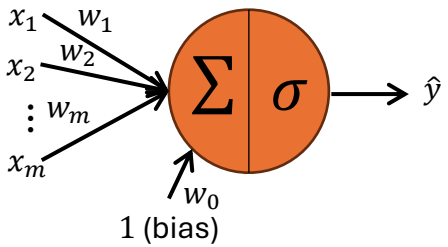
Data Mining

Classification – Alternative Techniques

Topics

- Rule-Based Classifier
- Nearest Neighbor Classifier
- Naive Bayes Classifier
- Logistic Regression
- **Artificial Neural Networks**
- Support Vector Machines
- Ensemble Methods

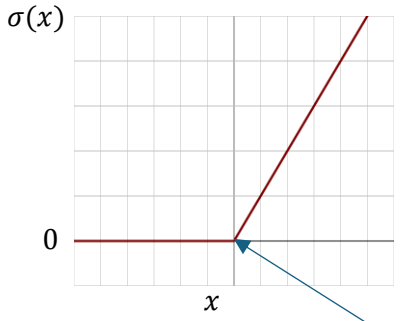
The Artificial Neuron



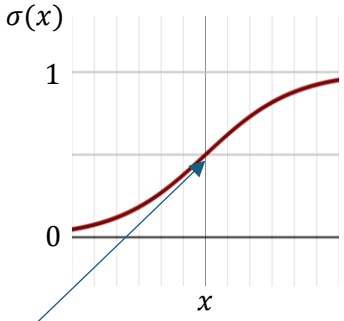
$$\hat{y} = \sigma\left(w_0 + \sum_{i=1}^{m} x_i w_i\right) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

- Input vector: $\boldsymbol{x} = (1, x_1, x_2, \ldots, x_m)$, where the first element is for the bias.
- Weight vector: $\boldsymbol{w} = (w_o, w_1, w_2, \ldots, w_m)$
- Activation function: $\sigma(\cdot)$ is a nonlinear function that transforms the weighted sum of inputs into an output value called the activation of the neuron. Typical activation functions are:

Rectified linear unit (ReLU)



Sigmoid (logistic function)

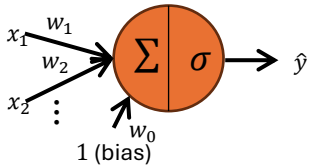


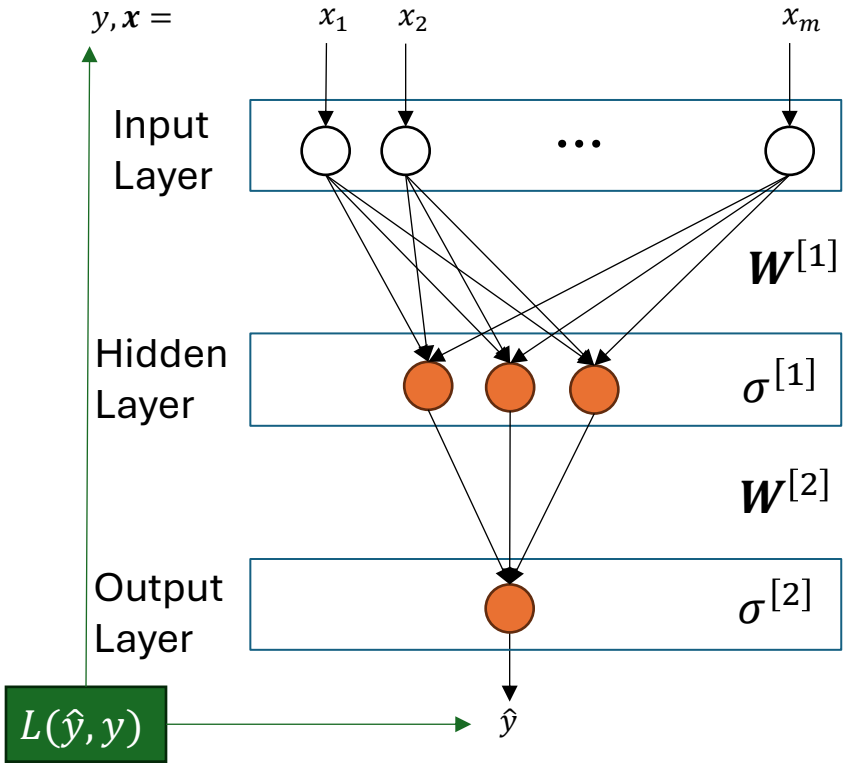
Bias shifts this point

Training: Find $\boldsymbol{w}$ that minimizes the loss $L(\hat{y}, y)$ using gradient descent.

Relationship to Logistic regression to predict a binary outcome probability: Log. Regression is a single artificial neuron with a logistic activation function and trained using binary cross-entropy loss (log loss).

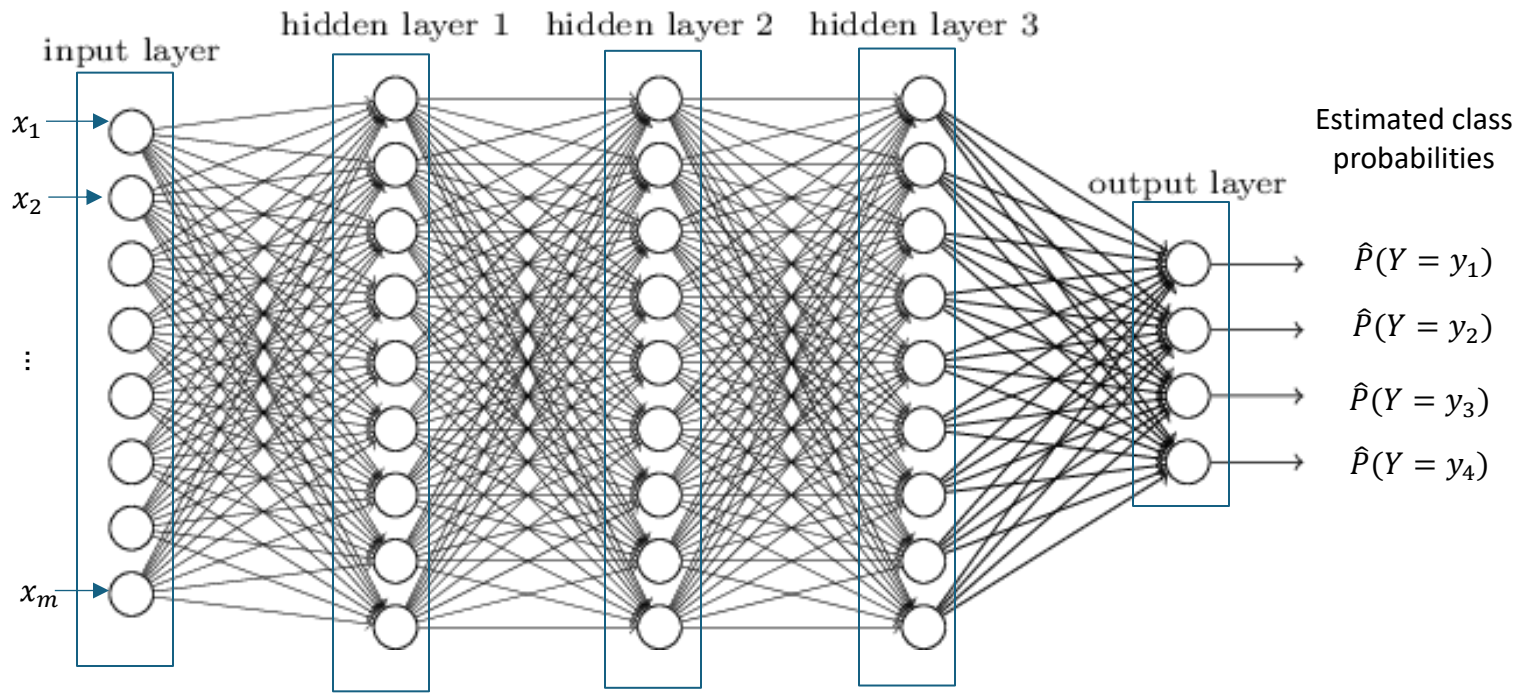
General Structure of an ANN





- Collect the weights of all neurons in a layer into a matrix:
$$\boldsymbol{W}^{[l]} = \left[\boldsymbol{w}_1^{[l]}, \boldsymbol{w}_2^{[l]}, \ldots\right]$$
- $\hat{y} = \sigma^{[2]}\left(\boldsymbol{W}^{[2]}\sigma^{[1]}\left(\boldsymbol{W}^{[1]}\boldsymbol{x}\right)\right)$
- **Training**: learn weights that minimize $L(\hat{y}, y)$ using gradient descent with backpropagation.
- ANNs with a single hidden layer are **universal approximators**, i.e., can approximate any function $y = f(x)$ with no error!

Deep Learning / Deep Neural Networks

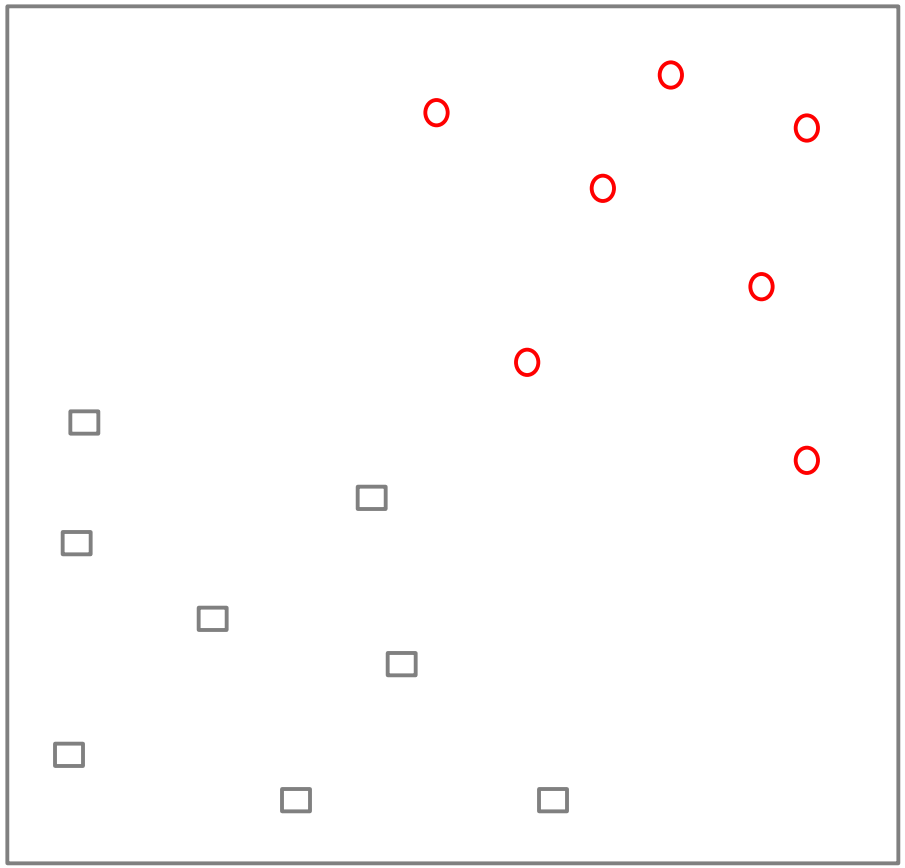


- **End-to-end machine learning**: The hidden layers can provide automatic feature engineering for the output layer.
- **Transfer learning**: Pretrained layers may be reused.
- **Issues**: Deciding on the network topology. Many hyperparameters. Needs lots of data + computation (GPU).
- **Applications**: computer vision, speech recognition, natural language processing, audio recognition, machine translation, bioinformatics, …
- **Tools**: Keras, Tensorflow, and many others.
- **Related**: Deep belief networks, recurrent neural networks (RNN), convolutional neural network (CNN), …

Topics

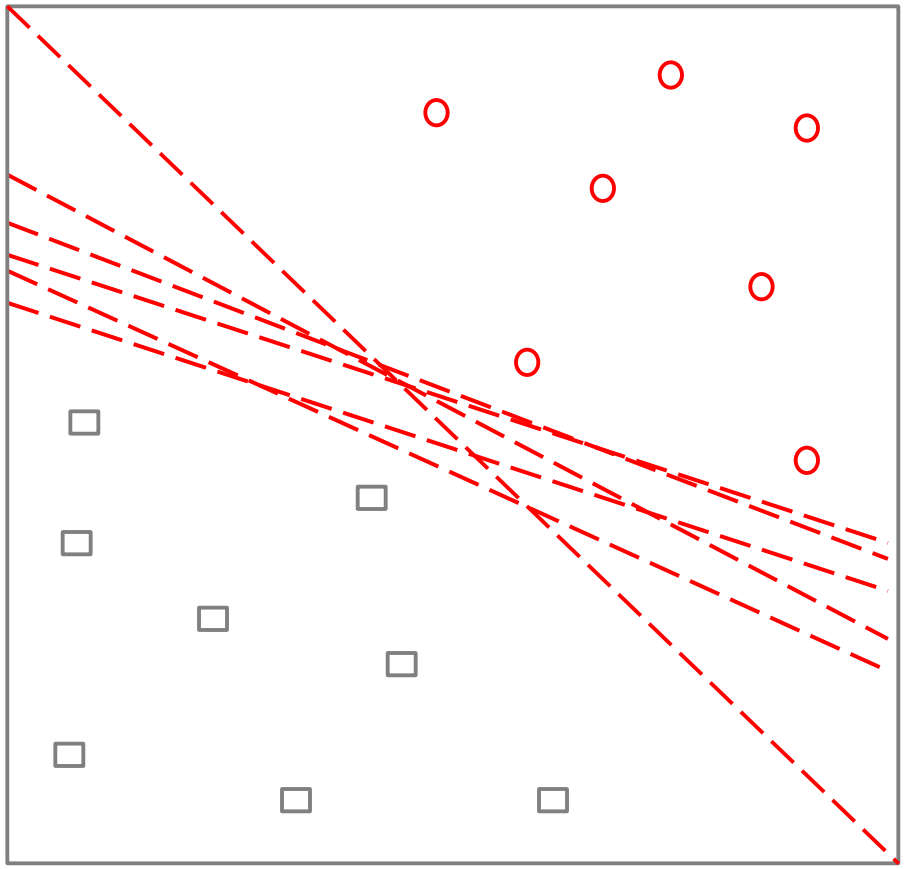
- Rule-Based Classifier
- Nearest Neighbor Classifier
- Naive Bayes Classifier
- Logistic Regression
- Artificial Neural Networks
- **Support Vector Machines**
- Ensemble Methods

Support Vector Machines



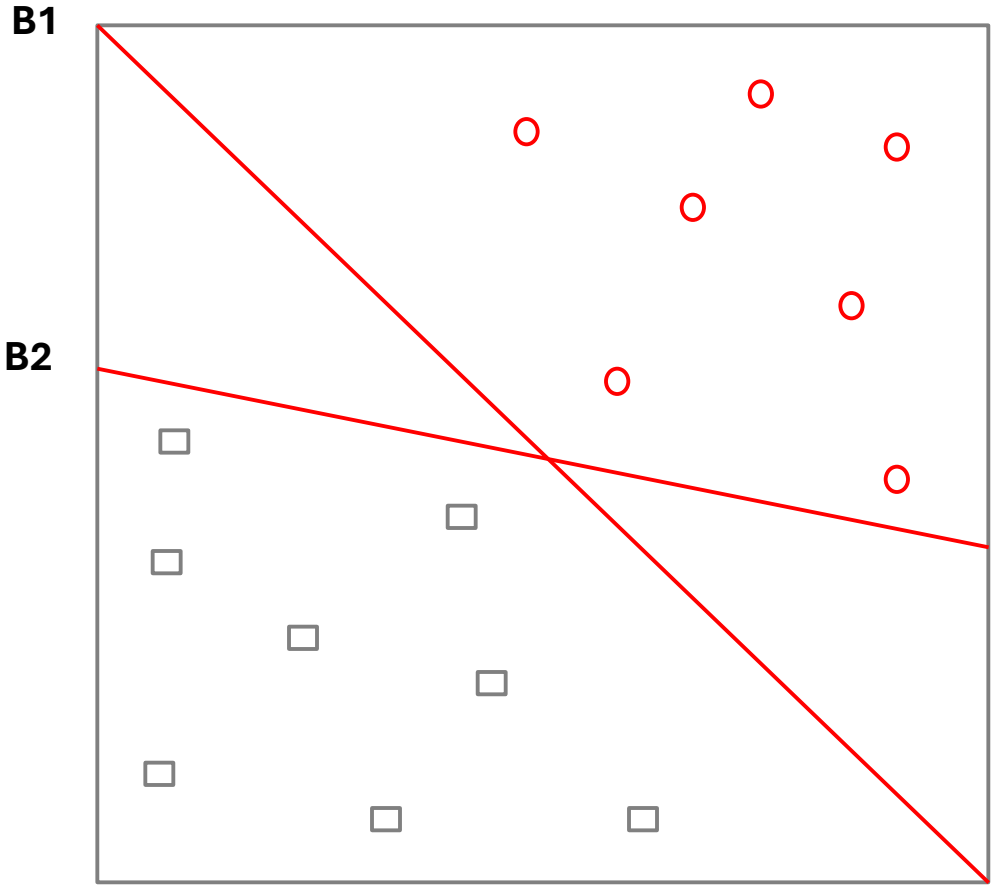
Goal: Find a linear hyperplane (decision boundary) that will separate the data.

Support Vector Machines



Many possible solutions

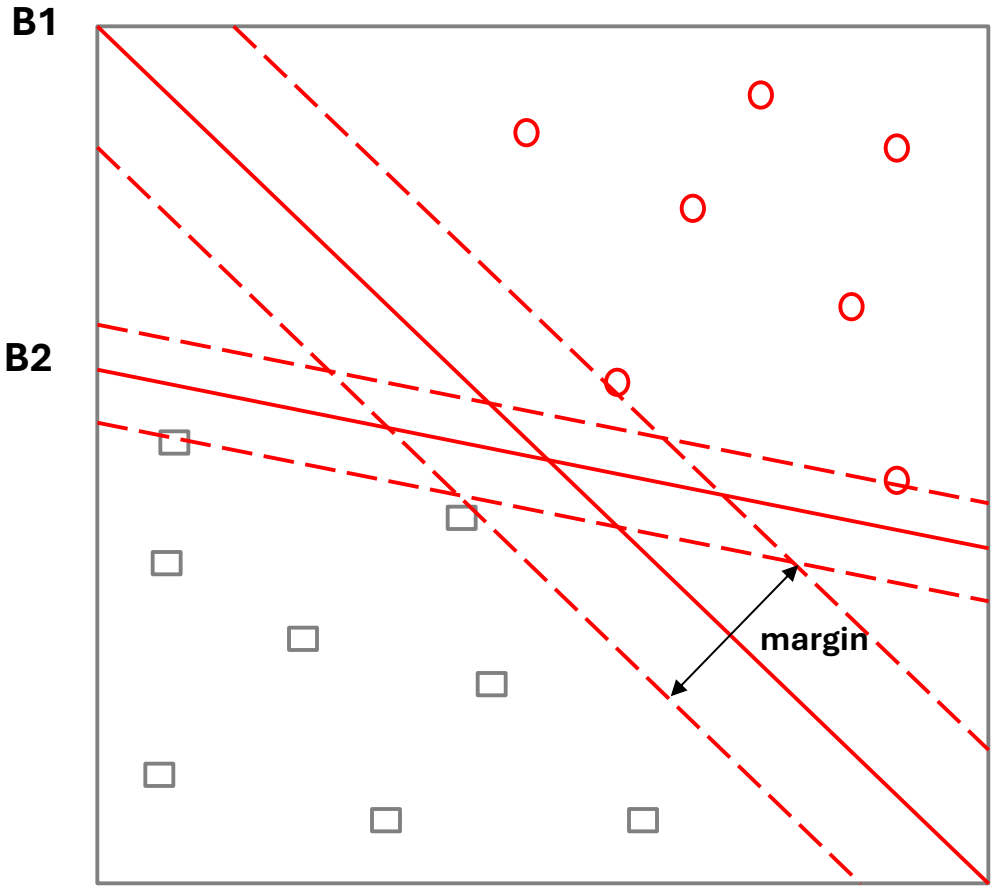
Support Vector Machines



Which one is better? B1 or B2?

How do you define better?

Support Vector Machines



Find the hyperplane with the maximal margin => B1 is better than B2

Larger margin = more robust = less expected generalization error

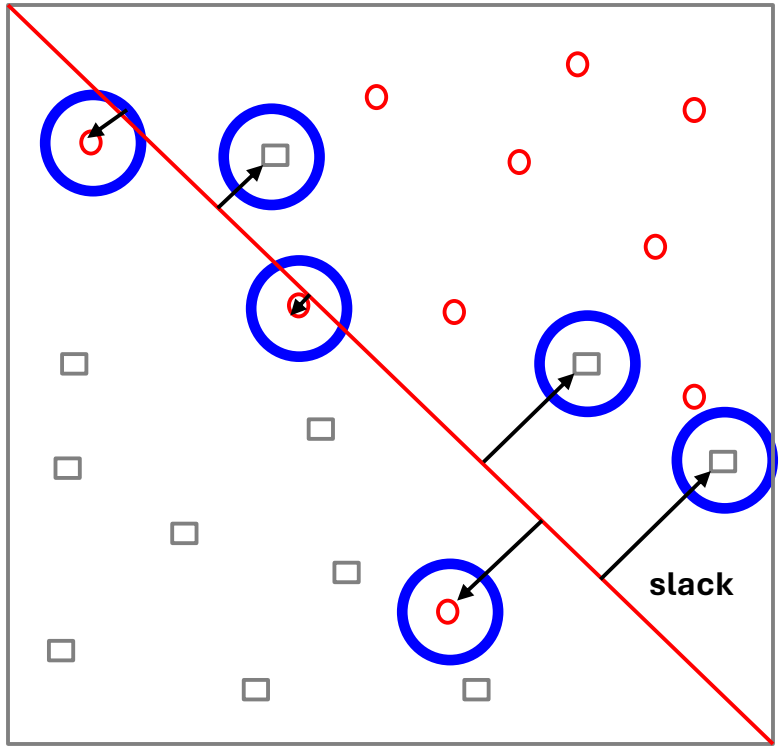
Support Vector Machines

What if the problem is not linearly separable?

- Use slack variables to account for violations.
- Use hyperplane that minimizes the total slack.

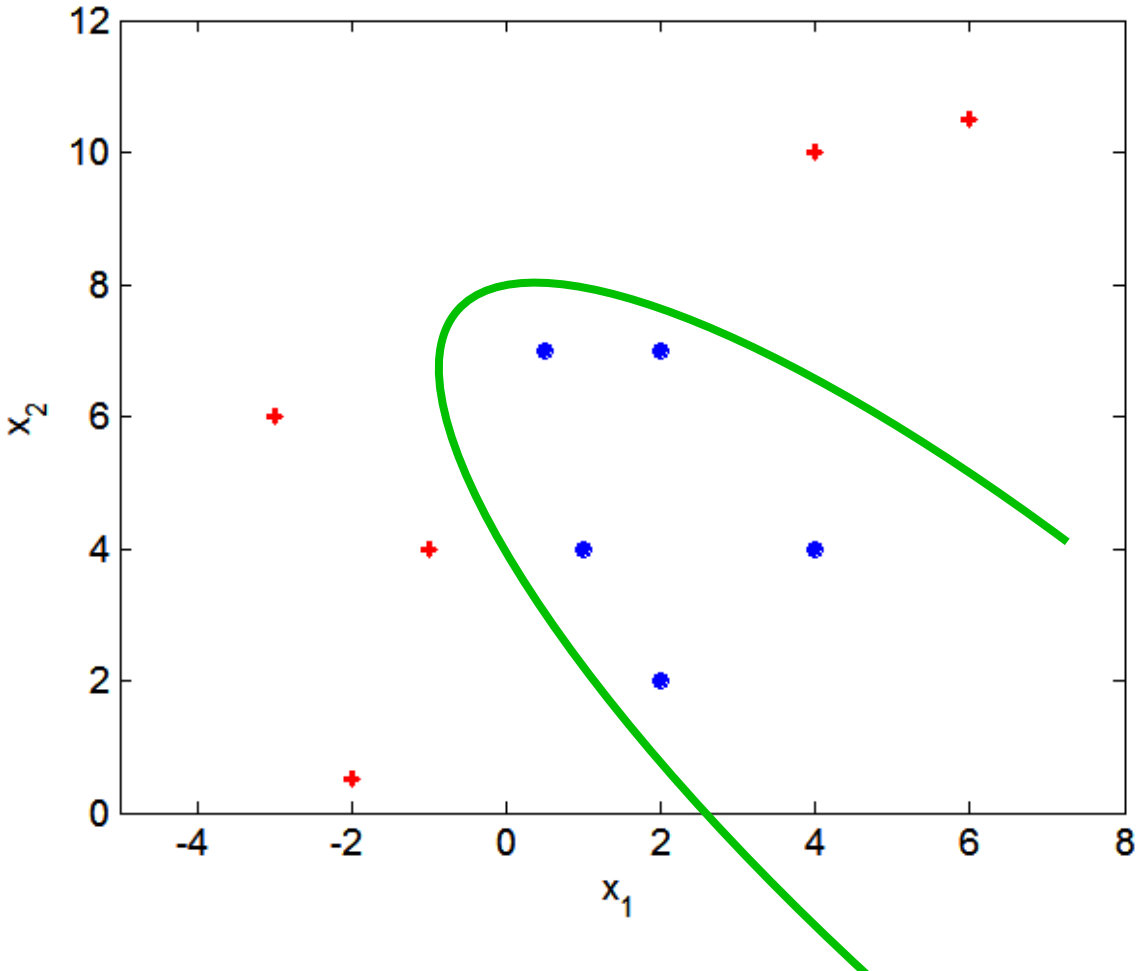
Solution:

- The optimization problem can be written as a **quadratic optimization problem** with linear constraints that **only depends on a few close data points** called the support vectors.

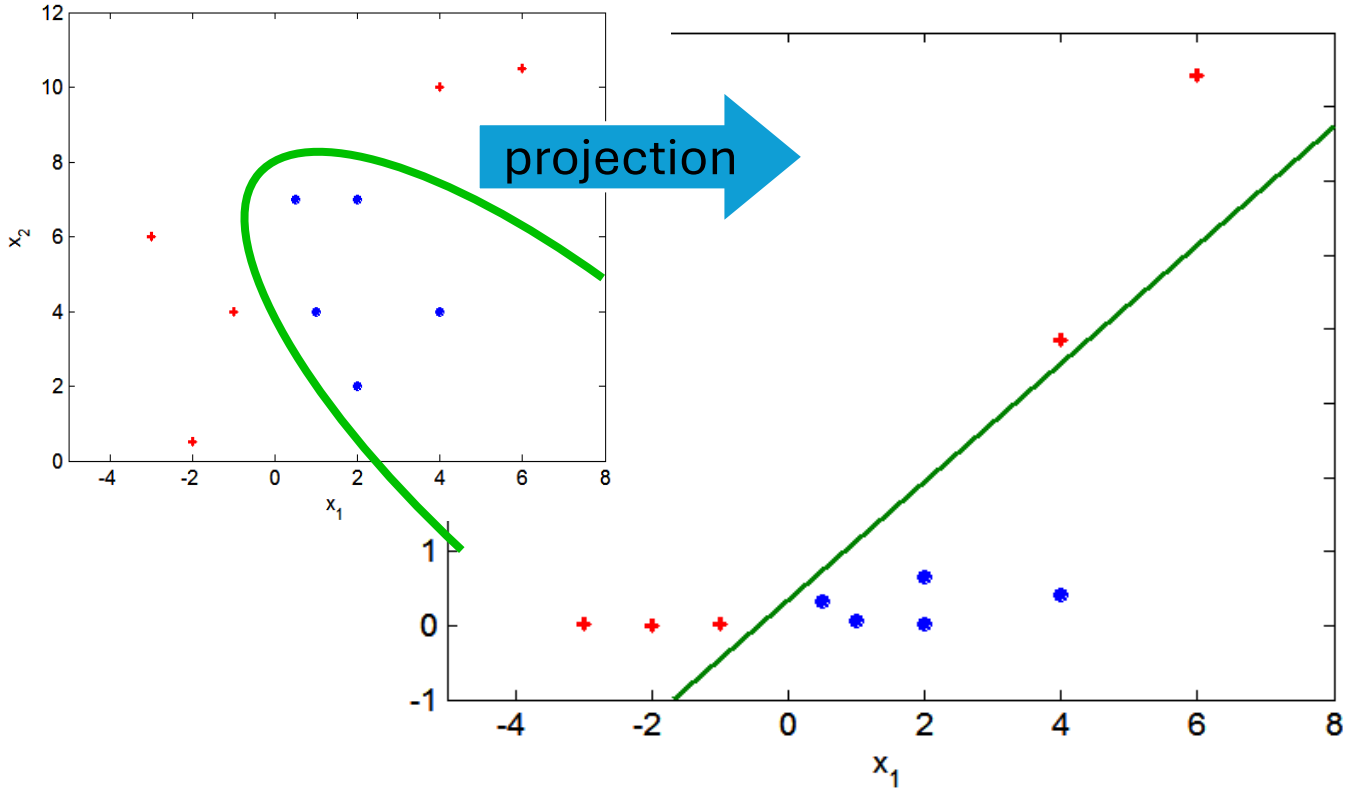


Nonlinear Support Vector Machines

SVMs look for linear decision boundaries. What if decision boundary is not linear?



Nonlinear Support Vector Machines



- Project data into a higher dimensional space where the classes are linearly separable.

- Projection is expensive!
  Kernel trick: Compute the similarity (inner product) in the projected space directly from the original data. This trick can be used with other method like clustering as well.

Topics

- Rule-Based Classifier
- Nearest Neighbor Classifier
- Naive Bayes Classifier
- Logistic Regression
- Artificial Neural Networks
- Support Vector Machines
- **Ensemble Methods**

Ensemble Methods

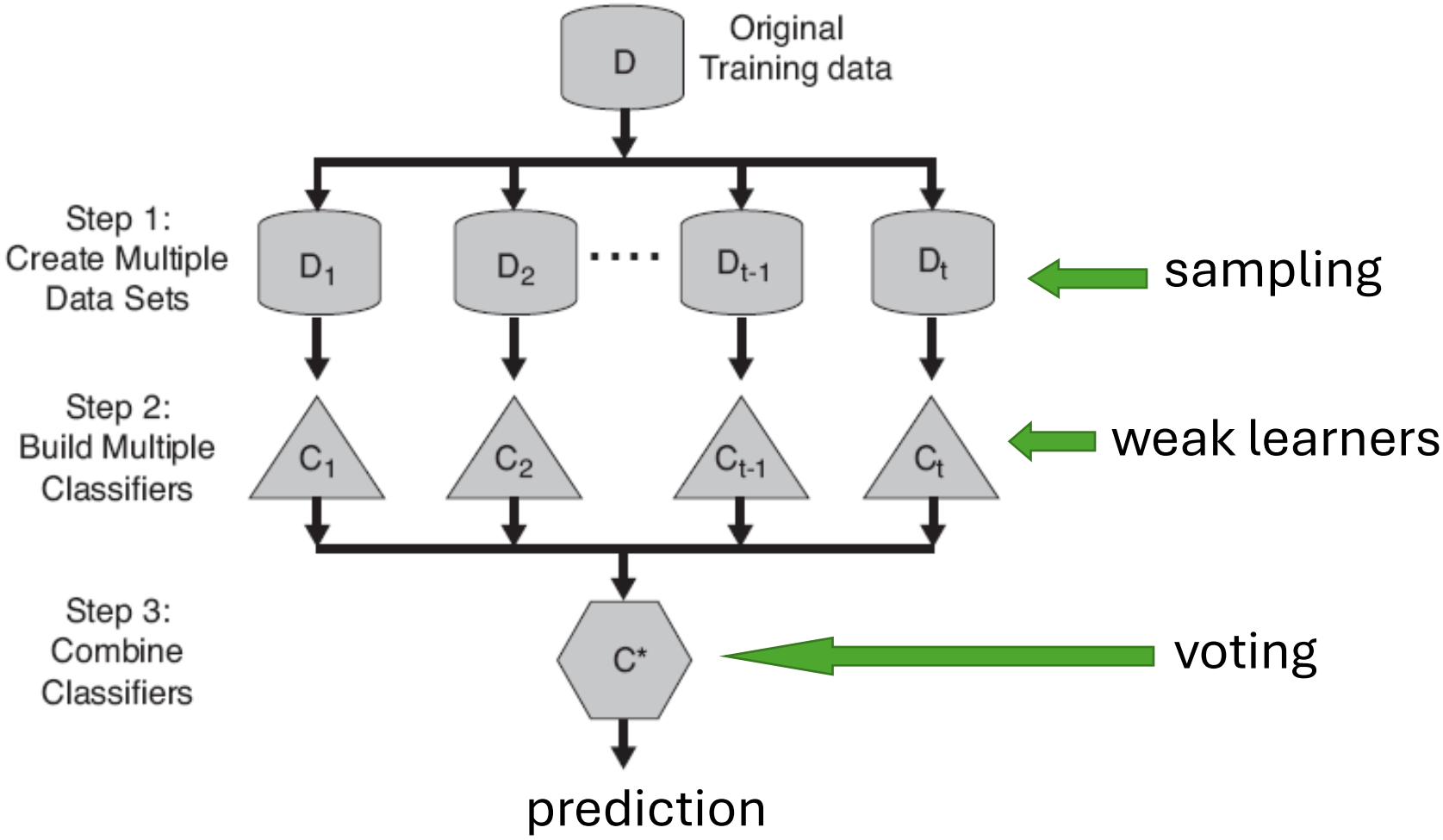
Method

1. Construct a set of (possibly weak) classifiers from the training data.
2. Predict class label of previously unseen records by aggregating predictions made by multiple classifiers.

Advantages

- Improve the stability and often also the accuracy of classifiers.
- Reduces variance in the prediction.
- Reduces overfitting.

General Idea



Why does it work?

- Suppose there are 25 base classifiers.
  - —Each classifier has error rate $\epsilon = 0.35$
  - —Assume classifiers are independent (different features and/or training data).
- Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{k=13}^{25} \binom{25}{i} \epsilon^k (1-\epsilon)^{25-k} = 0.06$$

= Probability that the majority (13 or more classifiers) make the wrong decision.

Note

- The binomial coefficient gives the number of ways you can choose k elements out of $n = 25$.

Examples of Ensemble Methods

- How to generate an ensemble of independent classifiers?

Bagging	Boosting	Random Features
• Use several samples	• Increase the weight for misclassified data points	• Use different features

Bagging (Bootstrap Aggregation)

1. **Sampling with replacement** (bootstrap sampling)

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

Note: some objects are chosen multiple times in a bootstrap sample while others are not chosen! A typical bootstrap sample contains about 63% of the objects in the original data.

2. **Build classifiers,** one for each bootstrap sample (classifiers are hopefully independent since they are learned from different subsets of the data)

3. **Aggregate** the classifiers' results by averaging or voting

Boosting

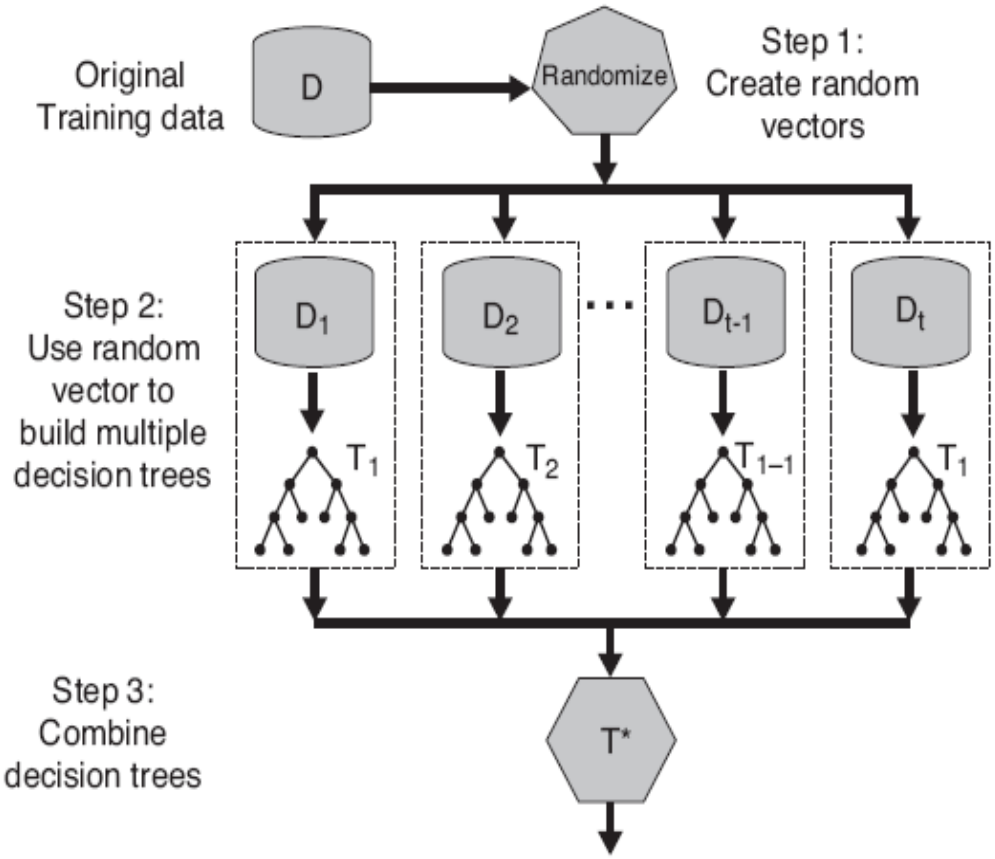
- Records that are incorrectly classified in one round will have their weights increased in the next

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

- Example 4 is hard to classify. Its weight is increased so it is more likely to be chosen again in subsequent rounds. This creates a larger error, and the classifier will try harder to predict it correctly.

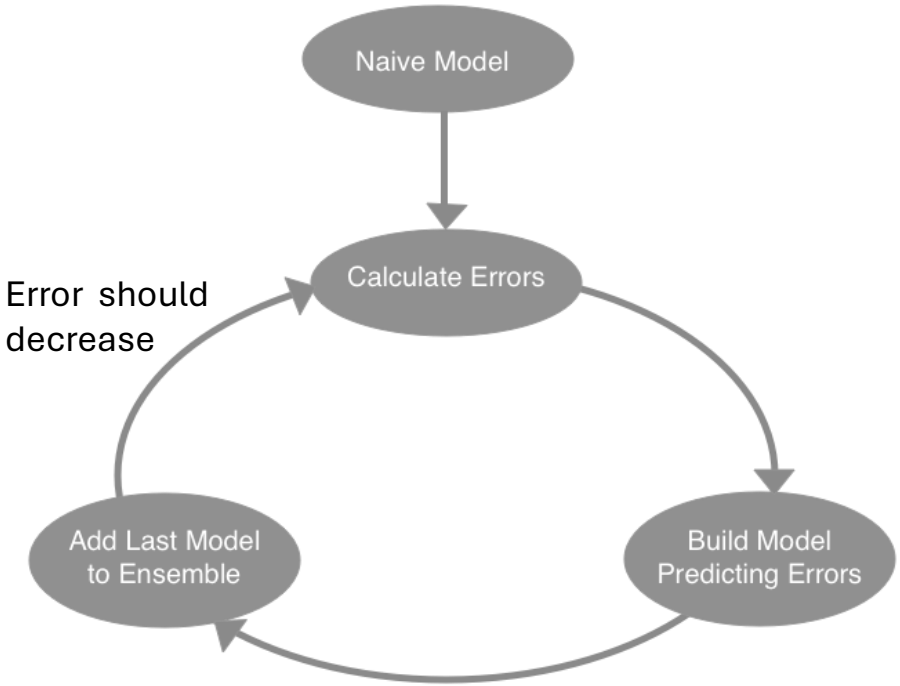
- **Popular algorithm**: AdaBoost (Adaptive Boosting) typically uses decision trees as the weak learner.

Random Forests



- Introduce two sources of randomness: “Bagging” and “Random input vectors”
- **Bagging method**: each tree is grown using a bootstrap sample of training data
- **Random vector method**: At each node, the best split is chosen only from a random sample of the m possible attributes.

Gradient Boosted Decision Trees (XGBoost)



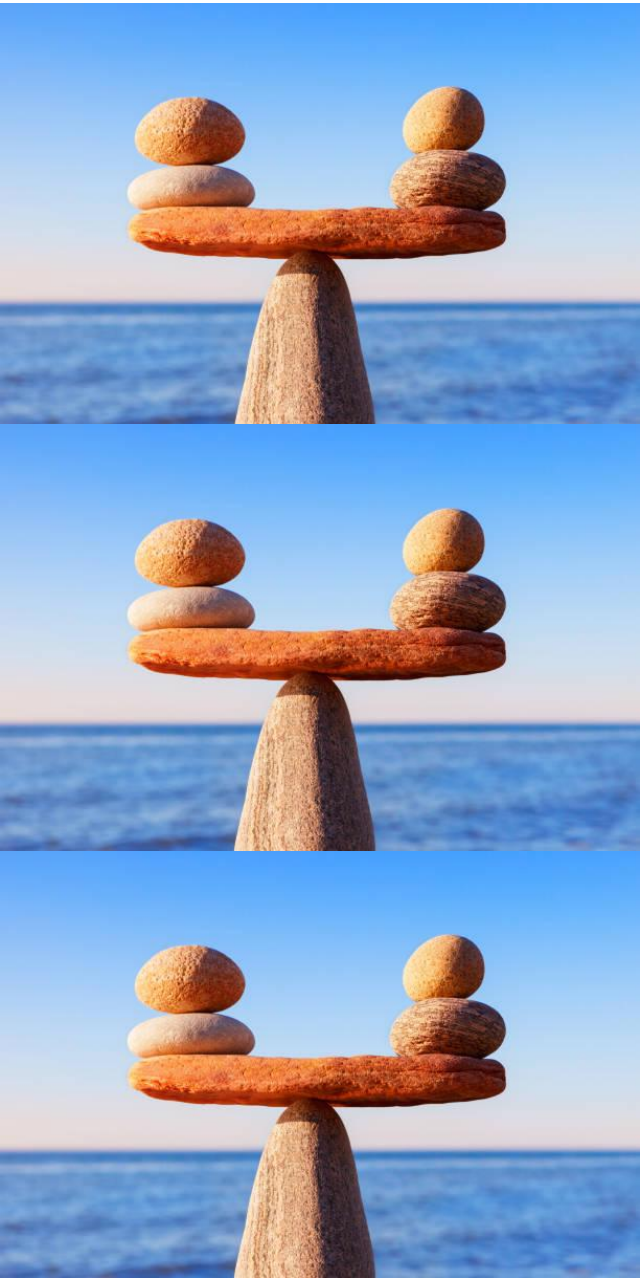
- **Idea**: build models (typically CART trees) to predict (correct) errors (= boosting). Used for classification or regression.

- **Approach**:
  1. Start with a naive (weak) model. E.g., a single tree.
  2. Calculate the error for each observation in the dataset.
  3. Build a new model to predict these errors and add to the ensemble.
  4. Go to 2.

- **Prediction**: Sum of the base tree and all the corrections.

$$y = \sum_{k=1}^{K} y_k$$

Other names: Gradient Boosting Machine (GBM), Multiple Additive Regression Trees (MART), Boosted Regression Trees (BRT), TreeNet



Topics

- Other Classification Methods
  - Rule-Based Classifier
  - Nearest Neighbor Classifier
  - Naive Bayes Classifier
  - Logistic Regression
  - Artificial Neural Networks
  - Support Vector Machines
  - Ensemble Methods
- **Class Imbalance Problem**

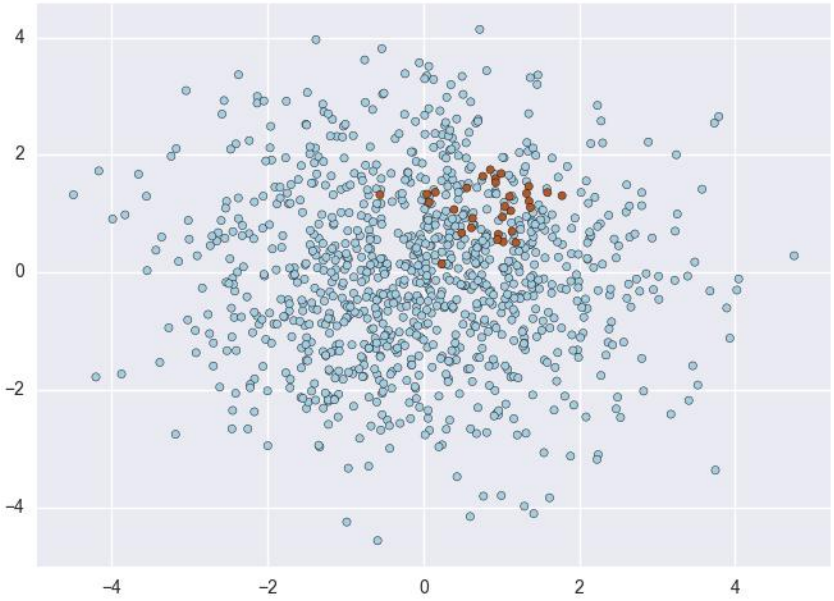
Class Imbalance Problem

Consider a 2-class problem
- —Number of Class 0 examples = 9990
- —Number of Class 1 examples = 10

A simple model:
- —Always predict Class 0
- —accuracy = 9990/10000 = 99.9 %
- — error = 0.1%

Issues:

1. Evaluation: accuracy is misleading.
2. Learning: Most classifiers try to optimize accuracy/error. **These classifiers will not learn how to find examples of Class 1!**



Class Imbalance Problem: Evaluation

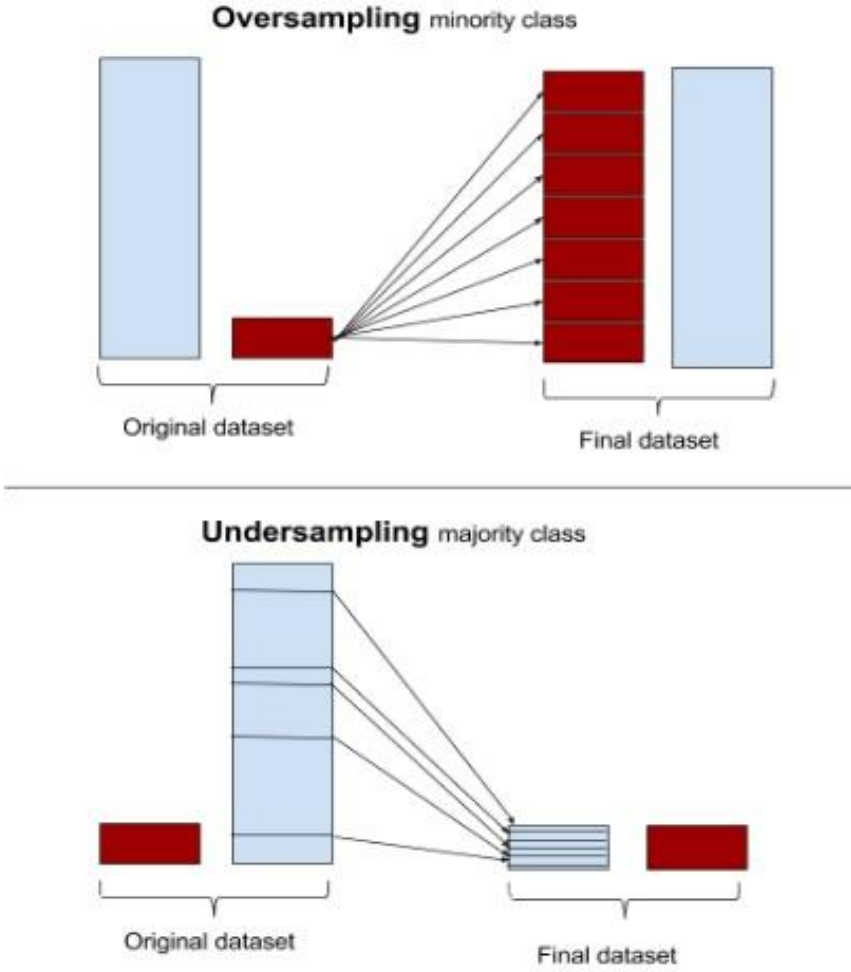
Do not use accuracy to evaluate problems with strong class imbalance!

Use instead:

- ROC curves and AUC (area under the curve) for binary classifiers.
- Cohen's Kappa which corrects for random accuracy.
- Misclassification cost.
- Precision/Recall plots or the F1 Score.

Class Imbalance Problem: Learning

- **Do nothing**. Sometimes you get lucky!
- **Balance the data set**: Down-sample the majority class and/or up-sample the minority class (use sampling with replacement). Synthesize new examples with SMOTE.
  This will artificially increase the error for a mistake in the minority class.
- Use **algorithms** that can deal with class imbalance (see next slide).
- Throw away minority examples and switch to an **anomaly detection** framework.



Class Imbalance Problem: Learning

Some algorithms that can deal with class imbalance:

- Use a **cost-sensitive classifier** that considers a cost matrix (not too many are available).
- Use boosting techniques like **AdaBoost**.
- Use a classifier that **predict a probability** $P(y \mid x)$ and instead of choosing $\text{argmax}_y P(y \mid x)$, choose the minority class if

$$\mathrm{P}\left(y_{\text{minority}} \mid x\right) > \theta,$$

where θ is the decision threshold that is made smaller to account for the imbalance.

Many classifiers naturally produce probabilities since they try to approximate the Bayesian decision rule.
For example, for decision trees probabilities can be estimated using the positive and negative training examples in each leaf node.

Conclusion

- **Bias**: There are many ways to implement the classification function. Each of them has a different inductive bias.
- **Feature extraction and feature creation** are important (e.g., interaction effects in linear models). Deep learning can also learn to create features.
- **Overfitting and model variability** are issues. Appropriate validation and test data need to be used to produce and evaluate models that generalize well.
- Accuracy is problematic for **imbalanced data sets**. Rebalancing the data may be a necessity.
- **Model explainability** is often important. Rules and trees may be easier to explain than other models.
- **(Deep) artificial neural networks** are a very powerful method, but other methods may be preferable since ANNs:
  - Have no or very low bias and need lots of training data.
  - Have many hyperparameters that need to be tuned experimentally.
  - The model represents a black box, and it is hard to explain why it makes decisions.