CMPE 258, Deep Learning

Logistic Regression

Feb 8, 2018

DMH 149A

Taehee Jeong

Ph.D., Data Scientist

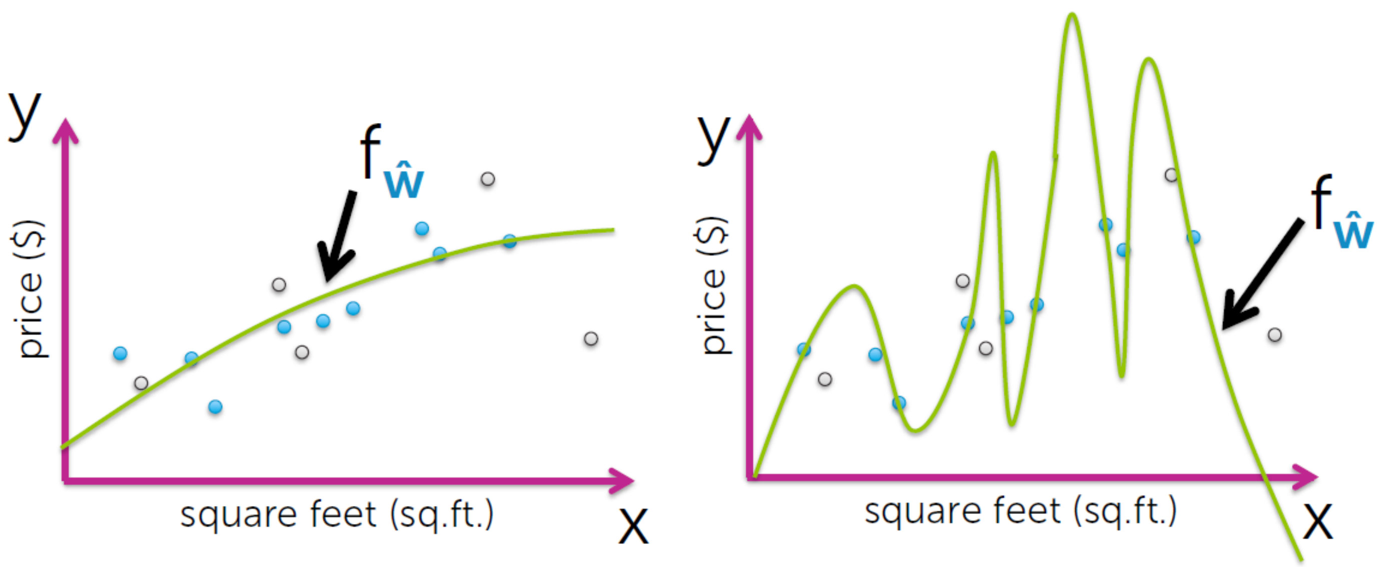
SJSU SAN JOSÉ STATE UNIVERSITY

Recap

- Overfitting of polynomial regression
- Regularization
  - sum of square value, L2 norm, Ridge
  - sum of absolute value, L1 norm, Lasso
- Bias & Variance trade off

Overfitting of polynomial regression

$$\hat{y} = W_0 + W_1x_1 + W_2x_1^2 + W_3x_1^3 + \cdots$$



<Machine Learning, Emily Fox & Carlos Guestrin>

Symptom of overfitting

- Very large value of regression coefficients, W
- Lots of input features
- Small number of observations

Avoiding Overfitting through Regularization

- **Sum of squares**

Ridge : L2 norm

$W_0^2 + W_1^2 + W_2^2 + \cdots = \sum_{j=1}^{n} W_j^2 = \|W\|_2^2$

Cost function $+\lambda\|W\|_2^2$

- **Sum of absolute value**

Lasso: L1 norm

$|w_0| + |w_1| + |w_2| + \cdots = \sum_{j=1}^{n} |w_j| = \|W\|_1$

Cost function $+\lambda\|W\|_1$

Regularized linear regression

Ridge regression (L2 penalty)

Cost function

$$J = \frac{1}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)^2 + \boxed{\frac{\lambda}{m}\sum_{j=1}^{n} W_j^2}$$

Regularized linear regression

Ridge regression (L2 penalty)

Gradient

$$\frac{\partial J}{\partial W_0} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)x_0^i$$ when j=0

$$\frac{\partial J}{\partial W_j} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)x_j^i + \boxed{\frac{2\lambda}{m}W_j}$$ when j>=1

Regularized linear regression

Ridge regression (L2 penalty)

Gradient descant

Repeat {

$$W_0 = W_0 - \alpha \frac{2}{m} \sum_{i=1}^{m} (\widehat{y^i} - y^i) x_0^i$$ when j=0

$$W_j = W_j - \alpha \left[\frac{2}{m} \sum_{i=1}^{m} (\widehat{y^i} - y^i) x_j^i + \frac{2\lambda}{m} W_j \right]$$ when j>=1

}

Regularized linear regression

Ridge regression (L2 penalty)

Matrix form

Cost function

$$\boxed{J = \frac{1}{m}[(W \cdot X - Y)^T (W \cdot X - Y) + \lambda W^T \cdot W]}$$

Gradient Descent

$$\frac{\partial J}{\partial W} = \frac{2}{m}[(X \cdot W - Y)^T \cdot X + \lambda W]$$

$$\boxed{\frac{\partial J}{\partial W} = \frac{2}{m}[(X \cdot W - Y)^T \cdot X + \lambda I \cdot W]}$$

I: identity matrix

SJSU SAN JOSÉ STATE UNIVERSITY

Regularized linear regression

Ridge regression (L2 penalty)

Normalization equation for ridge regression

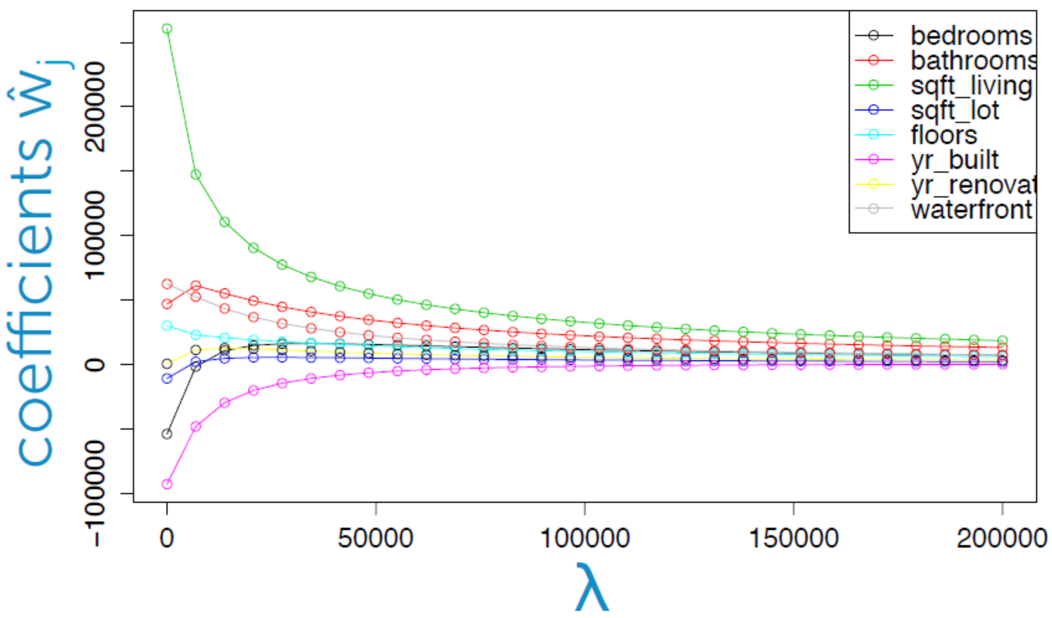
$$\boxed{W = (X^T \cdot X + \lambda I)^{-1} \cdot X^T \cdot Y}$$

when λ=0, $W = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$

When λ= infinite, W=0

Regularized linear regression

Ridge regression (L2 penalty)



<Machine Learning, Emily Fox & Carlos Guestrin>

Regularized linear regression

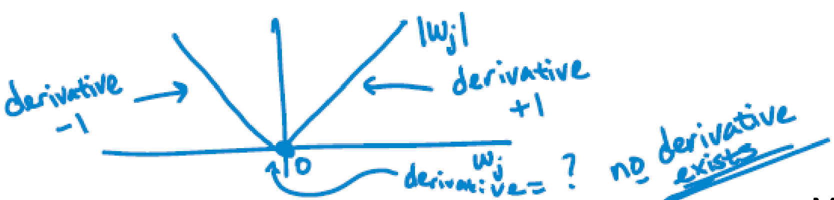
Lasso regression (L1 penalty)

Matrix form

Cost function

$$\boxed{J = \frac{1}{m}[(W \cdot X - Y)^T (W \cdot X - Y) + \lambda |W|]}$$

Gradient Descent



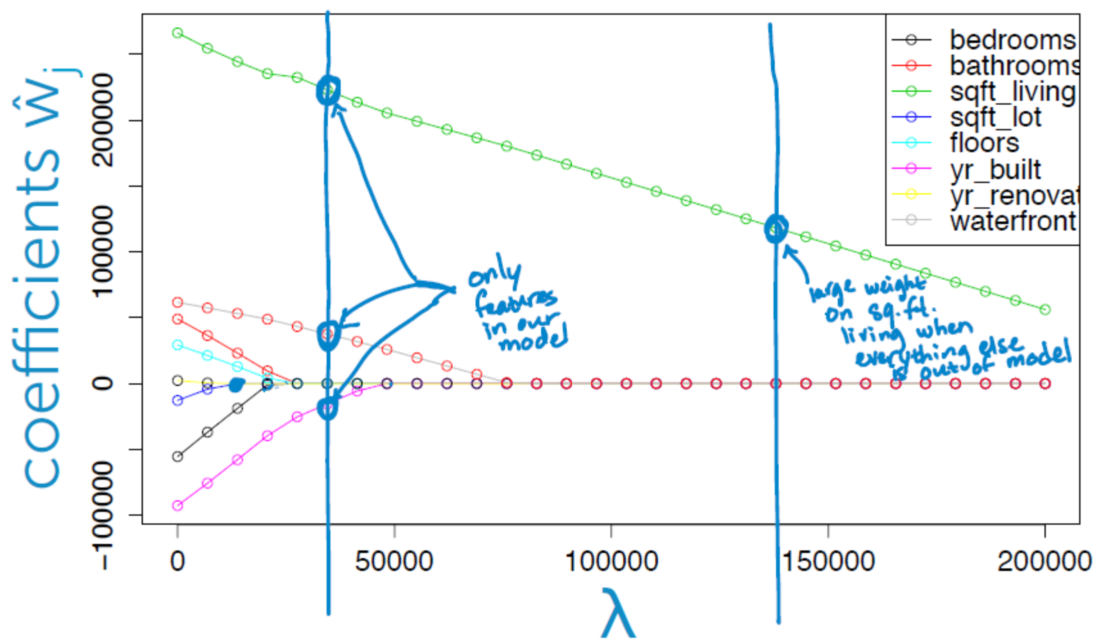
Coordinate descent method

<Machine Learning, Emily Fox & Carlos Guestrin>

$A = \pi r^2$

Regularized linear regression

Lasso regression (L1 penalty)



<Machine Learning, Emily Fox & Carlos Guestrin>

SAN JOSÉ STATE UNIVERSITY

$$A = \pi r^2$$

Bias / Variance Trade off

Large λ: high bias, low variance
Small λ: low bias, high variance

Bias

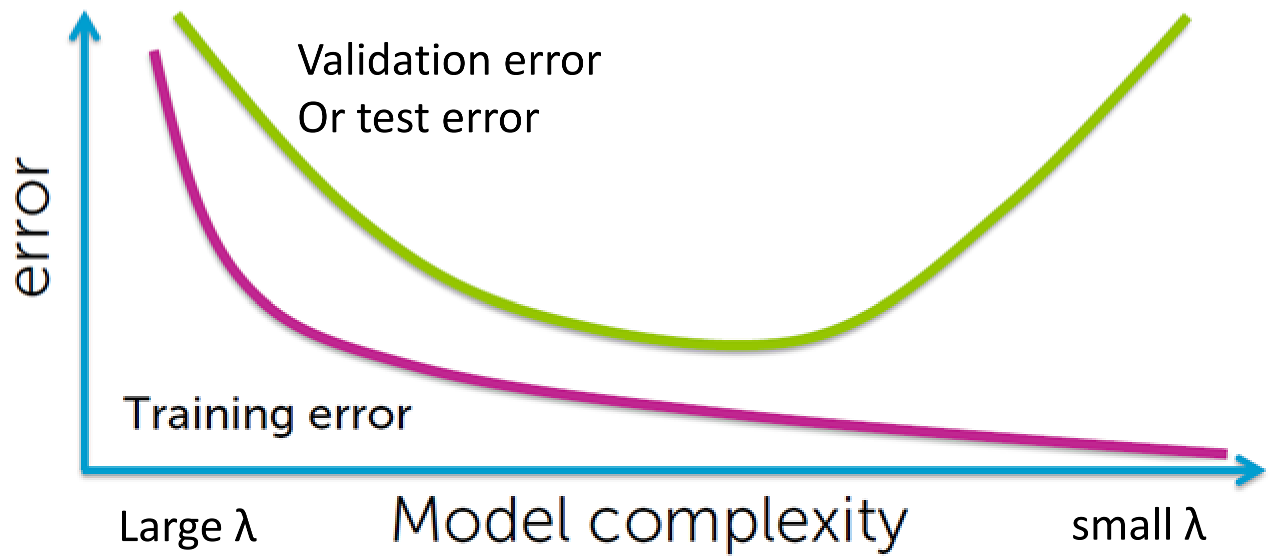
This part of the generalization error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic. A high-bias model is most likely to underfit the training data.

Variance

This part is due to the model's 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 to overfit the training data.

<Hands-On ML, Aurelien Geron>

Overfitting and regularization



<Machine Learning, Emily Fox & Carlos Guestrin>

SJSU SAN JOSÉ STATE UNIVERSITY

Assignment_2: Any question?

Due 2/15

1 (30pts). Polynomial regression / overfitting / regularization
2 (30pts). Polynomial regression with train/validation/test
3 (40pts). Regularization with Tensorflow

- using L2 penalty (Ridge)
- using L1 penalty (Lasso)
- using matrix (gradient descent)
- using scikit-learn linear regression model
- using TensorFlow gradient descent method

Logistic Regression

Binary classification

- Logistic regression is commonly used for binary classification.
- The output of classification is categorical value instead of continuous value.
- The output of binary classification is 1 or 0.
- For example,
  - Email: Spam / Not spam,
  - Online transaction: Fraudulent (Yes / No)

Logistic Regression

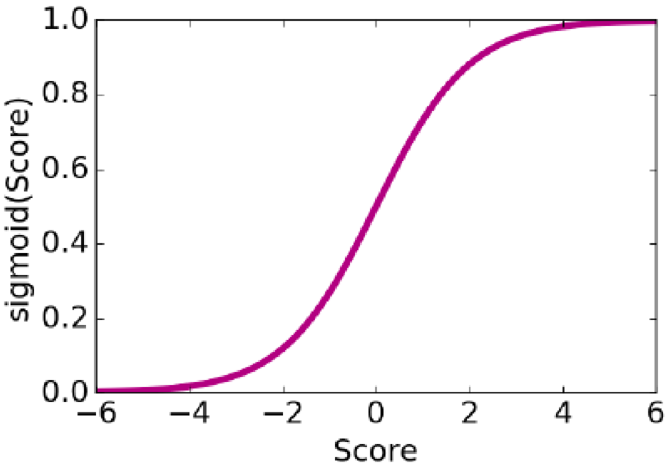
Given x, the probability if y = 1

$$\hat{y} = P(y = 1|x)$$

$$P = \sigma(W^T x + b)$$

Logistic function (sigmoid)

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



<Machine Learning, Emily Fox & Carlos Guestrin>

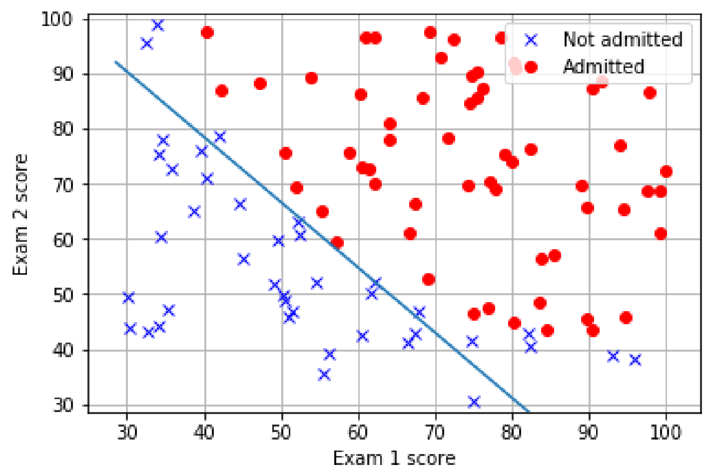
Logistic regression

Model Prediction

$$P(y = 1|x)$$

$$= \frac{1}{1+\exp(-W^T x - b)}$$

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



SJSU SAN JOSÉ STATE UNIVERSITY

Logistic Regression

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})]$$

If y = 1 $$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i})]$$

If y = 0 $$J = -\frac{1}{m}\sum_{i=1}^{m}[\log(1 - \widehat{y^i})]$$

<Machine Learning, Emily Fox & Carlos Guestrin>

Logistic Regression

Gradient (or derivative of cost funciton)

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\left(\widehat{y^i} - y^i\right)x_j^i]$$

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\left(P - y^i\right)x_j^i]$$

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\left(\sigma(W^T x + b) \ - y^i\right)x_j^i]$$

Logistic Regression

Regularization

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})] + \frac{2\lambda}{m}\sum_{j=1}^{n} W_j^2$$

gradient

$$\frac{\partial J}{\partial W_0} = -\frac{1}{m}\sum_{i=1}^{m}[\,(\widehat{y^i} - y^i)x_j^i] \qquad \text{for j=0}$$

$$\frac{\partial J}{\partial W_j} = -\frac{1}{m}\sum_{i=1}^{m}[\,(\widehat{y^i} - y^i)x_j^i]+\frac{\lambda}{m}W_j \qquad \text{for j} \geq 1$$

Confusion matrix

		Predicted label	
		(+1)	(-1)
True label	(+1)	1503 **tp**	203 **fn**
	(-1)	905 **fp**	1845 **tn**

Accuracy : (tp + tn) / (tp + fp + fn + tn) * 100

Precision : tp / predict positive = tp / (tp + fp) * 100

Recall (true positive rate) : tp / actual positive = tp / (tp + fn) * 100

True negative rate : tn / actual negative = tn / (fp + tn) * 100

Precision

Fraction of positive predictions that are correct.

$$\text{precision} = \frac{\#\ \text{true positives}}{\#\ \text{true positives} + \#\ \text{false positives}}$$

<Machine Learning, Emily Fox & Carlos Guestrin>

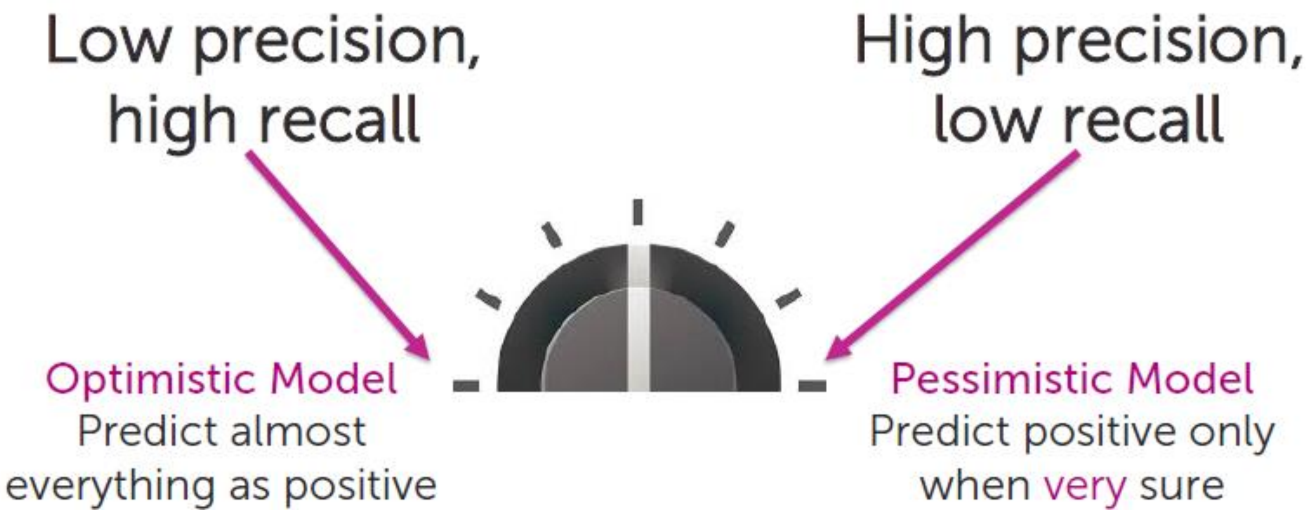
Recall

Fraction of positive data points correctly classified

$$\text{Recall} = \frac{\#\ \text{true positives}}{\#\ \text{true positives} + \#\ \text{false negatives}}$$

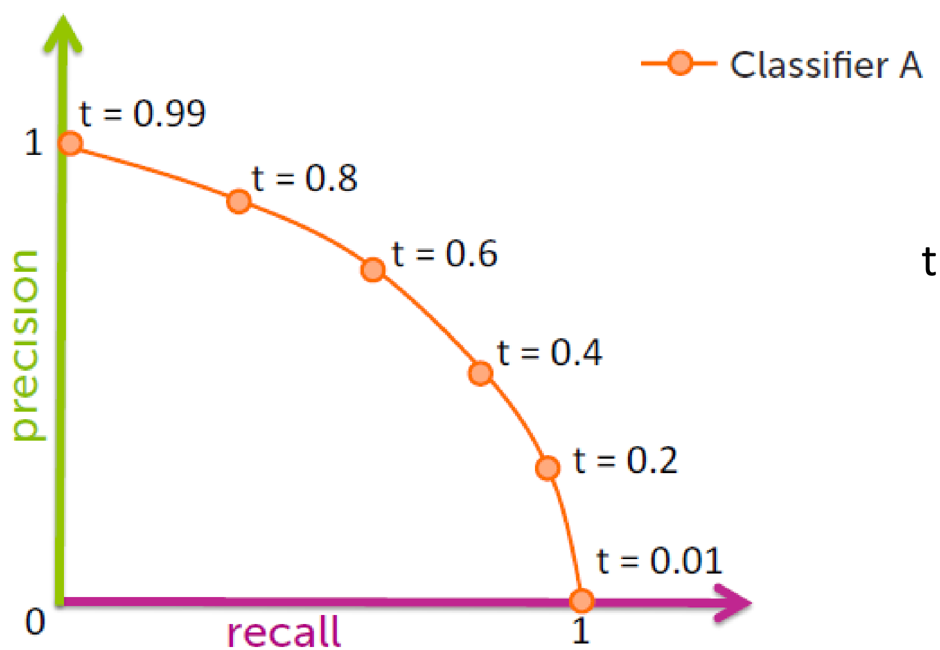
<Machine Learning, Emily Fox & Carlos Guestrin>

Tradeoff Precision & Recall



<Machine Learning, Emily Fox & Carlos Guestrin>

Precision & Recall curve



t : threshold

<Machine Learning, Emily Fox & Carlos Guestrin>

Mini-batch Gradient Descent

- Batch Gradient Descent

Computes the gradients based on full training set

Ex) Offline learning

- Stochastic Gradient Descent

Computes just one instance

Ex) Online learning

- Mini-batch Gradient Descent

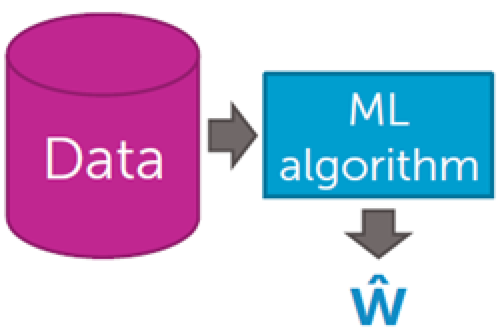
Computes the gradients on small random sets of instances

<Hands-On ML, Aurelien Geron>

Batch vs online learning

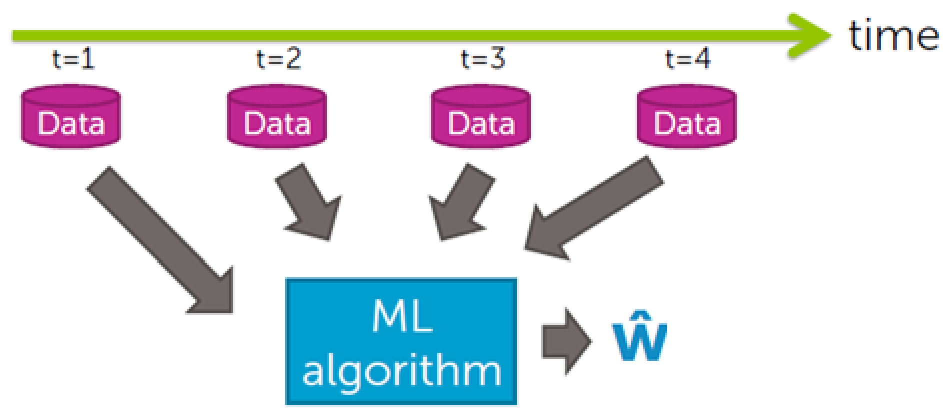
Batch learning

- All data is available at start of training time



Online learning

- Data arrives (streams in) over time
  - Must train model as data arrives!



<Machine Learning, Emily Fox & Carlos Guestrin>

Stochastic gradient descent

Batch Gradient descent

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\,(\widehat{y}^i - y^i)x_j^i]$$

(Sum over data points)

Stochastic Gradient descent

$$\frac{\partial J}{\partial W} = -(\widehat{y}^i - y^i)x_j^i$$

(Each time, pick different data point)

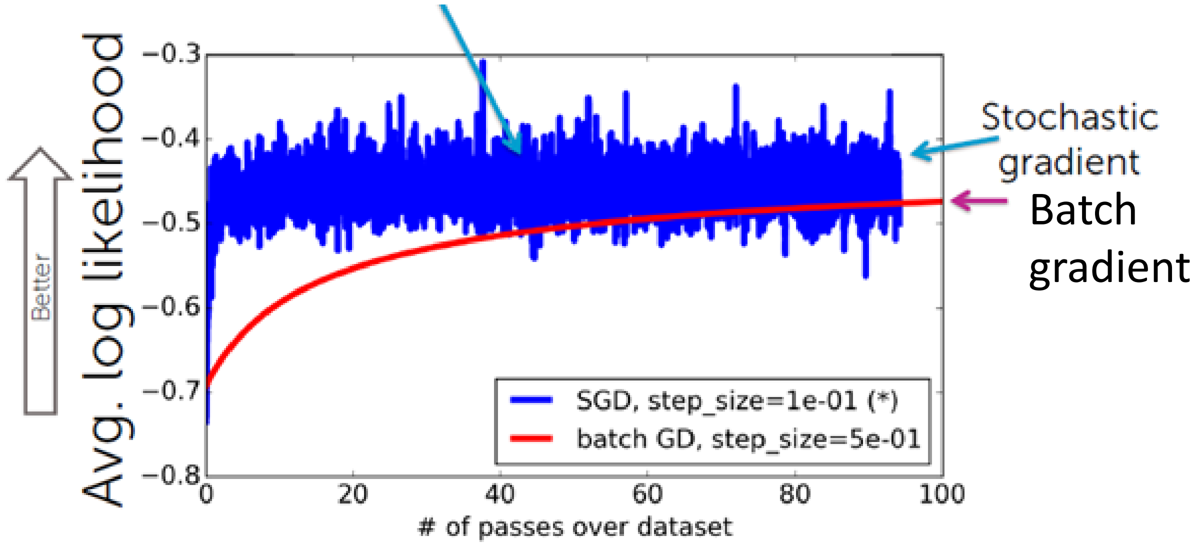
Batch vs. Stochastic gradient

Algorithm	Time per iteration	Total time to convergence for large data: In theory	Total time to convergence for large data: In practice	Sensitivity to parameters
Batch Gradient	Slow for large data	Slower	Often slower	Moderate
Stochastic gradient	Always fast	Faster	Often faster	Very high

<Machine Learning, Emily Fox & Carlos Guestrin>

Batch gradient descent vs stochastic gradient descent

Consider only 'average' value in stochastic gradient



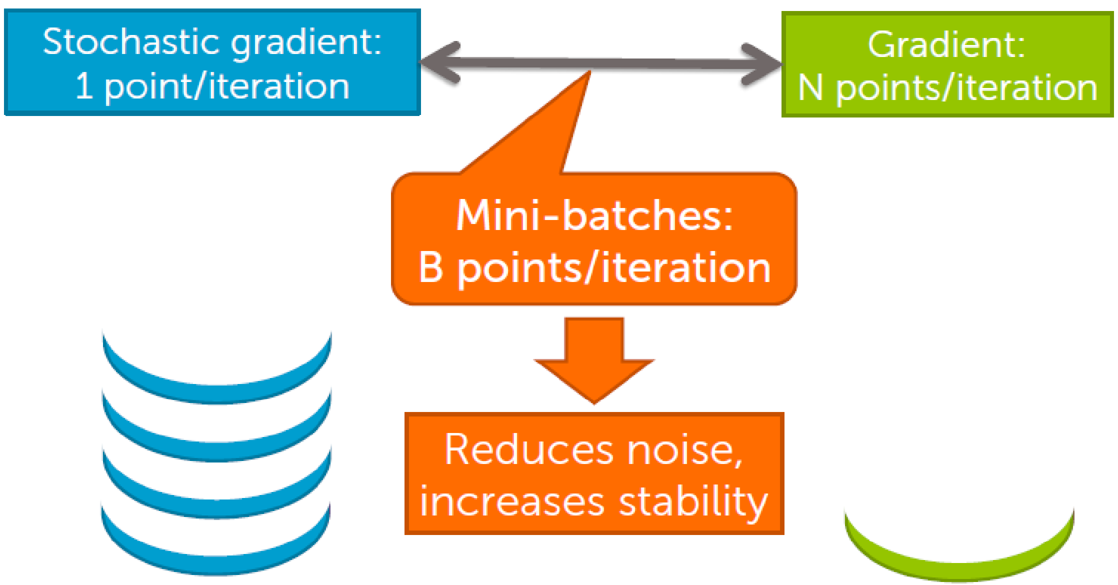
<Machine Learning, Emily Fox & Carlos Guestrin>

Stochastic gradient

- Tiny change to gradient ascent
- Much better scalability
- Huge impact in real-world
- Very tricky to get right in practice

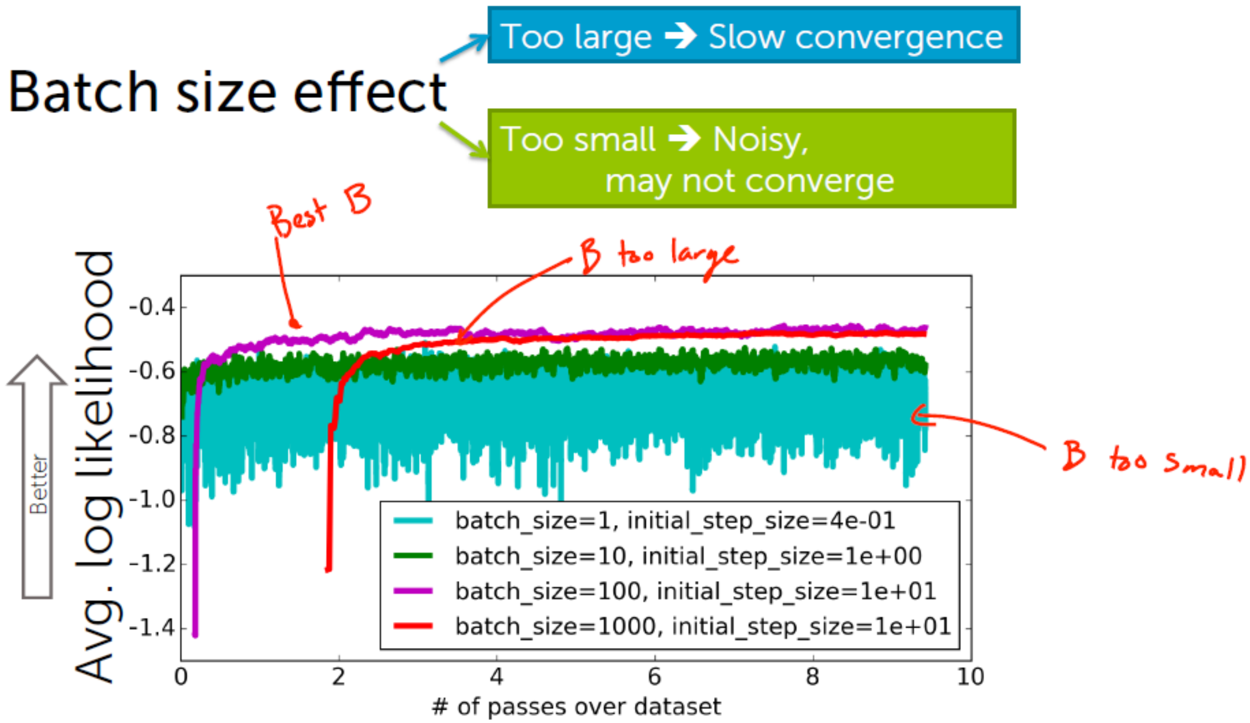
<Machine Learning, Emily Fox & Carlos Guestrin>

Batch / stochastic: two extremes



<Machine Learning, Emily Fox & Carlos Guestrin>

Batch size effect in mini-batch



<Machine Learning, Emily Fox & Carlos Guestrin>

Regression with Tensorflow

Load data and set up X, y variable

```
import tensorflow as tf
import numpy as np
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
m,n = housing.data.shape
mean = np.mean(housing.data,axis=0)
std = np.std(housing.data, axis=0)
normal_housing_data = (housing.data - mean)/std
normal_housing_data_plus_bias = np.c_[np.ones((m,1)),normal_housing_data]
X = tf.constant(normal_housing_data_plus_bias, dtype = tf.float32, name  = "X")
y = tf.constant(housing.target.reshape(-1,1), dtype = tf.float32, name = "y")
```

Regression with Tensorflow

L2 regularization

```
theta = tf.Variable(tf.random_uniform([n+1,1],-1.0,1.0), name = "theta")
y_pred = tf.matmul(X,theta,name = "Predictions")
error = y_pred - y
rmse = tf.sqrt(tf.reduce_mean(tf.square(error)), name = "rmse")
scale = 0.1
learning_rate = 0.01
base_loss = tf.reduce_mean(tf.square(error), name = "loss")
reg_loss = tf.reduce_sum(tf.square(theta))
loss = tf.add(base_loss, scale/m*reg_loss)
gradients = 2/m * tf.add(tf.matmul(tf.transpose(X),error),scale*theta)
training_op = tf.assign(theta, theta - learning_rate * gradients)
```

Regression with Tensorflow

L1 regularization

```
theta = tf.Variable(tf.random_uniform([n+1,1],-1.0,1.0), name = "theta")
y_pred = tf.matmul(X,theta,name = "Predictions")
error = y_pred - y
rmse = tf.sqrt(tf.reduce_mean(tf.square(error)), name = "rmse")
scale = 0.1
base_loss = tf.reduce_mean(tf.square(error), name = "loss")
reg_loss = tf.reduce_sum(tf.abs(theta))
loss = tf.add(base_loss, scale/m*reg_loss)
gradients = 2/m * tf.add(tf.matmul(tf.transpose(X),error),scale*theta)
training_op = tf.assign(theta, theta - learning_rate * gradients)
```

Summary

- Logistic regression
- Binary classification
- Confusion matrix: Accuracy, Precision, Recall
- Mini-batch gradient descent
- Regression with tensorflow