Slides 4: Linear Classifiers

ARISE 2020: ECE Machine Learning Lab

Haoran Zhu

Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, New York

July 16, 2021

Outline

Review of Day 3

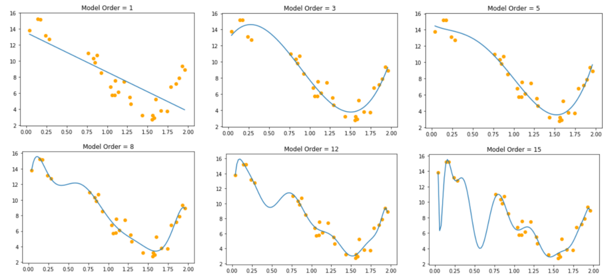
- Yesterday we learned about:
  - Polynomial Regression
  - Overfitting
  - Regularization
  - Cross-Validation

Review: Polynomial Regression

- Polynomial Model: $y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots w_D x^D$
- Design Matrix for Polynomial: $X = \begin{bmatrix} 1 & x_1 & x_2^2 & \cdots & x_1^D \\ 1 & x_2 & x_2^2 & \cdots & x_2^D \\ \vdots & & \ddots & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^D \end{bmatrix}$
- Think of this design matrix as creating new features that are powers of the original single feature
- This is Linear Regression, the process for calculating the weights w's are same
- Python function: sklearn or np.polyfit

Review: Over-fitting

- Train error always decreases as you use higher order models
- A model that is over-fitted on train data is unlikely to work well on new data



Cross-Validation

- Motivation: never determine a hyper-parameter based on training data
- **Hyper-Parameter**: a parameter of the algorithm that is not a model-parameter solved for in optimization.
  - Ex: λ weight regularization value vs. model weights (w)
- Solution: split dataset into three
  - **Training set**: to compute the model-parameters (w)
  - **Validation set**: to tune hyper-parameters (λ)
  - **Test set**: to compute the performance of the algorithm (MSE)

Review: Regularization

- Another method to combat over-fitting
- High weight terms usually lead to over-fitting
- Introduction of a new term in cost function
- $$J = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Review: Regularization

- Another method to combat over-fitting
- High weight terms usually lead to over-fitting
- Introduction of a new term in cost function
- $$J = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{D}(w_j)^2$$
- The new term penalizes the magnitude of weights
- Hyperparameter *lambda* determines how much you regularize: higher lambda $\leftarrow$ more regularization

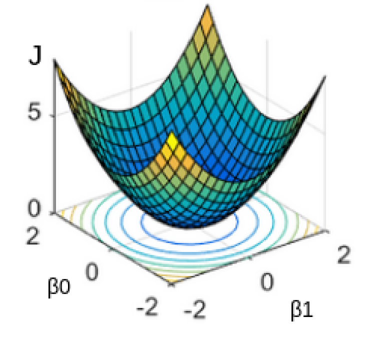
Outline

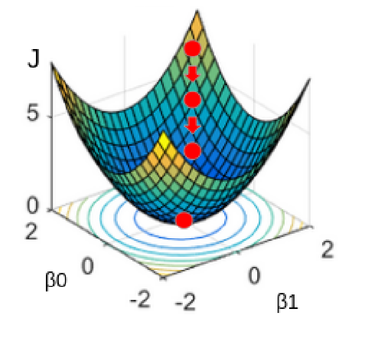
Motivation

- Cannot rely on closed form solutions
  - Computation efficiency: operations like inverting a matrix is not efficient
  - For more complex problems, like neural network, a closed form solution is not always available
- Need an optimization technique to find an optimal solution
  - Machine learning practitioners use gradient based methods

Understanding Optimization

- *Recap* $\hat{y} = w_0 + w_1 x$
- *Loss*, $J = \Sigma_{i=1}^{N}(y_i - \hat{y}_i)^2 \implies J = \Sigma_{i=1}^{N}(y_i - w_0 - w_1 x_i)^2$
- Want to find $w_0 and w_1$ that minimizes J





Gradient Descent Algorithm

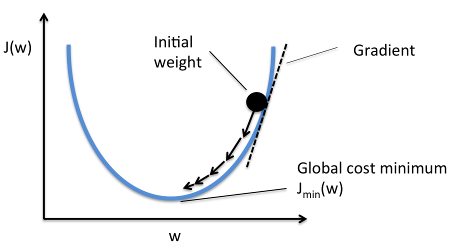
- Update Rule

  Repeat{

  $$w_{new} = w - \alpha \frac{dJ}{dw}$$

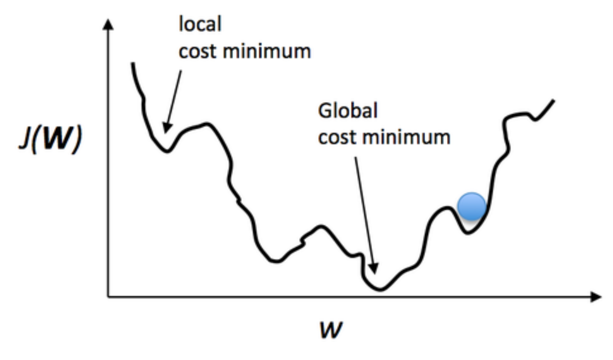
  }

  α is the learning rate

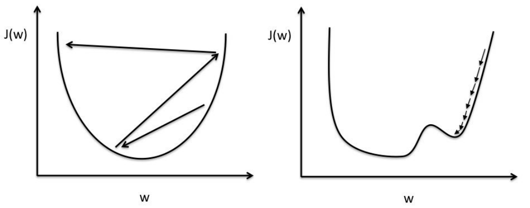


General Loss Function Contours

- Most loss function contours are not perfectly parabolic
- Our goal is to find a solution that is very close to global minimum by the right choice of hyper parameters

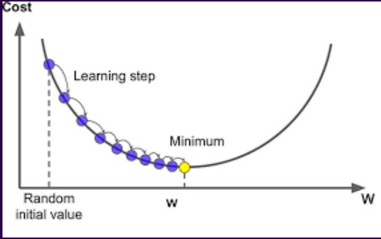


Understanding Learning Rate



Large learning rate: Overshooting.

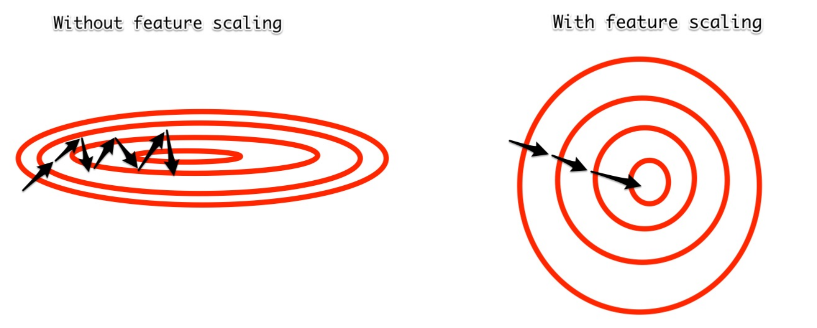
Small learning rate: Many iterations until convergence and trapping in local minima.



Correct learning rate

Importance of Feature Normalization (Optional)

- Helps improve the performance of gradient based optimization



Some Gradient Based Algorithms

- Gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent
- Gradient descent with momentum
- RMSprop
- Adam optimization algorithm

We have many frameworks that help us use these techniques in a single line of code (Eg: TensorFlow, PyTorch, Caffe, etc).

Outline

Demo: Diagnosing Breast Cancer

- We're going to use the breast cancer dataset to predict whether the patients' scans show a malignant tumour or a benign tumour.
- Let's try to find the best linear classifier using linear regression.

Outline

Classification

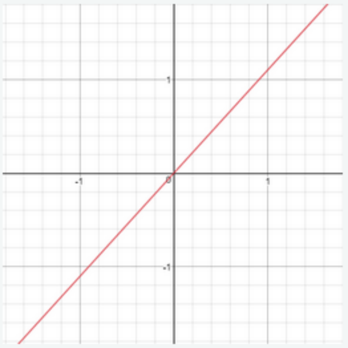
- One method is to use linear regression
  - Map all predictions ($\hat{y}$) greater than 0 as class 1 and all less than 0 as class 0
  - This method doesn't work well because classification is not actually a linear function
- Classification takes the only discrete values for prediction
  - For binary classification problem, y can only take two values, 0 and 1
  - Ex: If we want to build a spam classifier for email, then x may be some features of the email
    - The $y = 1$ if it is a spam
    - Otherwise, $y = 0$

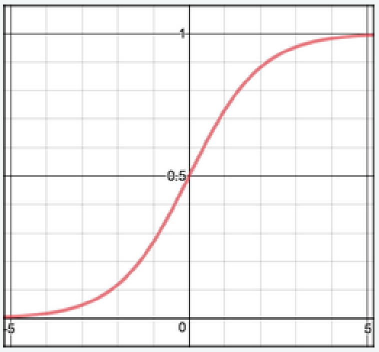
Hypothesis Representation

- Approach classification as old linear regression problem, ignoring the fact that y is discrete
  - We have seen that this approach performs poorly
- To fix this, develop an hypothesis such that $0 \leq \hat{y} \leq 1$
  - This is accomplished by using the Sigmoid function

Sigmoid Function

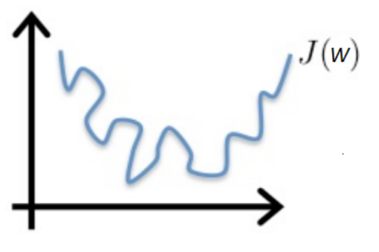
- Recall from linear regression $z = w_0 + w_1 x$
- On application of sigmoid function to z, we force $0 \leq \hat{y} \leq 1$
  - $\hat{y} = sigmoid(z) = \frac{1}{1+e^{-z}}$

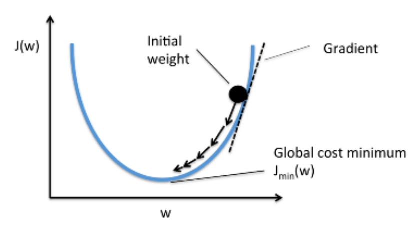




Classification Loss Function

- Cannot use the same cost function that we used for linear regression
  - Logistic function has many local optima
- Logistic cost function is $\frac{1}{m}\Sigma_{i=1}^{m}[-ylog(\hat{y}) - (1-y)log(1-\hat{y})$
  - This loss function is called binary cross entropy loss





- Derivative rules: https://en.wikipedia.org/wiki/Derivative
- Gradient descent for logistic regression (On board)

Outline

Types of Errors in Classification

- Correct predictions:
  - True Positive (TP) : Predict y = 1 when y =1
  - True Negative (TN) : Predict y = 0 when y = 0
- Two types of errors:
  - False Positive/False Alarm (FP): Predict y=1 when y=0
  - False Negative/Missed Detection (FN): Predict y=0 when y=1
- Confusion Matrix:

Accuracy of classifier can be measured by:
- $TPR = P(\hat{y} = 1|y = 1)$
- $FPR = P(\hat{y} = 1|y = 0)$
- Accuracy=$P(\hat{y} = 1|y = 1)$+ $P(\hat{y} = 0|y = 0)$
  - (percentage of correct classification)

predicted→ real↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

$$\text{TPR (sensitivity)} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

$$\text{FPR (1-specificity)} = \frac{\text{FP}}{\text{TN}+\text{FP}}$$

NYU TANDON SCHOOL OF ENGINEERING

Different Metrics for Error

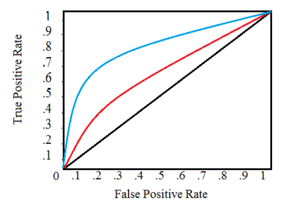
- Metrics to measure the error rate:
  - Recall/Sensitivity/TPR = TP/(TP+FN) (How many positives are detected among all positive?)
  - Precision = TP/(TP+FP) (How many detected positive is actually positive?)
  - Accuracy = (TP+TF)/(TP+FP+TN+FN) (percentage of correct classification)
  - F1-score $= \frac{Precision * Recall}{(Precision + Recall)/2}$
- Why accuracy alone is not a good measure for assessing the model
  - There might be an overwhelming proportion of one class over another
  - Example: A rare disease occurs 1 in ten thousand people
  - A test that classifies everyone as free of the disease can achieve 99.999% accuracy when tested with people drawn randomly from the entire population

Thresholding and ROC

- We can trade-off TPR (sensitivity) and FPR by changing the threshold
- Increasing $t \rightarrow$ Decreases false positives, but also reduces sensitivity
- Decreasing $t \rightarrow$ Increases sensitivity, but also increases false positive
- Why do we want this trade-off?
- Example:
  1. Detection for burglary into a building, need high sensitivity, we can tolerate a few false alarms $\rightarrow$ decrease t
  2. Making decision to buy a stock, making a false positive decision will lose millions $\rightarrow$ increase t

Thresholding and ROC

- ROC (Receiver Operating Characteristics) curve:
- Plot the change between TPR and FPR by varying the threshold
- Allow you to choose the threshold to meet a target TPR/FPR
- A good classifier will have large area under the curve
- A classifier with a higher area under the curve means that under same FPR, it has higher TPR



Outline

Outline

On Board:

- Previous Model: $\hat{y} = \sigma(x^T w)$

On Board:

- Previous Model: $\hat{y} = \sigma(x^T w)$
- Representing Multiple Classes:

On Board:

- Previous Model: $\hat{y} = \sigma(x^T w)$
- Representing Multiple Classes:
  - One-hot / 1-of-K vectors, ex: Class 2 of 4 $[0, 1, 0, 0]$

On Board:

- Previous Model: $\hat{y} = \sigma(\mathrm{x}^T w)$
- Representing Multiple Classes:
  - One-hot / 1-of-K vectors, ex: Class 2 of 4 $[0, 1, 0, 0]$
- Multiple Outputs:

$$\hat{\mathrm{y}} = softmax(W\mathrm{x})$$

On Board:

- Previous Model: $\hat{y} = \sigma(\mathrm{x}^T w)$
- Representing Multiple Classes:
  - One-hot / 1-of-K vectors, ex: Class 2 of 4 $[0, 1, 0, 0]$
- Multiple Outputs:

$$\hat{\mathrm{y}} = softmax(W\mathrm{x})$$

  - (K,1) = (K,M) x (M,1)

On Board:

- Previous Model: $\hat{y} = \sigma(x^T w)$
- Representing Multiple Classes:
  - One-hot / 1-of-K vectors, ex: Class 2 of 4 $[0, 1, 0, 0]$
- Multiple Outputs:

$$\hat{y} = softmax(Wx)$$

  - (K,1) = (K,M) x (M,1)

$$softmax(z)_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

On Board:

- Previous Model: $\hat{y} = \sigma(x^T w)$
- Representing Multiple Classes:
  - One-hot / 1-of-K vectors, ex: Class 2 of 4 $[0, 1, 0, 0]$
- Multiple Outputs:

$$\hat{y} = softmax(Wx)$$

  - (K,1) = (K,M) x (M,1)

$$softmax(z)_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

- Cross-Entropy: $J = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} ln(\hat{y}_{ik})$

Outline

Thank You!

- Next Class: Linear Regression
- The real machine learning will begin!