outrageously
AMBITIOUS

Linear Regression

Duke

PRATT SCHOOL of
ENGINEERING

Which are linear models?

A $y = w_0$

B $y = w_0 + w_1 x_1$

C $y = w_0 + w_1 x_1 + w_2 x_2$

D $y = w_0 + w_1 x_1^2 + w_2 x_2^{0.4}$

E $y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2$

F $y = w_0 + w_1 \int \sqrt[3]{x_1}\, dx_1 + w_2 g(x_2) + w_3 median(x_1, x_2, x_3)$

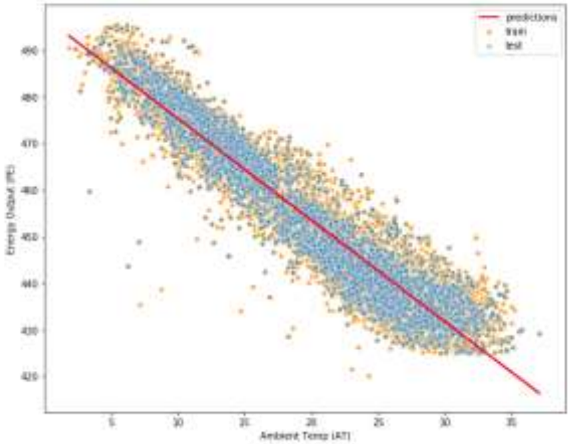
Linear models are **linear in the parameters *w***. They can be used to model non-linear relationships between inputs & outputs

Duke
PRATT SCHOOL *of* ENGINEERING

Linear regression

- What is linear regression?
  - Model which assumes linear relationships between the features and targets

- Why should we care about linear regression?
  - Forms the basis of more complex ML models
  - Can be surprisingly effective if you know how to use it
  - Great first model to apply to get a benchmark
  - Helps us understand relationships between inputs and outputs (feature and targets)



Simple vs. multiple linear regression

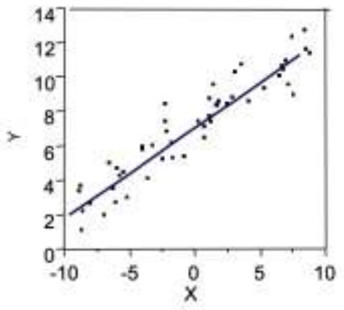
- Simple linear regression models y as a function of a single feature x
- Multiple linear regression models y as a function of a feature vector x containing multiple features x_1,x_2...x_n

Simple linear regression

$$\hat{y} = w_0 + w_1 x$$

Multiple linear regression

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p$$



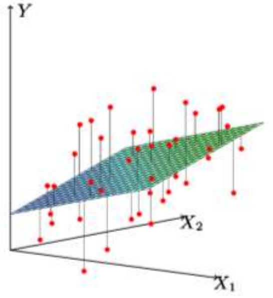


Image source: https://www.alsharif.info/iom530

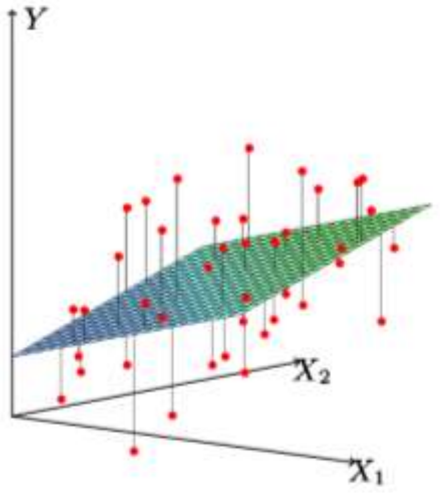
The Linear Regression Model

For an observation (x,y):

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p \Leftrightarrow \hat{y} = \sum_{i=0}^{p} w_i x_i \Leftrightarrow \hat{y} = w^T x$$

where $x_0 \triangleq 1$

- The parameters are easy to interpret
  - w_0 is the intercept (the average value for y if all the x's are 0)
  - w_i is the average increase in y when x_i is increased by 1 and **all other x's are held constant**

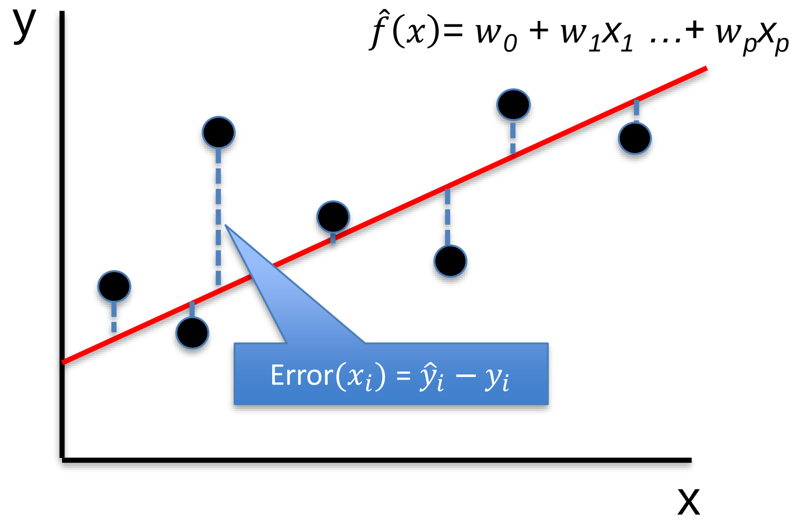


Estimating the parameters

- We seek a function $\hat{f}(x) = w_0 + w_1x_1 \ldots + w_px_p$ that minimizes the total error $\sum_{i=1}^{N}(\hat{y}_i - y_i)$
- Alternatively, we seek to minimize the *Sum of Squared Error (SSE)* = $\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$
- This error function we seek to minimize is called the ***cost function*** or ***loss function***

$$Cost\ function\ J(w) = \sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$$= \sum_{i=1}^{N}(w^T x_i - y_i)^2$$



y
$\hat{f}(x) = w_0 + w_1x_1 \ldots + w_px_p$
Error(x_i) = $\hat{y}_i - y_i$
x

Estimating the parameters

$$Cost\ function\ J(w) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

- We want to minimize this cost function
- We cannot change the data (*X*,y) so we adjust *w*
- How do we find values for $w_0 \ldots w_i$ that minimize this cost function?
  - **Closed form solution** – take the derivative of the cost function and set equal to 0
  - Apply a **gradient descent** search algorithm to converge on the optimal values

Duke
PRATT SCHOOL *of* ENGINEERING

Vectorized form

For a single observation:

$$\hat{y} = \sum_{i=0}^{p} w_i x_i \qquad \text{where } x_0=1$$

Written in vectorized form:

$$\hat{y} = w^T x = \begin{bmatrix} w_0 & w_1 & w_2 & \dots & w_p \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_p \end{bmatrix}$$

For many observations:

$$\hat{y} = Xw \qquad \begin{array}{ccccc} \text{Predictions} & & \text{Data} & \text{Coefficients} \\ \begin{bmatrix} N \times 1 \end{bmatrix} & = & \begin{bmatrix} N \times p{+}1 \end{bmatrix} & \begin{bmatrix} p{+}1 \times 1 \end{bmatrix} \\ \hat{y} & & X & w \end{array}$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \dots \\ \hat{y}_N \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,p} \\ 1 & x_{2,1} & \dots & x_{2,p} \\ & & \dots & \\ 1 & x_{N,1} & \dots & x_{N,p} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_p \end{bmatrix}$$

Duke PRATT SCHOOL of ENGINEERING

Closed form solution

Our model is: $$\hat{y} = \sum_{i=0}^{p} w_i x_i$$

Our cost function is: $$J(w) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

We can rewrite this in vectorized form as: $$J(w) = (Xw - y)^T (Xw - y)$$

Closed form solution

Minimize $J(w) = (Xw - y)^T(Xw - y)$

Set derivative wrt w = 0 and solve for w:

$$\nabla_w J(w) = 2(X^T Xw - X^T y) = 0$$

$$X^T Xw - X^T y = 0$$

$X^T Xw = X^T y$ (Normal equation)

$$\boxed{w^* = (X^T X)^{-1} X^T y} \Rightarrow \boxed{\hat{y} = Xw^*}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{bmatrix}$$

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ & & \dots & \\ x_{N,1} & x_{N,2} & \dots & x_{N,p} \end{bmatrix}$$

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_p \end{bmatrix}$$

Reference on inverse of a matrix: https://www.mathsisfun.com/algebra/matrix-inverse.html

Assumptions of linear regression

Linear regression makes **three key assumptions**:

1. Expected value of y for any given x is a linear function of x
2. Errors are independent, normally distributed random variables with 0 mean and constant variance σ^2
3. Feature variables are independent

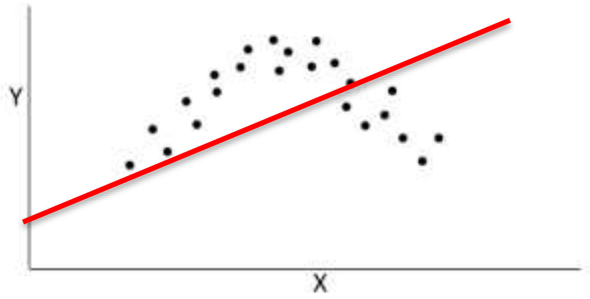
Model **fit issues can result** from violations of these assumptions:

- Non-linearity of the target-features relationship
- Non-constant variance or correlation of error terms
- Collinearity or multicollinearity of feature variables

Checking for problems

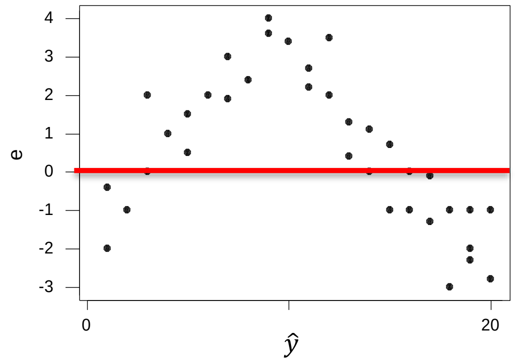
Problem: Non-linearity of the target-features relationship

Method 1: Scatterplot of target vs. each feature



Target should be roughly linearly related to the features (difficult to assess with many features)

Method 2: Scatterplot of residuals (errors) vs features or vs. predictions

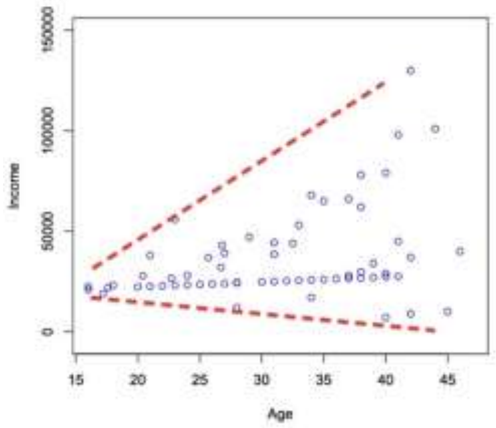


Errors should be uniformly distributed around the 0 line

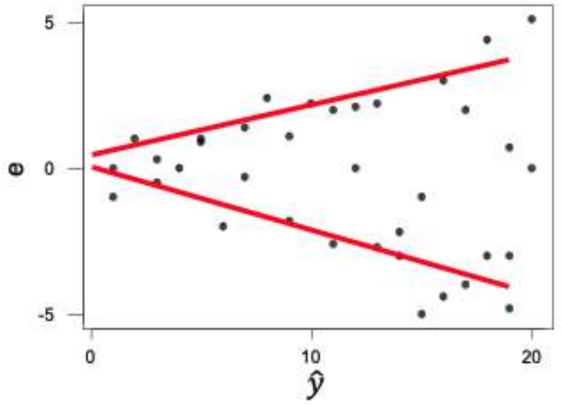
Checking for problems

Problem: Non-constant variance of errors (heteroscedasticity)

Scatterplot of target vs. a feature



Scatterplot of residuals (errors) vs. predictions

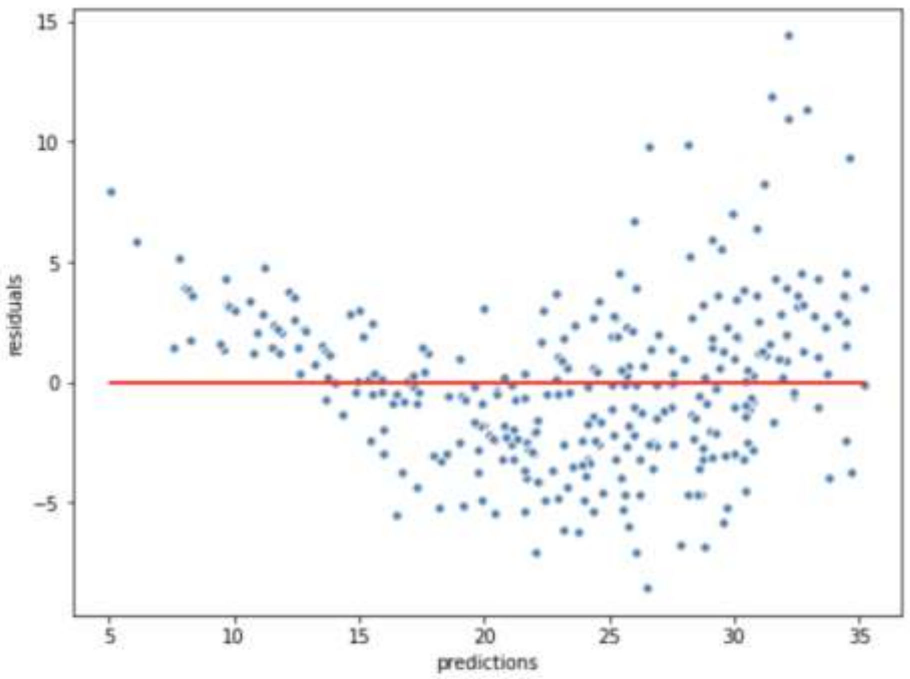


- In the residuals plot, variance should be evenly scattered around the 0 line and should not increase or decrease with the predicted values (no funnel shape)

Checking for problems

Based on the residuals plot, do you see evidence of any problems with using linear regression:

- Non-linearity of the target-features relationship?
- Non-constant variance or correlation of error terms?
- Collinearity or multicollinearity of feature variables?



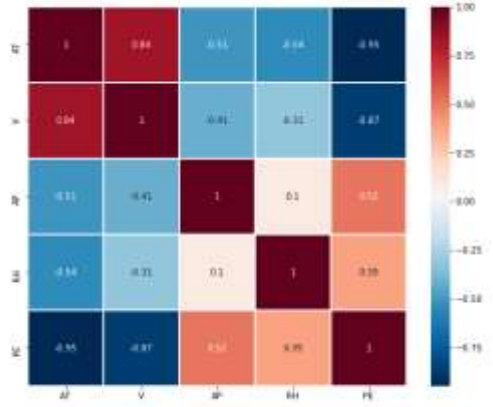
Checking for problems

Problem: Collinearity or multicollinearity of feature variables

Collinearity means that one variable is nearly linearly dependent on another variable. We can detect this by looking at the **correlation** between two features:

$$\rho_{x,y} = \operatorname{cor}(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})\,(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\,\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Visualize with a correlation matrix:



- Severe collinearity or multicollinearity can result in major swings of the correlated variable coefficients as the training data changes, making interpretation difficult
- However, it usually does not significantly impact model's prediction ability

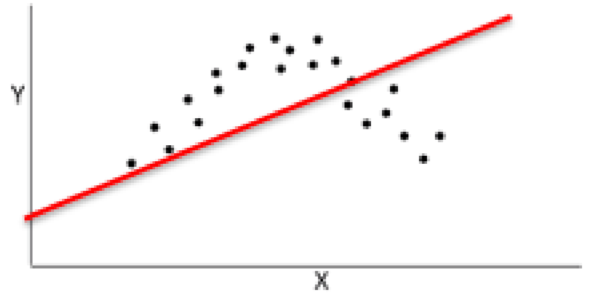
Checking for problems

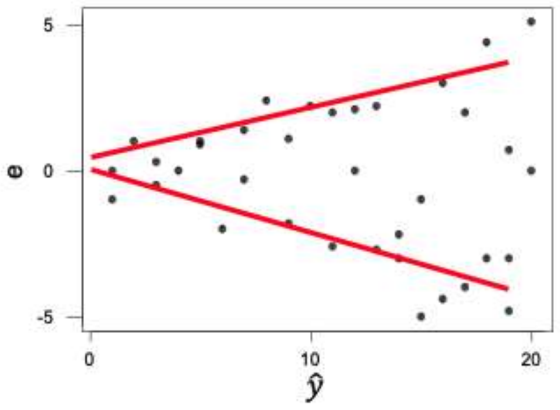
- Violations of the linear regression assumptions do NOT mean that your model has no value
- The purpose of checking for these problems is to guide us in how to improve our fit:
  - Non-linearity may imply we should transform our input feature variables
  - Non-constant variance may indicate we should transform our output target variable
  - Collinearity may mean we should reduce our features to simplify

What do we do if we violate an assumption?

- If we observe non-linearity between features and target:
  - We can transform our input features so that we have a linear relationship between the new features and the target

- If we observe non-constant variance of errors across values of y-hat:
  - We can transform our output so that we have constant variance of error terms





outrageously
AMBITIOUS

Handling Nonlinear Relationships

Duke
PRATT SCHOOL of
ENGINEERING

Feature variable transformation

- To model a nonlinear relationship with a feature, we can transform the feature by some nonlinear function to create a new feature *z*:

$$z = x^a \text{ OR } z = log(x)$$

- We can then use our new feature *z* as an input to our model
- If we can model our target as a linear function of our new feature *z*, our model performance should be improved
- This is often called “polynomial regression”

Duke
PRATT SCHOOL *of* ENGINEERING

Polynomial regression

- Generally when we use polynomial regression with degree = n we include a feature with every degree from 1 to n
- We now need to learn n+1 parameters in our model
- We can easily do this in Scikit Learn using the PolynomialFeatures class to transform our input data before we build a model

Simple degree-1 linear regression:

$$y = w_0 + w_1 x$$

Simple degree-3 linear regression:

$$y = w_0 + w_1 x + \ w_2 x^2 + w_3 x^3$$

Remember: Linear Regression is linear in the coefficients

Polynomial transformation

Degree-1 linear regression

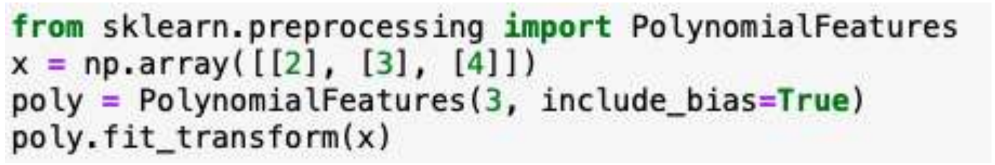
$$y = w_0 + w_1 x_1$$

$$X = \begin{array}{c} X_1 \\ \left[\begin{array}{c} 2 \\ 3 \\ 4 \end{array}\right] \end{array}$$

Degree-3 linear regression

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

$$X = \begin{array}{cccc} X_0 & X_1 & X_2 & X_3 \\ \end{array} \left[\begin{array}{c} 1,2,4,8 \\ 1,3,9,27 \\ 1,4,16,64 \end{array}\right]$$



```python
from sklearn.preprocessing import PolynomialFeatures
x = np.array([[2], [3], [4]])
poly = PolynomialFeatures(3, include_bias=True)
poly.fit_transform(x)
```

Interaction Terms

- Linear Regression assumes that the effect of each feature is independent of the other features
  - If we increase feature x_1 by 1 unit, we increase the target by w_1 units (coefficient of x_1)
- In real life, we sometimes have synergy effects (or “interaction” effects), of features with each other
  - Changing the value of one feature causes the relationship of another feature to the target to change
- These can be observed graphically, but usually rely on domain expertise to know they exist

Example – Power Outage Prediction

- Power outages are generally caused by wind blowing trees onto lines
- Thus, wind speed is a key feature
- However, when the ground is very wet from recent rain, smaller increases in wind speed can cause more trees to fall
- Thus, the presence of this feature (recent rain) changes the relationship between another feature (wind speed) and target (# of outages)

Using Interaction Terms

Normal linear regression:

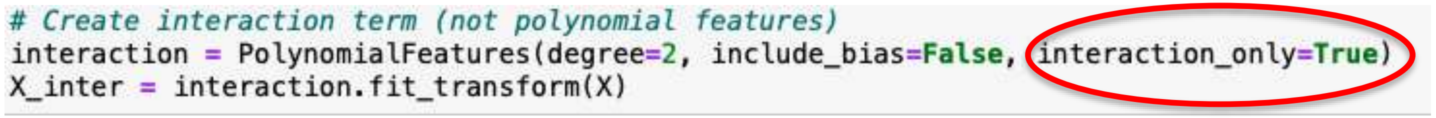
$$y = w_0 + w_1 x_1 + w_2 x_2$$

With interaction term:

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$x_3 = x_1 x_2$$

- We can add features to account for potential interaction terms
- Scikit Learn's PolynomialFeatures() creates interaction terms automatically using the degree provided
- We can also use ONLY the interaction terms if we wish and exclude the polynomial terms



```
# Create interaction term (not polynomial features)
interaction = PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)
X_inter = interaction.fit_transform(X)
```

IN-CLASS EXERCISE

Regularization

Motivation for Regularization

- The training method (minimizing sum of squared error) we have been using tends to reward complexity / overfitting
  - More complex models fit the training data better and reduce training error
- However, we know that complex models have higher variance and thus may not predict as well on new data
- How can we build a regression model in a more balanced way?
  - We add a penalty factor to our cost function to penalize complexity
- The penalty term we add to the cost function penalizes complexity in terms of number of features and their magnitude of contribution

Regularization

Normal linear regression cost function:

$$J(w) = SSE = \sum_{i=1}^{N} \left(y_i - \left(w_0 + w_1 x_{i,1} + \cdots + w_p x_{i,p}\right)\right)^2$$

- We add a penalty term to the cost function that is a function of the sum of the coefficients
- Now, higher number or values of coefficients increases the cost function
- Minimizing this new cost function seeks optimal balance of the model fit with the number & magnitude of coefficients

Cost function with regularization:

$$J(w) = \sum_{i=1}^{N} \left(y_i - \left(w_0 + w_1 x_{i,1} + \cdots + w_p x_{i,p}\right)\right)^2 + \lambda Penalty(w_1 \dots w_p)$$

λ controls strength of the penalty

outrageously
AMBITIOUS

Regularization: LASSO & Ridge Regression

Duke
PRATT SCHOOL of ENGINEERING

LASSO (L1) Regression

- LASSO: Least Absolute Shrinkage & Selection Operator
- Penalty factor is λ * sum of absolute value of coefficients (L1 norm)
- Form of the penalty factor forces coefficients to 0 if not relevant
- Can also be considered a feature selection method because it reduces # of features

Linear regression cost function:

$$J(w) = \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_{i,1} + \cdots + w_p x_{i,p}))^2$$

LASSO regression cost function:

$$J(w) = \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_{i,1} + \cdots + w_p x_{i,p}))^2 + \lambda \sum_{j=1}^{p} |w_j|$$

Duke
PRATT SCHOOL of ENGINEERING

Ridge (L2) Regression

- Penalty factor is the sum of squared coefficients (squared L2 norm)
- Forces the coefficients of irrelevant variables to be small but not to 0 -> does not perform feature selection
- Mathematically more convenient

Linear regression cost function:

$$J(w) = \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_{i,1} + \cdots + w_p x_{i,p}))^2$$

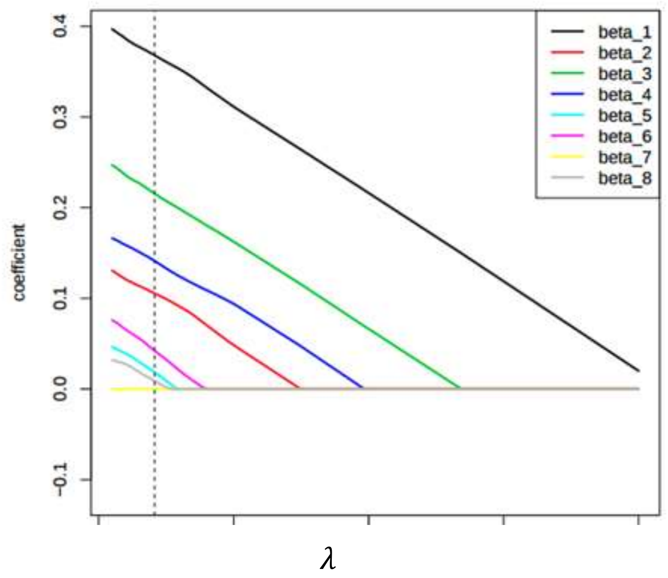
Ridge regression cost function:

$$J(w) = \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_{i,1} + \cdots + w_p x_{i,p}))^2 + \lambda \sum_{j=1}^{p} {w_j}^2$$

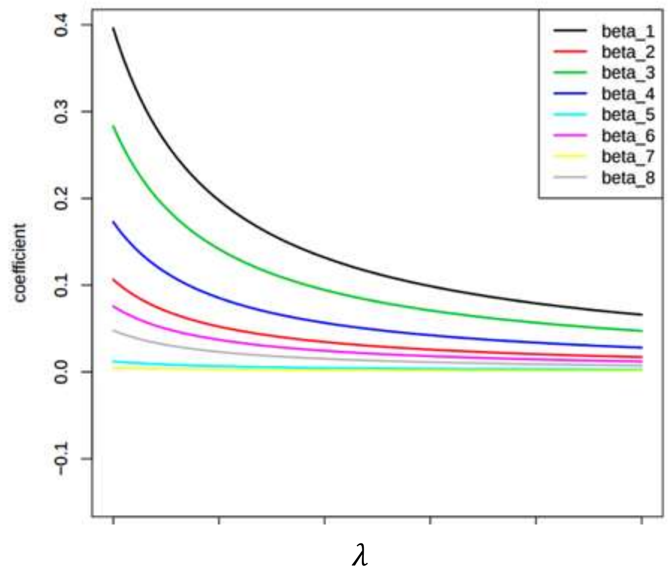
Duke
PRATT SCHOOL of ENGINEERING

LASSO vs. Ridge Regression

LASSO Regression



Ridge Regression



Example: LASSO vs Ridge

- Let's look at a possible set of coefficients for a linear regression model with four features:
  - Coefficient set A: w = [2, 2, 2, 2]

	Ridge penalty $\lambda \sum_{j=1}^{p} {w_j}^2$	LASSO penalty $\lambda \sum_{j=1}^{p} \lvert w_j \rvert$
Set A	λ*16	λ*8

- Ridge penalizes this model more aggressively than LASSO does, but both Ridge and LASSO would seek another coefficient set that yields a similar SSE at a lower penalty

Duke PRATT SCHOOL of ENGINEERING

Example: LASSO vs Ridge

- Now let's reduce the coefficients towards 0 and compare two options:
  - Coefficient set B: w = [1, 0, 0, 0]
  - Coefficient set C: w = $[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]$

	Ridge penalty $\lambda \sum_{j=1}^{p} {w_j}^2$	LASSO penalty $\lambda \sum_{j=1}^{p} \lvert w_j \rvert$
Set B	λ*1	λ*1
Set C	λ*1	λ*2

- The L2 norm is equal and so Ridge penalizes them equally (and picks whichever set minimizes SSE)
- LASSO penalizes Set C much more than the sparse Set B, driving towards a sparse solution

Duke
PRATT SCHOOL of ENGINEERING

A geometric perspective

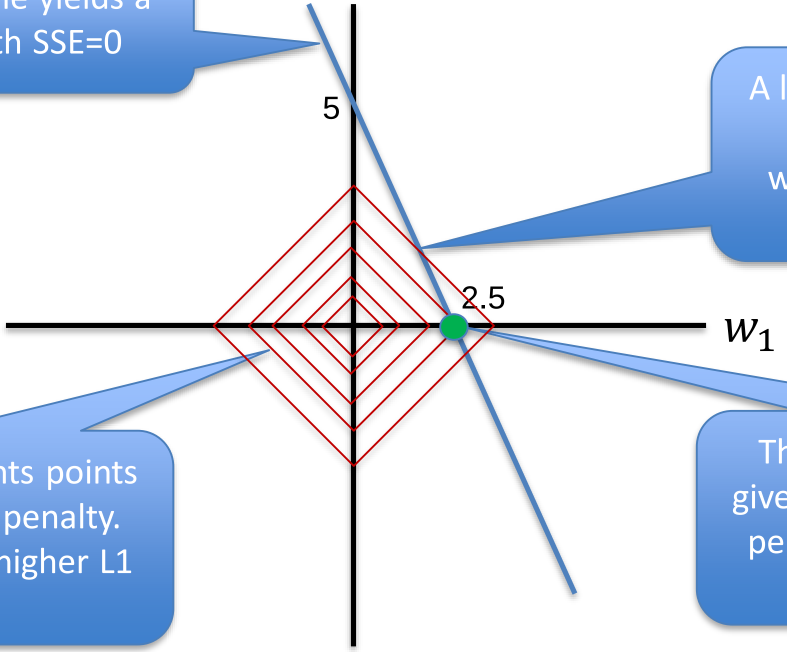
- Suppose we have a linear regression model $y = w_1x_1 + w_2x_2$
- Now, suppose we have a single point: x=[2,1] y=[5]
- To calculate our coefficients w_1, w_2 we plug in the point values:

$$5 = w_1 * 2 + w_2 * 1$$

- There are an infinite number of possible solutions to this, and they all fall on the line $w_2 = 5 - 2w_1$

A geometric perspective - LASSO

Cost = SSE + L1 penalty



Any point on this line yields a perfect model with SSE=0

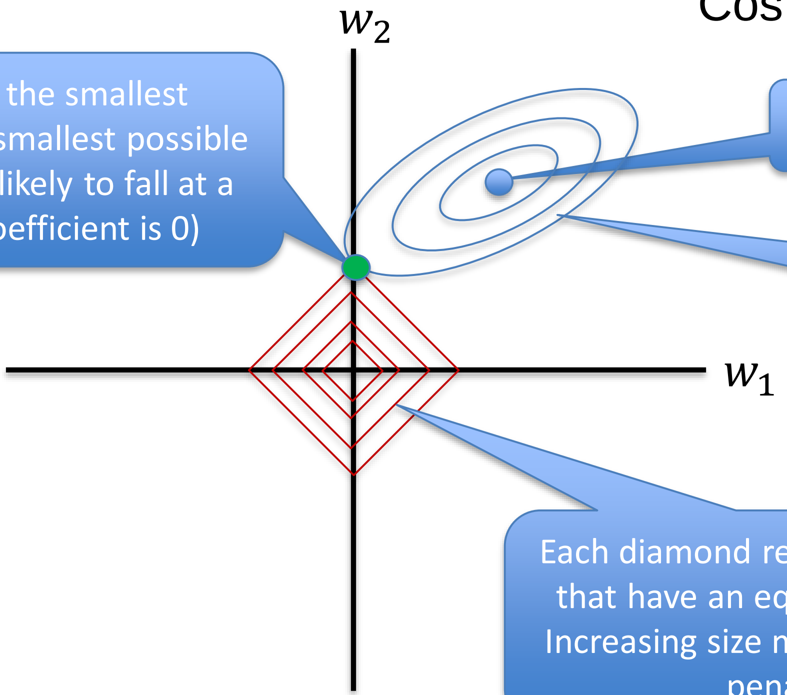
A larger diamond would intersect at a non-sparse point, but then it would not be minimizing the cost function (SSE + L1 penalty)

Each diamond represents points that have an equal L1 penalty. Increasing size means higher L1 penalty

The intersection of a diamond of given size (same L1 penalty) and the perfect model line falls at a sparse point (one coefficient is 0)

Now let's add more training data

Cost = SSE + L1 penalty



The intersection of the smallest possible diamond and smallest possible constant SSE curve is likely to fall at a sparse point (one coefficient is 0)

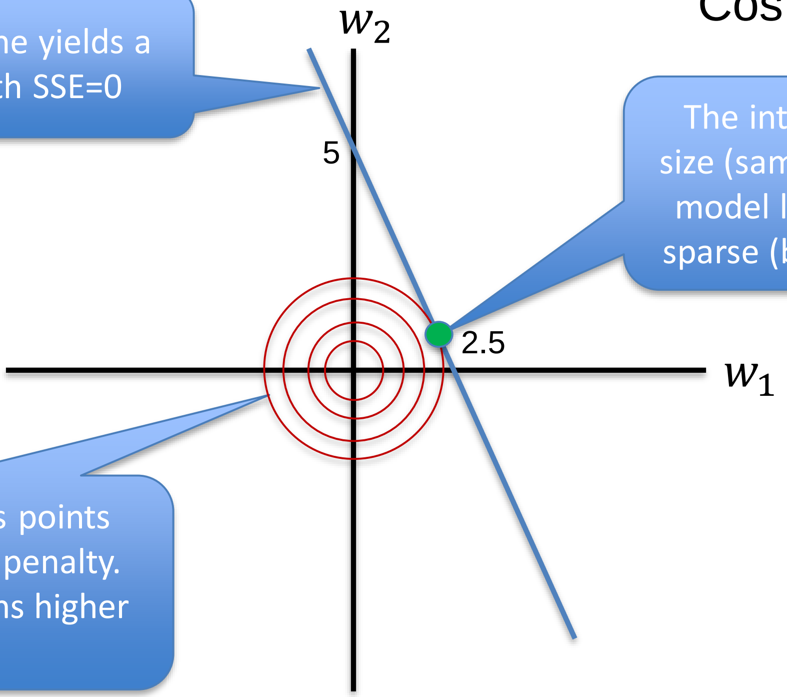
We now have a single set of values for w1,w2 with SSE=0

We can draw curves of constant SSE around the point

Each diamond represents points that have an equal L1 penalty. Increasing size means higher L1 penalty

A geometric perspective - Ridge

Cost = SSE + L2 penalty



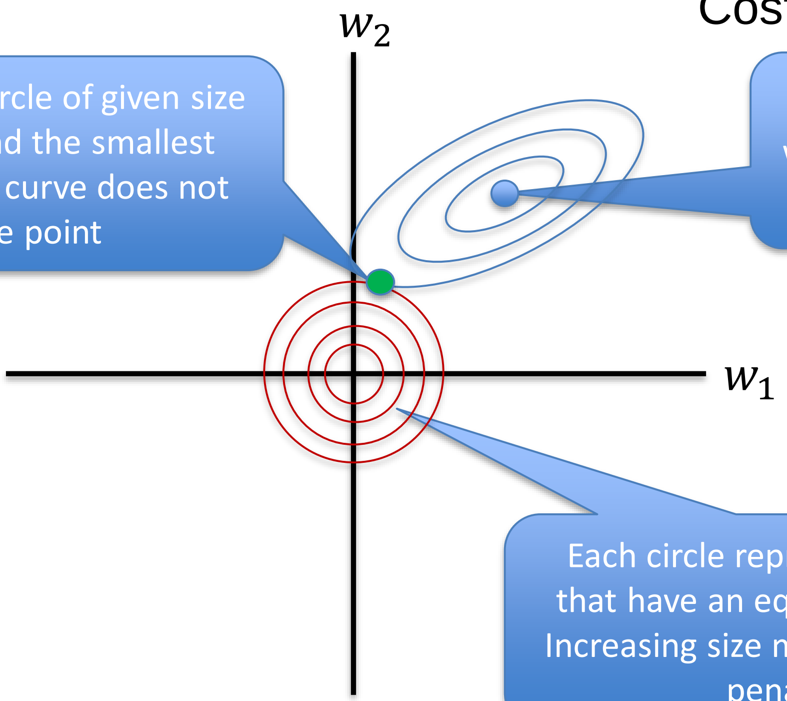
Any point on this line yields a perfect model with SSE=0

The intersection of a circle of given size (same L2 penalty) and the perfect model line falls at a point that is not sparse (but one coefficient close to 0)

Each circle represents points that have an equal L2 penalty. Increasing radius means higher L2 penalty

Now let's add more training data

Cost = SSE + L1 penalty



The intersection of a circle of given size (same L2 penalty) and the smallest possible constant SSE curve does not fall at a sparse point

We now have a single set of values for w1,w2 with SSE=0. We can draw curves of constant SSE around the point

Each circle represents points that have an equal L2 penalty. Increasing size means higher L2 penalty

Summary: LASSO vs. Ridge

LASSO	Ridge
• Minimizes SSE + sum of absolute value of coefficients	• Minimizes SSE + sum of coefficients squared
• Performs feature selection by forcing coefficients of irrelevant features to zero (rewards sparsity)	• Retains all features but shrinks coefficients toward each other – can underestimate coefficients of most significant features
• Produces simpler models that are highly interpretable	• Works better for more complex patterns in data influenced by many features
• In cases of collinearity, randomly picks one feature to keep out of the correlated features	• Performs better under collinearity – does not discard variables

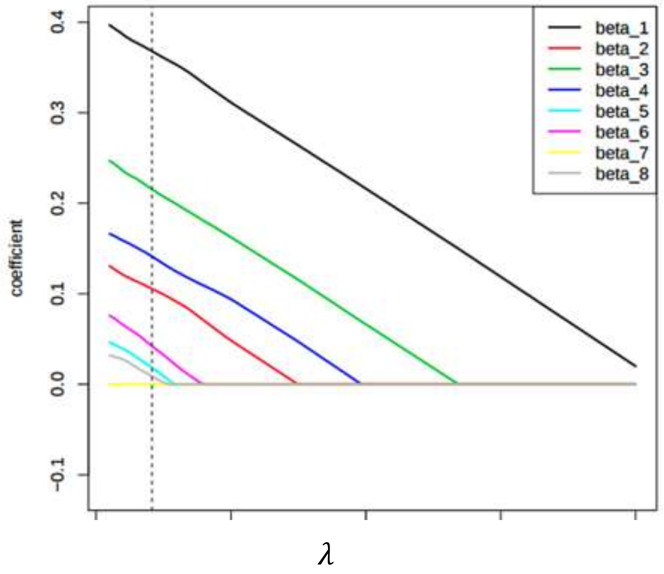
Duke

PRATT SCHOOL *of* ENGINEERING

Determining the optimal λ

- Lambda is a **hyperparameter** that should be optimized
- Low values of lambda result in more complex models (more variables with higher coefficients) and high lambdas result in simpler models
- The optimal lambda that minimizes error (MSE/MAE) can be found using a validation set or cross-validation

LASSO regression



Conclusion: LASSO & Ridge

- Applying regularization will often give us a better model when we are dealing with complex data (many features)
- You might have a reason to prefer one method or the other:
  - Desire a simpler, more interpretable model -> LASSO
  - Complex relationship of target to many features with collinearity -> Ridge
- If not sure, default to Ridge regression or evaluate both
- Whichever one you choose, be sure to:
  1. **Scale the data before you begin**
  2. **Determine the optimal lambda using a validation set / cross-validation**

Wrapping up Linear Regression

- For regression problems, linear regression is a great place to start
  - Establish a baseline for model performance
  - Better understand relationships between features and target
- Be sure to visualize your data and residuals to identify problems / opportunities to improve fit
- Use polynomial regression and/or regularization to improve fit
  - Use cross-validation to adjust hyperparameters (polynomial degree or lambda)
- Then, try more advanced algorithms and compare results

Duke
PRATT SCHOOL of ENGINEERING

DEMO

IN-CLASS EXERCISE