DEEP LEARNING (CONT'D)

Supervised Learning

- Have labeled examples of the correct behavior to train the model
- For example: handwritten digit classification with the MNIST dataset
  - Task: given an image of a handwritten digit, predict the digit class
  - Input: the image; Output: the digit class
- Dataset: 70,000 image of handwritten digits labeled by humans
  - Training set: first 60,000 images
  - Testing: last 10,000 images
  - Neural nets already achieved >99% accuracy in the 1990s – still we continue to learn a lot from it!

MNIST dataset

Unsupervised Learning

- In generative modeling, we want to learn a distribution over some dataset, such as natural images
- We can evaluate a generative model by sampling from the model and seeing if it looks like the data

Generative Models

- Unsupervised Learning: only use the input x for learning
  - Automatically extract meaningful features for your data
  - Leverage the availability of unlabeled data
  - Add a data-dependent regularizer to training

- Many neural network based unsupervised learning approaches exist:
  - Autoregressive Models
  - Autoencoders (Variational Autoencoders)
  - Generative Adversarial Networks
  - Flow Models
  - Diffusion Models

Bayesian Network is also a generative model!

Generative Modeling

Application

- Data generation

- Data Imputation, denoising, in-painting

Generative modeling

- Text: The models like BERT, GPT-3 perform unsupervised learning by reconstructing the next words in a sentence. The GPT-3 models learns from 499 Billion Tokens and has **175 Billion parameters**.

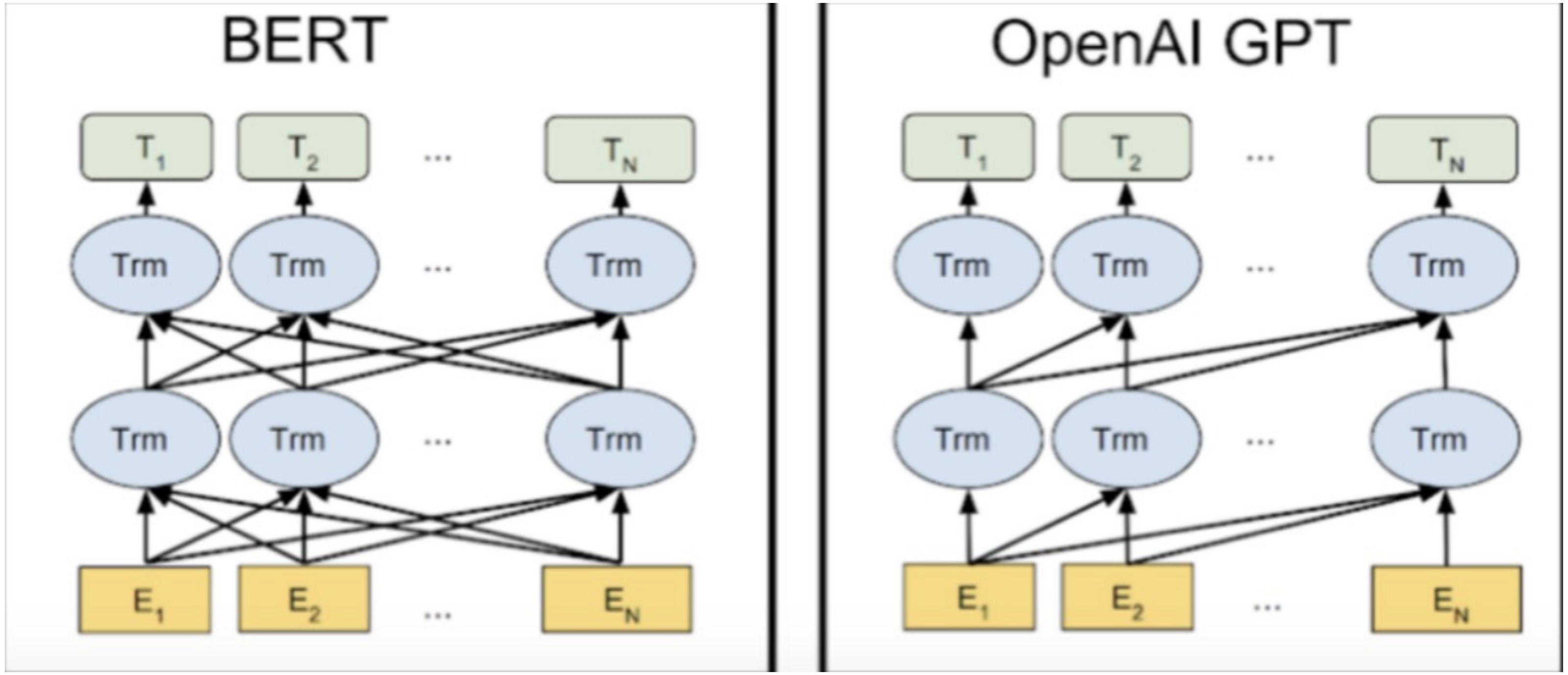
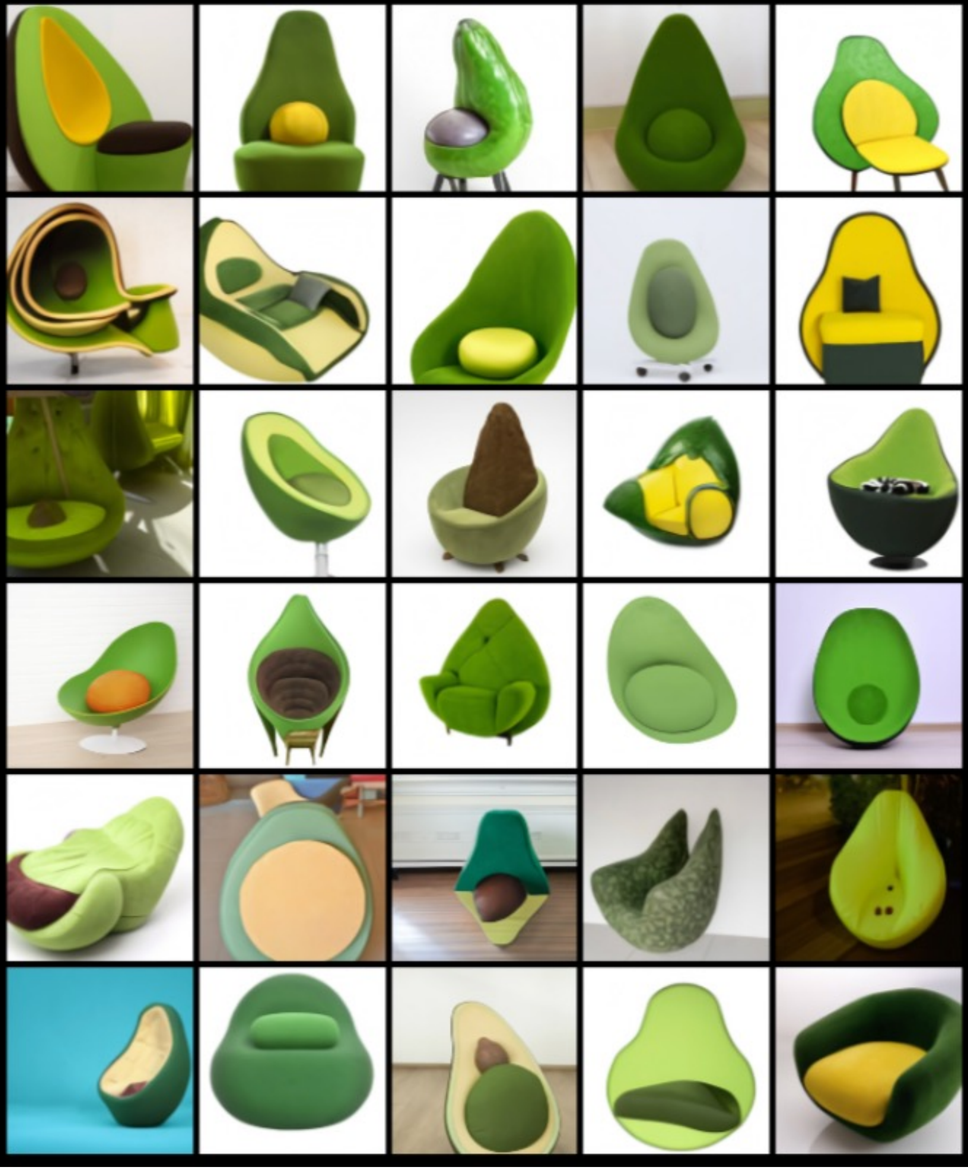


Image Generation Given Text

TEXT PROMPT

an armchair in the shape of an avocado. an armchair imitating an avocado.

AI-GENERATED IMAGES



In the preceding visual, we explored DALL·E's ability to generate fantastical objects by combining two unrelated ideas. Here, we explore its ability to take inspiration from an unrelated idea while respecting the form of the thing being designed, ideally producing an object that appears to be practically functional. We found that prompting DALL·E with the phrases "in the shape of," "in the form of," and "in the style of" gives it the ability to do this.

When generating some of these objects, such as "an armchair in the shape of an avocado", DALL·E appears to relate the shape of a half avocado to the back of the chair, and the pit of the avocado to the cushion. We find that DALL·E is susceptible to the same kinds of mistakes mentioned in the previous visual.

Image Generation Given Text

ENSEMBLE METHODS

Ensemble Method

- Combines multiple base classifiers into one
- Given a test record: output a prediction by taking a vote on predictions of base classifiers

Motivation

- Ensemble method of 25 base classifiers
- Each has error rate ε = 0.35
- What is the error rate of the ensemble?
  - *Identical base classifiers*:

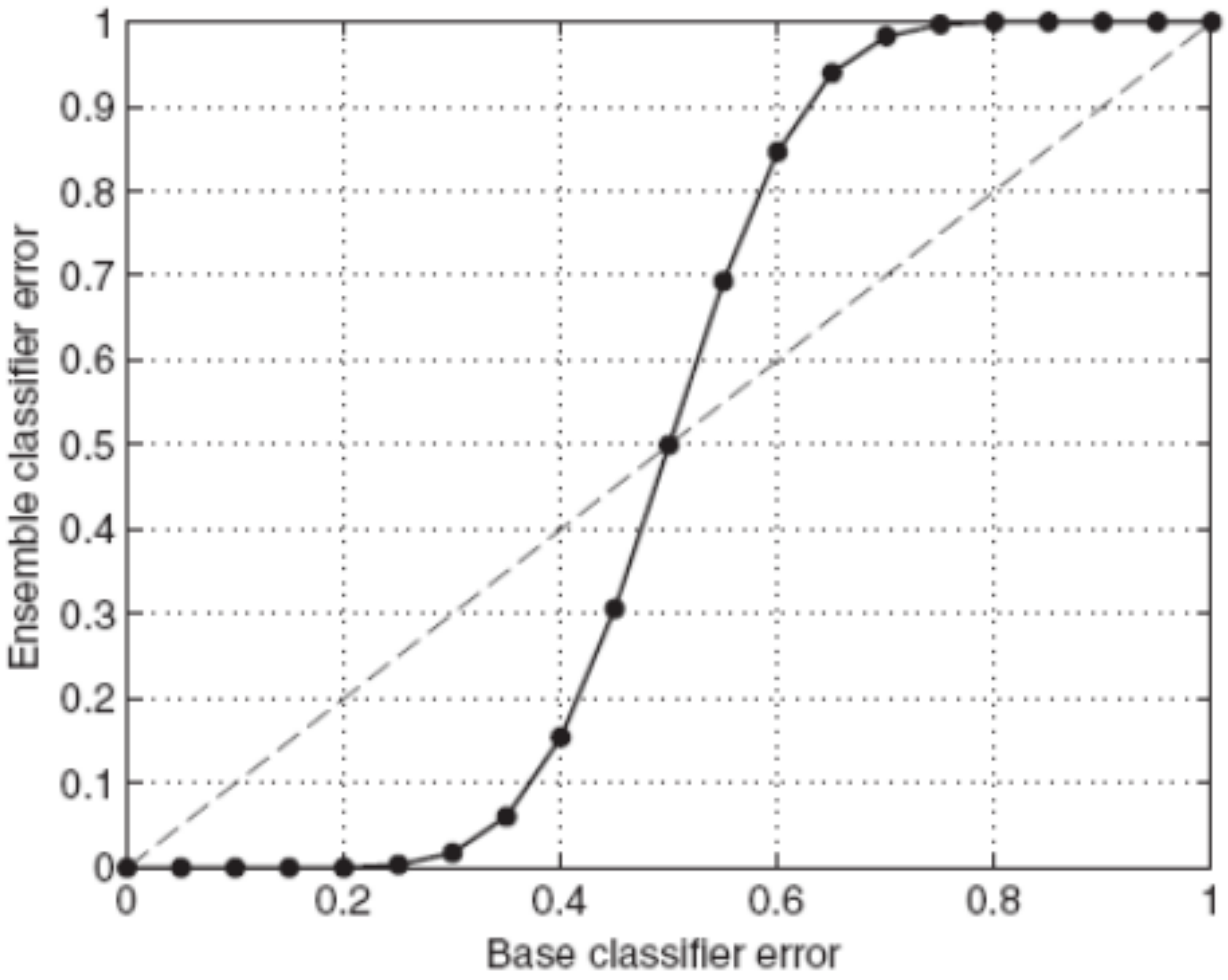
$$e_{ensemble} = 0.35$$

  - *Independent base classifiers*:

the ensemble method will make a wrong prediction only if more than ½ of the classifiers predict incorrectly (suppose 13), in this case the error rate is 0.06 which is lower than the error rate of the base classifier.

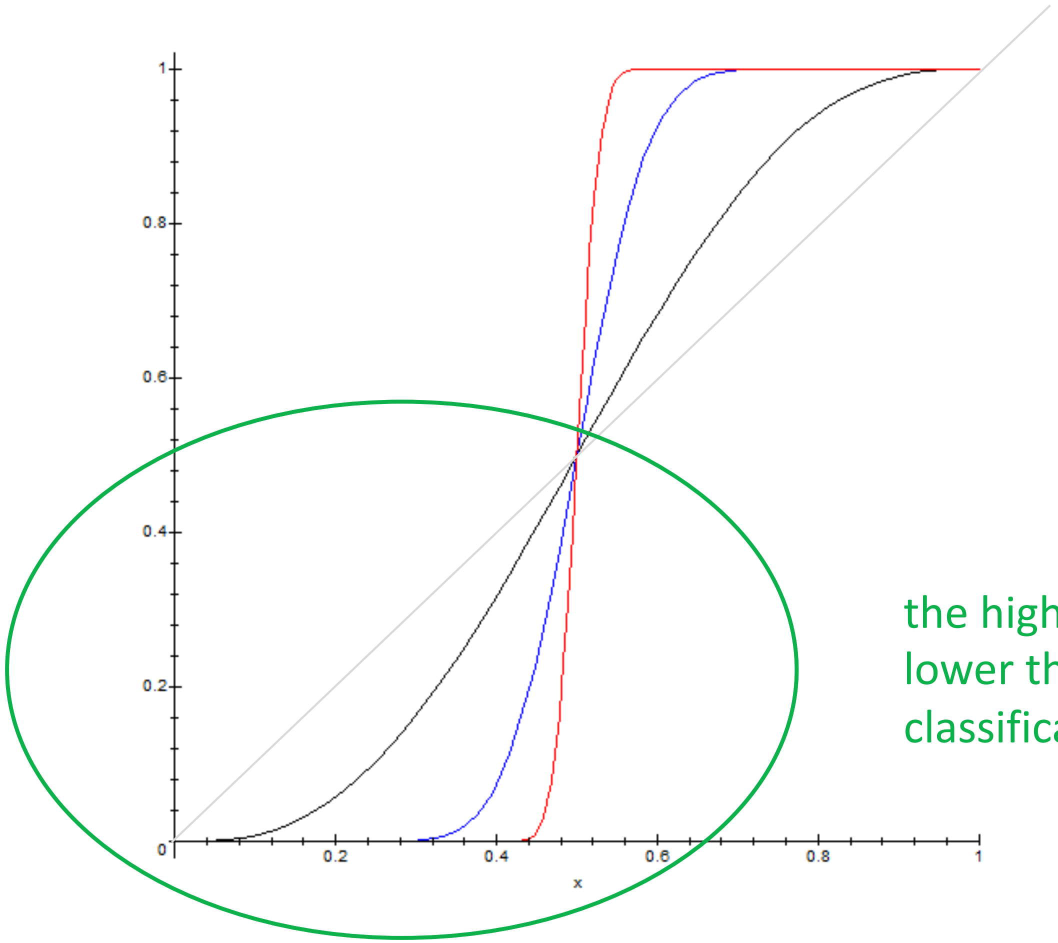
$$e_{ensemble} = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

When is ensemble better?



- *Independent base classifiers*
- $e_i < 0.5$

When is ensemble better?



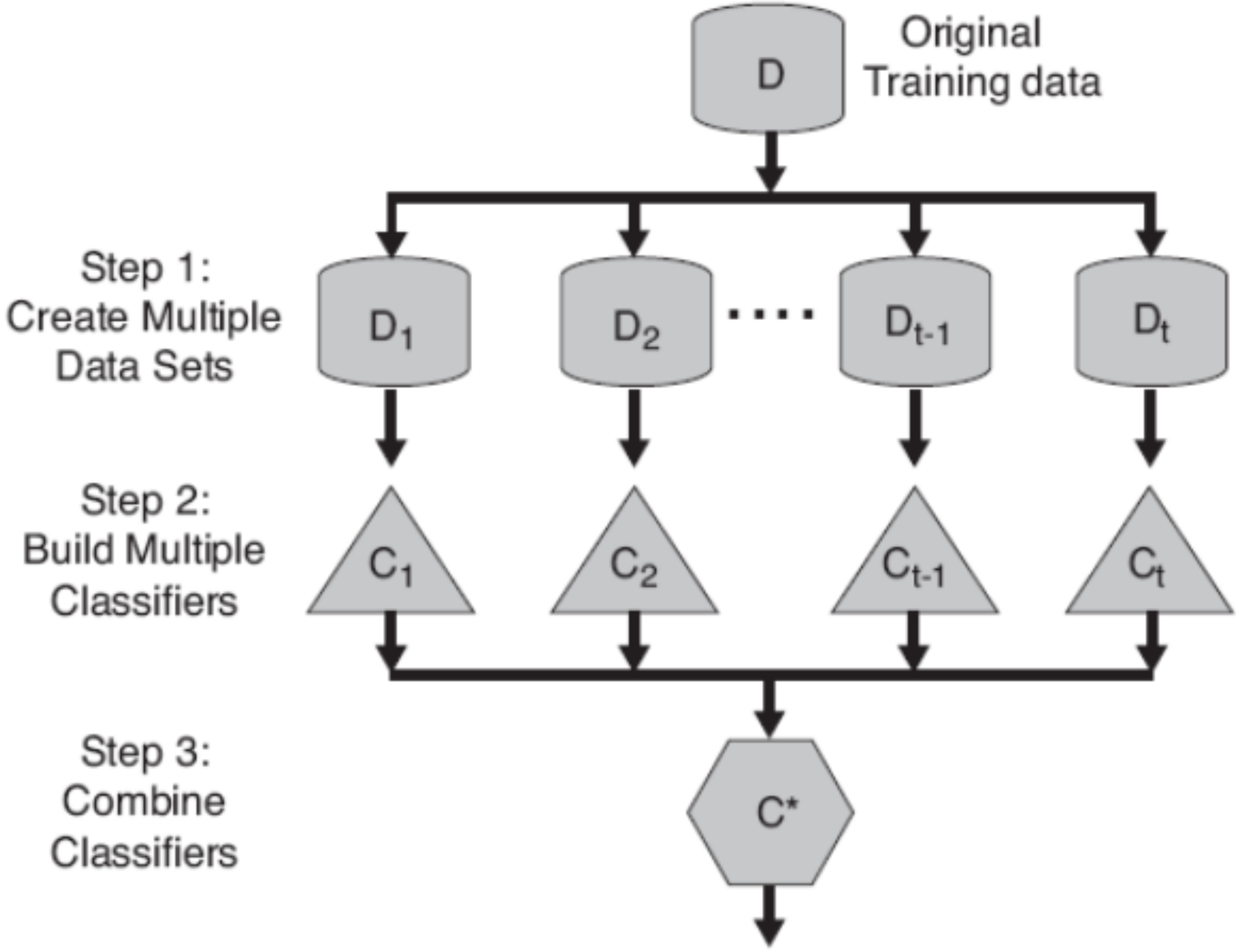
N = 5

N = 51

N = 501

the higher the number of base classifiers, the lower the error rate and the better the classification will be.

General Idea



Methods

- **Manipulate the training set:**
  - **Resampling**

e.g., bagging and boosting

- Manipulate the input features:
  - Use subset of features

The subset can be chosen randomly or based on a recommendation of experts. This approach works well with data sets that contain highly redundant features.

- Manipulate the class labels:
  - When large number of classes, partition into sets
  - Error correcting output coding
- Manipulate the learning algorithm (algorithm specific)
  - Change topology in a neural network
  - Inject randomness into decision tree growing

Algorithm

1: Let D denote the original training data, k the number of base classifiers, T the test data

2: for i = 1 to k do

3: Create training set D_i from D

4: Build a base classifier C_i from D_i

5: end for

6: for each test record x in T do

7: $C^*(x) = Vote(C_1(x), C_2(x), \ldots, C_k(x))$

8: end for

Bagging (Bootstrap Aggregating)

- Repeatedly creates samples with replacement according to uniform distribution
  - each time you randomly select an observation from the original dataset, you put it back into the dataset before the next selection
  - The same observation can be selected multiple times within the same training set D_i.
- Each record: selected with probability $1 – (1-1/N)^N$
- Pick class that receives highest number of votes

Algorithm

1: Let D denote the original training data, k the number of base classifiers, T the test data

2: for i = 1 to k do

3: Create a bootstrap sample D_i from D

4: Build a base classifier C_i from D_i

5: end for

6: for each test record x in T do

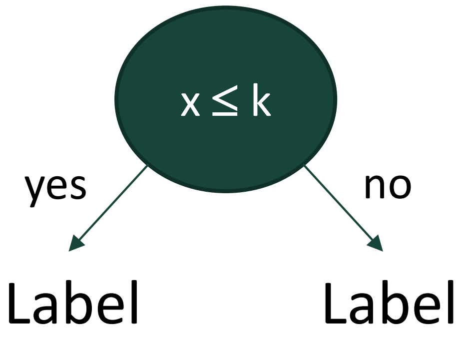
7: $C^*(x) = Vote(C_1(x), C_2(x), \ldots, C_k(x))$

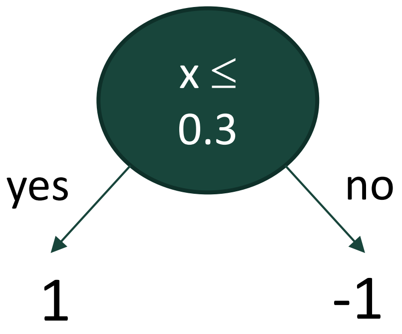
8: end for

Bagging example

X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier: decision tree with one level





- What is the best we can do?

Bagging example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Bagging example

If using training set for testing

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Ensemble Prediction → (Sign row)

Boosting

- Adaptively changes the distribution of training examples
- Focuses on the examples that are hard to classify
    - Assign a weight (for getting selected) for each training example
    - Generate a training set
    - Generate a classifier based on the training set
    - Adjust the weights based on classifier prediction
        - Higher weights for examples incorrectly classified
    - Repeat
- How are weights updated?
- How are predictions combined?

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Initially all examples are assigned the same weights, however some examples may be chosen more than once. For example 3 and 7. Because the sampling is done with replacement.
- Suppose 4 is hard to classify

  Its weight is increased => it is more likely to be chosen again in subsequent rounds

AdaBoost

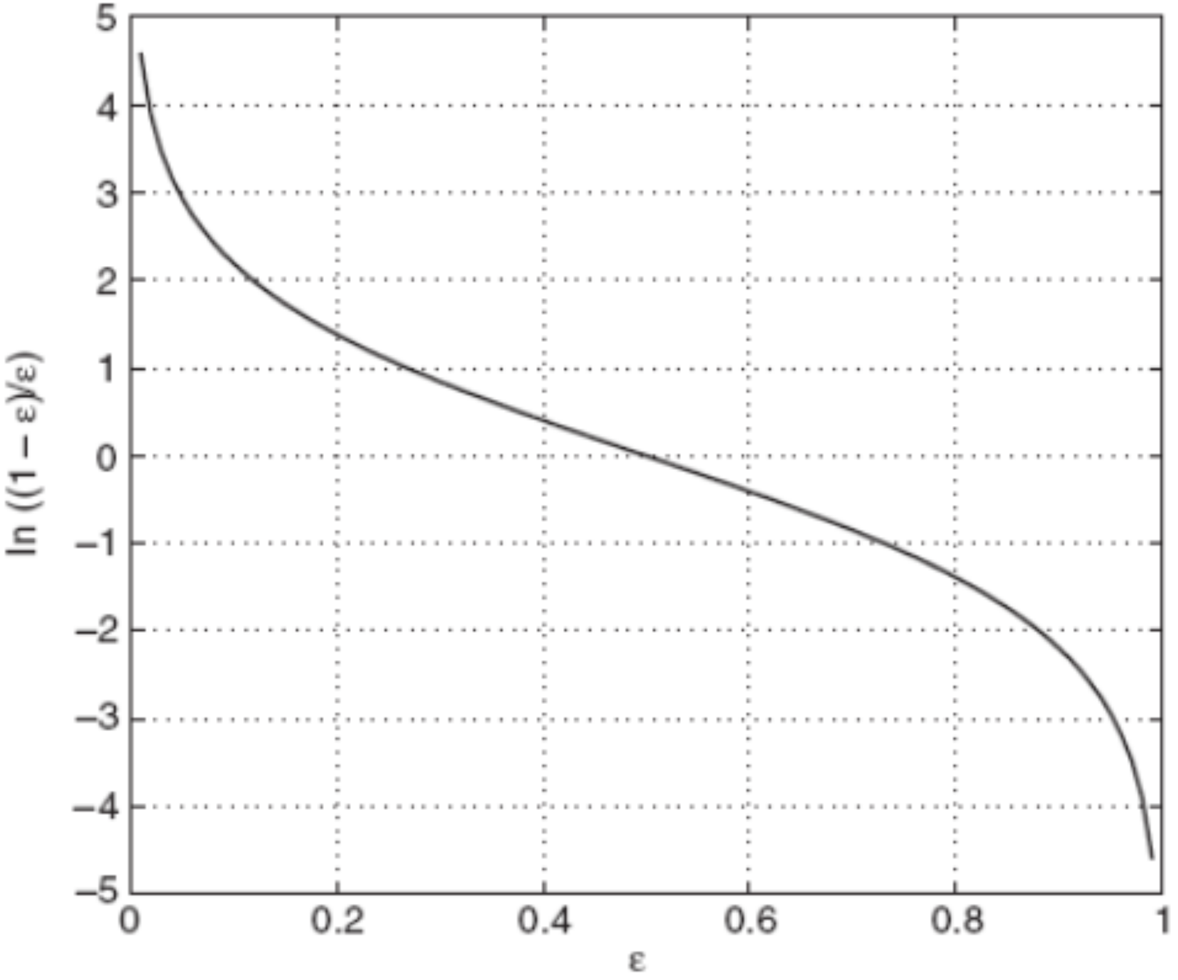
- The importance of a base classifier Ci depends on its error rate:

Error rate $$\varepsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

Importance measure $$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

- Use alpha to update the weights of records
- Use alpha to combine results

AdaBoost



The importance will have a large positive value if the error rate is close to 0 and a large negative value if the error rate is close to 1

Weight Update

- Use alpha to update the weights of records

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} e^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ e^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

make sure that the sum of all weights is equal to 1.

- Weights of correctly classified records decrease
- Weights of incorrectly classified records increase

- Additional step: if $e_i > 50\%$, $w_i = 1/N$

If any intermediate rounds produce an error rate higher than 50%, the weights are reverted back to their original uniform value and the resampling procedure is repeated.

Making prediction

- Each classifier contributes based on its weight

$$C^*(x) = \arg\max_{y} \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

allow the adaboost to penalize models that have poor accuracy

Example

Actual

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

Example

Round	Split Point	Left Class	Right Class	α
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

(a)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

(b)

- boosting technique focus on the training example that are wrongly classified, it can be susceptible to overfitting.
- i.e., higher testing error and poor generalization performance.
- How can we formally analyze the generalization error of a poor predictive model?

Bias – Variance (& noise) Decomposition

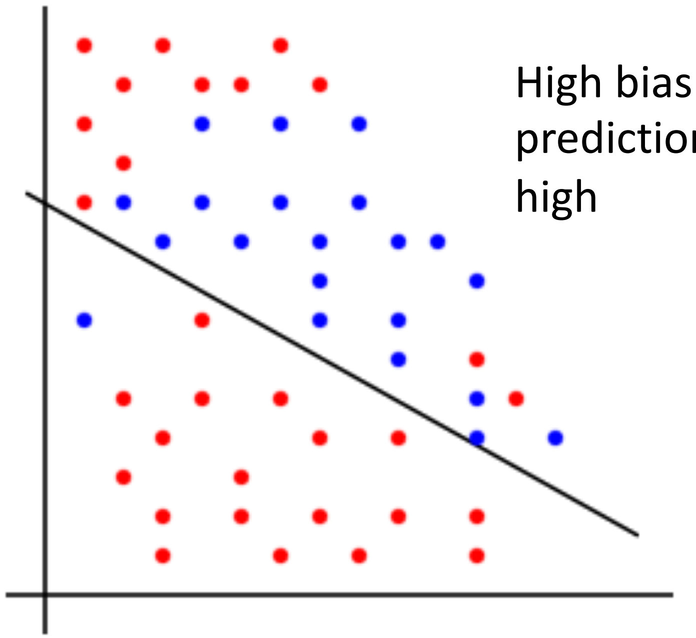
- Classification Error = Bias + Variance + Noise

- Bias:
  - The ability of the model to approximate the data
  - The error of the best classifier
- Variance:
  - Stability of the model in response to new training data
  - Error of the trained classifier with respect to the best classifier

Bias

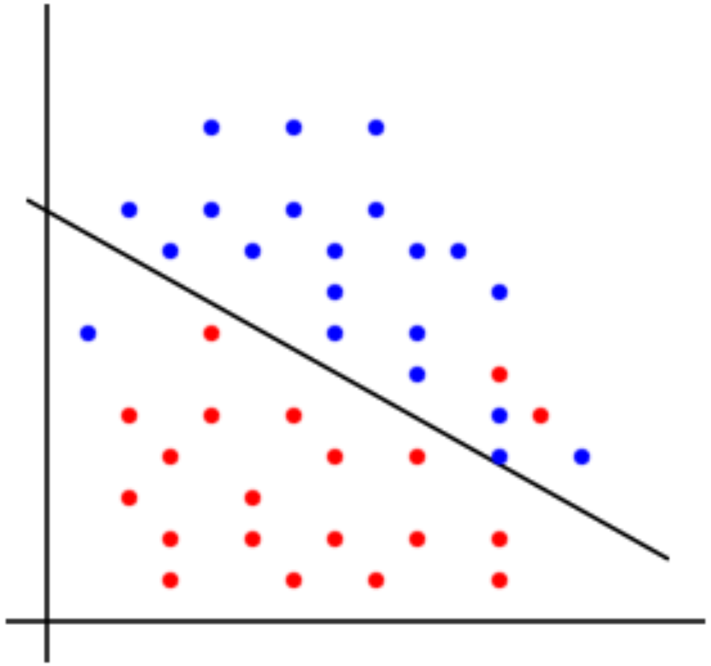
Independent of the training data

If the model is too simple, the solution is biased. It does not fit the data



High bias:
prediction error is
high

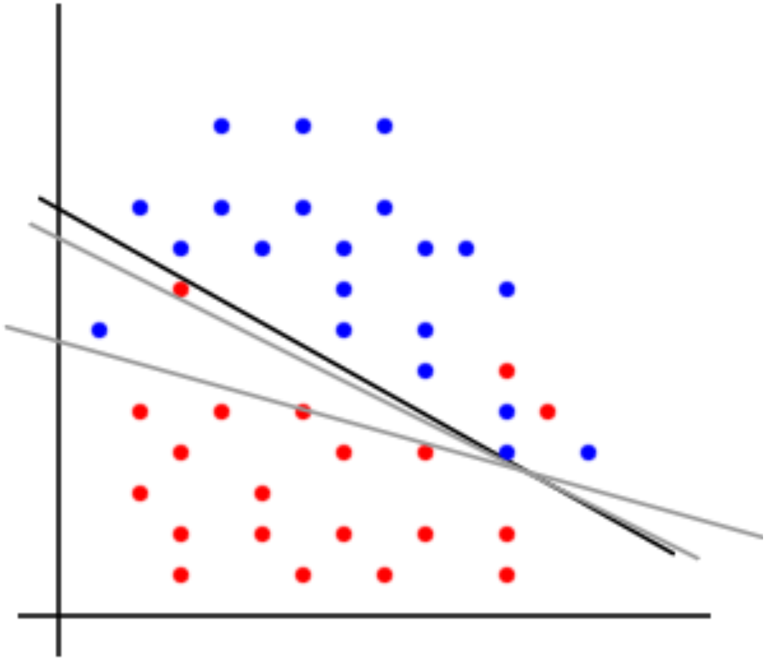
High Bias

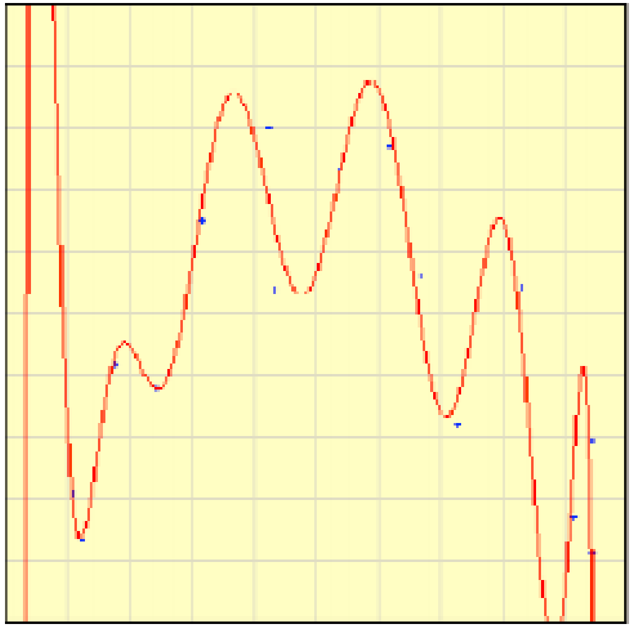


Low Bias

Variance

Depends on the training data, decreases with more data
If the model is too complex, it is very sensitive to small changes in the data





- a model shows better generalization performance if it has a lower bias and *lower variance.*
- If a model show low bias by *high variance*, it is susceptible to overfitting

Bagging vs. Boosting

- Bagging reduces variances by taking average
- Boosting reduces both bias and variances
- Boosting might hurt performance on noisy data. Bagging does not have this problem
- Bagging is easier to parallelize
- In practice, bagging and boosting are powerful techniques

CLASSIFIER COMPARISON

Classification

- Approaches:
  - Decision trees, nearest neighbors, Bayes classifiers, perceptron, artificial neural networks, SVM …
- Characteristics:
  - Prone/robust to noise and overfitting
  - Linear vs. non linear model
  - ***Different training/testing speeds***

Big Oh

- One of the most fundamental tools for computer scientists to analyze the cost of an algorithm
- Measure of how many operations needed to perform a calculation, typically worst case scenario
- We usually interested in the rate of growth which could be:
  - logarithmic: $O(\log n)$. An example, you divide the structure in half over and over again and do a constant number of operations for each split.
  - linear: $O(n)$
  - linearithmic: $O(n \log n)$
  - quadratic: $O(n^2)$
  - exponential: $O(n^c)$
- "worst case scenario": We only care about the biggest "term" here.

Big Oh

- Assume we have N instances, each instance has d features, there are only 2 classes, and they are balanced
  - Training time will be some function of N, denoted as O(N)
  - Focus on the powers over constants.
    - 2N calculations = O(N)
    - Comparing every instance with every other instance is $\mathrm{O}(N \times N) = O(N^2)$

- Notation can also be used for size of model
  - KNN stores all training instances in memory, thus needs O(Nd) memory
  - Naïve Bayes stores $P(x_i|Y = y_i)$ for each feature, x_i, thus needs O(d) memory

Comparison

N – number of instances
d – number of features
k – number of latent features (e.g. nodes in hidden layer)
e – number of epochs (iterations through data)

Model	Train Time	Test Time	Interpretable	Robust to noise	Robust to redundant features	Scalable to large dimensions
Decision Tree	$O(Nd\ log(N))$	$O(w)$ W = max depth	Yes	Yes	Yes	No
Nearest Neighbor	$O(1)$	$O(Nd)$	No	No	No	Yes
Naïve Bayes	$O(Nd)$	$O(d)$	Yes	Yes	No	Yes
Multilayer Neural Network	$O(Ndke)$	$O(dk)$	No	No	Yes	No
SVM	$O(N^2)$ or $O(N^3)$	$O(d)$	No	Yes* (soft margin)	No	Yes
Bagging	Classifier dependent		Classifier dependent	Yes	Classifier dependent	
AdaBoost	Classifier dependent		Classifier dependent	No	Classifier dependent	

sklearn implementation

Model	sklearn function
Decision Tree	sklearn.tree.DecisionTreeClassifier
Nearest Neighbor	sklearn.neighbors.KNeighborsClassifier
Naïve Bayes	sklearn.naive_bayes.GaussianNB
Multilayer Neural Network	sklearn.neural_network.MLPClassifier
SVM	sklearn.svm.SVC
Bagging	sklearn.ensemble.BaggingClassifier
AdaBoost	sklearn.ensemble.AdaBoostClassifier