ECE 20875

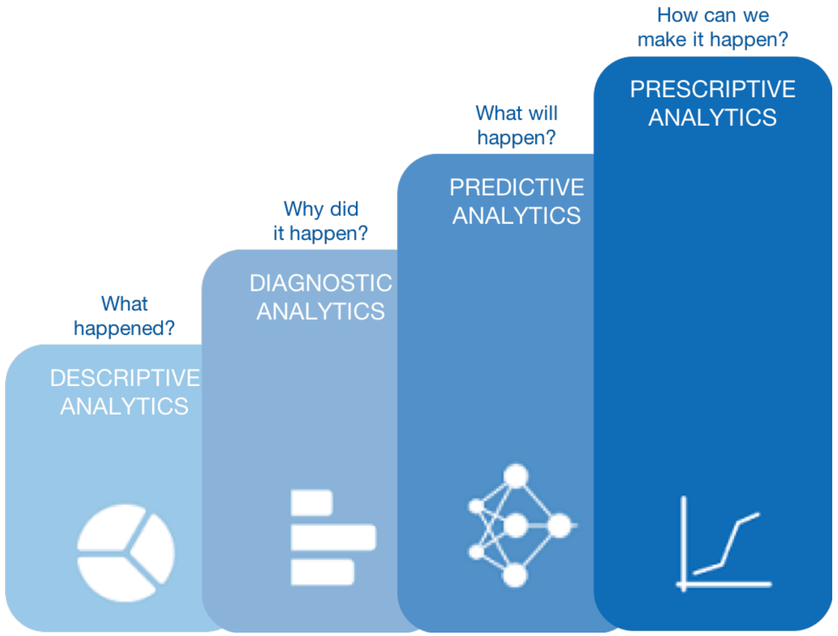
Python for Data Science

Milind Kulkarni and Chris Brinton

regression

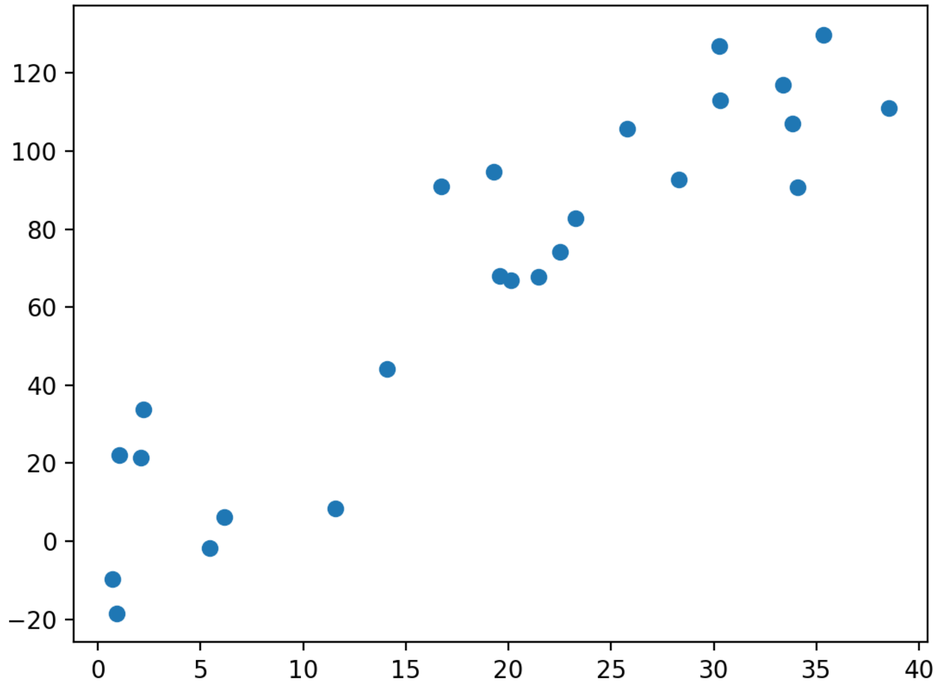
inference

- **Inference** is one of the basic problems that we want to solve in data science
  - Given a set of data that we know some facts about, what new conclusions can we draw, and with what certainty?
  - We will investigate several approaches to drawing conclusions from given sets of data
- This lecture: Making **predictions** about new data points given existing data using **linear regression**



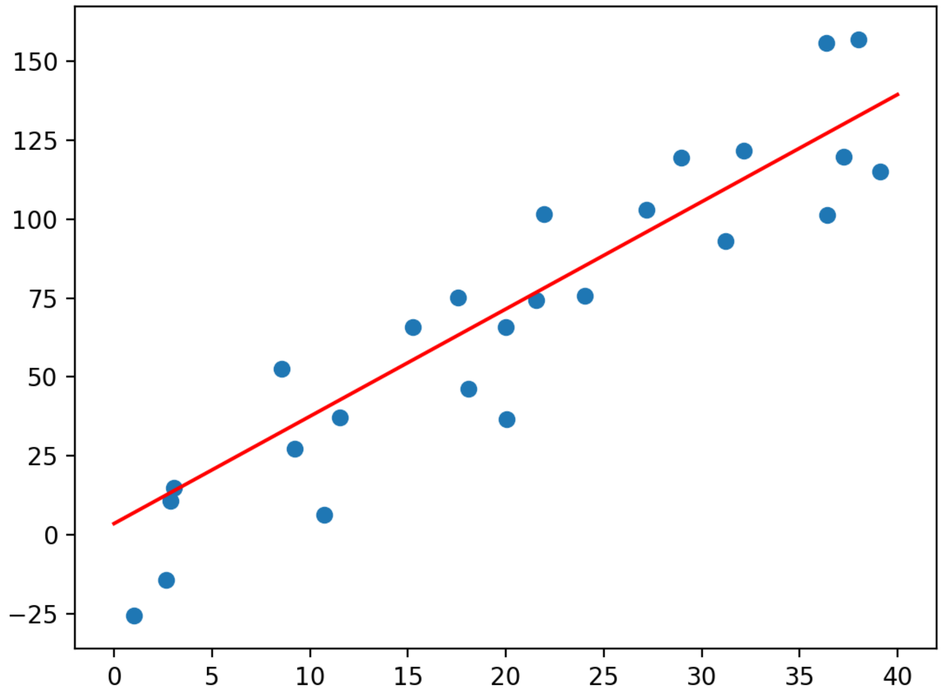
linear regression

- Basic modeling problem: I want to identify a relationship between …
  - **exploratory variables** (i.e., “input” features of a data point), and
  - a **target variable** (i.e., some “output” quantity)
- Can we learn what this relationship is?
- If we have a **model** for this relationship, we can use it to predict the target variable for new data points



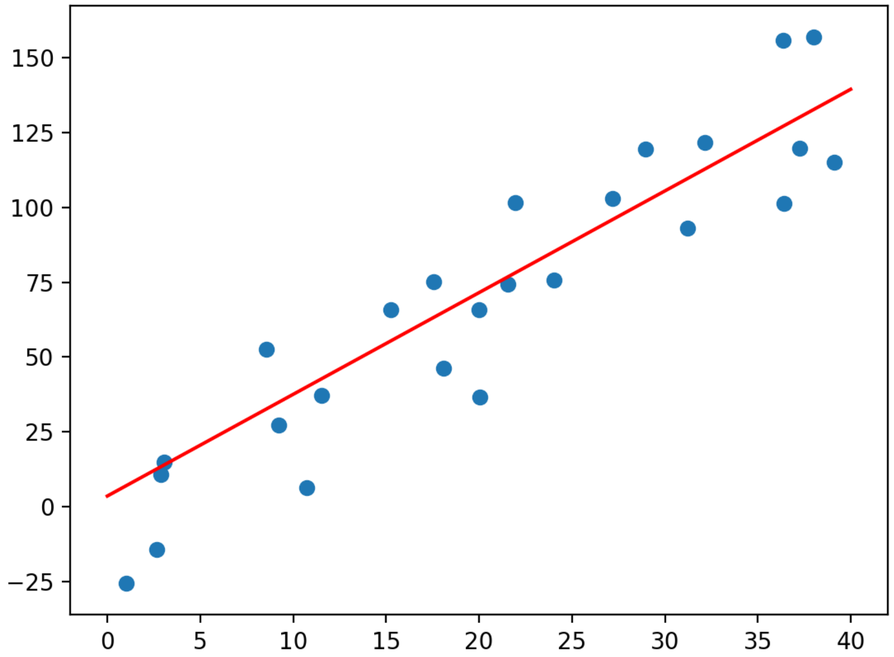
linear regression

- Basic modeling problem: I want to identify a relationship between …
  - **exploratory variables** (i.e., “input” features of a data point), and
  - a **target variable** (i.e., some “output” quantity)
- Can we learn what this relationship is?
- If we have a **model** for this relationship, we can use it to predict the target variable for new data points



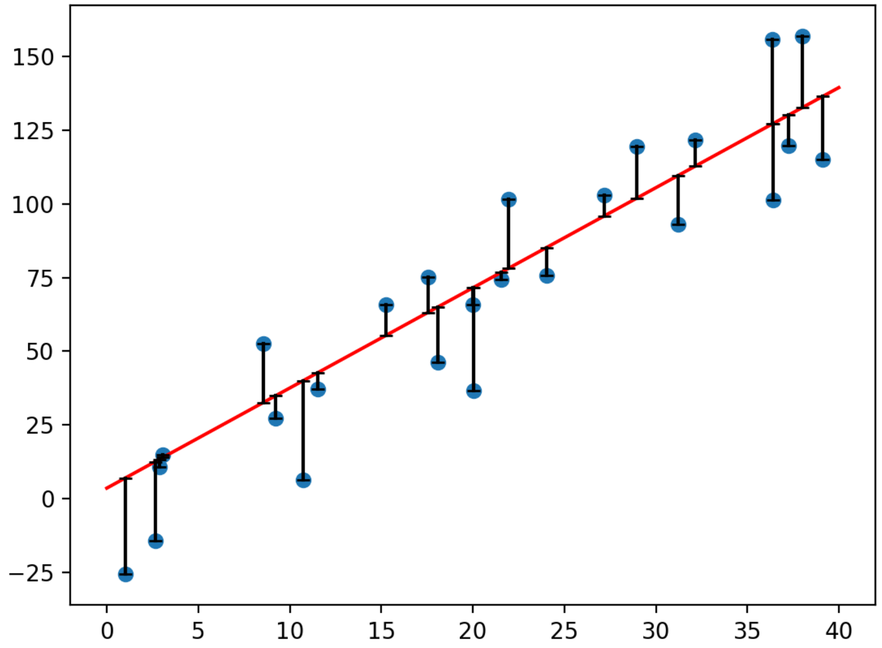
linear regression

- Can we learn the model from the data?
- Note that the model does not match the data exactly!
  - A model is (at best) a simplification of the real-world relationship
- What makes a good model?
  - Minimizes **error**: How far the model deviates from the observed data
  - Maximizes **generalizability**: How well the model is expected to hold up to unseen data



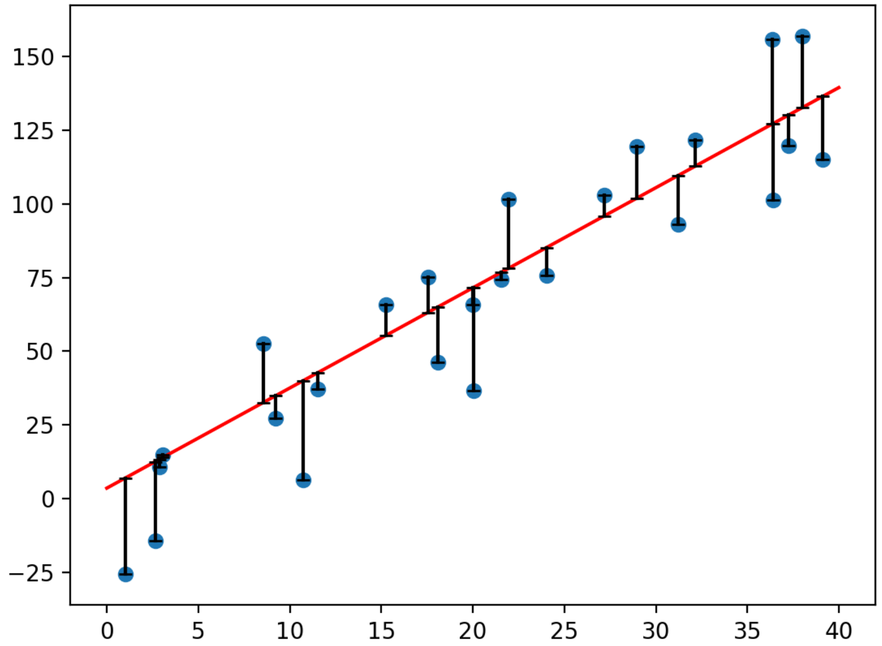
linear regression

- Can we learn the model from the data?
- Note that the model does not match the data exactly!
  - A model is (at best) a simplification of the real-world relationship
- What makes a good model?
  - Minimizes **error**: How far the model deviates from the observed data
  - Maximizes **generalizability**: How well the model is expected to hold up to unseen data



picking a model

- The **single-variable linear regression** model:
  $y_n = ax_n + b + \epsilon_n, \;\; n = 1,...,N$
- y_n is the **measured value** of the target variable for the nth data point
- $ax_n + b$ is the **estimated value** of the target, based on the explanatory x_n
- Each point n is associated with some (deterministic) model $ax_n + b$ plus some (random) error term ϵ_n
- How do we minimize this error?

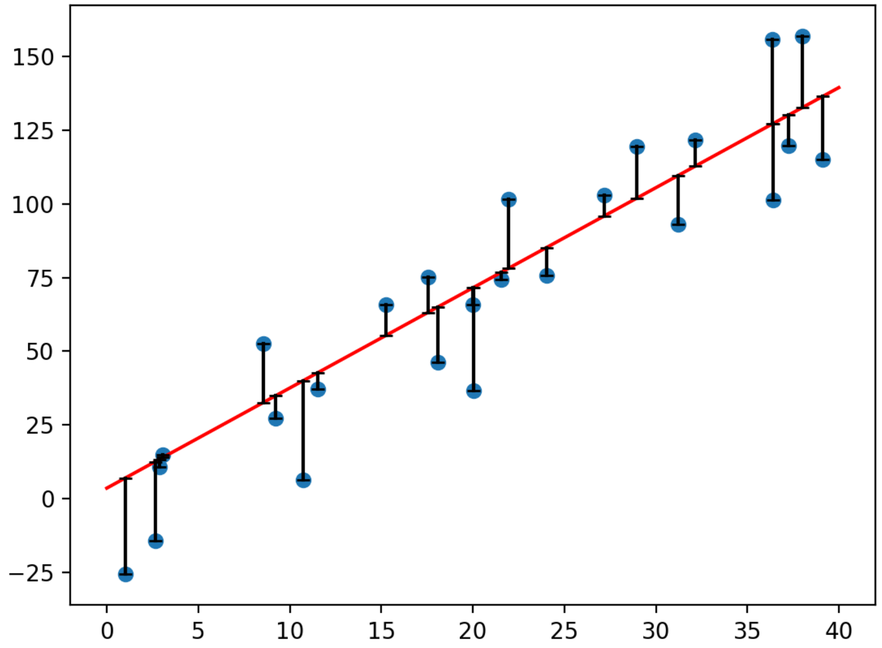


minimizing error

- Minimize **mean squared error** (MSE)

$$E = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - (a x_n + b) \right)^2$$

- Common error metric: Makes it easy to take derivatives to minimize!
  - We looked at MSE before, during histogram cross validation and estimators
  - With two **model parameters** a and b, this is reasonably easy to carry out by hand



minimizing error

- Set the derivatives with respect to a and b to zero:

$$\frac{dE}{da} = \frac{1}{N}\sum_{n=1}^{N} -2x_n\left(y_n - (ax_n + b)\right) = 0$$

$$\frac{dE}{db} = \frac{1}{N}\sum_{n=1}^{N} -2\left(y_n - (ax_n + b)\right) = 0$$

- Focusing first on the second equation, we have

$$\frac{-\sum_{n=1}^{N} y_n}{N} + a\frac{\sum_{n=1}^{N} x_n}{N} + b\frac{\sum_{n=1}^{N} 1}{N} = 0, \text{ or}$$

$$b = \frac{\sum_{n=1}^{N} y_n}{N} - a\frac{\sum_{n=1}^{N} x_n}{N} = \bar{y} - a\bar{x}$$

- As for the first equation,

$$\frac{-\sum_{n=1}^{N} x_n y_n}{N} + a\frac{\sum_{n=1}^{N} x_n^2}{N} + b\frac{\sum_{n=1}^{N} x_n}{N} = 0, \text{ so}$$

$$a\frac{\sum_{n=1}^{N} x_n^2}{N} = \frac{\sum_{n=1}^{N} x_n y_n}{N} - b\frac{\sum_{n=1}^{N} x_n}{N} = \frac{\sum_{n=1}^{N} x_n y_n}{N} - b\bar{x}$$

- Substituting our expression for b, we have:

$$a\frac{\sum_{n=1}^{N} x_n^2}{N} = \frac{\sum_{n=1}^{N} x_n y_n}{N} - (\bar{y} - a\bar{x})\bar{x}, \text{ or}$$

$$a\left(\frac{\sum_{n=1}^{N} x_n^2}{N} - \bar{x}^2\right) = \frac{\sum_{n=1}^{N} x_n y_n}{N} - \bar{y}\bar{x}$$

minimizing error

- Isolating a on the left hand side and simplifying, we get:

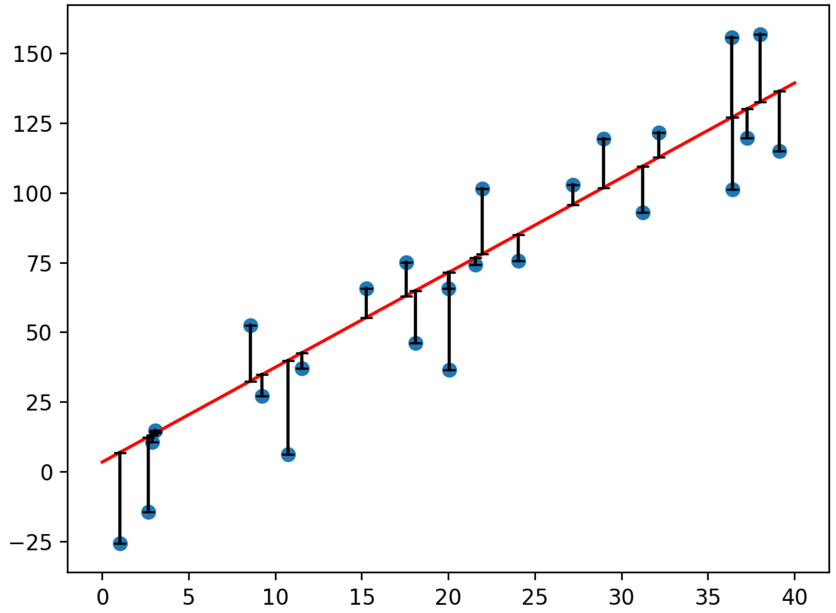
$$a = \frac{\sum_{n=1}^{N} x_n y_n - N\bar{y}\bar{x}}{\sum_{n=1}^{N} x_n^2 - N\bar{x}^2}$$

- Which we can then use to solve for b according to:

$$b = \bar{y} - a\bar{x}$$

- And then our linear regression predictor for a new datapoint i is

$$y_i = ax_i + b$$



matrix algebra review

- Let's say $\mathbf{x} = (x_1 \; x_2 \; \cdots \; x_n)^T$ and $\mathbf{y} = (y_1 \; y_2 \; \cdots \; y_n)^T$ are both n-dimensional vectors. Then

  $\mathbf{x}^T\mathbf{y} = x_1y_1 + x_2y_2 + \cdots + x_ny_n$ is the **inner product** or **dot product** of $\mathbf{x}$ and $\mathbf{y}$, which is the multiplication of a $1 \times n$ and $n \times 1$ vector and results in a scalar.

- More generally, define

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix}$$

  as two $m \times n$ matrices. Then

$$\mathbf{X}^T\mathbf{Y} = [\mathbf{x}_1 \; \mathbf{x_2} \; \cdots \; \mathbf{x}_n]^T [\mathbf{y}_1 \; \mathbf{y_2} \; \cdots \; \mathbf{y}_n] = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x_2}^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} [\mathbf{y}_1 \; \mathbf{y_2} \; \cdots \; \mathbf{y}_n] = \begin{bmatrix} \mathbf{x}_1^T\mathbf{y}_1 & \mathbf{x}_1^T\mathbf{y}_2 & \cdots & \mathbf{x}_1^T\mathbf{y}_n \\ \mathbf{x}_2^T\mathbf{y}_1 & \mathbf{x}_2^T\mathbf{y}_2 & \cdots & \mathbf{x}_2^T\mathbf{y}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n^T\mathbf{y}_1 & \mathbf{x}_n^T\mathbf{y}_2 & \cdots & \mathbf{x}_n^T\mathbf{y}_n \end{bmatrix}$$

  is the matrix multiplication of $\mathbf{X}$ and $\mathbf{Y}$, which results in an $n \times n$ matrix.

matrix algebra review

- If $\mathbf{X}$ is a **square** matrix (i.e., has dimension $n \times n$), then its inverse is $\mathbf{X}^{-1}$ if

$$\mathbf{X}^{-1}\mathbf{X} = \mathbf{X}\mathbf{X}^{-1} = \mathbf{I}, \text{ where } \mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

  is the $n \times n$ **identity matrix**

- In Python, we can use the numpy library to do matrix operations

  numpy.array(A) //Convert list to numpy array

  numpy.dot(A,B) //Matrix multiplication

  numpy.linalg.inv(A) //Matrix inverse

  A.sum(axis=0) //Sum over rows of matrix

- See https://scipy-lectures.org/intro/numpy/operations.html for more examples

least squares equations

- We typically have more than one explanatory variable
- To generalize to this case, it is more convenient to work with matrix equations
- For the single-variable case, if we define

$$\mathbf{X} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \quad \beta = \begin{bmatrix} a \\ b \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

then we can write

$$\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{y}$$

- The **multivariable linear regression model** with M explanatory variables is

$$y_n = a_1 x_{n,1} + a_2 x_{n,2} + \cdots + a_M x_{n,M} + b + \epsilon_n, \quad n = 1,...,N$$

- In this case, we define

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,M} & 1 \\ x_{2,1} & x_{2,2} & \cdots & x_{2,M} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,M} & 1 \end{bmatrix} \quad \beta = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_M \\ b \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

where $\mathbf{X}$ is the **feature matrix.** Then, as before, we can write

$$\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{y}$$

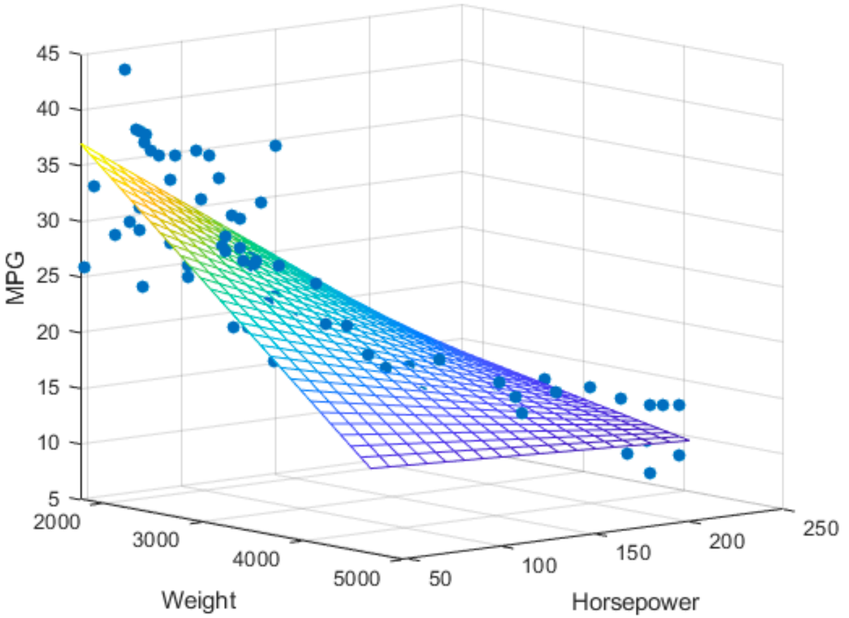
- These are called the **least squares equations**

solving for β

- If $\mathbf{X}^T\mathbf{X}$ is invertible, we can take a matrix inverse to solve for the model parameters β:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- But $\mathbf{X}^T\mathbf{X}$ is not always invertible
  - The inverse exists if and only if the columns of $\mathbf{X}$ are **linearly independent** of one another
  - This means that we cannot have the case where one column can be written as a linear combination of the others
- What does it mean when $\mathbf{X}^T\mathbf{X}$ is not invertible?
  - Infinitely many possible solutions
  - We typically choose the one where $||\beta||$ is smallest. Why?



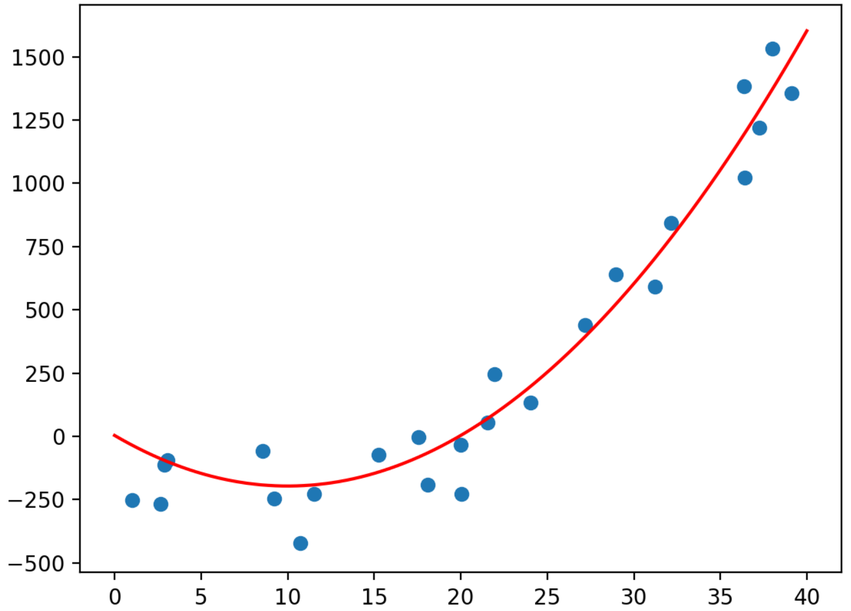
what about non-linear?

- A common misconception is that linear regression can only find linear relationships
  - The “linear” part refers to the parameter vector β, not the input features $\mathbf{X}$
- We can readily take nonlinear functions of our features
- For example, suppose we want to fit a quadratic model:

$$y_n = a_1(x_n)^2 + a_2 x_n + b$$

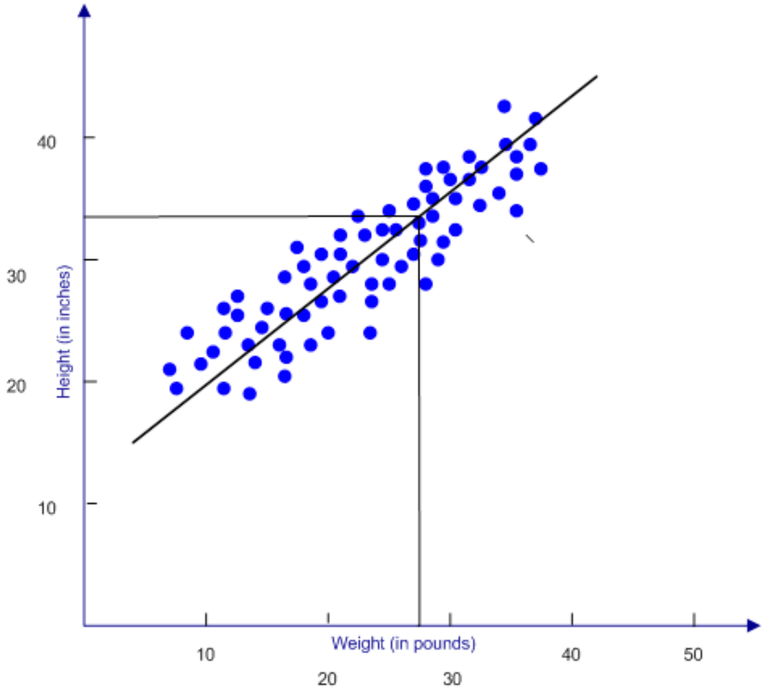
- We just create a “synthesized” feature that is quadratic:

$$\mathbf{X} = \begin{bmatrix} (x_1)^2 & x_1 & 1 \\ (x_2)^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ (x_N)^2 & x_N & 1 \end{bmatrix} \quad \beta = \begin{bmatrix} a_1 \\ a_2 \\ b \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$



using your model

- If you get a new data point described by its features, you can apply the linear regression to estimate the target variable
- New data point: $(x_1, x_2, \ldots)$
- Prediction: $\hat{y} = a_1 x_1 + a_2 x_2 + \cdots + b$
- How good is the prediction? Compare $\hat{y}$ to y (once it is known) in terms of MSE
- Make sure to take into account any **normalization** when using the model



linear regression in python

- You can solve the least squares equations directly using numpy (which you have to do in problem 1 of the homework)
- Given how common linear regression is, several variants are built in to the sklearn (scikit learn) library directly

```
from sklearn import linear_model

regr = linear_model.LinearRegression(fit_intercept=True)  //Define
linear regression object

regr.fit(X_train,y_train)  //Fit model to training set

regr.coef_  //View coefficients (a_1,…,a_n) of trained model

regr.intercept_  //View intercept (b) of trained model

y_pred = regr.predict(X_test)  //Apply model to test set
```

normalization

- Suppose I fit a linear regression model and get $\hat{y} = 10x_1 + 100x_2 + 5$
  - Does this mean that x_2 has a bigger impact on y than x_1?
  - Not necessarily, because we have said nothing about the ranges of x_1 and x_2
- One option to make interpretation more useful is to **normalize** the data before doing linear regression. For each feature:

1. Center the value: Subtract average feature value from each data point's feature
   - Also center y values
   - Useful to eliminate any spurious linear relationship between features (remember that linear relationship between features can break matrix inversion)

normalization

2. Divide by standard deviation: Divide each data point's feature by standard deviation

    - Re-scales features so that each feature is expressed new units: standard deviation from the mean

    - Makes sure that all features are in the same "range" for interpretation

- Mathematically, we are defining the following operation for each feature:

$$\mathbf{x}_i \leftarrow \frac{\mathbf{x}_i - \bar{x}_i}{s_{x_i}}$$

- Remember to apply same normalization when *using* model (center/normalize features, "un-center" the y you get back)

coefficient of determination

- How good is the fit of the regression to the dataset?
  - Points close to inferred model → good fit
  - Points farther away from inferred model → bad fit
- One simple formula that captures this: coefficient of determination r^2

$$r^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2} = 1 - \frac{MSE}{\sigma^2}$$

- Fraction of the variance in the sample explained by the model
- In sklearn:

```
from sklearn.metrics import r2_score

r2_score(y_true,y_pred)
```

y_i : **measured value**

$\hat{y}_i$: **predicted value**

interpreting results

- How should we interpret the results of linear regression?
- Recall multi-feature model: $y_n = a_1 x_{1,n} + a_2 x_{2_n} + b$
- If one feature weight (e.g., a_1) is higher than another (e.g., a_2), this can indicate that that feature is more important than the others (contributes more to the value of y)
- Need to be careful, though! If different features have different scales, then weights will naturally be different!
  - Normalization is useful as it standardizes the feature ranges

more interpretation

- Is a feature significant?
- Just because a feature is used in a model doesn't mean it is important in predicting the value of the output
- But the model will try to account for the feature anyway!
- Can perform a hypothesis test (see previous lectures):
  - Null hypothesis H_0: Coefficient is 0 (feature has no predictivity)
  - Alternative hypothesis H_1: Coefficient is not 0 (feature has predictivity)

hypothesis test for regression

- Test statistic is always: (value - hypothesized value) / standard error
- What is the standard error for a regression coefficient x?

$$SE_x = \frac{\sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{N-2}}}{\sqrt{\sum (x_i - \bar{x})^2}}$$

y_i : **measured value**

$\hat{y}_i$: **predicted value**

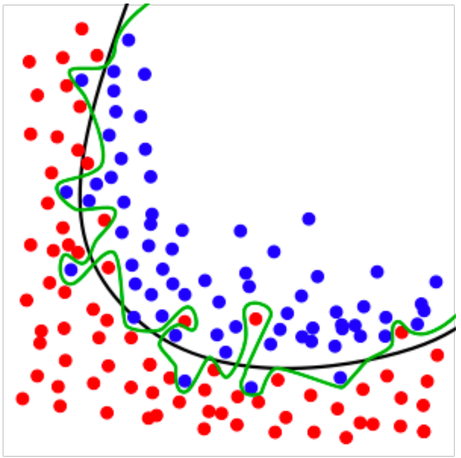
x_i : **feature value**

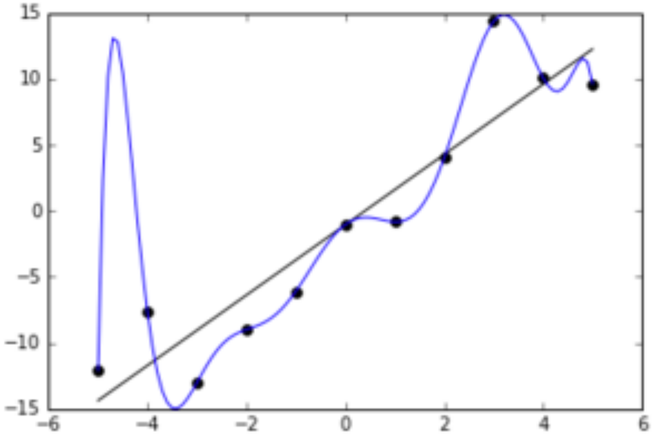
$\bar{x}$: **feature average**

- For a z-test, find p-value of SE_x against the z-distribution
- For a t-test, find p-value against a t-distribution with $N - k - 1$ degrees of freedom, where k is the number of features

overfitting

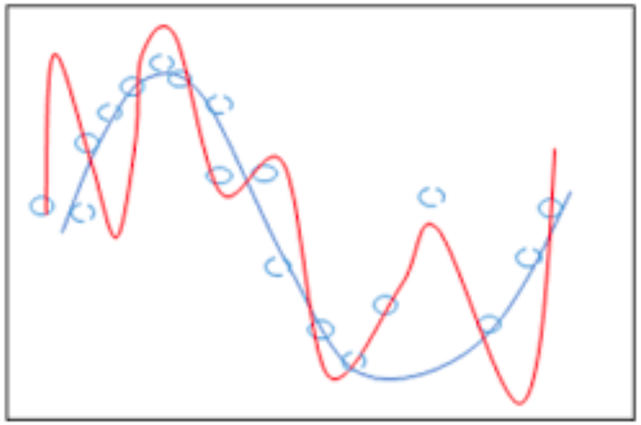
- If our goal was just to minimize error on the existing dataset, we'd keep adding features
  - E.g., adding more degrees to a polynomial
- But this sacrifices the generalizability of the model
- An **overfitted** model is one which contains too many parameters than can be justified by the data
  - High r^2 on training data, good MSE on training data, but bad MSE on testing data
- We can contrast this with **underfitting**, where we don't have enough parameters to drive down MSE on either training or testing data





regularization

- When we have a lot of features, we can use **regularization**, a class of techniques for mitigating overfitting by penalizing non-zero model coefficients



- The general expression we work with in regularization is:

  minimize (model error) + λ(coefficient weights)

- $\lambda \geq 0$ is the **regularization parameter**
  - Higher λ: Minimizing model parameters becomes more important
  - Lower λ: Minimizing model error becomes more important
- Several different regularization techniques: Lasso, **Ridge**, Elastic-Net, …

ridge regression

- In **ridge regression**, the regularization term is the sum of squares of the coefficients:

$$\underset{\beta}{\text{minimize}} \; ||\mathbf{X}\beta - y||_2^2 + \lambda ||\beta||_2^2$$

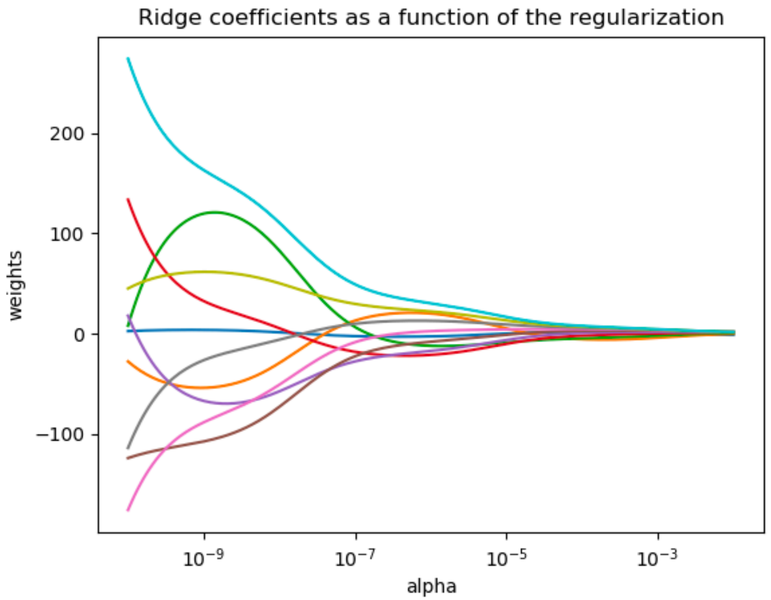
- This makes it easy to solve in matrix form as we can combine the squares:

$$\underset{\beta}{\text{minimize}} \; ||(\mathbf{X} + \mathbf{I}\lambda)\beta - y||_2^2$$

- In Python:

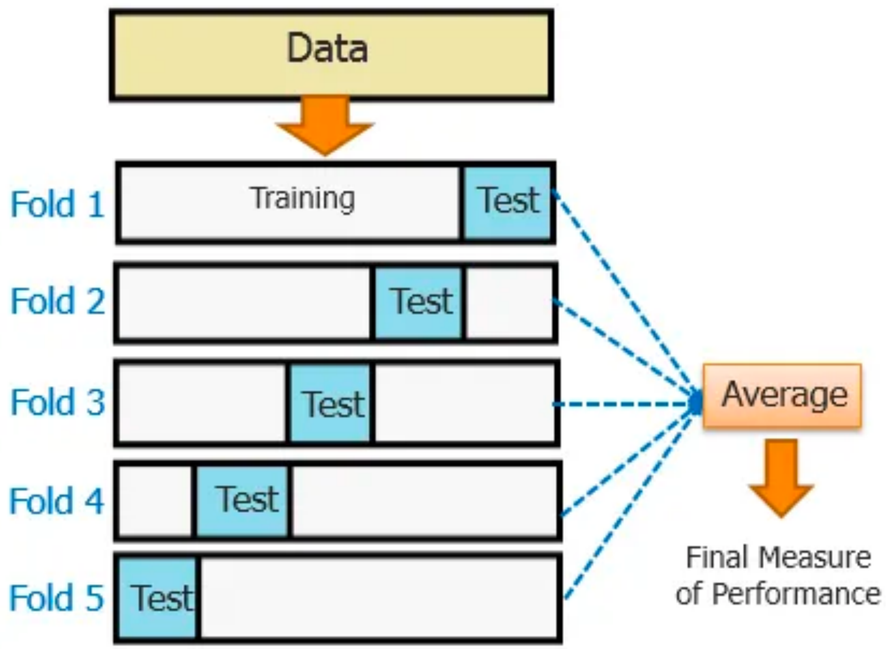
```
from sklearn import linear_model

reg = linear_model.Ridge(alpha=0.1, fit_intercept=True)
```

regularization parameter



cross validation

- How do we determine the right value of λ?
- Need to resort back to **cross validation**
  - Split the data into a training and testing set
  - Train the model on different values of λ
  - Check the MSE on the test set



- Need to test a wide range of λ typically, e.g., 0.01,…,0.1,…,1,…,10,…,100