CSC 411: Introduction to Machine Learning
Lecture 4: Ensemble I

Mengye Ren and Matthew MacKay

University of Toronto
We’ve seen two particular learning algorithms: k-NN and decision trees

Next two lectures: **combine multiple models into an ensemble** which performs better than the individual members

- Generic class of techniques that can be applied to almost any learning algorithms...
- ... but are particularly well suited to decision trees

Today

- Understanding generalization using the **bias/variance decomposition**
- Reducing variance using bagging

Next lecture

- Making a weak classifier stronger (i.e. reducing bias) using boosting
An ensemble of predictors is a set of predictors whose individual decisions are combined in some way to predict new examples.

- E.g., (possibly weighted) majority vote

For this to be nontrivial, the learned hypotheses must differ somehow, e.g.

- Different algorithm
- Different choice of hyperparameters
- Trained on different data
- Trained with different weighting of the training examples

Ensembles are usually easy to implement. The hard part is deciding what kind of ensemble you want, based on your goals.
This lecture: **bagging**
- Train classifiers independently on random subsets of the training data.

Next lecture: **boosting**
- Train classifiers sequentially, each time focusing on training examples that the previous ones got wrong.

Bagging and boosting serve very different purposes. To understand this, we need to take a detour to understand the bias and variance of a learning algorithm.
Loss Functions

- A **loss function** $L(y, t)$ defines how bad it is if, for some example $x$, the algorithm predicts $y$, but the target is actually $t$.

- Example: **0-1 loss** for classification

$$L_{0-1}(y, t) = \begin{cases} 
0 & \text{if } y = t \\
1 & \text{if } y \neq t
\end{cases}$$

  - Averaging the 0-1 loss over the training set gives the **training error rate**, and averaging over the test set gives the **test error rate**.

- Example: **squared error loss** for regression

$$L_{SE}(y, t) = \frac{1}{2}(y - t)^2$$

  - The average squared error loss is called **mean squared error (MSE)**.
Recall that overly simple models underfit the data, and overly complex models overfit.

We can quantify this effect in terms of the bias/variance decomposition.

Bias and variance of what?
Bias-Variance Decomposition: Basic Setup

- Suppose the training set $\mathcal{D}$ consists of $N$ pairs $(x_i, t_i)$ sampled independent and identically distributed (i.i.d.) from a single data generating distribution $p_{\text{data}}$.
  
  ▶ Let $p_{\text{train}}$ denote the induced distribution over training sets.

- Pick a fixed query point $x$ (denoted with a green $x$).

- Consider an experiment where we sample lots of training sets independently from $p_{\text{train}}$. 
Bias-Variance Decomposition: Basic Setup

- Let’s run our learning algorithm on each training set $\mathcal{D}$, producing a classifier $h_\mathcal{D}$
- We can compute each classifier’s prediction $h_\mathcal{D}(x) = y$ at the query point $x$.
- $y$ is a random variable, where the randomness comes from the choice of training set
  - $\mathcal{D}$ is random $\implies h_\mathcal{D}$ is random $\implies h_\mathcal{D}(x)$ is random
Bias-Variance Decomposition: Basic Setup

Here is the analogous setup for regression:

Since $y = h_D(x)$ is a random variable, we can talk about its expectation, variance, etc. over the distribution of training sets $p_{\text{train}}$. 
Recap of basic setup:

Assume (for the moment) that $t$ is deterministic given $x$

There is a distribution over the loss at $x$, with expectation $E_{D \sim p_{\text{train}}} [L(h_D(x), t)]$.

For each query point $x$, the expected loss is different. We are interested in quantifying how well our classifier does over the distribution $p_{\text{data}}$, averaging over training sets: $E_{x \sim p_{\text{data}}, D \sim p_{\text{train}}} [L(h_D(x), t)]$. 
Bias-Variance Decomposition

- For now, focus on squared error loss, \( L(y, t) = \frac{1}{2}(y - t)^2 \).

- We can decompose the expected loss (suppressing distributions \( x, D \) drawn from for compactness):

  \[
  \mathbb{E}_{x, D}[(h_D(x) - t)^2] = \mathbb{E}_{x, D}[(h_D(x) - \mathbb{E}_D[h_D(x)] + \mathbb{E}_D[h_D(x)] - t)^2]
  \]

  \[
  = \mathbb{E}_{x, D}[(h_D(x) - \mathbb{E}_D[h_D(x)])^2 + (\mathbb{E}_D[h_D(x)] - t)^2 + 2(h_D(x) - \mathbb{E}_D[h_D(x)])(\mathbb{E}_D[h_D(x)] - t)]
  \]

  \[
  = \mathbb{E}_{x, D}[(h_D(x) - \mathbb{E}_D[h_D(x)])^2] + \mathbb{E}_x[(\mathbb{E}_D[h_D(x)] - t)^2]
  \]

  \[
  \text{variance} + \text{bias}
  \]

- Bias: On average, how close is our classifier to true target? (corresponds to underfitting)

- Variance: How widely dispersed are our predictions as we generate new datasets? (corresponds to overfitting)
Bias and Variance

- Throwing darts = predictions for each draw of a dataset

- What doesn’t this capture?
  - We average over points \( x \) from the data distribution
Now, back to bagging!
Suppose we could somehow sample $m$ independent training sets $\{D_i\}_{i=1}^m$ from $p_{\text{train}}$.

We could then learn a predictor $h_i := h_{D_i}$ based on each one, and take the average $h = \frac{1}{m} \sum_{i=1}^m h_i$.

How does this affect the terms of the expected loss?

- **Bias: unchanged**, since the averaged prediction has the same expectation:

  $$\mathbb{E}_{D_1,\ldots,D_m \sim p_{\text{train}}} [h(x)] = \frac{1}{m} \sum_{i=1}^m \mathbb{E}_{D_i \sim p_{\text{train}}} [h_i(x)] = \mathbb{E}_{D \sim p_{\text{train}}} [h_D(x)]$$

- **Variance: reduced**, since we’re averaging over independent samples:

  $$\text{Var}_{D_1,\ldots,D_m} [h(x)] = \frac{1}{m^2} \sum_{i=1}^m \text{Var}_{D_i} [h_i(x)] = \frac{1}{m} \text{Var}_{D} [h_D(x)].$$
In practice, we don’t have access to the underlying data generating distribution $p_{\text{data}}$.

It is expensive to independently collect many datasets.

Solution: bootstrap aggregation, or bagging.

- Take a single dataset $\mathcal{D}$ with $n$ examples.
- Generate $m$ new datasets, each by sampling $n$ training examples from $\mathcal{D}$, with replacement.
- Average the predictions of models trained on each of these datasets.
Bagging: The Idea

- Problem: the datasets are not independent, so we don’t get the $1/m$ variance reduction.
  - Possible to show that if the sampled predictions have variance $\sigma^2$ and correlation $\rho$, then

$$\text{Var} \left( \frac{1}{m} \sum_{i=1}^{m} h_i(x) \right) = \frac{1}{m} (1 - \rho) \sigma^2 + \rho \sigma^2.$$ 

- Ironically, it can be advantageous to introduce *additional* variability into your algorithm, as long as it reduces the correlation between samples.
  - Intuition: you want to invest in a diversified portfolio, not just one stock.
  - Can help to use average over multiple algorithms, or multiple configurations of the same algorithm.
Random Forests

- **Random forests** = bagged decision trees, with one extra trick to decorrelate the predictions.

- When choosing each node of the decision tree, choose a random set of \( d \) input features, and only consider splits on those features.

- Random forests are probably the best black-box machine learning algorithm — they often work well with no tuning whatsoever.
  - one of the most widely used algorithms in Kaggle competitions.
Let’s return to quantifying expected loss and make the situation slightly more complicated (and realistic): what if \( t \) is not deterministic given \( x \)? i.e. have \( p(t|x) \)

Can no longer measure bias as expected distance from true target, since there’s a distribution over targets!

Instead, we’ll measure distance from \( y_*(x) = \mathbb{E}[t|\mathbf{x}] \)

- This is the best possible prediction, in the sense that it minimizes the expected loss
**Proof:** Start by fixing $x$. Want to show: $\arg\min_y \mathbb{E}_t[(y - t)^2] = y^* = \mathbb{E}_t[t]$

(Distribution of $t$ is $p(t|x)$)

\[
\mathbb{E}_t[(y - t)^2] = \mathbb{E}_t[y^2 - 2yt + t^2]
= y^2 - 2y\mathbb{E}_t[t] + \mathbb{E}_t[t^2]
= y^2 - 2y\mathbb{E}_t[t] + (\mathbb{E}_t[t])^2 + \text{Var}[t | x]
= y^2 - 2yy^* + y^* + \text{Var}[t | x]
= (y - y^*)^2 + \text{Var}[t | x]
\]

- The first term is nonnegative, and can be made 0 by setting $y = y^*$.
- The second term doesn’t depend on $y$! Corresponds to the inherent unpredictability, or **noise**, of the targets, and is called the **Bayes error**.

  - This is the best we can ever hope to do with any learning algorithm. An algorithm that achieves it is **Bayes optimal**.
Bayes Optimality

- We can again decompose the expected loss, this time including \( t \) in our expectation (check this!):

\[
E_{x,D,t}[(h_D(x) - t)^2] = E_x[(E_D[h_D(x)] - y_*(x))^2] + E_{x,D}[(h_D(x) - E_D[h_D(x)])^2] + Var[t | x]
\]

- Contrast if \( t \) is not random:

\[
E_x[(E_D[h_D(x)] - t)^2] + E_{x,D}[(h_D(x) - E_D[h_D(x)])^2]
\]

- We have no control over the Bayes error! In particular, bagging/boosting do not help
Bias/Variance Decomposition: Another Visualization

- We can visualize this decomposition in **output space**, where the axes correspond to predictions on the test examples.
- If we have an overly simple model (e.g. k-NN with large \( k \)), it might have
  - high bias (because it’s too simplistic to capture the structure in the data)
  - low variance (because there’s enough data to get a stable estimate of the decision boundary)
If you have an overly complex model (e.g. k-NN with \( k = 1 \)), it might have

- low bias (since it learns all the relevant structure)
- high variance (it fits the quirks of the data you happened to sample)
Bagging reduces overfitting by averaging predictions.

- Used in most competition winners
  - Even if a single model is great, a small ensemble usually helps.

- Limitations:
  - Does not reduce bias.
  - There is still correlation between classifiers.

- Random forest solution: Add more randomness.