Regularization and Feature Selection

Legal Notices and Disclaimers

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

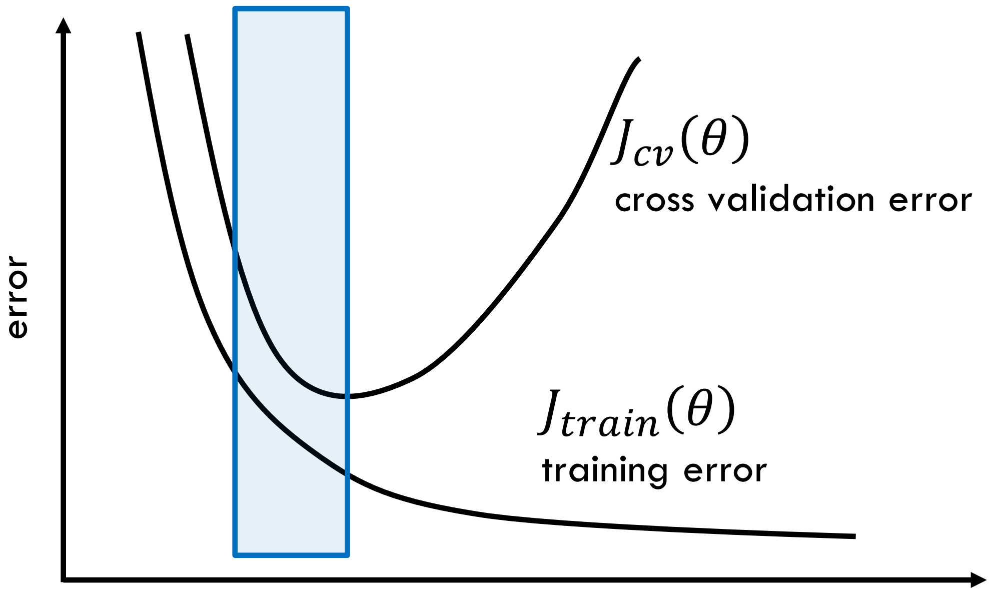
Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

This sample source code is released under the Intel Sample Source Code License Agreement.

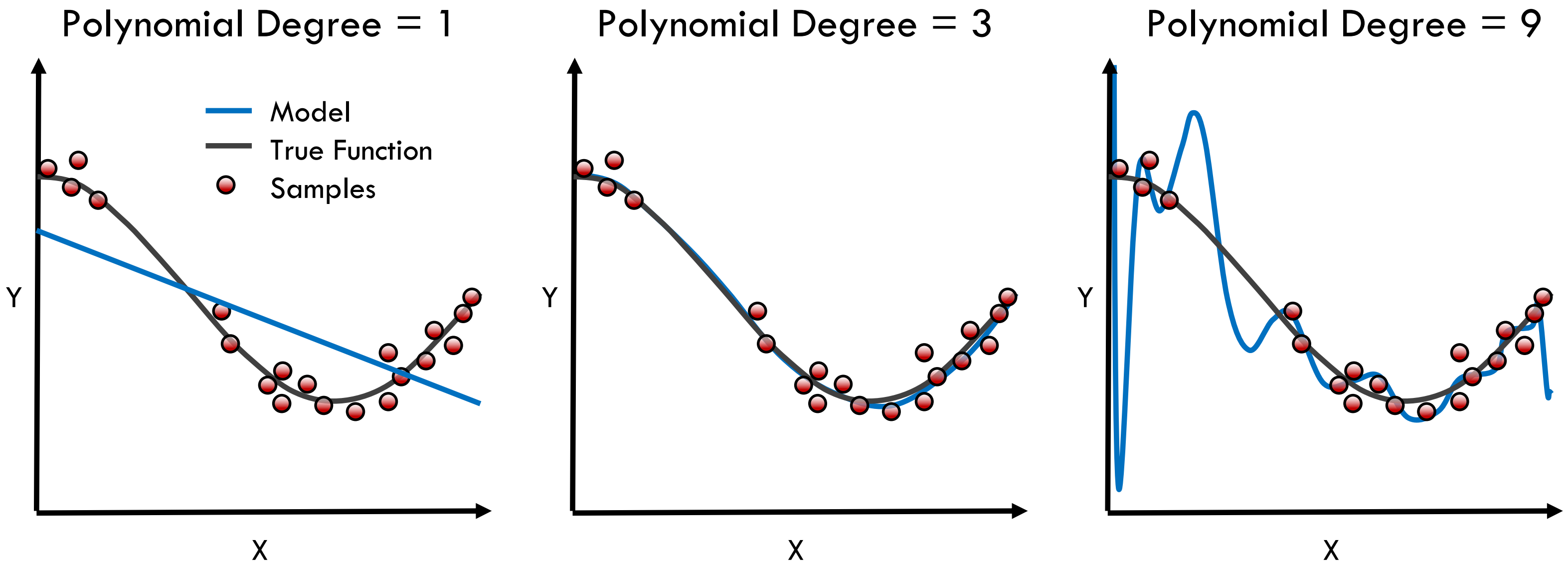
Learning Objectives

- Explain cost functions, regularization, feature selection, and hyper-parameters
- Summarize complex statistical optimization algorithms like gradient descent and its application to linear regression
- Apply Intel® Extension for Scikit-learn* to leverage underlying compute capabilities of hardware

Model Complexity vs Error

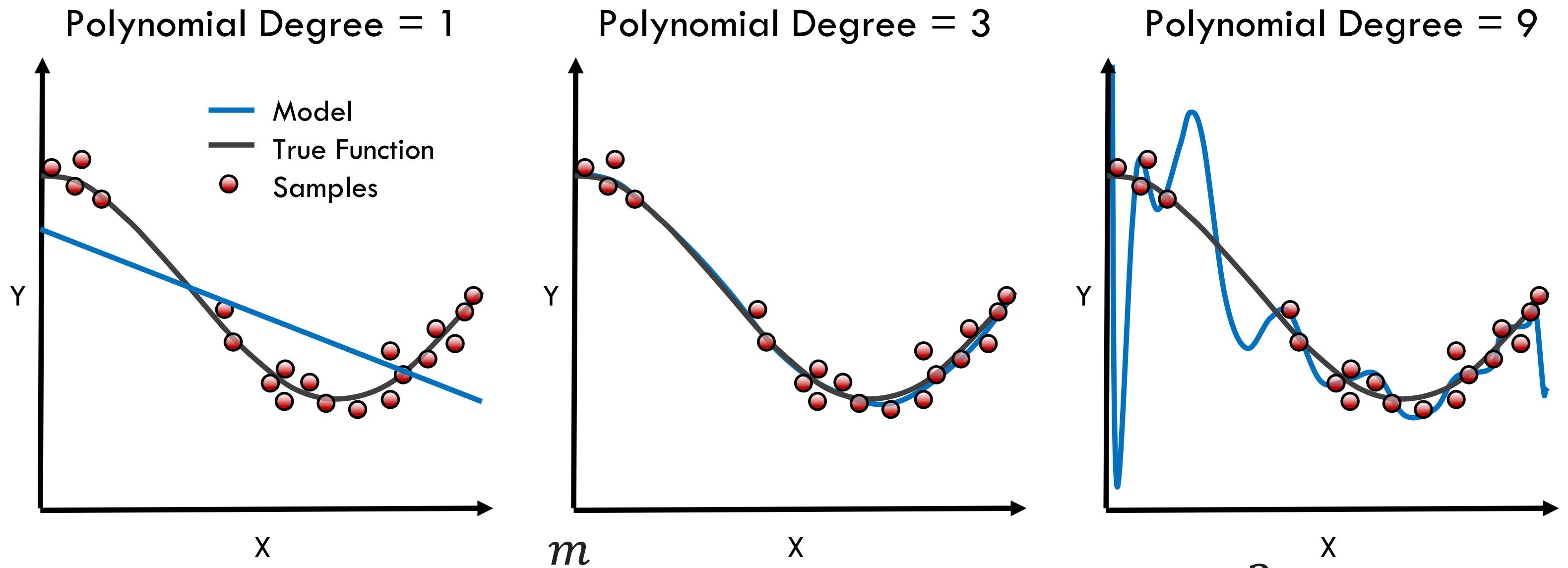


Preventing Under- and Overfitting



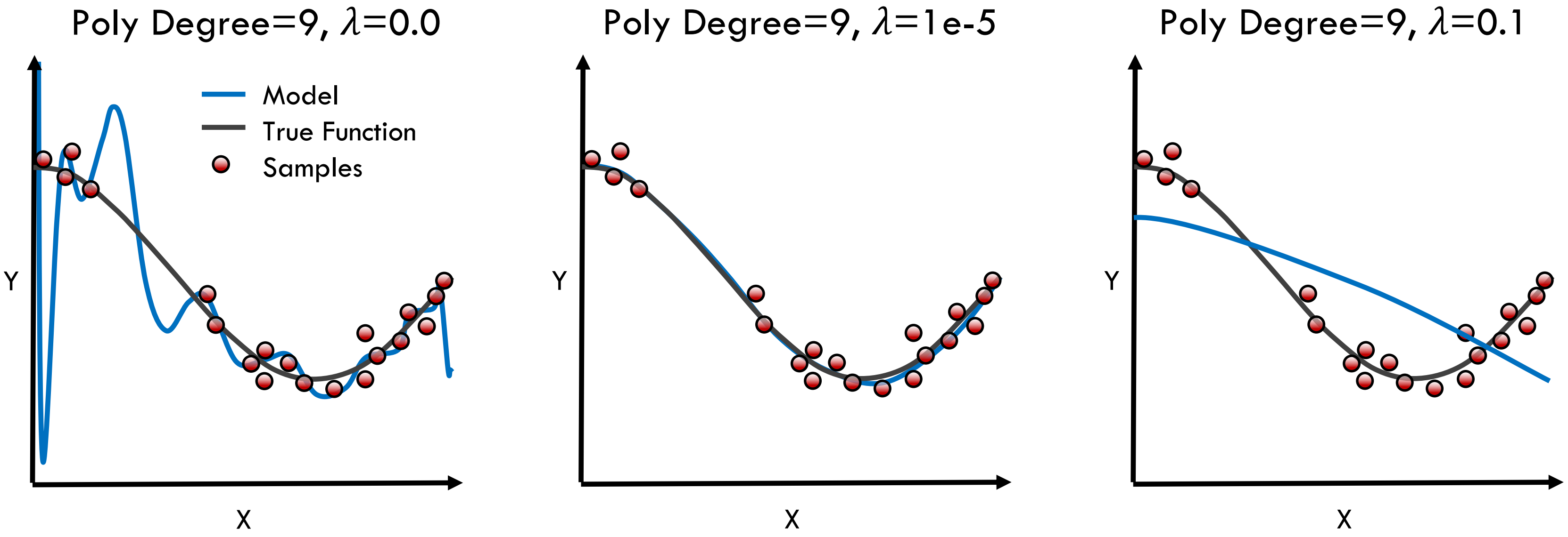
- How to use a degree 9 polynomial and prevent overfitting?

Preventing Under- and Overfitting



$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

Regularization



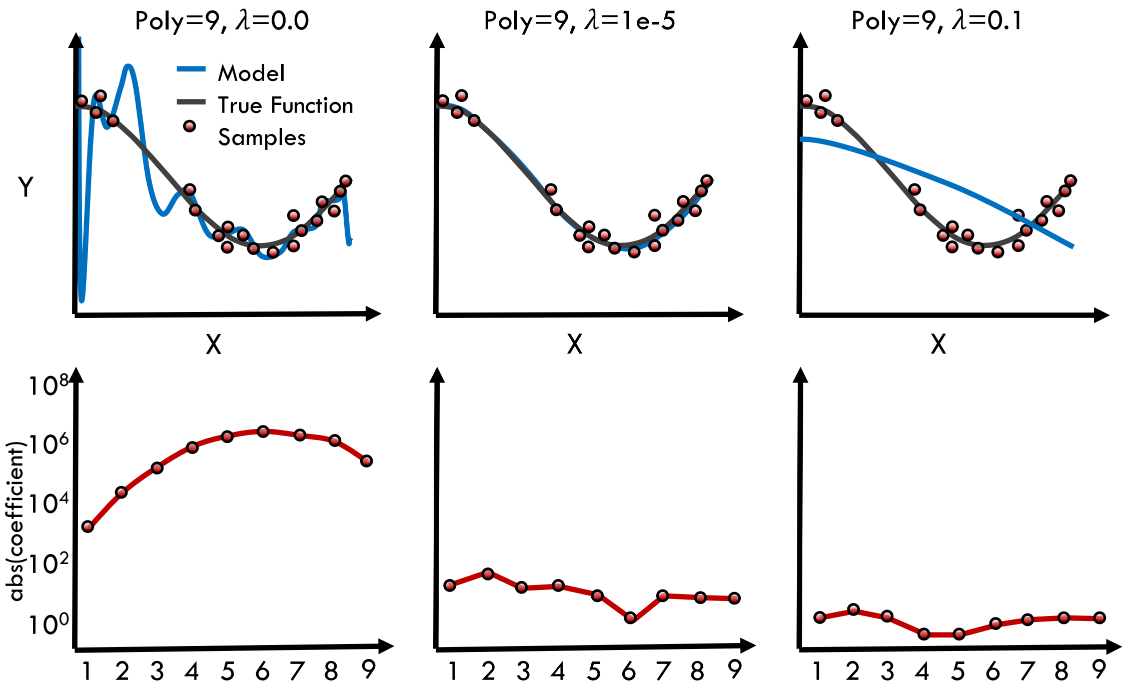
$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda \sum_{j=1}^{k} {\beta_j}^2$$

Ridge Regression (L2)

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda \sum_{j=1}^{k} {\beta_j}^2$$

- Penalty shrinks magnitude of all coefficients
- Larger coefficients strongly penalized because of the squaring

Effect of Ridge Regression on Parameters



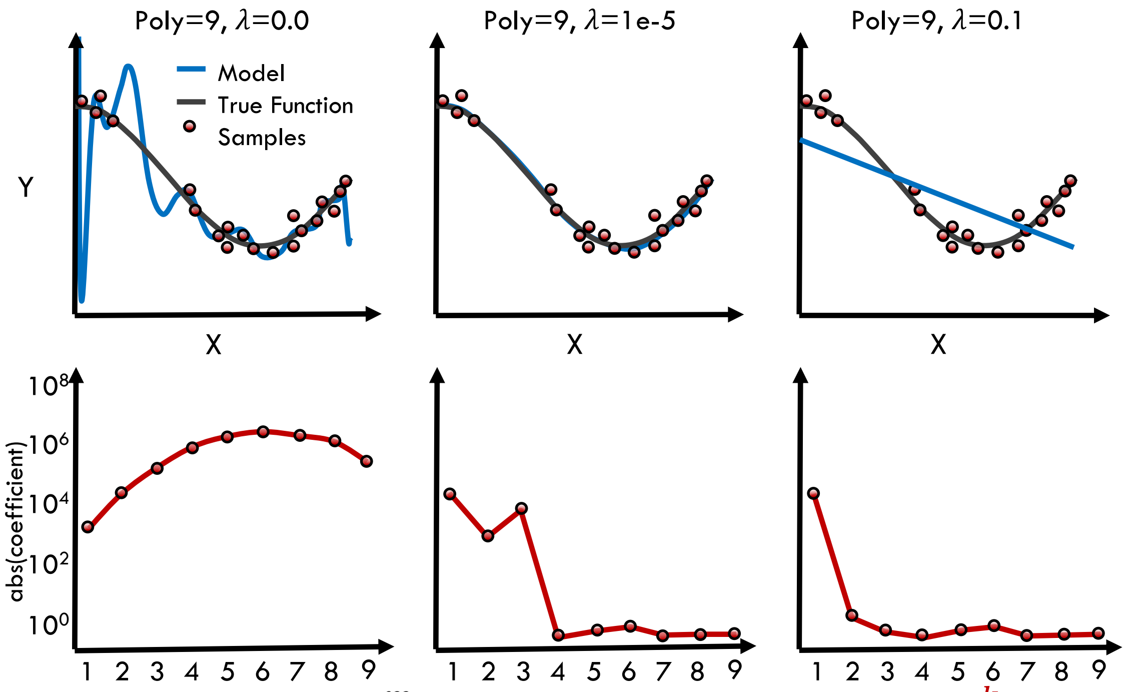
$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda \sum_{j=1}^{k} \beta_j{}^2$$

Lasso Regression (L1)

$$J(\beta_0, \beta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2 + \lambda\sum_{j=1}^{k}|\beta_j|$$

- Penalty selectively shrinks some coefficients
- Can be used for feature selection
- Slower to converge than Ridge regression

Effect of Lasso Regression on Parameters



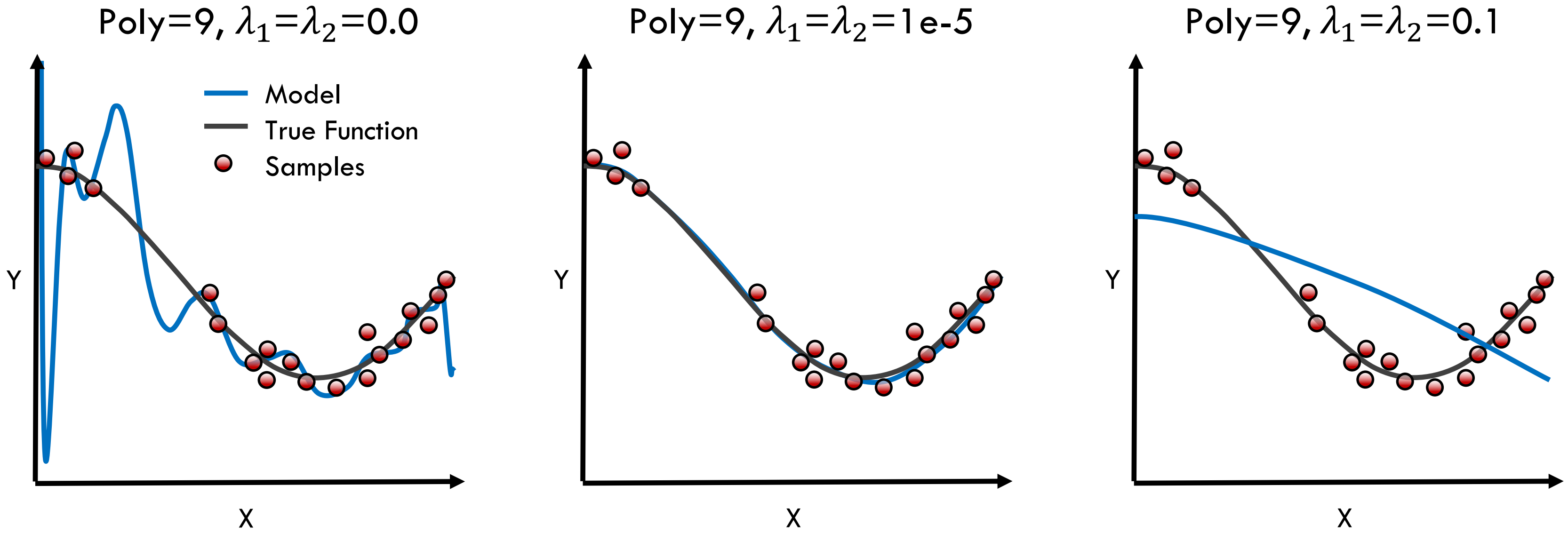
$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda \sum_{j=1}^{k} \left| \beta_j \right|$$

Elastic Net Regularization

$$J(\beta_0, \beta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2 + \lambda_1 \sum_{j=1}^{k}\left|\beta_j\right| + \lambda_2 \sum_{j=1}^{k}\beta_j^2$$

- Compromise of both Ridge and Lasso regression
- Requires tuning of additional parameter that distributes regularization penalty between L1 and L2

Elastic Net Regularization



$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda_1 \sum_{j=1}^{k} |\beta_j| + \lambda_2 \sum_{j=1}^{k} \beta_j^2$$

Hyperparameters and Their Optimization

- Regularization coefficients (λ_1 and λ_2) are empirically determined

Use Test Data to Tune λ?

0	2013-11-22	The Hunger Games: Catching Fire	130000000	424668047	Francis Lawrence	PG-13	146
1	2013-05-03	Iron Man 3	200000000	409013994	Shane Black	PG-13	129
2	2013-11-22	Frozen	150000000	400738009	Chris BuckJennifer Lee	PG	108
3	2013-07-03	Despicable Me 2	76000000	368061265	Pierre CoffinChris Renaud	PG	98
4	2013-06-14	Man of Steel	225000000	291045518	Zack Snyder	PG-13	143
5	2013-10-04	Gravity	[illegible]	[illegible]	…uaron	PG-13	91
6	2013-06-21	Monsters University	[illegible]	[illegible]	…on	G	107
7	2013-12-13	The Hobbit: The De…	[illegible]	[illegible]	…kson	PG-13	161
8	2013-05-24	Fast & Furious 6	160000000	238679850	Justin Lin	PG-13	130
9	2013-03-08	Oz The Great and Powerful	215000000	234911825	Sam Raimi	PG	127
10	2013-05-16	Star Trek Into Darkness	190000000	228778661	J.J. Abrams	PG-13	123
11	2013-11-08	Thor: The Dark World	170000000	206362140	Alan Taylor	PG-13	120
12	2013-06-21	World War Z	190000000	202359711	Marc Forster	PG-13	116
13	2013-03-22	The Croods	135000000	187168425	Kirk De MiccoChris Sanders	PG	98
14	2013-06-28	The Heat	[illegible]	[illegible]	[illegible]	R	117
15	2013-08-07	We're the Millers	[illegible]	[illegible]	…larshall Thurber	R	110
16	2013-12-13	American Hustle	[illegible]	[illegible]	…Russell	R	138
17	2013-05-10	The Great Gatsby	105000000	144840419	Baz Luhrmann	PG-13	143

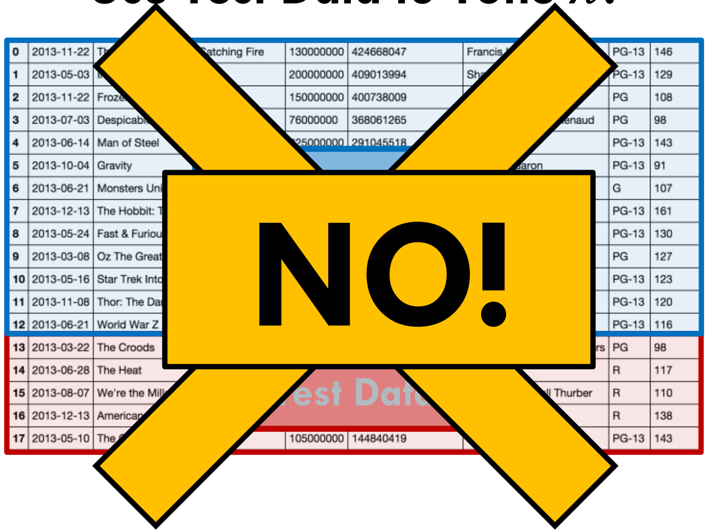
Training Data

Test Data

Hyperparameters and Their Optimization

- Regularization coefficients (λ_1 and λ_2) are empirically determined
- Want value that generalizes—do not use test data for tuning

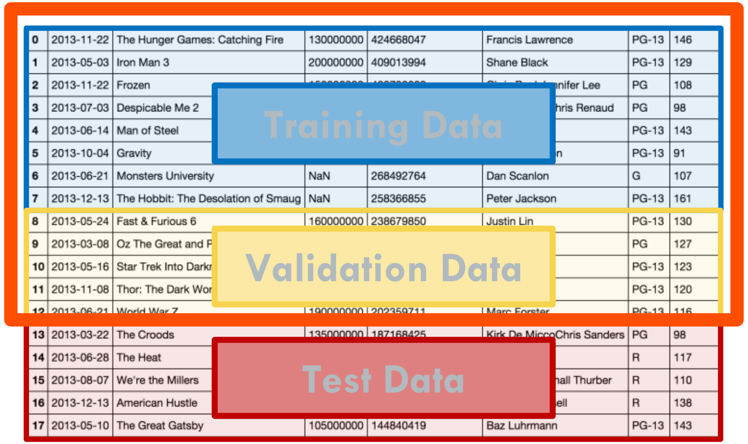
Use Test Data to Tune λ?



Hyperparameters and Their Optimization

- Regularization coefficients (λ_1 and λ_2) are empirically determined
- Want value that generalizes—do not use test data for tuning
- Create additional split of data to tune hyperparameters—validation set

Tune λ with Cross Validation



Ridge Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Ridge
```

To use the Intel® Extension for Scikit-learn* variant of this algorithm:

- **Install Intel® oneAPI AI Analytics Toolkit (AI Kit)**
- **Add the following two lines of code after the above code:**

```
import patch_sklearn
patch_sklearn()
```

Ridge Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Ridge
```

Ridge Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Ridge
```

Create an instance of the class

```
RR = Ridge(alpha=1.0)
```

Ridge Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Ridge
```

Create an instance of the class

```
RR = Ridge(alpha=1.0)
```

regularization parameter

Ridge Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Ridge
```

Create an instance of the class

```
RR = Ridge(alpha=1.0)
```

Fit the instance on the data and then predict the expected value

```
RR = RR.fit(X_train, y_train)
y_predict = RR.predict(X_test)
```

Ridge Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Ridge
```

Create an instance of the class

```
RR = Ridge(alpha=1.0)
```

Fit the instance on the data and then predict the expected value

```
RR = RR.fit(X_train, y_train)
y_predict = RR.predict(X_test)
```

The RidgeCV class will perform cross validation on a set of values for alpha.

Lasso Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Lasso
```

Create an instance of the class

```
LR = Lasso(alpha=1.0)
```

Fit the instance on the data and then predict the expected value

```
LR = LR.fit(X_train, y_train)
y_predict = LR.predict(X_test)
```

The LassoCV class will perform cross validation on a set of values for alpha.

Lasso Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import Lasso
```

Create an instance of the class

```
LR = Lasso(alpha=1.0)
```

regularization parameter

Fit the instance on the data and then predict the expected value

```
LR = LR.fit(X_train, y_train)
y_predict = LR.predict(X_test)
```

The LassoCV class will perform cross validation on a set of values for alpha.

Elastic Net Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import ElasticNet
```

Create an instance of the class

```
EN = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

Fit the instance on the data and then predict the expected value

```
EN = EN.fit(X_train, y_train)
y_predict = EN.predict(X_test)
```

The ElasticNetCV class will perform cross validation on a set of values for l1_ratio and alpha.

Elastic Net Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import ElasticNet
```

Create an instance of the class

```
EN = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

alpha is the regularization parameter

Fit the instance on the data and then predict the expected value

```
EN = EN.fit(X_train, y_train)
y_predict = EN.predict(X_test)
```

The ElasticNetCV class will perform cross validation on a set of values for l1_ratio and alpha.

Elastic Net Regression: The Syntax

Import the class containing the regression method

```
from sklearn.linear_model import ElasticNet
```

Create an instance of the class

```
EN = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

l1_ratio distributes alpha to L1/L2

Fit the instance on the data and then predict the expected value

```
EN = EN.fit(X_train, y_train)
y_predict = EN.predict(X_test)
```

The ElasticNetCV class will perform cross validation on a set of values for l1_ratio and alpha.

Feature Selection

- Regularization performs feature selection by shrinking the contribution of features

Feature Selection

- Regularization performs feature selection by shrinking the contribution of features
- For L1-regularization, this is accomplished by driving some coefficients to zero

Feature Selection

- Regularization performs feature selection by shrinking the contribution of features
- For L1-regularization, this is accomplished by driving some coefficients to zero
- Feature selection can also be performed by removing features

Why is Feature Selection Important?

- Reducing the number of features is another way to prevent overfitting (similar to regularization)

Why is Feature Selection Important?

- Reducing the number of features is another way to prevent overfitting (similar to regularization)
- For some models, fewer features can improve fitting time and/or results

Why is Feature Selection Important?

- Reducing the number of features is another way to prevent overfitting (similar to regularization)
- For some models, fewer features can improve fitting time and/or results
- Identifying most critical features can improve model interpretability

Recursive Feature Elimination: The Syntax

Import the class containing the feature selection method

```
from sklearn.feature_selection import RFE
```

Create an instance of the class

```
rfeMod = RFE(est, n_features_to_select=5)
```

Fit the instance on the data and then predict the expected value

```
rfeMod = rfeMod.fit(X_train, y_train)
y_predict = rfeMod.predict(X_test)
```

The RFECV class will perform feature elimination using cross validation.

Recursive Feature Elimination: The Syntax

Import the class containing the feature selection method

```
from sklearn.feature_selection import RFE
```

Create an instance of the class

```
rfeMod = RFE(est, n_features_to_select=5)
```

est is an instance of the model to use

Fit the instance on the data and then predict the expected value

```
rfeMod = rfeMod.fit(X_train, y_train)
y_predict = rfeMod.predict(X_test)
```

The RFECV class will perform feature elimination using cross validation.

Recursive Feature Elimination: The Syntax

Import the class containing the feature selection method

```
from sklearn.feature_selection import RFE
```

Create an instance of the class

```
rfeMod = RFE(est, n_features_to_select=5)
```

final number of features

Fit the instance on the data and then predict the expected value

```
rfeMod = rfeMod.fit(X_train, y_train)
y_predict = rfeMod.predict(X_test)
```

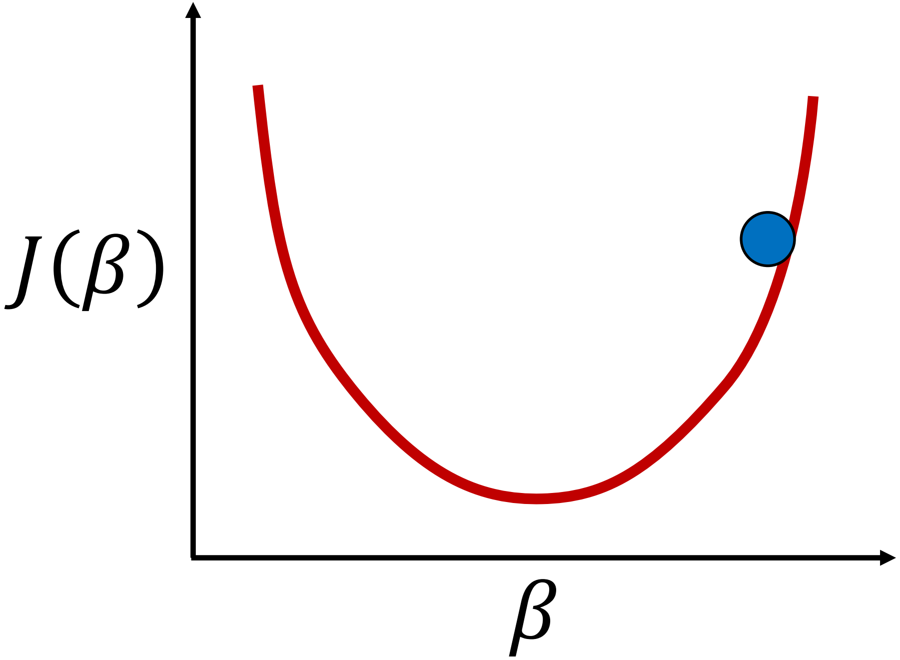
The RFECV class will perform feature elimination using cross validation.

Software

Gradient Descent

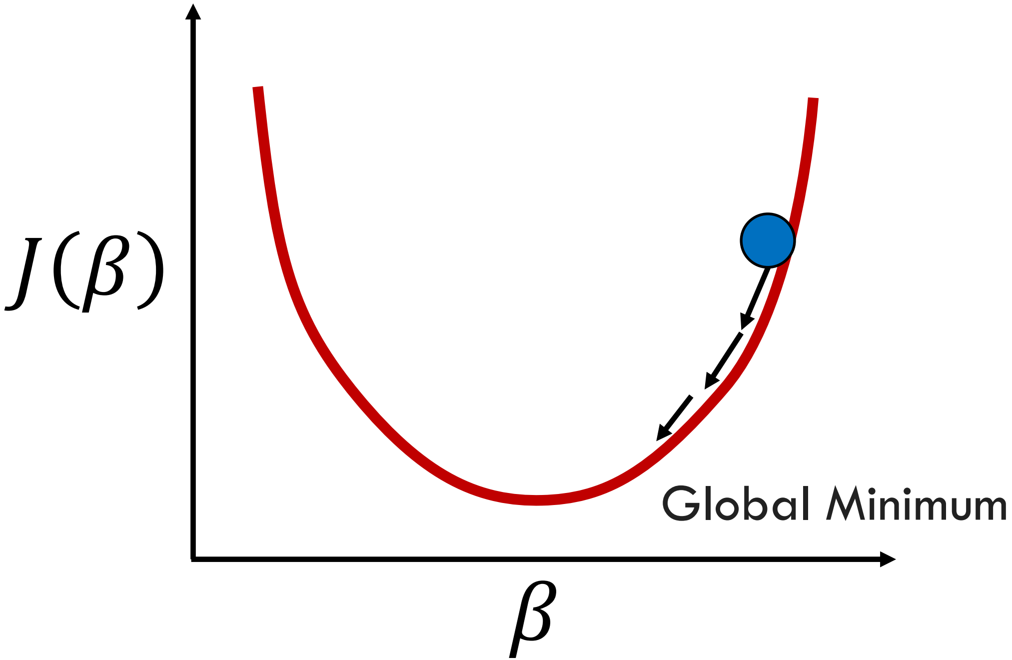
Gradient Descent

Start with a cost function J(β):



Gradient Descent

Start with a cost function $J(\beta)$:



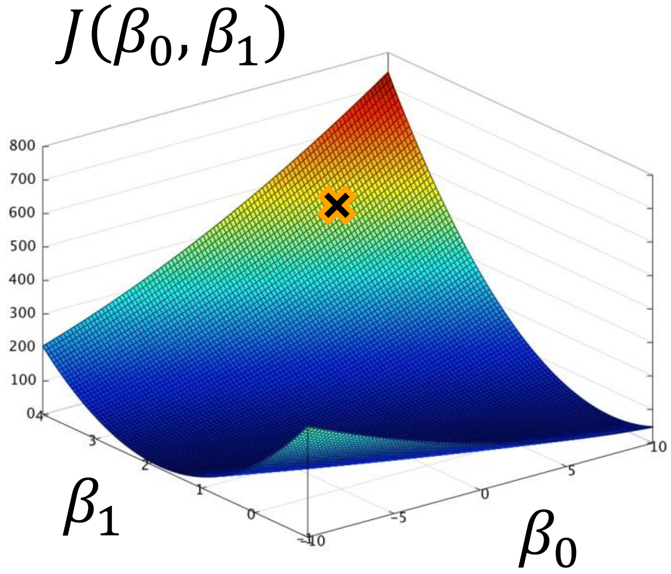
Then gradually move towards the minimum.

Gradient Descent with Linear Regression

- Now imagine there are two parameters
  (β_0, β_1)

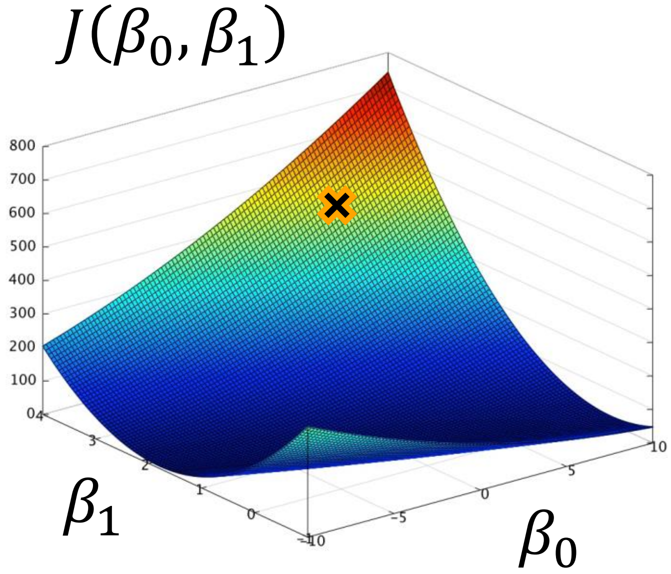
Gradient Descent with Linear Regression

- Now imagine there are two parameters (β_0, β_1)
- This is a more complicated surface on which the minimum must be found
- How can we do this without knowing what



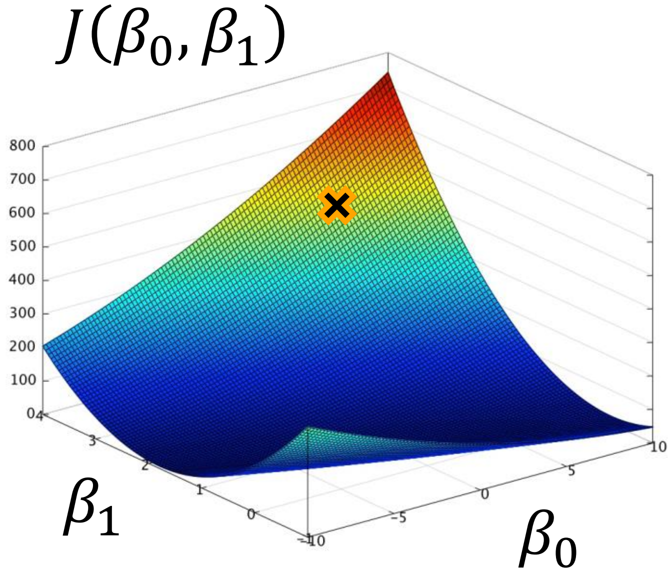
Gradient Descent with Linear Regression

- Now imagine there are two parameters (β_0, β_1)
- This is a more complicated surface on which the minimum must be found
- How can we do this without knowing what $J(\beta_0, \beta_1)$ looks like?



Gradient Descent with Linear Regression

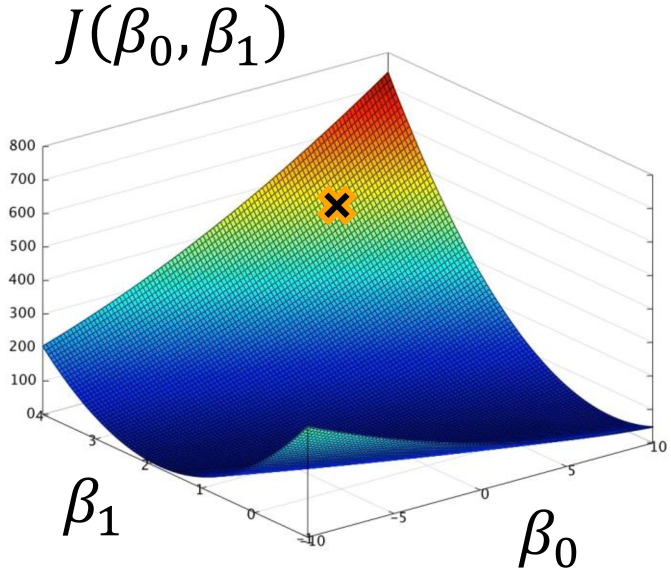
- Compute the gradient, $\nabla J(\beta_0, \beta_1)$, which points in the direction of the biggest increase!
- -$\nabla J(\beta_0, \beta_1)$ (negative gradient) points to the biggest decrease at that point!



Gradient Descent with Linear Regression

- The gradient is the a vector whose coordinates consist of the partial derivatives of the parameters

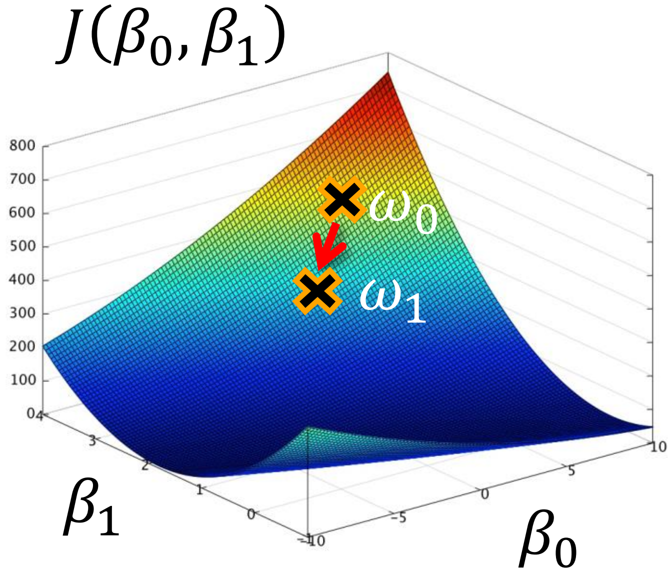
$$\nabla J(\beta_0, \ldots, \beta_n) = < \frac{\partial J}{\partial \beta_0}, \ \ldots, \frac{\partial J}{\partial \beta_n} >$$



Gradient Descent with Linear Regression

- Then use the gradient (∇) and the cost function to calculate the next point (ω_1) from the current one (ω_0):

$$\omega_1 = \omega_0 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

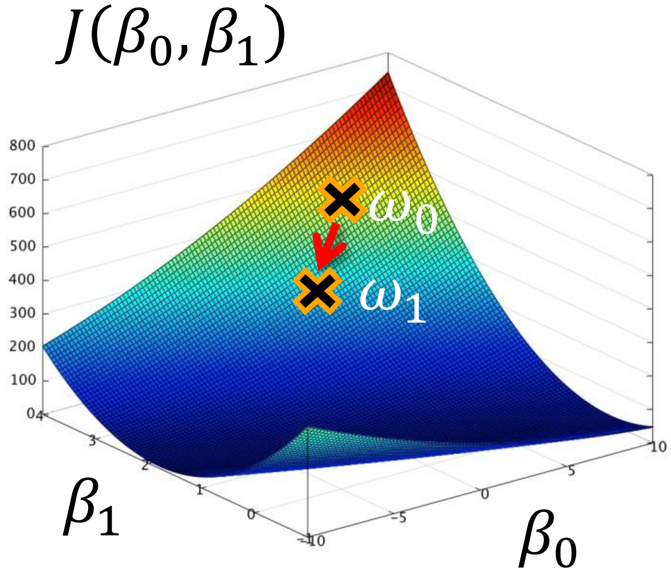


Gradient Descent with Linear Regression

- Then use the gradient (∇) and the cost function to calculate the next point (ω_1) from the current one (ω_0):

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\sum_{i=1}^{m}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$

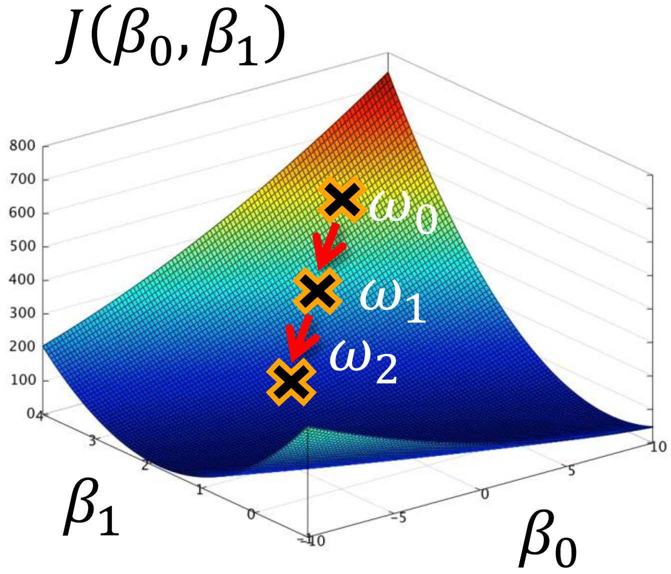
- The learning rate (α) is a tunable parameter that determines step size



Gradient Descent with Linear Regression

- Each point can be iteratively calculated from the previous one

$$\omega_2 = \omega_1 - \alpha\nabla\frac{1}{2}\sum_{i=1}^{m}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$

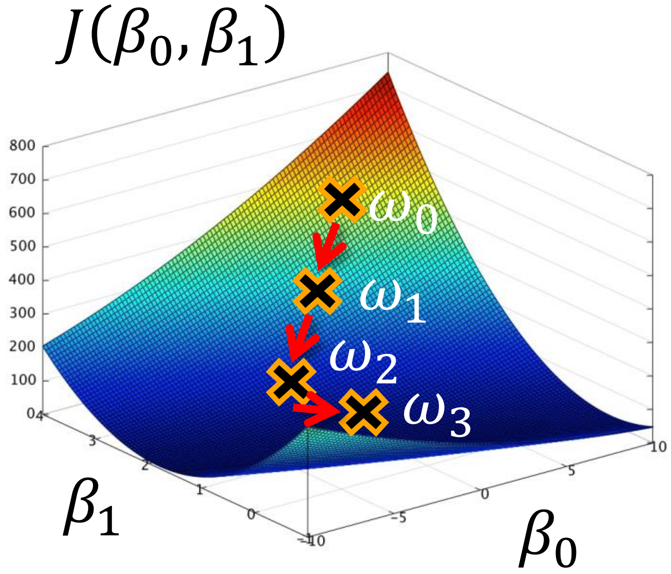


Gradient Descent with Linear Regression

- Each point can be iteratively calculated from the previous one

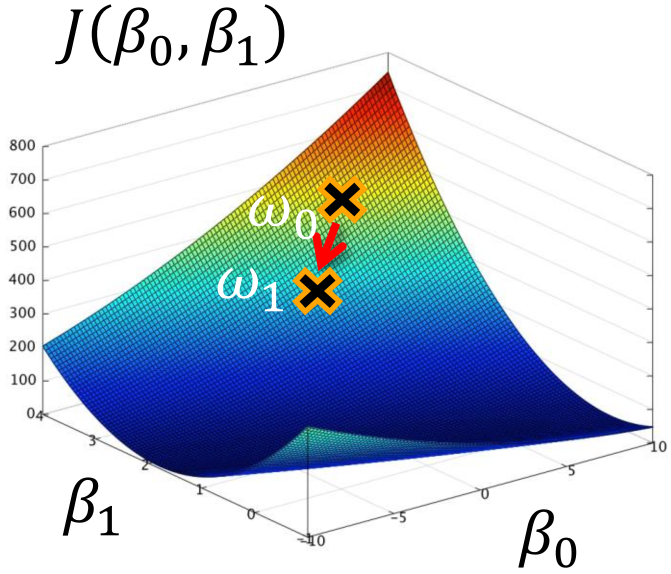
$$\omega_2 = \omega_1 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

$$\omega_3 = \omega_2 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$



Stochastic Gradient Descent

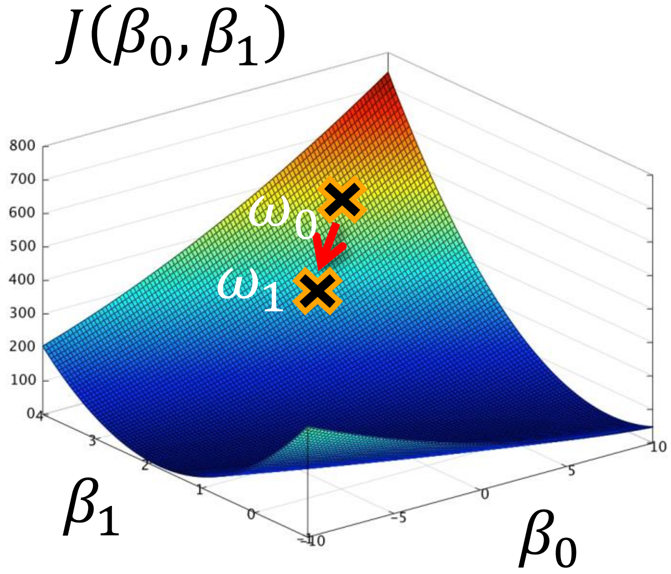
- Use a single data point to determine the gradient and cost function instead of all the data



Stochastic Gradient Descent

- Use a single data point to determine the gradient and cost function instead of all the data

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\sum_{i=1}^{m}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$

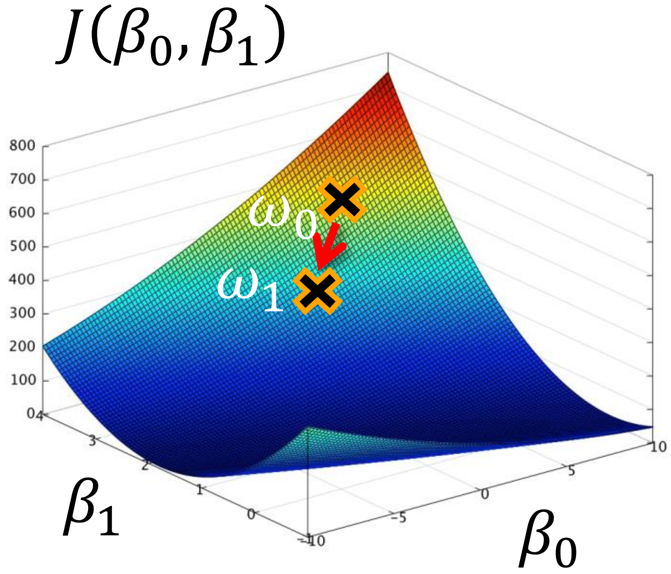


Stochastic Gradient Descent

- Use a single data point to determine the gradient and cost function instead of all the data

$$\omega_1 = \omega_0 - \alpha \nabla \frac{1}{2} \cancel{\sum_{i=1}^{m}} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

$$\boxed{\omega_1 = \omega_0 - \alpha \nabla \frac{1}{2} \left(\left(\beta_0 + \beta_1 x_{obs}^{(0)} \right) - y_{obs}^{(0)} \right)^2}$$



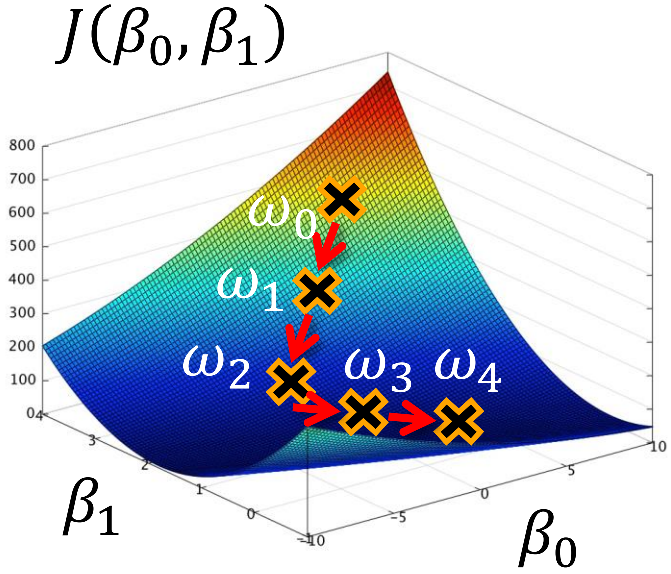
Stochastic Gradient Descent

- Use a single data point to determine the gradient and cost function instead of all the data

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(0)}\right) - y_{obs}^{(0)}\right)^2$$

$$\dots$$

$$\omega_4 = \omega_3 - \alpha\nabla\frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(3)}\right) - y_{obs}^{(3)}\right)^2$$



Stochastic Gradient Descent

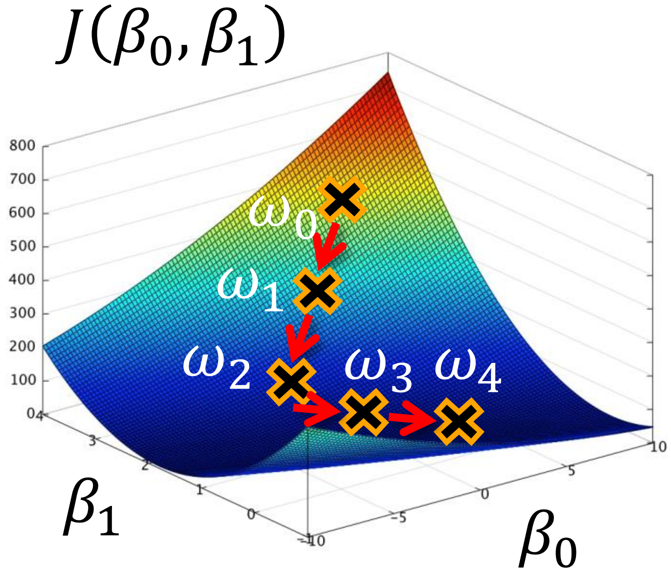
- Use a single data point to determine the gradient and cost function instead of all the data

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(0)}\right) - y_{obs}^{(0)}\right)^2$$

$$\dots$$

$$\omega_4 = \omega_3 - \alpha\nabla\frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(3)}\right) - y_{obs}^{(3)}\right)^2$$

- Path is less direct due to noise in single data point—"stochastic"

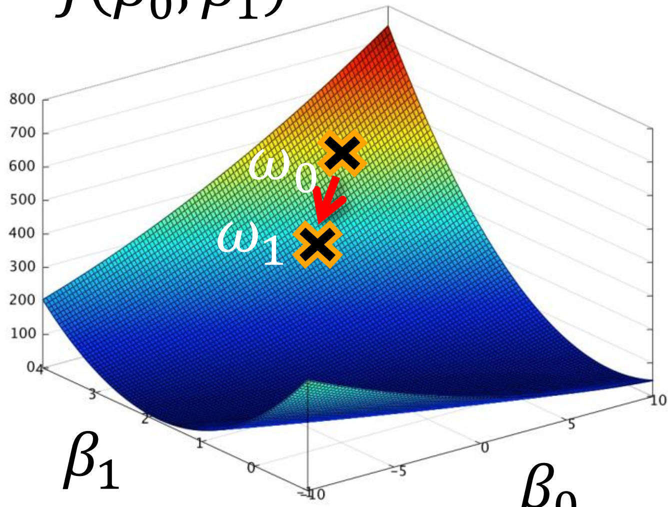


Mini Batch Gradient Descent

- Perform an update for every n training examples

$$\omega_1 = \omega_0 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{n} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

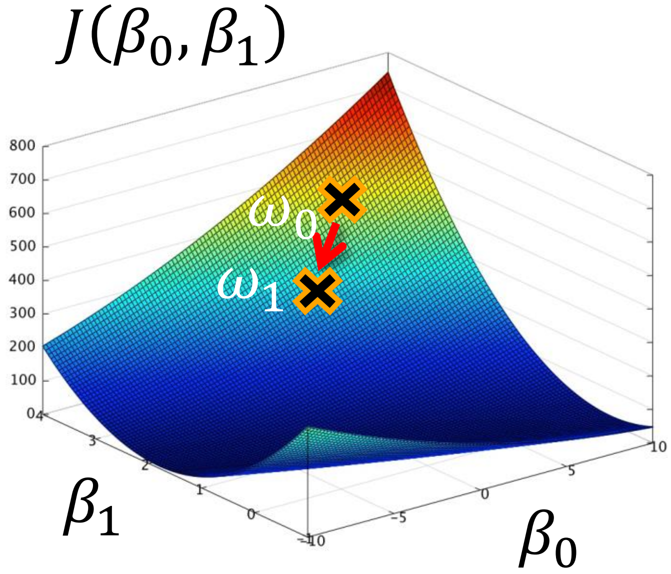
$J(\beta_0, \beta_1)$



Mini Batch Gradient Descent

- Perform an update for every n training examples

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\boxed{\sum_{i=1}^{n}}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$



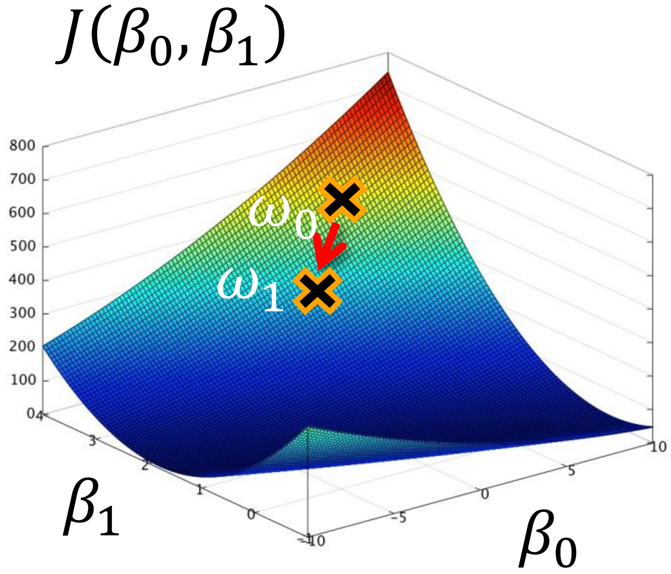
Mini Batch Gradient Descent

- Perform an update for every n training examples

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\sum_{i=1}^{n}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$

Best of both worlds:

- Reduced memory relative to "vanilla" gradient descent
- Less noisy than stochastic gradient descent



Mini Batch Gradient Descent

- Mini batch implementation typically used for neural nets

Mini Batch Gradient Descent

- Mini batch implementation typically used for neural nets
- Batch sizes range from 50-256 points

Mini Batch Gradient Descent

- Mini batch implementation typically used for neural nets
- Batch sizes range from 50-256 points
- Trade off between batch size and learning rate (α)

Mini Batch Gradient Descent

- Mini batch implementation typically used for neural nets
- Batch sizes range from 50–256 points
- Trade off between batch size and learning rate (α)
- Tailor learning rate schedule: gradually reduce learning rate during a given epoch

Stochastic Gradient Descent Regression: Syntax

Import the class containing the regression model

```
from sklearn.linear_model import SGDRegressor
```

Stochastic Gradient Descent Regression: Syntax

Import the class containing the regression model

```
from sklearn.linear_model import SGDRegressor
```

Create an instance of the class

```
SGDreg = SGDRregressor(loss='squared_loss',
                alpha=0.1, penalty='l2')
```

Stochastic Gradient Descent Regression: Syntax

Import the class containing the regression model

```
from sklearn.linear_model import SGDRegressor
```

Create an instance of the class

```
SGDreg = SGDRregressor(loss='squared_loss',
            alpha=0.1, penalty='l2')
```

squared_loss = linear regression

Stochastic Gradient Descent Regression: Syntax

Import the class containing the regression model

```
from sklearn.linear_model import SGDRegressor
```

Create an instance of the class

```
SGDreg = SGDRregressor(loss='squared_loss',
                alpha=0.1, penalty='l2')
```

regularization parameters

Stochastic Gradient Descent Regression: Syntax

Import the class containing the regression model

```
from sklearn.linear_model import SGDRegressor
```

Create an instance of the class

```
SGDreg = SGDRregressor(loss='squared_loss',
                       alpha=0.1, penalty='l2')
```

Fit the instance on the data and then transform the data

```
SGDreg = SGDreg.fit(X_train, y_train)
y_pred = SGDreg.predict(X_test)
```

Stochastic Gradient Descent Regression: Syntax

Import the class containing the regression model

```
from sklearn.linear_model import SGDRegressor
```

Create an instance of the class

```
SGDreg = SGDRregressor(loss='squared_loss',
                       alpha=0.1, penalty='l2')
```

Fit the instance on the data and then transform the data

```
SGDreg = SGDreg.partial_fit(X_train, y_train)
y_pred = SGDreg.predict(X_test)
```

mini-batch version

Stochastic Gradient Descent Regression: The Syntax

Import the class containing the regression model

```
from sklearn.linear_model import SGDRegressor
```

Create an instance of the class

```
SGDreg = SGDRregressor(loss='squared_loss',
                alpha=0.1, penalty='l2')
```

Fit the instance on the data and then transform the data

```
SGDreg = SGDreg.fit(X_train, y_train)

y_pred = SGDreg.predict(X_test)
```

Other loss methods exist: epsilon_insensitive, huber, etc.

Stochastic Gradient Descent Classification: The Syntax

Import the class containing the classification model

```
from sklearn.linear_model import SGDClassifier
```

Stochastic Gradient Descent Classification: The Syntax

Import the class containing the classification model

```
from sklearn.linear_model import SGDClassifier
```

Create an instance of the class

```
SGDclass = SGDClassifier(loss='log',
                         alpha=0.1, penalty='l2')
```

Stochastic Gradient Descent Classification: The Syntax

Import the class containing the classification model

```
from sklearn.linear_model import SGDClassifier
```

Create an instance of the class

```
SGDclass = SGDClassifier(loss='log',
                   alpha=0.1, penalty='l2')
```

log loss =
logistic regression

Stochastic Gradient Descent Classification: The Syntax

Import the class containing the classification model

```
from sklearn.linear_model import SGDClassifier
```

Create an instance of the class

```
SGDclass = SGDClassifier(loss='log',
                         alpha=0.1, penalty='l2')
```

Fit the instance on the data and then transform the data

```
SGDclass = SGDclass.fit(X_train, y_train)

y_pred = SGDclass.predict(X_test)
```

Stochastic Gradient Descent Classification: The Syntax

Import the class containing the classification model

```
from sklearn.linear_model import SGDClassifier
```

Create an instance of the class

```
SGDclass = SGDClassifier(loss='log',
                    alpha=0.1, penalty='l2')
```

Fit the instance on the data and then transform the data

```
SGDclass = SGDclass.partial_fit(X_train, y_train)
y_pred = SGDclass.predict(X_test)
```

mini-batch version

Stochastic Gradient Descent Classification: The Syntax

Import the class containing the classification model

```
from sklearn.linear_model import SGDClassifier
```

Create an instance of the class

```
SGDclass = SGDClassifier(loss='log',
                    alpha=0.1, penalty='l2')
```

Fit the instance on the data and then transform the data

```
SGDclass = SGDclass.fit(X_train, y_train)
y_pred = SGDclass.predict(X_test)
```

Other loss methods exist: hinge, squared_hinge, etc.

Stochastic Gradient Descent Classification: The Syntax

Import the class containing the classification model

```
from sklearn.linear_model import SGDClassifier
```

Create an instance of the class

```
SGDclass = SGDClassifier(loss='log',
                alpha=0.1, penalty='l2')
```

Fit the instance on the data and then transform the data

```
SGDclass = SGDclass.fit(X_train, y_train)

y_pred = SGDclass.predict(X_test)
```

Other loss methods exist: hinge, squared_hinge, etc.

See SVM lecture (week 7)

intel®

Software