ECE 20875
Python for Data Science
David Inouye and Qiang Qiu
(Adapted from material developed by Profs. Milind Kulkarni, Stanley Chan, Chris Brinton, David Inouye)

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.

- Over the next few lectures: Making **predictions** about new data points given existing data using **linear regression**.
linear regression

• Basic modeling problem: I want to identify a relationship between …

• explanatory variables (i.e., the “inputs”, often referred to as the features of a data point), and

• a target variable (i.e., some “output” quantity that we want to estimate)

• 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 …

• explanatory variables (i.e., the “inputs”, often referred to as the features of a data point), and

• a target variable (i.e., some “output” quantity that we want to estimate)

• 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 **observed 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 observed error: How far the model deviates from the observed data
  • Maximizes generalizability: How well the model is expected to hold up to unseen data
Simple linear regression model

- The simple linear regression model has a single explanatory variable:
  \[ y_n = ax_n + b + \epsilon_n, \quad n = 1, \ldots, N \]

- \( y_n \) is the measured value of the target variable for the \( n \)th data point

- \( ax_n + b \) is the estimated value of the target, based on the explanatory \( x_n \)

- Each \( y_n \) is associated with a model prediction component \( ax_n + b \) plus some error term \( \epsilon_n \)

- How do we minimize this error?
minimizing error

- The **mean squared error** (MSE) for simple linear regression is

\[ E(a, b) = \frac{1}{N} \sum_{n=1}^{N} (y_n - (ax_n + b))^2 \]

- Common error metric: We looked at already when we studied the choice of histogram bin widths

- We want to minimize \( E \), denoted: \( \min_{a,b} E(a, b) \)

- With two **model parameters** \( a \) and \( b \), this is reasonably easy to carry out by hand

- The square makes it easy to take the derivative
minimizing error: derivation

• Set the derivatives with respect to $a$ and $b$ to zero:

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

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

• Set the derivatives with respect to $a$ and $b$ to zero:

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

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

• Focusing first on the second equation, we have:

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

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

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

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

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

- Focusing first on the second equation, we have:

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

so

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

• 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:

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

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

• As for the first equation,

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

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

• Substituting our expression for \( b \), we have:

\[
a \left( \frac{\sum_{n=1}^{N} x_n^2}{N} - \bar{x}^2 \right) = \frac{\sum_{n=1}^{N} x_n y_n}{N} - \bar{y}\bar{x}
\]
minimizing error: formulas

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

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

• Here, $\bar{x}$ and $\bar{y}$ are the averages of the $x_n$ and $y_n$, respectively

• We can then use $a$ to solve for $b$ according to:

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

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

$$y_i = ax_i + b$$
minimizing error: formulas

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

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

• Here, $\bar{x}$ and $\bar{y}$ are the averages of the $x_n$ and $y_n$, respectively

• We can then use $a$ to solve for $b$ according to

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

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

$$y_i = ax_i + b$$

• What do we do if there is more than one explanatory variable?

• To generalize to this case, it is more convenient to work with matrix equations
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

$\mathbf{x}^T \mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$

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.

• For example, suppose $\mathbf{x} = (3 \ 4 \ 5)^T$, $\mathbf{y} = (1 \ 0 \ 2)^T$. Then:

$\mathbf{x}^T \mathbf{y} = (3 \ 4 \ 5) \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} = 3 \times 1 + 4 \times 0 + 5 \times 2 = 13$

• The L2-norm of a vector $\mathbf{x} = (x_1 \ x_2 \ \cdots \ x_n)^T$ is a generalization of the Pythagorean theorem for finding the “length”:

$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$
matrix algebra review

• More generally, define two $m \times n$ matrices:

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

Then the matrix multiplication of $X^T$ and $Y$, which results in an $n \times n$ matrix, is:

$$X^T Y = \begin{bmatrix}
x_1 & x_2 & \cdots & x_n
\end{bmatrix}
\begin{bmatrix}
y_1 & y_2 & \cdots & y_n
\end{bmatrix}$$

• For example, with $A$ and $B$ defined below, we get:

$$A = \begin{bmatrix}
-1 & 0 & 1 \\
0 & 2 & 3
\end{bmatrix}, \quad B = \begin{bmatrix}
1 & 2 & 3 \\
3 & 0 & 1
\end{bmatrix} \quad \rightarrow \quad A^T B = \begin{bmatrix}
-1 & 0 & \cdot \\
0 & 2 & \cdot \\
\cdot & \cdot & \cdot
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 3 \\
3 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
-1 & -2 & -3 \\
6 & 0 & 2 \\
10 & 2 & 6
\end{bmatrix}$$
matrix algebra review

- If \( X \) has dimension \( a \times b \), and \( Y \) has dimension \( c \times d \), then the matrix product \( XY \) is only possible if \( b = c \) (i.e., the inner dimensions match), which will have dimension \( a \times d \) (outer dimensions).

- If \( X \) is a square matrix (i.e., has dimension \( n \times n \)), then its inverse is \( X^{-1} \) (if it exists), and:

\[
X^{-1}X = XX^{-1} = I,
\]

where \( I = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} \)

is the \( n \times n \) identity matrix.

- For example, with \( A \) and \( B \) defined as below, we can verify \( B = A^{-1} \), since \( AB = I \):

\[
A = \begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.2 & 0.2 & 0 \\ -0.2 & 0.3 & 1 \\ 0.2 & -0.3 & 0 \end{bmatrix}, \quad AB = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]
• But how do we perform matrix manipulations, like taking inverses, on large matrices in general?

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

```python
import numpy as np

np.array(A)  //Convert list to numpy array
np.matmul(A,B)  //Matrix multiplication (or A@B)
np.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, as well as the notebook
Now, back to regression.

For simple linear regression, if we define

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

then we can write the equations for all data points compactly using the following matrix equation:

\[
y = X\beta + \epsilon
\]

The multivariable linear regression model with \( M \) explanatory variables is

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

In this case, we define

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

where \( X \) is the feature matrix. Then, as before, we can write

\[
y = X\beta + \epsilon
\]
least squares equations

• With this matrix notation, we can write our original optimization for minimizing MSE as:

$$
\min_{\beta} \frac{1}{N} \sum_{n=1}^{N} (y_n - x_n^T \beta)^2
$$

• Or, equivalently, this can be written using the vector norm:

$$
\min_{\beta} \frac{1}{N} \|y - X\beta\|_2^2
$$

• Similar to 1D case, we can take the gradient (multidimensional derivative) and set to 0 (i.e., the vector of zeros) to find minimum:

$$
\nabla \left((1/N)\|y - X\beta\|_2^2\right) = \left(\frac{2}{N}\right)X^T X \beta - \left(\frac{2}{N}\right)X^T y = 0
$$

• This yields the least squares equations for solving for $\beta$:

$$
X^T X \beta = X^T y
$$
solving for $\beta$

- If $X^TX$ is invertible, we can take a matrix inverse to solve for the model parameters $\beta$:
  \[ \beta = (X^TX)^{-1}X^Ty \]

- But $X^TX$ is not always invertible
  - The inverse exists if and only if the columns of $X$ 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 $X^TX$ is not invertible?
  - Infinitely many possible solutions
  - We typically choose the one where $\|\beta\|$ is smallest. Why?
Suppose we collect five data points consisting of two features \( x_1, x_2 \) and a target variable \( y \) in the form \((x_1, x_2, y)\): \((1, 2, 10), (-3, 6, 0), (0, 0, 3), (1, -1, 4), (5, -2, 20)\). We want to fit a linear regression model to this dataset.

What are the least squares equations?

What is the resulting model?

What would be the prediction for a new datapoint with \( x_1 = -1, x_2 = 1 \)?
The model we want to fit is \( \hat{y} = a_1 x_1 + a_2 x_2 + b \), where \( \beta = (a_1 \ a_2 \ b)^T \) is the parameter vector.

The feature matrix \( X \), target vector \( y \), and least squares equations are:

\[
X = \begin{bmatrix}
1 & 2 & 1 \\
-3 & 6 & 1 \\
0 & 0 & 1 \\
1 & -1 & 1 \\
5 & -2 & 1 \\
\end{bmatrix}, \quad y = \begin{bmatrix}
10 \\
0 \\
3 \\
4 \\
20 \\
\end{bmatrix},
\]

\[
\begin{bmatrix}
1 & -3 & 0 & 1 & 5 \\
2 & 6 & 0 & -1 & -2 \\
1 & 1 & 1 & 1 & 1 \\
\end{bmatrix} \begin{bmatrix}
1 & 2 & 1 \\
-3 & 6 & 1 \\
0 & 0 & 1 \\
1 & -1 & 1 \\
5 & -2 & 1 \\
\end{bmatrix} \beta = \begin{bmatrix}
1 & -3 & 0 & 1 & 5 \\
2 & 6 & 0 & -1 & -2 \\
1 & 1 & 1 & 1 & 1 \\
\end{bmatrix} \begin{bmatrix}
10 \\
0 \\
3 \\
4 \\
20 \\
\end{bmatrix}
\]

\[
X^T X \beta = X^T y
\]
solution: model and test prediction

Using the numpy commands for inverse, transpose, and multiplication, we compute the solution: \( \beta = (X^T X)^{-1} X^T y \)

\[
\beta = (4.2308, 1.7538, 2.2615)^T
\]

Which means that our model is

\[
\hat{y} = 4.2308x_1 + 1.7538x_2 + 2.2615
\]

And the prediction for \( x_1 = -1, x_2 = 1 \) is

\[
\hat{y} = 4.2308 \cdot -1 + 1.7538 \cdot 1 + 2.2615 = -0.2154
\]
interpreting results

• How should we interpret the results of linear regression?

• Recall multi-feature model, e.g.,
  \[ y_n = a_1 x_{n,1} + a_2 x_{n,2} + b \]

• If one feature weight (e.g., \(a_1\)) is higher than another (e.g., \(a_2\)), this can indicate that this feature is more important than the other (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

Here, \(x_1\) has a range of 8, while \(x_2\) only has a range of 2
normalization for interpretation

• **Problem:** 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 \( x_2 \) that resulted in \( a_1 = 10 \) and \( a_2 = 100 \).

• **One solution:** **Normalize** the data before doing linear regression so that coefficients are comparable over a consistent range.
standard normalization

• For every feature column, do the following to make them all have a mean of 0 and standard deviation of 1:

  1. *Center values*: Subtract the column average from each feature sample
     • Useful to eliminate any bias contained in the features
  2. *Scale values*: Divide each feature sample by the column standard deviation
     • Re-scales features so that each is expressed in new units: standard deviations from the mean (similar to how we calculate $z$-scores)

• Mathematically, we are defining the following operation for each feature column $\mathbf{x}_m$:

$$\tilde{x}_m = \frac{x_m - \bar{x}_m}{s_m},$$

where $\bar{x}_m$ and $s_m$ are the sample mean and standard deviation of feature $m$. 
How good is the fit of the regression to the dataset?

To answer this, one possibility is using the MSE.

Another commonly used quantity is the coefficient of determination, called $r^2$

$$r^2 = 1 - \frac{\sum_{n=1}^{N} (y_n - \hat{y}_n)^2}{\sum_{n=1}^{N} (y_n - \bar{y})^2} = 1 - \frac{MSE}{\sigma_Y^2}$$

- $y_n$: Measured value, $\hat{y}_n$: Predicted value
- $\bar{y}$: Mean measured value, $\sigma_Y^2$: Variance of measured value

$r^2$ gives the fraction of variance in the data that is explained by the model.

Typically between 0 (bad, no better than horizontal line) and 1 (perfect fit).

Sometimes preferred to MSE in regression problems for this reason.
using your model after fitting

- After fitting a linear regression model, you can estimate (or predict) the target $y$ of new data points using your model
  - 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?
    - Squared error between $\hat{y}$ and $y$ (once it is known)
    - MSE or $r^2$ over a set of new data points
  - When using the model, make sure to take into account any normalization that was used (i.e., normalize new data points before inputting them, “un-normalize” the $\hat{y}$ you get back)
linear regression in python

- You can solve the least squares equations directly using numpy.
- Given how common linear regression is, several variants are built in to the sklearn (scikit learn) library directly:

```python
from sklearn import linear_model, from sklearn.metrics import mean_squared_error, r2_score

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 \( (a_1, \ldots, a_M) \) of trained model
regr.intercept_  # View intercept \( (b) \) of trained model
y_pred = regr.predict(X_test)  # Apply model to test set
r2_score(y_true,y_pred)  # r2 score between true and predicted
```
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 $a_m$ is 0 (feature has no predictivity, $y$ does not depend on $x_m$)
  • *Alternative hypothesis* $H_1$: Coefficient $a_m$ is not 0 (feature has predictivity, $y$ does depend on $x_m$)
hypothesis test for regression

• Test statistic is always: \( \frac{\hat{a}_m - a_m}{SE_{a_m}} \) / standard error

\[
\frac{\hat{a}_m - a_m}{SE_{a_m}} = \frac{\hat{a}_m}{SE_{a_m}}
\]

• What is the standard error for a regression coefficient \( a_m \)?

\[
SE_{a_m} = \sqrt{\frac{\sum_{n=1}^{N} (y_n - \hat{y}_n)^2}{N - 2}} \sqrt{\sum_{n=1}^{N} (x_{n,m} - \bar{x}_m)^2}
\]

• For a \( z \)-test, find \( p \)-value of \( SE_{a_m} \) 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

\( \hat{y}_n \): Measured value, \( x_{n,m} \): Feature value
\( \hat{y}_n \): Predicted value, \( \bar{x}_m \): Feature average
a linear model may be wrong

- In these graphs, all 4 datasets have the same ...
  - linear regression line
  - coefficient of determination
  - mean and variance of both x and y
- Yet clearly, the relationship between x and y is different in each case
- It is important to visualize the results, and possibly try non-linear models!
what about non-linear?

• A common (and understandable) misconception is that linear regression can only find linear relationships

• The “linear” part refers to the parameter vector $\beta$, not the input features in $X$

• We can readily take nonlinear functions of our features

• For example, suppose we want to fit a quadratic model:

$$y_n = a_1(x_n)^2 + a_2x_n + b$$

• We create a “synthesized” feature matrix that has the quadratic form:

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

• If we use a higher degree $d$ of polynomials, we can reduce MSE:

- $d = 0$
- $d = 1$
- $d = 2$
- $d = 4$
- $d = 8$
- $d = 16$

• But, is this a good thing to do?
overfitting

• If our goal was just to minimize error on the existing dataset, we’d keep adding features (e.g., increasing the degree $d$ of 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 $r^2$ and low MSE on training data, but low $r^2$ and high 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
• 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:

$$\text{minimize } (\text{model error}) + \lambda (\text{coefficient weights})$$

• $\lambda \geq 0$ is the regularization parameter

  • Higher $\lambda$: Minimizing model parameters becomes more important
  
  • Lower $\lambda$: 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:

\[
\min_{\beta} \|X\beta - y\|_2^2 + \lambda \|\beta\|_2^2
\]

• This makes it easy to solve in matrix form as:

\[
\beta^* = (X^TX + \lambda I)^{-1}X^Ty
\]

• In Python (where \( \alpha \) is the regularization parameter):

```python
from sklearn import linear_model
reg = linear_model.Ridge(alpha=0.1, fit_intercept=True)
```
regularization can alleviate overfitting

- Polynomial of degree $d = 10$, with different amounts of regularization:

  - A higher value of $\lambda$ has a "smoothing" effect on the model
evaluating predictive performance

• Descriptive and diagnostic analysis (classical statistics, data mining)
  • Focus: Understand and interpret statistical relationships in observed dataset
  • Evaluation: e.g., MSE or $r^2$ on training data (data used to fit the model)

• Predictive and prescriptive analysis (machine learning)
  • Focus: Predict target value for new or future unseen data
  • Evaluation: e.g., MSE or $r^2$ on test data (data not used to fit the model)
why evaluate on test data?

• Analogy to class
  • **Training data** is like homeworks, sample problems and sample exams
  • **Testing data** is like the real exam

• If we train and evaluate on the same data, the model may not generalize well

• Reasons for computing performance on *test data* (the standard ML approach):
  • **Model evaluation**: Quantify the model’s predictive performance *if deployed*
    • e.g., describing the model and its business implications to the CEO
  • **Model selection**: Select which model should be deployed
    • e.g., which polynomial degree or regularization value should be used?
choosing model based on test MSE

- We can use MSE on a held-out test set to determine the best model:

  - Blue points: Training set
  - Orange points: Held-out test set
choosing model based on test MSE

- We can use MSE on a held-out test set to determine the best model:

- The best model has the lowest test MSE

- This is often achieved when there is a small difference between training and test MSE
simulating testing data

• Ultimately, we’d like to actually test the model in the real world (e.g., predict tomorrow’s temperature)

• However, this is usually quite costly, time consuming, or downright impossible, so we have to simulate it

• To do this, we can split our dataset into:
  
  • **Training data**: A subset we use to train/fit the model
  
  • **Testing data**: A subset we used to report the generalized performance

  • Common splits: 90/10 (i.e., 90% training and 10% test) and 80/20

• **Note**: It is important that the algorithm never sees the testing data (just like it is important that students don’t see the real midterm)
• **$k$-fold cross validation** (often abbreviated CV) repeats the train/test split idea $k$ times, across different **folds** of the data

• The data is divided into $k$ parts

• In each fold, one part is used as the testing set, and the other $k - 1$ are used as the training set

• Thus, there are $k$ models fit throughout this process, and we can average testing performance (and sometimes the coefficients)

• How many folds should be used?

• 3-fold, 5-fold and 10-fold are common

• **Leave-one-out CV**: $k$ is the number of datapoints, i.e., one is held out in each fold (computationally expensive)
cross validation for model selection

• How do we determine the right value of $\lambda$?

• Test a wide range of $\lambda$ typically log scale, e.g., 0.01,...,0.1,...,1,...,10,...,100

• Use multiple CV iterations, one for each value of $\lambda$:

  - Train all folds with $\lambda = 0.01$
  - Train all folds with $\lambda = 0.1$
  - Train all folds with $\lambda = 1$
  - Train all folds with $\lambda = 10$

  Choose $\lambda^*$ whose CV performance is the best

  For final model, train model with all data using $\lambda^*$

Dataset $\rightarrow$ Training with $\lambda^*$ $\rightarrow$ Final model
Suppose we collect three data points with a single feature $x$ and target variable $y$. In the form $(x, y)$, they are, approximately: $(2.18, 2.26)$, $(0.13, -14.57)$, $(2.75, 16.74)$.

Find the linear regression model $\hat{y} = ax + b$ and corresponding regularization parameter $\lambda$ which has minimum cross validation error.

Use the Ridge model, $k = 3$ folds, and test $\lambda = 0, 0.1, 1$. Note that the coefficient $b$ should NOT be regularized.
solution

• We need to solve the least squares equations for three values of lambda, and three folds each (i.e., 9 cases total). Here is the math for \( \lambda = 0, 0.1 \) and the second fold:

\[
\begin{align*}
\text{fold}=2, \lambda=0.0 \\
x &\sim [2.18, 0.13, 2.75] \\
y &\sim [2.26, -14.57, 16.74] \\
X: \\
\begin{bmatrix}
2.17997451 & 1. \\
2.74831239 & 1.
\end{bmatrix} \\
X^T @ X: \\
\begin{bmatrix}
12.30550986 & 4.9282869 \\
4.9282869 & 2.
\end{bmatrix} \\
X^T @ X + \lambda I: \\
\begin{bmatrix}
12.4 & 4.9282869 \\
4.9282869 & 2.
\end{bmatrix} \\
(X^T @ X + \lambda I)^{-1}: \\
\begin{bmatrix}
3.82403369 & -9.42296757 \\
-9.42296757 & 23.71954383
\end{bmatrix} \\
(X^T @ X + \lambda I)^{-1} @ X^T: \\
\begin{bmatrix}
-1.0866716 & 1.0866716 \\
3.1777147 & -2.1777147
\end{bmatrix} \\
(X^T @ X + \lambda I)^{-1} @ X^T @ y: \\
\begin{bmatrix}
-1.75951672 & 1.75951672 \\
4.8357016 & -3.8357016
\end{bmatrix} \\
\text{Only coefficient is changed by } \lambda, \text{ intercept is not regularized} \\
\text{Notice how different the inverse is just from a small } \lambda
\end{align*}
\]

\[
\begin{align*}
\text{fold}=2, \lambda=0.1 \\
x &\sim [2.18, 0.13, 2.75] \\
y &\sim [2.26, -14.57, 16.74] \\
X: \\
\begin{bmatrix}
2.17997451 & 1. \\
2.74831239 & 1.
\end{bmatrix} \\
X^T @ X: \\
\begin{bmatrix}
12.30550986 & 4.9282869 \\
4.9282869 & 2.
\end{bmatrix} \\
X^T @ X + \lambda I: \\
\begin{bmatrix}
12.40550986 & 4.9282869 \\
4.9282869 & 2.
\end{bmatrix} \\
(X^T @ X + \lambda I)^{-1}: \\
\begin{bmatrix}
3.82403369 & -9.42296757 \\
-9.42296757 & 23.71954383
\end{bmatrix} \\
(X^T @ X + \lambda I)^{-1} @ X^T: \\
\begin{bmatrix}
-1.0866716 & 1.0866716 \\
3.1777147 & -2.1777147
\end{bmatrix} \\
(X^T @ X + \lambda I)^{-1} @ X^T @ y: \\
\begin{bmatrix}
-1.0866716 & 1.0866716 \\
3.1777147 & -2.1777147
\end{bmatrix} \\
\text{notice how different the inverse is just from a small } \lambda
\end{align*}
\]
fold=2, lambda=0.0

\[
x = [2.18, 0.13, 2.75] \\
y = [2.26, -14.57, 16.74]
\]

\[
X = \begin{bmatrix}
2.17997451 & 1.00 \\
2.74831239 & 1.00
\end{bmatrix}
\]

\[
X^T @ X = \begin{bmatrix}
12.3050986 & 4.9282869 \\
4.9282869 & 2.00
\end{bmatrix}
\]

\[
X^T @ X + \lambda I = \begin{bmatrix}
12.3050986 & 4.9282869 \\
4.9282869 & 2.00
\end{bmatrix}
\]

\[
(X^T @ X + \lambda I)^{-1} = \begin{bmatrix}
6.19179817 & -15.25747891 \\
-15.25747891 & 35.09661673
\end{bmatrix}
\]

\[
(X^T @ X + \lambda I)^{-1} @ X^T @ y = \begin{bmatrix}
-1.75951672 & 1.75951672 \\
3.1777147 & -2.1777147
\end{bmatrix}
\]

\[
fold=2, lambda=0.1
\]

\[
x = [2.18, 0.13, 2.75] \\
y = [2.26, -14.57, 16.74]
\]

\[
X = \begin{bmatrix}
2.17997451 & 1.00 \\
2.74831239 & 1.00
\end{bmatrix}
\]

\[
X^T @ X = \begin{bmatrix}
12.3050986 & 4.9282869 \\
4.9282869 & 2.00
\end{bmatrix}
\]

\[
X^T @ X + \lambda I = \begin{bmatrix}
12.3050986 & 4.9282869 \\
4.9282869 & 2.00
\end{bmatrix}
\]

\[
(X^T @ X + \lambda I)^{-1} = \begin{bmatrix}
3.82403369 & -9.42296757 \\
-9.42296757 & 23.71954383
\end{bmatrix}
\]

\[
(X^T @ X + \lambda I)^{-1} @ X^T @ y = \begin{bmatrix}
-1.0866716 & 1.0866716 \\
3.1777147 & -2.1777147
\end{bmatrix}
\]

\[
\lambda^* = 0.10 \text{ has best average test MSE}
\]