Week 6 - Regression, Gradient Descent, Clustering and K-means

Sequential Gradient Descent (SGD)

- Follow the slope (the derivative of a f(x)) to find the best w to minimize the loss

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

  - Linear function $f(x) = wx + b$, slope =

D

  - Quadratic function $f(x) = x^2$, slope = delta f over delta x

$$\frac{\triangle f}{\triangle x}$$

  - Derivative

$$d(x^n) = nx^{n-1}$$

- **How to adjust the model parameters?**
  - Derivative is **zero** at **minima** and **maxima** of a function, $f'(x) = 0$, which is used as way of finding optimal values of objectives.
  - In multiple dimensions, the **gradient** $f_w(w)$ (slope of the **tangent plane**) is a **vector of derivatives** in each dimension
    - $f(w_1, w_2) = w_1^2 + w_2^2$

- **Matrix notation:** $w = \begin{bmatrix} w1 \\ w2 \end{bmatrix} = [w1\ w2]^T$**,** $f_w(w) = \nabla f(w) = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix}$ **(** $\nabla f(w)$ **pronounced as "nabla f" or "grad f")**
- **The gradient gives a direction to which the function increases**

- Explanation
  - General algorithm for finding a value of the model parameters *w* such that error function $F(w)$ is minimized, expressed using **multivariable calculus** as $F_w(w) = 0$ where *Fw* is the (**partial**) **derivative** of *F* with respect to vector *w. (*Note that this is actually a *vector* of D zeros, but we use the usual zero as a shorthand.)

$$F_w(w) = \begin{bmatrix} 0 \\ \cdots \\ 0 \end{bmatrix}$$

  - Idea: starting with a guess for w_n, take a "step" in direction of **steepest descent** of the loss function, $-F_w(w_n)$, use this as a better guess w_{n+1}
  - Size of the step, $\alpha > 0$ ("learning rate"), determines how quickly the minimum is reached, but can **overshoot** and also **diverge (发散)** if *α* is too large; not guaranteed to find the minimum, unless the error function is **convex (凸函数)** with respect to *w*
  - If a loss function has one global minimum, then it is a **convex** function.
  - More advanced versions are widely used in modern deep learning

- **SGD: algorithm**
  - **Step 1**. *Initialization*: Select an initial guess for *w*0, a convergence tolerance
    ε > 0, step size (learning rate) parameter *α* > 0, set
    iteration number
    n=0

- **Step 2**. *Gradient descent step*: Compute new model parameters, $w_{n+1} = w_n - \alpha F_w(w_n)$
- **Step 3**. *Convergence test*: Compute new loss function value $F(w_{n+1})$, and loss function improvement, $\triangle F = |F(w_{n+1}) - F(w_n)|$ and if $\triangle F < \varepsilon$, exit with solution $w^* = w_{n+1}$ (when the difference between current Loss and last Lost ≤ tolerance, it converges.)
- **Step 4**. *Iteration*: update *n*=*n*+1 and go to step 2.

```
def sgd():
        '''sequential gradient descent'''
        # step 1
        select an initial guess for w0
        select convergence tolerance ε > 0
        select αlpha > 0

        for n in range(R):
                # step 2
                w_{n+1} = w_n − α * F_w(w_n)

                # step 3
                ΔF = abs(F(w_{n+1}) - F(w_n))
                if ΔF < ε:
                        return w^* = w_{n+1}

                # step 4: go back to step 2
```

- Euclidean distance

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

Supervised Learning

Regression

1. Regression model (D-dimensional):

$$(wtransposex)$$

2. For linear regression, we transform $f(x) = wx + b$ to $f(x) = w_1x^1 + w_2x^2$ where w_2x^2 is b. Make it easier to compute the sum of square error **(we represent it as the latter, because transform can mean linear transformation) (w2 is b, x2 is set to 1)**
3. sum of squares error function:

4. Problem: find the optimal set of w

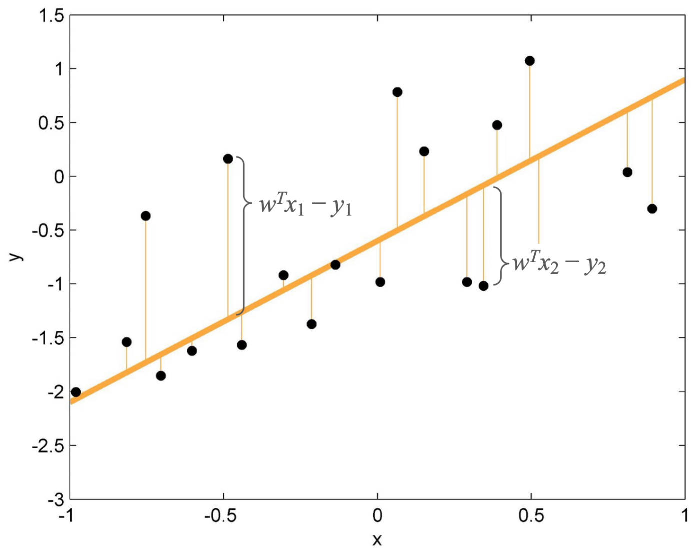
$$w^* = \arg\min_{w' \in W} F(w')$$

5. Gradient with respect to w:

   (multiply the result by 2 to account for the derivative of the squared term itself which is 2)
6. SGD parameter update step

7. Sum of Squared Error (SSE)

$$SSE = \sum_{i=1}^{k} \sum_{j=1}^{n_i} || \ x_{ij} - \mu_i \ ||^2$$



- Large error (bad fit):

$$F(w) = \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

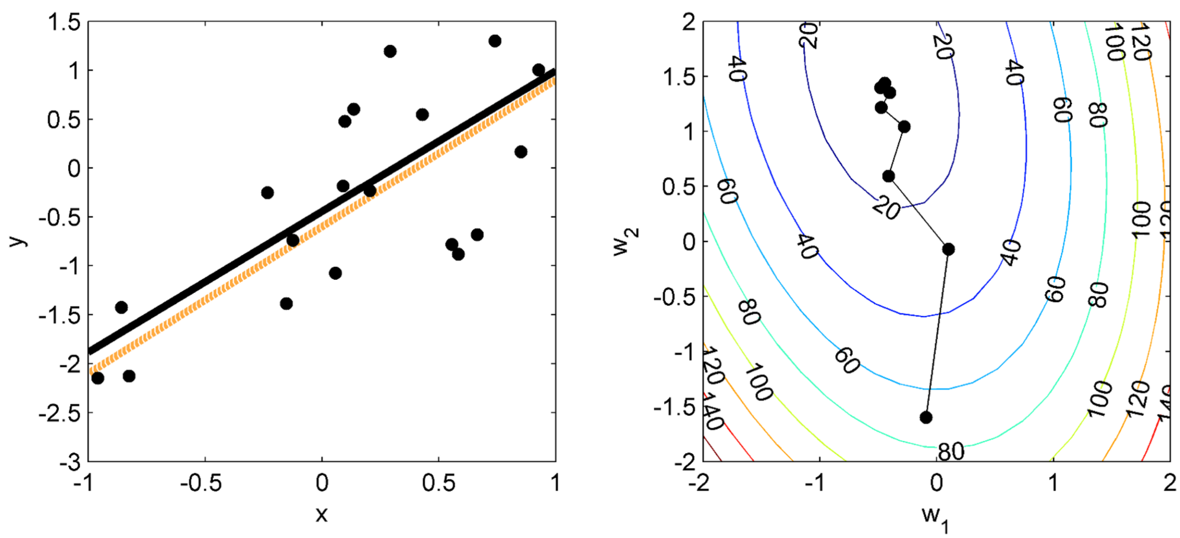
$$F(w) = 11.65$$

- Error is sum of squares of perpendicular distances to best fit line

perpendicular distances

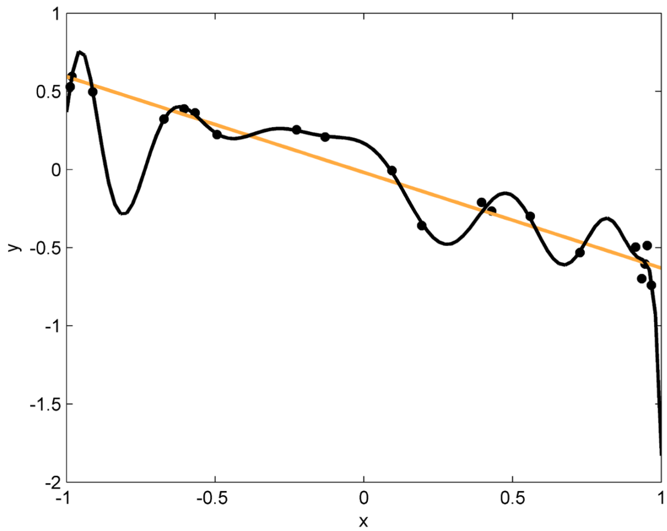
Linear regression: SGD in action

n=7, $F(w)$=11.69, w_7=[-0.44,1.44], ΔF=0.03 < ε = 0.05 (exit), w^*=[-0.44,1.44]



1. **A trade off between** **model complexity** **and** **test error**: common feature of most machine learning models: want a model which is as simple as possible but no simpler **(Occam's razor)**

Model complexity and Occam's razor



— $F(w^*) = 0.04$ (order 15 polynomial)

— $F(w^*) = 0.17$ (linear)

Unsupervised Learning

Clustering: K-means (minimizes the distance between all data points and their assigned clusters)

1. Given N data points in D-dimensional data space R^D, clustering finds the optimal way to **partition** the set of data into K groups
2. Conditions
   a. High intra-cluster similarity 类内相似度高
   b. Low inter-cluster similarity 类间相似度低
3. For a fixed value of clusters, *K*,
   a. Assign a cluster to each data point, e.g.,

$$x_1 = \begin{array}{c} \begin{matrix} k_1 & k_2 \end{matrix} \\ \begin{bmatrix} 0 & 1 \end{bmatrix} \end{array}$$

   b. Compute the average value of each feature for the cluster (**indicator function** notation X_{ik})

$$X_{ik} = \begin{cases} 1 & \text{data item } x_i \text{ is assigned to cluster } k \\ 0 & \text{otherwise} \end{cases}. \tag{10.1}$$

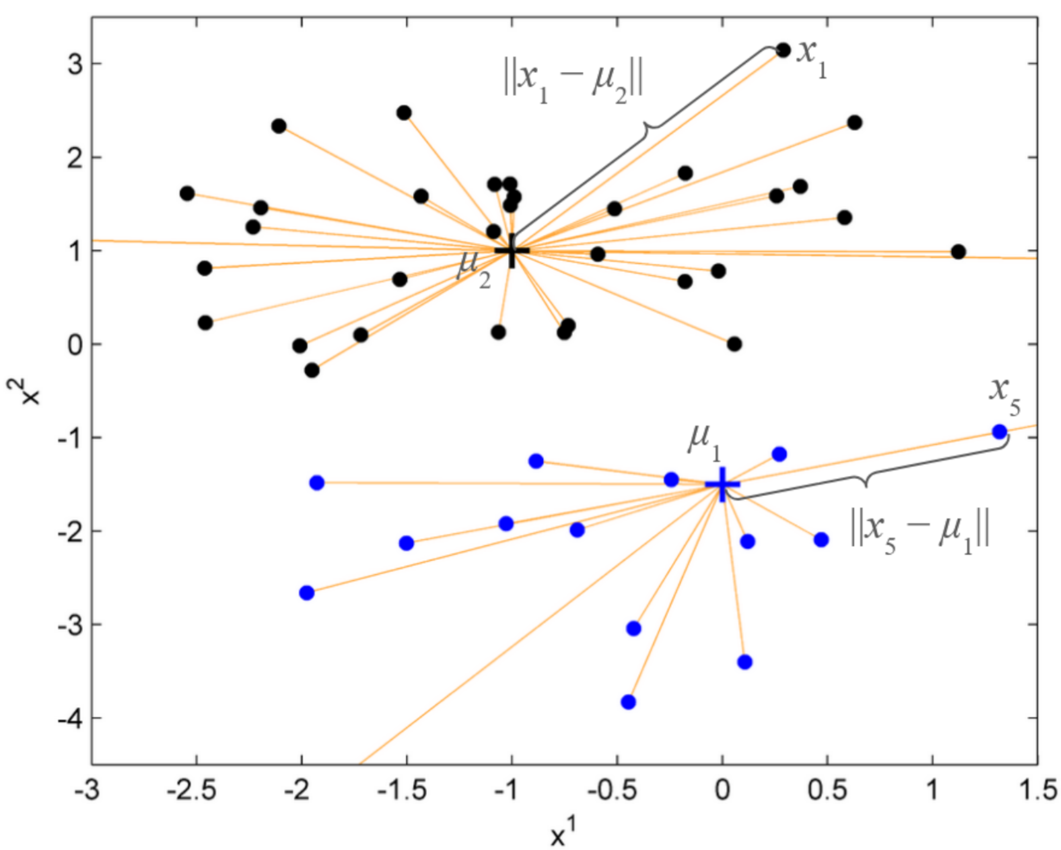
As an example, if we have data $x_1, \ldots, x_5$ and $K = 3$ classes, then the configuration,

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{10.2}$$

$$\mu_1 = \frac{1}{N_k} \sum_{i=1}^{N} X_{ik} \cdot x_i$$

c. Compute the Euclidean distance between each point in the cluster and its average

$$Euclidean\ Distance = ||x_i - \mu_k||$$



d. **K-means objective**, a widely-used measure of clustering quality.
The sum of squared distances between all data points and their assigned clusters in the set partition X

$$F(X,\mu) = \sum_{i=1}^{N}\sum_{k=1}^{K} X_{ik} \cdot ||x_i - \mu_i||^2$$

$$||v||^2 = \sum_{d=1}^{D}(v^d)^2$$

e. K-means clustering attempts to solve the following optimization problem

$$(X^*, \mu^*) = \underset{(X', \mu') \in W}{\arg\min}\ F(X', \mu')$$

where W is the set of all possible partitions (i.e. just those indicators where each data item is assigned to a unique class), along with their corresponding centroid averages.

4. Algorithm
   a. Step 1. Initialization: Select an initial guess for all μ_k^0 for *k*=1, 2 ,..., *K*, set iteration number *n*=0
   b. Step 2. Update configuration: Set X^{n+1}=0, except where

$$k = \underset{k'=1,2,\ldots,K}{\arg\min}\ ||x_i - \mu_{k'}||^2$$

for which

$$X_{ik}^{n+1} = 1$$

   c. Step 3. Update centroids μ: Compute cluster averages,

$$\mu_k^{n+1} = \frac{1}{N_k}\sum_{i=1}^{N} X_{ik}^{n+1} x_i$$

where

$$N_k \sum_{i=1}^{N} X_{ik}^{n+1}$$

d. Step 4. Convergence check: If *n*>0 and $X^{n+1} = X^n$ then exit with solution $X^* = X^{n+1}$ and $\mu^* = \mu^{n+}1$

e. Step 5. Iteration: update *n*=*n*+1 and go back to step 2.

```
# step 1
initial guess for all μ with a cluster k

for n in range(R):
        # step 2 Update configuration

        # step 3 Update centroids

        # step 4 Convergence check
        if X^{n+1} = X^n:
                return X^* = X^{n+1}, μ^* = μ^{n+1}
```

f. K-means clustering: analysis

   i. $F(X_{n+1}, \mu_{n+1}) \leq F(X_n, \mu_n)$ i.e. the **K-means objective function** is never increasing

5. Applications

   a. Compress gray scale images (image dictionary-based methods)

6. Compute the value of the centroid as the average of the x value and the average of the y value.

$$centroid = (x_{avg}, y_{avg})$$

7. Euclidean distance (distance from the centroid = sum of squared difference for each dimension)

$$(x_{currentPoint} - x_{centroid})^2 + (y_{currentPoint} - y_{centroid})^2$$

- Support session