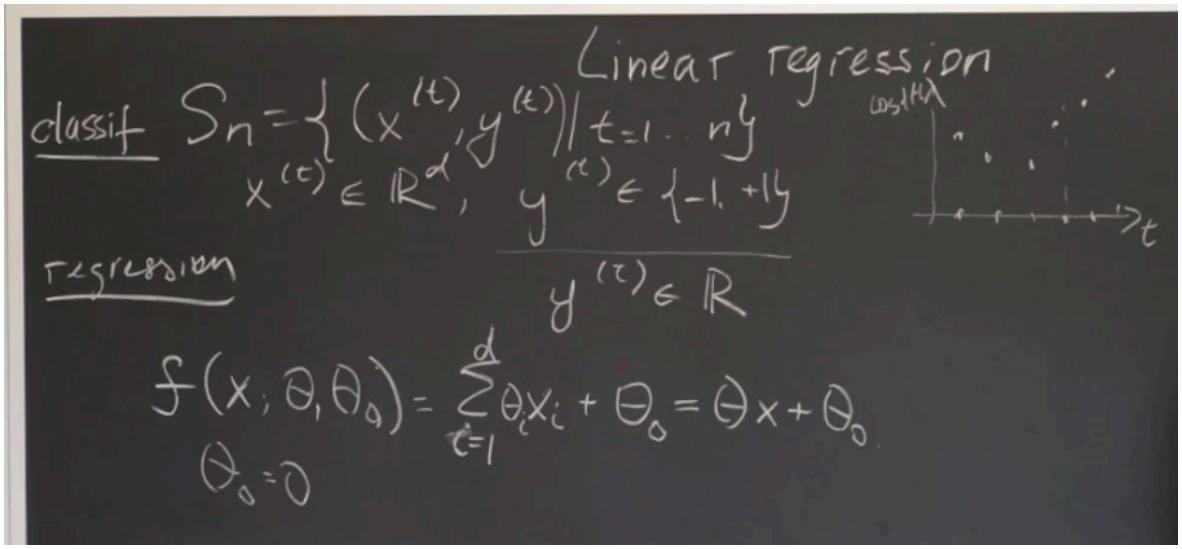
Unit 2 Nonlinear Classification, Linear regression, Collaborative Filtering

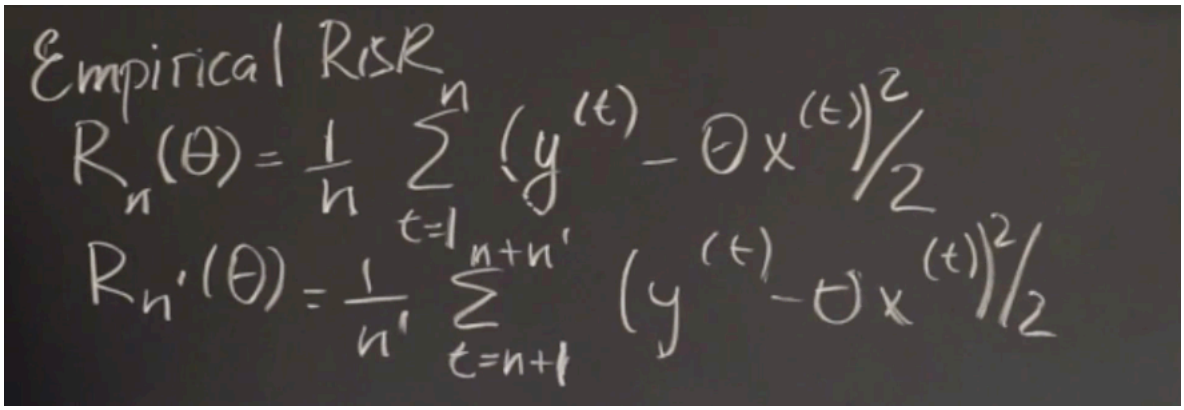
Lecture 5. Linear Regression

1. Intro



- Objective
- Learning algorithm : gradient based (approximate) or closed form (not approximate)
- Regularization (make fitting more robust in the context of linear regression)

2. The objective: Empirical Risk



- Structural error: linear model cannot fit (nonlinear)
- Estimation error: linear but we cannot estimate correctly because of limited data
- If we are given a large amount of training data and successfully obtain 0 empirical risk, The model we learned is not guaranteed to perform well on the test data and is very likely to be overfitted.

- Error Decomposition

$$y = f(x) + \epsilon$$

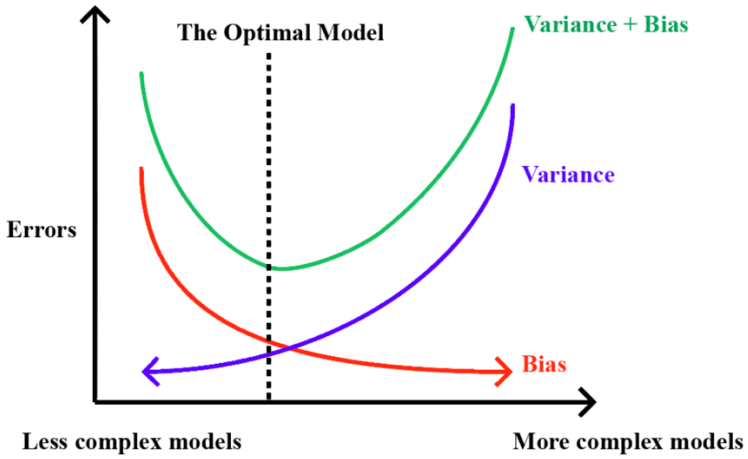
$$\text{where } \epsilon \sim \mathcal{N}(0, \sigma^2)$$

$$\mathbb{E}[(y - \hat{f}(x))^2] = \mathbb{E}[(f(x) + \epsilon - \hat{f}(x))^2]$$

$$= (f(x) - \mathbb{E}[\hat{f}(x)])^2 + \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] + \mathbb{E}[\epsilon^2]$$

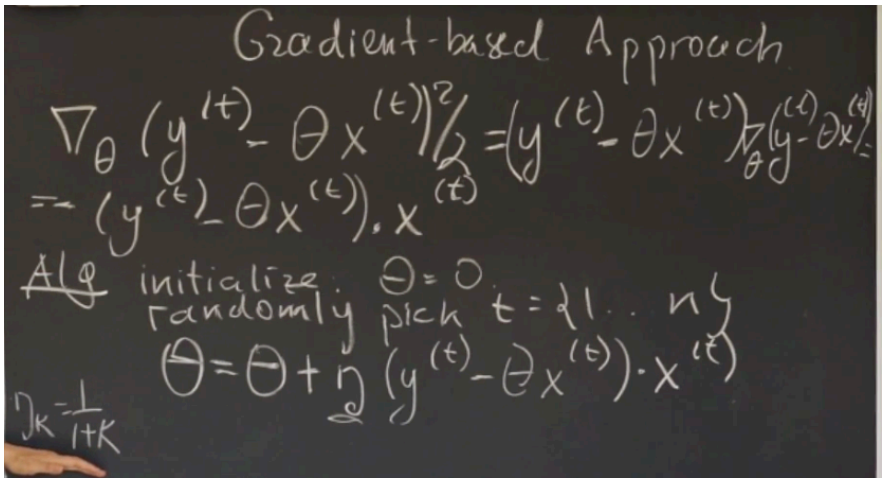
- The first term is bias, the second term is variance of the estimator, the third term is the error from the inherent noise of the data

- Bias-variance trade off



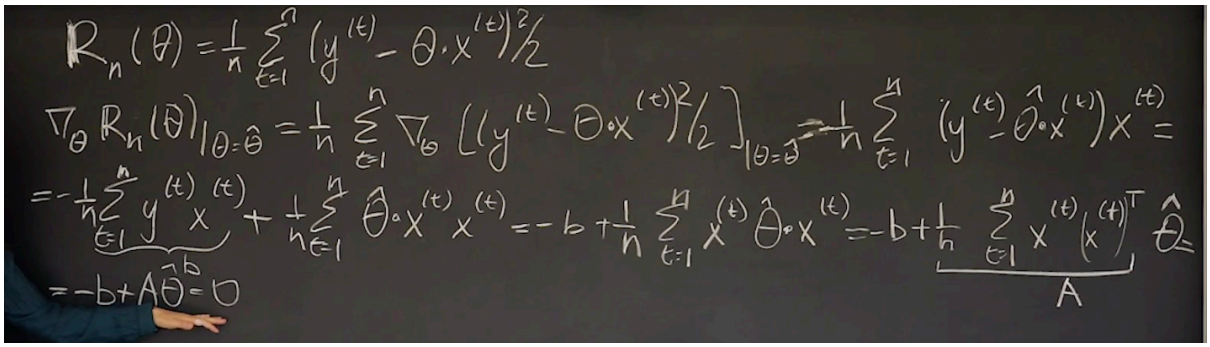
3. Learning Algorithm - Gradient Base Approach

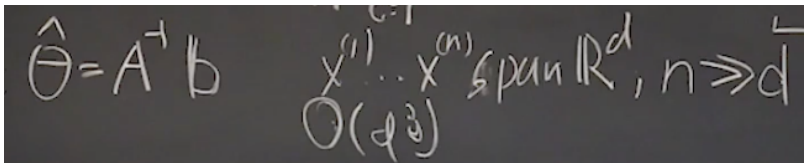
- Gradient Descent



- Stochastic gradient descent is faster than gradient descent

4. Closed Form Solution



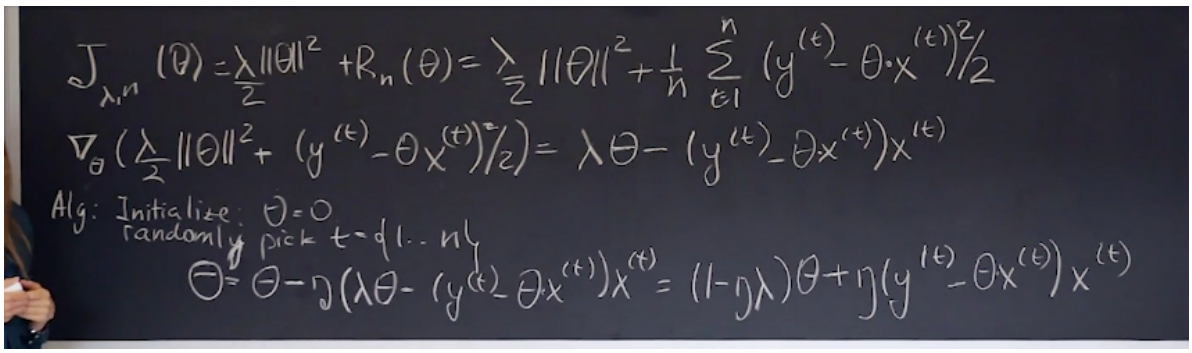


- If A is invertible

5. Motivation for regularization

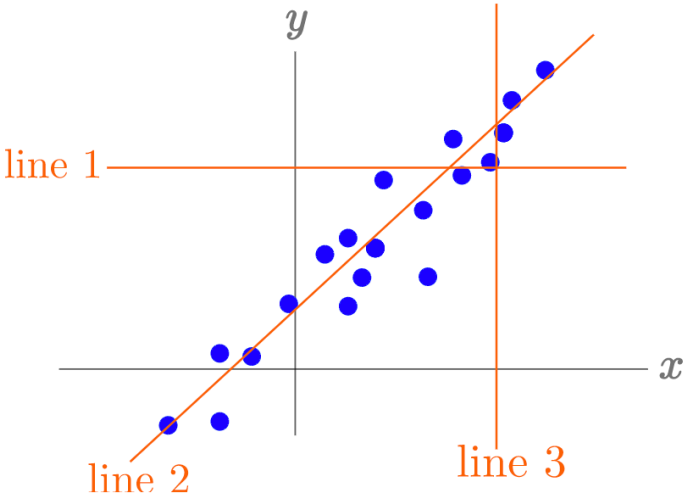
- Regularization will push you away from trying to perfectly fit your training example.
- In non-regularized cases, when we are trying to fit a model, we are trying our best to find the theta to fit training data, which means we also fit all the mistakes that are in training or some error that are in training and other things (randomness?)
- What regularization will do is that it would actually push all the centers to be close to zero.

6. Ridge Regression



- $$J_{n,\lambda}(\theta, \theta_0) = \frac{1}{n}\sum_{t=1}^{n}\frac{(y^{(t)} - \theta \cdot x^{(t)} - \theta_0)^2}{2} + \frac{\lambda}{2}\|\theta\|^2$$

  Lambda is the regularization factor



F(x) converges to line 1 if lambda goes to infinity (equivalent to minimizing norm(theta)^2)
converges to line 2 if lambda goes to 0 (equivalent to minimizing loss function or cost function)

- Including lambda makes training error bigger, but we are hoping that this generalization would **make test error smaller.**

7. Equivalence of regularization to a Gaussian Prior on weights

- The regularized linear regression can be interpreted from a probabilistic point of view. Suppose we are fitting a linear regression model with n data points $(x^1,y^1),(x^2,y^2),...,(xn,yn)$. Let's assume the ground truth is that y is linearly related to x but we also observed some noise ϵ for y:

$$yt=\theta\cdot xt+\epsilon$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Then the likelihood of our observed data is

$$\prod_{t=1}^{n} \mathcal{N}(y_t|\theta x_t, \sigma^2).$$

Now, if we impose a Gaussian prior $\mathcal{N}(\theta|0, \lambda^{-1})$, the likelihood will change to

$$\prod_{t=1}^{n} \mathcal{N}(y_t|\theta x_t, \sigma^2)\,\mathcal{N}(\theta|0, \lambda^{-1}).$$

Take the logarithim of the likelihood, we will end up with

$$\sum_{t=1}^{n} -\frac{1}{2\sigma^2}(y_t - \theta x_t)^2 - \frac{1}{2}\lambda\|\theta\|^2 + \text{constant}.$$

Lecture 6. Non Linear classification

1. Outline

- Non-linear classification and regression
- Feature maps, their inner products
- Kernel functions induced from feature maps
- Kernel methods, kernel perceptron
- Other non-linear classifiers (e.g., Random Forest)

1. Higher Order Feature Vectors

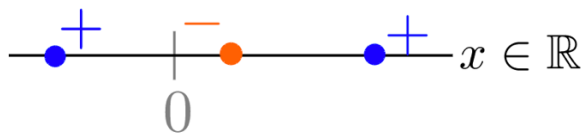
- Map datapoints to higher dimension

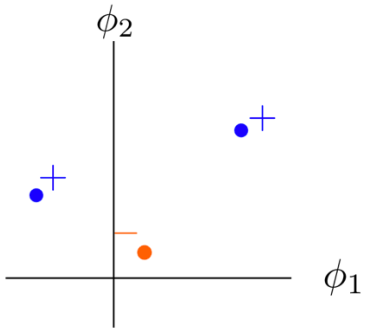
We can use linear classifiers to make non-linear predictions. The easiest way to do this is to first map all the examples $x \in \mathbb{R}d$ to different feature vectors $\phi(x) \in \mathbb{R}p$ where typically p is much larger than d. We would then simply use a linear classifier on the new (higher dimensional) feature vectors, pretending that they were the original input vectors. As a result, all the linear classifiers we have learned remain applicable, yet produce non-linear classifiers in the original coordinates.

- Example:

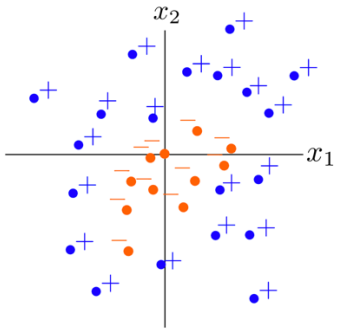
$$\phi(x) = [\phi_1(x), \phi_2(x)]^T = \left[x, x^2\right]^T,$$

Maps points in 1 dimension space to corresponding feature vector in a2 dimension space





Ex.



Since a possible boundary is an elipse, and we recall from geometry that the equation of any ellipse can be given as

$$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \theta_0 = \theta \cdot [x_1,\ x_2,\ x_1 x_2,\ x_1^2,\ x_2^2]^T + \theta_0 = 0,$$

for some $\theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]$, we see that defining the feature map to be $\phi(x) = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$ (the last choice) will allow us to write a linear decision boundary in the ϕ-coordinates:

$$\theta \cdot \phi(x) + \theta_0 = 0 \iff \theta \cdot [x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T + \theta_0 = 0.$$

2. Intro to Nonlinear classification

- Polynomial features

‣ We can add more polynomial terms

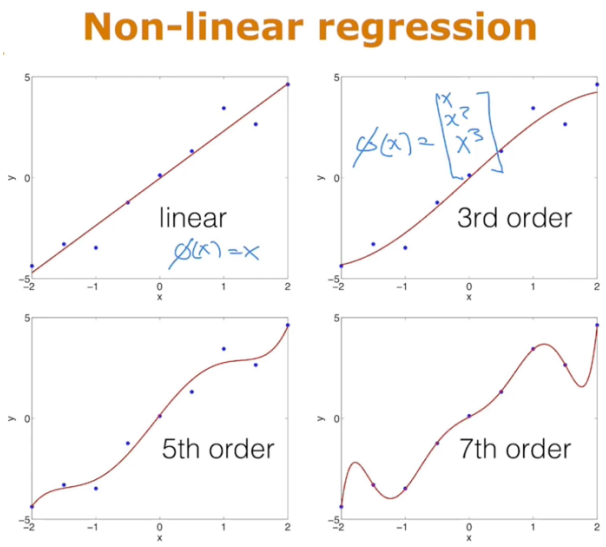
$x \in \mathbb{R}$, $\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \\ x^4 \\ \vdots \end{bmatrix}$

‣ Means lots of features in higher dimensions

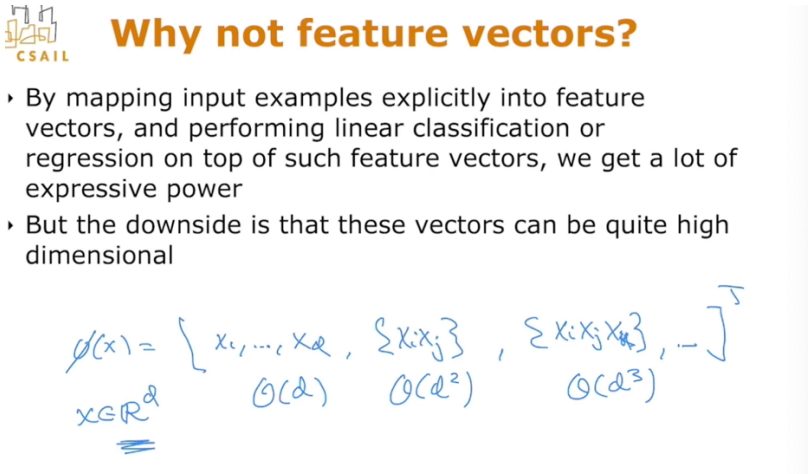
$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$ $\quad \phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix} \in \mathbb{R}^5$

- Non-linear regression

MIT 6.86x Note



- Which one to choose? We can use validation / cross validation to compare models
- Feature vector



Ex.

$$\phi(x) = \left[\underbrace{x_1, \ldots, x_i, \ldots, x_{150}}_{\text{deg 1}}, \underbrace{x_1^2, x_1x_2, \ldots, x_ix_j, \ldots x_{150}^2}_{\text{deg 2}}, \underbrace{x_1^3, x_1^2x_2, \ldots, x_ix_jx_k, \ldots, x_{150}^3}_{\text{deg 3}}\right]$$

$\phi(x) \in \mathbb{R}^q$ for what q?

For each of the feature transformations (power 1, power 2, power 3), there are n-multichoose-power combinations. Thus $\binom{150}{1} + \binom{151}{2} + \binom{152}{3} = 585275$. **Remark:** We see that the dimension of the space that the feature vectors live grows quickly as a function of d, the dimension we started with if $x \in \mathbb{R}^d$.

- Thus the high dimension will require high computation power which costs a lot

3. **Motivation for Kernels: Computation Efficiency**

Inner products, kernels

- Computing the inner product between two feature vectors **can be** cheap even if the vectors are very high dimensional

$$\phi(x) = [x_1, x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2]^T \ldots$$
$$\phi(x') = [x'_1, x'_2, x'^2_1, \sqrt{2}x'_1x'_2, x'^2_2]^T \ldots$$

$K(x,x') = \phi(x) \cdot \phi(x') = (x \cdot x') + (x \cdot x')^2 + (x \cdot x')^3$

Kernel f.

Kernels vs features

- For **some** feature maps, we can evaluate the inner products very efficiently, e.g.,

$K(x,x') = \phi(x) \cdot \phi(x') = (1 + x \cdot x')^p$

$p = 1, 2, \ldots$

- In those cases, it is advantageous to express the linear classifiers (regression methods) in terms of kernels rather than explicitly constructing feature vectors

$\text{sign}(\theta \cdot \phi(x) + \theta_0) \longrightarrow K(x,x')$

4. Kernel Perceptron Algorithm - updating alpha

$$\theta = \sum_{j=1}^{n} \alpha_j y^{(j)} \phi\left(x^{(j)}\right)$$

Kernel Perceptron $\left(\left\{ \left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n, T \right\}\right)$

Initialize $\alpha_1, \alpha_2, \ldots, \alpha_n$ to some values;
for $t = 1, \ldots, T$
  for $i = 1, \ldots, n$
    if $\left(\text{Mistake Condition Expressed in } \alpha_j\right)$
      Update α_j appropriately

5. Kernel Composition Rules

Feature engineering, kernels

- Composition rules:
  1. $K(x, x') = 1$ is a kernel function. $\phi(x) = 1$
  2. Let $f : \mathbb{R}^d \to \mathbb{R}$ and $K(x, x')$ is a kernel. Then so is $\tilde{K}(x, x') = f(x) K(x, x') f(x')$ $\phi(x) = f(x)\phi(x)$
  3. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then $K(x, x') = K_1(x, x') + K_2(x, x')$ is a kernel $\phi(x) = \begin{bmatrix} \phi^{(1)}(x) \\ \phi^{(2)}(x) \end{bmatrix}$
  4. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then $K(x, x') = K_1(x, x') K_2(x, x')$ is a kernel

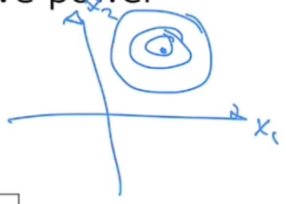
$(x \cdot x') + (x \cdot x')^2$

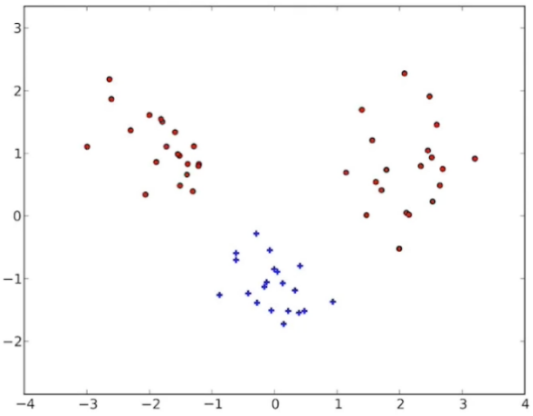
$\phi(x) = x$

6. The Radial Basis Kernel

- The feature vectors can be infinite dimensional... this means that they have unlimited expressive power

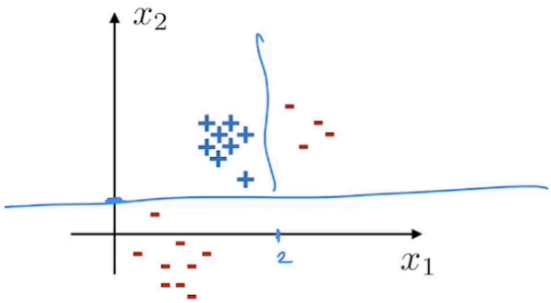
$$K(x, x') = \exp(-\frac{1}{2}\|x - x'\|^2)$$

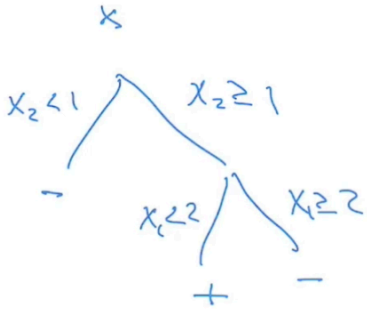




Other non-linear classifiers

- Random forest is a good default classifier for (almost) any setting



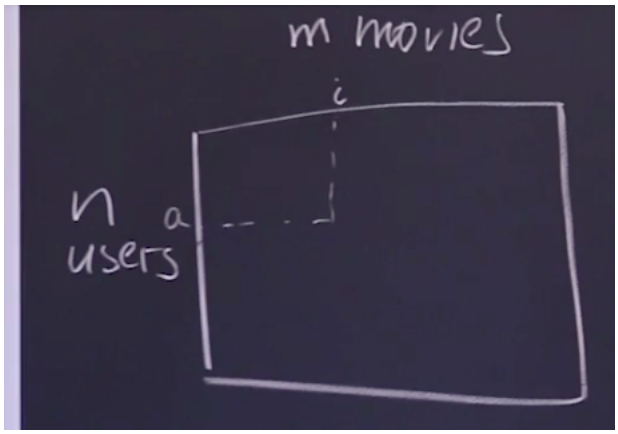


- Procedure:
  - boostrap sample
  - build a (randomized) decision tree
  - average predictions (ensemble)

Lecture 7. Recommender Systems

MIT 6.86x Note

1. **Intro**



- Usually, users have not ranked enough movies to predict users' future movie rankings (so we cannot use regression)
- We know what features are important. However we do not know.

2. **K Nearest Neighborhood**

K-Nearest Neighbour

The K-Nearest Neighbor method makes use of ratings by K other "similar" users when predicting Y_{ai}.

Let $\mathrm{KNN}(a)$ be the set of K users "similar to" user a, and let $\mathrm{sim}(a, b)$ be a **similarity measure** between users a and $b \in \mathrm{KNN}(a)$. The K-Nearest Neighbor method predicts a ranking Y_{ai} to be :

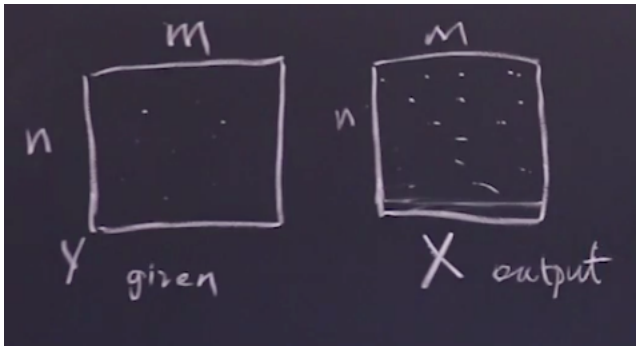
$$\widehat{Y}_{ai} = \frac{\sum_{b \in \mathrm{KNN}(a)} \mathrm{sim}(a, b)\, Y_{bi}}{\sum_{b \in \mathrm{KNN}(a)} \mathrm{sim}(a, b)}.$$

The similarity measure $\mathrm{sim}(a, b)$ could be any distance function between the feature vectors x_a and x_b of users a and b, e.g. the euclidean distance $\|x_a - x_b\|$ and the cosine similarity $\cos\theta = \frac{x_a \cdot x_b}{\|x_a\|\,\|x_b\|}$. We will use these similarity measures again in *Unit 4 Unsupervised Learning*.

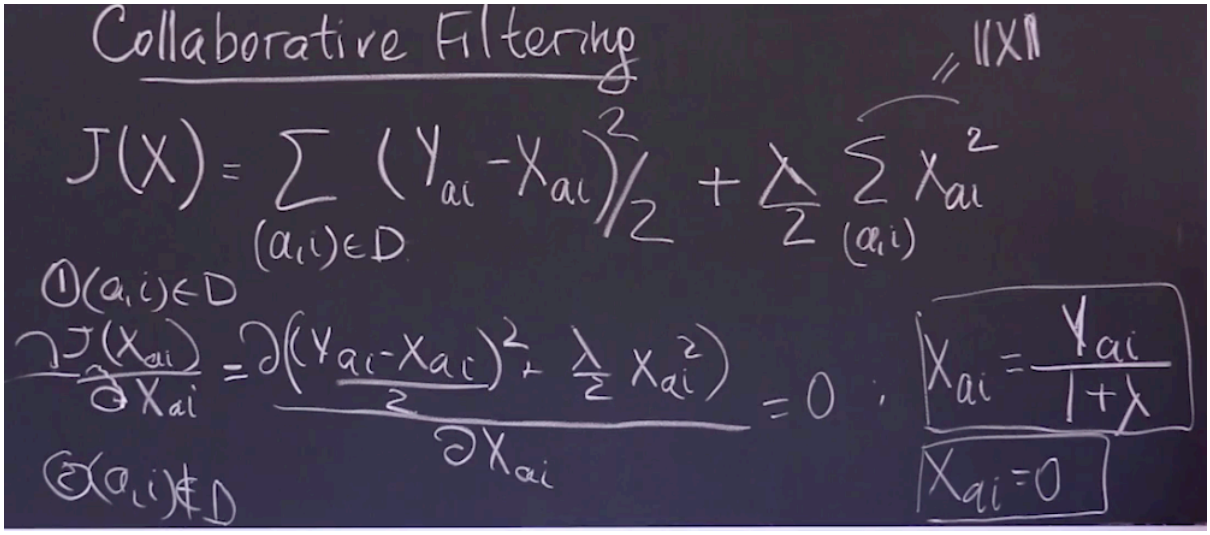
A drawback of this method is that the success of the K-Nearest Neighbor method depends heavily on the choice of the similarity measure. In the next section, we will discuss collaborative filtering, which will free us from the need to define a good similarity measure.

- This method does not enable us to detect the hidden structure in data. (one may be similar to another in a dimension, but different from another in another dimension.

3. **Collaborative Filtering: the naïve approach**



We are given very few data points but want many data points as output

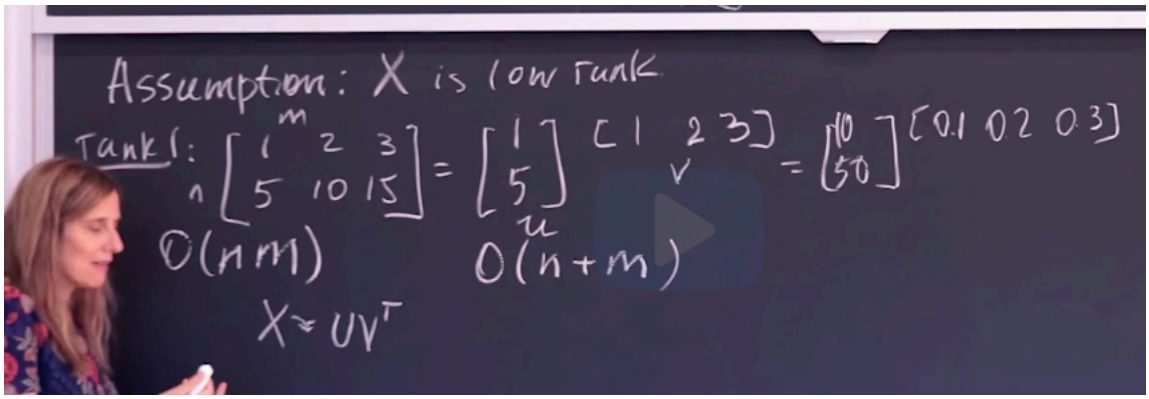


The is our objective function. The first part is about minimizing the difference between predicted output and the labels given. The second part is regularization.

- The solution here is **worse** than the original data we got.

4. Collaborative Filtering with **Matrix Factorization**

Back to linear algebra, rank captures how much dependency do we see between entries of a matrix.



In the collaborative filtering approach, we impose an additional constraint on X:

$$X = UV^T$$

for some $n \times d$ matrix U and $d \times m$ matrix V^T. The number d is the **rank** of the matrix X.

Assume we have a 3 by 2 matrix X i.e. we have 3 users and 2 movies. Also, X is given by

$$X = \begin{bmatrix} \text{User 1's rating on movie 1} & \text{User 1's rating on movie 2} \\ \text{User 2's rating on movie 1} & \text{User 2's rating on movie 2} \\ \text{User 3's rating on movie 1} & \text{User 3's rating on movie 2} \end{bmatrix} = UV^T$$

for some $3 \times d$ matrix U and $d \times 2$ matrix V^T.

The first row of U represents information on user 1's rating tendency.
The first column of V^T represents information on movie 1

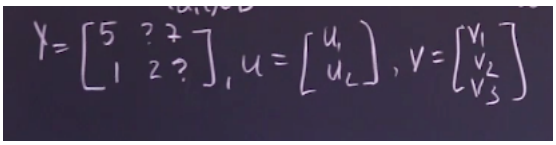
5. Alternating Minimization

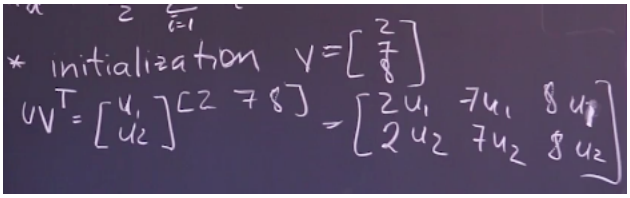
we now want to find U and V that minimize our new objective

$$J = \sum_{(a,i) \in D} \frac{(Y_{ai} - [UV^T]_{ai})^2}{2} + \frac{\lambda}{2}\left(\sum_{a,k} U_{ak}^2 + \sum_{i,k} V_{ik}^2\right).$$

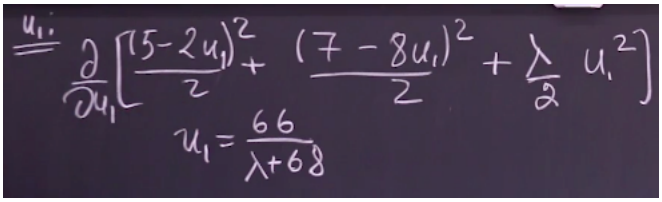
To simplify the problem, we fix U and solve for V, then fix V to be the result from the previous step and solve for U, and repeat this alternate process until we find the solution.

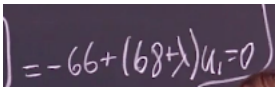
- Ex.



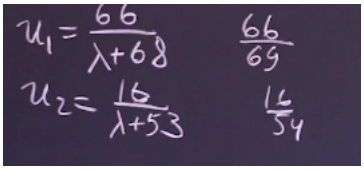


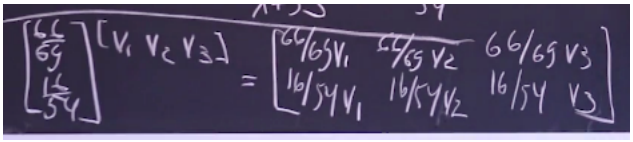
Initialize vector V randomly





Similarly we get u_2 (If we make lambda equal 1, then we have)





Compare this matrix with Y provided, repeat this computation process till convergence (no change of the value of objective function)