Linear Models: Regression

DS 392 JAN2022

Deepak Subramani
Assistant Professor
Dept. of Computational and Data Science
Indian Institute of Science Bengaluru

Department of Computational and Data Sciences

INDIAN INSTITUTE OF SCIENCE
भारतीय विज्ञान संस्थान

$h_\theta = X\theta$

θ ① Normal Eqn

$\rightarrow Nu = k \cdot Pr^{\bar{a}} Re^{\bar{b}}$

$\theta_0 \quad \theta_1$

$\ln Nu = \ln k + a \ln Pr + b \ln Re$

θ_2

Feature x_1 x_2 y

x_1	x_2	y			
Pr	Re	Nu	$\ln Pr$	$\ln Re$	$\ln Nu$

$$Nu = e^{\theta_0} \cdot Pr^{\theta_1} \cdot Re^{\theta_2}$$

Feature Engineering

- The same ideas that we have discussed so far can be used even when the input-output relationship is nonlinear
- The key property of our Linear Model so far is that the loss function that is minimized (the obj. function) is linear in the parameter θ (the decision variable)
- So we can use a model such as

$$h(\boldsymbol{x};\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2$$

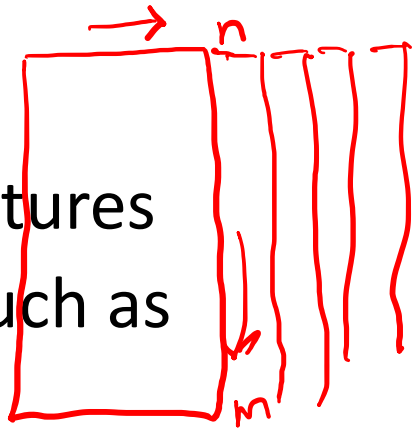
- Here the new polynomial features added are part of our feature engineering process
- It can be anything, really!

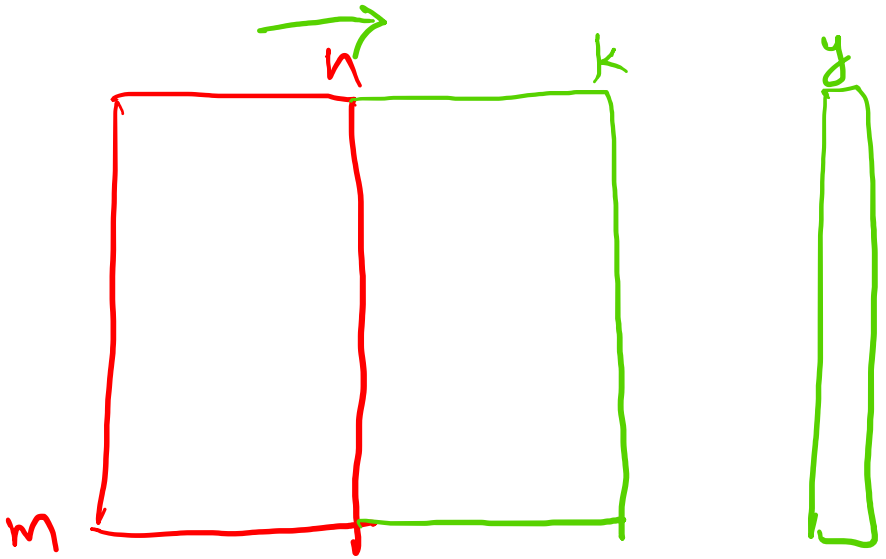
$$h(\boldsymbol{x};\theta) = \theta_0 + \theta_1 \frac{x_1}{x_2} + \theta_2 \frac{x_2 x_3}{x_1^2} + \theta_3 x_1 x_2$$

Polynomial Regression - sklearn

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

- X_poly now contains the original features, its square and all pair-wise products as well.
  - Beware of the combinatorial explosion of features!
- Now we can do LinearRegression() with the engineered features
- You can transform data to any other engineered features such as trigonometry, logarithm, etc





$$\theta^* = (X^T X)^{-1} X^T y$$

$k = m$

$X_{m \times m}$

$$\theta^* = \underbrace{(X^T X)^{-1} X^T}_{\downarrow} y$$

$$\theta^* = X^{-1} y$$

$$(X^T X)^{-1} = X^{-1} \underbrace{(X^T)^{-1} X^T}_{} y$$

$$= X^{-1} y$$

$$\hat{y} = X\theta^*$$

$$= X X^{-1} y$$

$$= y$$

Training error = 0

[Complex Model
Large no: of parameters]

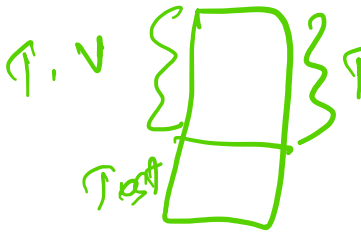
Goodness of Fit

- How do we know that the model that we have is good?
- For Linear Regression problems, traditionally a figure called "Coefficient of Determination" or R^2 is used.
- A linear model explains one variable (the output) in terms of other variables (features/regressors)
- R^2 measures how much explanatory power the linear model has compared to just the natural variability in the data
- $R^2 = 1 - \frac{\sum_j (y_j - \widehat{y_j})^2}{\sum_j (y_j - \bar{y})^2} = 1 - \frac{sum\ of\ squared\ error}{sum\ of\ squared\ deviations\ from\ mean}$
- When error=0, $R^2 = 1$, and when predictions are just the mean, $R^2 = 0$

Problems with R^2

- Can be used only with OLS regression
- High R^2 doesn't necessarily mean a good predictive model and low doesn't mean a bad predictive model
- Just increasing the regressors will increase R^2
- Heavy duty feature engineering, especially using high dimensional polynomials will lead to overfitting the training data and high R^2
- We must have better ways to estimate goodness when other loss functions are used and have a scheme of determining the performance of the data-driven model

Overfit or Underfit?

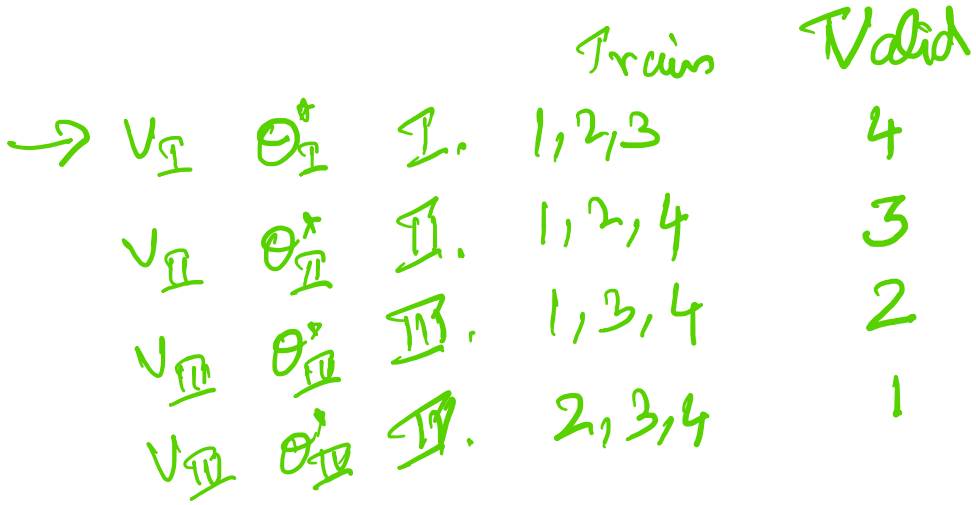


- Question: How to know if you are overfitting or underfitting?
- Answer 1: Cross-Validation
  - If a model performs well on the training data but generalizes poorly according to the cross-validation metrics, then your model is overfitting.
  - If it performs poorly on both, then it is underfitting.
- Answer 2: Learning Curves
  - Plots of the model's performance on the training set and the validation set as a function of the training set size (or the training iteration)
  - Train the model several times on different sized subsets of the training set

Cross-Validation

- Cross-Validation involves using the train set to construct multiple train-validate splits and perform the model building/training activity
- Test set should not be used for model building purpose

- K Fold Cross Validation
  - from sklearn.model_selection import cross_val_score
  - scores = cross_val_score(tree_reg, housing_prepared, housing_labels, scoring="neg_mean_squared_error", cv=10)
  - tree_rmse_scores = np.sqrt(-scores)



- GridSearchCV for Hyper Parameter Tuning

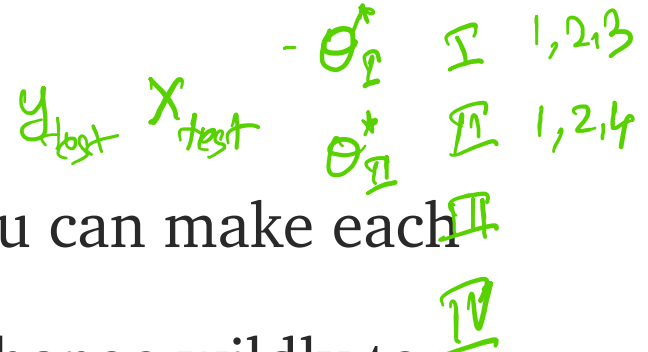
Bias/Variance Trade-off

- A model's generalization error can be expressed as the sum of three very different errors:
  - *Bias:* due to wrong assumptions, such as assuming that the data is linear when it is quadratic.
    - A high-bias model likely underfits the training data.
    - Note: This bias is not the bias-term/intercept.
  - *Variance*: excessive sensitivity to small variations in the training data.
    - A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance and thus overfit the training data.
  - *Irreducible Error*: Due to inherent noise in the data
- Bias-Variance tradeoff
  - Increasing a model's complexity will increase its variance and reduce its bias.
  - Reducing a model's complexity will reduce its variance and increase its bias

Deepak Subramani, deepakns@iisc.ac.in

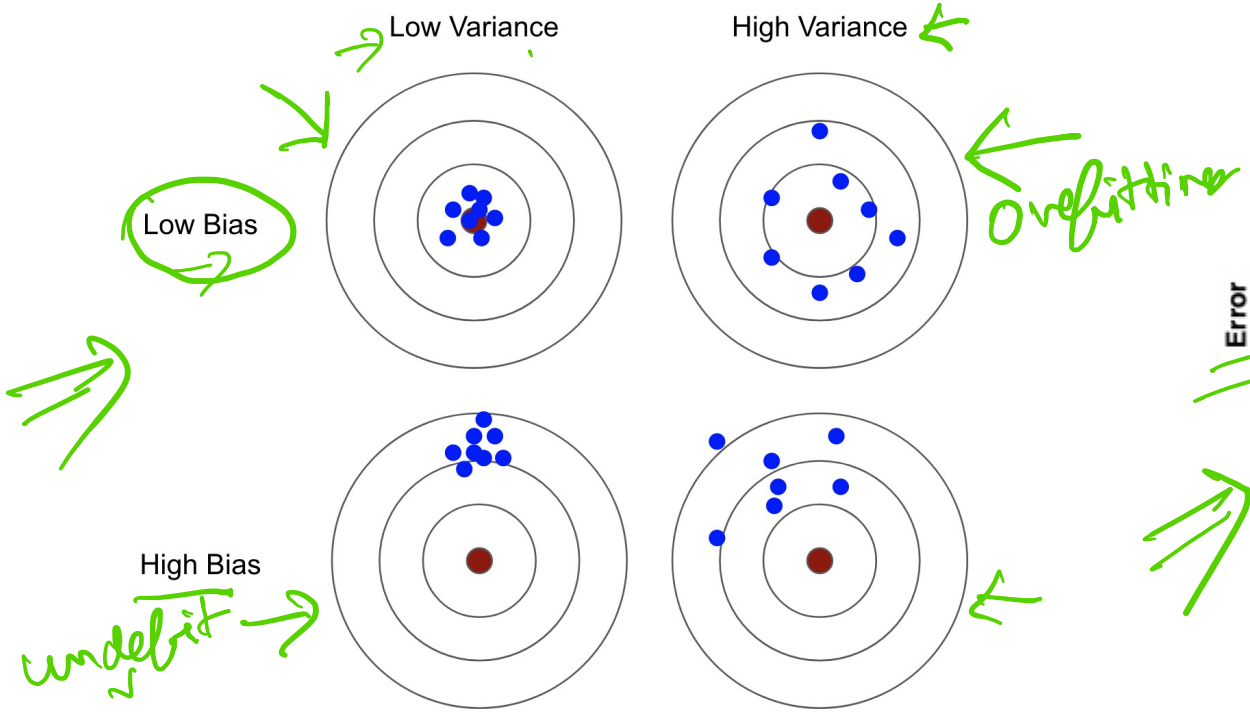
Bias-Variance: Revisit

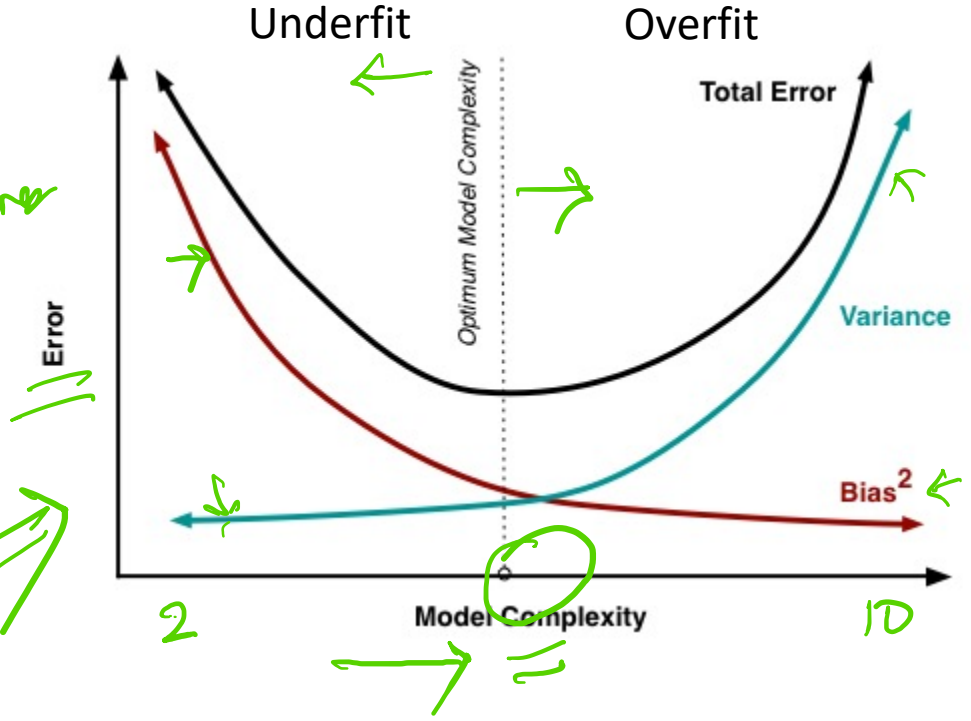
- Bias: Error from erroneous assumptions in the model
  - High bias -> underfitting
  - Model has limited flexibility to learn
  - Analogy: Overly simplistic assumptions about people make you a biased person
- Variance: Error from sensitivity of model to small perturbations in training data
  - High variance -> overfitting
  - Model has too much flexibility to learn
  - If you have a lot of data and over complex model, you can make each parameter of the model "by-heart" one data point
  - When a new data point comes, then the model will change wildly to accommodate the new point



Bias/Variance Equation

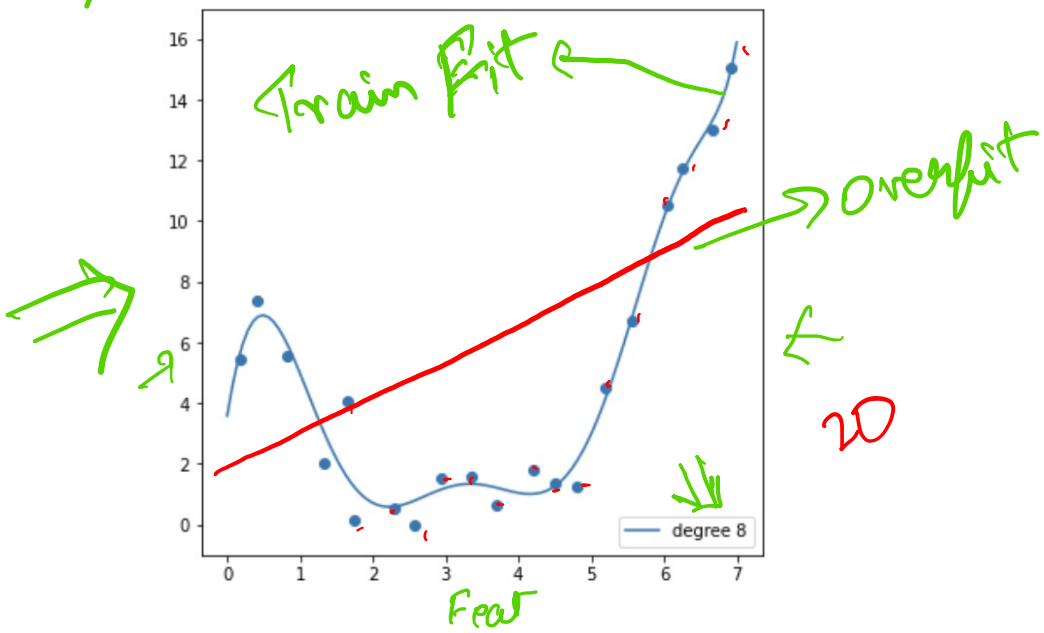
- $Error = E[(y - \hat{y})^2]$ –> RMSE
- $Error = (E[\hat{y}] - y)^2 + E[(\hat{y} - E[\hat{y}])^2] + \sigma_e^2$
- Error = Bias^2 + Variance + Irreducible Error

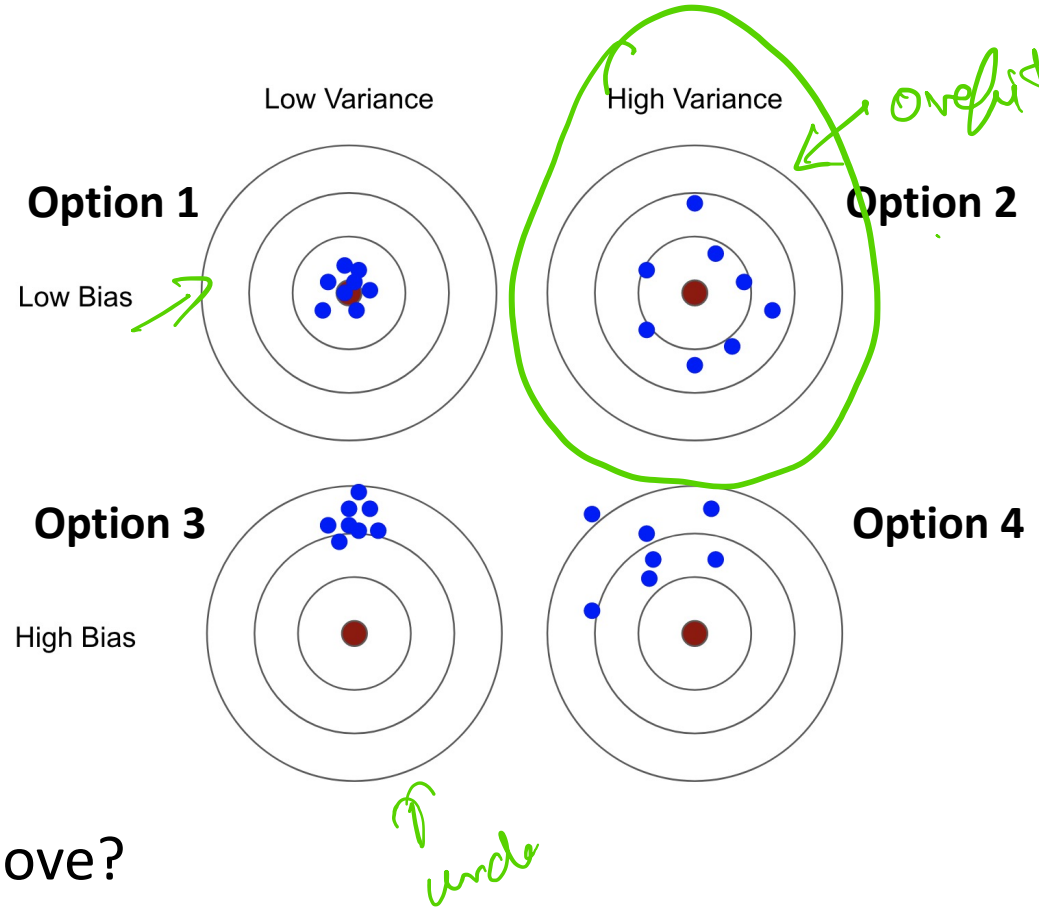




Audience Poll

Question 1: What condition is this fit?





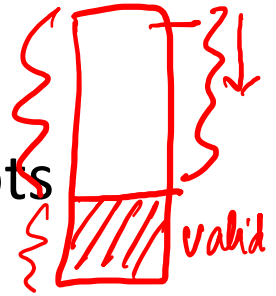
Question 2: What type of fit is the above?

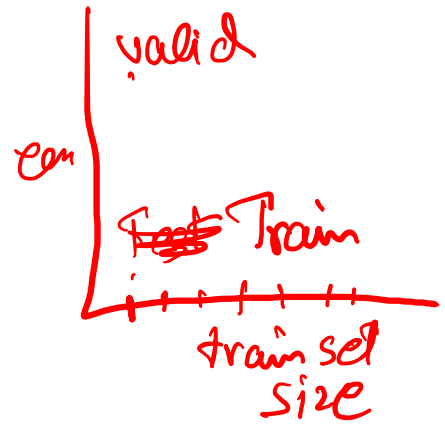
1. Overfit; 2. Underfit; 3. Optimal Fit; 4. None of the above

Deepak Subramani, deepakns@iisc.ac.in

Learning Curves

- To generate learning curves, train the model several times on different sized subsets of the training set.
- The following code defines a function that, given some training data, plots the learning curves of a model
  - **from sklearn.metrics import** mean_squared_error
  - **from sklearn.model_selection import** train_test_split
  - **def** plot_learning_curves(model, X, y):
    - X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
    - train_errors, val_errors = [], []
    - **for** m **in** range(1, len(X_train)):
      - model.fit(X_train[:m], y_train[:m])
      - y_train_predict = model.predict(X_train[:m])
      - y_val_predict = model.predict(X_val)
      - train_errors.append(mean_squared_error(y_train[:m], y_train_predict))
      - val_errors.append(mean_squared_error(y_val, y_val_predict))
    - plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
    - plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")



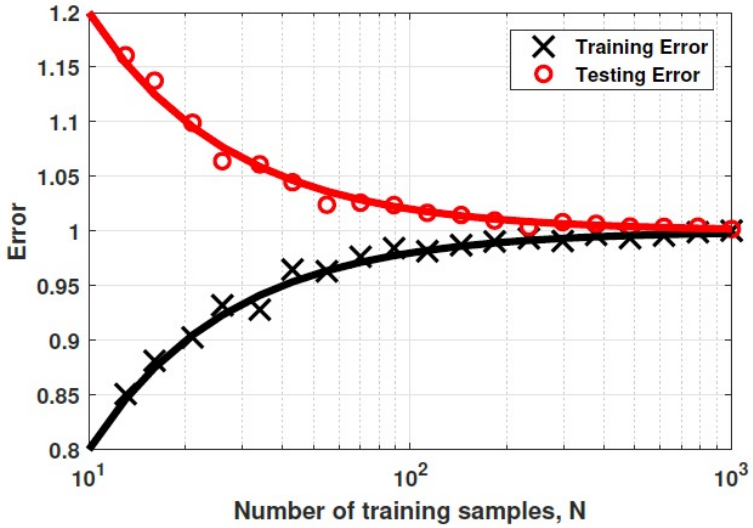


Formula for Linear Regression Errors

- For $m + t$ data points and a linear model with n features
- We fit a linear model $h_\theta(X; \theta)$ from the train set with m data points
  - Consider them first m points
- t data points are used for testing
- Let us analyze the result for train error and test error
- $\epsilon_{train} = MSE(\hat{y}_{1:m}, y_{1:m})$
- $\epsilon_{test} = MSE(\hat{y}_{m+1:m+t}, y_{m+1:m+t})$
- If σ is the variance of y, then we can derive that
  - $\epsilon_{train} = \sigma^2\left(1 - \frac{n}{m}\right)$
  - $\epsilon_{test} = \sigma^2\left(1 + \frac{n}{m}\right)$

Train and Test Errors

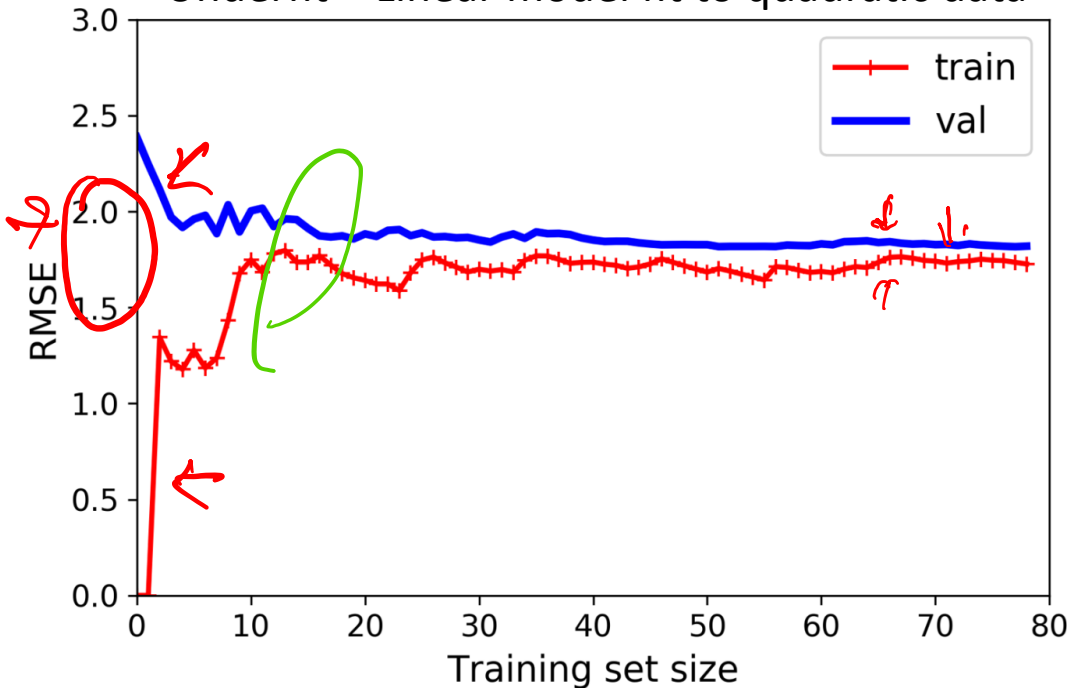
- $\epsilon_{train} = \sigma^2\left(1 - \frac{n}{m}\right)$; $\epsilon_{test} = \sigma^2\left(1 + \frac{n}{m}\right)$
- Usually $n \ll m$ – We want smaller number of parameters than model
  - Else it is over-determined system – infinitely many solution
  - We can always get train error to zero is $n > m$
- Learning curves are formed by increasing m and seeing how $\epsilon_{train}, \epsilon_{test}$ change
- How do train and test error change w.r.t. m?
  - $\epsilon_{train} \uparrow$ as $m \uparrow$. Thus more training samples make fitting harder.
  - $\epsilon_{test} \downarrow$ as $m \uparrow$. Thus more training samples improve generalization.
- How do train and test error change w.r.t. d?
  - $\epsilon_{train} \downarrow$ as $d \uparrow$. Thus a more complex model makes fitting easier.
  - $\epsilon_{test} \uparrow$ as $m \uparrow$. Thus a more complex mode makes generalization harder.



Theoretical learning curve for a perfect linear model with Gaussian noise

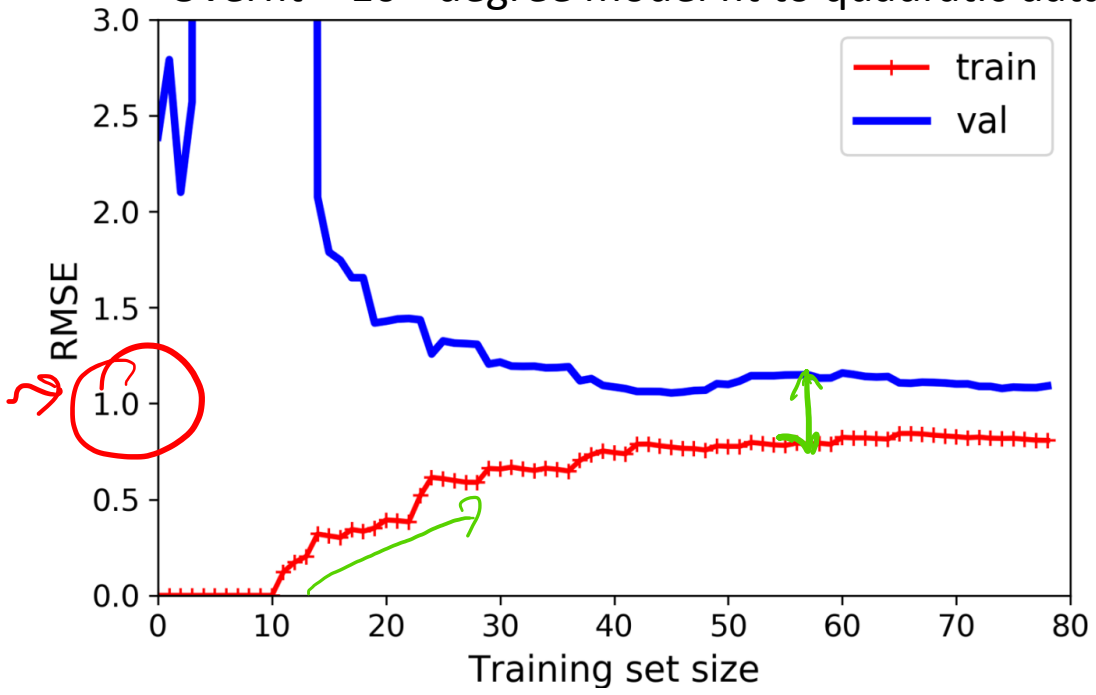
Example of Underfit and Overfit

Underfit – Linear model fit to quadratic data



- With one instance, fit is perfect
- Thereafter training error increases
- Plateaus fairly quickly
- RMSE is high. Magnitude of training is around 2
- RMSE on validation and training are very close

Overfit – 10th degree model fit to quadratic data



- With 10 instances, fit is perfect
- Validation error very high until order of fit
- Difference between training and validation is high
- RMSE of training is lower than underfit

[Figures from Geron]

Regularization

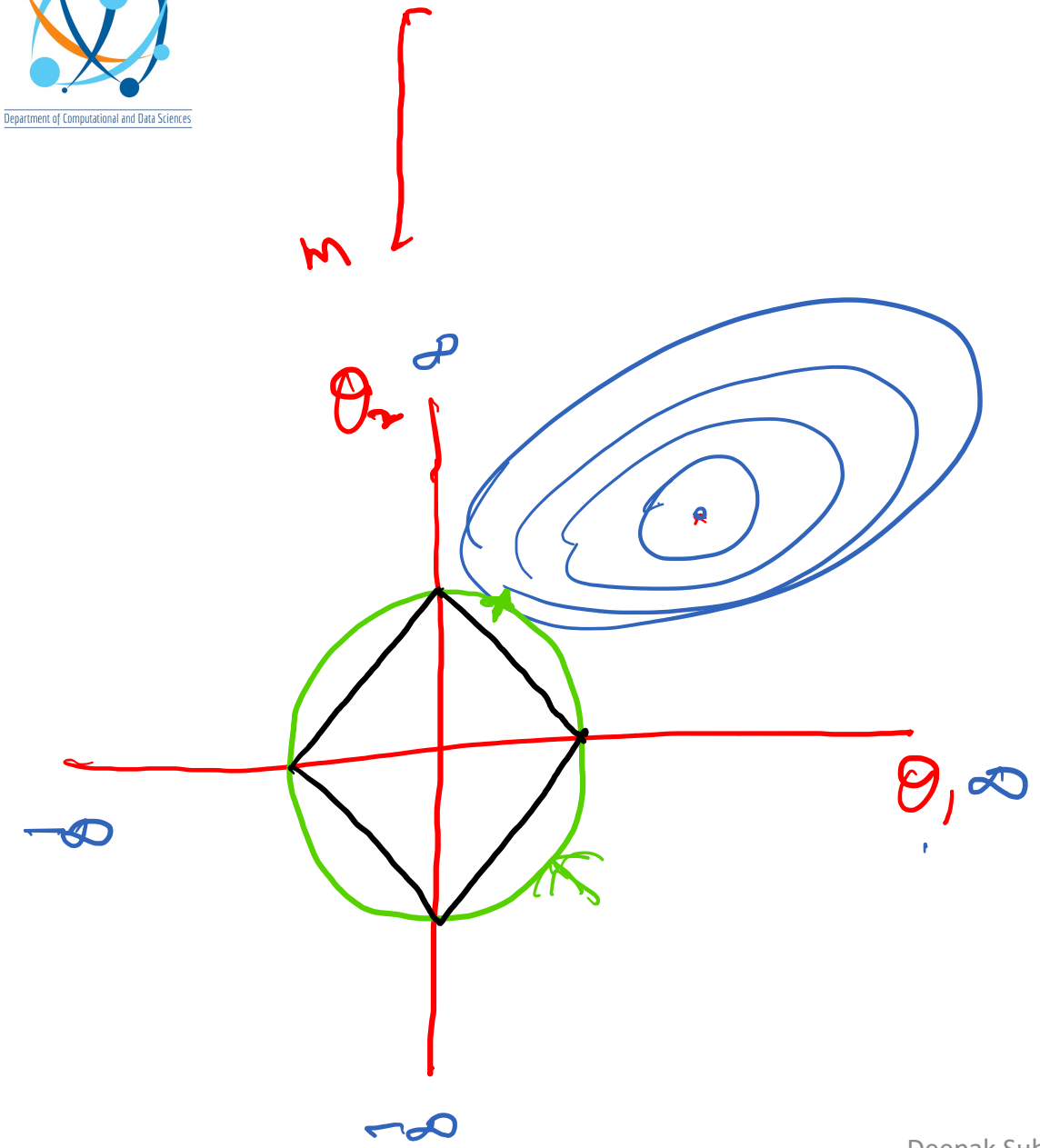
- The fewer degrees of freedom a model has, the harder it will be for overfit
- If we put constraints on θ, it will reduce overfitting and ensure models generalize well beyond training data
- We will study three different ways to constrain the weights
  - Ridge Regression – L2 Regularization, Tikhonov Regularization
  - Lasso Regression – L1 Regularization
  - Elastic Net – L1+L2 Regularization

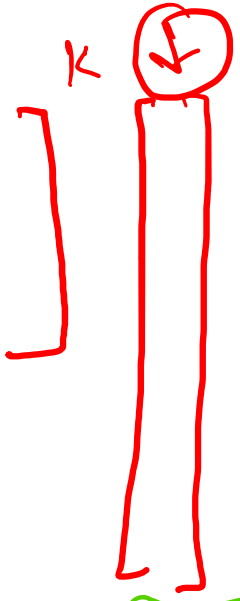
L1, L2 norm vectors

$$L_p = \left[\theta_1^p + \theta_2^2 + \dots + \theta_n^p\right]^{1/p}$$

$$L2 = \sqrt{\theta_1^2 + \theta_2^2 + \dots + \theta_n^2}$$

$$L1 = |\theta_1| + |\theta_2| + |\theta_3| + \dots |\theta_n|$$





Inverse
Prior
constraint

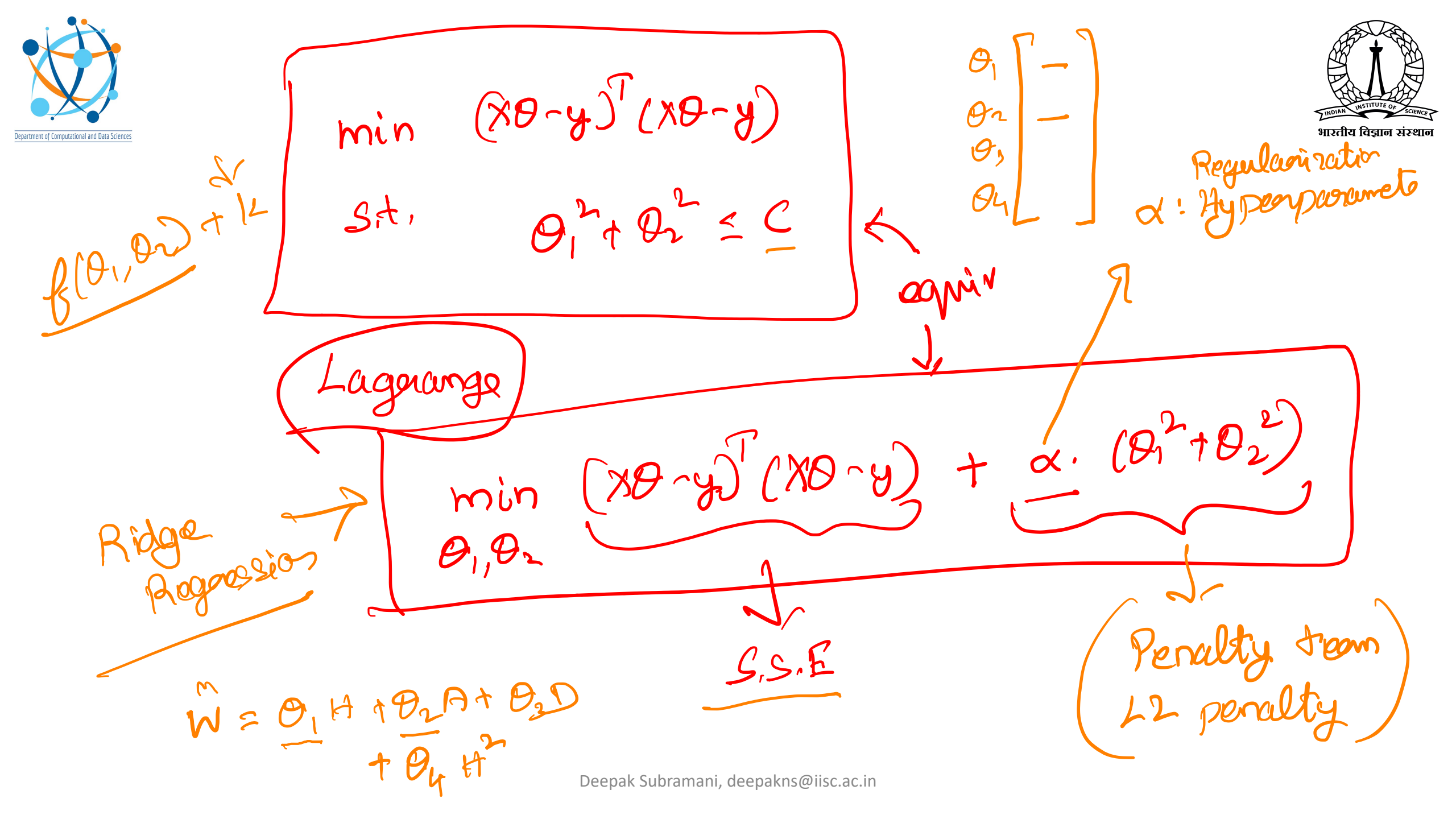
$$\hat{w} = \theta_1 A + \theta_2 A$$

$$\min_{\theta_1,\theta_2} (X\theta - y)^T (X\theta - y)$$

$$\theta_1^2 + \theta_2^2 \leq C_1$$

constrained optim..

$$|\theta_1| + |\theta_2| \leq C_2$$


Department of Computational and Data Sciences
INDIAN INSTITUTE OF SCIENCE
भारतीय विज्ञान संस्थान
min (Xθ−y)ᵀ(Xθ−y)
s.t. θ₁² + θ₂² ≤ c
f(θ₁, θ₂) + λ
θ₁ θ₂ θ₃ θ₄
Regularization
α : Hyperparameter
equiv
Lagrange
min θ₁, θ₂ (Xθ−y)ᵀ(Xθ−y) + α·(θ₁² + θ₂²)
Ridge Regression
S.S.E
Penalty term
L2 penalty
W = θ₁H + θ₂A + θ₃D + θ₄H²

Ridge Regression

- Add a regularization term $\alpha \sum_{i=1}^{n} \theta_i^2$ to the cost function

$$f(\theta) = \frac{1}{2} r(\theta)^T r(\theta) + \frac{1}{2} \alpha \sum_{i=1}^{n} \theta_i^2$$

- When using regularization, we don't include the bias term θ_0.
- We only use the weights of the features for regularization
- Hyperparameter α controls amount of regularization
  - If α = 0, then Ridge Regression is just Linear Regression.
  - If α is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean.
- It is important to scale the data (e.g., using a StandardScaler) before performing Ridge Regression, as it is sensitive to the scale of the input features.
- Scaling is needed for most regularized models.

Solution to Ridge Regression

- A closed form solution similar to the normal equation can be found for the Ridge Regression problem
- The final solution is $\theta^* = (X^T X + \alpha A)^{-1} X^T y$, where A is an identity matrix with 0 in the (0,0) location corresponding to the bias term
- Using Cholesky Factorization
  - from sklearn.linear_model import Ridge
  - ridge_reg = Ridge(alpha=1, solver="cholesky")
  - ridge_reg.fit(X, y)
- Using SGD
  - sgd_reg = SGDRegressor(penalty="l2")
  - sgd_reg.fit(X, y.ravel())

Different Loss and Evaluation Functions?

- Use regularized cost function only for training, not for testing
- Apart from regularization, another reason they might be different is that a good training cost function should have optimization-friendly derivatives, while the performance measure used for testing should be as close as possible to the final objective.
- For example, classifiers are often trained using a cost function such as the log loss (discussed shortly) but evaluated using precision/recall.

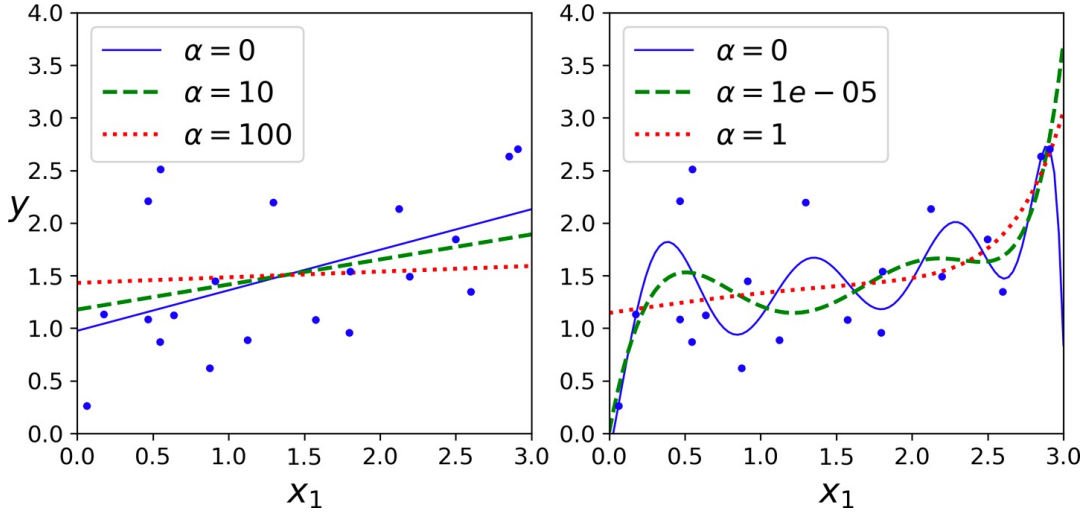
Lasso Regression

- Least Absolute Shrinkage and Selection Operator Regression (usually simply called Lasso Regression) is another regularization technique
- Unlike Ridge Regression, here the ℓ_1 norm is added to the cost function

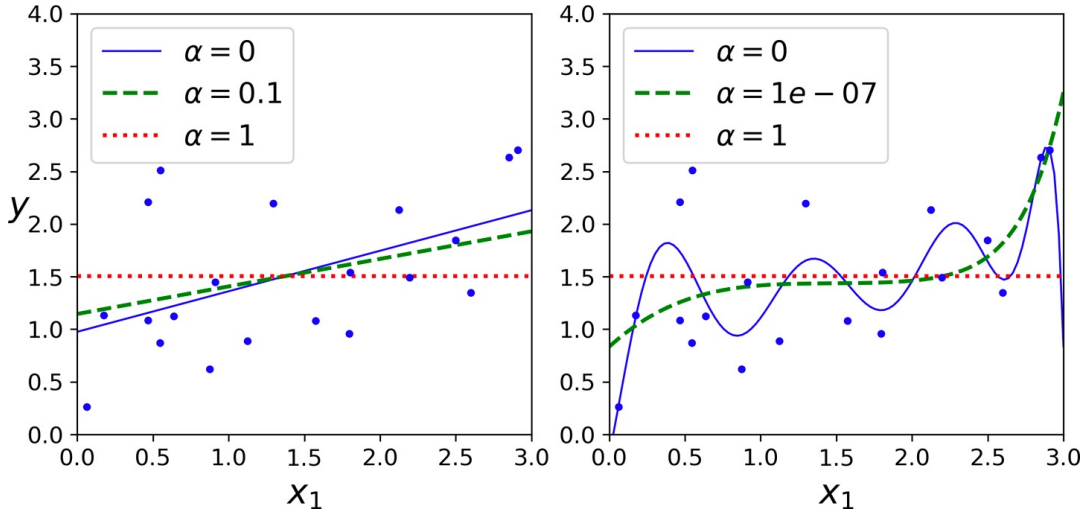
$$f(\theta) = \frac{1}{2} r(\theta)^T r(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

- Like Ridge, here also α is the amount of regularization applied
- LASSO is a sparsity promoting regularizer – it tends to eliminate the weights of the least important features (i.e., set them to zero).
- Notice: There is no ½ in front of the regularization term.

Ridge, Lasso with Different Levels of Regularization



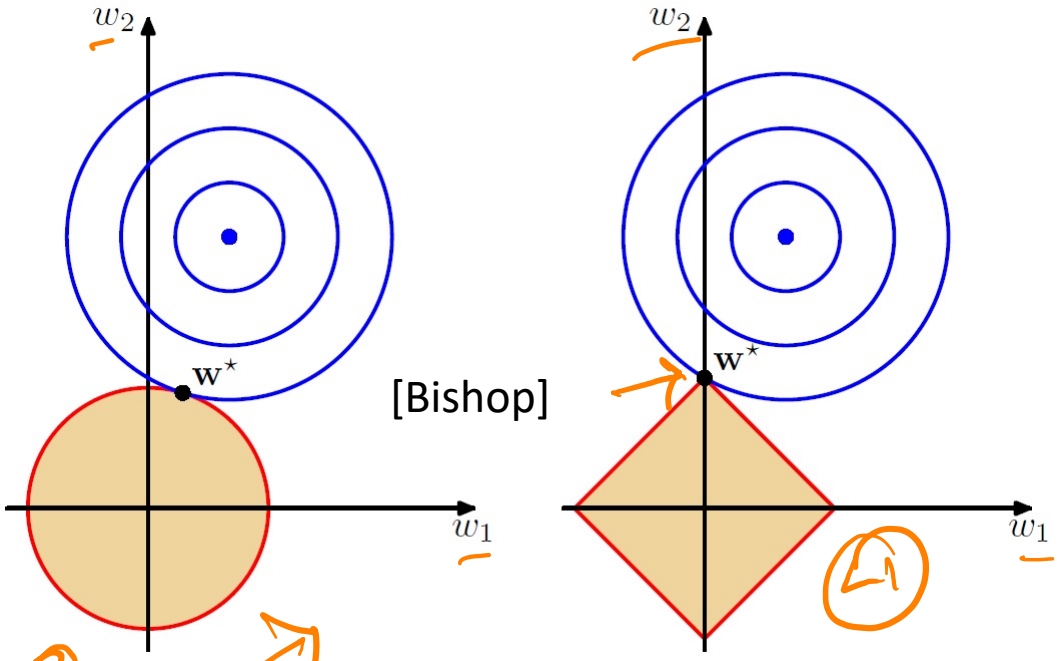
Ridge Regression on Linear Data
Left: Linear Regressors
Right: 10th Degree Polynomial Regressors



LASSO Regression on Linear Data
Left: Linear Regressors
Right: 10th Degree Polynomial Regressors
Lasso promotes sparsity more than Ridge

[Figures from Geron]

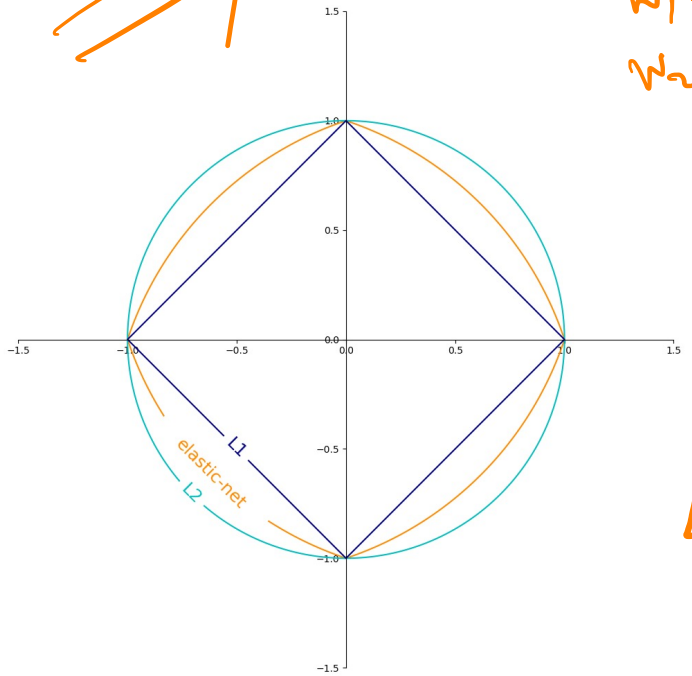
Ridge, Lasso Solutions



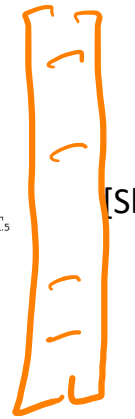
[Bishop]

$w_1 = 0$

$w_2 \neq 0$



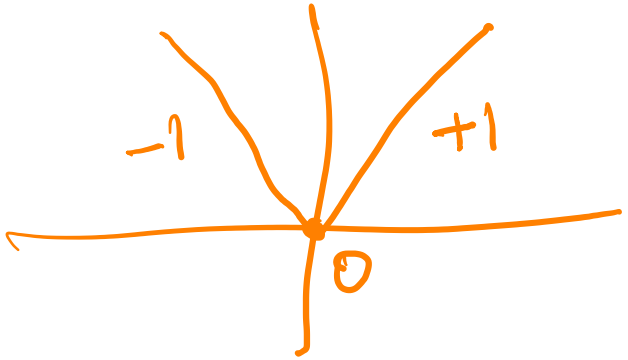
[Sklearn Website]



- A 2-d example
- The minima of the penalty terms is at (0,0)
- Lasso objective function is not differentiable – Needs Sub-Gradients
- Ridge objective function is differentiable
- For large α the optimal point goes close to 0, but never exactly zero
- In Ridge, the gradients reduce magnitude as the iteration gets closer to the true optimal
- In Lasso the iterates oscillate around due to the sub-gradient use

Subgradients for LASSO

- The ℓ_1 cost function is not differentiable at $\theta_i = 0$
- GD still works if a subgradient is used
- $g(\theta) = X^T r(\theta) + \alpha \begin{bmatrix} sign(\theta_1) \\ sign(\theta_2) \\ \cdot \\ sign(\theta_n) \end{bmatrix}$ where $sign(\theta_i) = \begin{cases} -1 \; if \; \theta_j < 0 \\ 0 \; if \; \theta_j = 0 \\ +1 \; if \; \theta_j > 0 \end{cases}$



Elastic Net

- Elastic net uses a combination of l1 and l2 regularization
- The regularization term is $\alpha \left[r \sum_{i=1}^{n} |\theta_i| + (1 - r) \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 \right]$
- r is the mixing ratio –
  - r = 0 -> Elastic Net = Ridge Regression,
  - r = 1 -> Elastic Net = Lasso Regression

Guidelines

- When to use Plain, Ridge, Lasso, Elastic net?
  - Generally avoid plain Linear Regression - almost always preferable to have some regularization
  - Ridge is a good default, but if we believe that only a few features are useful (from domain expertise), then prefer Lasso or Elastic
  - Between lasso and elastic, prefer elastic as lasso may behave irratically when $n > m$

Scikit Learn

- Ridge
  - from sklearn.linear_model import Ridge
  - ridge_reg = Ridge(alpha=0.1)
  - ridge_reg.fit(X, y)
  - SGD - SGDRegressor(penalty="l2", alpha=0.1).
- Lasso
  - from sklearn.linear_model import Lasso
  - lasso_reg = Lasso(alpha=0.1)
  - lasso_reg.fit(X, y)
  - SGD - SGDRegressor(penalty="l1", alpha=0.1).
- Elastic Net
  - from sklearn.linear_model import ElasticNet
  - elastic_reg = ElasticNet(alpha=0.1, l1_ratio=0.5)
  - elastic_reg.fit(X, y)
  - SGD - SGDRegressor(penalty="elasticnet",alpha=0.1,l1_ratio=0.5).
- Ridge Implementation: 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'
- Lasso and ElasticNet Implementation: coordinate descent

Audience Poll

1. L1 regularizer is differentiable
   - True, False
2. L2 Regularization with linear regression has an exact closed form solution
   - True, False

$(X^TX + \alpha I)\theta = X^Ty$

3. Elastic Net promotes Sparsity
   - True, False
4. Ridge Regression is L1 regularization
   - True, False

Summary

- Sklearn.linear_model.LinearRegression solves the normal equation
- SGDRegressor solves the regression problem with SGD
  - Different types of loss are possible
- Cross Validation and Learning Curves are used to figure out if a model is overfitting or underfitting
- Regularization helps to build accurate compact models that avoid overfitting
  - Ridge, Lasso, Elastic Net
  - Remember the rules of thumb for choosing regularization

Linear Models: Classification

DA-224-O JAN2022

Deepak Subramani
Assistant Professor
Dept. of Computational and Data Science
Indian Institute of Science Bengaluru

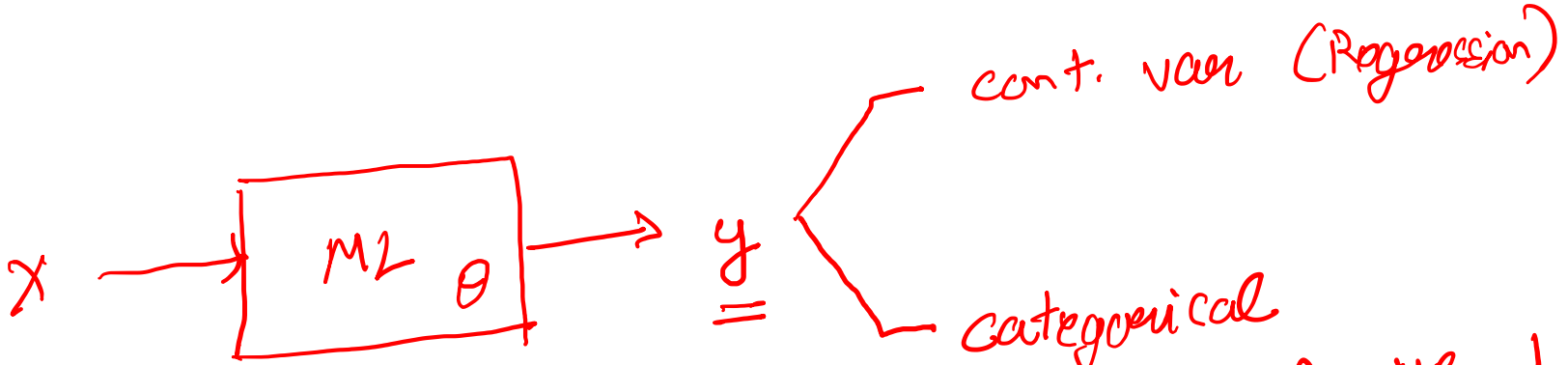
The Story So Far

- Sklearn.linear_model.LinearRegression solves the normal equation $h_\theta = X\theta$
- SGDRegressor solves the regression problem with SGD
  - Different types of loss are possible
- Underfit vs Overfit
- Regularization helps to build accurate compact models that avoid overfitting
  - Ridge, Lasso, Elastic Net
  - Remember the rules of thumb for choosing regularization

1) Loss Fn
2) Optimization

Deepak Subramani, deepakns@iisc.ac.in

Classification



$\Rightarrow \hat{y} = h_\theta(x; \theta)$

$\min_\theta g(\hat{y} - y)$

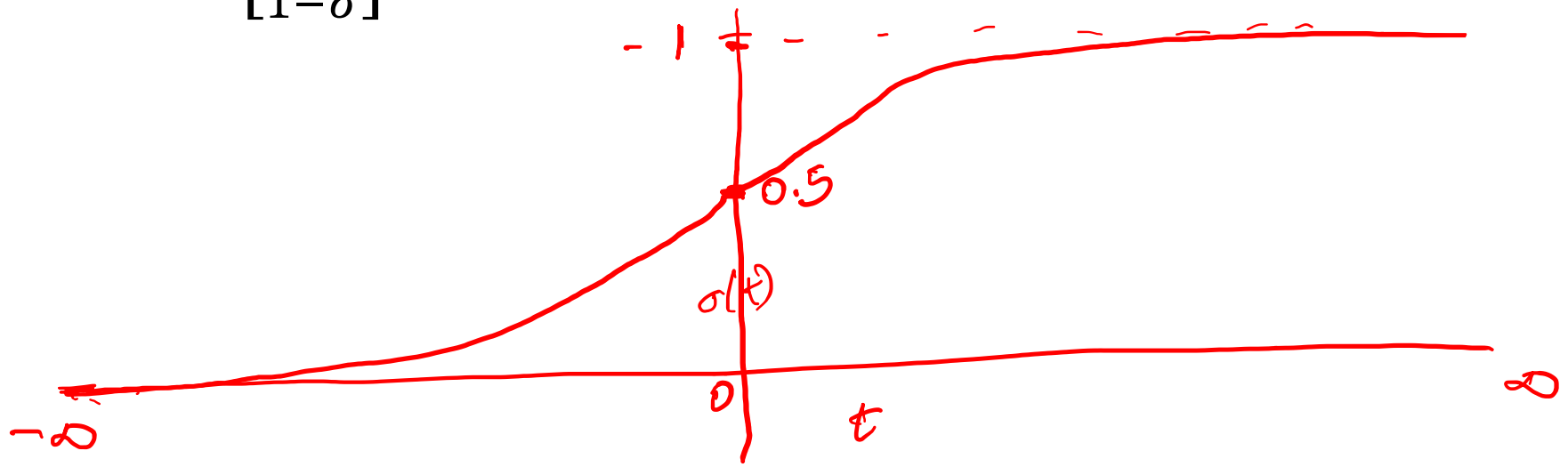
$\underline{x}$ = image
- cat
- dog
- elephant

Logistic Regression

- When the task is binary classification, we can set the target to the probability that a particular label is obtained
- To make regression a binary classification task:
  - Predict a probability
  - If the estimated probability > 50%, predict that the instance is in the positive class, labeled “1”, and
  - otherwise predict that it belongs to the negative class, labeled “0”.
- How to predict the probability?
  - Use a sigmoid function

Logistic or Sigmoid Function

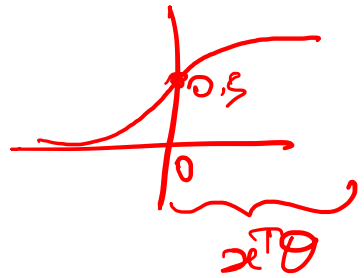
- $\sigma(t) = \frac{1}{1+\exp(-t)}$
- Takes an input in $(-\infty, \infty)$ and returns a value between 0 and 1
- $logit(t) = log\left[\frac{\sigma}{1-\sigma}\right]$



Logistic Regression Model

- Estimate the probability of class using $h(\theta)$
- $\hat{p} = h_\theta(\boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{x}^T \theta)$ – per instance
- Compare Regular Regression where the output itself is predicted with no modification by the sigmoid function
- After estimating probability, class is predicted as

$$\hat{y} = \begin{cases} 1 \; if \; \hat{p} \geq 0.5 \\ 0 \; otherwise \end{cases}$$

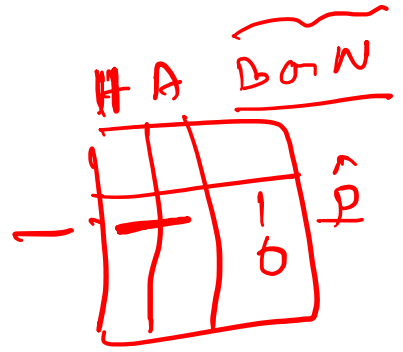


- The sigmoid function is such that $\sigma(t) \geq 0.5$ when t is positive
- When $x^T\theta \geq 0$, the class label is 1, else 0

Audience Poll - 1

- Logistic and Sigmoid Function are the same
  - True, False
- Sigmoid takes any real number as input and outputs a number between 0 and 1
  - True, False
- In Logistic Regression, the output itself is predicted
  - True, False

What and How to Train?



- Loss function for a single training instance
- $c(\theta) = \begin{cases} -\log(\hat{p}) & if\ y = 1 \\ -\log(1-\hat{p}) & if\ y = 0 \end{cases}$
- -log(t) becomes very large when t approaches zero
- When $y = 1$, we want $\hat{p} = 1$, and vice versa.
- The cost function over full training set is

$$f(\theta) = -\frac{1}{m}\sum_{j=1}^{m}\left[y_j \log(\hat{p}_j) + (1-y_j)\log(1-\hat{p}_j)\right]$$

- This is called the ***log loss*** cost function
- No closed form expression to minimize it (no Normal Equation)
- Fortunately, function is convex and gradient descent will find global optima

Deepak Subramani, deepakns@iisc.ac.in

Gradients

- Partial derivatives of the cost function are as follows

$$\nabla_i f(\theta) = \frac{1}{m}\sum_{j=1}^{m}\left(\sigma(\boldsymbol{x}_j^T\boldsymbol{\theta}) - \boldsymbol{y}_j\right)x_{j,i}$$

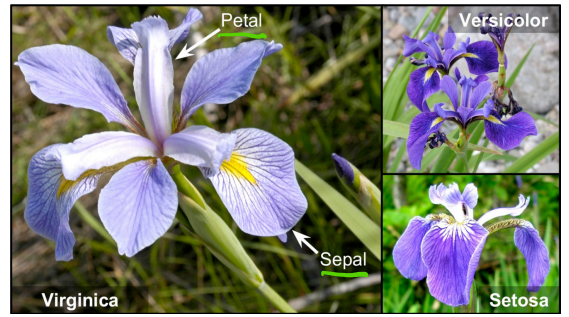
- Use these gradients to perform optimization via
  - 1st order: Batch GD, SGD or Mini-Batch GD,
  - 2nd order: L-BFGS → Quasi Newton
- That is all to train a Logistic Regression model for classification

Gradients

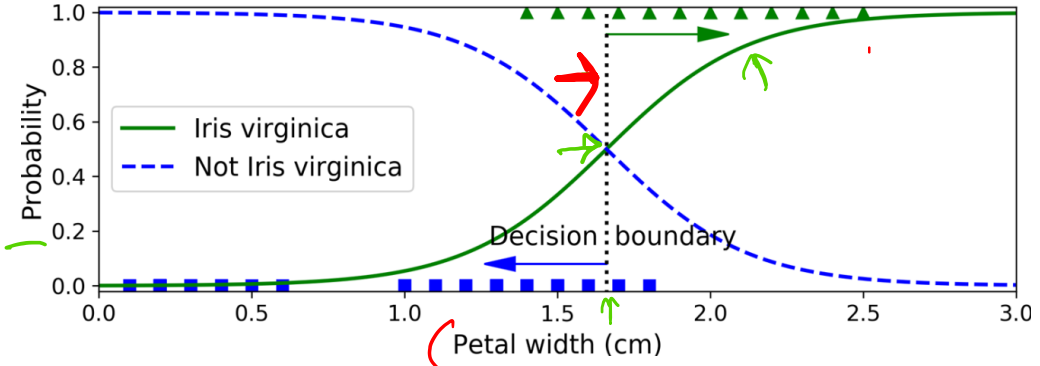
- Implementation in sklearn
  - from sklearn.linear_model import LogisticRegression
  - log_reg = LogisticRegression(penalty='l2',solver='lbfgs')
  - log_reg.fit(X, y)
- This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers.
- **Regularization is applied by default**
  - The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization.
  - The Elastic-Net regularization is only supported by the 'saga' solver.

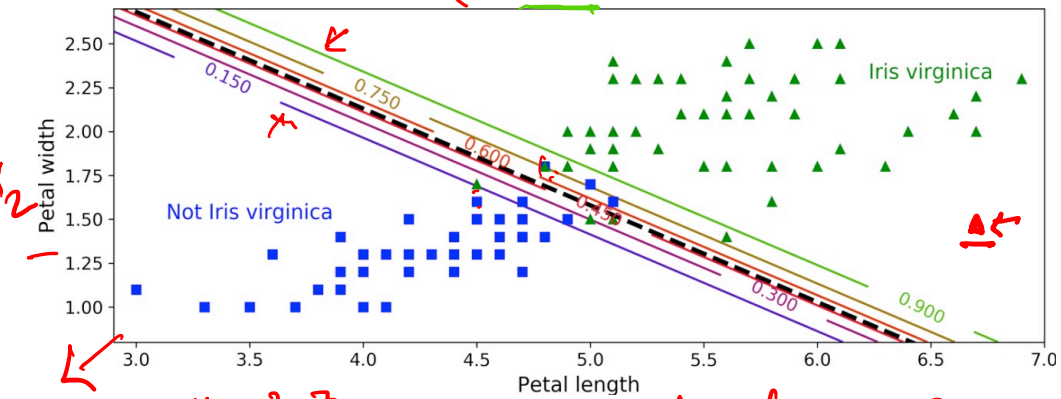
Deepak Subramani, deepakns@iisc.ac.in

Decision Boundaries: Logistic Regression in Action



- Iris Dataset: 3 species (Iris setosa, Iris versicolor and Iris virginica)
- 150 samples with sepal and petal length and width
- **Binary classifier for Iris virginca or not**
  - **from sklearn import** datasets
  - iris = datasets.load_iris()
  - list(iris.keys())
    - ['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
  - X = iris["data"][:, 3:] *# petal width*
  - y = (iris["target"] == 2).astype(np.int) *# 1 if Iris virginica, else 0*
  - **from sklearn.linear_model import** LogisticRegression
  - log_reg = LogisticRegression()
  - log_reg.fit(X, y)
- Right Middle: 1d decision boundary
- Right Bottom: 2d decision boundary





[Figures from Geron]

```
def predict_can_crash(X):
    return False
```

accuracy 99%

Unbalanced Datasets

- Weighted Logistic Regression
- Consider a data set with 99% of Class 0 and 1% of Class 1
  - 0: Majority Class; 1: Minority Class
- Penalize Label 1's error more than Label 0's error
- Weighted Log-Loss Function

$$f(\theta) = -\frac{1}{m}\sum_{j=1}^{m}\left[w_1 y_j \log(\widehat{p_j}) + w_0\left(1-y_j\right)\log(1-\widehat{p_j})\right]$$

- Sklearn
  - `w = {0:1, 1:99} #define class weights`
  - `lg = LogisticRegression(random_state=42, class_weight=w) # define model`
  - `lg.fit(X_train,y_train) # fit it`

Audience Poll - 2

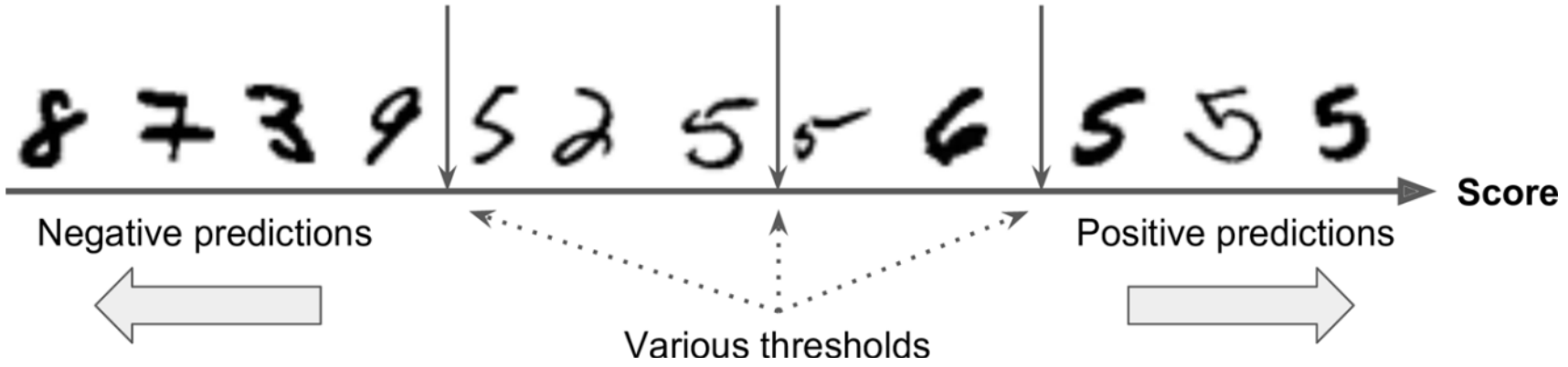
- Logistic Regression returns probability of belonging to a class
  - True, False
- In Classification, we often train with a different loss function than the evaluation metric
  - True, False
- Usual Logistic Regression can be used for imbalanced datasets
  - True, False

Confusion Matrix: Revisit

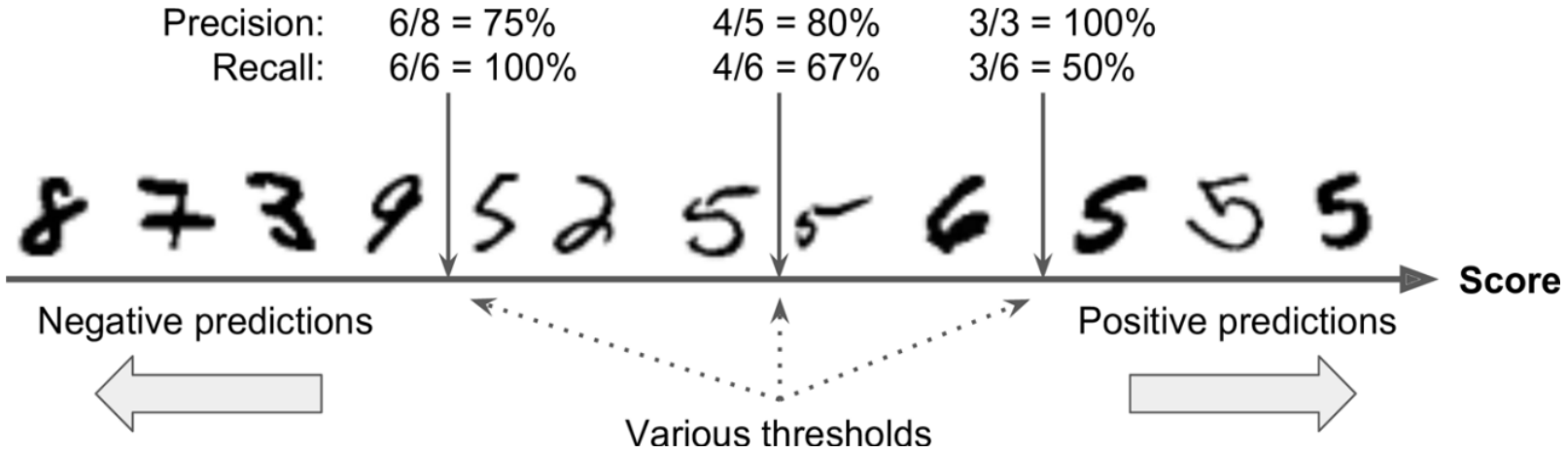
	True Label A	True Label ~A
Pred. Label A	True Positive	False Positive
Pred. Label ~A	False Negative	True Negative

- Precision = $\frac{TP}{TP+FP}$;
  - Among all prediction of Label A, how many are actually Label A
  - Trivial 100% Precision – Make only one Prediction of Label A, and ensure that it is correct. Then TP=1, FP=0, and Precision=1
- Recall = $\frac{TP}{TP+FN}$;
  - Of all true Label A, how many does our classifier predict as Label A
  - Combined with Precision, we now have a good sense of the goodness of our classifier
- Typically, we want high precision and high recall
- F1 Score = $\frac{2}{\frac{1}{precision}+\frac{1}{recall}} = \frac{TP}{TP+\frac{FN+FP}{2}}$
- F1 Score is the harmonic mean of precision and recall. Hence F1 will be high only if precision and recall are high

Precision Recall Tradeoff



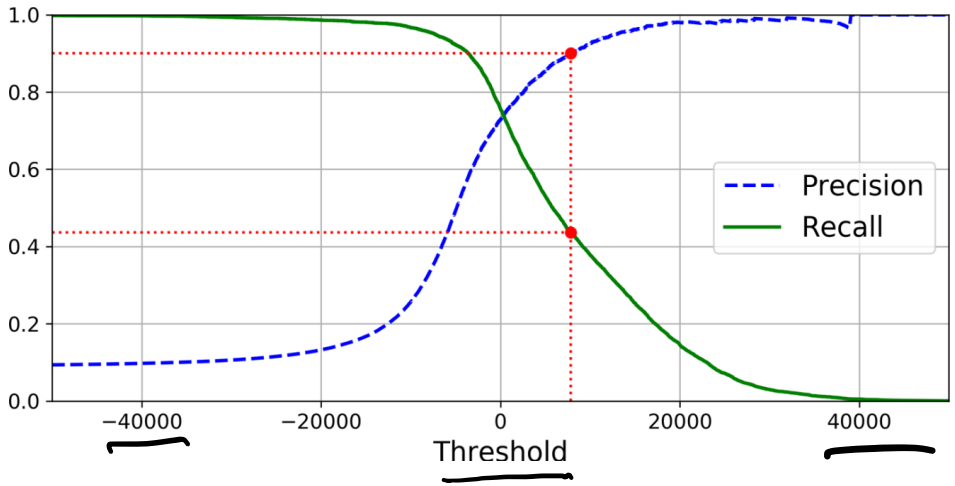
Precision-Recall Tradeoff



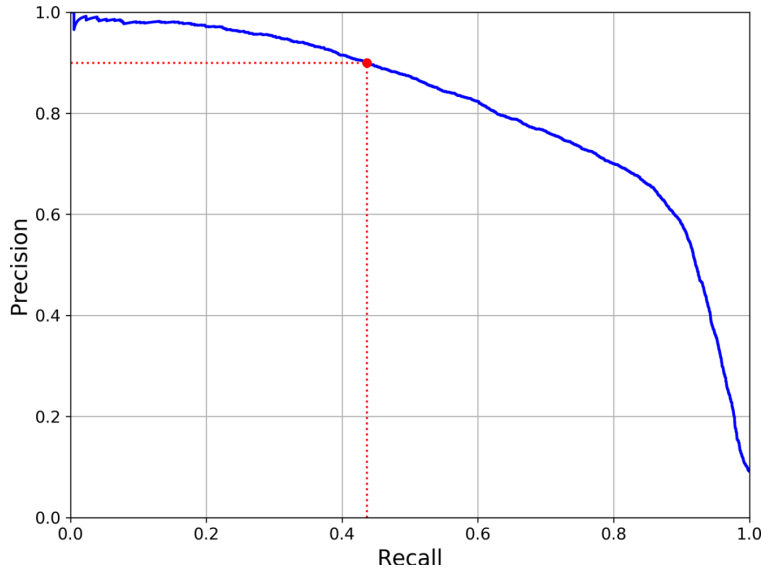
- Classification is based on a score calculation and a threshold set
- Based on where the threshold is placed, precision increases at the expense of recall, and vice versa
- While F1 Score is good, in some applications, we prefer high recall even at the expense of low precision
  - For example, fraud detection. We want to catch all the fraud cases, even if some non fraud cases are also flagged.

Precision-Recall Tradeoff

	True Label A	True Label ~A
Pred. Label A	True Positive	False Positive
Pred. Label ~A	False Negative	True Negative



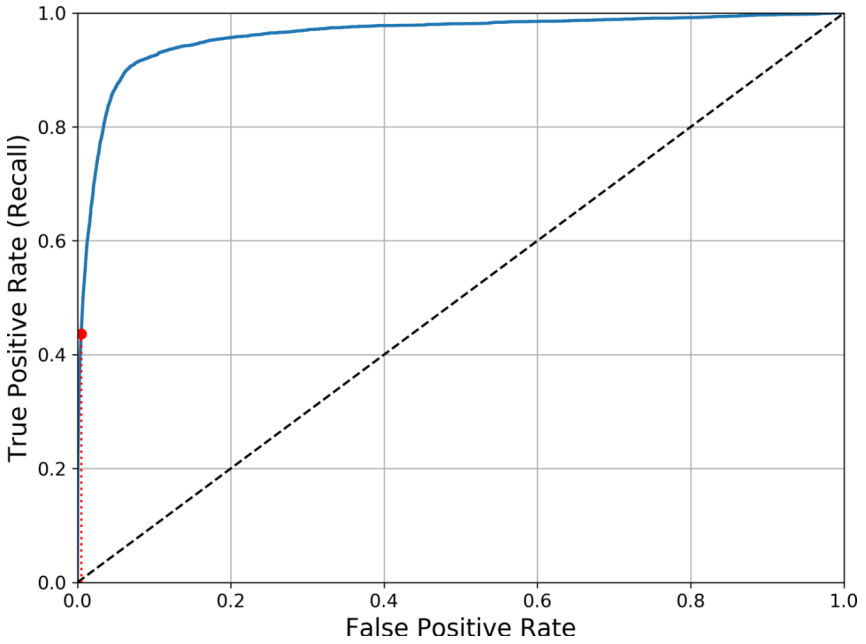
- For different thresholds, we usually plot both precision and recall vs threshold.
- We can also plot Precision vs Recall
- Choose the right threshold from the above curve and settle for a Precision-Recall tradeoff



Plots from Geron Chapter 3

Receiver Operating Characteristic (ROC) Curve

- True Positive Rate (TPR) – Recall or Sensitivity
- True Negative Rate (TNR) – Specificity – Ratio of negative instances correctly classified as negative
- False Positive Rate = 1 – True Negative Rate
- ROC plot is sensitivity vs 1-specificity, i.e., TPR vs FPR
- Compute TPR and FPR for different thresholds and plot it
- The ROC must be as much away from the 45-degree line as possible
- Calculate the Area Under the Curve (AUC) for quantifying goodness of the classifier
- AUC=1 is a perfect classifier, and AUC=0.5 for a random classifier



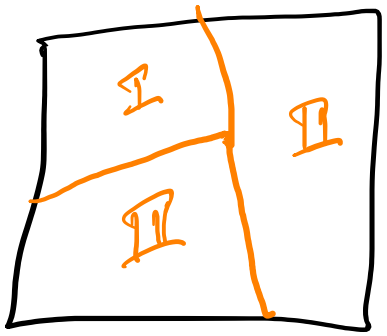
Binary ~~to~~ for Multiclass Approach

- What if we have multiple classes, say K classes?

cat
dog
not

I cat vs oth
II dog vs oth
III not vs oth

- One-vs-Rest (OvR) Strategy
  - Consider a problem of predicting one of 10 lables
  - Convert to Class 0 or Not, Class 1 or Not, etc



- One-vs-Others (OvO) Strategy
  - Convert to Class 0 vs Class 1, Class 0 vs Class 2, etc.
  - There are C(K,2) such classifiers

$^{K}C_{2}$

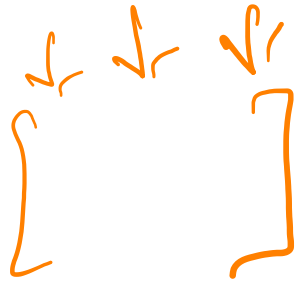
- Use scores (or probabilities) from all classifiers and choose the highest and assign the final class label

Multinomial Logistic Regrssion: Softmax

- Idea
  - Compute a score for each class $s_k(\boldsymbol{x})$
  - Estimate probability of each class by applying the softmax function (a.k.a normalized exponential)
- Softmax score for each input instance $\boldsymbol{x}$ is written as a linear regression model

$$s_k(\boldsymbol{x}) = \boldsymbol{x}^T \theta^{(k)}$$

  - Each class has its own parameter set $\theta^{(k)}$



Multinomial Logistic Regrssion: Softmax

- Probability estimated from softmax score by applying the softmax function

$$\hat{p}_k = \frac{\exp(s_k(\boldsymbol{x}))}{\sum_{k=1}^{K} \exp(s_k(\boldsymbol{x}))}$$

- Final prediction is the class with the highest probability
- $\hat{y} = \underset{k}{\operatorname{argmax}}\, \hat{p}_k = \underset{k}{\operatorname{argmax}}\, s_k(\boldsymbol{x}) = \underset{k}{\operatorname{argmax}}\, \boldsymbol{x}^{\boldsymbol{T}} \theta^{(k)}$
- The above is true for every data instance.
- Question: How to train? How to find the optimal $\boldsymbol{\theta}$

Deepak Subramani, deepakns@iisc.ac.in

Training Softmax Regression

- Objective: Model must predict high probabilities for the target class
- What is the objective function or cost function?

$$f(\Theta) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{k=1}^{K} y_{k,j}\log\left(\hat{p}_{k,j}\right)$$

→ cross entropy

$y_{k,j}$ → [0, 1, 2]

- Target $y_{k,j}$ is a one-hot vector
  - 1 for the true class; 0 everywhere else
- If the predicted probability is 1 for the target class, then error = 0.
- The term in the second summation (over K class labels) is called the cross entropy between $\boldsymbol{y}$ and $\hat{\boldsymbol{p}}$
  - K=2 -> Simple Logistic Regression

Deepak Subramani, deepakns@iisc.ac.in

Gradient of Softmax Cost Function

- Partial derivatives of the cost function are as follows

$$\nabla_{\theta(k)} f(\boldsymbol{\theta}) = \frac{1}{m} \sum_{j=1}^{m} \left(\hat{p}_{k,j} - y_{k,j} \right) \boldsymbol{x}_j$$

- Gradient vector per class is computed and an optimization method is performed to compute the optimal $\boldsymbol{\theta}$
- Implementation in sklearn
  - from sklearn.linear_model import LogisticRegression
  - Softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=10)
  - This applies a l2 regularization by default controlled using C
    - C is the inverse of alpha that we used in LinearRegression

Audience Poll - 3

- Which of the following is a multiclass classification algorithm
  - OvO
  - Linear Regression
  - Logistic Regression
  - OvB
- If the softmax score of 3 classes A,B,C are log(10), log(20) and log(30). What is the probability of class B
  - 1/3
  - 1/6
  - 1/2
  - 2/3

$$\frac{20}{10+20+30} = \frac{20}{60} = \frac{1}{3}$$

- There are three classes A,B and C. What is the one-hot vector for a data point of Class C?
  - [0,0,1]
  - [0,1,0]
  - [1,0,0]
  - [1,1,0]

What is in a name?

- Log-Linear Models – direct name!
- (Multinomial) Logistic Regression – in traditional stats/data analytics
- Softmax Regression – in Machine Learning
- Generalized Linear Models – by theoretical CS/AI/Math/Stats folks
  - With a Binomial Distribution MaxEnt
- Very Shallow (Sigmoidal) Neural Networks – to sound cool today!

Summary of Logistic Regression

- sklearn.linear_model.LogisticRegression solves a classification problem
- SGDClassifier solves the classification problem with SGD
- Softmax Regression can be used for multiclass classification
- Log Loss in SGD solves the logistic regression problem

Comparison of algorithms for Linear Regression

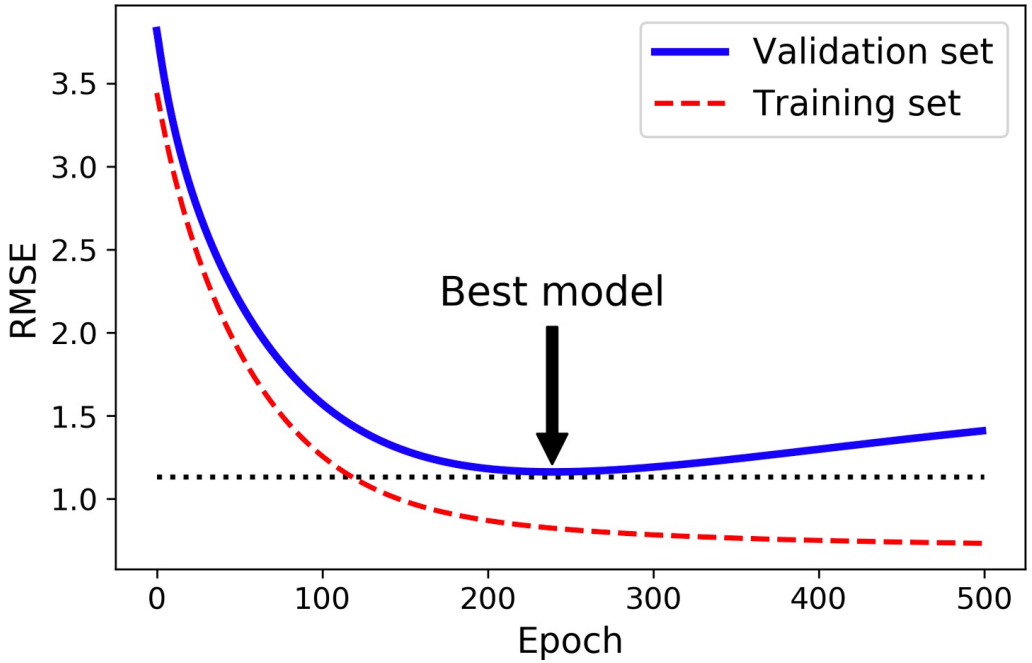
Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

Geron Chapter 4, Page 128, Table 4-1

Out of core algorithms are those that can process data that are too large to fit into a computer's main memory at once.

Early Stopping as a Regularizer

- The Free Regularizer – Just stop the training (i.e., optimization of parameters) as soon as the validation error takes a U-turn
- Calculate and plot the RMSE on the training set and test set with epochs
- As soon as the validation error reaches a minimum and starts increasing stop
- This U-turn indicates that the model has started overfitting the train data
- With SGD and MiniBatch, the curves won't be smooth. So what to do?
  - Stop only when validation error has been above the last known minimum for some time
  - Then roll back to the minimum



[Figures from Geron]