Lecture 16 EE 559 Mar 20, 2024

Machine Learning I: Supervised Methods

B. Keith Jenkins

Announcements

- Slido event code: 4509075
- Homework 6 will be posted later this week.
- Midterm exams are being graded
  - Estimate completion by Monday
  - If you thought the exam was too long, remember that class will be graded primarily on the curve.

Reading

- Bishop 5.0-5.3
  - Feedforward artificial neural networks (ANN)

Today's lecture

- Support vector regression (part 2) (Jenkins)
- Validation and test sets (Sagar)
  - Model selection
  - Cross validation
  - Cross validation for model selection
- Introduction to VC dimension (Sagar)
  - Classifier complexity, d.o.f., constraints
- Feature selection and dimensionality reduction (part 1) (if time) (Sagar)
  - PCA

Review from Lecture 14:

Support Vector Regression (SVR) [Bishop 7.1.4] (non-augmented notation)

SVM adapted to regression problems

$\underline{x} \rightarrow$ [NL mapping $\underline{\phi}(\underline{x})$ or $k(\underline{x}, \underline{x}')$] $\rightarrow \underline{u} \rightarrow$ [Linear regresor] $\rightarrow \hat{y}$

Motivation: to have a regression model suitable to sparse data (e.g., high D', relatively low N).

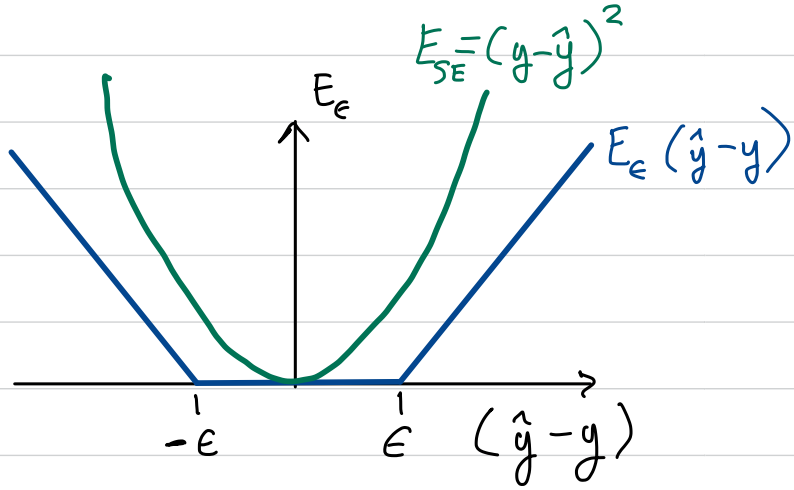
In Ridge Regression, we used a criterion fcn.: MSE + l_2^2 regularizer:

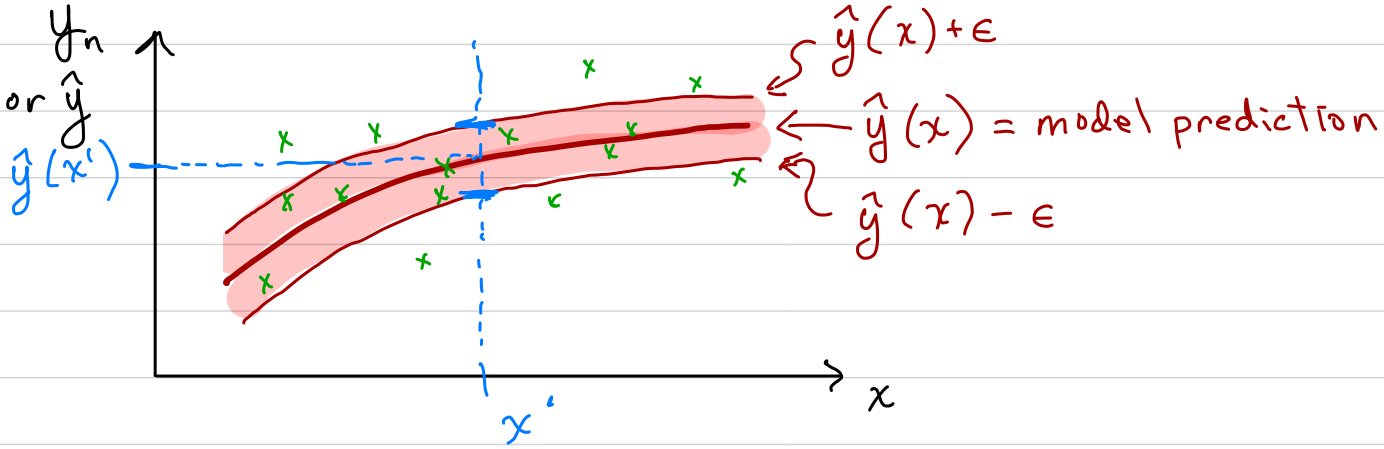
$$J_{RR}(\underline{w}) = \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left[\hat{y}(\underline{x}_n) - y_n\right]^2}_{\text{MSE}} + \underbrace{\lambda \|\underline{w}\|_2^2}_{l_2^2 \text{ regularizer}}, \quad \lambda \geq 0$$

For SVR: use "ϵ-insensitive" loss instead of squared-error loss; keep l_2^2 regularizer

ϵ-insensitive loss E_ϵ:

$$E_\epsilon(\hat{y}-y) = \begin{cases} 0, & \text{if } |\hat{y}-y| < \epsilon \\ |\hat{y}-y| - \epsilon & \text{if } |\hat{y}-y| \geq \epsilon \end{cases}$$





Any data points in the shaded region have $E_\epsilon = 0$.

- 0 error for predictions $\hat{y}(\underline{x}_n)$ within ϵ of known output y_n, means learning algorithm is less likely to overfit to noise or small variations in data.

$\Rightarrow$ Use error function (before adding constraints):

$$\tilde{J}(\underline{w}) = C \sum_{n=1}^{N} E_\epsilon \left[\hat{y}(\underline{x}_n) - y_n\right] + \frac{1}{2}\|\underline{w}\|_2^2$$

↑ C here by convention.

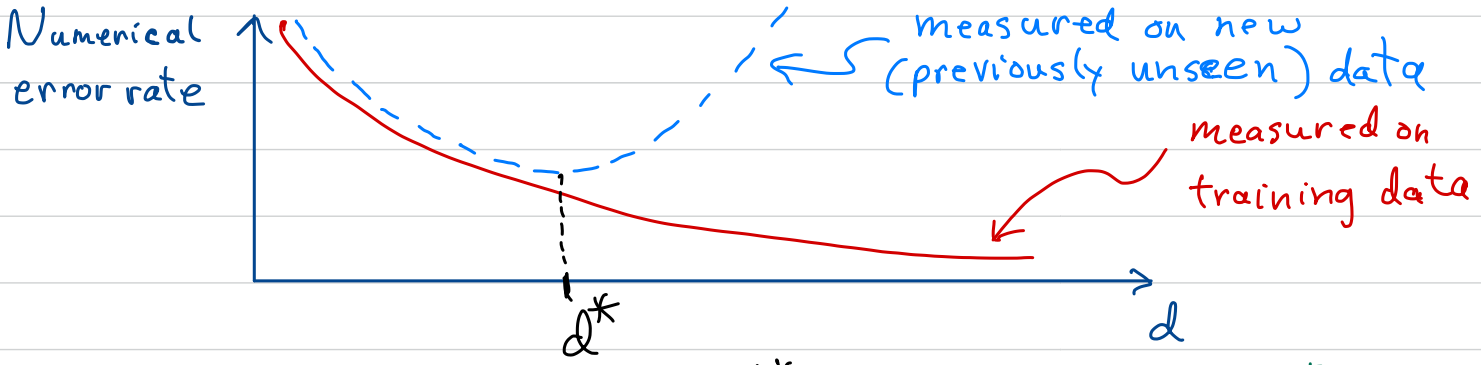
* Pages 4-5 were deferred to Lecture 17 due to technical difficulties with Webex audio.

Validation and Error Estimation, including model selection

How can we:
(i) – Estimate error rate on unseen data ?
(ii) – Choose parameter values and dimensionality for a high-accuracy result without overfitting ?

(ii) Validation (data) set – for model selection or evaluation ("validation")

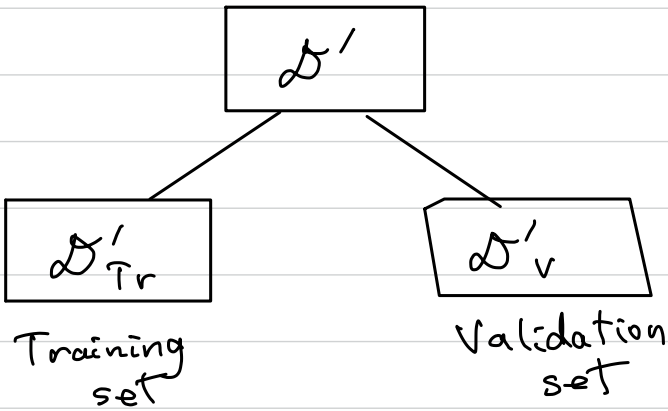
Ex: Linear classifier
Parameter: d = # dimensions (variable)
[d here is #dimensions, not necessarily polynomial degree.]



→ Find optimal $d = d^*$.

e.g.: $g(\underline{x}) = \underline{w}^T \underline{x} + w_0$
⇒ $d+1$ w_j values.

Given a labelled dataset = $\mathcal{D}'$



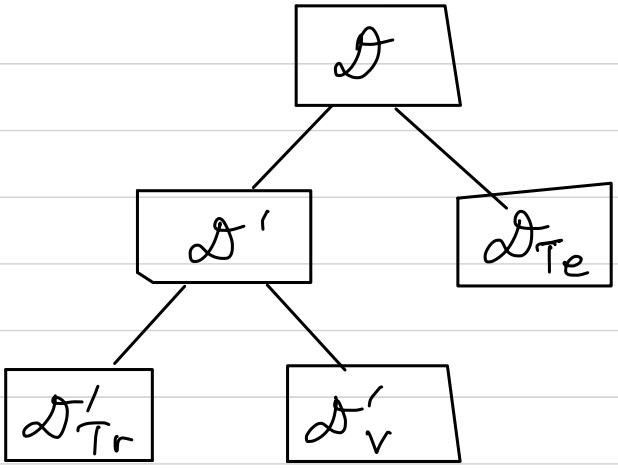
Note:
$\mathcal{D}'_{Tr} \cap \mathcal{D}'_{V} = \emptyset$.

(i) Test (data)set – for estimating error rate of final classifier.

→ To get a good estimate of error rate on unknowns ("out of sample error"), use a test set, $\mathcal{D}_{Te}$, that has not been seen or used by:

- training algorithm
- model selection / validation set
- person designing or optimizing the M.L. system (until the system is fully designed).

Procedure:



$\leftarrow \mathcal{D}' \cap \mathcal{D}_{Te} = \emptyset$

$\leftarrow \mathcal{D}'_{Tr} \cap \mathcal{D}'_{V} = \emptyset$

Validation for Model Selection

Model selection:

Ex: Suppose use SVM with RBF kernel:

$$k(\underline{x}, \underline{x}_n) = \exp\{-p\|\underline{x}-\underline{x}_n\|^2\}, \quad p > 0.$$

We want to find best choice of p.

[Comment: consider p here to be a dummy variable for a parameter that we want to optimize by using model selection.]

Suppose we use a search over values of p.

→ Use a validation set

For $k = 0, 1, 2, \cdots, K$

Use model p: $p = p_{min} + k\Delta p$

Initialize training procedure

Train classifier using p on $\mathcal{D}_{Tr}'$ → $\hat{\underline{w}}(p)$

Evaluate classifier on $\mathcal{D}_V'$

Store error $\mathcal{E}_p$

Next k

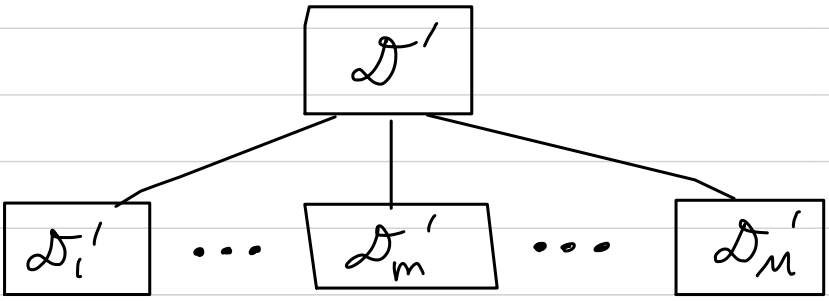
Pick value $p^* = \underset{p}{\text{argmin}}\ \mathcal{E}_p$

If we want to use data efficiently (keeping $\mathcal{D}_{Tr}'$ as large as possible), we can use cross validation instead.

Cross Validation for Model Selection

→ Re-use (carefully) data points in $\mathcal{D}_{Tr}'$, $\mathcal{D}_V'$.

$\mathcal{D}' \longrightarrow M$ ~equal-sized subsets.
(Chosen randomly (without replacement), but typically subject to constraint that % of data pts. in each class ~ same in each subset as in $\mathcal{D}'$.) ("stratified")



(for M-fold cross-val.)

⇒ all subsets $\mathcal{D}_i'$ are disjoint.

For $i = 1, \cdots, M$ (i = "fold" of cross-validation)
Let $\mathcal{D}_V' = \mathcal{D}_i'$, $\mathcal{D}_{Tr}' = \bigcup_{j \neq i} \mathcal{D}_j'$

For $k = 0, 1, 2, \cdots, K$
Use model p: $p = p_{min} + k\Delta p$

crucial step → Initialize training procedure
Train classifier using p on $\mathcal{D}_{Tr}'$
Evaluate classifier on $\mathcal{D}_V'$
Store error $\mathcal{E}_{pi}$
Next k

similar to validation above

Optionally store $p_i'^* = \underset{p}{\text{argmin}}\ \mathcal{E}_{pi}$
Next i

If run cross-val. only once, could:

$$\mathcal{E}'_p = \frac{1}{M} \sum_{i=1}^{M} \mathcal{E}_{pi}$$

$$p^* = \underset{p}{\text{argmin}}\ \mathcal{E}'_p$$

Typically run a number of times (new partitioning of $\mathcal{D}' \to \mathcal{D}'_i$ each time) to get better estimates.

Run T times. $(t = 1, 2, \cdots, T)$

Define:

$$\mathcal{E}_p = \frac{1}{MT} \sum_{t=1}^{T} \sum_{i=1}^{M} \mathcal{E}_{pi}^{(t)}$$

= average error for parameter value p.

Then: pick

$$\longrightarrow \quad p^* = \underset{p}{\text{argmin}}\ \mathcal{E}_p .$$

Then, how find best $\underline{w} = \underline{w}^*$ for $p = p^*$?

→ Train again using $p = p^*$, and using $\mathcal{D}'$ Call (training data).

THEN, FOR ERROR ESTIMATION (ON UNSEEN DATA)

(1) Use $\mathcal{D}_{Te}$ to test classifier (or M.L. system) that uses p^*, $\underline{w}^*$ (the chosen model). [one pass method]

or

(2) Add an outer loop of cross validation for $\mathcal{D}$ → $\mathcal{D}'$, $\mathcal{D}_{Te}$

Degrees of Freedom, Classifier Complexity, and VC Dimension

Earlier: d.o.f. = # indep. variables or parameters (during learning)
constraints ≈ # training data points (e.g., in MSE problem like pseudoinverse)
Risk of overfitting if # constr. < (3-10) × d.o.f.

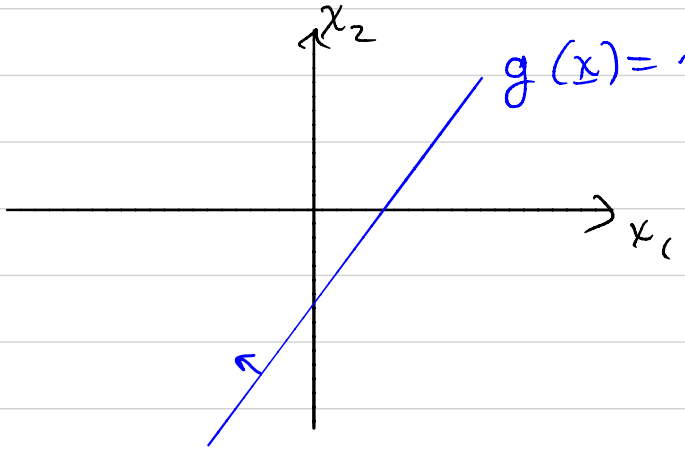
Instead of d.o.f. (for classifier complexity), a more rigorous metric is

VC dimension [Vapnik-Chervonenkis],

which quantifies the "flexibility" or "capacity" of a decision boundary (or dec. regions).

Examples:

1. Linear boundary in (D+1) space [D features] (2-class problem)



$g(\underline{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$ (D=2)

$\Rightarrow$ d.o.f. = 3

generally: d.o.f. = D+1

VC dimension = D+1.

2. Polynomial boundary $\left[g(\underline{x}) \underset{S_2}{\overset{S_1}{\gtrless}} 0, \text{ with } g(\underline{x}) = \text{polyn. fcn. of } \underline{x} \right]$ ($C=2$ classes)

If g has $D'+1$ weights, then: VC dim $= D'+1$.

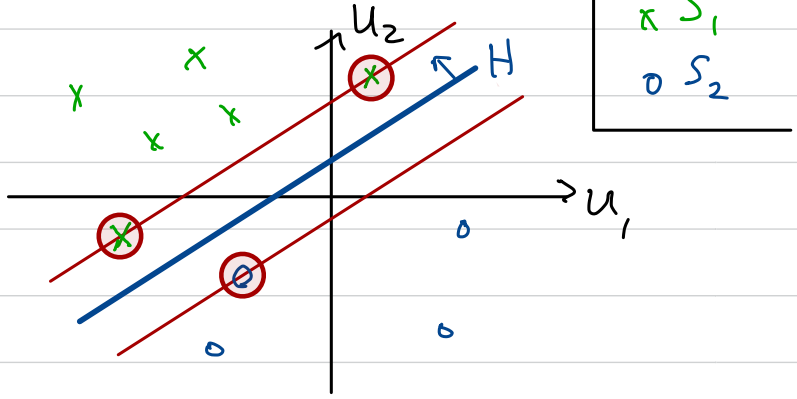
Note that: higher VC dim $\Rightarrow$ higher capacity of dec. boundary.

$\therefore$ VC dim $\longleftrightarrow N$ is important (overfit, underfit etc.)

$\Rightarrow$ can tell us how good we expect the generalization (performance on unknowns) to be.

What about SVM?

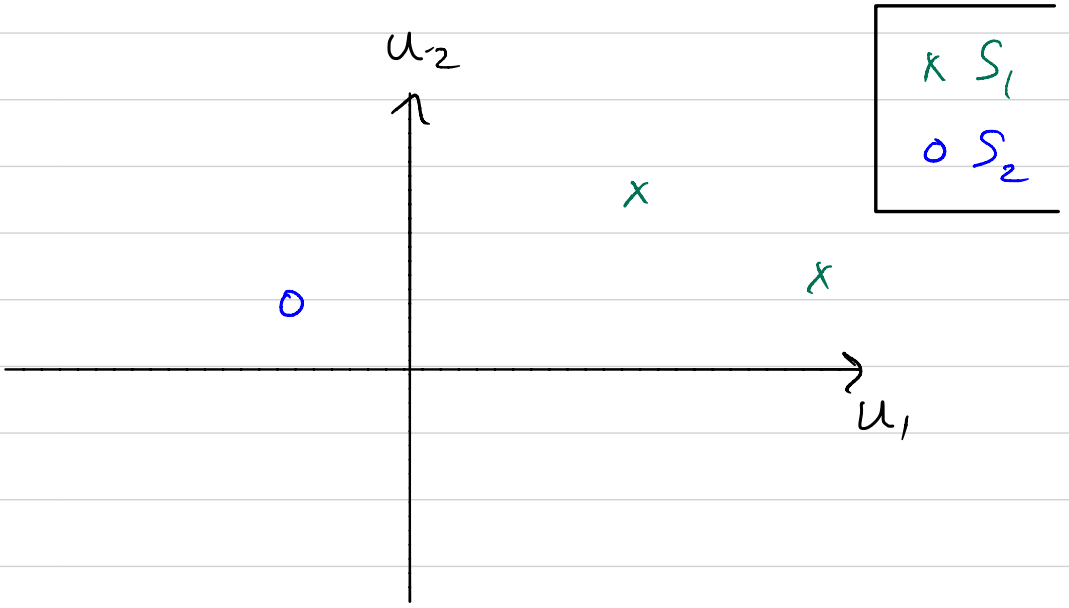
- For SVM, because of its additional constraints included in the criterion function, its VC dim can be $< D'+1$, and e.g. can be a fcn. of #support vectors, instead of # weights.
  - The support vectors are the data points that lie on the margin boundary (after training).
  (linearly separable case)



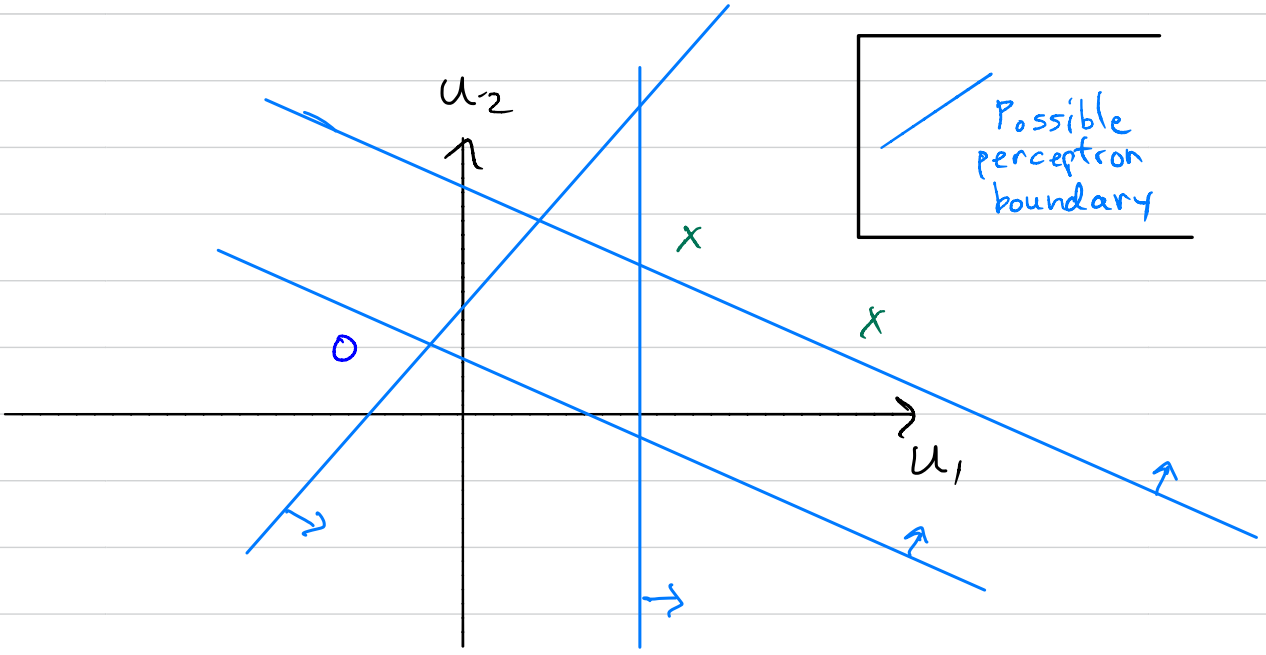
○ Support vector

Intuitive example:

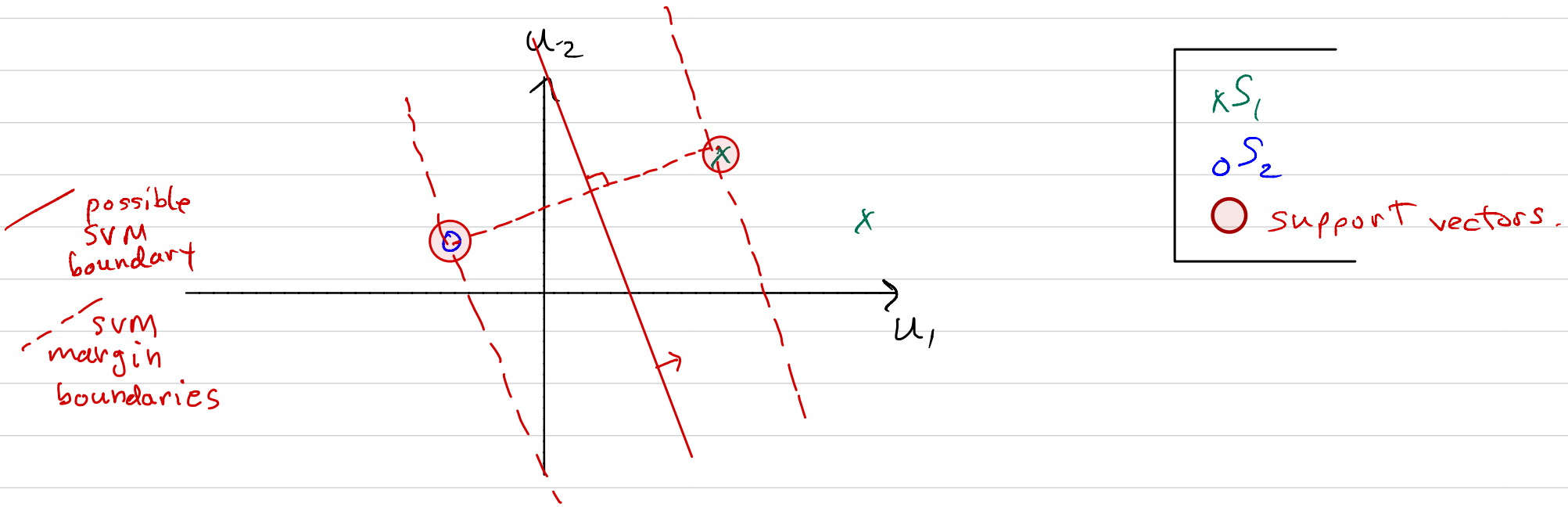
- Given 3 data points in D=2 unaugmented feature space →



- What decision boundaries could perceptron converge to? ↘



- What decision boundaries could SVM converge to? ↘



- Thus, SVM can generalize much better with sparsely populated feature space ([# weights] × (3-10) $\nless N$], than many other classifiers (e.g., pseudo-inv. or perceptron)

- This allows ϕ mapping to higher dimensional space in cases that would yield too many weights using other classifiers (that add no constraints).

Feature Selection and Dimensionality Reduction

Why?

1. Reduce computation requirements
2. Reduce likelihood of overfitting
   - by balancing d.o.f. and N.
3. Reduce correlation among features
4. Eliminate irrelevant features

Effect: reduce feature space dimension $D \rightarrow D' < D$.

Two overall scenarios

(1) Feature selection.

Choose some features to eliminate (using some criterion)
$\Rightarrow D' < D$, features of D' are a subset of features of D.

(2) Feature-space transformation

Transform to a new, lower dimensional, feature space.

- PCA: keep D' principal components. ($D' \leq D$).
- FLD: find an optimal transformation using FLD criterion (binary classification problems).
- MDA: extends FLD to $C > 2$ classes.

$\Rightarrow D' < D$ typically. Features of D' are linear combinations of features of D.

Principal Components Analysis (PCA) [Bishop 12.1]

Also called: Karhunen-Loeve (KL) Transform

Idea: Project data (of all classes) onto a lower-dimensional space.

$D \rightarrow D'$, $D' < D$.

using a linear transform.

PCA's criterion:

- Minimize MSE between projected points $\tilde{x}_n$ and original points $\underline{x}_n$.

PCA can be computed by orthonormal transformation

Need $\underline{\underline{\Sigma}}$ = covariance matrix. (will be covered next week)
Use an estimate based on training dataset -

(1) Sample covariance matrix: Let $\hat{\underline{m}}$ = sample mean = $\frac{1}{N}\sum_{i=1}^{N} \underline{x}_i$

$$\hat{\underline{\underline{\Sigma}}} \triangleq \frac{1}{N}\sum_{i=1}^{N} (\underline{x}_i - \hat{\underline{m}})(\underline{x}_i - \hat{\underline{m}})^T$$

($\hat{\underline{\underline{\Sigma}}}$ is real and symmetric)

(2) Find eigenvectors $\underline{e}_d$ and eigenvalues λ_d of $\hat{\underline{\underline{\Sigma}}}$ [1]

(3) Keep " corresponding to D' largest eigenvalues.

(4) Transform all data pts. $\underline{x}_i$ to new feature space:

$$\tilde{\underline{x}}_i = \underline{\underline{E}}^T \underline{x}_i, \quad \underline{\underline{E}} \triangleq [\underline{e}_1\ \underline{e}_2 \cdots \underline{e}_{D'}] \text{ such that}$$

$\{\underline{e}_1, \underline{e}_2, \cdots, \underline{e}_{D'}\}$ is orthonormal. (each $\underline{e}_j$ is normalized to unit length)

[1] $\hat{\underline{\underline{\Sigma}}}$ is assumed positive definite (more later), so its eigenvalues are all positive.

Suppose we have 2-class classification problem with $D = 2$.

We use PCA to reduce feature space to $D = 1$.
Then we use a linear classifier in the 1D feature space.
How well will the classifier perform on the following 2 datasets?

How well does PCA work?

Ex: $C = 2$ classes

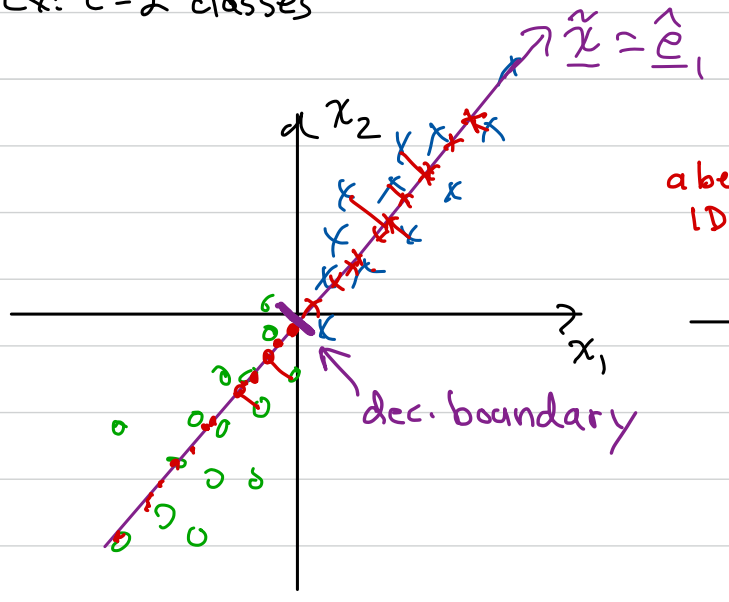


Fig. A

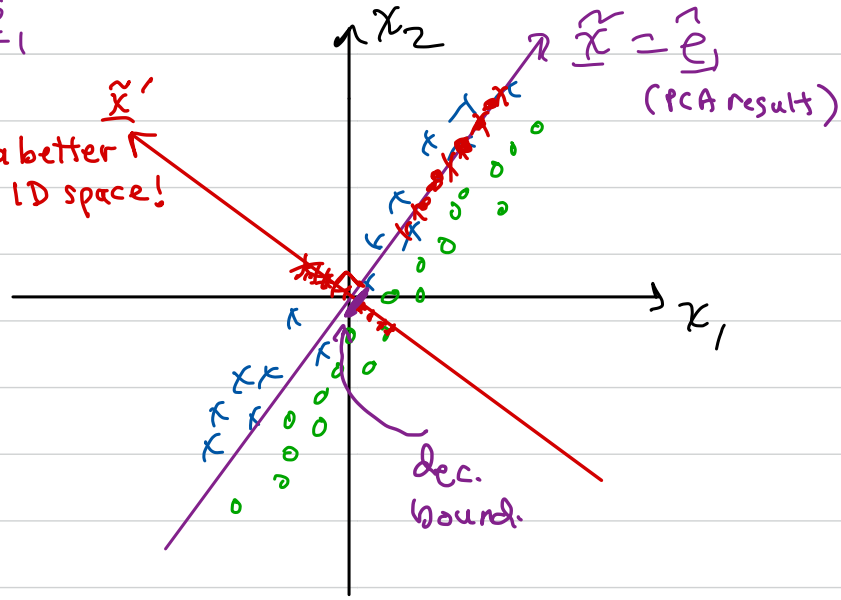


Fig. B

PCA should work well

PCA works poorly.

but, using $\tilde{x}'$ as 1D feature space could work well.