COMS30035, Machine learning: Ensemble methods

James Cussens

School of Computer Science
University of Bristol

13th November 2024

Acknowledgement

- These slides are adapted from ones originally created by Edwin Simpson.

Agenda

- Model Selection
- Model Averaging
- Ensembles: Bagging
- Ensembles: Boosting and Stacking
- Tree-based Models
- Mixtures of Experts

The Model Selection Problem

- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

The Model Selection Problem

- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

The Model Selection Problem

- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

The Model Selection Problem

- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

The Model Selection Problem

- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

The Model Selection Problem

- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

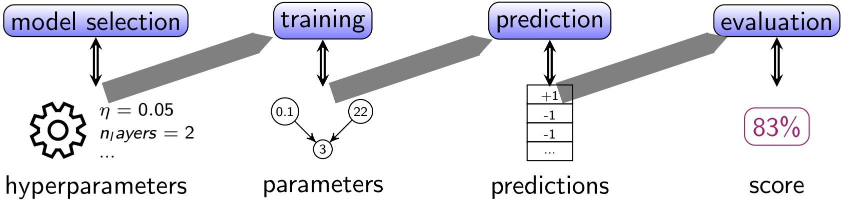
The Model Selection Problem

- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

The Model Selection Problem

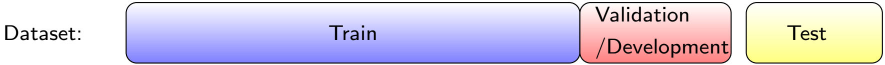
- Select a model h from a set of possible models in a set H,
- The models in H can differ in various ways, such as:
  - The structure of the model, e.g., differences in graphical models;
  - The choice of prior distribution $p(\theta)$ over model parameters;
  - The learning algorithm used, e.g., EM, MCMC, backpropagation,;
  - Parameters of the learning algorithm, like learning rate for stochastic gradient descent (SGD);
  - The features of the data used as inputs to the model;
  - The examples in the dataset the model is trained on;
  - Random initialisation of parameters for EM or SGD

Hyperparameters



- It's useful to characterise all of these modelling decisions as *hyperparameters*
- Hyperparameters = all modelling choices that are fixed before training

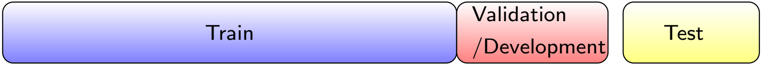
Model Selection on a Validation Set



- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
  - Can't tune on the training set as it would lead to overfitting
- Strength: optimises a performance metric we really care about.
- Weakness: we didn't use all the data in training;
- Weakness: if the validation set is small, we might choose the wrong model!

Model Selection on a Validation Set

Dataset:



Train

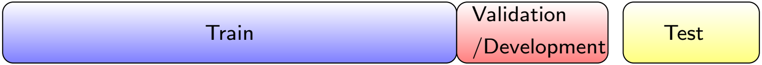
Validation /Development

Test

- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
  - Can't tune on the training set as it would lead to overfitting
- Strength: optimises a performance metric we really care about.
- Weakness: we didn't use all the data in training;
- Weakness: if the validation set is small, we might choose the wrong model!

Model Selection on a Validation Set

Dataset:



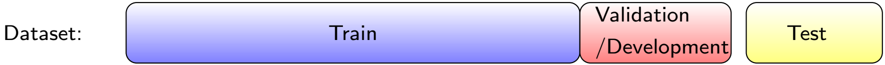
Train

Validation /Development

Test

- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
    - Can't tune on the training set as it would lead to overfitting
- Strength: optimises a performance metric we really care about.
- Weakness: we didn't use all the data in training;
- Weakness: if the validation set is small, we might choose the wrong model!

Model Selection on a Validation Set



Dataset:

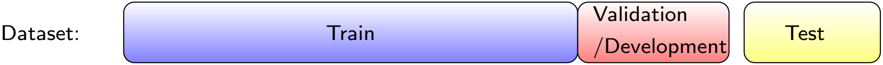
Train

Validation /Development

Test

- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
  - Can't tune on the training set as it would lead to overfitting
- Strength: optimises a performance metric we really care about.
- Weakness: we didn't use all the data in training;
- Weakness: if the validation set is small, we might choose the wrong model!

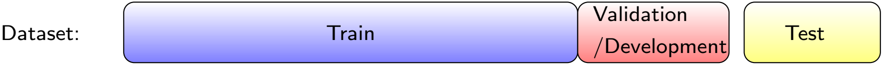
Model Selection on a Validation Set



Dataset: Train | Validation /Development | Test

- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
- How do we come up with H?
- Random search: test random combinations of hyperparameters
- Grid search: For each hyperparameter, define a set of values to test
  - Use your knowledge of the problem to test only reasonable values
  - For numerical hyperparameters, e.g., learning rate, choose a set of evenly-spaced values within a sensible range
  - H contains all combinations of the chosen hyperparameter values
- Reduce the number of tests needed to find a good combination using a more intelligent strategy such as Bayesian Optimisation

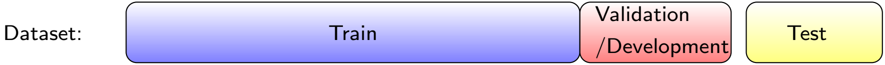
Model Selection on a Validation Set



Dataset: Train | Validation /Development | Test

- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
- How do we come up with H?
- Random search: test random combinations of hyperparameters
- Grid search: For each hyperparameter, define a set of values to test
  - Use your knowledge of the problem to test only reasonable values
  - For numerical hyperparameters, e.g., learning rate, choose a set of evenly-spaced values within a sensible range
  - H contains all combinations of the chosen hyperparameter values
- Reduce the number of tests needed to find a good combination using a more intelligent strategy such as Bayesian Optimisation

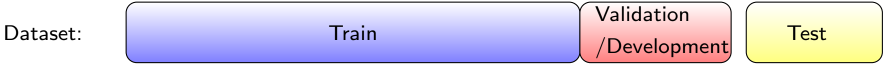
Model Selection on a Validation Set



Dataset: Train | Validation /Development | Test

- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
- How do we come up with H?
- Random search: test random combinations of hyperparameters
- Grid search: For each hyperparameter, define a set of values to test
  - Use your knowledge of the problem to test only reasonable values
  - For numerical hyperparameters, e.g., learning rate, choose a set of evenly-spaced values within a sensible range
  - H contains all combinations of the chosen hyperparameter values
- Reduce the number of tests needed to find a good combination using a more intelligent strategy such as Bayesian Optimisation

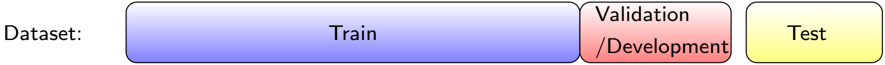
Model Selection on a Validation Set



Dataset: Train | Validation /Development | Test

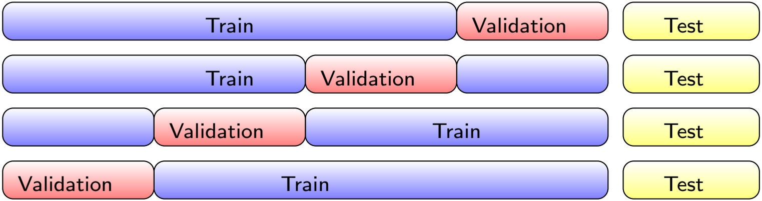
- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
- How do we come up with H?
- Random search: test random combinations of hyperparameters
- Grid search: For each hyperparameter, define a set of values to test
  - Use your knowledge of the problem to test only reasonable values
  - For numerical hyperparameters, e.g., learning rate, choose a set of evenly-spaced values within a sensible range
  - H contains all combinations of the chosen hyperparameter values
- Reduce the number of tests needed to find a good combination using a more intelligent strategy such as Bayesian Optimisation

Model Selection on a Validation Set


Dataset: Train | Validation /Development | Test

- Train a set of models H with different hyperparameters
- Choose the model h that maximises performance on a validation set.
- How do we come up with H?
- Random search: test random combinations of hyperparameters
- Grid search: For each hyperparameter, define a set of values to test
  - Use your knowledge of the problem to test only reasonable values
  - For numerical hyperparameters, e.g., learning rate, choose a set of evenly-spaced values within a sensible range
  - H contains all combinations of the chosen hyperparameter values
- Reduce the number of tests needed to find a good combination using a more intelligent strategy such as Bayesian Optimisation

Cross-validation



Train | Validation | Test

Train | Validation | | Test

| Validation | Train | Test

Validation | Train | Test

- Split the training data into k random, equally-sized subsets;
- For each of the k folds: leave out the kth subset from training, train on the rest and test on the kth subset;
- Compute the average performance across all k folds;
- Avoids overfitting by tuning on training set performance...
- And avoids tuning on a single small validation set.

- Model Selection
- Model Averaging
- Ensembles: Bagging
- Ensembles: Boosting and Stacking
- Tree-based Models
- Mixtures of Experts

Bayesian Model Averaging (BMA)

- Model selection does not always pick out one model, h, as a clear winner
- We may be *uncertain* about which model is correct
- We can express uncertainty by assigning a probability to each model given the training data, $p(h|\boldsymbol{X})$.

Bayesian Model Averaging (BMA)

- Model selection does not always pick out one model, h, as a clear winner
- We may be *uncertain* about which model is correct
- We can express uncertainty by assigning a probability to each model given the training data, $p(h|\boldsymbol{X})$.

Bayesian Model Averaging (BMA)

- Rather than choosing a single model, we can now take an expectation.
- Our predictions now come from a *weighted sum* over models, where $p(h|\boldsymbol{X})$ are weights :

$$p(\boldsymbol{z}|\boldsymbol{X}) = \sum_{h=1}^{H} p(\boldsymbol{z}|\boldsymbol{X}, h)p(h|\boldsymbol{X}) \tag{1}$$

Bayesian Model Averaging (BMA)

- Apply Bayes' rule to estimate the weights:

$$p(h|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|h)p(h)}{\sum_{h'=1}^{H} p(\boldsymbol{X}|h')p(h')} \tag{2}$$

- What happens as we increase the amount of data in $\boldsymbol{X}$?
- $p(h|\boldsymbol{X})$ becomes more focussed on one model.
- So BMA is soft model selection, it does not *combine* models to make a more powerful model.

Bayesian Model Averaging (BMA)

- Apply Bayes' rule to estimate the weights:

$$p(h|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|h)p(h)}{\sum_{h'=1}^{H} p(\boldsymbol{X}|h')p(h')} \tag{2}$$

- What happens as we increase the amount of data in $\boldsymbol{X}$?
- $p(h|\boldsymbol{X})$ becomes more focussed on one model.
- So BMA is soft model selection, it does not *combine* models to make a more powerful model.

Ensemble Methods

- Model Selection
- Model Averaging
- Ensembles: Bagging
- Ensembles: Boosting and Stacking
- Tree-based Models
- Mixtures of Experts

Wisdom of the crowd

Guess the weight!

In 1907, Sir Francis Galton asked 787 villagers to guess the weight of an ox. None of them got the right answer, but when Galton averaged their guesses, he arrived at a near perfect estimate.

The combination was more effective than any one 'model'.

Wisdom of the crowd

Guess the weight!
In 1907, Sir Francis Galton asked 787 villagers to guess the weight of an ox. None of them got the right answer, but when Galton averaged their guesses, he arrived at a near perfect estimate.
The combination was more effective than any one 'model'.

Wisdom of the crowd

Guess the weight!
In 1907, Sir Francis Galton asked 787 villagers to guess the weight of an ox. None of them got the right answer, but when Galton averaged their guesses, he arrived at a near perfect estimate.
The combination was more effective than any one 'model'.

Ensemble Methods

- Ensemble: a combination of different models.
- Often outperforms the average individual, and sometimes even the best individual.
- Different principle to BMA:
  - BMA weights try to identify a single, correct model
  - BMA weights do not provide the optimal combination

Expected Error of an Ensemble

- Given a set of models, $1, ..., M$,
- $y_m(\boldsymbol{x})$ is the prediction from model m.
- Simple ensemble: the mean of the individual predictions, $y_{COM} = \frac{1}{M}\sum_{m=1}^{M} y_m(\boldsymbol{x})$,
- Let's compare the sum-of-squares error of y_{COM} with that of the individual models...

Expected Error of an Ensemble

- Given a set of models, $1, ..., M$,
- $y_m(\boldsymbol{x})$ is the prediction from model m.
- Simple ensemble: the mean of the individual predictions, $y_{COM} = \frac{1}{M} \sum_{m=1}^{M} y_m(\boldsymbol{x})$,
- Let's compare the sum-of-squares error of y_{COM} with that of the individual models...

Expected Error of an Ensemble

Firstly, the error of our combination for a particular input $\boldsymbol{x}$ is:

$$(y(\boldsymbol{x}) - y_{COM}(\boldsymbol{x}))^2 = \left(\frac{1}{M} \sum_{m=1}^{M} (y(\boldsymbol{x}) - y_m(\boldsymbol{x})) \right)^2 . \tag{3}$$

Expected Error of an Ensemble

Firstly, the expected error of our combination is:

$$E_{COM} = \mathbb{E}_{\boldsymbol{x}}[(y(\boldsymbol{x}) - y_{COM}(\boldsymbol{x}))^2] = \mathbb{E}_{\boldsymbol{x}}\left[\left(\frac{1}{M}\sum_{m=1}^{M}(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))\right)^2\right]. \tag{4}$$

Expected Error of an Ensemble

- The expected error of an individual model *m* is: $E_m = \mathbb{E}_{\boldsymbol{x}}[(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))^2]$.

Expected Error of an Ensemble

- The **average** expected error of an individual model is: $E_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\boldsymbol{x}}\left[(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))^2\right]$.
- Remember: $E_{COM} = \mathbb{E}_{\boldsymbol{x}}\left[\left(\frac{1}{M}\sum_{m=1}^{M}(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))\right)^2\right]$.
- ...so we have $E_{COM} = \frac{1}{M}E_{AV}$, since for E_{COM}, the $\frac{1}{M}$ is squared
- This relies on two assumptions...
  1. The errors of each model have zero mean;
  2. The errors of different models are not correlated;
- Intuition: if models make different, random errors, they will tend to cancel out.

Expected Error of an Ensemble

- The **average** expected error of an individual model is: $E_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\boldsymbol{x}}\left[(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))^2\right]$.
- Remember: $E_{COM} = \mathbb{E}_{\boldsymbol{x}}\left[\left(\frac{1}{M}\sum_{m=1}^{M}(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))\right)^2\right]$.
- ...so we have $E_{COM} = \frac{1}{M}E_{AV}$, since for E_{COM}, the $\frac{1}{M}$ is squared
- This relies on two assumptions...
  1. The errors of each model have zero mean;
  2. The errors of different models are not correlated;
- Intuition: if models make different, random errors, they will tend to cancel out.

Expected Error of an Ensemble

- The **average** expected error of an individual model is: $E_{AV} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\boldsymbol{x}}\left[(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))^2\right]$.
- Remember: $E_{COM} = \mathbb{E}_{\boldsymbol{x}}\left[\left(\frac{1}{M}\sum_{m=1}^{M}(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))\right)^2\right]$.
- ...so we have $E_{COM} = \frac{1}{M}E_{AV}$, since for E_{COM}, the $\frac{1}{M}$ is squared
- This relies on two assumptions...
  1. The errors of each model have zero mean;
  2. The errors of different models are not correlated;
- Intuition: if models make different, random errors, they will tend to cancel out.

Expected Error of an Ensemble

- The **average** expected error of an individual model is: $E_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\boldsymbol{x}}\left[(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))^2\right]$.
- Remember: $E_{COM} = \mathbb{E}_{\boldsymbol{x}}\left[\left(\frac{1}{M}\sum_{m=1}^{M}(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))\right)^2\right]$.
- ...so we have $E_{COM} = \frac{1}{M}E_{AV}$, since for E_{COM}, the $\frac{1}{M}$ is squared
- This relies on two assumptions...
  1. The errors of each model have zero mean;
  2. The errors of different models are not correlated;
- Intuition: if models make different, random errors, they will tend to cancel out.

Expected Error of an Ensemble

- The **average** expected error of an individual model is: $E_{AV} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{\boldsymbol{x}} \left[(y(\boldsymbol{x}) - y_m(\boldsymbol{x}))^2 \right]$.
- Remember: $E_{COM} = \mathbb{E}_{\boldsymbol{x}} \left[\left(\frac{1}{M} \sum_{m=1}^{M} (y(\boldsymbol{x}) - y_m(\boldsymbol{x})) \right)^2 \right]$.
- ...so we have $E_{COM} = \frac{1}{M} E_{AV}$, since for E_{COM}, the $\frac{1}{M}$ is squared
- This relies on two assumptions...
  1. The errors of each model have zero mean;
  2. The errors of different models are not correlated;
- Intuition: if models make different, random errors, they will tend to cancel out.

Expected Error of an Ensemble

- $E_{COM} = \frac{1}{M} E_{AV}$ is pretty amazing, but is it realistic?
- No, because we have made extreme assumptions about the models' errors – in practice, they are usually highly correlated and biased.
- However, the combined error cannot be worse than the average error:$E_{COM} \leq E_{AV}$[1]
- The results tells us that the models should be *diverse* to avoid repeating the same errors.

[1] This bound is due to *Jensen's inequality*.

Expected Error of an Ensemble

- $E_{COM} = \frac{1}{M} E_{AV}$ is pretty amazing, but is it realistic?
- No, because we have made extreme assumptions about the models' errors – in practice, they are usually highly correlated and biased.
- However, the combined error cannot be worse than the average error:$E_{COM} \leq E_{AV}$[1]
- The results tells us that the models should be *diverse* to avoid repeating the same errors.

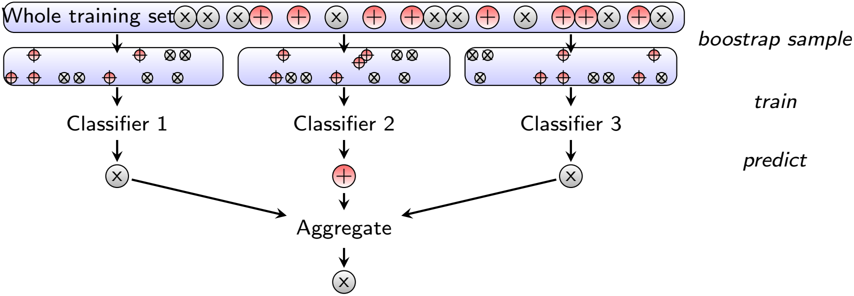
[1] This bound is due to *Jensen's inequality*.

Expected Error of an Ensemble

- $E_{COM} = \frac{1}{M} E_{AV}$ is pretty amazing, but is it realistic?
- No, because we have made extreme assumptions about the models' errors – in practice, they are usually highly correlated and biased.
- However, the combined error cannot be worse than the average error:$E_{COM} \leq E_{AV}$[1]
- The results tells us that the models should be *diverse* to avoid repeating the same errors.

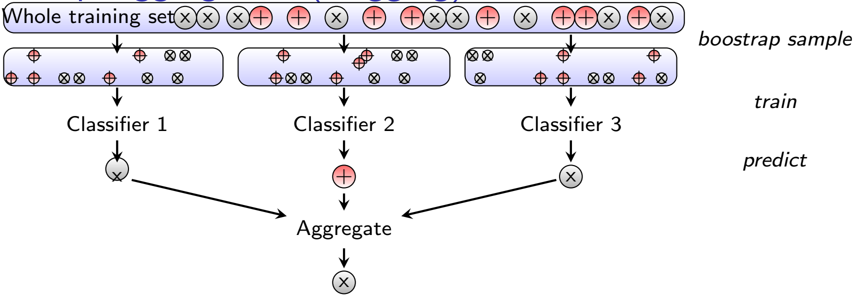
[1] This bound is due to *Jensen's inequality*.

Bootstrap Aggregation (Bagging)



- Create diversity by training models on different samples of the training set.

Bootstrap Aggregation (Bagging)



- Create diversity by training models on different samples of the training set.
- For each model m, randomly sample N data points with replacement from a training set with N data points and train m on the subsample.
- In each sample, some data points will be repeated and others will be omitted.
- Combine predictions by taking the mean or majority vote.

Boosting

- Can we do better than choosing training sets at random?
- Train *base* models sequentially, ensuring that each base model addresses the weaknesses of the ensemble.
- Instead of random sampling, weight the data points in the training set according to the performance of previous base models.
- *AdaBoost* is a popular boosting method originally designed for *binary classification*.

Boosting

- Can we do better than choosing training sets at random?
- Train *base* models sequentially, ensuring that each base model addresses the weaknesses of the ensemble.
- Instead of random sampling, weight the data points in the training set according to the performance of previous base models.
- *AdaBoost* is a popular boosting method originally designed for *binary classification*.

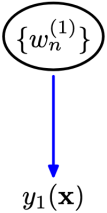
Boosting

- Can we do better than choosing training sets at random?
- Train *base* models sequentially, ensuring that each base model addresses the weaknesses of the ensemble.
- Instead of random sampling, weight the data points in the training set according to the performance of previous base models.
- *AdaBoost* is a popular boosting method originally designed for *binary classification*.

AdaBoost

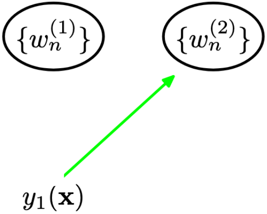
Training sequence →

train new classifier on weighted data that outputs class labels $+1$ or -1



AdaBoost

Training sequence →



$\{w_n^{(1)}\}$

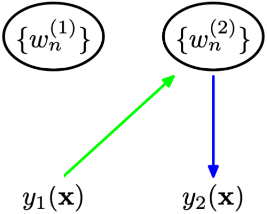
$\{w_n^{(2)}\}$

$y_1(\mathbf{x})$

compute weights from performance of previous classifier

AdaBoost

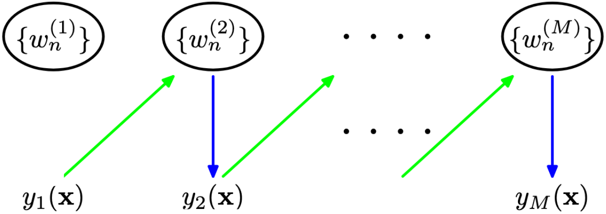
Training sequence →



compute weights from performance of previous classifier

AdaBoost

Training sequence →

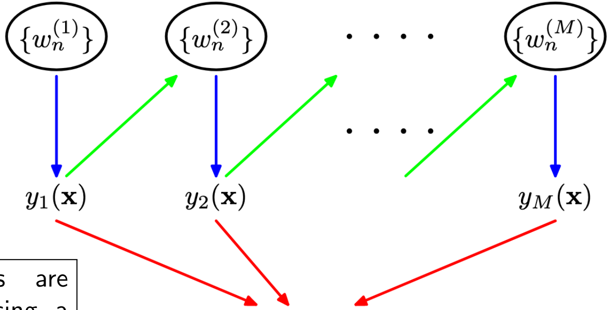


compute weights from performance of previous classifier

$$\mathrm{n}\left(\sum_{m}^{M} \alpha_m y_m(\mathbf{x})\right)$$

AdaBoost

Training sequence →



after all classifiers are trained, combine using a weighted sum

$$Y_M(\mathbf{x}) = \operatorname{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

AdaBoost: Data Weights

1. Initialise $w_n^{(1)} = \frac{1}{N}$ for all data points n;
2. Compute the weighted error rate of model m:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} [y_m(\mathbf{x}_n) \neq y(\mathbf{x}_n)]}{\sum_{n=1}^{N} w_n^{(m)}} \tag{5}$$

3. Update the weight for each data point n:

$$w_n^{(m+1)} = \begin{cases} w_n^{(m)} \left(\frac{1-\epsilon_m}{\epsilon_m}\right) & \text{if } y_m(\mathbf{x}_n) \neq y(\mathbf{x}_n) \\ w_n^{(m)} & \text{if } y_m(\mathbf{x}_n) = y(\mathbf{x}_n) \end{cases} \tag{6}$$

- The weight is increased when m makes an incorrect prediction (assuming $\epsilon_m < \frac{1}{2}$).
- Let's look at Bishop p.660.

AdaBoost: Data Weights

1. Initialise $w_n^{(1)} = \frac{1}{N}$ for all data points n;
2. Compute the weighted error rate of model m:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} [y_m(\boldsymbol{x}_n) \neq y(\boldsymbol{x}_n)]}{\sum_{n=1}^{N} w_n^{(m)}} \tag{5}$$

3. Update the weight for each data point n:

$$w_n^{(m+1)} = \begin{cases} w_n^{(m)} \left(\frac{1-\epsilon_m}{\epsilon_m}\right) & \text{if } y_m(\boldsymbol{x}_n) \neq y(\boldsymbol{x}_n) \\ w_n^{(m)} & \text{if } y_m(\boldsymbol{x}_n) = y(\boldsymbol{x}_n) \end{cases} \tag{6}$$

- The weight is increased when m makes an incorrect prediction (assuming $\epsilon_m < \frac{1}{2}$).
- Let's look at Bishop p.660.

AdaBoost: Data Weights

1. Initialise $w_n^{(1)} = \frac{1}{N}$ for all data points n;
2. Compute the weighted error rate of model m:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} [y_m(\boldsymbol{x}_n) \neq y(\boldsymbol{x}_n)]}{\sum_{n=1}^{N} w_n^{(m)}} \tag{5}$$

3. Update the weight for each data point n:

$$w_n^{(m+1)} = \begin{cases} w_n^{(m)} \left(\frac{1-\epsilon_m}{\epsilon_m}\right) & \text{if } y_m(\boldsymbol{x}_n) \neq y(\boldsymbol{x}_n) \\ w_n^{(m)} & \text{if } y_m(\boldsymbol{x}_n) = y(\boldsymbol{x}_n) \end{cases} \tag{6}$$

- The weight is increased when m makes an incorrect prediction (assuming $\epsilon_m < \frac{1}{2}$).
- Let's look at Bishop p.660.

AdaBoost: Final Classifier Weights

$$y_M(\boldsymbol{x}_n) = \sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x}_n) \tag{7}$$

- Choosing the weight α_m for m that minimises the exponential loss of m gives us:

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\} \tag{8}$$

- Weights are higher for classifiers with a lower error rate.
- Note that α_m is a log-odds function: AdaBoost optimises the approximation to the log-odds ratio.
- Other loss functions can be used to derive similar boosting schemes for regression and multi-class classification.

AdaBoost: Final Classifier Weights

$$y_M(\boldsymbol{x}_n) = \sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x}_n) \tag{7}$$

- Choosing the weight α_m for m that minimises the exponential loss of m gives us:

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\} \tag{8}$$

- Weights are higher for classifiers with a lower error rate.
- Note that α_m is a log-odds function: AdaBoost optimises the approximation to the log-odds ratio.
- Other loss functions can be used to derive similar boosting schemes for regression and multi-class classification.

AdaBoost: Final Classifier Weights

$$y_M(\boldsymbol{x}_n) = \sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x}_n) \tag{7}$$

- Choosing the weight α_m for m that minimises the exponential loss of m gives us:

$$\alpha_m = \ln \left\{ \frac{1-\epsilon_m}{\epsilon_m} \right\} \tag{8}$$

- Weights are higher for classifiers with a lower error rate.
- Note that α_m is a log-odds function: AdaBoost optimises the approximation to the log-odds ratio.
- Other loss functions can be used to derive similar boosting schemes for regression and multi-class classification.

AdaBoost: Final Classifier Weights

$$y_M(\boldsymbol{x}_n) = \sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x}_n) \tag{7}$$

- Choosing the weight α_m for m that minimises the exponential loss of m gives us:

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\} \tag{8}$$

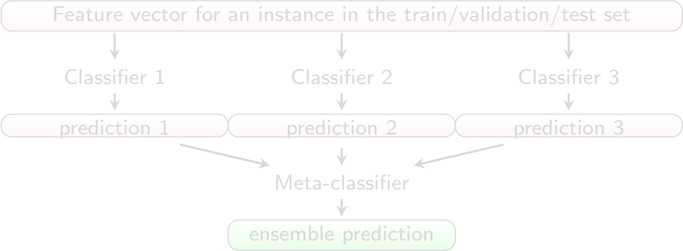
- Weights are higher for classifiers with a lower error rate.
- Note that α_m is a log-odds function: AdaBoost optimises the approximation to the log-odds ratio.
- Other loss functions can be used to derive similar boosting schemes for regression and multi-class classification.

Stacking

Given a set of trained base classifiers, what's the best way to combine them?

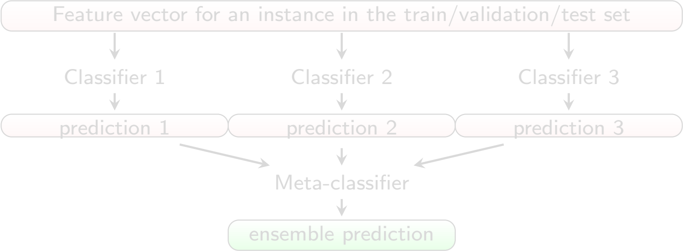
Stacking

- Given a trained set of base classifiers, learn the combination function!
- Bagging: the combination function was majority vote, which is *unweighted*;
- Adaboost: a *weighted* sum of classifier outputs determined by model error rates;
- Stacking: use another classifier/regressor to learn a combination function that minimises the error rate of the entire ensemble



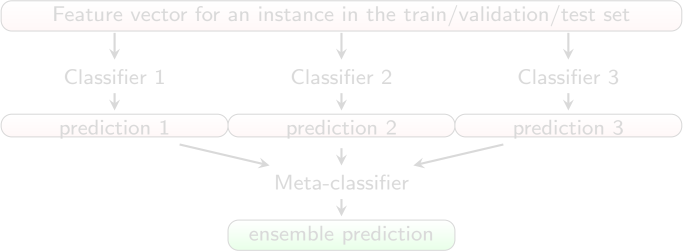
Stacking

- Given a trained set of base classifiers, learn the combination function!
- Bagging: the combination function was majority vote, which is *unweighted*;
- Adaboost: a *weighted* sum of classifier outputs determined by model error rates;
- Stacking: use another classifier/regressor to learn a combination function that minimises the error rate of the entire ensemble



Stacking

- Given a trained set of base classifiers, learn the combination function!
- Bagging: the combination function was majority vote, which is *unweighted*;
- Adaboost: a *weighted* sum of classifier outputs determined by model error rates;
- Stacking: use another classifier/regressor to learn a combination function that minimises the error rate of the entire ensemble


Feature vector for an instance in the train/validation/test set

Classifier 1

Classifier 2

Classifier 3

prediction 1

prediction 2

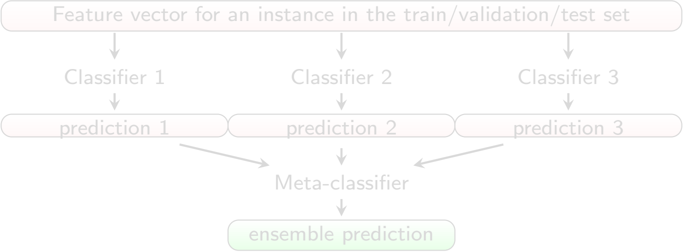
prediction 3

Meta-classifier

ensemble prediction

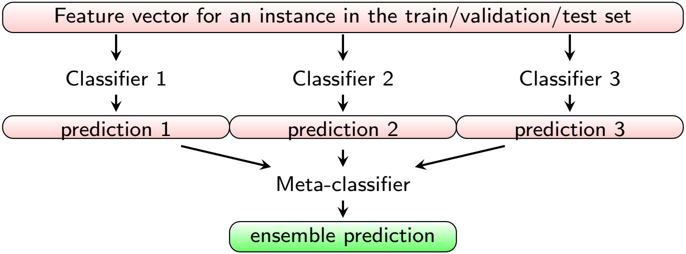
Stacking

- Given a trained set of base classifiers, learn the combination function!
- Bagging: the combination function was majority vote, which is *unweighted*;
- Adaboost: a *weighted* sum of classifier outputs determined by model error rates;
- Stacking: use another classifier/regressor to learn a combination function that minimises the error rate of the entire ensemble



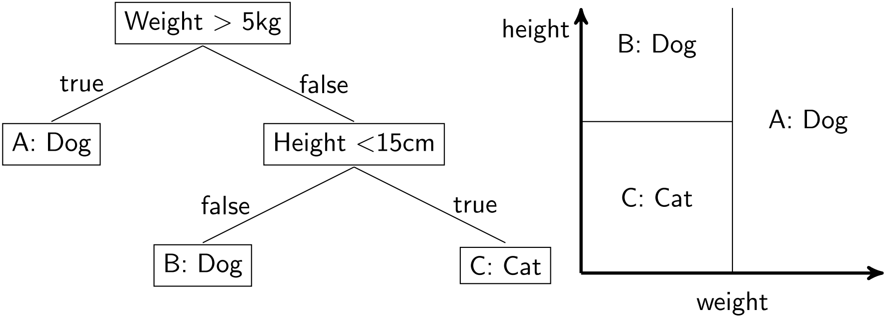
Stacking

- Given a trained set of base classifiers, learn the combination function!
- Bagging: the combination function was majority vote, which is *unweighted*;
- Adaboost: a *weighted* sum of classifier outputs determined by model error rates;
- Stacking: use another classifier/regressor to learn a combination function that minimises the error rate of the entire ensemble



- Model Selection
- Model Averaging
- Ensembles: Bagging
- Ensembles: Boosting and Stacking
- Tree-based Models
- Mixtures of Experts

Decision Trees



Random Forest – Adapting Bagging to Trees

- With bagging, base models make similar splits on the same features – the strongest predictors – meaning their errors become correlated
- Random forest modifies the training procedure for each tree, m:
  1. Randomly sample N data points with replacement from a training set with N data points.
  2. Learn the tree using the greedy CART algorithm but when determining each split, consider only $d \ll D$ randomly-chosen features.
- As with bagging, combine predictions by taking mean/majority vote.
- Extremely effective in many applications

Random Forest – Adapting Bagging to Trees

- With bagging, base models make similar splits on the same features – the strongest predictors – meaning their errors become correlated
- Random forest modifies the training procedure for each tree, m:
  1. Randomly sample N data points with replacement from a training set with N data points.
  2. Learn the tree using the greedy CART algorithm but when determining each split, consider only $d \ll D$ randomly-chosen features.
- As with bagging, combine predictions by taking mean/majority vote.
- Extremely effective in many applications

Random Forest – Adapting Bagging to Trees

- With bagging, base models make similar splits on the same features – the strongest predictors – meaning their errors become correlated
- Random forest modifies the training procedure for each tree, m:
  1. Randomly sample N data points with replacement from a training set with N data points.
  2. Learn the tree using the greedy CART algorithm but when determining each split, consider only $d \ll D$ randomly-chosen features.
- As with bagging, combine predictions by taking mean/majority vote.
- Extremely effective in many applications

Random Forest – Adapting Bagging to Trees

- With bagging, base models make similar splits on the same features – the strongest predictors – meaning their errors become correlated
- Random forest modifies the training procedure for each tree, m:
  1. Randomly sample N data points with replacement from a training set with N data points.
  2. Learn the tree using the greedy CART algorithm but when determining each split, consider only $d \ll D$ randomly-chosen features.
- As with bagging, combine predictions by taking mean/majority vote.
- Extremely effective in many applications

Random Forest – Adapting Bagging to Trees

- With bagging, base models make similar splits on the same features – the strongest predictors – meaning their errors become correlated
- Random forest modifies the training procedure for each tree, m:
  1. Randomly sample N data points with replacement from a training set with N data points.
  2. Learn the tree using the greedy CART algorithm but when determining each split, consider only $d \ll D$ randomly-chosen features.
- As with bagging, combine predictions by taking mean/majority vote.
- Extremely effective in many applications

Agenda

- Model Selection
- Model Averaging
- Ensembles: Bagging
- Ensembles: Boosting and Stacking
- Tree-based Models
- Mixtures of Experts

Recap

- Model selection: choose the right model for the whole dataset → *hard selection*
- Bayesian model averaging (BMA): probabilistically select the right model for the whole dataset → *soft selection*
- Decision trees: split the feature space and model each region by one leaf node → *hard selection depending on features*
- Mixtures of experts: perform a soft, probabilistic split of the feature space → *soft selection depending on features*

Recap

- Model selection: choose the right model for the whole dataset → *hard selection*
- Bayesian model averaging (BMA): probabilistically select the right model for the whole dataset → *soft selection*
- Decision trees: split the feature space and model each region by one leaf node → *hard selection depending on features*
- Mixtures of experts: perform a soft, probabilistic split of the feature space → *soft selection depending on features*

Conditional Mixture Models

- Similar to the mixture models we saw in earlier lectures
- For regression or classification, rather than clustering
- The prediction for the output target variable y depends, i.e. is *conditional* on, the input feature vector $\boldsymbol{x}$
- We will only be looking at one conditional mixture model called *Mixture of Experts*

Mixture of Experts

- Each data point is processed by a weighted combination of specialised 'expert' models
- Weights: probabilities that depend on the features $\boldsymbol{x}_n$ of the data point.
- Think of medical diagnosis: based on the patient's symptoms, a GP refers the patient to a specialist
- If they are unsure what is causing the symptoms, they may send the patient to multiple specialists for examination.
- Similarly, some inputs $\boldsymbol{x}_n$ may require a combination of expert models
- Contrast with decision trees, which assign each data point to a single leaf node

Mixture of Experts

- Each data point is processed by a weighted combination of specialised 'expert' models
- Weights: probabilities that depend on the features $\boldsymbol{x}_n$ of the data point.
- Think of medical diagnosis: based on the patient's symptoms, a GP refers the patient to a specialist
- If they are unsure what is causing the symptoms, they may send the patient to multiple specialists for examination.
- Similarly, some inputs $\boldsymbol{x}_n$ may require a combination of expert models
- Contrast with decision trees, which assign each data point to a single leaf node

Mixture of Experts

- Each data point is processed by a weighted combination of specialised 'expert' models
- Weights: probabilities that depend on the features $\boldsymbol{x}_n$ of the data point.
- Think of medical diagnosis: based on the patient's symptoms, a GP refers the patient to a specialist
- If they are unsure what is causing the symptoms, they may send the patient to multiple specialists for examination.
- Similarly, some inputs $\boldsymbol{x}_n$ may require a combination of expert models
- Contrast with decision trees, which assign each data point to a single leaf node

Mixture of Experts

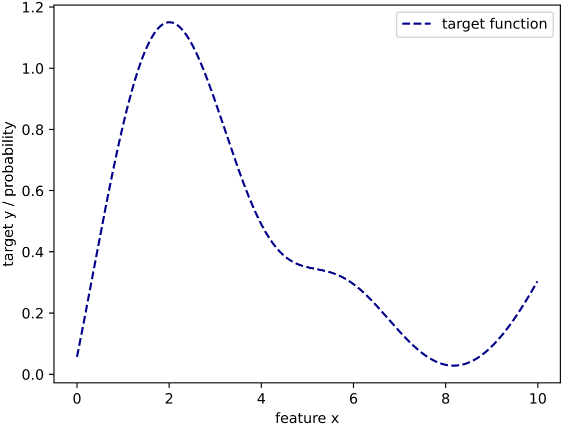
- Each data point is processed by a weighted combination of specialised 'expert' models
- Weights: probabilities that depend on the features $\boldsymbol{x}_n$ of the data point.
- Think of medical diagnosis: based on the patient's symptoms, a GP refers the patient to a specialist
- If they are unsure what is causing the symptoms, they may send the patient to multiple specialists for examination.
- Similarly, some inputs $\boldsymbol{x}_n$ may require a combination of expert models
- Contrast with decision trees, which assign each data point to a single leaf node

Mixture of Experts

- Each data point is processed by a weighted combination of specialised 'expert' models
- Weights: probabilities that depend on the features $\boldsymbol{x}_n$ of the data point.
- Think of medical diagnosis: based on the patient's symptoms, a GP refers the patient to a specialist
- If they are unsure what is causing the symptoms, they may send the patient to multiple specialists for examination.
- Similarly, some inputs $\boldsymbol{x}_n$ may require a combination of expert models
- Contrast with decision trees, which assign each data point to a single leaf node

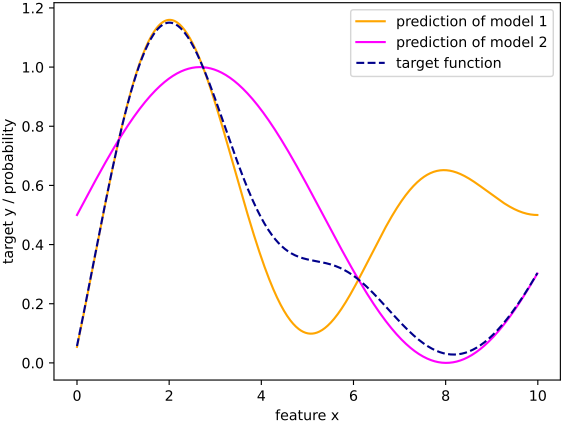
Mixture of Experts

Imagine we would like to learn a model of this function:



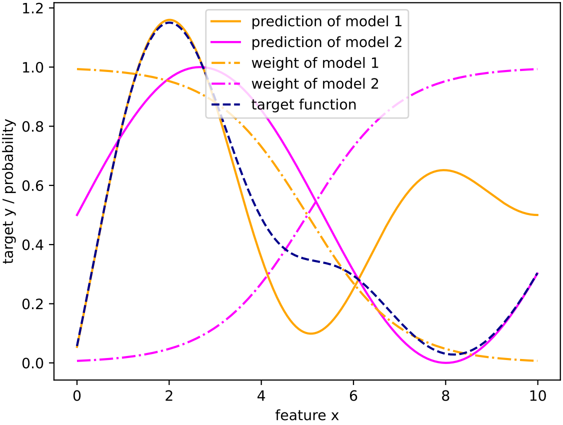
Mixture of Experts

We have two expert models: model 1 is close to our target on the left, model 2 is close to our target on the right.



Mixture of Experts (MoE)

MoE reproduces the target function by learning weights for each model and taking a weighted sum of their predictions.



Mixture of Experts

- Goal: predict target variable t_n given features $\boldsymbol{x}_n$
- Component distribution depends on input feature vector $\boldsymbol{x}_n$.

$$p(t_n|\boldsymbol{x}_n, \phi, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x}_n) p(t_n|\boldsymbol{x}_n, \phi_k) \tag{9}$$

- Note that here, like Bishop, we are assuming that each component provides a distribution $p(t_n|\boldsymbol{x}_n, \phi_k)$ over the target and the mixture thus also gives us a distribution over the target: $p(t_n|\boldsymbol{x}_n, \phi, \boldsymbol{\pi})$. We can use e.g. the mode of this distribution to provide a predicted value for the target.
- $\pi_k(\boldsymbol{x}_n)$ is the weight for model k in a combination of models.
- The weights can be learned as part of EM (see Bishop 14.5.3 for more)

Reading

- Bishop Chapter 14, but you can skip: §14.3.1, §14.3.2, §14.5.1, §14.5.2.
- Murphy §18.2–§18.5.3

Problems and quizzes

- Problems:
  - Bishop 14.2
  - Bishop 14.3
- Quizzes:
  - Week 8: Model Selection and Averaging
  - Week 8: Ensembles
  - Week 8: Conditional Mixture Models