L17: linear discriminant functions

Perceptron learning

Minimum squared error (MSE) solution

Least-mean squares (LMS) rule

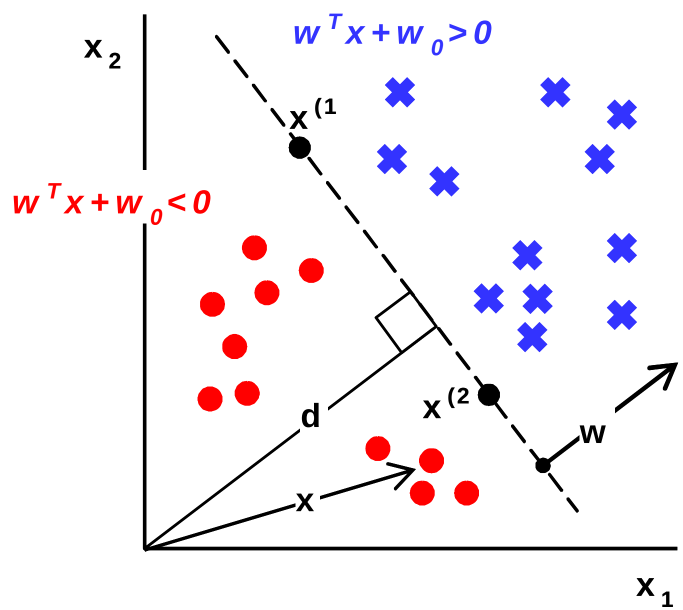
Ho-Kashyap procedure

Linear discriminant functions

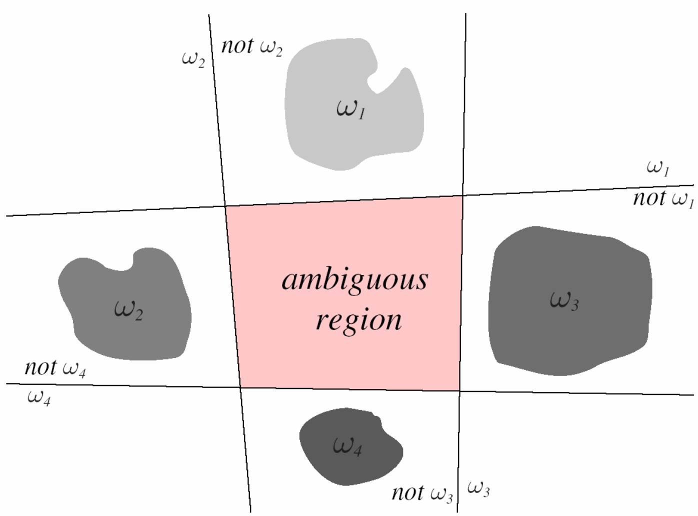
The objective of this lecture is to present methods for learning linear discriminant functions of the form

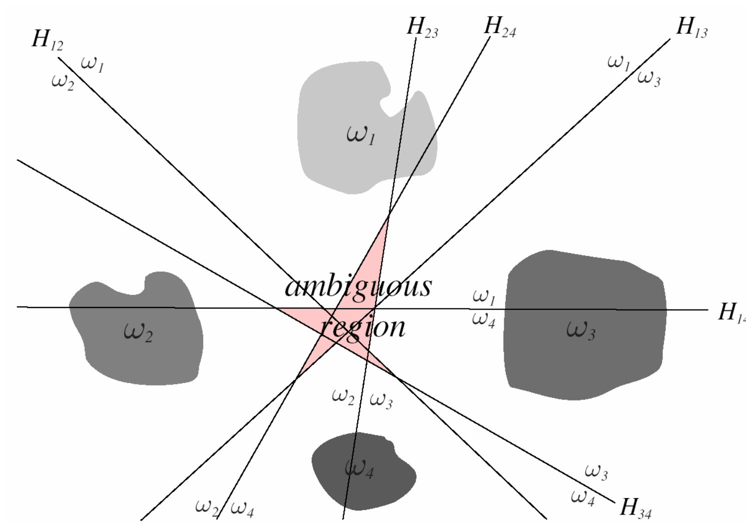
$$g(x) = w^T x + w_0 \Leftrightarrow \begin{cases} g(x) > 0 & x \in \omega_1 \\ g(x) < 0 & x \in \omega_2 \end{cases}$$

– where w is the weight vector and w_0 is the threshold weight or bias (not to be confused with that of the bias-variance dilemma)



- Similar discriminant functions were derived in L5 as a special case of the quadratic classifier
  - In this lecture, the discriminant functions will be derived in a non-parametric fashion, that is, no assumptions will be made about the underlying densities
- For convenience, we will focus on the binary classification problem
- Extension to the multi-category case can be easily achieved by
  - Using $\omega_i/\neg\omega_i$ dichotomies
  - Using ω_i/ω_j dichotomies





Gradient descent

GD is a general method for function minimization

– Recall that the minimum of a function $J(x)$ is defined by the zeros of the gradient

$$x^* = argmin_{\forall x}[J(x)] \Rightarrow \nabla_{\mathrm{x}}\mathrm{J}(x) = 0$$

- Only in special cases this minimization function has a closed form solution
- In some other cases, a closed form solution may exist, but is numerically ill-posed or impractical (e.g., memory requirements)

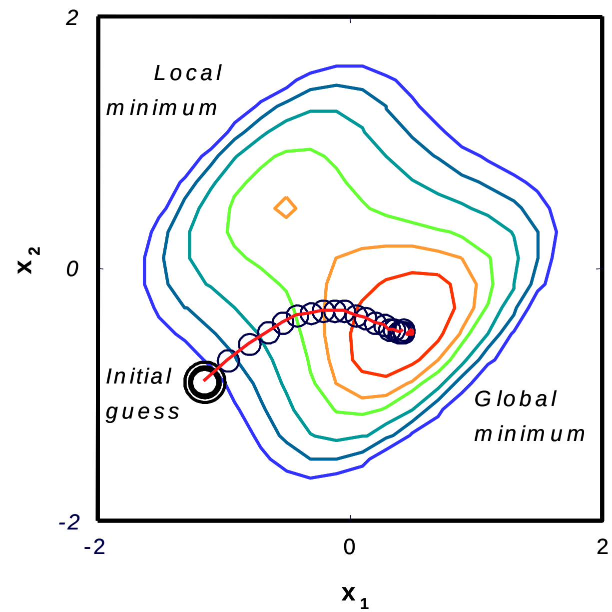
– Gradient descent finds the minimum in an iterative fashion by moving in the direction of steepest descent

1. Start with an arbitrary solution $x(0)$
2. Compute the gradient $\nabla_{\mathrm{x}}\mathrm{J}\big(x(k)\big)$
3. Move in the direction of steepest descent

$$x(k+1) = x(k) - \eta\nabla_{\mathrm{x}}\mathrm{J}\big(x(k)\big)$$

4. Go to 2 (until convergence)

- where η is a learning rate



Perceptron learning

Let's now consider the problem of learning a binary classification problem with a linear discriminant function

- As usual, assume we have a dataset $x = \{x^{(1}, x^{(2} \dots x^{(N}\}$ containing examples from the two classes
- For convenience, we will absorb the intercept w_0 by augmenting the feature vector x with an additional constant dimension

$$w^T x + w_0 = [w_0 \quad w^T] \begin{bmatrix} 1 \\ x \end{bmatrix} = a^T y$$

- Keep in mind that our objective is to find a vector a such that

$$g(x) = a^T y \begin{cases} > 0 & x \in \omega_1 \\ < 0 & x \in \omega_2 \end{cases}$$

- To simplify the derivation, we will "normalize" the training set by replacing all examples from class ω_2 by their negative

$$y \leftarrow [-y] \quad \forall y \in \omega_2$$

- This allows us to ignore class labels and look for vector a such that

$$a^T y > 0 \quad \forall y$$

[Duda, Hart and Stork, 2001]

To find a solution we must first define an objective function $J(a)$

- A good choice is what is known as the **Perceptron** criterion function

$$J_P(a) = \sum_{y \in Y_M} (-a^T y)$$

  - where Y_M is the set of examples misclassified by a
  - Note that $J_P(a)$ is non-negative since $a^T y < 0$ for all misclassified samples

To find the minimum of this function, we use gradient descent

- The gradient is defined by

$$\nabla_a J_P(a) = \sum_{y \in Y_M} (-y)$$

- And the gradient descent update rule becomes

$$a(k+1) = a(k) + \eta \sum_{y \in Y_M} y$$

Perceptron rule

- This is known as the **perceptron batch update rule**
  - The weight vector may also be updated in an "on-line" fashion, this is, after the presentation of each individual example

$$a(k+1) = a(k) + \eta y^{(i}$$

  - where $y^{(i}$ is an example that has been misclassified by $a(k)$

Perceptron learning

If classes are linearly separable, the perceptron rule is guaranteed to converge to a valid solution

- Some version of the perceptron rule use a variable learning rate $\eta(k)$
- In this case, convergence is guaranteed only under certain conditions (for details refer to [Duda, Hart and Stork, 2001], pp. 232-235)

However, if the two classes are not linearly separable, the perceptron rule will not converge

- Since no weight vector a can correctly classify every sample in a non-separable dataset, the corrections in the perceptron rule will never cease
- One ad-hoc solution to this problem is to enforce convergence by using variable learning rates $\eta(k)$ that approach zero as $k \to \infty$

Minimum Squared Error (MSE) solution

The classical MSE criterion provides an alternative to the perceptron rule

- The perceptron rule seeks a weight vector a^T such that $a^T y^{(i} > 0$
  - The perceptron rule only considers misclassified samples, since these are the only ones that violate the above inequality
- Instead, the MSE criterion looks for a solution to the equality $a^T y^{(i} = b^{(i}$, where $b^{(i}$ are some pre-specified target values (e.g., class labels)
  - As a result, the MSE solution uses ALL of the samples in the training set

[Duda, Hart and Stork, 2001]

The system of equations solved by MSE is

$$\begin{bmatrix} y_0^{(1)} & y_1^{(1)} & \cdots & y_D^{(1)} \\ y_0^{(2)} & y_1^{(2)} & \cdots & y_D^{(2)} \\ & & & \\ y_0^{(N)} & y_1^{(N)} & \cdots & y_D^{(N)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \\ a_D \end{bmatrix} = \begin{bmatrix} b^{(1)} \\ b^{(2)} \\ \\ \\ b^{(N)} \end{bmatrix} \Leftrightarrow Ya = b$$

- where a is the weight vector, each row in Y is a training example, and each row in b is the corresponding class label
  - For consistency, we will continue assuming that examples from class ω_2 have been replaced by their negative vector, although this is not a requirement for the MSE solution

An exact solution to $Ya = b$ can sometimes be found

- If the number of (independent) equations (N) is equal to the number of unknowns ($D+1$), the exact solution is defined by

$$a = Y^{-1}b$$

- In practice, however, Y will be singular so its inverse does not exist
  - Y will commonly have more rows (examples) than columns (unknowns), which yields an over-determined system, for which an exact solution cannot be found

The solution in this case is to minimizes some function of the error between the model (aY) and the desired output (b)

- In particular, MSE seeks to Minimize the sum Squared Error

$$J_{MSE}(a) = \sum_{i=1}^{N}\left(a^T y^{(i} - b^{(i}\right)^2 = \|Ya - b\|^2$$

- which, as usual, can be found by setting its gradient to zero

The pseudo-inverse solution

The gradient of the objective function is

$$\nabla_a J_{MSE}(a) = \sum_{i=1}^{N} 2\left(a^T y^{(i} - b^{(i)}\right) y^{(i} = 2Y^T(Ya - b) = 0$$

- with zeros defined by

$$Y^T Ya = Y^T b$$

- Notice that $Y^T Y$ is now a square matrix!

If $Y^T Y$ is nonsingular, the MSE solution becomes

$$a = (Y^T Y)^{-1} Y^T b = Y^\dagger b$$ **Pseudo-inverse solution**

- where $Y^\dagger = (Y^T Y)^{-1} Y^T$ is known as the pseudo-inverse of Y since $Y^\dagger Y = I$
  - Note that, in general, $YY^\dagger \neq I$

Ridge-regression solution

If the training data is collinear (extremely correlated), the matrix Y^TY becomes near singular

- As a result, the smaller eigenvalues (the noise) dominate the computation of the inverse $(Y^TY)^{-1}$, which results in numerical problems

The collinearity problem can be solved through regularization

- This is equivalent to adding a small multiple of the identity matrix to the term Y^TY, which results in

$$a = \left[(1-\epsilon)Y^TY + \epsilon\frac{tr(Y^TY)}{D}I\right]^{-1} Y^Tb$$

Ridge regression

- where ϵ $(0 < \epsilon < 1)$ is a regularization parameter that controls the amount of shrinkage to the identity matrix. This is known as the ridge-regression solution
  - If the features have significantly different variances, the regularization term may be replaced by a diagonal matrix of the feature variances

Selection of the regularization parameter

- For $\epsilon = 0$, ridge-regression solution is equivalent to the pseudo-inverse solution
- For $\epsilon = 1$, the ridge-regression solution is a constant function that predicts the average classification rate across the entire dataset
- An appropriate value for ϵ is typically found through cross-validation

[Gutierrez-Osuna, 2002]

Least-mean-squares solution

The objective function $J_{MSE}(a)$ can also be minimize using a gradient descent procedure

– This avoids the problems that arise when $Y^T Y$ is singular

– In addition, it also avoids the need for working with large matrices

Looking at the expression of the gradient, the obvious update rule is

$$a(k+1) = a(k) + \eta(k) Y^T \big(b - Y a(k)\big)$$

– It can be shown that if $\eta(k) = \eta(1)/k$, where $\eta(1)$ is any positive constant, this rule generates a sequence of vectors that converge to a solution to $Y^T(Ya - b) = 0$

– The storage requirements of this algorithm can be reduced by considering each sample sequentially

$$a(k+1) = a(k) + \eta(k)\big(b^{(i} - y^{(i} a(k)\big) y^{(i}$$

LMS rule

– This is known as the **Widrow-Hoff**, **least-mean-squares** (LMS) or **delta** rule [Mitchell, 1997]

[Duda, Hart and Stork, 2001]

Numerical example

Compute the perceptron and MSE solution for the dataset

- $X1 = [\,(1,6),(7,2),(8,9),(9,9)]$
- $X2 = [\,(2,1),(2,2),(2,4),(7,1)]$

Perceptron leaning

- Assume $\eta = 0.1$ and an online update rule
- Assume $a(0) = [0.1, 0.1, 0.1]$
- SOLUTION
  - Normalize the dataset

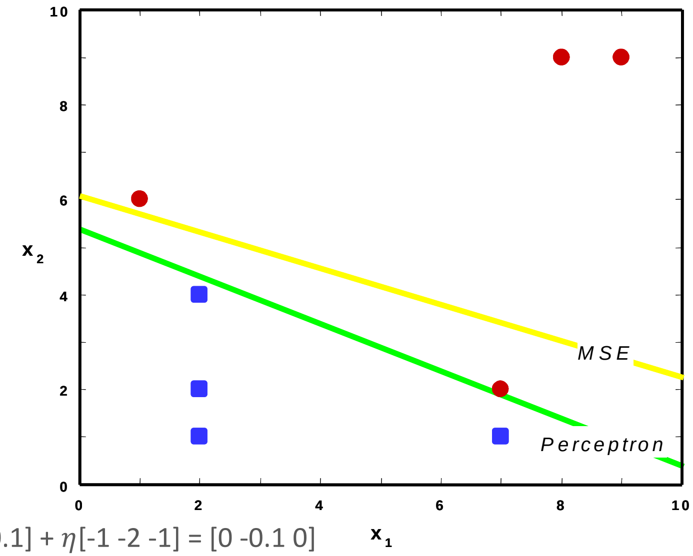
$$Y = \begin{bmatrix} 1 & 1 & 6 \\ 1 & 7 & 2 \\ 1 & 8 & 9 \\ 1 & 9 & 9 \\ -1 & -2 & -1 \\ -1 & -2 & -2 \\ -1 & -2 & -4 \\ -1 & -7 & -1 \end{bmatrix}$$

  - Iterate through all the examples and update $a(k)$ on the ones that are misclassified
    - Y(1) ⇒ [1 1 6]*[0.1 0.1 0.1]^T>0 ⇒ no update
    - Y(2) ⇒ [1 7 2]*[0.1 0.1 0.1]^T>0 ⇒ no update

      …
    - Y(5) ⇒ [-1 -2 -1]*[0.1 0.1 0.1]^T<0 ⇒ update a(1) = [0.1 0.1 0.1] + η[-1 -2 -1] = [0 -0.1 0]
    - Y(6) ⇒ [-1 -2 -2]*[0 -0.1 0]^T>0 ⇒ no update

      ….
    - Y(1) ⇒ [1 1 6]*[0 -0.1 0]^T<0 ⇒ update a(2) = [0 -0.1 0] + η[1 1 6] = [0.1 0 0.6]
    - Y(2) ⇒ [1 7 2]*[0.1 0 0.6]^T>0 ⇒ no update

      …



  - In this example, the perceptron rule converges after 175 iterations to $a = [-3.5\ 0.3\ 0.7]$
  - To convince yourself this is a solution, compute Ya (you will find out that all terms are non-negative)

MSE

- The MSE solution is found in one shot as $a = (Y^T Y)^{-1} Y^T b = [-1.1870\ \ 0.0746\ 0.1959]$
  - For the choice of targets b = [1 1 1 1 1 1 1 1]^T
  - As you can see in the figure, the MSE solution misclassifies one of the samples

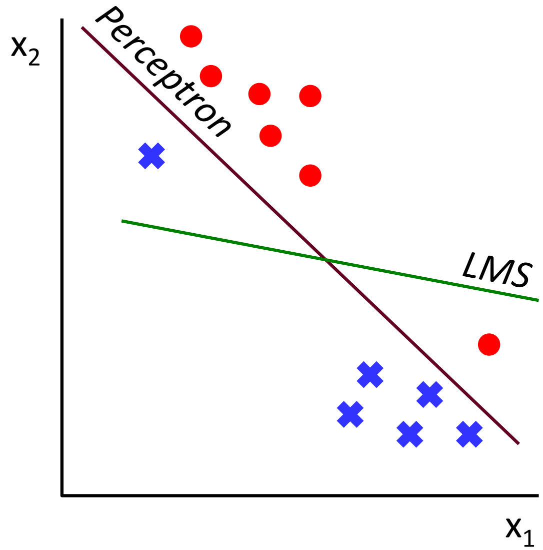
Summary: perceptron vs. MSE

Perceptron rule

– The perceptron rule always finds a solution if the classes are linearly separable, but does not converge if the classes are non-separable

MSE criterion

– The MSE solution has guaranteed convergence, but it may not find a separating hyperplane if classes are linearly separable
  - Notice that MSE tries to minimize the sum of the squares of the distances of the training data to the separating hyperplane, as opposed to finding this hyperplane



The Ho-Kashyap procedure

The main limitation of the MSE criterion is the lack of guarantees that a separating hyperplane will be found in the linearly separable case

- All we can say about the MSE rule is that it minimizes $\|Ya - b\|^2$
- Whether MSE finds a separating hyperplane or not depends on how properly the target outputs $b^{(i}$ are selected

Now, if the two classes are linearly separable, there must exist vectors a^* and b^* such that[1] $Ya^* = b^* > 0$

- If b^* were known, one could simply use the MSE solution ($a = Y^\dagger b$) to compute the separating hyperplane
- **However, since $\boldsymbol{b}^*$ is unknown, one must then solve for BOTH $\boldsymbol{a}$ and $\boldsymbol{b}$**

This idea gives rise to an alternative training algorithm for linear discriminant functions known as the Ho-Kashyap procedure

1) Find the target values b through gradient descent
2) Compute the weight vector a from the MSE solution
3) Repeat 1) and 2) until convergence

[1] Here we also assume $y \leftarrow [-y]\ \forall y \in \omega_2$)

Solution

- The gradient $\nabla_b J$ is defined by

$$\nabla_b J_{MSE}(a, b) = -2(Ya - b)$$

  - which suggest a possible update rule for b
- Now, since b is subject to the constraint $b > 0$, we are not free to follow whichever direction the gradient may point to
  - The solution is to start with an initial solution $b > 0$, and refuse to reduce any of its components
  - This is accomplished by setting to zero all the positive components of $\nabla_b J$

$$(\nabla_b J)^- = \frac{1}{2}[\nabla_b J - |\nabla_b J|]$$

  - Once b is updated, the MSE solution $a = Y^\dagger b$ provides the zeros of $\nabla_a J$
- The resulting iterative procedure is

$$b(k+1) = b(k) - \eta \frac{1}{2}[\nabla_b J - |\nabla_b J|]$$
$$a(k+1) = Y^\dagger b(k+1)$$

Ho-Kashyap procedure