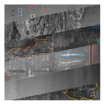
Machine Learning
Lecture 5
Gradient Descent (and Beyond)

Sardar Hamidian

The George Washington University
Department of Computer Science

Kaggle **Active**☺ or Completed Projects

13 Active Competitions



Severstal: Steel Defect Detection — $120,000, 1,911 teams
Can you detect and classify defects in steel?
Featured · Code Competition · 20 days to go · manufacturing, image data

Lyft 3D Object Detection for Autonomous Vehicles — $25,000, 193 teams
Can you advance the state of the art in 3D object detection?
Featured · a month to go · image data, object detection

RSNA Intracranial Hemorrhage Detection — $25,000, 648 teams
Identify acute intracranial hemorrhage and its subtypes
Featured · a month to go · health foundations and medical research, image data

The 3rd YouTube-8M Video Understanding Challenge — $25,000, 274 teams
Temporal localization of topics within video
Research · 7 days to go · object detection, video data

Kuzushiji Recognition — $15,000, 242 teams
Opening the door to a thousand years of Japanese culture
Playground · 10 days to go · multiclass classification, japan, image data, history

Understanding Clouds from Satellite Images — $10,000, 621 teams
Can you classify cloud structures from satellites?
Research · a month to go · atmospheric sciences, image data

Some active research shared tasks

ACL:
https://www.aclweb.org/portal/content/shared-task-reproduction-research-results-science-and-technology-language

Codalab
https://competitions.codalab.org/competitions/

CL-AFF (Started September 2018)
https://sites.google.com/view/affcon2020/cl-aff-shared-task

You propose…

Datasets

Dataset search
Kaggle datasets
Deepmind datasets
Google n-gram viewer
Google public data sets
Data Studio
Forbes/30 datasets
Your Choice

Highly recommend to get familiarized with could platforms and services

AWS

Google Cloud

Microsoft Azure

IBM Watson*

.....

Midterm will be discussed next week
Solution will be posted on BB
Midterm 2 will be postponed

How to deal with loss functions?

Convex, continuous and differentiable loss function ℓ(w)

Examples:

- SVM

$$\min_{\mathbf{w},b} \underbrace{\mathbf{w}^T\mathbf{w}}_{l_2-regularizer} + C \sum_{i=1}^{n} \underbrace{\max\left[1 - y_i(\mathbf{w}^T\mathbf{x} + b), 0\right]}_{hinge-loss}$$

- Logistic Regression

$$\underset{\vec{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1 + e^{-y_i \vec{w}^T \vec{x}})$$

Finding the minimum of Loss (L(w))

We don’t know how does the function look like?

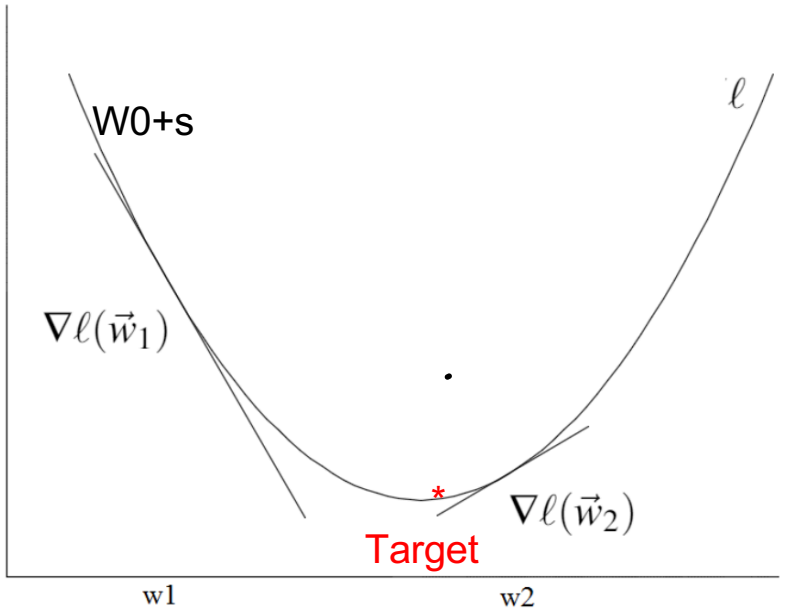
Linear and quadratic.

- **Taylor Expansion**
  - **Assumption hold for small s only**

$$\ell(\vec{w}+\vec{s}) \approx \ell(\vec{w}) + g(\vec{w})^{\top}\vec{s}$$

$$\ell(\vec{w}+\vec{s}) \approx \ell(\vec{w}) + g(\vec{w})^{\top}\vec{s} + \frac{1}{2}\vec{s}^{\top}H(\vec{w})\vec{s}$$

Faster with Hessian



Gradient Descent

- Using the first order approximation
  - Prove that $\ell(\vec{w}+\vec{s})<\ell(\vec{w})$

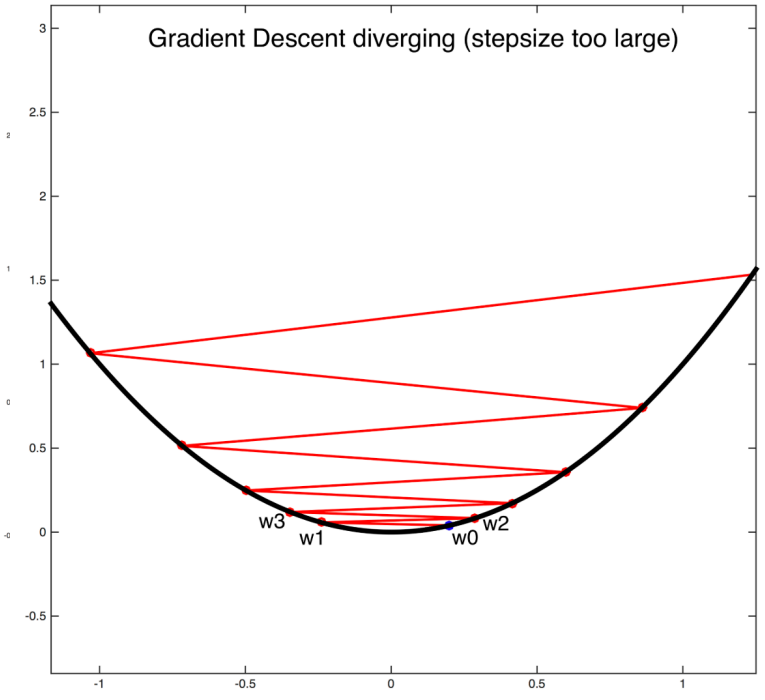
$$\ell(\vec{w}+\vec{s}) \approx \ell(\vec{w}) + g(\vec{w})^{\top}\vec{s}$$

$$\underbrace{\ell(\vec{w}+(-\alpha\vec{g}(\vec{w}))}_{after\ one\ update} \approx \ell(\vec{w}) - \underbrace{\overbrace{\alpha\vec{g}(\vec{w})^{T}\vec{g}(\vec{w})}^{>0}}_{>0} < \underbrace{\ell(\vec{w})}_{before}$$

How about learning rate?
Is there any guarantee that we will converge?

learning rate α

$$\vec{s} = -\alpha\, g(\vec{w}),$$



Adagrad

- Changes the step based on function
- Why this is important to have different learning rate for different function?
  - Each feature has its own learning rate

Adagrad Algorithm:

Initialize $\vec{w}_0$ and $\vec{z}$: $\forall d$: $w_d^0 = 0$ and $z_d = 0$

Repeat until converge:

$\vec{g} = \frac{\partial f(\vec{w})}{\partial \vec{w}}$ # Compute gradient

$\forall d$: $z_d \leftarrow z_d + g_d^2$

$\forall d$: $w_d^{t+1} \leftarrow w_d^t - \alpha \frac{g_d}{\sqrt{z_d + \epsilon}}$

If $\|\vec{w}^{t+1} - \vec{w}^t\|_2 < \delta$, converged! # for some small $\delta > 0$.

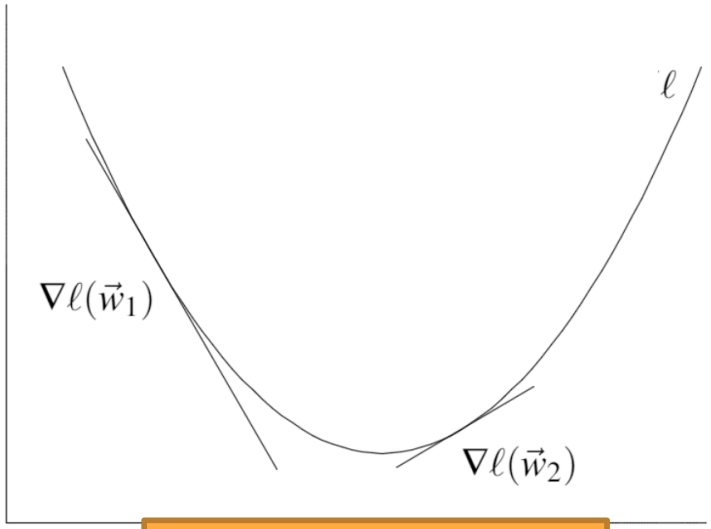
Newton's Method (2nd order Approximation)

- Assumption (loss ℓ is twice differentiable)
- Taylors approximation

$$\ell(\vec{w}+\vec{s}) \approx \ell(\vec{w}) + g(\vec{w})^\top \vec{s} + \frac{1}{2}\vec{s}^\top H(\vec{w})\vec{s}$$

Quadratic function with respect to s

$$H(\mathbf{w}) = \begin{pmatrix} \frac{\partial^2 \ell}{\partial w_1^2} & \frac{\partial^2 \ell}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \ell}{\partial w_1 \partial w_n} \\ \vdots & \cdots & \cdots & \vdots \\ \frac{\partial^2 \ell}{\partial w_n \partial w_1} & \cdots & \cdots & \frac{\partial^2 \ell}{\partial w_n^2} \end{pmatrix},$$



Hessian Matrix
Expensive to Compute

Newton's Method (2nd order Approximation)

- Assumption (loss ℓ is twice differentiable)
- Taylors approximation

$$\ell(\vec{w}+\vec{s}) \approx \ell(\vec{w}) + g(\vec{w})^\top \vec{s} + \frac{1}{2}\vec{s}^\top H(\vec{w})\vec{s}$$

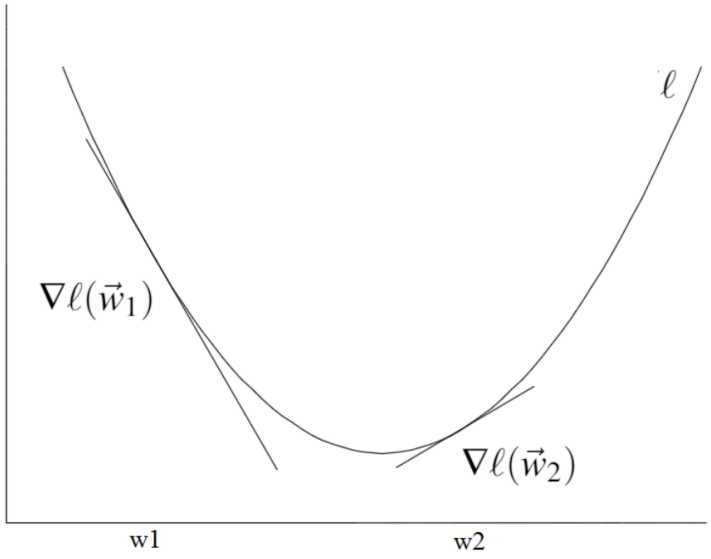
Quadratic function with respect to s

minimize

$$\underset{s}{argmin}\ \ell(\vec{w}) + g(\vec{w})^\top \vec{s} + \frac{1}{2}\vec{s}^\top H(\vec{w})\vec{s}$$

$$0 = g(\vec{w}) + H(\vec{w})\vec{s}$$

$$\Rightarrow \vec{s} = -[H(\vec{w})]^{-1} g(\vec{w}).$$



What is the drawbacks of Newton?
Doesn't work often!!!!!!☹Why?

Challenges for Newton's Method

1. Both the gradient and Hessian are normally computed **on the entire dataset**. This means many passes over the data to complete a series of Newton steps.

2. The Hessian has size $O(M^2)$ if there are M features. This is impractical for large feature spaces, e.g. text or event data.

3. Newton's method converges very fast when its feasible. But there are several challenges to using it with large datasets.

Challenges for Stochastic Gradient

Stochastic gradient has some serious limitations however, especially if the **gradients vary widely in magnitude**. Some coefficients change very fast, others very slowly. (The Hessian corrects for this in Newton's method).

This happens for **text, user activity and social media data** (and other power-law data), because gradient magnitudes scale with feature frequency, i.e. over several orders of magnitude.

Its not possible to set a single learning rate that trains the frequent and infrequent features at the same time.

Bias-Variance Tradeoff

- Glossary
- Bias and variance: conceptual definition
- Bias and variance: graphical definition
- Bias and variance: mathematical definition
- Learning curves
- Overfitting and underfitting: brief introduction

Glossary

- Target function
- Hypothesis and Hypothesis set
- Expected value:
- ȳ = expected label ḡ = expected hypothesis $\bar{h}$ = expected classifier
- Hypothesis complexity
- Computational complexity
- Noise, error, distribution variance, standard deviation, classifier variance
- Generalization error

Expected Test Error (quality of a machine learning algorithm)

Expected Test Error (given h_D):

$$E_{(\mathbf{x},y)\sim P}\left[(h_D(\mathbf{x})-y)^2\right] = \int_x \int_y (h_D(\mathbf{x})-y)^2 \Pr(\mathbf{x},y)\partial y \partial \mathbf{x}.$$

machine learning algorithm $h_D = \mathcal{A}(D)$
D training points
(x,y) pairs are the test points

Expected Test Error (given $\mathcal{A}$):

$$E_{\substack{(\mathbf{x},y)\sim P \\ D\sim P^n}}\left[(h_D(\mathbf{x})-y)^2\right] = \int_D \int_{\mathbf{x}} \int_y (h_D(\mathbf{x})-y)^2 \mathrm{P}(\mathbf{x},y)\mathrm{P}(D)\partial \mathbf{x}\partial y \partial D$$

Decomposition of Expected Test Error

$$
\begin{aligned}
E_{\mathbf{x},y,D}\left[\left[h_D(\mathbf{x}) - y\right]^2\right] &= E_{\mathbf{x},y,D}\left[\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right) + \left(\bar{h}(\mathbf{x}) - y\right)\right]^2\right] \\
&= E_{\mathbf{x},D}\left[\left(\bar{h}_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)^2\right] + 2\, E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x}) - y\right)\right] + E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x}) - y\right)^2\right]
\end{aligned}
$$

Decomposition of Expected Test Error

$$
\begin{aligned}
E_{\mathbf{x},y,D}\left[\left[h_D(\mathbf{x}) - y\right]^2\right] &= E_{\mathbf{x},y,D}\left[\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right) + \left(\bar{h}(\mathbf{x}) - y\right)\right]^2\right] \\
&= E_{\mathbf{x},D}\left[\left(\bar{h}_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)^2\right] + 2\, E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x}) - y\right)\right] + E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x}) - y\right)^2\right]
\end{aligned}
$$

$$
\begin{aligned}
E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x}) - y\right)\right] &= E_{\mathbf{x},y}\left[E_D\left[h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right]\left(\bar{h}(\mathbf{x}) - y\right)\right] \\
&= E_{\mathbf{x},y}\left[\left(E_D\left[h_D(\mathbf{x})\right] - \bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x}) - y\right)\right] \\
&= E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x}) - \bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x}) - y\right)\right] \\
&= E_{\mathbf{x},y}\left[0\right] \\
&= 0
\end{aligned}
$$

Decomposition of Expected Test Error

Returning to the earlier expression, we're left with the variance and another term

$$E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x})-y\right)^2\right] = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-y\right)^2\right]$$

Decomposition of Expected Test Error

Returning to the earlier expression, we're left with the variance and another term

$$E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x})-y\right)^2\right] = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-y\right)^2\right]$$

$$\begin{aligned} E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-y\right)^2\right] &= E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)+\left(\bar{y}(\mathbf{x})-y\right)^2\right] \\ &= \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x})-y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2} + 2\,E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\left(\bar{y}(\mathbf{x})-y\right)\right] \end{aligned}$$

$$\underbrace{E_{\mathbf{x},y,D}\left[(h_D(\mathbf{x})-y)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[(\bar{y}(\mathbf{x})-y)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$$

- Variance: If we have the best possible model for our training data, how far off are we from the average classifier?

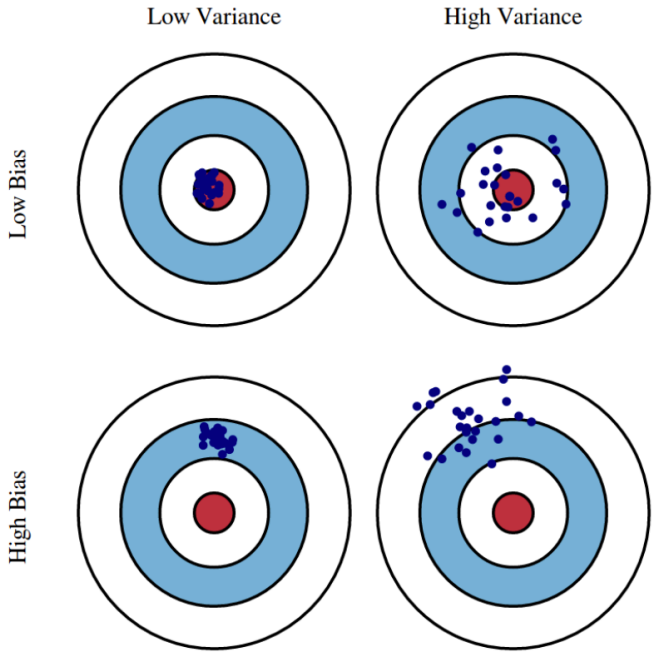
$$\underbrace{E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x})-y\right)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x})-y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$$

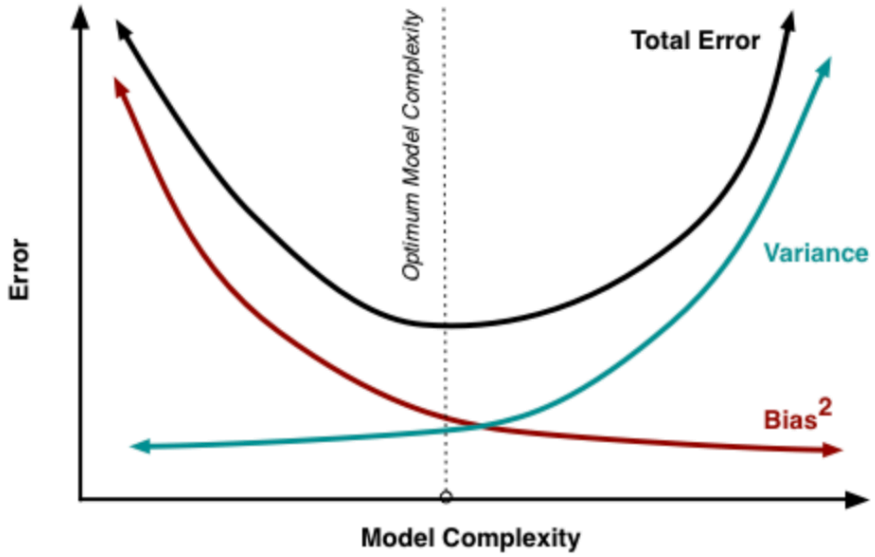
- **Bias**: What is the inherent error that you obtain from your classifier even with infinite training data? This is due to your classifier being "biased" to a particular kind of solution (e.g. linear classifier). In other words, bias is inherent to your model

$$\underbrace{E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x})-y\right)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x})-y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$$

- **Noise**: How big is the data-intrinsic noise? This error measures ambiguity due to your data distribution and feature representation. You can never beat this, it is an aspect of the data

Conceptual Example





Bias/Variance and Model Selection

- Flash back to ERM

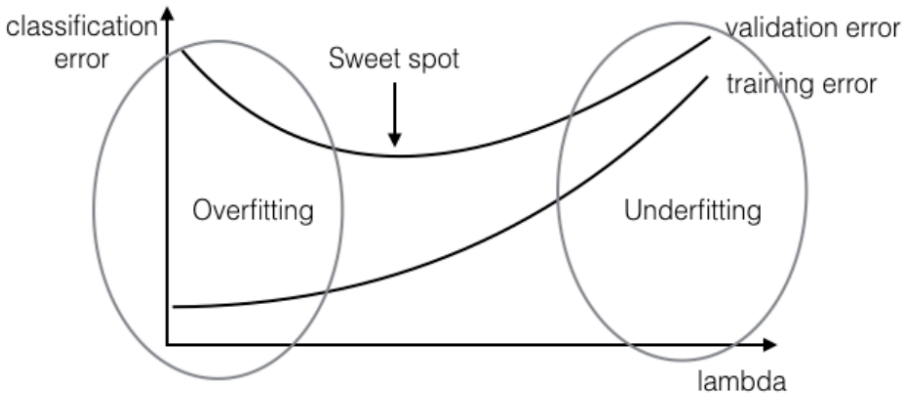
$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \underbrace{l_{(s)}(h_{\mathbf{w}}(\mathbf{x}_i), y_i)}_{Loss} + \underbrace{\lambda r(w)}_{Regularizer}$$

$\|\mathbf{w}\|_p$

Bias/Variance and Model Selection

- Flash back to ERM
- What Regularization Parameter does? $\|\mathbf{w}\|_p$

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \underbrace{l_{(s)}(h_{\mathbf{w}}(\mathbf{x}_i), y_i)}_{Loss} + \underbrace{\lambda r(w)}_{Regularizer}$$

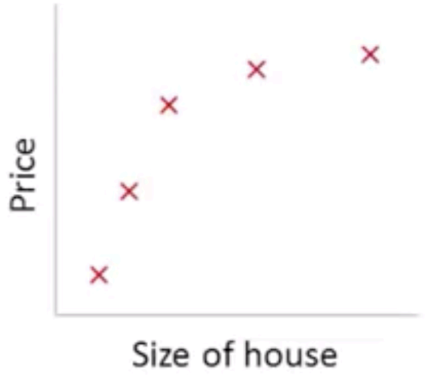


Flashback to regularizers

In regularized linear regression, we choose θ to minimize

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



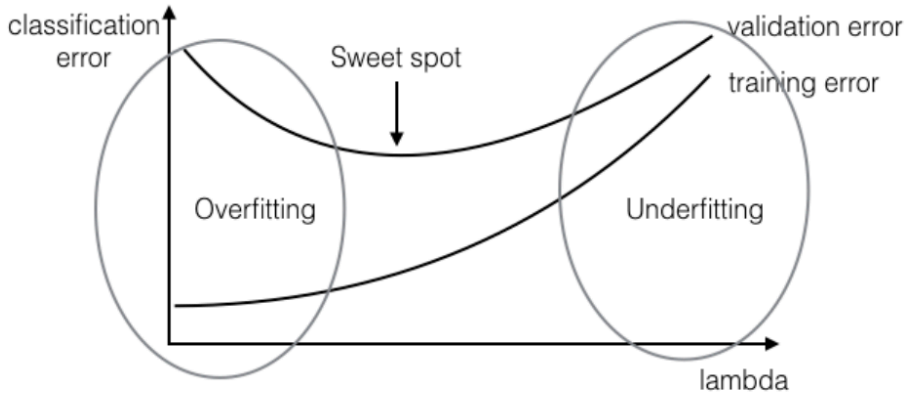
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Bias/Variance and Model Selection

- Flash back to ERM
- What Regularization Parameter does?
  - **Telescopic search**

$\|\mathbf{w}\|_p$

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \underbrace{l_{(s)}(h_{\mathbf{w}}(\mathbf{x}_i), y_i)}_{Loss} + \underbrace{\lambda r(w)}_{Regularizer}$$



Questions for you.

- How to find the best Lambda?
- How to find best regularizers?

Questions for you.

- How to find the best Lambda?
  - Grid Search
  - K-fold Cross validation

What is the risk of finding the best best lambda?

Questions for you.

- How to find the best Lambda?
  - Grid Search
  - K-fold Cross validation (How to decide about K)
  - Early Stopping

  What is the risk of finding the best best lambda?

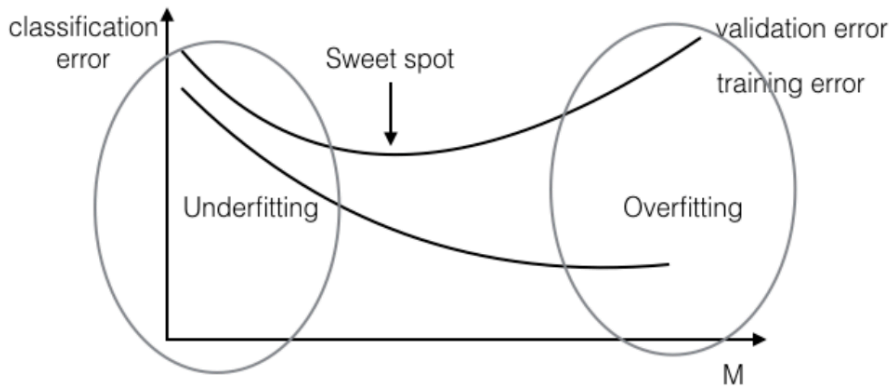
  Overfitting on validation set

Early Stopping?

Early Stopping?

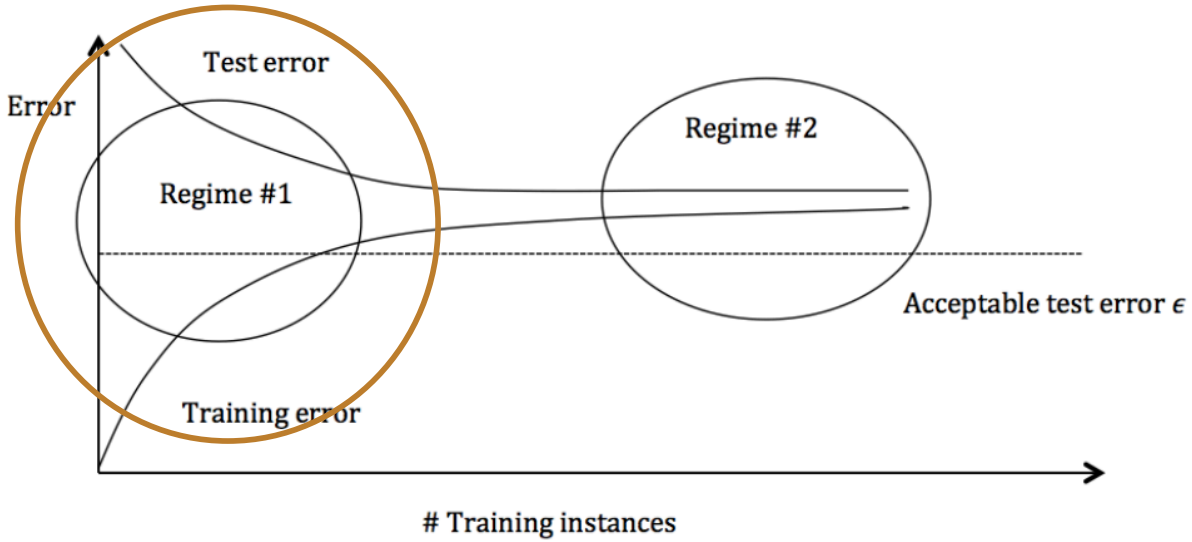
Stop your optimization after M (>= 0) number of gradient steps, even if optimization has not converged yet.

1) How regularizers and early stopping are related? More iteration is less regularization
2) Why this is better than trying out different lambda?



Detecting High Bias and High Variance

1) Diagnosing the problem.



High variance.

Symptoms:

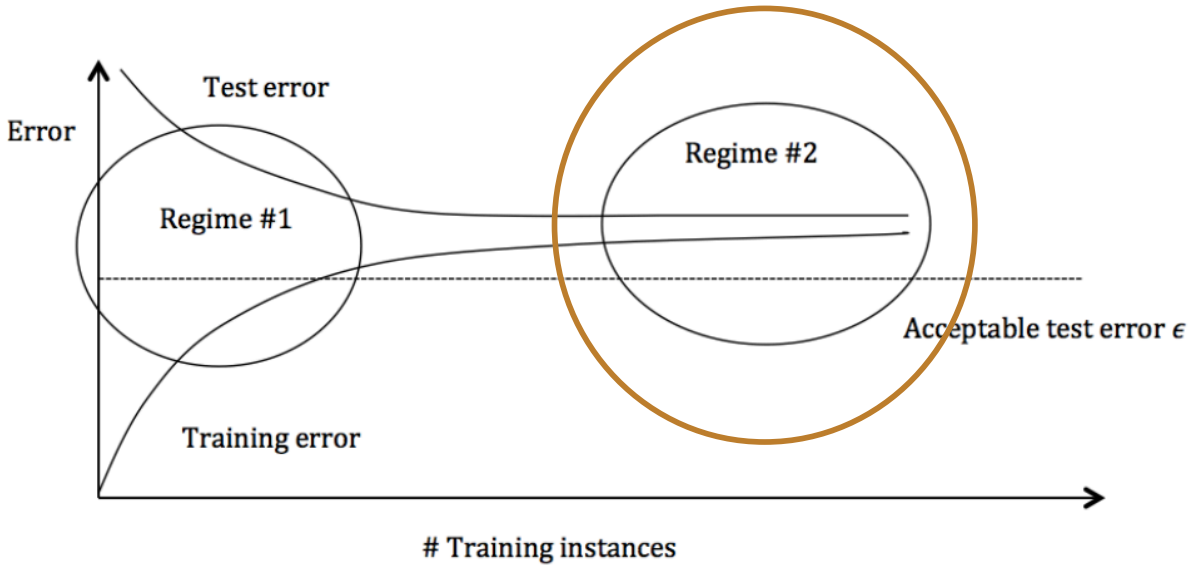
- Training error is much lower than test error
- Training error is lower than ϵ
- Test error is above ϵ

Remedies:

- Add more training data
- Reduce model complexity (complex models are prone to high variance
- Bagging (will be covered later in the course)

Detecting High Bias and High Variance

1) Diagnosing the problem.



High Bias

Symptoms:

Training error is higher than ϵ

Remedies:

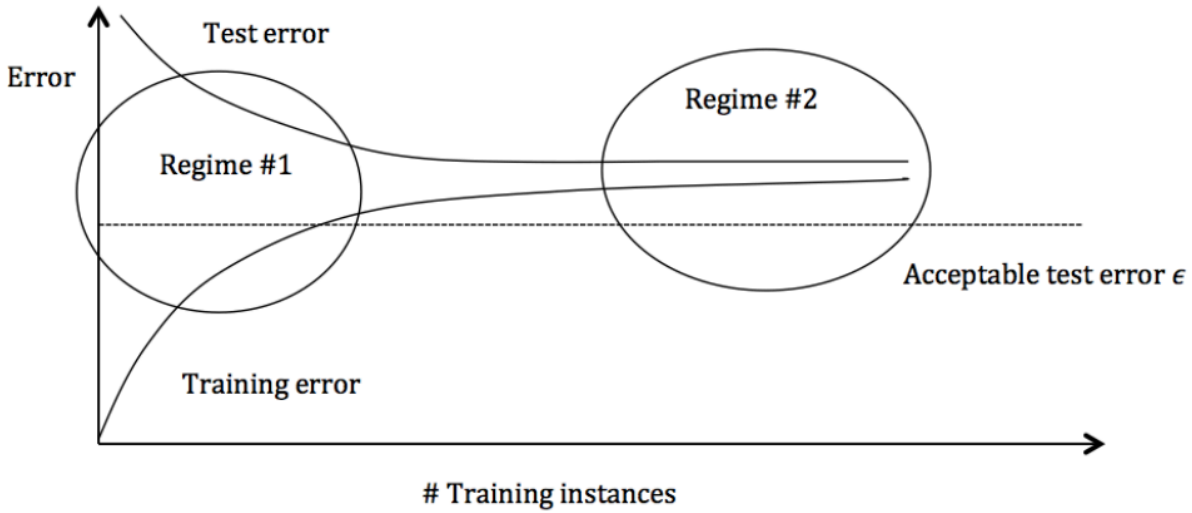
Use more complex model (e.g. kernelize, use non-linear models)

Add features

Boosting (will be covered later in the course)

Detecting High Bias and High Variance

1) Diagnosing the problem.



Q1)
How about regularization in Bias regime?

Q2)
How about noise?

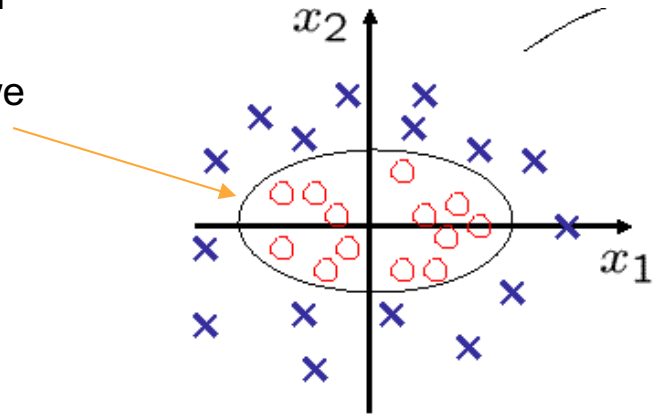
Q3)
Adding more data in Regime 2?

Q4)
Can test error go below training error?

Kernels

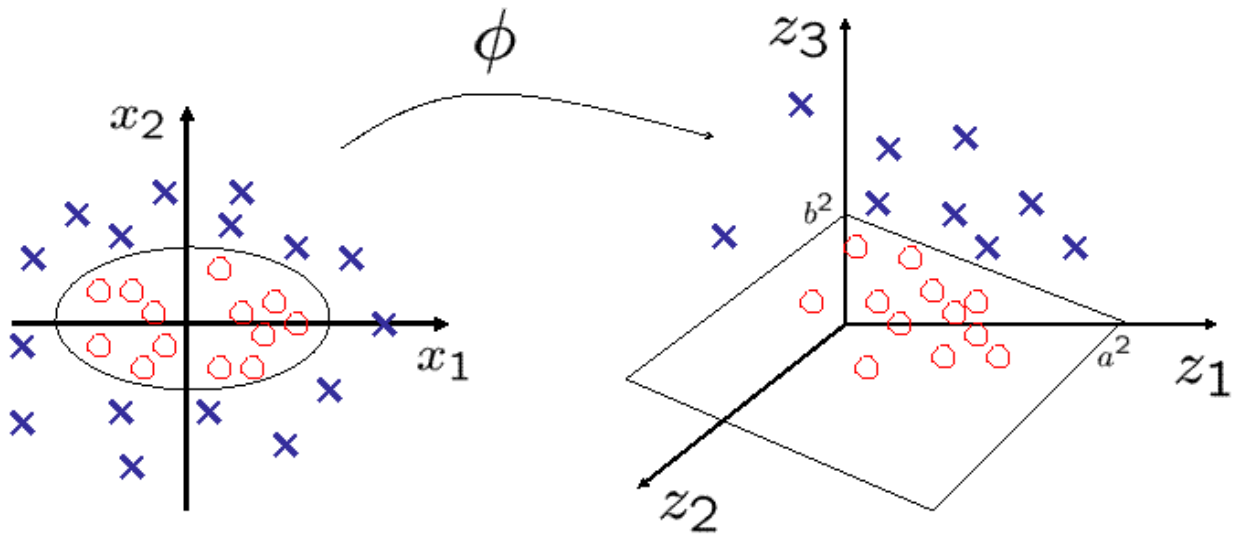
- Can we make non-linear classifiers by applying basis function (feature transformations) on input?

This is our ideal classifier but we have linear classifier. What should we do?

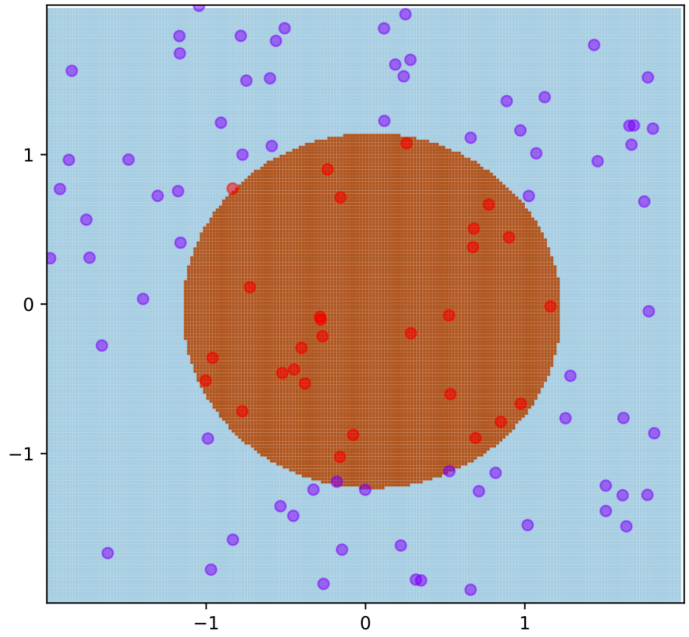


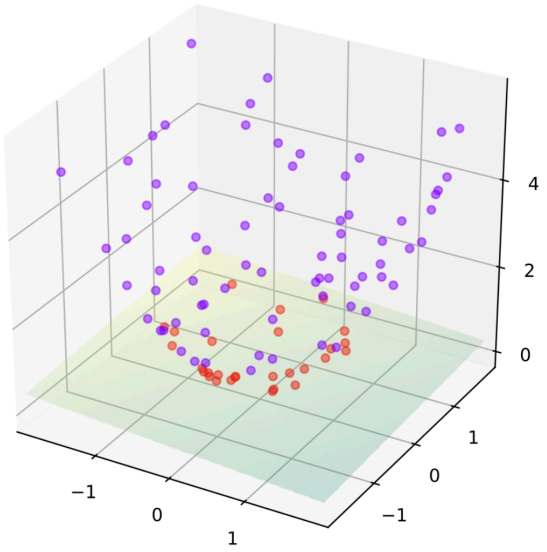
Kernels

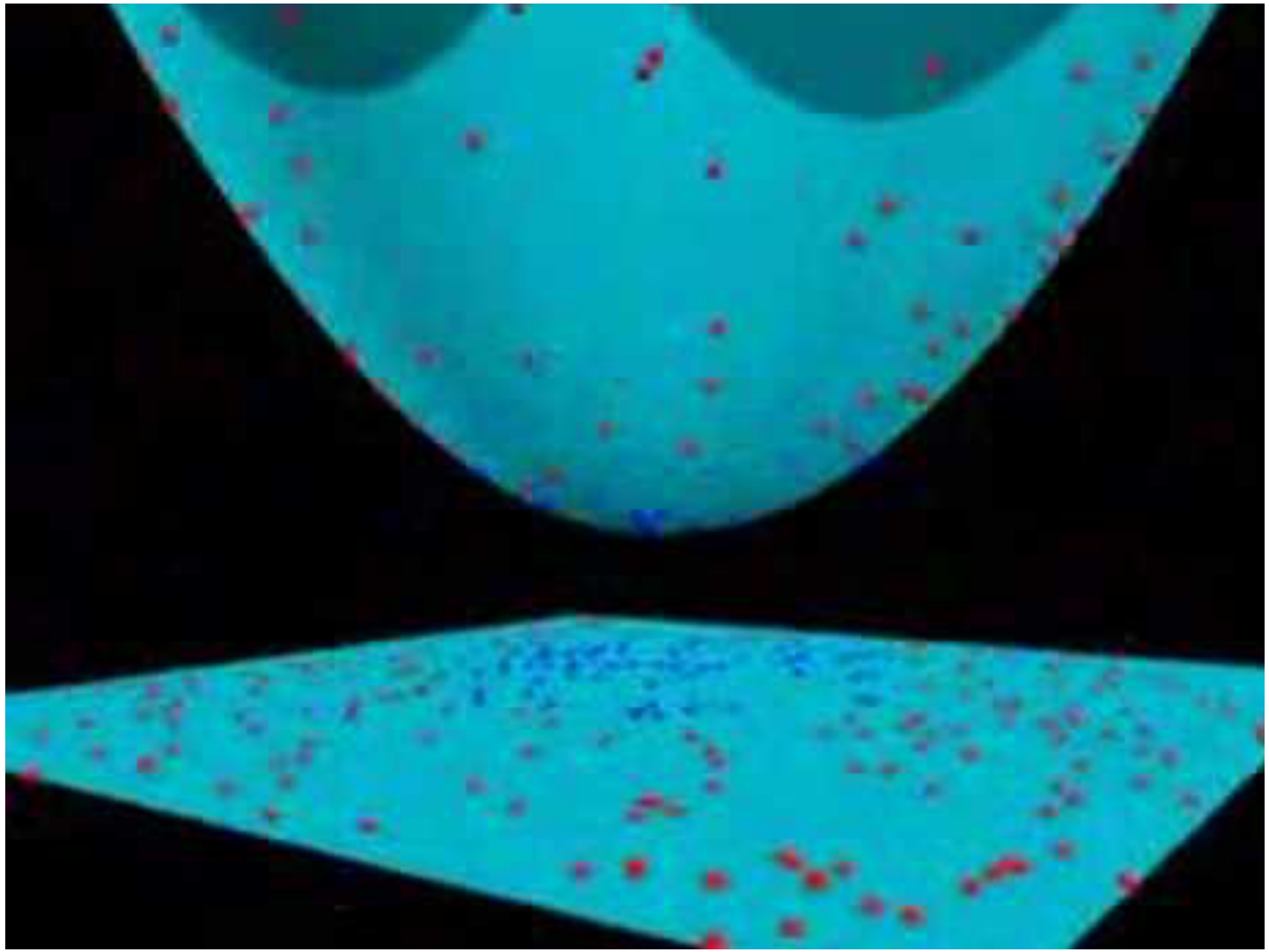
- Can we make non-linear classifiers by applying basis function (feature transformations) on input?



Example

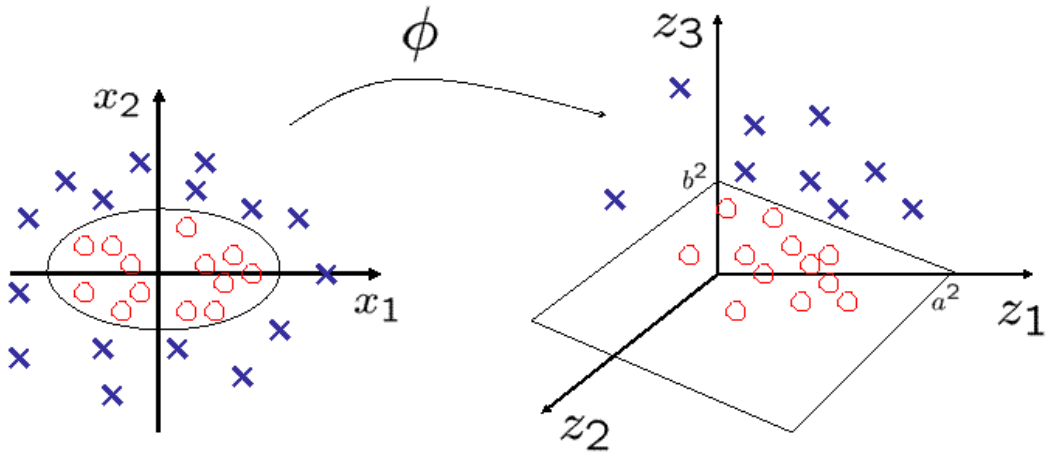






Kernels

- Can we make non-linear classifiers by applying basis function (feature transformations) on input?
- **Disadvantage**: ϕ(x) might be very high dimensional
- How to add new features?



Extending dimetionality

Q1) What is the max size of dimensionality for all the interaction of features?

Consider the following example: $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$, and define $\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}$.

Extending dimetionality

Q1) What is the max size of dimensionality for all the interaction of features?

Consider the following example: $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$, and define $\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}$.

The Kernel Trick

- Take squared loss for example:

$$\ell(\mathbf{w}) = \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

The gradient descent rule, with step-size $s > 0$, updates $\mathbf{w}$ over time,

$$w_{t+1} \leftarrow w_t - s\left(\frac{\partial \ell}{\partial \mathbf{w}}\right) \text{ where: } \frac{\partial \ell}{\partial \mathbf{w}} = \sum_{i=1}^{n} \underbrace{2(\mathbf{w}^\top \mathbf{x}_i - y_i)}_{\gamma_i \text{ : function of } \mathbf{x}_i, y_i} \mathbf{x}_i = \sum_{i=1}^{n} \gamma_i \mathbf{x}_i$$

Linear Combination of all input vectors

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i.$$

A)

$$w_{t+1} \leftarrow w_t - s(\frac{\partial \ell}{\partial \mathbf{w}}) \text{ where: } \frac{\partial \ell}{\partial \mathbf{w}} = \sum_{i=1}^{n} \underbrace{2(\mathbf{w}^\top \mathbf{x}_i - y_i)}_{\gamma_i \text{ : function of } \mathbf{x}_i, y_i} \mathbf{x}_i = \sum_{i=1}^{n} \gamma_i \mathbf{x}_i$$

B)

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i.$$

B A

$$\begin{aligned}
\mathbf{w}_1 &= \mathbf{w}_0 - s \sum_{i=1}^{n} 2(\mathbf{w}_0^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^0 \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^0 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^1 \mathbf{x}_i && (\text{with } \alpha_i^1 = \alpha_i^0 - s\gamma_i^0) \\
\mathbf{w}_2 &= \mathbf{w}_1 - s \sum_{i=1}^{n} 2(\mathbf{w}_1^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^1 \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^1 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^2 \mathbf{x}_i && (\text{with } \alpha_i^2 = \alpha_i^1 \mathbf{x}_i - s\gamma_i^1) \\
\mathbf{w}_3 &= \mathbf{w}_2 - s \sum_{i=1}^{n} 2(\mathbf{w}_2^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^2 \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^2 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^3 \mathbf{x}_i && (\text{with } \alpha_i^3 = \alpha_i^2 - s\gamma_i^2) \\
&\dots \qquad\qquad \dots && \dots \\
\mathbf{w}_t &= \mathbf{w}_{t-1} - s \sum_{i=1}^{n} 2(\mathbf{w}_{t-1}^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^{t-1} \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^{t-1} \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^t \mathbf{x}_i && (\text{with } \alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1})
\end{aligned}$$

The update-rule for α_i^t is thus

$$\alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1}, \text{ and we have } \alpha_i^t = -s \sum_{r=0}^{t-1} \gamma_i^r.$$

Magic just happend

- Inner product of training inputs

$$\mathbf{w}^\top \mathbf{x}_j = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^\top \mathbf{x}_j.$$

- Re-writing the loss function

$$\ell(\mathbf{w}) = \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

$$\ell(\alpha) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j \mathbf{x}_j^\top \mathbf{x}_i - y_i \right)^2$$

$$h(\mathbf{x}_t) = \mathbf{w}^\top \mathbf{x}_t = \sum_{j=1}^{n} \alpha_j \mathbf{x}_j^\top \mathbf{x}_t.$$

How about the inner product?

We need to calculated inner product instead of W right?

$$\phi(\mathbf{x})^\top \phi(\mathbf{z})$$

$$\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}.$$

The sum of 2^d terms becomes the product of d terms. We can compute the inner-product from above formula in time $O(d)$ instead of $O(2^d)$! In fact, we can pre-compute them and store them in a kernel matrix

$$\underbrace{\mathsf{K}(\mathbf{x}_i, \mathbf{x}_j)}_{\text{this is called the } \textbf{Kernel Matrix}} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j).$$

$$\phi(\mathbf{x})^\top \phi(\mathbf{z}) = 1 \cdot 1 + x_1 z_1 + x_2 z_2 + \cdots + x_1 x_2 z_1 z_2 + \cdots + x_1 \cdots x_d z_1 \cdots z_d = \prod_{k=1}^{d} (1 + x_k z_k).$$

How about the inner product?

We need to calculated inner product instead of W right?

$$\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}. \urcorner$$

$$\phi(\mathbf{x})^\top \phi(\mathbf{z})$$

The sum of 2^d terms becomes the product of d terms. We can compute the inner-product from above formula in time $O(d)$ instead of $O(2^d)$! In fact, we can pre-compute them and store them in a kernel matrix

$$\underbrace{\mathsf{K}(\mathbf{x}_i, \mathbf{x}_j)}_{\text{this is called the } \textbf{Kernel Matrix}} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j).$$

$$\phi(\mathbf{x})^\top \phi(\mathbf{z}) = 1 \cdot 1 + x_1 z_1 + x_2 z_2 + \cdots + x_1 x_2 z_1 z_2 + \cdots + x_1 \cdots x_d z_1 \cdots z_d = \prod_{k=1}^{d} (1 + x_k z_k).$$

Can we use other inner product function?(Kernel functions)

Following all some common kernel functions:

Linear: $\mathsf{K}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$.

(The linear kernel is equivalent to just using a good old linear classifier - but it can be faster to use a kernel matrix if the dimensionality d of the data is high.)

Polynomial: $\mathsf{K}(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^\top \mathbf{z})^d$.

Radial Basis Function (RBF) (aka Gaussian Kernel): $\mathsf{K}(\mathbf{x}, \mathbf{z}) = e^{\frac{-\|\mathbf{x}-\mathbf{z}\|^2}{\sigma^2}}$.

Other Kernel

Exponential Kernel: $\mathsf{K}(\mathbf{x}, \mathbf{z}) = e^{\frac{-\|\mathbf{x}-\mathbf{z}\|}{2\sigma^2}}$

Laplacian Kernel: $\mathsf{K}(\mathbf{x}, \mathbf{z}) = e^{\frac{-|\mathbf{x}-\mathbf{z}|}{\sigma}}$

Sigmoid Kernel: $\mathsf{K}(\mathbf{x}, \mathbf{z}) = \tanh(\mathbf{a}\mathbf{x}^\top + c)$

Think about it: Can any function $\mathsf{K}(\cdot, \cdot)$ be used as a kernel?

No, the matrix $\mathsf{K}(\mathbf{x}_i, \mathbf{x}_j)$ has to correspond to real inner-products after some transformation $\mathbf{x} \to \phi(\mathbf{x})$. This is the case if and only if K is positive semi-definite.

Definition: A matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite iff $\forall \mathbf{q} \in \mathbb{R}^n$, $\mathbf{q}^\top A \mathbf{q} \geq 0$.

Why is that?

Remember $\mathsf{K}(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$. A matrix of form $A = \begin{pmatrix} \mathbf{x}_1^\top \mathbf{x}_1, \ldots, \mathbf{x}_1^\top \mathbf{x}_n \\ \vdots\vdots \\ \mathbf{x}_n^\top \mathbf{x}_1, \ldots, \mathbf{x}_n^\top \mathbf{x}_n \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} (\mathbf{x}_1, \cdots \mathbf{x}_n)$

must be positive semi-definite because: $\mathbf{q}^\top A \mathbf{q} = (\underbrace{(\mathbf{x}_1, \cdots \mathbf{x}_n)\, \mathbf{q}}_{\text{a vector with the same dimension of } \mathbf{x}_i})^\top ((\mathbf{x}_1, \cdots \mathbf{x}_n)\, \mathbf{q}) \geq 0$

for $\forall \mathbf{q} \in \mathbb{R}^n$.