.1.1 Recall

$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

Where tp is true positive, fn is false negative and fp is false positive.

4.1.2 Precision

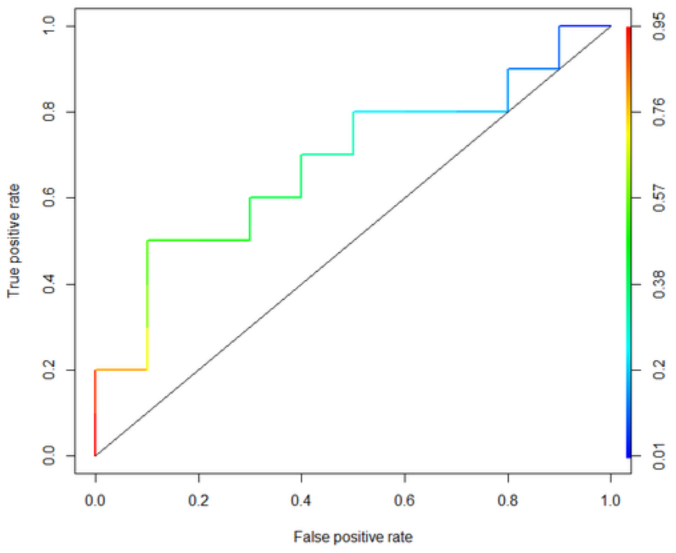
$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

4.1.3 F-1

$$f_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

ROC curve

Receiver operating characteristic (ROC) curve summaries the trade-off between false positive rate and true positive rate, for different predictions thresholds. The AUC value is the area under ROC curve. Below is an visualization of an ROC curve:



And the AUC value would be, for example, 0.7.

4.2 Regression metrics

4.2.1 MSE

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\text{RMSE} = \sqrt{\text{MSE}}$$

4.2.2 MAE

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

4.2.3 MAPE

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

4.2.4 R^2

R-squared represents the proportion of the variance of the target that is explained by the model. It is defined by:

$$R^2 = 1 - \frac{\mathrm{SS}_{res}}{\mathrm{SS}_{tot}}$$

$$\mathrm{SS}_{res} = \sum_i (y_i - \hat{y}_i)^2$$

$$\mathrm{SS}_{tot} = \sum_i (y_i - \bar{y})^2$$

where $\hat{y}_i$ is the prediction and $\bar{y}$ is the mean value.