# Fundamentals of Machine Learning:

## Ensemble Models

Prof. Andrew D. Bagdanov (andrew.bagdanov AT unifi.it)


UNIVERSITÀ
DEGLI STUDI
FIRENZE
DINFO
DIPARTIMENTO DI
INGEGNERIA
DELL'INFORMAZIONE

# Outline

# Introduction

# The power of consensus

- All of the regression and classification models we have considered thus far are singular – we fit one model, and predict using one model.
- Often we can improve over singular models by combining multiple ones.
- We might train $M$ models, and then take the average of their predictions.
- Or, we could train models sequentially and encourage subsequent models to compensate for errors made by previous ones.
- Or, we might train multiple models and then select a best one to make predictions for a given $\mathbf{x}$.
- Such models are usually called ensemble models since they make predictions based on a set of trained models instead of just one.

# Lecture objectives

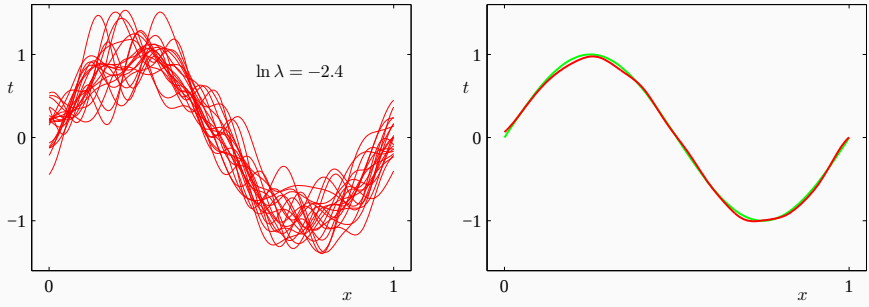
After this lecture you will:

- Understand how bagging (or bootstrap aggregation) can be used to generate multiple datasets from a single one.
- Understand how committees are formed from fitting multiple models to bootstrap datasets.
- Understand how and why committees reduce model error through averaging.
- Understand how boosting works to train a sequence of weak learners and how these learners are combined into a stronger committee via weighting.
- Understand how tree models partition the space into cuboid regions and how to learn their structure and parameters.

# Committees and Bagging

# The power of consensus

- We saw way back in one of our first lectures how sampling multiple models and averaging helps reduce variance:
- Can we take advantage of this without a fully-Bayesian model?



# Bootstrap sampling

- Consider a regression problem where we are predicting a single continuous target.
- The variance reduction approach worked by fitting a low bias model to multiple datasets and averaging the outputs.
- In practice we have only one dataset, however.
- The bootstrap method works by sampling $M$ datasets of size $N' < N$ from $\mathcal{D}$ *with replacement*.
- Each of these bootstrap datasets reflects the underlying distribution of $\mathcal{D}$ but is incomplete.

# Bootstrap aggregation (or *bagging*)

- We can then fit a model $y_m(\mathbf{x})$ to each bootstrap dataset (using any method).
- And then form the committee model by averaging the $M$ base models:

$$y_{\text{com}}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} y_m(\mathbf{x})$$

- Denote by $h(\mathbf{x})$ the true regression function generating $\mathcal{D}$.
- The output of each model can be written as this true function plus an error:

$$y_m(\mathbf{x}) = h(\mathbf{x}) + \varepsilon_m(\mathbf{x})$$

- Note that we haven't done anything yet – except identify the error in each of our models.

# Quantifying error

- The expected squared error of each model is then:

$$\mathbb{E}_{\mathbf{x}}\left[\{h(\mathbf{x}) - y_m(\mathbf{x})\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\varepsilon_m(\mathbf{x})^2\right]$$

- And the average error of the individual models is:

$$E_{\text{av}} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathbf{x}}\left[\varepsilon_m(\mathbf{x})^2\right]$$

- Again, we haven't really said anything earth shattering: the average squared error made by the models is the average of all of the squared errors made by the models.

# The committee error

- We can compute the expected error of the committee similarly:

$$
\begin{aligned}
E_{\text{com}} &= \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} h(\mathbf{x}) - y_m(\mathbf{x})\right\}^2\right] \\
&= \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \varepsilon_m(\mathbf{x})\right\}^2\right]
\end{aligned}
$$

- OK, but squaring that $\sum$ is going to be a huge mess...

# The committee error

- How can we overcome this? Answer: assumptions!

$$E_{\text{com}} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M}\varepsilon_m(\mathbf{x})\right\}^2\right]$$

- If we assume the errors are zero-mean and uncorrelated:

$$\begin{aligned} E_{\text{com}}\left[\varepsilon_m(\mathbf{x})\right] &= 0 \\ E_{\text{com}}\left[\varepsilon_m(\mathbf{x})\varepsilon_l(\mathbf{x})\right] &= 0 \qquad m \neq l \end{aligned}$$

- Then we have:

$$E_{\text{com}} = \frac{1}{M}E_{\text{av}}$$

# Averaging models reduces expected error

- So if we average $M$ models we reduce the expected error of the single models by a factor of $1/M$!

$$E_{\text{com}} = \frac{1}{M} E_{\text{av}}$$

- Remember the assumptions, though: errors are almost never uncorrelated.
- However, there are guarantees that the committee error will never exceed the expected model error (i.e. $E_{\text{com}} \leq E_{\text{av}}$).

# SKIPPABLE Boosting

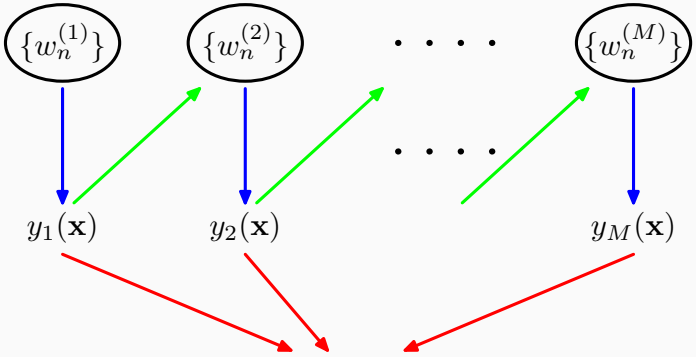
# SKIPPABLE Weighted averaging

- The committee model computes a simple average over base models:

$$y_{\text{com}}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} y_m(\mathbf{x})$$

- These models are fit by bagging: multiple samples from training set with replacement.
- The base models are independently trained on the bootstrap datasets (i.e. the samples from bagging).
- What if we don't treat all models as equal in the committee?
- What if we train them in sequence so they "help each other out" in some way?

- This is precisely what boosting does:



# SKIPPABLE AdaBoost

- The AdaBoost algorithm does this for binary classification problems.
- Assume we have a dataset $\mathcal{D} = \{ (\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \}$ for $t_n \in \{ -1, 1 \}$.
- The AdaBoost idea:
  - We associate with each sample a weight, which is initially $\frac{1}{N}$.
  - Assume we have a way to train a base classifier with weighted samples.
  - After training $M$ base classifiers we combine them into a committee with coefficients giving different weights to each classifier.
- Note: in boosting, these base classifiers are usually called weak learners.

# SKIPPABLE AdaBoost

AdaBoost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

   and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}. \tag{14.17}$$

   (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

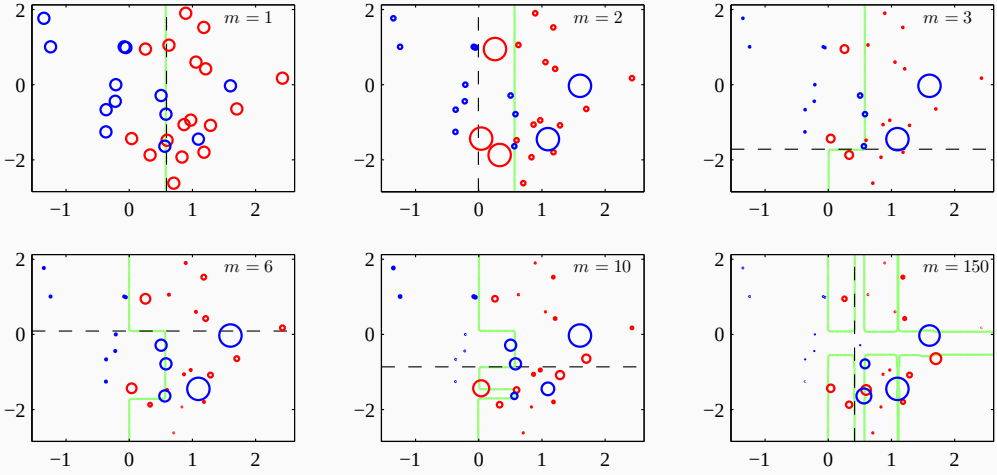
3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}) \right). \tag{14.19}$$

# SKIPPABLE AdaBoost

- The $\varepsilon_m$ represent the (weighted) error rates of the classifier $y_m$.
- The weighting coefficients $\alpha_m$ assign higher weight to more accurate classifiers in (14.19).
- And are used to give higher weight to misclassified samples in (14.18).
- AdaBoost works very well in practice, especially with many, potentially very weak learners.

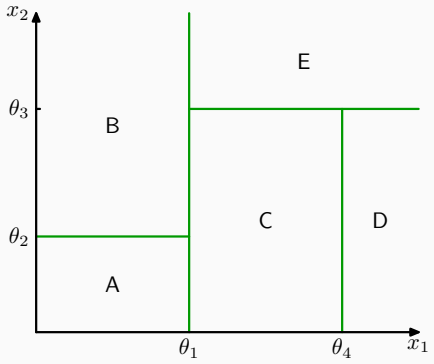
# SKIPPABLE AdaBoost

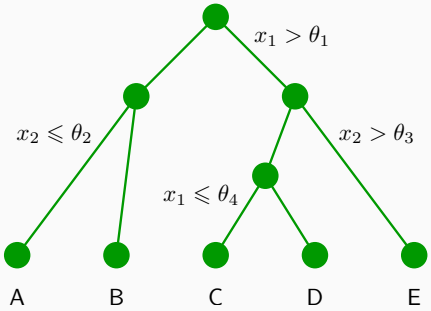


# Tree Models

# Recursive space partitioning

- We can think of classification (also regression) as making constant predictions on partitions of the input space:



# Decision Trees

- Tree models are simple but *widely used* models that work by partitioning.
- Axis-aligned trees partition the space into cuboids aligned with the axes.
- They are ensemble models: a single model is responsible for each partition.
- To select which model we traverse the tree in a sequential decision process.
- Tree models are often considered interpretable by humans.
- The model we consider is the Classification and Regression Tree (CART).

# Learning a tree model

- Consider a again a regression problem with dataset $\mathcal{D} = \{ (\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \}$, where now our $t_n$ are continuous targets.
- We wish to partition the space $\mathbb{R}^D$ so that in each partition our estimate of the target there minimizes the squared error.
- This is pretty easy: we just need the average of the targets $t_n$ of all points $\mathbf{x}_n$ falling in that partition.
- The real problem is *how should we determine this partitioning*.
- Optimizing over all possible partitionings is combinatorially intractable, even for axis-aligned trees...

# Greedy learning

- Instead, we will use a recursive, greedy policy:
  1. Start from a single root node corresponding to the whole space.
  2. Iterate over each of the $D$ possible splitting variables:
     2.1 Consider each possible threshold to split the data into two sets.
     2.2 Select the one that minimizes the sum-of-squared errors in the splits.
  3. Recursively apply this procedure to the children.
- When should we stop this procedure?

# Greedy learning

- Instead, we will use a recursive, greedy policy:
  1. Start from a single root node corresponding to the whole space.
  2. Iterate over each of the $D$ possible splitting variables:
     2.1 Consider each possible threshold to split the data into two sets.
     2.2 Select the one that minimizes the sum-of-squared errors in the splits.
  3. Recursively apply this procedure to the children.
- When should we stop this procedure?
- The usual strategy is to oversegment the input space by building a deep and broad tree.
- Why might this be problematic?

# Greedy learning

- Instead, we will use a recursive, greedy policy:
  1. Start from a single root node corresponding to the whole space.
  2. Iterate over each of the $D$ possible splitting variables:
     2.1 Consider each possible threshold to split the data into two sets.
     2.2 Select the one that minimizes the sum-of-squared errors in the splits.
  3. Recursively apply this procedure to the children.
- When should we stop this procedure?
- The usual strategy is to oversegment the input space by building a deep and broad tree.
- Why might this be problematic?
- And then prune back the tree to balance complexity and error minimization on the training data.

# Greedy learning

- Assume we already have a partitioning (a tree), with leaf nodes indexed by $\tau = 1, \ldots, T$.
- Call the partition corresponding to each leaf $\mathcal{R}_\tau$.
- The optimal prediction and squared error in at each node are:

$$y_\tau = \frac{1}{N_r} \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} t_n \qquad Q_\tau(T) = \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} (t_n - y_\tau)^2$$

- So we can define a pruning criterion like:

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda |T|$$

# For classification problems

- If we have a classification problem, we just need to change the error.
- Define $p_{\tau k}$ as the proportion of samples of class $k$ at node $\tau$.
- For example the negative cross entropy:

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k} \ln p_{\tau k}$$

- Or the Gini Index:

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k}(1 - p_{\tau k})$$

# Concluding Remarks

# The power of consensus (and diversity!)

- Ensemble models are a powerful tool for combining simple models into better ones.
- Committees reduce the average error rate – but only if the assumptions are valid.
- Trees are interpretable models that partition the input space into homogeneous regions.

# Reading and Homework Assignments

Reading Assignment:

- Bishop: Chapter 14 (14.1, 14.2, 14.3, 14.4, 14.5)
- Scikit-learn User Guide (chapter 1.11)