Machine Learning: Ensemble Methods
Ensemble Methods

- An ensemble is a set of classifiers (either different algorithms or different settings of the same algorithm, or the same algorithm on different samples of the dataset) that learn a target function, and their individual predictions are combined to classify new examples.
- Ensembles generally improve the generalization performance of a set of classifiers on a domain.
- Based on the following idea: A large number of relatively uncorrelated models operating as a committee will outperform any of the individual constituent models.
# Example: Weather Forecast

100% CORRECT!

<table>
<thead>
<tr>
<th>GROUND TRUTH</th>
<th>PREDICTOR1</th>
<th>PREDICTOR2</th>
<th>PREDICTOR3</th>
<th>PREDICTOR4</th>
<th>PREDICTOR5</th>
<th>Combine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>![Cloudy]</td>
<td>![Cloudy]</td>
<td>![Cloudy]</td>
<td>![Cloudy]</td>
<td>![Cloudy]</td>
<td>![Cloudy]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
<tr>
<td>![Cloudy]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
<td>![Sunny]</td>
</tr>
</tbody>
</table>
Why to use Ensemble Methods?

● Statistical reasons:
  ➢ A set of classifiers with similar training performances may have different \textit{generalization performances}.
  ➢ Combining outputs of several classifiers \textit{reduces the risk of selecting a poorly performing (weak) classifier}.

● Too large volumes of data:
  ➢ If the amount of data to be analyzed is too large, a single classifier may not be able to handle it; train \textit{different classifiers on different partitions of data}.

● Too little data:
  ➢ Ensemble systems can also be used when there is too little data; \textit{resampling techniques}.
Why to use Ensemble Methods?

- Remember: From the lesson on evaluation, the error of a learning algorithm has three components: the noise, the bias, and the variance:

\[
\text{Error}(x) = \text{Bias}(x)^2 + \text{Variance}(x) + \text{Noise}(X)
\]

➢ The **noise** is the **irreducible error**,  
➢ The **bias** is the **systematic error** that the learning algorithm is expected to make
➢ The **variance** measures the **sensitivity of the algorithm to the specific training set (and/or hyper-parameters) used**.

Ensemble helps to reduce these bias and variance (except noise, which is irreducible error)
Why to use Ensemble Methods?

- When combining multiple independent and diverse decisions each of which is at least more accurate than random guessing, the variance is reduced and so does the error rate.

\[ \text{Var} \left( \text{Ensemble}(h_i(x, D)) \right) = \frac{\sum_m \text{Var}(h_i(x, D_i))}{m} \]

- In practice:

  If we use “simple” models trained on smaller samples, the individual variance of each hypothesis can be higher than for a more complex learner, so we obtain an ensemble variance higher than the best case, but still in most cases lower than if we use one single complex learner.
Example

- Suppose there are 25 “simple” classifiers
  - Each classifier has an average error rate, $e = 0.35$ (which is a mid-high rate)
  - Assume classifiers $h_i(x)$ are independent and final class is predicted with majority voting
  - The probability that the ensemble classifier makes a wrong prediction (it is wrong if at least 13 out of 25 make the wrong prediction):
    \[
    \sum_{i=13}^{25} \binom{25}{i} e^i (1-e)^{25-i} = 0.06
    \]

If classifiers are independent, the probability that the ensemble makes an error is very low!
Learning Ensembles

- Learn multiple alternative models using **different training data** or **different learning algorithms**.
- **Combine** decisions of multiple definitions, e.g. using weighted voting.

![Diagram of Learning Ensembles]

- **Training Data**
  - **Data1**
  - **Data2**
  - **...**
  - **Data m**

- **Learners**
  - **Learner1**
  - **Learner2**
  - **...**
  - **Learner m**

- **Models**
  - **Model1**
  - **Model2**
  - **...**
  - **Model m**

- **Model Combiner**

- **Final Model**
Methods for Constructing Ensembles

● **By manipulating the training set:** Create multiple training sets by resampling the original data according to some sampling distribution.

● **By manipulating the input features:** Choose a subset of input features to form each training set.

● **By manipulating the class labels:** Transform the training data into a binary class problem by randomly partitioning the class labels into two disjoint subsets (e.g. for 4 labels: (AB) (CD)).

● **By manipulating the learning algorithm:** Manipulate the learning algorithm to generate different models (e.g. different hyperparameters)
Homogeneous Ensembles

- Use a **single, arbitrary learning algorithm** but **manipulate training data** to make it learn multiple models \( h_1(x) \), \( h_2(x) \), .. \( h_m(x) \)
  - \( \text{Data1} \neq \text{Data2} \neq \ldots \neq \text{Data m} \)
  - \( \text{Learner1} = \text{Learner2} = \ldots = \text{Learner m} \)

- Different methods for **changing training data**:
  
  A. **Bagging**: Resample data (Useful to reduce the variance)
  
  B. **Boosting**: Re-weight data (Useful to reduce the bias)

  C. **Random Forests**: Combines multiples Dtrees (avoids overfitting)
Bagging

A high variance for a model is not good, suggesting its performance **is sensitive to the training data provided**. So, even if more training data is provided, the model may still perform poorly. And, this may not even reduce the variance of our model.

- **Solution**: BAGGING, a shorthand for the combination of **Bootstrapping** and **aggregating**

![Diagram showing Bagging process]
Bagging: Bootstrapping

Bootstrapping is a method to help decrease the variance of the classifier and reduce overfitting, by resampling data from the training set. The model created should be less overfitted than a single individual model.

- **How**: Each individual classifiers randomly extracts a sample of $m$ instances over the training set of $n$ instances with replacement (instances are put back in the urn, therefore they can be sampled more than one time). Usually, for bootstrapping the sample has the same cardinality as the original set $m=n$, called 0.632 bootstrap.

- **When**: Effective method in limited data contexts (high variance, overfitting).
Bootstrapping

Example with $n=12$ and $m=5$
Instances that are not extracted during bagging, are used for testing.

So each instance has probability of \((1/n)^m\) of being selected in a training sample and \((1 - 1/n)^m\) of being selected as test data, in each bagging round.

<table>
<thead>
<tr>
<th>Data ID</th>
<th>Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Data</td>
<td>1  2  3  4  5  6  7  8  9  10</td>
</tr>
<tr>
<td>Bagging (Round 1)</td>
<td>7  8  10  8  2  5  10  10  5  9</td>
</tr>
<tr>
<td>Bagging (Round 2)</td>
<td>1  4  9  1  2  3  2  7  3  2</td>
</tr>
<tr>
<td>Bagging (Round 3)</td>
<td>1  8  5  10  5  10  9  6  3  7</td>
</tr>
</tbody>
</table>

Original training dataset has \(n=10\) instances. \(n=m\) in this example.
Example

Each instance has a probability $p = 1/n$ of being extracted out of $n$ instances. Since extraction is “with replacement” (the instance is put back in the urn after having been extracted) the probability is always the same at each extraction.
Original Dataset

<table>
<thead>
<tr>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Bootstrap

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
</tbody>
</table>
Original Dataset

1 6
2 7
3 8
4 9
5 10

Bootstrap

1 10
7
3
Original Dataset

1 6
2 7
3 8
4 9
5 10

Bootstrap

1
10
7
3
10
Original Dataset

<table>
<thead>
<tr>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Bootstrap

<table>
<thead>
<tr>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>
Original Dataset

1  6
2  7
3  8
4  9
5  10

Bootstrap

1  6
10 10
7  8
3  1
10  6

Unselected

2
4
5
9

Training set

Test set
The 0.632 bootstrap

- Each example in the original dataset has a selection probability of 1/n
- If n=m on average, 36.8% of the data-points are left unselected and can be used for testing

Why?
The 0.632 bootstrap

This method is also called the 0.632 bootstrap

➢ If we make m extraction on n instances, each instance has a probability $\frac{1}{n}$ of being picked and $1 - \frac{1}{n}$ of not being picked at each extraction.

➢ Thus its probability of ending up in the test data (=not being selected in any of n extractions) is (with $m = n$):

$$\left(1 - \frac{1}{n}\right)^n \approx e^{-1} = 0.368$$

➢ This means the training data will contain approximately 63.2% of the instances.
Example of Bagging

We aim to learn a classifier $C(x)$ in $\mathbb{R}^1$. Assume that the “real” (unknown) classification function $f(x)$ is:

$$f(x) = 0.3x + 1 - 1$$

**Goal:** find a collection of 10 simple thresholding (=linear) classifiers that collectively can classify correctly.

i.e. , each classifier $C_i$ learn a single threshold $T_i$ such that:

$\text{IF } x \leq T_i \text{ then } C$

$\text{else not}(C)$

- I.e. data is not linearly separable, a classifier for these data must learn a decision region, e.g.:

$$\text{IF } T_1 \leq x \leq T_2 \text{ then } C$$

$$\text{else not}(C)$$

➢ In our example, “true” values for decision boundaries are $T_1=0.3$ and $T_2=0.8$. 

Goal: find a collection of 10 simple thresholding (=linear) classifiers that collectively can classify correctly.
Training set

So this is the training set: we have 10 pairs \((x, C(x))\)

<table>
<thead>
<tr>
<th>X</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y=C(x)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We now create 10 samples of this data, and on any sample, we train a “simple” threshold classifier.

**Remember:** Each sample is one training set (bagging round) of size 10. This means that for 10 times (bagging rounds) we sample 10 instances from the original dataset with replacement. The extracted instances in a round\((i)\) are used to train the \(i\)-th learner \(h_i\), and the non-extracted instances are used for testing.
Note: In each round the same training example can be extracted more than one time, and some examples are not extracted.

Note that each classifier is **inconsistent** on training data!
E.g. classifier 1 is wrong on the last two items of “sampled” learning set: $c(0.9) = -1$ (and is instead 1)

For each bagging, we show the threshold learned by each classifier.
Bagging: Aggregation

- In the previous example, given an initial training set of 10 examples, we bag the data 10 times and we learn 10 threshold classifiers $C_i$ ($i=1,\ldots,10$), (our $h_i(x)$) each with an expected error rate $E( h_i(x) - f(x) ) = e_i$

- We then need to combine these results (ensemble method)

- There exists several methods to determine the final score:

1. Voting Classifiers
2. Combining (averaging) classification functions
Bagging: Aggregation
Voting Classifiers

There are two types of voting you can do for classifiers:

1. **Majority voting** (Hard voting): we just need a majority of classifiers to determine what the result could be.

A simple version of Majority voting for **binary classifiers** (with values +1, -1):

\[
\text{IF } \text{sign}(\sum_i C_i(x_j)) = 1, \text{ then } C(x_j) = 1
\]

➢ This means: if majority says “1” then, predicted class is 1.
Bagging with Majority Voting

IF applied to training data

Accuracy of ensemble classifier: 100% 😊

If \( \text{sign}(\sum C_i(x)) = 1 \) then \( C(x) = 1 \)

<table>
<thead>
<tr>
<th>Round</th>
<th>x=0.1</th>
<th>x=0.2</th>
<th>x=0.3</th>
<th>x=0.4</th>
<th>x=0.5</th>
<th>x=0.6</th>
<th>x=0.7</th>
<th>x=0.8</th>
<th>x=0.9</th>
<th>x=1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sum</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>-6</td>
<td>-6</td>
<td>-6</td>
<td>-6</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Sign</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>True Class</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5.36. Example of combining classifiers constructed using the bagging approach.
Bagging: Aggregation

Voting Classifiers

2. **Weighted voting** (Soft voting): A predicted class probability from each learned model $h_i(x)$ for each record is collected and multiplied by each classifier weight, and finally averaged. The final class label is then derived from the class label with the highest average probability.

**Tip:** In reality weights are hard to find using intuition. To counter this subjective process, a *linear optimization equation* or *neural network (perceptron)* could be used to find the correct weighting for each of the models to optimize the accuracy of the ensemble.
Example of averaging over ensembles:
Creating a non-linear classifier out of many linear classifiers (e.g. perceptrons)
By combining all the lines we may obtain a perfect classifier.
So the basic idea is that n simple classifiers work like (or better than) a complex classifier.
Surprisingly competitive performance & rarely overfits

- Main goal is to reduce \textit{variance} of combined models
- Improves ability to ignore irrelevant features

- Works well if all instances \textit{have an equal probability} $p$ \textit{of being classified correctly or wrongly} (means: $\text{Pr}(f(x)\neq h(x))=p$ for all $x$ in $X$)
Bagging:
Notable Benefits

● Does not focus on any particular instance of the training data - **assumption is that all instances have the same probability of misclassification** (often wrong assumption, e.g. image recognition)

● What if we want to focus on a particular instance of training data?
  ➢ E.g. some instance can be more difficult to classify than others (**and on these difficult instances most “simple” classifiers may be wrong, so majority voting won’t work**
Example 1: Handwriting Recognition

Some signs are more difficult than others, even for humans.
Example 2: Face Recognition

Not all faces are recognized here..
Idea: The main idea of boosting is to add models to the overall ensemble **sequentially**. At each iteration, a new model is created and the new base-learner model is trained (updated) from the errors of the previous learners.

As for bagging, the algorithm creates multiple weak models whose output is finally combined to get an overall prediction.

- The main goal of boosting is to reduce the **bias** (errors due to the classifier itself, not to its sensitivity to data variations)
Boosting

**How:** An iterative procedure to adaptively change the distribution of training data by focusing (in each iteration) on previously misclassified examples.

1. Get a dataset
2. Take a bootstrap, and train a model on it
3. See which examples the model got wrong
4. Upweight those ‘hard ‘ examples, downweight the ‘easy’ ones
5. Go back to step 2, but with a **weighted bootstrap**

Each new member of the ensemble «focuses» on the instances that the previous ones got wrong!
Instances $x_i$ are sampled with a probability that depends on their weight $w_i$ ($P(X=x_i)=w_i$). Initially, instances are equally weighted.

In iteration (round) $j$, instances that are wrongly classified by current learner will have their weights increased (so that their probability of being sampled in next boosting round will grow).

Those that are classified correctly will have their weights decreased.
Boosting Example

- Suppose example 4 is hard to classify (round 1 classifier is wrong on example 4)
- Its weight is increased, therefore it is more likely to be extracted again in subsequent weighted sampling rounds (2)
- If round 2 classifier is again wrong on example 4, its probability of being extracted increases again

<table>
<thead>
<tr>
<th>Original Data</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boosting (Round 1)</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Boosting (Round 2)</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Boosting (Round 3)</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
The key idea is that, since the learning set (in step i) includes more examples of the “complex” cases, the current learner $h_i$ is trained on those cases. E.g., recognizing faces of people with a hat, or with dark glasses.
At first, equal weights are assigned to each out of $n$ training instances (1/n for round 1), so all instances have the same probability of being sampled.

After a model $h_i$ is learned, the instance weights are adjusted to allow the subsequent model $h_{i+1}$ to “pay more attention” to data that were misclassified by $h_i$. Higher weights = higher probability for an instance of being extracted.

Final boosted model $h^*$ combines the votes of each individual learner.

- Weight of each model’s vote is a function of its accuracy.

Adaboost – most popular boosting algorithm for classifiers.
Adaboost
Adaptive Boost

● Input:
  ➢ Training set D containing \( n \) instances
  ➢ A classification algorithm (e.g., NN or DTtree)
  ➢ \( T \) iterations (i.e. rounds) (\( i=1, \ldots, T \)). A classifier \( C_i \) is learned at each round.

● Output:
  ➢ A composite classifier \( C^* \)
Adaboost: Training Phase

- Training data \( D \) contains \( n \) labeled data
  - \((x_1,y_1), (x_2,y_2), (x_3,y_3), \ldots, (x_n,y_n)\)
  - where \( y_i \) are the correct classifications
- Initially assign equal weight \( 1/n \) to each example
- To generate \( T \) “base” classifiers, we need \( T \) rounds
- Round \( i \):
  - instances from \( D \) are sampled with replacement, to generate the dataset \( D_i \) (\(|D_i| = n|\))
- Each instance’s chance of being selected in the next rounds depends on its weight
  - Each time the new sample is generated directly from the training data \( D \) with different sampling probability according to the weights;
AdaBoost: Testing Phase

- Testing occurs on individual classifiers $C_i$ at the end of each round.
- The performance of each classifier on training set is used to assess the “importance” or authority of $C_i$.
- **Final testing is performed on unseen data (i.e. a validation set).** To combine individual classifications by each $C_i$, the decision of each classifier is taken into consideration proportionally to its importance.
AdaBoost:

**Testing Phase**

Training phase of $C_i$ depends on previous testing phase on $C_{i-1}$

- Base classifier $C_i$, is learned from training data of set $D_i$
- Error of $C_i$ is tested using $D_i$ (test set = training set)
- Weights of training data are **adjusted** depending on how they were classified
  - Correctly classified: Decrease weight
  - Incorrectly classified: Increase weight
- Weight of a data instance indicates how hard it is to classify it
AdaBoost:

Testing Phases for individual classifier

- “Base” learned classifiers: $C_1, C_2, ..., C_T$

- Loss Function (Error rate of $C_i$ on sample $D_i$):

  $$\text{error}(C_i) = \varepsilon_i = \sum_{j=1}^{n} w_j \delta(C_i(x_j) \neq y_j)$$

  - $i =$ index of boosting round
  - $j =$ index of instance in training data
  - If $C(x_j) = y_j$ then prediction is correct
  - $w_j =$ current weight of $x_j$ ($1/n$ in round 1)

- Importance of a classifier: weight of a classifier $C_i$’s vote is:

  $$\alpha_i = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$
AdaBoost:  
Weight updating rule (before (i+1)\(^{th}\) round)

- Weight updating rule **on all training data in** \(D\):

\[
\begin{align*}
\omega_j^{(i+1)} &= \frac{w_j^{(i)}}{Z_i} \times \\
&= \begin{cases} 
\exp^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\
\exp^{\alpha_i} & \text{if } C_i(x_j) \neq y_j 
\end{cases}
\end{align*}
\]

where \(Z_i\) is a normalization factor

\(\alpha_i\) is the “importance” of classifier \(C_i\), as previously computed \(Z_i\) is to obtain that weights \(w_j\) are sampling probabilities for \(x_j\)

If classification of \(x_j\) is correct, decrease weight (divide by \(\exp^{\alpha_i}\)) else increase (multiply by \(\exp^{\alpha_i}\))
The lower a classifier’s $C_i$ error estimate $\varepsilon_i$ is, the more accurate it is, and therefore, the higher its importance $\alpha_i$ when final voting is performed.

Final Testing (on unseen data, test set):

- For each class value $y_j$, sum the weights of each classifier that assigned class $y_j$ to the instance $x_{test}$.
- The class with the highest sum is the WINNER!

$$y = C^* (x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta(C_i(x_{test}) = y)$$

$\delta(x) = 1$ if $C(x)=y$. For any possible classification value $y$ of $x_{test}$, the weighted sum of classifiers with output $y$ is computed. The $y$ value that maximised this sum is taken as the predicted value.
Given $D: \langle x_i, y_i \rangle$, $|D| = n$

1. **Set** weights $w_j = 1/n$

2. **For** $i = 1 \ldots T$
   
   a. **Bootstrap** $D_i$ from $D$ using $P(X = x_j) = w_j$, and train $C_i$
   
   b. **Test** $C_i$ and **compute error** rate on $D_i$, $\varepsilon_i$

3. **IF** $\varepsilon_i > 1/2$ then $T = t - 1$ **abort** loop
   
   a. **Compute** $\alpha_i$
   
   b. **Update** $w_j$

4. **Output:** for any unseen $x_{test}$

   $$ C^*(x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta(C_i(x_{test}) = y) $$
Illustrating AdaBoost

Initial weights for each data point

Original Data

Boosting Round 1

C1 is wrong on these 2 examples, hence their weight is increased

$\alpha = 1.9459$
AdaBoost

Boosting Round 1

Boosting Round 2

Boosting Round 3

Overall

\[ \alpha = 1.9459 \]

\[ \alpha = 2.9323 \]

\[ \alpha = 3.8744 \]
AdaBoost: Example Round 1

$D_1$
AdaBoost:
Example Round 2

\[ h_1 \]

\[ \varepsilon_1 = 0.30 \]
\[ \alpha_1 = 0.42 \]

\[ D_2 \]
AdaBoost: Example Round 3

$\epsilon_2 = 0.21$

$\alpha_2 = 0.65$

$h_2$
AdaBoost: Example

- The 3 learned classifiers
AdaBoost: The Final Hypothesis

\[ H_{\text{final}} = \text{sign}(0.42 + 0.65 + 0.92) \]
What are the base classifiers?

Usually, «base» classifiers for ensemble methods are:

➢ Linear classifiers
➢ Decision Trees

A popular ensemble of Decision Trees is the Random Forest.
Random Forests

Ensemble method designed for decision/regression tree classifiers:

- Combines predictions made by many unpruned d-trees.
- Each tree is generated based on a bootstrap sample of training data and a vector of randomly chosen attributes (i.e. random vector)
- The random vectors are generated from a fixed probability distribution.
- Final classification is chosen with a voting method like Majority Voting (Forest chooses the classification result having the majority of votes over all the trees in the forest)
Random Forests

Introduce two sources of randomness:

- **Bagging method**: each tree is grown using a bootstrap sample of training data (as in Bagging and Boosting)

- **Random vector method**: At each decision node, the best attribute to test is chosen from a random sample of $m$ attributes, rather than from all attributes
Random Forests: Random Vector Method

The random vector method add the following rule to the d-tree training algorithm.

For each node of the tree:

a. Choose $m$ features randomly on which to base the decision at that node.

b. Calculate the best split based on these $m$ variables in the training set (e.g., test on best attribute in $m$ based on Infogain).

Why random vectors? if one or a few features are very strong predictors for the output class (or value for regressors), these features will be selected in many of the trees, causing them to become correlated. To avoid this, a set of features from which to find the best split are selected randomly.
Random Forests: Algorithm

1. Divide training examples into multiple training sets using Bootstrap (i.e. choosing \( n \) times with replacement from all \( N \) available training cases)

2. For each training set:
   a. Train a decision tree with the random vector method
   b. Estimate the error of the decision tree using the rest of the examples.

3. Aggregate the predictions of each tree to make classification decisions using a voting methods (usually Majority Voting)
Random Forests: Visual Explanation
Why use Random Vectors?

- Because in an ensemble, we want independent base learner.
- Averaging over trees with different training samples reduces the dependence of the predictions on a particular training sample.
- Using random vectors of features further reduce dependency. Typically, for a classification problem with $p$ features, $\sqrt{p}$ (rounded down) features are used in each split.
- Increasing the number of trees does not increase the risk of overfitting the data.
- In fact the main advantage of RF is that they do not to overfit.
Advantage of Random Forest

- Since each tree only handles a subset of features, this can be considered a good choice when instances are described by very many features.
- It is also considered a good “dimensionality reduction” method.
Random Forest VS other ensambles or non-ensables

- Empirically comparable classification accuracies to AdaBoost.
- More robust to outliers and noise than AdaBoost.
- Runs much faster than AdaBoost.
- Compares well also with deep methods.
Suggested Lectures

- General Overview: [LINK]
- Boosting and Bagging: [LINK], [LINK]
- Random Trees and Extra Trees: [LINK]

And the heterogeneous ensemble?

- Stacking and Bleeding: [LINK]