9. Random forests, perceptrons and support vector machines

Shabana K M
PhD Research Scholar
Computer Science and Engineering

IIT Palakkad

21 November 2021

INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

Recap

- Decision trees
    - classification and regression trees
    - learning a decision tree
    - selecting the splitting attribute
    - information gain, Gini impurity
    - pros and cons
    - avoiding overfitting of trees
    - bias and variance

Bagging

Ensemble method

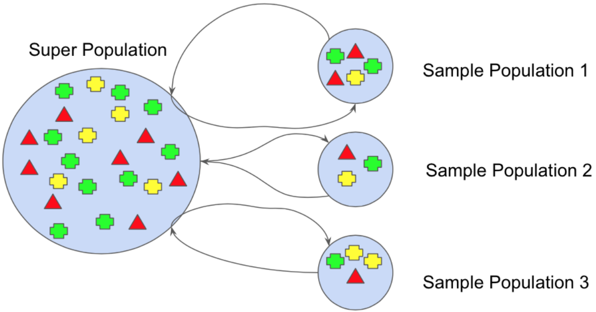
- combines predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model

Bootstrap Aggregation or Bagging

- simple and very powerful ensemble method
- used to reduce the variance for algorithms with high variance (eg:- decision trees)
- each model in the ensemble built using a bootstrap sample of training data

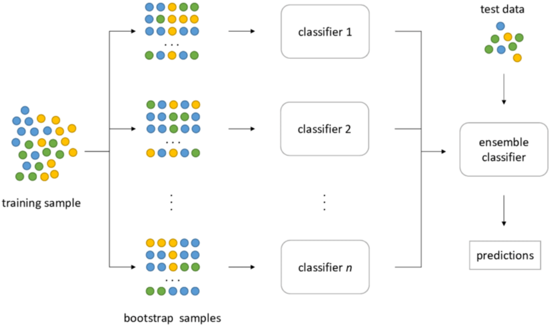
Bootstrap sampling

- Given a dataset D containing N training examples, generate another dataset D' by drawing N samples at random with replacement from D



Bagging

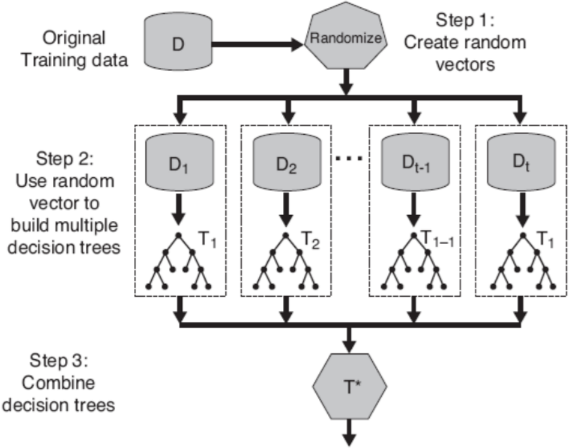
1. Create k bootstrap samples $D_1, \dots, D_k$
2. Train distinct model on each D_i
3. Classify new instance by majority vote/average



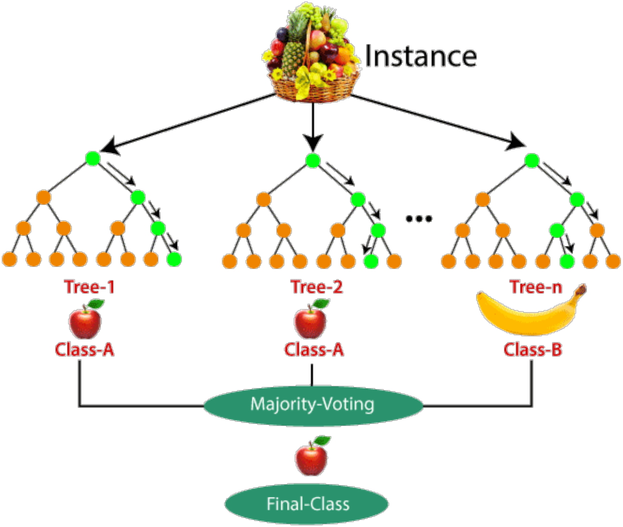
Random forests

- Grow a forest of many trees
- Each tree grown on an independent **bootstrap sample** from the training data
- At each node:
  1. **Random vector method:** select m features at random out of all possible features (independently for each node)
  2. find the best split based on the m selected features
- Grow the trees to maximum depth (classification)
- Vote/average the trees to get predictions for new data

Random forests: The idea



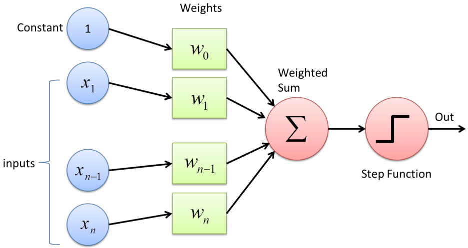
Prediction using random forest



Perceptrons

- supervised learning algorithm for binary classification
- goal: find a separating hyperplane
  - divide the input space into two classes based on their labels
  - for separable data, guaranteed to find one
- linear classifier
  - makes predictions by combining a set of weights with the feature vector
- online learning algorithm
  - processes one example at a time

Perceptron as an artificial neuron



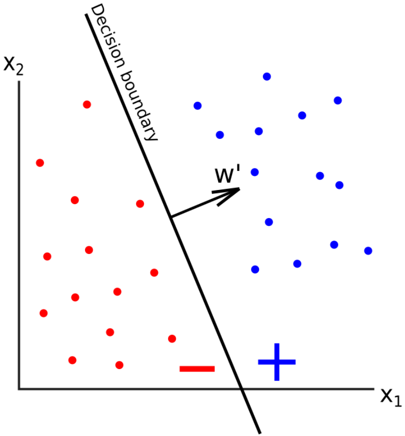
- receives input from D other neurons, one for each input feature
- each incoming connection has a weight
  - feature values are strength of inputs
- neuron sums up all the weighted inputs
- based on this sum decides whether to "fire" or not
  - *firing*: positive example
  - *not-firing*: negative example

Perceptron: Activation

- The amount of **activation** of a neuron is given by: $a = \sum_{i=1}^{D} w_i x_i$
- Let b be the threshold for activation, i.e., if $a \geq b$, the neuron fires
- $\sum_{i=1}^{D} w_i x_i \geq b$ is equivalent to $\sum_{i=1}^{D} w_i x_i - b \geq 0$
  where b is referred to as **bias**
- Thus if $w^T x \geq 0$, then x is predicted as a positive example
  where $w = [w_0 \;\; w_1 \;\; \ldots \;\; w_D], x = [-1 \;\; x_1 \;\; \ldots \;\; x_D]$
  Here w_0 represents the *bias*

Perceptron: decision boundary

$$y = w^T x$$



Perceptron learning algorithm

Input: A sequence of training examples $(x_1, y_1), (x_2, y_2), \dots$ where all $x_i \in \mathcal{R}^D$, $y_i \in \{-1, 1\}$

- Initialize $w_0 = 0 \in \mathcal{D}$
- for iter=1, ..., *maxIter*
  - for each training example (x_i, y_i):
    - predict $y' = \text{sign}(w_t^T x_i)$
    - if $y_i \neq y'$: $w_{t+1} = w_t + y_i x_i$

Perceptron learning algorithm

- **online algorithm:** processes one example at a time
- **error-driven algorithm:** updates parameters only when it makes an error
- weight updations
  - on positive examples: $w_{t+1} = w_t + x_i$
  - on negative examples: $w_{t+1} = w_t - x_i$

Intuition behind the weight update

Suppose the perceptron has made a mistake on a positive example i.e., $y = 1$ and $w_t^T x < 0$

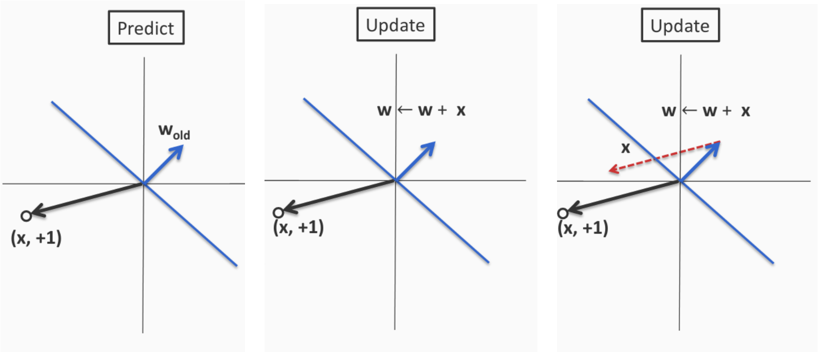
New weight vector $\boldsymbol{w_{t+1}} = w_t + x$

New dot product will be: $\boldsymbol{w_{t+1}^T x} = (w_t + x)^T x = w_t^T x + x^T x > w_t^T x$

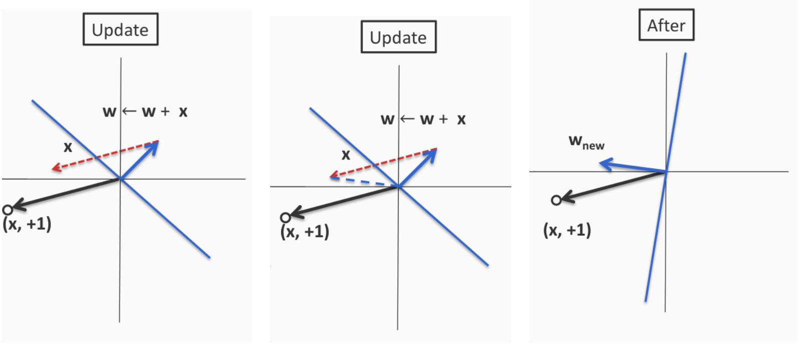
For a positive example, the perceptron update will increase the score assigned to the same input

Similar reasoning for negative examples

Geometry of the perceptron update

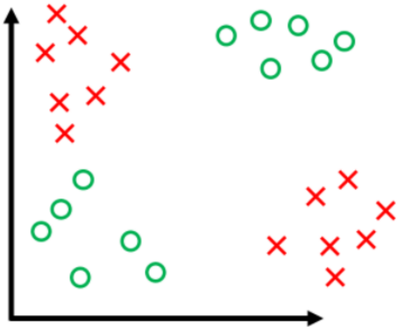


Geometry of the perceptron update



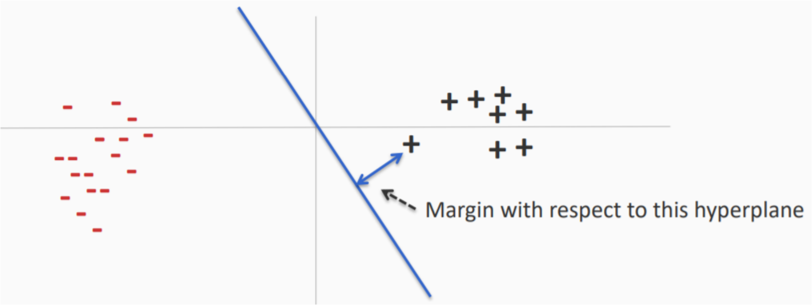
Limitations of perceptron

1. **Works only for linearly separable data**



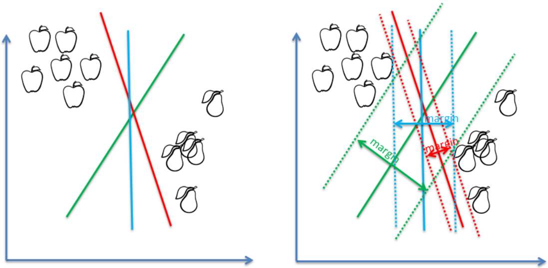
Margin

The **margin** of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it



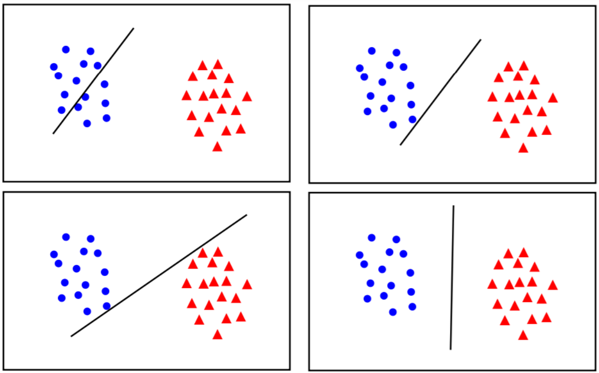
Limitations of perceptron

2. **Finds any** `hyperplane` **separating the two classes**



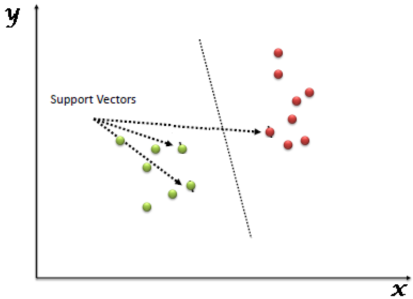
- decision boundary close to training data (`unstable`)
- prefer a larger margin for generalization

Which decision boundary to choose?



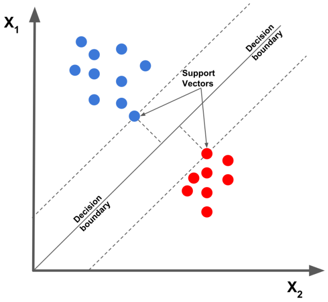
maximum margin solution: most stable under perturbations of the inputs

Support vectors



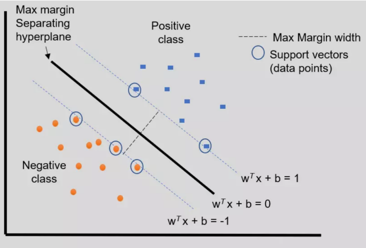
- data points that lie closest to the decision surface/hyperplane
- data points most difficult to classify
- have direct bearing on the optimum location of the decision surface

Support vector machines (SVM)



- maximize the margin around the separating hyperplane
- decision function fully specified by the support vectors

Support vector machines



- Choose normalization for w such that $w^T x_+ + b = +1$ and $w^T x_- + b = -1$
  - x_+: support vector for positive class
  - x_-: support vector for negative class
- **Margin** is given by $\frac{w}{||w||}.(x_+ - x_-) = \frac{w^T x_+ - w^T x_-}{||w||} = \frac{2}{||w||}$

SVM - Optimization

- Learning the SVM can be formulated as an optimization problem

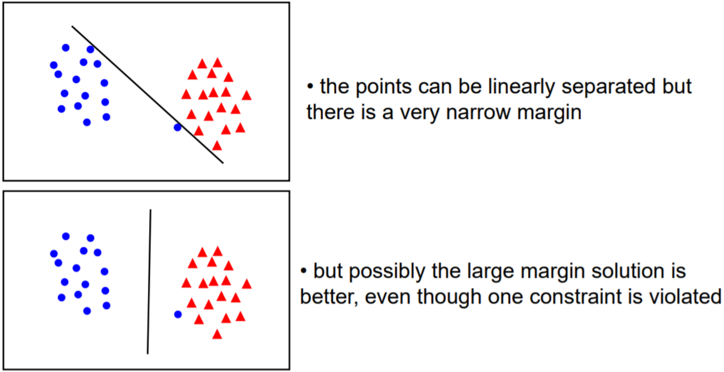
$$\max_{w} \frac{2}{||w||} \text{ subject to } w^T x_i + b \begin{cases} \geq +1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases} \text{ for } i = 1, .., N$$

- Or equivalently,

$$\min_{w} ||w||^2 \text{ subject to } y_i(w^T x_i + b) \geq 1 \text{ for } i = 1, .., N$$

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

Linear separability: Which hyperplane?



• In general there is a trade off between the margin and the number of mistakes on the training data

References

1. https://people.csail.mit.edu/dsontag/courses/ml16/slides/lecture11.pdf
2. https://www.cs.utah.edu/~zhe/pdf/lec-10-perceptron-upload.pdf
3. http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf
4. http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf
5. https://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

Thanks Google for the pictures!