ESS2222

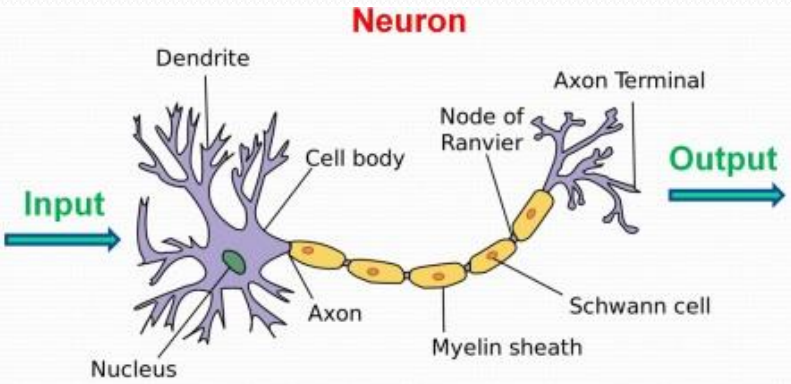
Lecture 3 – Bias-Variance Trade-off

Hosein Shahnas

University of Toronto, Department of Earth Sciences,

Outline

- Bias-Variance Trade-off
- Overfitting & Regularization
- Ridge & Lasso Regression
- Nonlinear Transformation
- Cross-Validation
- Support Vector Machine



Review of Lecture 2

What does ν say about μ?

For a sample of size N, ν is probably close to μ (within ϵ).

$\mathbf{P}[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$ Hoeffding's inequality

$\mathbf{P}[|E_{in}(g) - E_{out}(g)| > \epsilon] \leq 2Me^{-2\epsilon^2 N}$ (Generalized)

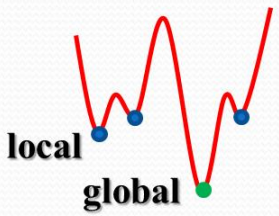
bad event

Stochastic gradient decent (SGD) method,

$\mathbf{w} = \mathbf{w} + \Delta w$

$\Delta w = -\eta \, \Delta J(w)$

$\Delta wj = -\eta \frac{\partial J}{\partial w_j} = \eta \left(y^i - \phi(z^i)\right) x^i_j$ Based on random sampl..



a) Error is noisier , **b)** Convergence faster, **c)** Local minima can be escaped faster

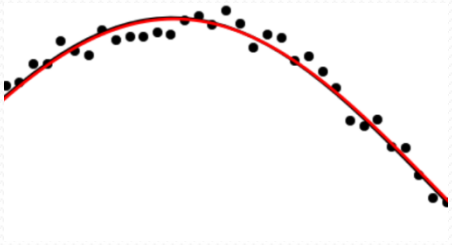
Scaling is important for optimal performance,

Bias-Variance Trade-off

Trade-off Between Approximation and Generalization:

A common issue in machine learning is overfitting, which occurs when the model is complex or the model not only captures the signal but also the noise in a dataset. In order to generalize the model to out-of-sample we have to avoid overfitting.

This is the difference between approximation (fitting) and learning (generalization).



Regularization:

Regularization is a powerful method for reducing overfitting. A good bias-variance trade-off can be obtained by tuning the complexity of the model via regularization. This can be achieved by introducing a penalty (bias) term for model complexity.

Bias-Variance Trade-off

Trade-off Between Approximation and Generalization:

Our goal: Small E_{out} → good approx. of *f* out of sample

However:

More complex *H* → better chance of approximating *f* in sample
Less complex *H* → better chance of generalizing *f* out of sample

The best hypothesis is within the hypothesis set *H*. But the only way to navigate through this set to find the good candidate (g) is via the samples (to find the performance of one hypothesis versus another).

Error

$$E_{out}(g^D) = E_X\left[(g^D(x) - f(x))^2\right]$$

$$E_D\left[E_{out}(g^D)\right] = ?$$

It can be shown that:

$$E_D\left[E_{out}(g^D)\right] = \underbrace{E_X\left[E_D\left[\left[(g^D(x) - \bar{g}(x))^2\right]\right]\right]}_{\text{var}} + \underbrace{E_X\left[\left[(\bar{g}(x) - f(x))^2\right]\right]}_{\text{bias}}$$

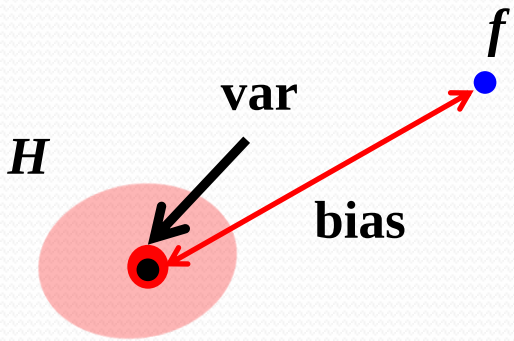
where $\bar{g}(x) = E_D\left[g^D(x)\right] \approx \frac{1}{k}\sum_k g^{D_k}(x)$ **D: data**

: Expected value over D (different data)

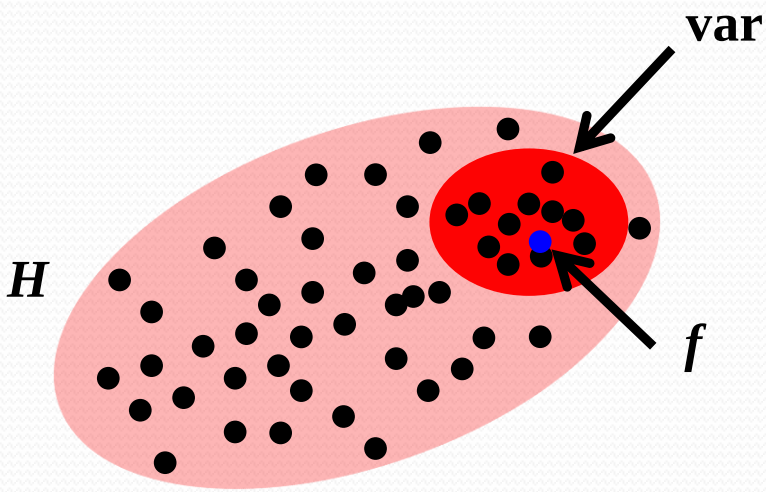
The Trade-off

$$bias = E_X\left[\left[(\bar{g}(x) - f(x))^2\right]\right]$$

$$variance = E_X\left[E_D\left[\left[(g^D(x) - \bar{g}(x))^2\right]\right]\right]$$



small *H*



large *H (and complex)*

So as we go from small H to large H:

H↑ bias↓ var↑

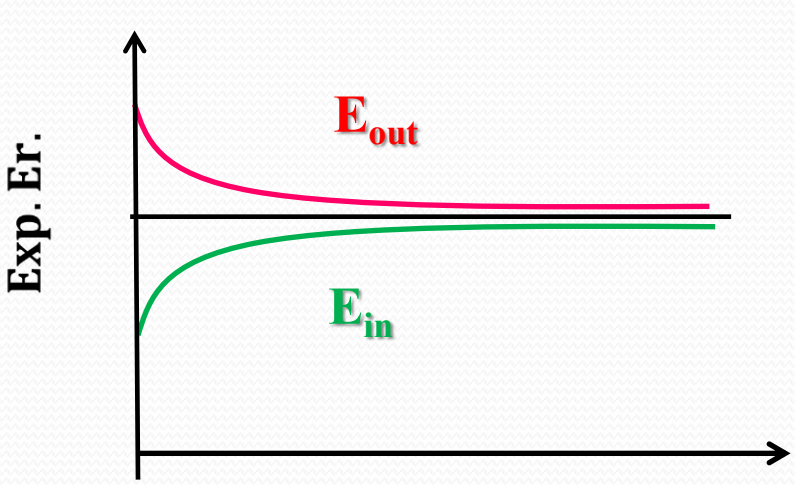
Expected E_{out} and E_{in}

Consider data set D of size N:

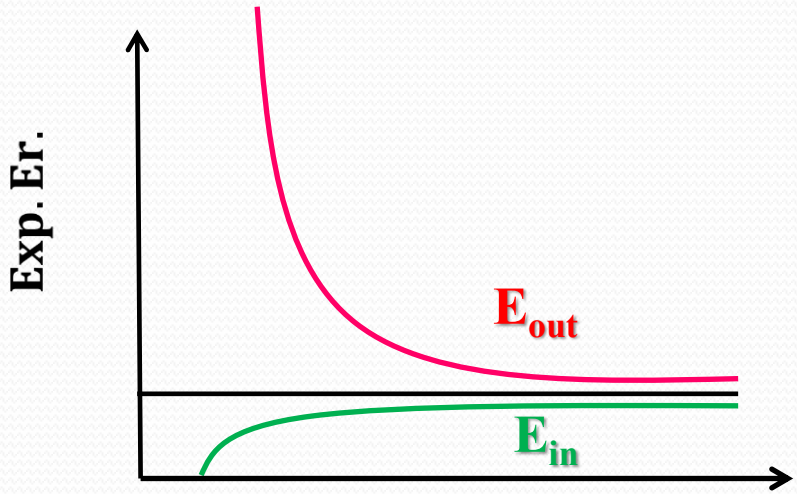
$E_D\,[E_{out}(gD)]$ **Expected out-of-sample error**

$E_D\,[E_{in}(gD)]$ **Expected in-sample error**

How do they vary with N?



Simple model (*H* small)



Complex model (*H* large)

Bias-Variance Trade-off
Example

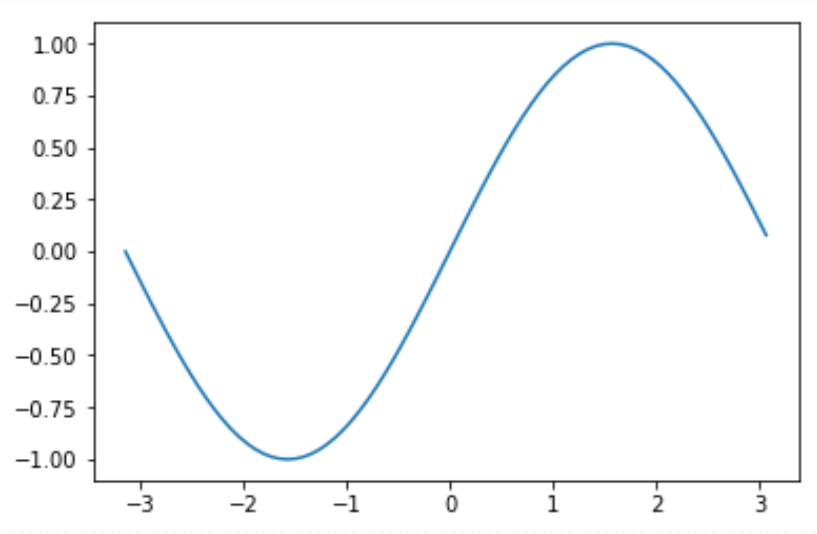
F(x) = Sin(2πx)

Suppose we have two samples! N = 2
We try two models (hypotheses):

H_0: h(x) = b simple
H_1: h(x) = ax + b complex

Which one is better, H_0 or H_1?

Better in what?



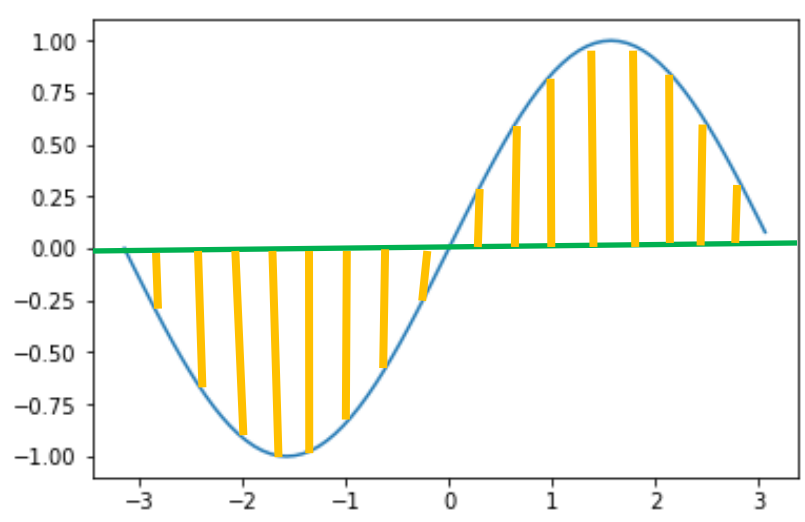
Approximation - H_0 vs H_1

$F(x) = \sin(2\pi x)$

Error

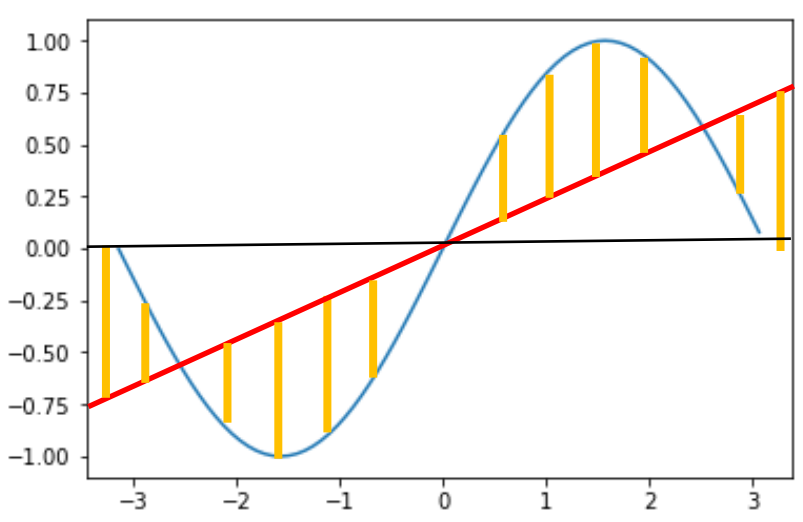
Approximate F(x) with H_0 and H_1

H_0



$E_{out} = 0.5$

H_1

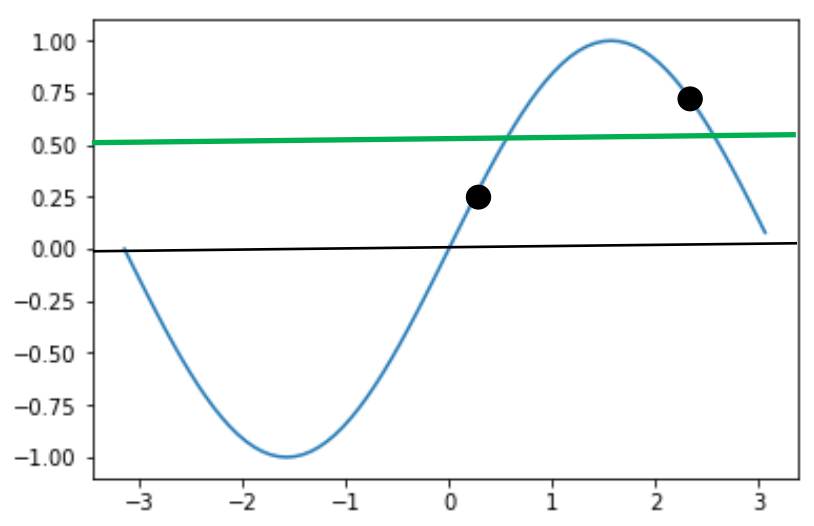


$E_{out} = 0.20$

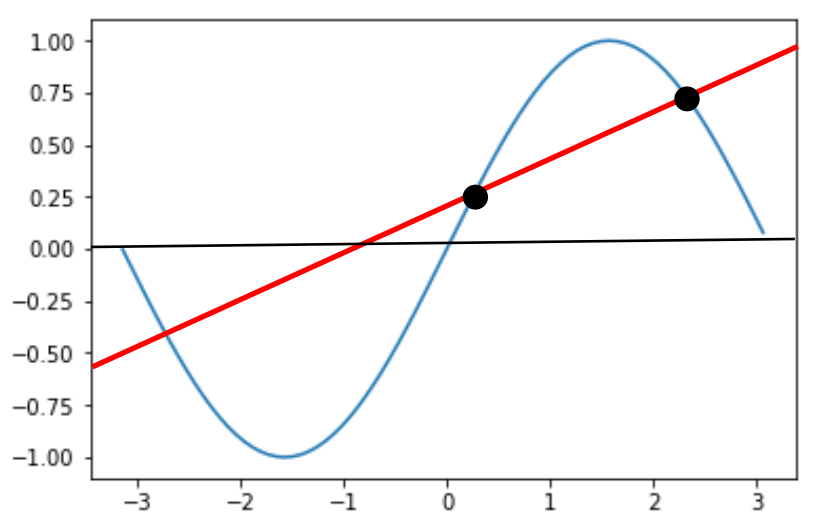
Learning - H_0 vs H_1

$F(x) = \sin(2\pi x)$

H_0



H_1

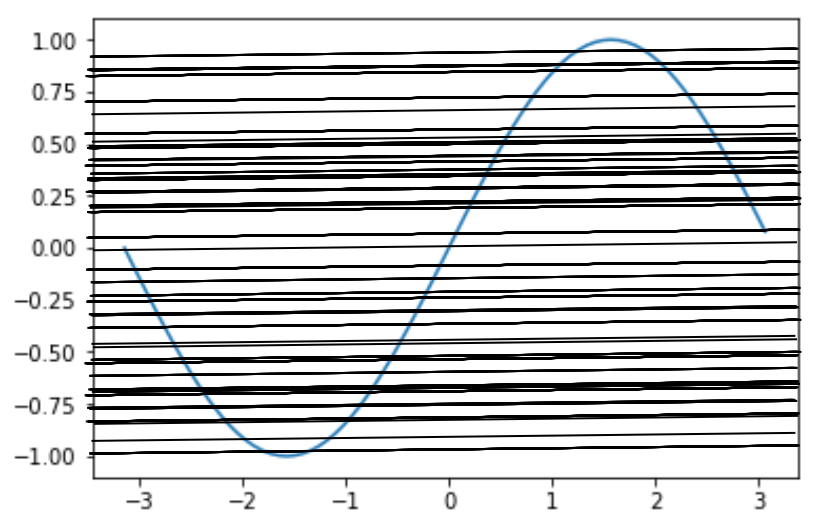


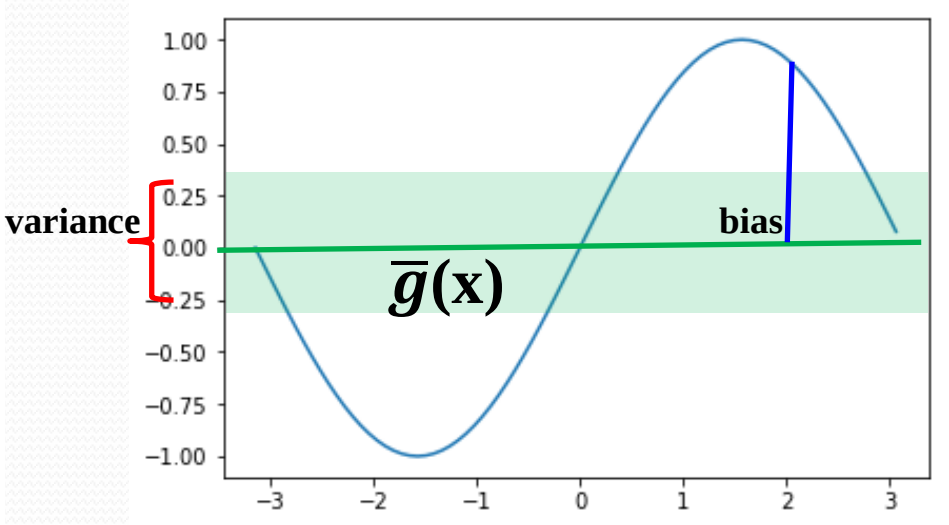
Final hypothesis using two samples

Bias and Variance - H_0

F(x) = Sin(2πx)

$$\bar{g}(x) \approx \frac{1}{k}\sum_k g^{D_k}(x)$$ **D: Sum over different data**



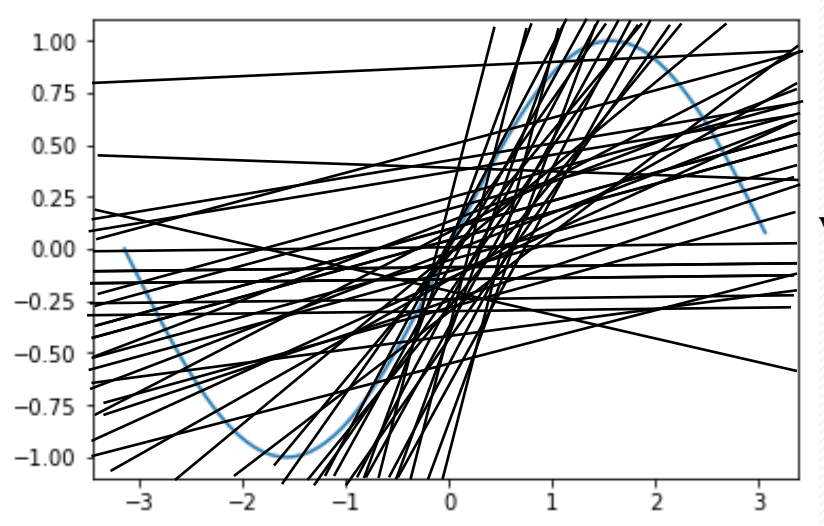


Note that $\bar{g}(x)$ is not the output of learning. The output of learning is one of these lines based on given samples.

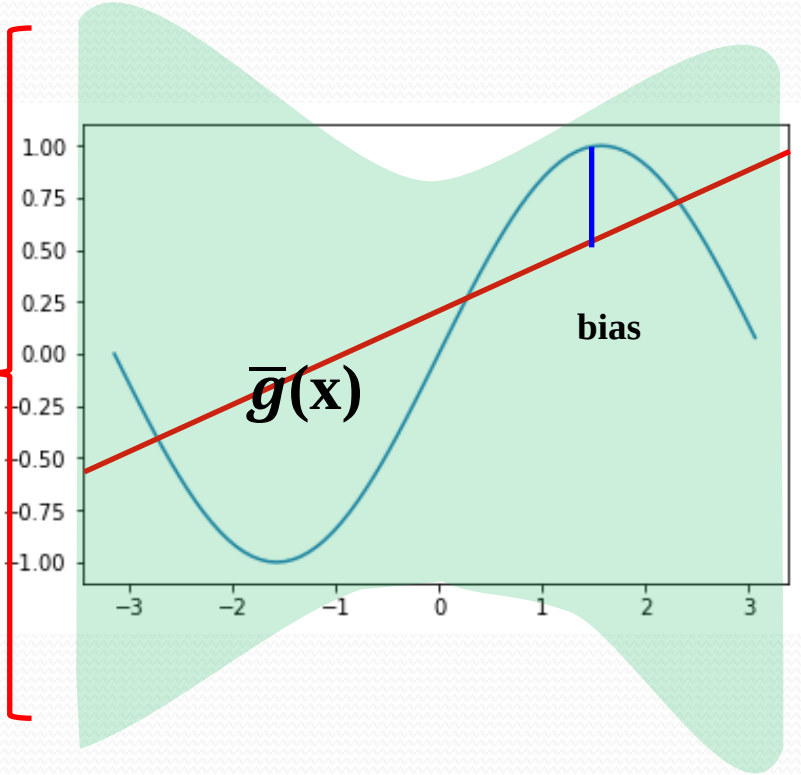
Bias and Variance - H_1

$F(x) = \sin(2\pi x)$

$\bar{g}(x) \approx \frac{1}{k}\sum_k g^{D_k}(x)$ D: Sum over different data



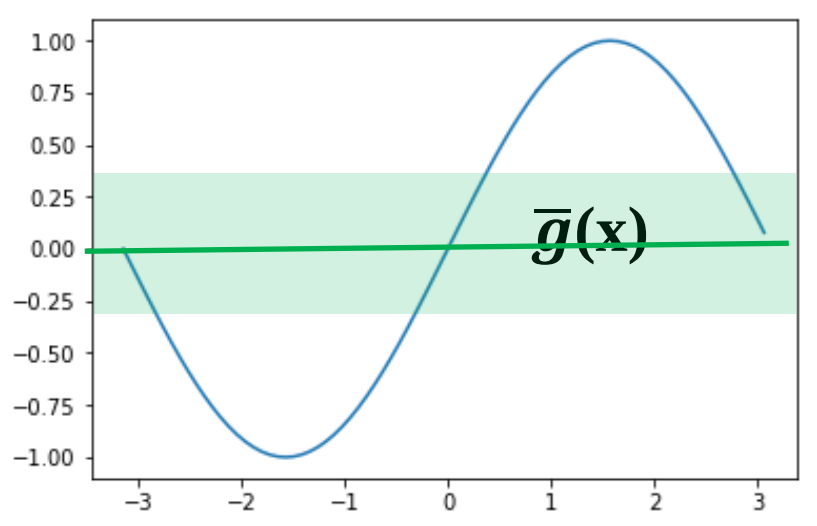
variance



H_0 vs H_1

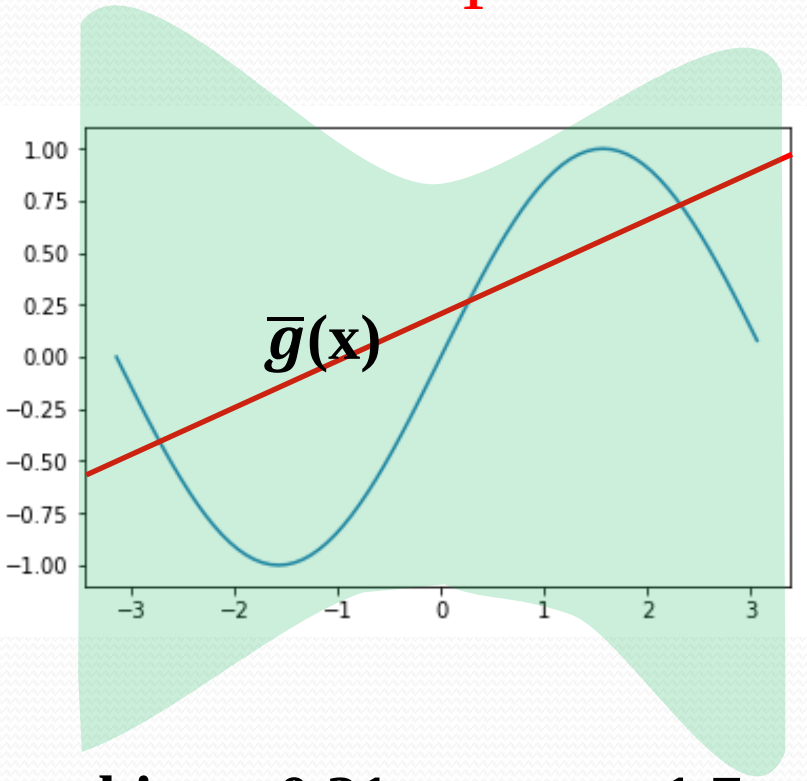
$F(x) = \sin(2\pi x)$

H_0



bias = 0.5 var = 0.25

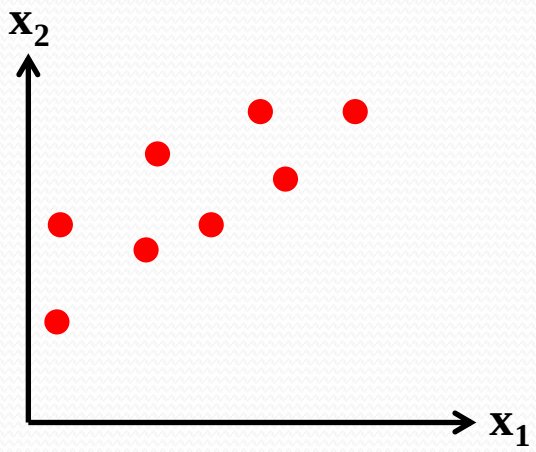
H_1

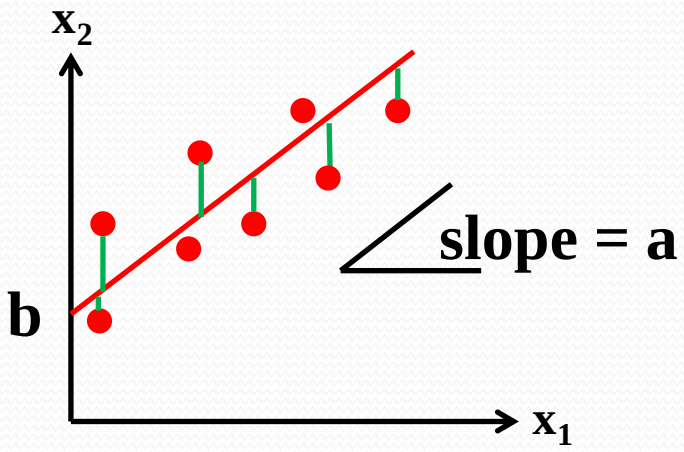


bias = 0.21 var = 1.7

Ridge Regression
An Example

Suppose we want to train a model using a number of the training samples.

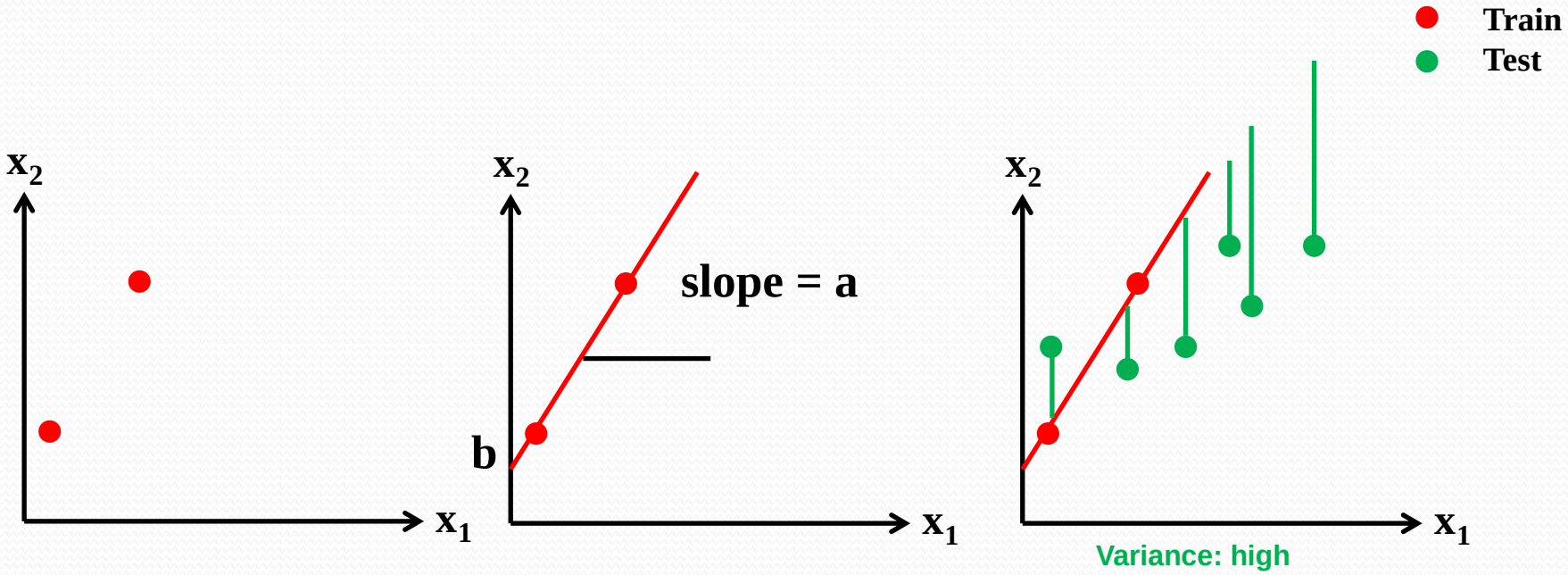




$x_2 = a\,x_1 + b \quad (a \equiv w_1, b \equiv w_0)$

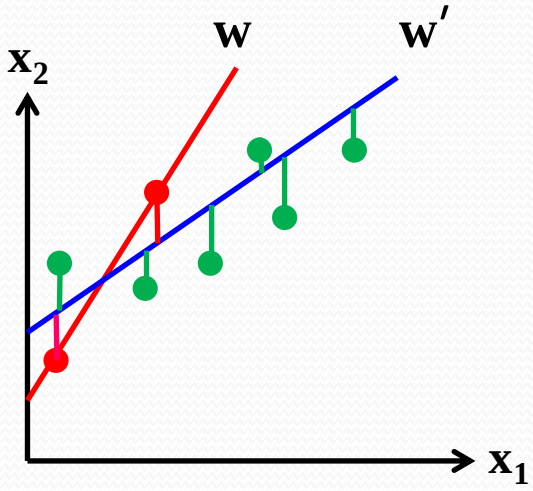
Ridge Regression
An Example

Now suppose we have only two samples for training.
Chose two of them as training samples and the rest as test samples.



High variance: The learned model overfits to the training data.

Overfitting & Regularization
Ridge Regression



$$J(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2$$

$$J'(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2 + \underbrace{\frac{\lambda}{2}\sum_i w^2{}_i}_{\text{Bias term}}$$

Train
Test

The idea behind the ridge regression is that to find a new line which doesn't fit to the training data very tightly.

Adding a small bias can highly decrease the variance.

Slightly worse fit, better generalization.

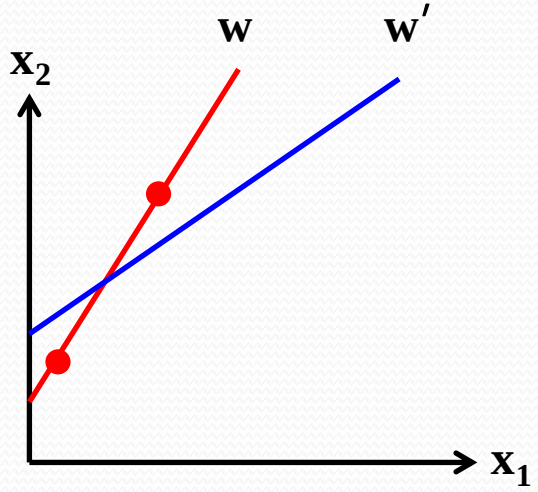
Ridge Regression
An Example

Suppose Φ(z) = z Adeline

$z = x.w = \sum_0^n w_i x_i$

then: Φ(z) = $x.w = \sum_0^n w_i x_i$ *activation func.*

$\Phi(z) = w_1 x_1 + w_0$



$$J(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2$$

$$J'(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2 + \frac{\lambda}{2}\sum_i w^2{}_i$$

$$J(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2 = J(w) = \frac{1}{2}[l_{x1} - w_1 x_1 - w_0]^2 + [l_{x2} - w_1 x_2 - w_0]^2$$

$$J'(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2 = J(w) = \frac{1}{2}[l_{x1} - w_1 x_1 - w_0]^2 + [l_{x2} - w_1 x_2 - w_0]^2 + \frac{\lambda}{2}\left[w^2{}_0 + w^2{}_1\right]$$

l_{xi} : label

$\frac{\partial J'}{\partial w_0}, \frac{\partial J'}{\partial w_1}$

Ridge Regression
An Example

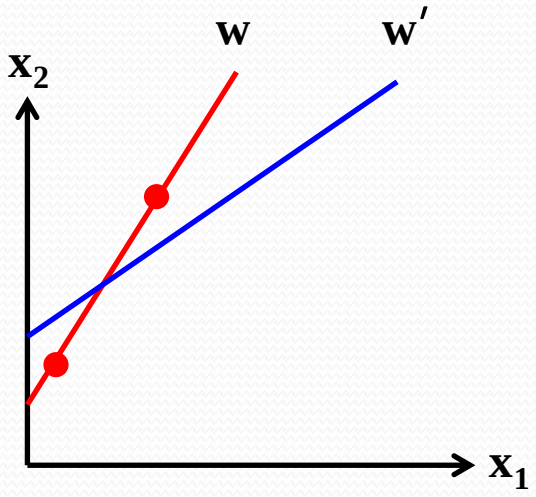
Note that our goal is minimizing the cost function J

With ridge regression penalty:

$$J(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2$$

$$J'(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2 + \frac{\lambda}{2}$$

$J'(w)_{\text{min}} < J'(w)_{\text{min}}$ **for blue line**



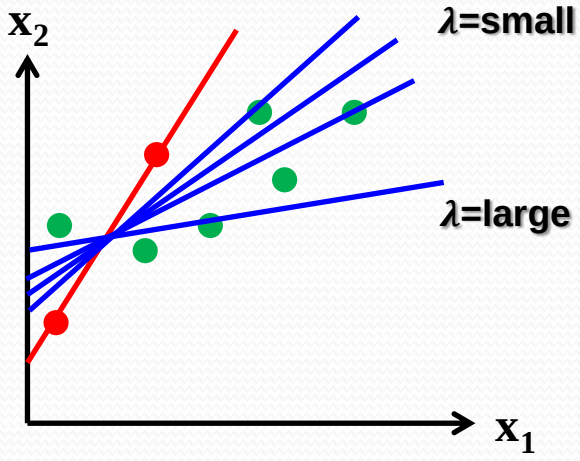
So we chose ridge regression line over least squares line.

Ridge Regression
λ -parameter

$0 \leq \lambda < \infty$

How to chose lambda?

We try different values for λ and use cross-Validation, typically 10-fold cross-validation to determine which one results in the lowest variance.



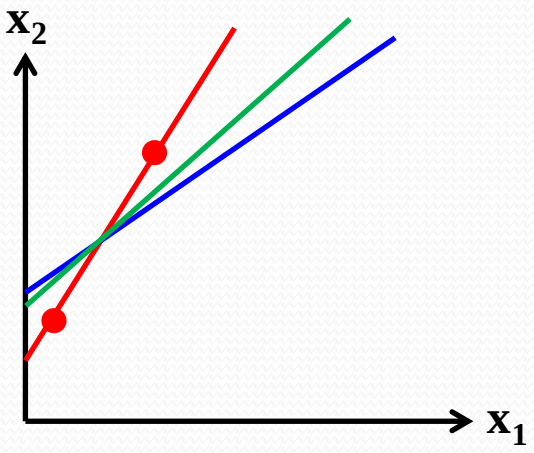
Cross-Validation

16%	16%	16%	16%	16%	20%
Training data	Training data	Training data	Training data	Validation	Testing data
Training data	Training data	Training data	Validation	Training data	Testing data
Training data	Training data	Validation	Training data	Training data	Testing data
Training data	Validation	Training data	Training data	Training data	Testing data
Validation	Training data	Training data	Training data	Training data	Testing data

5-fold cross-validation

Overfitting & Regularization
Lasso Regression

Lasso regression is similar to the ridge regression except for an important difference!



L2-rgression (Ridge)

$$J'_r(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2 + \frac{\lambda}{2}\sum_i w^2{}_i$$

L1-rgression (Lasso)

$$J'_l(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2 + \frac{\lambda}{2}\sum_i |w_i|$$

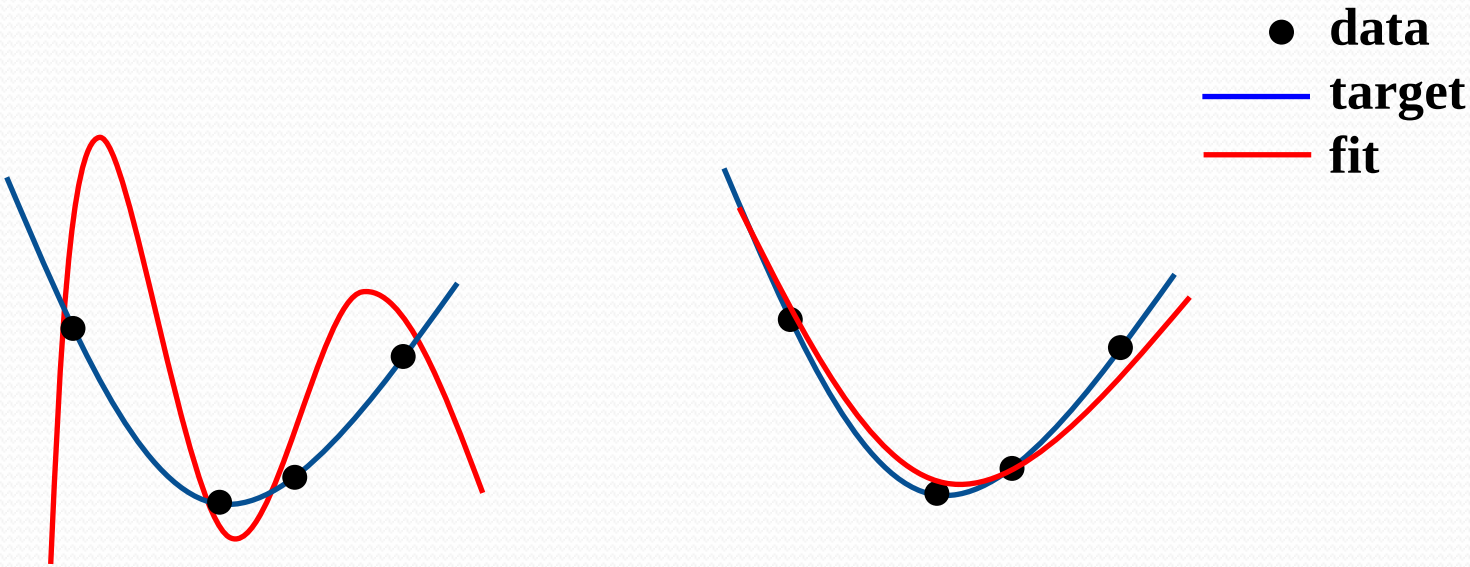
In lasso regression the weight factors of less important (relevant) features shrink faster which is good.

Lesson

When we train a model, we are matching the model complexity to the data resources, not the target complexity (which should be avoided).

When the number of sample data is not large, we have to avoid training complex models.

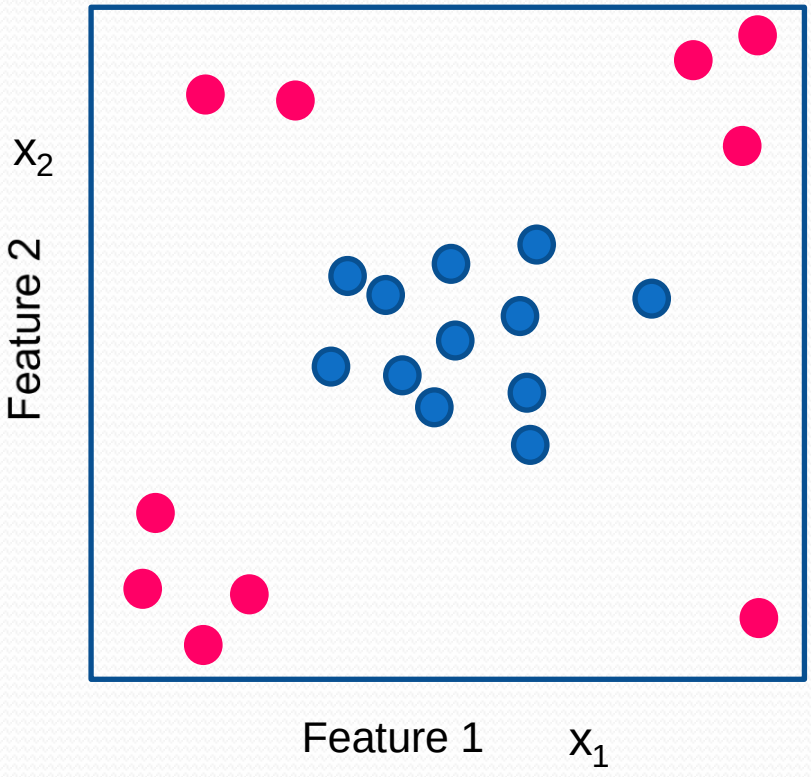
Allowing $E_{in} \rightarrow E_{in} + \in$ (e.g., $\in \sim 1\% E_{in}$) can highly improve the out-of-sample performance (E_{out} and variance).

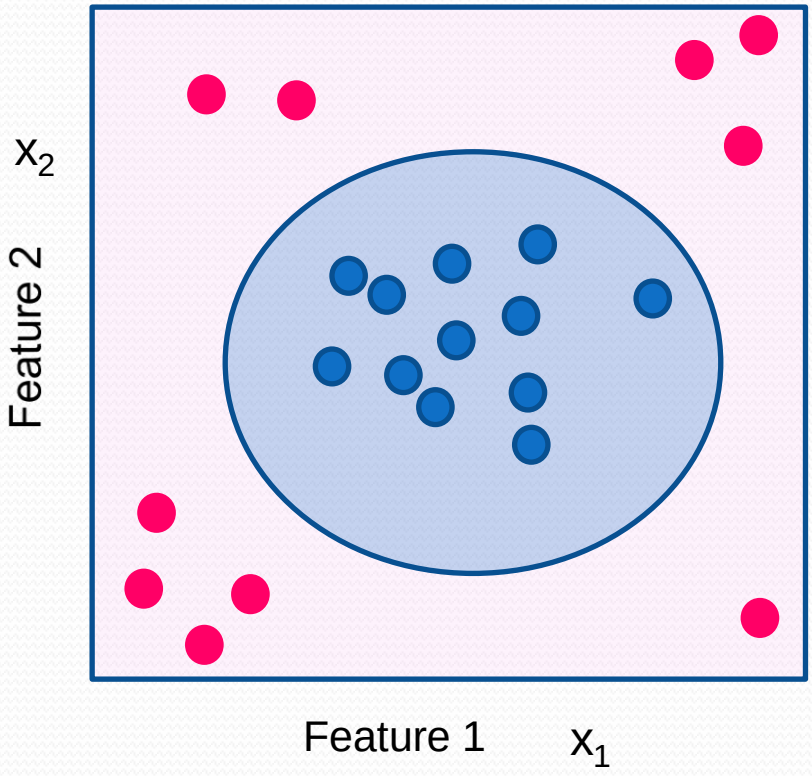


fit **restrained fit**

Nonlinearity

Can we apply linear leaning algorithms for nonlinear problems?
Note that feature is a higher level representation of raw input.





Nonlinearly separable

Nonlinearity

Can we apply linear leaning algorithms to nonlinear problems?

$h(x) = \sum_{i=0}^{m} w_i x_i$ **is linear in both w and x.**

Being linear in w is important because the algorithm works because of linearity in the weights.

We still can use linear algorithm for nonlinear problems.

How?

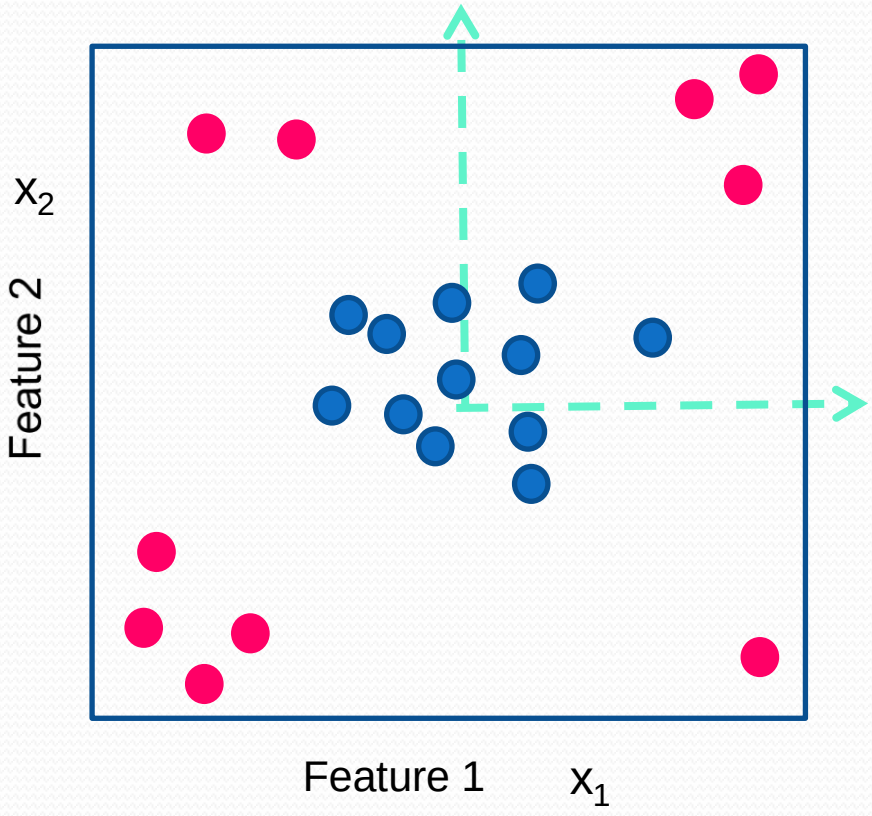
Nonlinearity

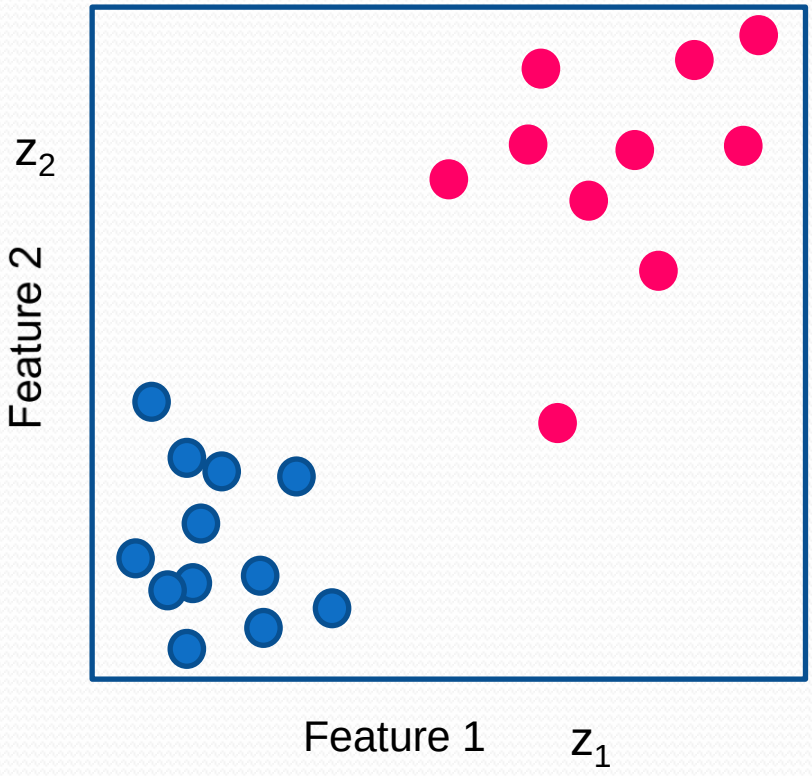
$$x \rightarrow \varphi(x)$$

$$(\mathbf{x_1}, \mathbf{x_2},) \rightarrow (\mathbf{z_1}, \mathbf{z_2}) = (\mathbf{x_1}^2, \mathbf{x_2}^2)$$

$$z_n = \varphi(x_n) \in \boldsymbol{Z}$$

$x_n \in \boldsymbol{X}$





Linearly separable

Nonlinearity

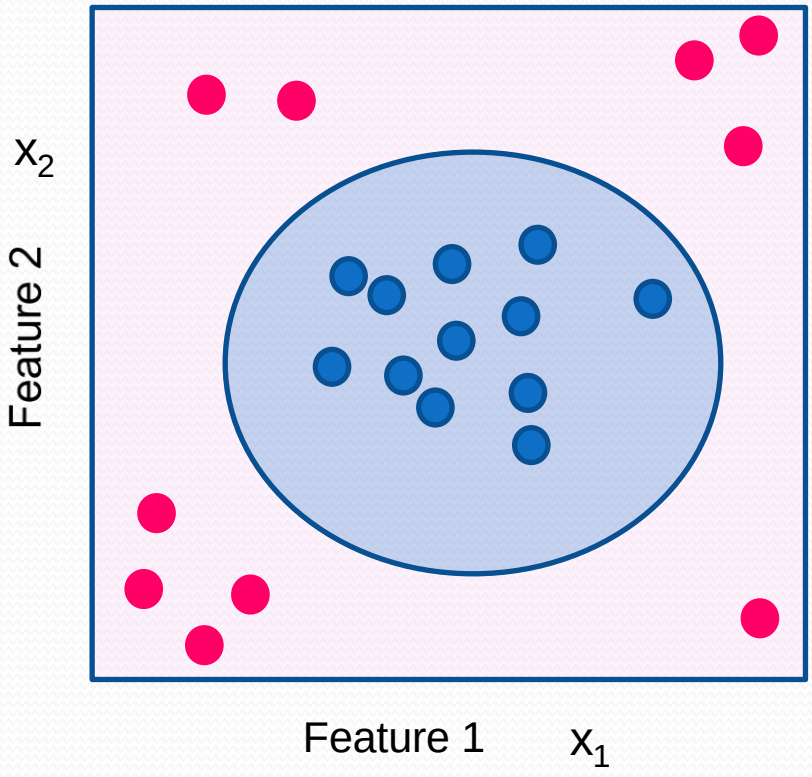
Classify in X-space

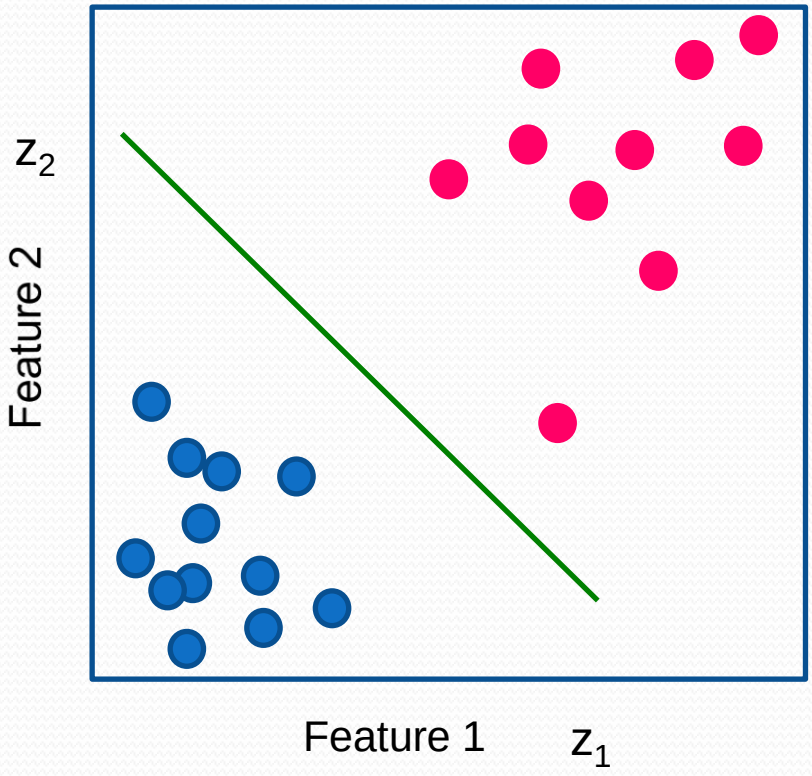
g(x) = $\tilde{g}$ ($\varphi(x)$) = sign($w^T \varphi(x)$)

$x \leftarrow \varphi^{-1}(x)$

Separate in Z-space

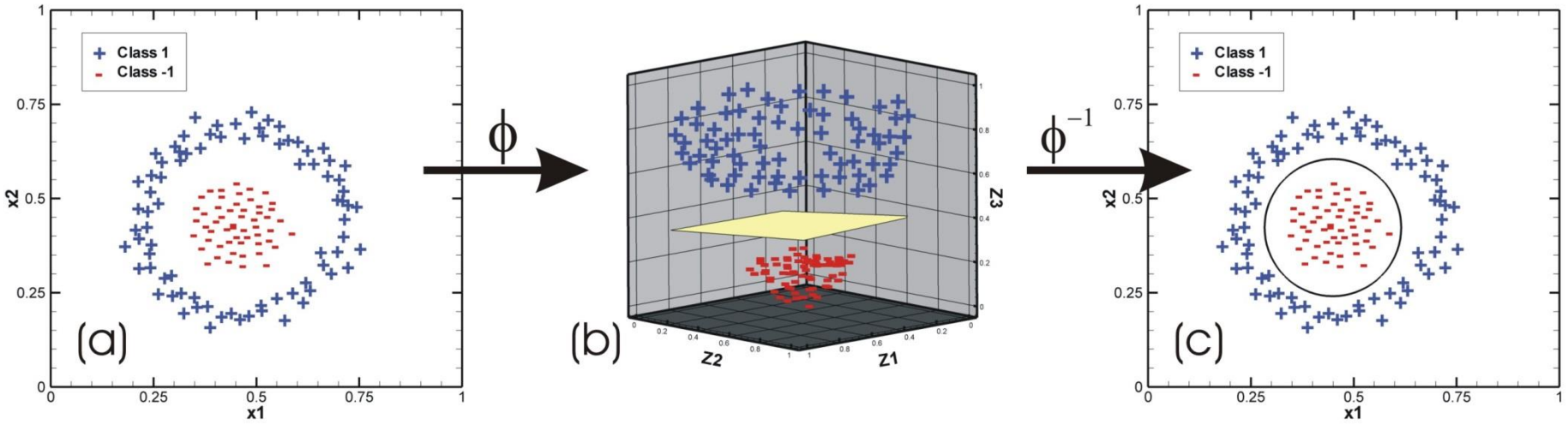
g(z) = sign($\tilde{w}^T z$) = sign($\tilde{w}^T \varphi(x)$))





Nonlinearity

Mapping onto a Higher Dimensional Feature Space



$$\varphi\colon R^2 \to R^3 \qquad x \to \varphi(x) \qquad \mathbf{z} = \mathbf{x}.\mathbf{w} \to \mathbf{z} = \varphi(x).\mathbf{w}$$

$$\varphi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)\ .$$

Nonlinearity - Kernel Methods

$$z = \varphi(x).w$$

$$w = \sum_{i=1}^{m} \alpha_i \varphi(x_i) \text{ Assumption}$$

$$z = \sum_{i=1}^{m} \alpha_i \varphi(x_i).\varphi(x)$$

$$K(x_i, x) = \varphi(x_i).\varphi(x)$$

$$z = \sum_{i=1}^{m} \alpha_i K(x_i, x)$$

$$K(x_i, x) = (x_i, x) \text{ Linear}$$

$$K(x_i, x) = (\gamma(x_i, x) + r)^d \text{ Polynomial}$$

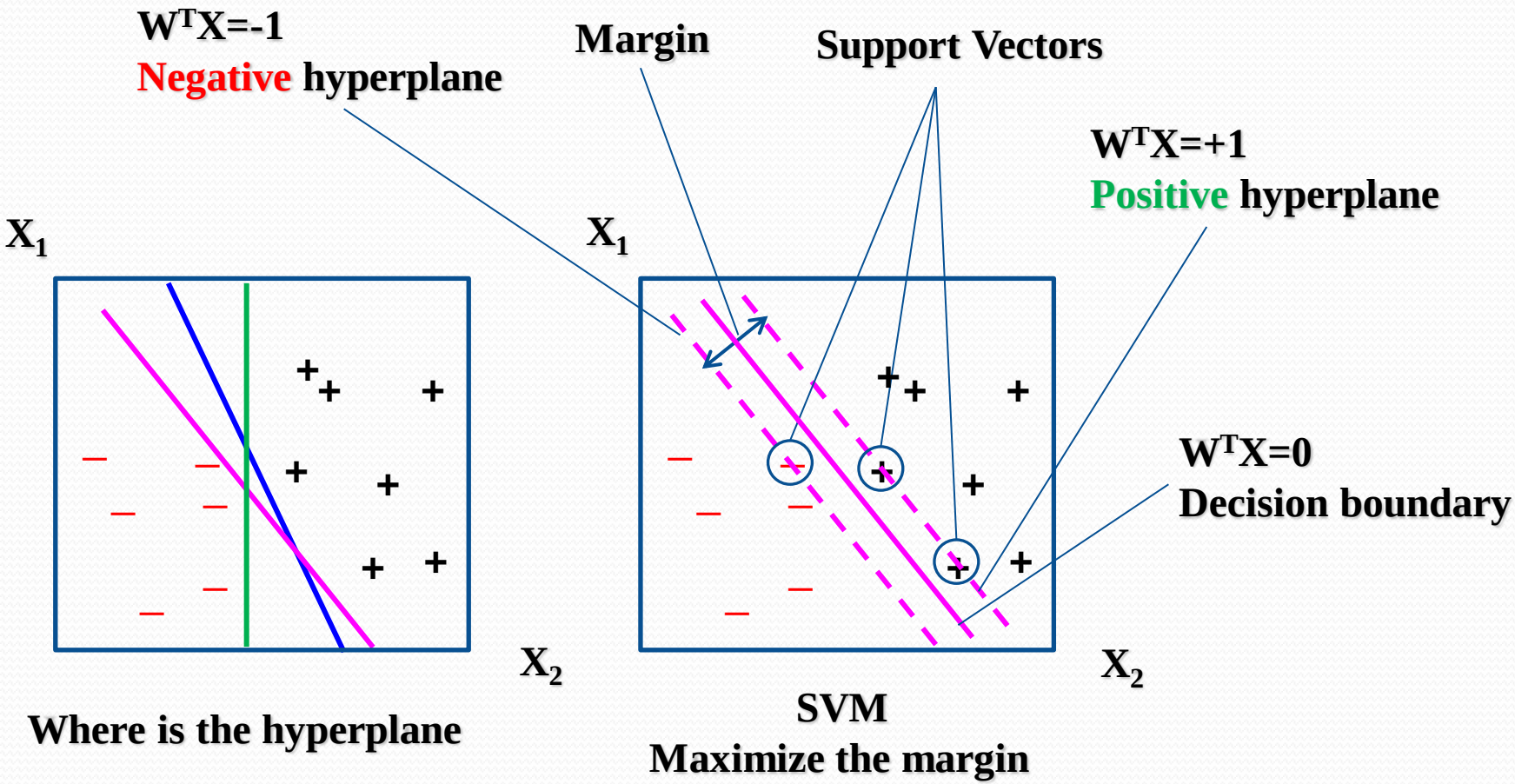
$$K(x_i, x) = Exp(-\gamma \|x_i, -x\|^2) \text{ RBF}$$

$$K(x_i, x) = tanh(\gamma(x_i, x) + r) \text{ Sgmiod}$$

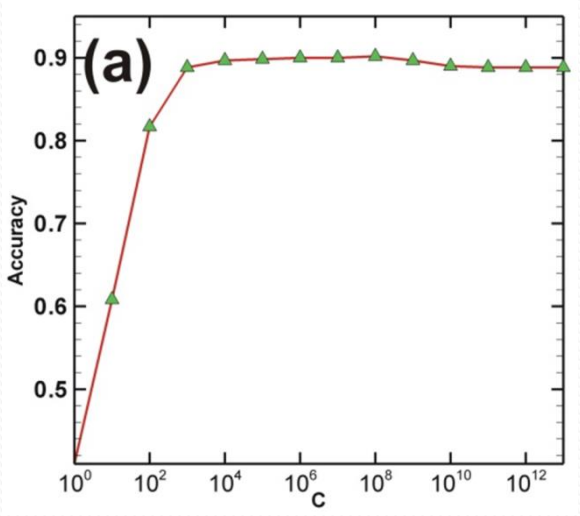
$$k(x^i, x^j) = \exp\left(-\frac{\|x^i - x^j\|^2}{2}\right) = \exp\left(-\frac{1}{2} < x^i - x^j, x^i - x^j >\right) =$$

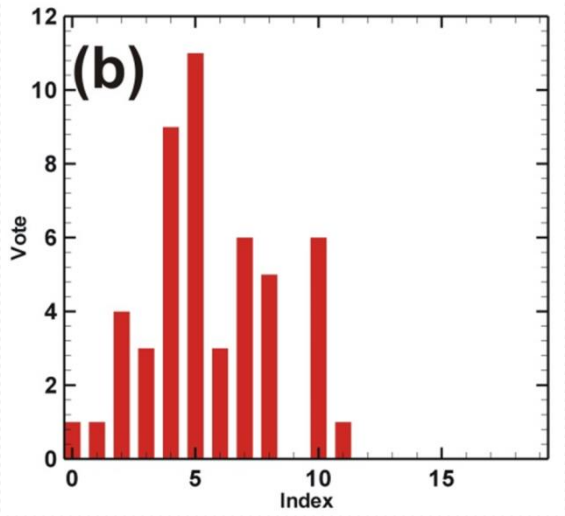
$$\exp\left(-\frac{1}{2}\|x^i\| - \frac{1}{2}\|x^j\|\right) \exp(< x^i, x^j >) = C \sum_{k=0}^{\infty} \frac{< x^i, x^j >^k}{k!},$$

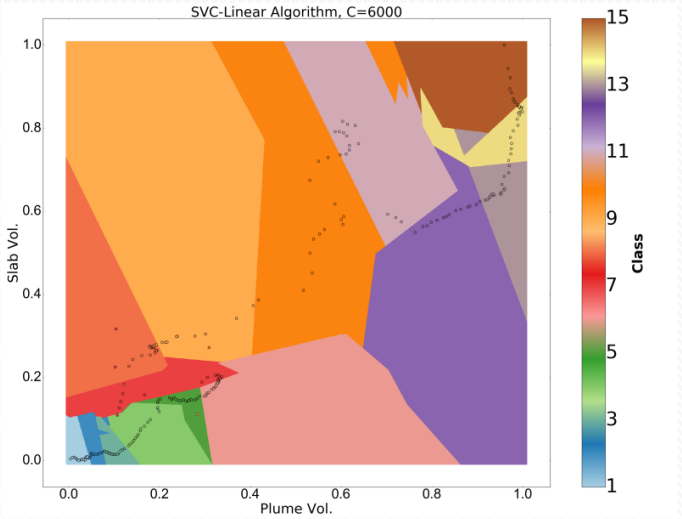
Support Vector Machine (SVM)



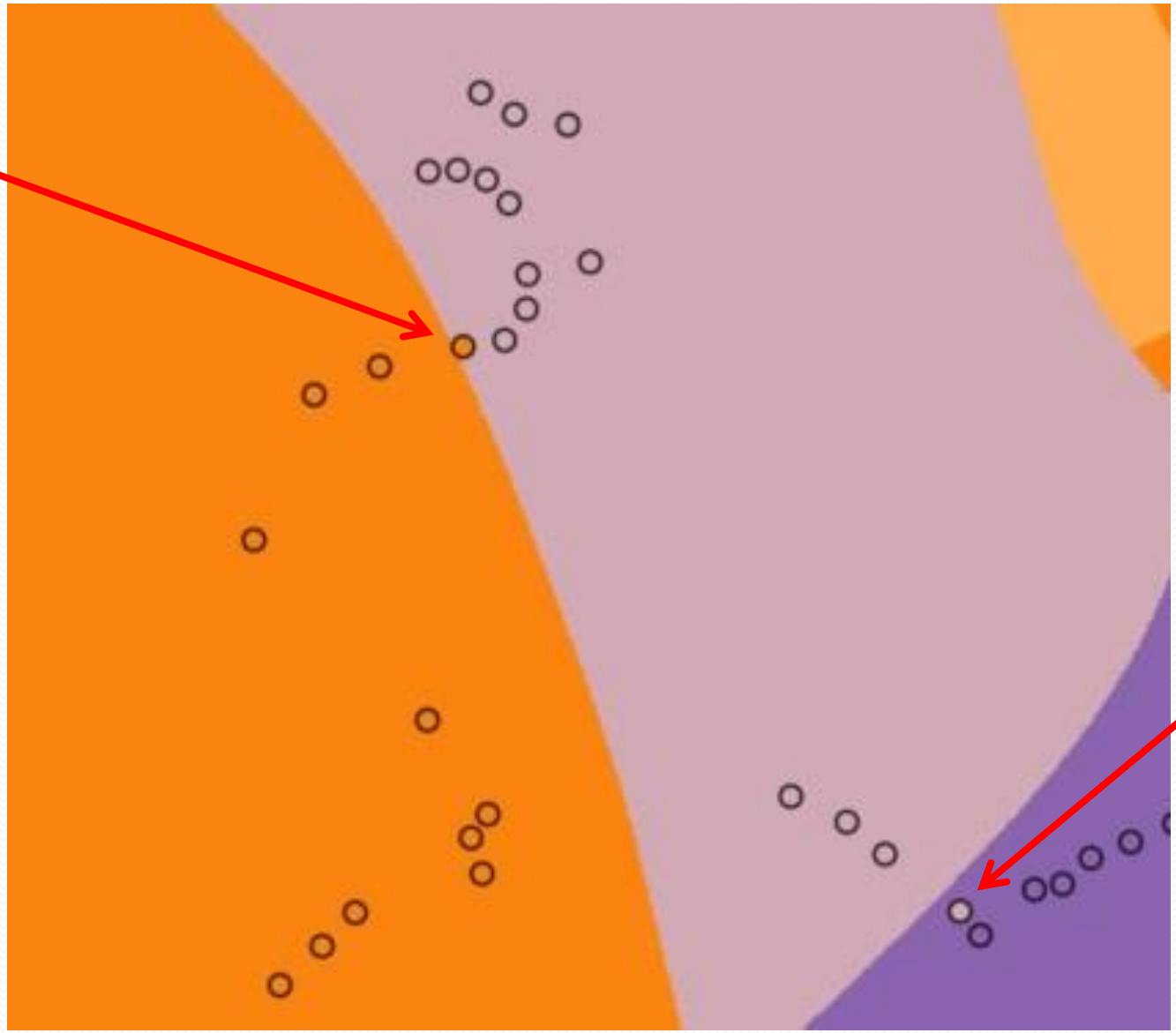
Assignment 1 - SVC-Linear







Assignment 1 - SVC-Linear



Coarse Grid Search – Related to Assignment 1

Cross validator

My_cv = StratifiedShuffleSplit(n_splits=10, test_size=0.2, train_size = None, random_state=19)

StratifiedShuffleSplit: Stratified ShuffleSplit cross-validator
Stratification is the process of rearranging the data as to ensure each fold is a good representative of the whole (each fold comprises around the same fraction of classes).

My_cv: user defined cross-validator
n_splits: number of splits (default = 10)
test_size: default = 0.1 if train_size is unspecified, otherwise it will complement the specified train_size. It should be specified if train_size = None.
random_state: seed for random shuffling

Grid search

grid = GridSearchCV(SVC(kernel='linear'), param_grid=grid_parameters, cv=My_cv, return_train_score=False)

GridSearchCV implements a “fit” and a “score” method. It also implements “predict”.

SVC(kernel='linear'): estimator
param_grid: grid parameters
cv: cross validator
return_train_score: if False, the cv_results_ attribute will not include training scores

grid.fit(feature_data, class_labels)

Fine Grid Search – Related to Assignment 1

Cross validator

```
My_cv = KFold(n_splits=splits_num, shuffle=True, random_state=i)

My_svm = svm.SVC(kernel='linear')  # linear kernel
```

Grid search

```
grid = GridSearchCV(estimator=My_svm, param_grid=p_grid, cv=My_cv, return_train_score = False)

My_svm = svm.SVC(kernel='linear')

grid.fit(feature_data, class_labels)

grid.best_score_
grid.best_index_
```

Learning Model – Related to Assignment 1

Cross validator

```
X = feature_train
y = class_labels_train
my_C = 6000.0  # SVM regularization parameter
svm_SVC_lin = svm.SVC(kernel='linear', C=my_C)
svm_SVC_lin.fit(X, y)

grid.best_score_
grid.best_index_
```