CS 559 Machine Learning

Linear Classification

In Suk Jang
Department of Computer Science
Stevens Institute of Technology

Lecture Outline

- Model Selection
- Optimization
- The Bias-Variance Tradeoff
- Classification
- Linear Discriminant Analysis
- The Perception Algorithm
- Naïve Bayes

What is Model Selection?

Given a set of models $M = \{M_1, M_2, \ldots, M_R\}$, choose the model that is expected to do the best on the test data. M may consist of:

- Same learning model with different complexities or hyperparameters.
  - Nonlinear regression: polynomials with different degrees K-Nearest Neighbors: Different choices of K
  - Decision Trees: Different choices of the number of levels/leaves SVM: Different choices of the misclassification penalty Regularized models: Different choices of the regularization parameter
  - Kernel based methods: Different choices of kernels ...and
  - almost any learning problem
- Different learning models (e.g. SVM, kNN, DT, etc)

Note: usually considered in supervised learning but unsupervised learning faces this issue too.

Held-out Data

- Set aside a fraction (10-20%) of the training data.
- This part becomes our held-out data (validation/development)
- **Remember**: Held-out data is NOT the test data
- Train each model using the remaining training data
- Evaluate error on the held-out data
- Choose the model with the smallest held-out error
- Problems:
    - wastes training data
    - if there was an unfortunate split (can be alleviated by repeated random subsampling)

Cross-Validation

K-fold Cross-Validation on N training examples

- Create K equal sized partitions of the training data
- Each partition has N/K examples
- Train using $K - 1$ partitions, validate on the remaining partition
- Repeat the same K times, each with a different validation partition
- Choose the model with the smallest average validation error
- Usually K is chosen as 10

Leave-One-Out (LOO) Cross-Validation

Special case of K-fold Cross-Validation when $K = N$

- Each partition is now an example
- Train using $N - 1$ examples, validate on the remaining example
- Repeat the same N times, each with a different validation example
- Choose the model with the smallest average validation error
- can be expensive for large N. Typically used when N is small

Random Subsampling Cross-Validation

- Randomly subsample a fixed fraction $\alpha N (0 < \alpha < 1)$ of examples; call it the validation set
- Training using the rest of the examples, measure error on the validation set
- Repeat K times, each with a different randomly chosen validation set
- Choose the model with the smallest average validation error
- Usually α is chose as 0.1, K as 10

Bootstrapping

- Given a set of N examples
- Idea: Sample N elements from this set with replacement (already sampled elements can e picked again)
- Use this new set as the training data
- The set of examples not selected as the validation data
- For large N, training data consists of about only **63% unique** examples
- Expected model error:

$$e = 0.632 \times e_{\text{test}} + 0.368 e_{\text{training}}$$

- This can break down if we overfit and $e_{\text{training}} = 0$

Bradley Efron & Robert Tibshirani. *Improvements on Cross-Validation: The 632+ Bootstrap Method*

Information Criteria based methods

- Akaike Information Criteria (AIC)

$$\text{AIC} = 2k - 2\log(L)$$

- Bayesian Information Criteria (BIC)

$$\text{BIC} = k\log(N) - 2\log(L)$$

- k: # of model parameters
- n: # of data examples
- L: maximum value of the model likelihood
- Applicable for probabilistic models
- AIC/BIC penalize model complexity

Feature Selection

Selecting a useful subset from all the features. Why?

- Some algorithms scale (computationally) poorly with increased dimension
- Irrelevant features can confuse some algorithms
- Redundant features adversely affect regularization
- Removal of features can increase (relative) margin (and generalization)
- Reduces data set and resulting model size
    - Note: Feature Selection is different from Feature Extraction. The latter transforms original features to get a small set of new features
    - More on feature extraction when we cover **Dimensionality Reduction**

Feature Selection Methods

- Methods agnostic to the learning algorithm
    - Preprocessing based methods
        - E.g., remove a binary feature if its ON in very few or most examples
    - Filter Feature Selection methods
        - Use some ranking criteria to rank features Select the top ranking features
- Wrapper Methods (keep the learning algorithm in the loop)
  Requires repeated runs of the learning algorithm with different set of features
    - Can be computationally expensive

Optimization – How to optimize?

Gradient:

$$\nabla_{\boldsymbol{w}} \text{TrainLoss}(\boldsymbol{w})$$

The direction that increases the loss the most.

```
Algorithm: Gradient Descent
Initialize w
For $t = 1 \ldots.,T$
        $w \leftarrow w - \eta \nabla_{\boldsymbol{w}} \text{TrainLoss}(\mathbf{w})$
```

Objective Function

$$\text{TrainLoss}(\mathrm{w}) = \frac{1}{\mathrm{N}} \sum_{(x,y) \in D_{train}} \left(\mathbf{w}^{\mathbf{T}}\mathbf{x} - \mathrm{y}\right)^2$$

Optimization – How to optimize?

Gradient:

$$\nabla_{\boldsymbol{w}} \text{TrainLoss}(\boldsymbol{w}) = \frac{1}{\text{N}} \sum_{(x,y) \in D_{train}} 2\left(\mathbf{w}^{\mathbf{T}}\mathbf{x} - \text{y}\right)^2 \mathbf{x}$$

Gradient descent update:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\boldsymbol{w}} \text{TrainLoss}(\mathbf{w})$$

Each iteration requires going over all training examples – expensive when have lots of data

Stochastic and Mini-batch Gradient Descent

Stochastic Gradient Descent

- Update for each training example.

Mini-batch Gradient Descent

- Takes the best of both standard Gradient Descent and SGD and performs an update for every batch with n training examples in each batch.

Bias and Variance Tradeoff

Assuming a training set $(x_1, \ldots, x_n)$ and their real associated *y* values. There is a function with noise $y = f(x) + \epsilon$ where the noise ϵ has mean 0 and variance σ^2.

We want to find a function $\hat{f}(x)$ that approximates the true function $f(x)$. Expected squared prediction error at a point x is:

$$E\left[\left(y - \hat{f}(x)\right)^2\right] = \left(E[\hat{f}(x)] - f(x)\right)^2 + \left(E[\hat{f}(x)^2] - E^2[\hat{f}(x)]\right) + \sigma^2$$

expected loss = $(\text{bias})^2$+variance+noise

Bias and Variance

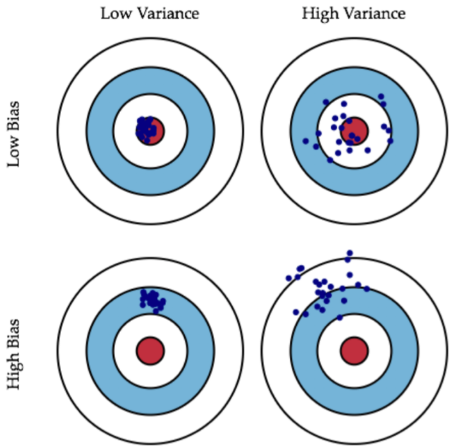


Figure: Graphical illustration of bias vs. variance

Classification

Classification task: finding a function f that classifies examples into given set of categories $\{C_1, C_2, .., C_k\}$

$$\mathbf{x} \rightarrow \boxed{\text{function } f} \rightarrow y \in \{C_1, C_2, .., C_k\}$$

A classification example:

$\rightarrow \boxed{\text{function } f} \rightarrow$ “Cat”

Decision Theory for Classification

Decision theory, when combined with probability theory, allows us to make optimal decisions in situations involving uncertainty.

- Training data: input values X and target values $\mathbf{y}$
- Inference stage: use the training data to learn a model for $p(C_k|\mathbf{x})$
- Decision stage: use the given posterior probabilities to make optimal class assignments.

Generative Methods

- Solve the inference problem of estimating the class-conditional densities $p(\mathbf{x}|C_k)$ for each class C_k
- Infer the prior class probabilities $p(C_k)$
- Use Bayes' theorem to find the class posterior probabilities:

$$p(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{p(\boldsymbol{x})}$$

where

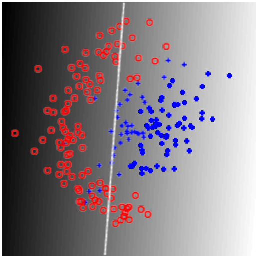
$$p(x) = \sum_{\mathrm{k}} \mathrm{p}(\mathbf{x}|\mathrm{C_k})\mathrm{p}(\mathrm{C_k})$$

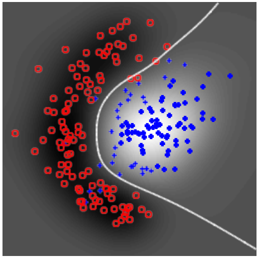
- Use decision theory to determine class membership for each new input **x**

Discriminative Methods

- Solve directly the inference problem of estimating the class posterior probabilities $p(C_k|\mathbf{x})$
- Discriminative Functions: Find a function $f(x)$ which maps each input directly onto a class label. Probabilities play no role here.
- Use decision theory to determine class membership for each new input $\mathbf{x}$

Linear Discriminant Functions





Of course, linear algorithms can be used together with nonlinear feature spaces or nonlinear basis functions in order to solve nonlinear classification problems!

Linear discriminants separate the space by a hyperplane, and the parameters define its normal vector.

- Decision function: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \omega_0$
- Classification:

  if $f(\mathbf{x}) > 0$ say $\mathbf{x}$ belongs to class 1 if
  $f(\mathbf{x}) < 0$ say $\mathbf{x}$ belongs to class -1

- The decision-surface has equation $f(\mathbf{x}) = 0$, and is a hyperplane of dimensionality $D - 1$.
- $\mathbf{w}$ is the normal vector to the hyperplane, and points into the positive class or negative class.
- $\omega_0$determines the location of the decision-surface
- $|f(\mathbf{x})|$ is proproptional to the perpendicular distance to the decision-surface (with factor 1 if $||\mathbf{w}|| = 1$).

Linear Discriminant Functions-Geometrical Properties

- Decision boundary:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \omega_0 = 0$$

- Let $\mathbf{x}_1$, $\mathbf{x}_2$ be two points which lie on the decision boundary

$$f(\mathbf{x}_1) = \mathbf{w}^T\mathbf{x}_1 + \omega_0 = 0, f(\mathbf{x}_2) = \mathbf{w}^T\mathbf{x}_2 + \omega_0 = 0$$
$$\Rightarrow \mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0$$

- **w** represents the orthogonal direction to the decision boundary.

Linear Discriminant Functions-Geometrical Properties Cont.

$$x_1 = x_2 + r\frac{w}{\|w\|}$$

$$f(x_2) = f\left(x_1 - r\frac{w}{\|w\|}\right)$$
$$= (w^T x_1 + w_0) - r\|w\|$$
$$f(x_1) - r = 0$$

$$\rightarrow r = \frac{f(x_1)}{\|w\|}$$

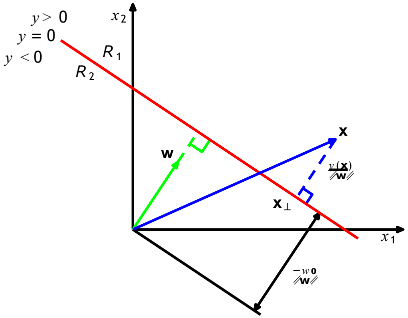
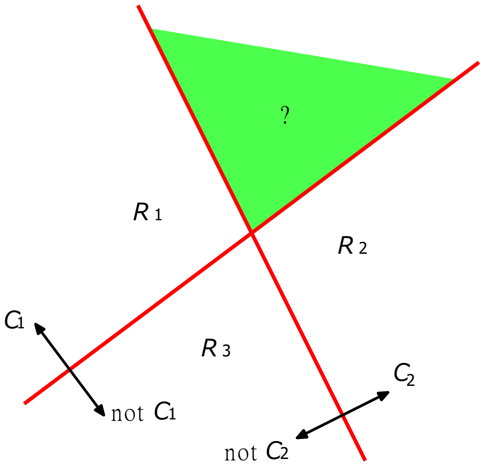


Figure: Signed orthogonal distance of the origin from the decision

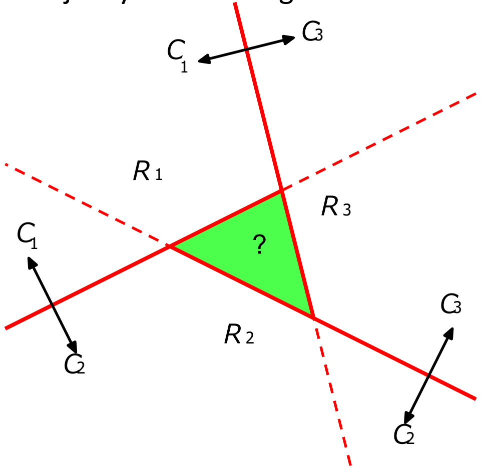
Linear Discriminant Functions: Multiple classes

one-versus-the-rest: K-1 classifiers each of which solves a two-class problem of separating points of C_k from points not in that class.



Linear Discriminant Functions: Multiple classes

one-versus-one: K(K-1)/2 binary discriminant functions, one for every possible pair of class. Each point is then classified according to a majority vote amongst the discriminant functions.

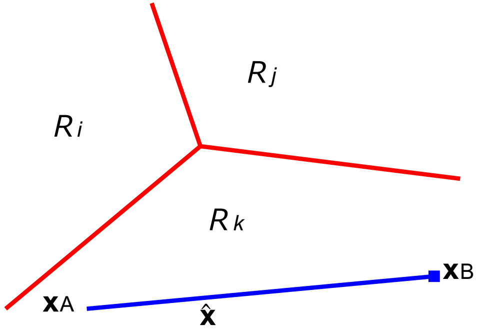


Linear Discriminant Functions: Multiple classes

- Solution: consider a single K-class discriminant comprising K linear functions of the form

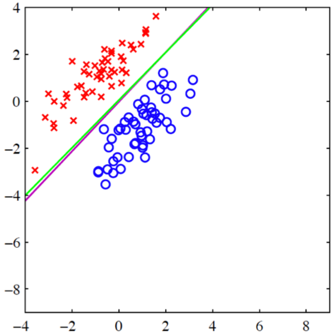
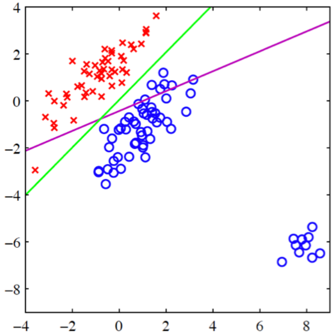
$$f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

- Assign a point $\mathbf{x}$ to class C_k if $f_k(\mathbf{x}) > f_j(\mathbf{x}) \forall j \neq k$
- The decision boundary between class C_k and class C_j is given by: $f_k(\mathbf{x}) = f_j(\mathbf{x}) \Rightarrow (\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} + (w_{k0} - w_{j0}) = 0$



Least square classification

- We have to fit the function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \omega_0$ to data.
- Simply do a linear regression from $\mathbf{x}$ to y by minimizing the sum-of-squared errors $\sum_n (f(\boldsymbol{x_n}) - y_n)^2$.
- $$w_{reg} = \left(\sum_n x_n x_n^T\right)^{-1} \sum_n x_n y_n$$
- Q: In what situations might this be a bad idea?

Least square classification

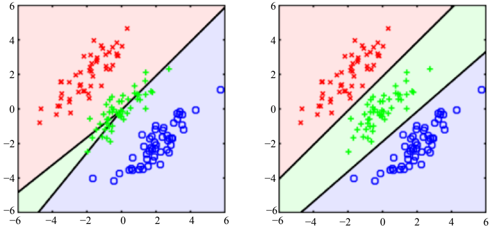


Figure: Left: using a least-squares discriminant; Right: using logistic regression

Bishop PRML Figure 4.5

Fisher’s linear discriminant

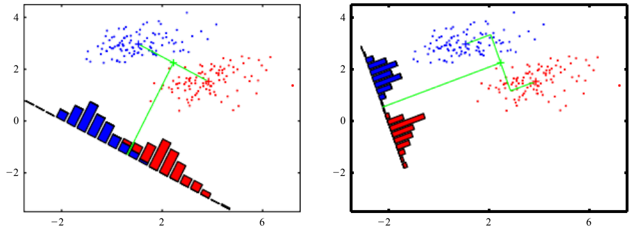
- One way to view a linear classification model is in terms of dimensionality reduction.
- Two class case: suppose we project **x** onto one dimension:

$$f = \mathbf{w}^T \mathbf{x}$$

- Set a threshold t:

$$\begin{array}{ll} \text{if } f \leq t & \text{assign } C_1 \text{ to } \mathbf{x} \\ \text{otherwise} & \text{assign } C_2 \text{ to } \mathbf{x} \end{array}$$

Fisher’s linear discriminant



- Find an orientation along which the projected samples are well separated;
- This is exactly the goal of linear discriminant analysis (LDA);
- In other words: we are after the linear projection that best separates the data, i.e. best discriminates data of different classes.

Fisher's linear discriminant

- Two classes: $\{C_1, C_2\}$
- N_1 samples of class C_1
- N_2 samples of class C_2
- Consider $\mathbf{w} \in \mathrm{R}^d$ with $||\mathbf{w}|| = 1$
- Then: $\mathbf{w}^T \mathbf{x}$ is the projection of $\mathbf{x}$ along the direction of $\mathbf{w}$.
- We want the projections $\mathbf{w}^T \mathbf{x}$ where $\mathbf{x} \in C_1$ separated from the projections $\mathbf{w}^T \mathbf{x}$ where $\mathbf{x} \in C_2$

Fisher’s linear discriminant

- A measure of the separation between the projected points is the difference of the sample means:
    - Sample mean of class C_+:

$$m_1 = \frac{1}{N_1} \sum\nolimits_{x \in C_1} x$$

    - Sample mean for the projected points:

$$m_1 = \frac{1}{N_1} \sum_{x \in C_1} \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x} = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{m}_{\boldsymbol{1}}$$

$$\rightarrow |m_1 - m_2| = \boldsymbol{w}^{\boldsymbol{T}} (\boldsymbol{m}_{\boldsymbol{1}} - \boldsymbol{m}_{\boldsymbol{2}})$$

- We wish to make the above difference as large as we can. In addition, ...

Fisher's linear discriminant

To obtain good separation of the projected data, we really want the difference between the means to be large relative to some measure of the standard deviation of each class:

- Scatter of the projected samples of class *C1*:

$$s_1^2 = \sum_{x \in C_1} (w^T x - m_1)^2$$

- Total within-class scatter of the projected samples:

$$s_1^2 + s_2^2$$

- Fisher linear discriminant analysis:

$$\arg\max_{w} \frac{|m_1 - m_2|^2}{s_1^2 + s_2^2}$$

Fisher's linear discriminant

Fisher's criterion *J(**w**)*: $J(w) = \frac{|m_1 - m_2|^2}{s_1^2 + s_2^2}$

To obtain *J(**w**)* as an explicit function of **w**, we define the following matrices:

$$S_1 = \sum_{x \in C_1} (\boldsymbol{x} - \boldsymbol{m_1})(\boldsymbol{x} - \boldsymbol{m_2})^T$$

Within-class scatter matrix:

$$S_w = S_1 + S_2$$

Then:

$$s_1^2 = \sum_{x \in C_1} \left(\boldsymbol{w^T x} - m_1\right)^2 = \sum_{x \in C_1} \left(\boldsymbol{w^T x} - \boldsymbol{w^T m_1}\right)^2$$

$$= \sum_{x \in C_1} \boldsymbol{w^T}(\boldsymbol{x} - \boldsymbol{m_1})(\boldsymbol{x} - \boldsymbol{m_1})^{\boldsymbol{T}}\boldsymbol{w} = \boldsymbol{w^T S_1 w}$$

Fisher's linear discriminant

So, $s_1^2 = w^T S_1 w$ and $s_2^2 = w^T S_2 w$

Thus,

$$\begin{aligned} s_w = s_1^2 + s_2^2 &= \boldsymbol{w^T S_1 w} + \boldsymbol{w^T S_2 w} \\ &= \boldsymbol{w^T (S_1 + S_2) w} \\ &= \boldsymbol{w^T S_w w} \end{aligned}$$

Similarly:

$$\begin{aligned} (m_1 - m_2)^2 &= \left(\boldsymbol{w^T m_1} - \boldsymbol{w^T m_2}\right)^2 \\ &= \boldsymbol{w^T (m_1 - m_2)(m_1 - m_2)^T w} \\ &= \boldsymbol{w^T S_B w}. \end{aligned}$$

where $S_B = \boldsymbol{(m_1 - m_2)(m_1 + m_2)^T}$ is the between-class scatter matrix.

Fisher’s linear discriminant

We have obtained:

$$J(w) = \frac{w^T S_B w}{w^T S_w w}$$

J(w) is maximized when

$$(\boldsymbol{w^T S_B w})\boldsymbol{S_w w} = (\boldsymbol{w^T S_w w})\boldsymbol{S_B w}$$

We observe that

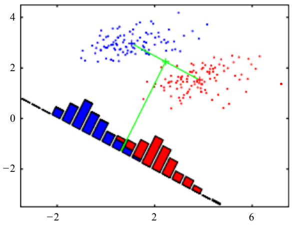
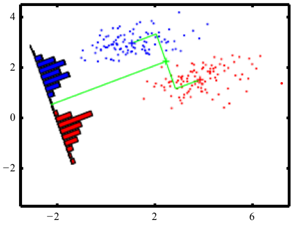
$$\boldsymbol{S_B w} = (\boldsymbol{m_1} - \boldsymbol{m_2})(\boldsymbol{m_1} + \boldsymbol{m_2})^T \boldsymbol{w}$$

where $(\boldsymbol{m_1} + \boldsymbol{m_2})^T \boldsymbol{w}$ is a scalar and always in the direction of $(\boldsymbol{m_1} - \boldsymbol{m_2})$.

Solution:

$$\boldsymbol{w} = \boldsymbol{S_w^{-1}}(\boldsymbol{m_1} - \boldsymbol{m_2})$$

Fisher's linear discriminant Summary



Bishop PRML Figure 4.6

- $\boldsymbol{m}_1 = \frac{1}{N}\sum_{n \in C_1} \boldsymbol{x_n}$ & $\boldsymbol{m}_2 = \frac{1}{N_1}\sum_{n \in C_2} \boldsymbol{x_n}$
- Maximize Projection-distance of class means $\boldsymbol{w_{simple}} \propto \boldsymbol{m_1} - \boldsymbol{m_2}$
- Maximizing distance between means ignores that the projected variances might also be big.
- Fix: Maximize the ratio of between-class variance to within-class variance ('signal to noise'). Fisher criterion:

$$J_w = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

$$w_{Ida} = S_w^{-1}(m_1 - m_2)$$

Fisher’s linear discriminant Summary: Multi-Class

- The analysis can be extended to multiple classes.
- $S_w = \sum_{k=1}^{K} \sum_{\boldsymbol{x_i} \in C_k} (\boldsymbol{x_i} - \boldsymbol{m_k})(\boldsymbol{x_i} - \boldsymbol{m_k})^T$
- $S_B = \sum_{k=1}^{K} m_k (\boldsymbol{m_k} - \boldsymbol{m})(\boldsymbol{m_k} - \boldsymbol{m})^T$ where $\boldsymbol{m}$ is the global mean and m_k is the number of samples in class k.
- Solve: $S_B \mathbf{v} = \lambda S_W \mathbf{v}$ the generalized eigenvalue problem
- At most K-1 distinct solution eigenvalues
- The optimal projection matrix V to a subspace of dimension k is given by the eigenvectors corresponding to the largest k eigenvalues

Fisher's linear discriminant Summary: Multi-Class

- LDA is a linear technique for dimensionality reduction: it projects the data along directions that can be expressed as linear combination of the input features.
- The "appropriate" transformation depends on the data and on the task we want to perform on the data. Note that LDA uses class labels.
- Non-linear extensions of LDA exist (e.g., generalized LDA).

The Perceptron Algorithm (Frank Rosenblatt, 1957)

- First learning algorithm for neural networks.
- Originally introduced for character classification, where each character is represented as an image.

- Total input to output node:

$$\sum_j w_j x_j$$

- Output unit performs the function (activation function):

$$H(x) = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}$$

Perceptron: Learning Algorithm

- Goal: compute a mapping from inputs to the outputs.
- Example: two class character recognition problem.
    - Training set: set of images representing either the character 'a' or the character 'b' (supervised learning);
    - Learning task: learn the weights so that when a new unlabelled image comes in, the network can predict its label.
    - Setting: *d* input units (intensity level of a pixel), 1 output unit.

- The algorithm proceeds as follows:
  - Initial random setting of weights;
  - The input is a random sequence $\{\mathbf{x}_k\}$
  - For each element of class C_1, if output = 1 (correct), do nothing; otherwise, update weights;
  - For each element of class C_2, if output = 0 (correct), do nothing; otherwise, update weights;

Perceptron: Learning Algorithm

- More formally: $\mathbf{x} = (x_1, x_2, .., x_d)^T$, $\mathbf{w} = (w_1, w_2, .., w_d)^T$
- θ: Threshold of the output unit
- Unit output: $\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + ... + x_dx_d$
- Output class 1 if $\mathbf{w}^T\mathbf{x} - \theta \geq 0$
- To eliminate the explicit dependence on θ: Output class 1 if: $\mathbf{w}^T\mathbf{x} \geq 0$

Perceptron: Learning Algorithm

- We want to learn values of the weights so that the perceptron correctly discriminate elements of C_1 from elements of C_2.

- Given **x** in input, if **x** is classified correctly, weights are unchanged, otherwise:

$$w = \begin{cases} w + x & \text{if an element of class } C_1 \text{ was classified as in } C_2 \\ w - x & \text{if an element of class } C_2 \text{ was classified as in } C_1 \end{cases}$$

Perceptron: Learning Algorithm

- 1st case: $\mathbf{x} \in C_1$ and was classified in C_2. The correct answer is 1, which corresponds to: $\mathbf{w}^T \mathbf{x} \geq 0$, we have $\mathbf{w}^T \mathbf{x} < 0$. We want to get closer to the correct answer: $\mathbf{w}^T \mathbf{x} < \mathbf{w}'^T \mathbf{x}$.

$$\mathbf{w}^T \mathbf{x} < \mathbf{w}'^T \mathbf{x}, \text{ iff } \mathbf{w}^T \mathbf{x} < (\mathbf{w} + \ \mathbf{x})^T \mathbf{x} \quad (\mathbf{w} + \ \mathbf{x})^T \mathbf{x} =$$

$$\mathbf{w}^T \mathbf{x} + \ \mathbf{x}^T \mathbf{x} = \ \mathbf{w}^T \mathbf{x} + \ \|\mathbf{x}\|^2$$

because $\|\mathbf{x}\|^2 > 0$, the condition is verified.

- 2nd case: $\mathbf{x} \in C_2$ and was classified in C_1. The correct answer is 0, which corresponds to: $\mathbf{w}^T \mathbf{x} < 0$, we have $\mathbf{w}^T \mathbf{x} \geq 0$. We want to get closer to the correct answer: $\mathbf{w}^T \mathbf{x} > \mathbf{w}'^T \mathbf{x}$.

$$\mathbf{w}^T \mathbf{x} > \mathbf{w}'^T \mathbf{x}, \text{ iff } \mathbf{w}^T \mathbf{x} < (\mathbf{w} - \mathbf{x})^T \mathbf{x} \quad (\mathbf{w} - \mathbf{x})^T \mathbf{x} = \ \mathbf{w}^T$$

$$\mathbf{x} - \mathbf{x}^T \mathbf{x} = \ \mathbf{w}^T \mathbf{x} - \|\mathbf{x}\|^2$$

because $\|\mathbf{x}\|^2 > 0$, the condition is verified.

Perceptron: Learning Algorithm

In summary:

- A random sequence $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k$ is generated such that $x_i \in C_1 \cup C_2$
- If $\mathbf{x}_k$ is correctly classified, then $\mathbf{w}_{k+1} = \mathbf{w}_k$ otherwise:

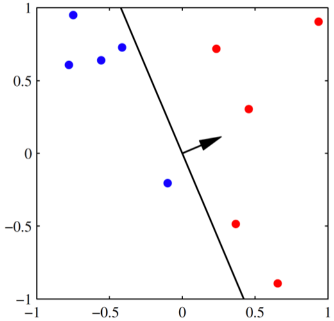
$$\mathbf{w_{k+1}} = \begin{cases} \boldsymbol{w_k} + \boldsymbol{x_k}\ if\ \boldsymbol{x_k} \in C_1 \\ \boldsymbol{w}_{\mathbf{k}} - \boldsymbol{x_k}\ if\ \boldsymbol{x_k} \in C_2 \end{cases}$$

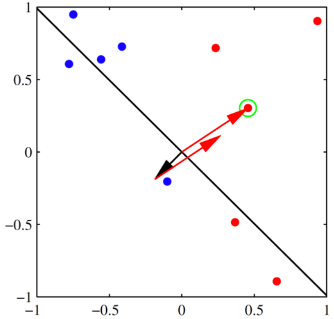
- Convergence theorem: regardless of the initial choice of weights, if the two classes are linearly separable, there exists $\mathbf{w}$ such that:

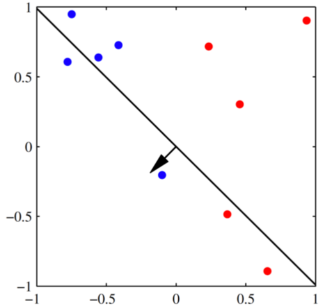
$$= \begin{cases} \boldsymbol{w^T x} \geq 0 \\ \boldsymbol{w^T x} < 0 \end{cases}$$

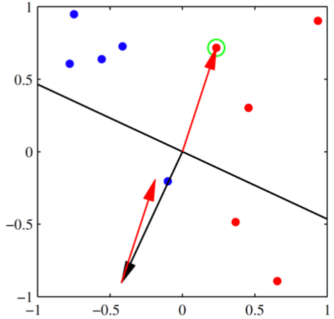
then the learning rule will find such solution after a finite number of steps.

Perceptron: Example









Naive Bayes: not (necessarily) a Bayesian method

- A and B are independent iff $p(A, B) = p(A)p(B)$
- A and B are conditionally independent given C iff $p(A, B|C) = p(A|C)p(B|C)$