Chapter 2. Regression

Reza Rezazadegan

Sharif University of Technology

October 16, 2022

Ordinary Least Squares (OLS) Regression

- The assumption behind OLS is that data is linear and the deviation of data from a straight line is just noise.

Ordinary Least Squares (OLS) Regression

- The assumption behind OLS is that data is linear and the deviation of data from a straight line is just noise.
- Problem: given a set of datapoints $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ find a linear function $f(\mathbf{x})$ that "best" approximates (or fits) D.

Ordinary Least Squares (OLS) Regression

- The assumption behind OLS is that data is linear and the deviation of data from a straight line is just noise.
- Problem: given a set of datapoints $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ find a linear function $f(\mathbf{x})$ that “best” approximates (or fits) D.
- We have $y_i \in \mathbb{R}$ but $\mathbf{x}_i \in \mathbb{R}^d$.

Ordinary Least Squares (OLS) Regression

- The assumption behind OLS is that data is linear and the deviation of data from a straight line is just noise.
- Problem: given a set of datapoints $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ find a linear function $f(\mathbf{x})$ that "best" approximates (or fits) D.
- We have $y_i \in \mathbb{R}$ but $\mathbf{x}_i \in \mathbb{R}^d$.
- Best approximation means that f is a minimum of the error function $E(f) = \sum_i (y_i - f(\mathbf{x}_i))^2$.

Ordinary Least Squares (OLS) Regression

- The assumption behind OLS is that data is linear and the deviation of data from a straight line is just noise.
- Problem: given a set of datapoints $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ find a linear function $f(\mathbf{x})$ that "best" approximates (or fits) D.
- We have $y_i \in \mathbb{R}$ but $\mathbf{x}_i \in \mathbb{R}^d$.
- Best approximation means that f is a minimum of the error function $E(f) = \sum_i (y_i - f(\mathbf{x}_i))^2$.
- Geometrically, this is equivalent to finding a hypersurface in $\mathbb{R}^{d+1}$ which minimizes the sum of the squares of distances of the points $\{(\mathbf{x}_i, y)\}_{i=1}^n$.

Ordinary Least Squares (OLS) Regression

- The assumption behind OLS is that data is linear and the deviation of data from a straight line is just noise.
- Problem: given a set of datapoints $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ find a linear function $f(\mathbf{x})$ that "best" approximates (or fits) D.
- We have $y_i \in \mathbb{R}$ but $\mathbf{x}_i \in \mathbb{R}^d$.
- Best approximation means that f is a minimum of the error function $E(f) = \sum_i (y_i - f(\mathbf{x}_i))^2$.
- Geometrically, this is equivalent to finding a hypersurface in $\mathbb{R}^{d+1}$ which minimizes the sum of the squares of distances of the points $\{(\mathbf{x}_i, y)\}_{i=1}^n$. Equivalently, it maximizes the sum of the squares of lengths of the projections of the points to the plain (called the *best-fit hypersurface* for the points).

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and
  $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.
- Then

$$E(f) = E(\mathbf{a}) = ||\mathbf{y} - X\mathbf{a}||^2 \tag{1}$$

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.
- Then

$$E(f) = E(\mathbf{a}) = ||\mathbf{y} - X\mathbf{a}||^2 \tag{1}$$

- The solution $\mathbf{a}$ is the "closest" vector to $X^{-1}\mathbf{y}$.

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.
- Then
$$E(f) = E(\mathbf{a}) = ||\mathbf{y} - X\mathbf{a}||^2 \tag{1}$$
- The solution $\mathbf{a}$ is the "closest" vector to $X^{-1}\mathbf{y}$.
- $E(\mathbf{a}) = (\mathbf{y} - X\mathbf{a})^t(\mathbf{y} - X\mathbf{a})$

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.
- Then

$$E(f) = E(\mathbf{a}) = ||\mathbf{y} - X\mathbf{a}||^2 \tag{1}$$

- The solution $\mathbf{a}$ is the "closest" vector to $X^{-1}\mathbf{y}$.
- $E(\mathbf{a}) = (\mathbf{y} - X\mathbf{a})^t(\mathbf{y} - X\mathbf{a}) = \mathbf{y}^t\mathbf{y} - \mathbf{y}^t X\mathbf{a} - \mathbf{a}^t X^t\mathbf{y} + \mathbf{a}^t X^t X\mathbf{a}$

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.
- Then
$$E(f) = E(\mathbf{a}) = ||\mathbf{y} - X\mathbf{a}||^2 \tag{1}$$
- The solution $\mathbf{a}$ is the "closest" vector to $X^{-1}\mathbf{y}$.
- $E(\mathbf{a}) = (\mathbf{y} - X\mathbf{a})^t(\mathbf{y} - X\mathbf{a}) = \mathbf{y}^t\mathbf{y} - \mathbf{y}^t X\mathbf{a} - \mathbf{a}^t X^t\mathbf{y} + \mathbf{a}^t X^t X\mathbf{a}$
- $E(\mathbf{a}) = \mathbf{y}^t\mathbf{y} - 2\mathbf{a}^t X^t\mathbf{y} + \mathbf{a}^t X^t X\mathbf{a}$
- $\frac{d}{d\mathbf{a}} E(\mathbf{a}) = -2X^t\mathbf{y} + 2X^t X\mathbf{a}$

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.
- Then
$$E(f) = E(\mathbf{a}) = ||\mathbf{y} - X\mathbf{a}||^2 \qquad (1)$$
- The solution $\mathbf{a}$ is the "closest" vector to $X^{-1}\mathbf{y}$.
- $E(\mathbf{a}) = (\mathbf{y} - X\mathbf{a})^t(\mathbf{y} - X\mathbf{a}) = \mathbf{y}^t\mathbf{y} - \mathbf{y}^t X\mathbf{a} - \mathbf{a}^t X^t \mathbf{y} + \mathbf{a}^t X^t X\mathbf{a}$
- $E(\mathbf{a}) = \mathbf{y}^t\mathbf{y} - 2\mathbf{a}^t X^t \mathbf{y} + \mathbf{a}^t X^t X\mathbf{a}$
- $\frac{d}{d\mathbf{a}} E(\mathbf{a}) = -2X^t\mathbf{y} + 2X^t X\mathbf{a}$
- $\frac{d}{d\mathbf{a}} E(\mathbf{a}) = 0$ if and only if $X^t X\mathbf{a} = X^t\mathbf{y}$

Solving the optimization problem for linear regression

- Write $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$,
  Let X be the $n \times d$ matrix whose rows are the vectors $\mathbf{x}_i$ and $\mathbf{y} = (y_1 - b, y_2 - b, \ldots, y_n - b)$.
- Then

$$E(f) = E(\mathbf{a}) = ||\mathbf{y} - X\mathbf{a}||^2 \tag{1}$$

- The solution $\mathbf{a}$ is the "closest" vector to $X^{-1}\mathbf{y}$.
- $E(\mathbf{a}) = (\mathbf{y} - X\mathbf{a})^t(\mathbf{y} - X\mathbf{a}) = \mathbf{y}^t\mathbf{y} - \mathbf{y}^tX\mathbf{a} - \mathbf{a}^tX^t\mathbf{y} + \mathbf{a}^tX^tX\mathbf{a}$
- $E(\mathbf{a}) = \mathbf{y}^t\mathbf{y} - 2\mathbf{a}^tX^t\mathbf{y} + \mathbf{a}^tX^tX\mathbf{a}$
- $\frac{d}{d\mathbf{a}}E(\mathbf{a}) = -2X^t\mathbf{y} + 2X^tX\mathbf{a}$
- $\frac{d}{d\mathbf{a}}E(\mathbf{a}) = 0$ if and only if $X^tX\mathbf{a} = X^t\mathbf{y}$
- If X^tX is invertible then $\mathbf{a} = (X^tX)^{-1}X^t\mathbf{y}$. (The normal equation)

Collinearity of features

- Remember: the solution to the linear regression problem is given by $\mathbf{a} = (X^t X)^{-1} X^t \mathbf{y}$ where the rows of $X_{n \times d}$ are the data points $\mathbf{x_i}$.

Collinearity of features

- Remember: the solution to the linear regression problem is given by $\mathbf{a} = (X^t X)^{-1} X^t \mathbf{y}$ where the rows of $X_{n \times d}$ are the data points $\mathbf{x_i}$.
- $X^t X$ is invertible if $n \geq d$ $\operatorname{rank} X = d$

Collinearity of features

- Remember: the solution to the linear regression problem is given by $\mathbf{a} = (X^t X)^{-1} X^t \mathbf{y}$ where the rows of $X_{n\times d}$ are the data points $\mathbf{x_i}$.
- $X^t X$ is invertible if $n \geq d$ $\operatorname{rank} X = d$ i.e. if the features are linearly independent.
- **Collinearity:** when the features are nearly linearly dependent and thus, $\det X$ is close to zero.

Collinearity of features

- Remember: the solution to the linear regression problem is given by $\mathbf{a} = (X^t X)^{-1} X^t \mathbf{y}$ where the rows of $X_{n \times d}$ are the data points $\mathbf{x_i}$.
- $X^t X$ is invertible if $n \geq d$ $\operatorname{rank} X = d$ i.e. if the features are linearly independent.
- **Collinearity:** when the features are nearly linearly dependent and thus, $\det X$ is close to zero.
- For example the size and the price of a house are nearly linearly correlated.

Collinearity of features

- Remember: the solution to the linear regression problem is given by $\mathbf{a} = (X^t X)^{-1} X^t \mathbf{y}$ where the rows of $X_{n\times d}$ are the data points $\mathbf{x_i}$.
- $X^t X$ is invertible if $n \geq d$ rank $X = d$ i.e. if the features are linearly independent.
- **Collinearity:** when the features are nearly linearly dependent and thus, $\det X$ is close to zero.
- For example the size and the price of a house are nearly linearly correlated.
- In case of collinearity, the entries of $(X^t X)^{-1}$ can be very large and will jump by small perturbations of data.

Correlation

- Correlation between a feature and the target variable good, correlation between two features bad!

Correlation

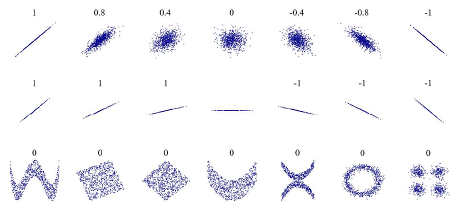
- Correlation between a feature and the target variable good, correlation between two features bad!
- **Linear Correlation** (Pearson Correlation) between two random variables A, B is given by

$$\frac{E[AB] - E[A]E[B]}{\sigma_A \sigma_b}$$

Correlation

- Correlation between a feature and the target variable good, correlation between two features bad!
- **Linear Correlation** (Pearson Correlation) between two random variables A, B is given by

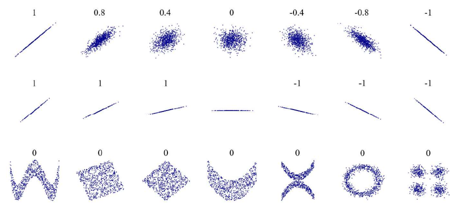
$$\frac{E[AB] - E[A]E[B]}{\sigma_A \sigma_b}$$



Correlation

- Correlation between a feature and the target variable good, correlation between two features bad!
- **Linear Correlation** (Pearson Correlation) between two random variables A, B is given by

$$\frac{E[AB] - E[A]E[B]}{\sigma_A \sigma_b}$$

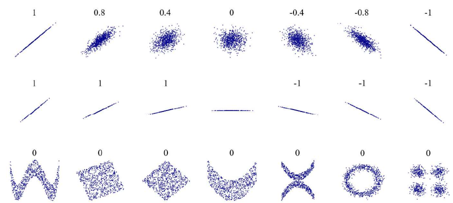


- Available in Python as `scipy.stats.pearsonr(A,B)` and `scikit.feature_selection.r_regression(X, y)`

Correlation

- Correlation between a feature and the target variable good, correlation between two features bad!
- **Linear Correlation** (Pearson Correlation) between two random variables A, B is given by

$$\frac{E[AB] - E[A]E[B]}{\sigma_A \sigma_b}$$

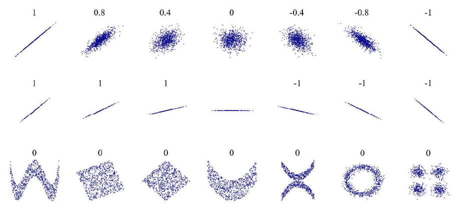


- Available in Python as `scipy.stats.pearsonr(A,B)` and `scikit.feature_selection.r_regression(X, y)`
- **Spearman Correlation** is the Pearson correlation between the rank variables of the random variables A, B.

Correlation

- Correlation between a feature and the target variable good, correlation between two features bad!
- **Linear Correlation** (Pearson Correlation) between two random variables A, B is given by

$$\frac{E[AB] - E[A]E[B]}{\sigma_A \sigma_b}$$

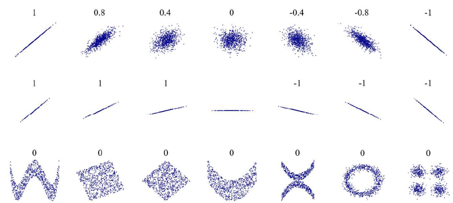


- Available in Python as `scipy.stats.pearsonr(A,B)` and `scikit.feature_selection.r_regression(X, y)`
- **Spearman Correlation** is the Pearson correlation between the rank variables of the random variables A, B. Rank variable R_A of A uses the rank of a value a among all the values of A.

Correlation

- Correlation between a feature and the target variable good, correlation between two features bad!
- **Linear Correlation** (Pearson Correlation) between two random variables A, B is given by

$$\frac{E[AB] - E[A]E[B]}{\sigma_A \sigma_b}$$

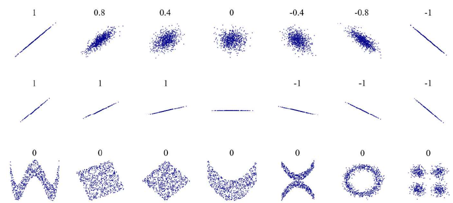


- Available in Python as `scipy.stats.pearsonr(A,B)` and `scikit.feature_selection.r_regression(X, y)`
- **Spearman Correlation** is the Pearson correlation between the rank variables of the random variables A, B. Rank variable R_A of A uses the rank of a value a among all the values of A.
- Available in Python as `scipy.stats.spearmanr(A,B)`

Correlation

- Correlation between a feature and the target variable good, correlation between two features bad!
- **Linear Correlation** (Pearson Correlation) between two random variables A, B is given by

$$\frac{E[AB] - E[A]E[B]}{\sigma_A \sigma_b}$$



- Available in Python as `scipy.stats.pearsonr(A,B)` and `scikit.feature_selection.r_regression(X, y)`
- **Spearman Correlation** is the Pearson correlation between the rank variables of the random variables A, B. Rank variable R_A of A uses the rank of a value a among all the values of A.
- Available in Python as `scipy.stats.spearmanr(A,B)`

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.
- The diagonal elements of D are the nonzero eigenvalues of X^tX, called the *singular values* of X.

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.
- The diagonal elements of D are the nonzero eigenvalues of $X^t X$, called the *singular values* of X.
- We have $(X^t X)^{-1} X^t = VD^{-1} U^t$

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.
- The diagonal elements of D are the nonzero eigenvalues of $X^t X$, called the *singular values* of X.
- We have $(X^t X)^{-1} X^t = VD^{-1}U^t$
- Computing SVD is computationally less complex than inverting $X^t X$.

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.
- The diagonal elements of D are the nonzero eigenvalues of X^tX, called the *singular values* of X.
- We have $(X^tX)^{-1}X^t = VD^{-1}U^t$
- Computing SVD is computationally less complex than inverting X^tX.
- The `linear_model.LinearRegression` class in Scikit-Learn uses SVD.

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.
- The diagonal elements of D are the nonzero eigenvalues of X^tX, called the *singular values* of X.
- We have $(X^tX)^{-1}X^t = VD^{-1}U^t$
- Computing SVD is computationally less complex than inverting X^tX.
- The `linear_model.LinearRegression` class in Scikit-Learn uses SVD. It sets the diagonal elements of D smaller than a threshold to zero.

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.
- The diagonal elements of D are the nonzero eigenvalues of X^tX, called the *singular values* of X.
- We have $(X^tX)^{-1}X^t = VD^{-1}U^t$
- Computing SVD is computationally less complex than inverting X^tX.
- The `linear_model.LinearRegression` class in Scikit-Learn uses SVD. It sets the diagonal elements of D smaller than a threshold to zero.
- Note: X^tX is a $d \times d$ matrix where d is the number of features.

Singular Value Decomposition (SVD)

- Theorem. Each real matrix X has a decomposition of the form $X = UDV^t$ where U, V are orthonormal and D is diagonal with positive real entries. [Foundations of Data Science, Chapter 3]
- Moreover the columns of V span the best-fitting subspace for the rows of X.
- D is an $r \times r$ matrix such that r is the rank of X.
- The diagonal elements of D are the nonzero eigenvalues of X^tX, called the *singular values* of X.
- We have $(X^tX)^{-1}X^t = VD^{-1}U^t$
- Computing SVD is computationally less complex than inverting X^tX.
- The `linear_model.LinearRegression` class in Scikit-Learn uses SVD. It sets the diagonal elements of D smaller than a threshold to zero.
- Note: X^tX is a $d \times d$ matrix where d is the number of features. Thus SVD is $O(d^2)$ but linear in terms of the number of instances!

Polynomial Regression

- Polynomial regression can be done using linear regression, by adding the nonlinear terms as new features.

Polynomial Regression

- Polynomial regression can be done using linear regression, by adding the nonlinear terms as new features.
- For example if we have two features z_1, z_2 and want degree 3 polynomial regression, we then add $z_1^2, z_1 z_2, z_2^2, z_1^3, z_1^2 z_2, z_1 z_2^2, z_2^3$ as new features.

Polynomial Regression

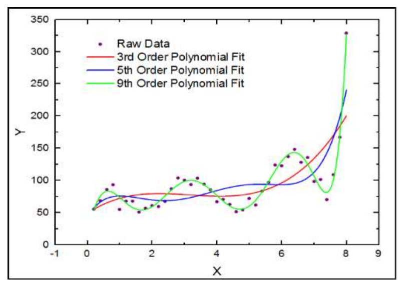
- Polynomial regression can be done using linear regression, by adding the nonlinear terms as new features.
- For example if we have two features z_1, z_2 and want degree 3 polynomial regression, we then add $z_1^2, z_1 z_2, z_2^2, z_1^3, z_1^2 z_2, z_1 z_2^2, z_2^3$ as new features.
- The degree of the polynomial is a *hyperparameter* of the model.

Polynomial Regression

- Polynomial regression can be done using linear regression, by adding the nonlinear terms as new features.
- For example if we have two features z_1, z_2 and want degree 3 polynomial regression, we then add $z_1^2, z_1 z_2, z_2^2, z_1^3, z_1^2 z_2, z_1 z_2^2, z_2^3$ as new features.
- The degree of the polynomial is a *hyperparameter* of the model.
- Unlike parameters, hyperparameters of the model are not learned during training.

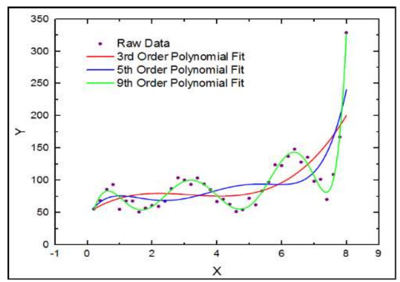
Polynomial Regression

- Polynomial regression can be done using linear regression, by adding the nonlinear terms as new features.
- For example if we have two features z_1, z_2 and want degree 3 polynomial regression, we then add $z_1^2, z_1z_2, z_2^2, z_1^3, z_1^2z_2, z_1z_2^2, z_2^3$ as new features.
- The degree of the polynomial is a *hyperparameter* of the model.
- Unlike parameters, hyperparameters of the model are not learned during training.



Polynomial Regression

- Polynomial regression can be done using linear regression, by adding the nonlinear terms as new features.
- For example if we have two features z_1, z_2 and want degree 3 polynomial regression, we then add $z_1^2, z_1z_2, z_2^2, z_1^3, z_1^2z_2, z_1z_2^2, z_2^3$ as new features.
- The degree of the polynomial is a *hyperparameter* of the model.
- Unlike parameters, hyperparameters of the model are not learned during training.



- In Python, polynomial features can be obtained using `sklearn.preprocessing.PolynomialFeatures`

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.
- This is called the **Method of Gradient Descent**

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.
- This is called the **Method of Gradient Descent**

$$\mathbf{a}_{i+1} = \mathbf{a}_i - \lambda \cdot \nabla L(\mathbf{a}_i) \tag{2}$$

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.
- This is called the **Method of Gradient Descent**

$$\mathbf{a}_{i+1} = \mathbf{a}_i - \lambda \cdot \nabla L(\mathbf{a}_i) \tag{2}$$

$\mathbf{a}_0$ is chosen randomly.

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.
- This is called the **Method of Gradient Descent**

$$\mathbf{a}_{i+1} = \mathbf{a}_i - \lambda \cdot \nabla L(\mathbf{a}_i) \tag{2}$$

  $\mathbf{a}_0$ is chosen randomly.
- λ is a hyperparameter called *learning rate*.

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.
- This is called the **Method of Gradient Descent**

$$\mathbf{a}_{i+1} = \mathbf{a}_i - \lambda \cdot \nabla L(\mathbf{a}_i) \tag{2}$$

  $\mathbf{a}_0$ is chosen randomly.
- λ is a hyperparameter called *learning rate*.
- For linear regression, $L(\mathbf{a}) = \sum_k (\mathbf{a}^t \mathbf{x}_k - y_k)^2$ and thus $\frac{\partial}{\partial_{a_i}} L(\mathbf{a}) = 2 \sum_{k=1}^{n} (\mathbf{a}^t \mathbf{x}_k - y_k) X_{k,i}$. $\quad \nabla L(\mathbf{a}) = 2(\mathbf{a}^t X - \mathbf{y}) X^t$

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.
- This is called the **Method of Gradient Descent**

$$\mathbf{a}_{i+1} = \mathbf{a}_i - \lambda \cdot \nabla L(\mathbf{a}_i) \tag{2}$$

  $\mathbf{a}_0$ is chosen randomly.
- λ is a hyperparameter called *learning rate*.
- For linear regression, $L(\mathbf{a}) = \sum_k (\mathbf{a}^t \mathbf{x}_k - y_k)^2$ and thus $\frac{\partial}{\partial_{a_i}} L(\mathbf{a}) = 2 \sum_{k=1}^{n} (\mathbf{a}^t \mathbf{x}_k - y_k) X_{k,i}$. $\quad \nabla L(\mathbf{a}) = 2(\mathbf{a}^t X - \mathbf{y}) X^t$
- Note that at each point $\mathbf{a}_i$ in the *parameter space*, the cost function has to be evaluated on the whole dataset!

The Method of Gradient Descent

- Not all optimization problems in machine learning have a closed-form solution as in linear regression e.g. neural networks or evolutionary algorithms.
- For differentiable *cost (loss or error) functions* L, we know that at each point $\mathbf{a}$, moving slightly in the direction of $-\nabla L(\mathbf{a})$, the value of L decreases.
- This is called the **Method of Gradient Descent**

$$\mathbf{a}_{i+1} = \mathbf{a}_i - \lambda \cdot \nabla L(\mathbf{a}_i) \tag{2}$$

  $\mathbf{a}_0$ is chosen randomly.
- λ is a hyperparameter called *learning rate*.
- For linear regression, $L(\mathbf{a}) = \sum_k (\mathbf{a}^t \mathbf{x}_k - y_k)^2$ and thus $\frac{\partial}{\partial_{a_i}} L(\mathbf{a}) = 2 \sum_{k=1}^{n} (\mathbf{a}^t \mathbf{x}_k - y_k) X_{k,i}$. $\quad \nabla L(\mathbf{a}) = 2(\mathbf{a}^t X - \mathbf{y}) X^t$
- Note that at each point $\mathbf{a}_i$ in the *parameter space*, the cost function has to be evaluated on the whole dataset! But GD scales well with the number of features.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

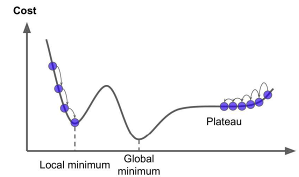


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

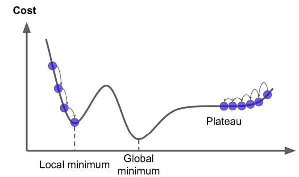


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- **Pitfall II:** It may take a very long time for the method to converge.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

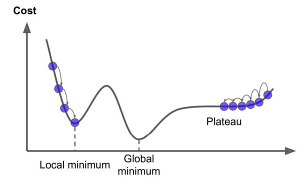


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- **Pitfall II:** It may take a very long time for the method to converge.
- Convex cost functions have only one minimum. E.g. in linear regression.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

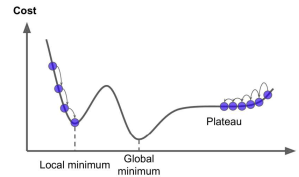


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- **Pitfall II:** It may take a very long time for the method to converge.
- Convex cost functions have only one minimum. E.g. in linear regression.
- **Exploration:** means searching different parts of the "fitness landscape" in search of the global minimum.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

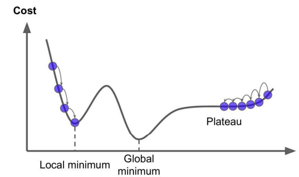


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- **Pitfall II:** It may take a very long time for the method to converge.
- Convex cost functions have only one minimum. E.g. in linear regression.
- **Exploration:** means searching different parts of the "fitness landscape" in search of the global minimum. Corresponds to a large learning rate.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

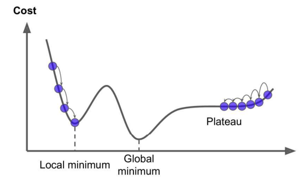


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- **Pitfall II:** It may take a very long time for the method to converge.
- Convex cost functions have only one minimum. E.g. in linear regression.
- **Exploration:** means searching different parts of the "fitness landscape" in search of the global minimum. Corresponds to a large learning rate.
- **Exploitation:** means using the regularity of the cost function to make local improvements to the cost function.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

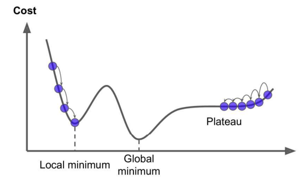


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- **Pitfall II:** It may take a very long time for the method to converge.
- Convex cost functions have only one minimum. E.g. in linear regression.
- **Exploration:** means searching different parts of the "fitness landscape" in search of the global minimum. Corresponds to a large learning rate.
- **Exploitation:** means using the regularity of the cost function to make local improvements to the cost function. Corresponds to small values of the learning rate.

Pitfalls of the Method of Gradient Descent

- **Pitfall I:** The search may end up in a *local* minimum.

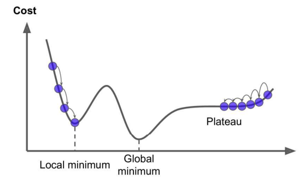


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- **Pitfall II:** It may take a very long time for the method to converge.
- Convex cost functions have only one minimum. E.g. in linear regression.
- **Exploration:** means searching different parts of the "fitness landscape" in search of the global minimum. Corresponds to a large learning rate.
- **Exploitation:** means using the regularity of the cost function to make local improvements to the cost function. Corresponds to small values of the learning rate.
- More complex (or rugged) fitness landscapes need more exploration.

Different types of Gradient Descent algorithms

- **Batch Gradient Descent:** uses the whole dataset to compute the gradient at each step. Slow for large datasets!

Different types of Gradient Descent algorithms

- **Batch Gradient Descent:** uses the whole dataset to compute the gradient at each step. Slow for large datasets!
- **Stochastic Gradient Descent (SDG):** uses a random instance to compute the gradient at each step.

Different types of Gradient Descent algorithms

- **Batch Gradient Descent:** uses the whole dataset to compute the gradient at each step. Slow for large datasets!
- **Stochastic Gradient Descent (SDG):** uses a random instance to compute the gradient at each step.
- **Mimi-batch Gradient Descent:** computes the gradient on small random subsets of the training dataset (called *mini-batches*).

Different types of Gradient Descent algorithms

- **Batch Gradient Descent:** uses the whole dataset to compute the gradient at each step. Slow for large datasets!
- **Stochastic Gradient Descent (SDG):** uses a random instance to compute the gradient at each step.
- **Mimi-batch Gradient Descent:** computes the gradient on small random subsets of the training dataset (called *mini-batches*).
- SGD is much faster than batch gradient descent.

Different types of Gradient Descent algorithms

- **Batch Gradient Descent:** uses the whole dataset to compute the gradient at each step. Slow for large datasets!
- **Stochastic Gradient Descent (SDG):** uses a random instance to compute the gradient at each step.
- **Mimi-batch Gradient Descent:** computes the gradient on small random subsets of the training dataset (called *mini-batches*).
- SGD is much faster than batch gradient descent. Can be used in online learning too.

Different types of Gradient Descent algorithms

- **Batch Gradient Descent:** uses the whole dataset to compute the gradient at each step. Slow for large datasets!
- **Stochastic Gradient Descent (SDG):** uses a random instance to compute the gradient at each step.
- **Mimi-batch Gradient Descent:** computes the gradient on small random subsets of the training dataset (called *mini-batches*).
- SGD is much faster than batch gradient descent. Can be used in online learning too.
- For linear regression, this is equivalent to replacing $L(\mathbf{a}) = \sum_k (\mathbf{a}^t \mathbf{x}_k - y_k)^2$ with $(\mathbf{a}^t \mathbf{x}_k - y_k)^2$ for a random k.

Different types of Gradient Descent algorithms

- **Batch Gradient Descent:** uses the whole dataset to compute the gradient at each step. Slow for large datasets!
- **Stochastic Gradient Descent (SDG):** uses a random instance to compute the gradient at each step.
- **Mimi-batch Gradient Descent:** computes the gradient on small random subsets of the training dataset (called *mini-batches*).
- SGD is much faster than batch gradient descent. Can be used in online learning too.
- For linear regression, this is equivalent to replacing $L(\mathbf{a}) = \sum_k (\mathbf{a}^t \mathbf{x}_k - y_k)^2$ with $(\mathbf{a}^t \mathbf{x}_k - y_k)^2$ for a random k.

Stochastic Gradient Descent

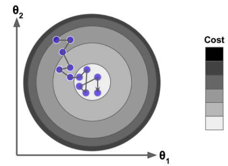


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

Stochastic Gradient Descent

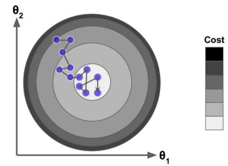


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- SGD is much less regular than Batch GD.

Stochastic Gradient Descent

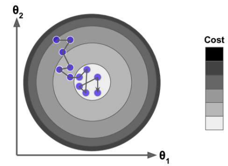


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- SGD is much less regular than Batch GD. It also does not stop after reaching a minimum.

Stochastic Gradient Descent

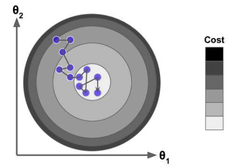


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- SGD is much less regular than Batch GD. It also does not stop after reaching a minimum. Less likely to get stuck in a local minimum.

Stochastic Gradient Descent

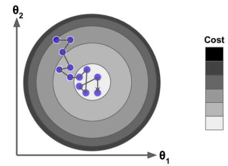


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- SGD is much less regular than Batch GD. It also does not stop after reaching a minimum. Less likely to get stuck in a local minimum.
- To make SGD stop after reaching a minimum, we use a *learning schedule* i.e. decreasing the learning rate gradually.

Stochastic Gradient Descent

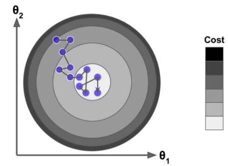


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- SGD is much less regular than Batch GD. It also does not stop after reaching a minimum. Less likely to get stuck in a local minimum.
- To make SGD stop after reaching a minimum, we use a *learning schedule* i.e. decreasing the learning rate gradually.
- If n is the size of the training dataset, each round of n iterations of the SGD is called an *epoch*.

Stochastic Gradient Descent

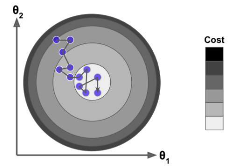


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- SGD is much less regular than Batch GD. It also does not stop after reaching a minimum. Less likely to get stuck in a local minimum.
- To make SGD stop after reaching a minimum, we use a *learning schedule* i.e. decreasing the learning rate gradually.
- If n is the size of the training dataset, each round of n iterations of the SGD is called an *epoch*.
- In each epoch, some datapoints may not be chosen at all while others may be chosen several times.

Stochastic Gradient Descent

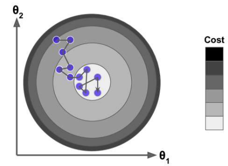


Figure: Credit: Aurelien Geron, Hands-on Machine Learning

- SGD is much less regular than Batch GD. It also does not stop after reaching a minimum. Less likely to get stuck in a local minimum.
- To make SGD stop after reaching a minimum, we use a *learning schedule* i.e. decreasing the learning rate gradually.
- If n is the size of the training dataset, each round of n iterations of the SGD is called an *epoch*.
- In each epoch, some datapoints may not be chosen at all while others may be chosen several times.
- In Python, SGD is implemented as `sklearn.linear_model.SGDRegressor`

Regularization

- Regularization remedies overfitting and collinearity of features.

Regularization

- Regularization remedies overfitting and collinearity of features.
- Remember: if features are collinear then the design matrix $X^t X$ may not be invertible.

Regularization

- Regularization remedies overfitting and collinearity of features.
- Remember: if features are collinear then the design matrix X^tX may not be invertible.
- A complex model has more parameters and uses more features. If the training dataset is small, it is more likely that features are collinear.

Regularization

- Regularization remedies overfitting and collinearity of features.
- Remember: if features are collinear then the design matrix $X^t X$ may not be invertible.
- A complex model has more parameters and uses more features. If the training dataset is small, it is more likely that features are collinear.
- For example, polynomial regression of degree m, with d original features, has a total of $\binom{m+d}{m}$ features.

Regularization

- Regularization remedies overfitting and collinearity of features.
- Remember: if features are collinear then the design matrix X^tX may not be invertible.
- A complex model has more parameters and uses more features. If the training dataset is small, it is more likely that features are collinear.
- For example, polynomial regression of degree m, with d original features, has a total of $\binom{m+d}{m}$ features.
- **Tikhonov Regularization.** If A is a singular linear operator then $A + \alpha I$ is invertible for small α.

Regularization

- Regularization remedies overfitting and collinearity of features.
- Remember: if features are collinear then the design matrix X^tX may not be invertible.
- A complex model has more parameters and uses more features. If the training dataset is small, it is more likely that features are collinear.
- For example, polynomial regression of degree m, with d original features, has a total of $\binom{m+d}{m}$ features.
- **Tikhonov Regularization.** If A is a singular linear operator then $A + \alpha I$ is invertible for small α. This is because determinant function is continuous!

Regularization

- Regularization remedies overfitting and collinearity of features.
- Remember: if features are collinear then the design matrix X^tX may not be invertible.
- A complex model has more parameters and uses more features. If the training dataset is small, it is more likely that features are collinear.
- For example, polynomial regression of degree m, with d original features, has a total of $\binom{m+d}{m}$ features.
- **Tikhonov Regularization.** If A is a singular linear operator then $A + \alpha I$ is invertible for small α. This is because determinant function is continuous!
- **Ridge Regression**

$$\mathbf{a} = (X^tX + \alpha I)^{-1} X^t \mathbf{y} \tag{3}$$

α is a hyperparameter.

Regularization

- Regularization remedies overfitting and collinearity of features.
- Remember: if features are collinear then the design matrix X^tX may not be invertible.
- A complex model has more parameters and uses more features. If the training dataset is small, it is more likely that features are collinear.
- For example, polynomial regression of degree m, with d original features, has a total of $\binom{m+d}{m}$ features.
- **Tikhonov Regularization.** If A is a singular linear operator then $A + \alpha I$ is invertible for small α. This is because determinant function is continuous!
- **Ridge Regression**

$$\mathbf{a} = (X^tX + \alpha I)^{-1}X^t\mathbf{y} \tag{3}$$

  α is a hyperparameter.
- Ridge regression corresponds to the cost function

$$L_{Ridge}(\mathbf{a}) = \frac{1}{n}||\mathbf{a}^tX - y||^2 + \frac{1}{2}\alpha||\mathbf{a}||^2 \tag{4}$$

Ridge Regression

- This cost function penalizes against larger parameter values.

Ridge Regression

- This cost function penalizes against larger parameter values.
- By restricting the degrees of freedom in the model, it reduces the chance of overfitting.

Ridge Regression

- This cost function penalizes against larger parameter values.
- By restricting the degrees of freedom in the model, it reduces the chance of overfitting.
- Increasing α increases bias but decreases variance.

Ridge Regression

- This cost function penalizes against larger parameter values.
- By restricting the degrees of freedom in the model, it reduces the chance of overfitting.
- Increasing α increases bias but decreases variance.
- Gradient Descent for Ridge regression: the gradient of the cost function for Ridge Regression is given by adding $2\alpha\mathbf{a}$ to the gradient of $L(\mathbf{a})$.

Ridge Regression

- This cost function penalizes against larger parameter values.
- By restricting the degrees of freedom in the model, it reduces the chance of overfitting.
- Increasing α increases bias but decreases variance.
- Gradient Descent for Ridge regression: the gradient of the cost function for Ridge Regression is given by adding $2\alpha\mathbf{a}$ to the gradient of $L(\mathbf{a})$.

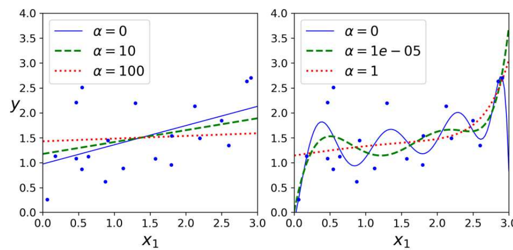


Figure: Credit: A. Geron, Hands-on Machine Learning

Ridge Regression

- This cost function penalizes against larger parameter values.
- By restricting the degrees of freedom in the model, it reduces the chance of overfitting.
- Increasing α increases bias but decreases variance.
- Gradient Descent for Ridge regression: the gradient of the cost function for Ridge Regression is given by adding $2\alpha\mathbf{a}$ to the gradient of $L(\mathbf{a})$.

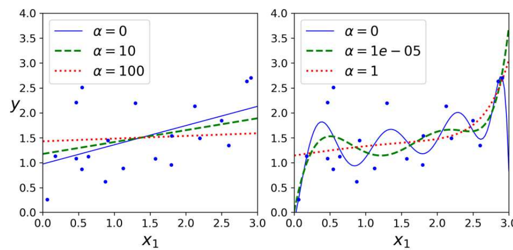


Figure: Credit: A. Geron, Hands-on Machine Learning

- In Python, the closed-form solution to Ridge regression is provided by `sklearn.linear_model.Ridge`

Ridge Regression

- This cost function penalizes against larger parameter values.
- By restricting the degrees of freedom in the model, it reduces the chance of overfitting.
- Increasing α increases bias but decreases variance.
- Gradient Descent for Ridge regression: the gradient of the cost function for Ridge Regression is given by adding $2\alpha\mathbf{a}$ to the gradient of $L(\mathbf{a})$.

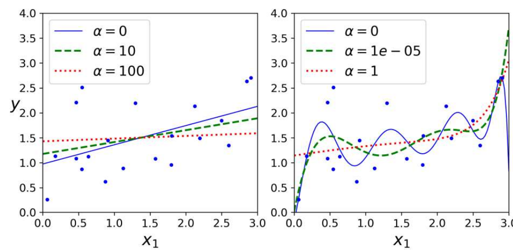


Figure: Credit: A. Geron, Hands-on Machine Learning

- In Python, the closed-form solution to Ridge regression is provided by `sklearn.linear_model.Ridge` and its SGD version is provided as `SGDRegressor(penalty="l2")`

Lasso Regression

- LASSO regression uses ℓ_1 norm of **a** instead of ℓ_2, for regularization:

$$L_{Lasso}(\mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{a}^t X - \mathbf{y})^2 + \alpha \sum_{j=1}^{d} |a_j|. \tag{5}$$

Lasso Regression

- LASSO regression uses ℓ_1 norm of **a** instead of ℓ_2, for regularization:

$$L_{Lasso}(\mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{a}^t X - \mathbf{y})^2 + \alpha \sum_{j=1}^{d} |a_j|. \tag{5}$$

- The penalty term in Lasso regression forces the coefficients a_i corresponding unnecessary/unimportant features to go to zero.

Lasso Regression

- LASSO regression uses ℓ_1 norm of **a** instead of ℓ_2, for regularization:

$$L_{Lasso}(\mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{a}^t X - \mathbf{y})^2 + \alpha \sum_{j=1}^{d} |a_j|. \qquad (5)$$

- The penalty term in Lasso regression forces the coefficients a_i corresponding unnecessary/unimportant features to go to zero. Thus, it does automatic *feature selection* for us!

Lasso Regression

- LASSO regression uses ℓ_1 norm of **a** instead of ℓ_2, for regularization:

$$L_{Lasso}(\mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{a}^t X - \mathbf{y})^2 + \alpha \sum_{j=1}^{d} |a_j|. \tag{5}$$

- The penalty term in Lasso regression forces the coefficients a_i corresponding unnecessary/unimportant features to go to zero. Thus, it does automatic *feature selection* for us!
- Lasso regression gives us a *sparse model* meaning a model in which there are a few nonzero weights.

Lasso Regression

- LASSO regression uses ℓ_1 norm of **a** instead of ℓ_2, for regularization:

$$L_{Lasso}(\mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{a}^t X - \mathbf{y})^2 + \alpha \sum_{j=1}^{d} |a_j|. \quad (5)$$

- The penalty term in Lasso regression forces the coefficients a_i corresponding unnecessary/unimportant features to go to zero. Thus, it does automatic *feature selection* for us!
- Lasso regression gives us a *sparse model* meaning a model in which there are a few nonzero weights.
- Sparse models are more explainable because the relation between the features and the target variable is easier to see.

Gradient Descent for Lasso Regression

- The Lasso cost function is not differentiable when $a_i = 0$ for some i.

Gradient Descent for Lasso Regression

- The Lasso cost function is not differentiable when $a_i = 0$ for some i. But it has a *subgradient*.

Gradient Descent for Lasso Regression

- The Lasso cost function is not differentiable when $a_i = 0$ for some i. But it has a *subgradient*. A **subgradient** for a convex function L at a point $\mathbf{a}$ is a vector $\mathbf{v}$ such that

$$L(\mathbf{b}) - L(\mathbf{a}) \geq \mathbf{v}^t(\mathbf{b} - \mathbf{a}) \tag{6}$$

Gradient Descent for Lasso Regression

- The Lasso cost function is not differentiable when $a_i = 0$ for some i. But it has a *subgradient*. A **subgradient** for a convex function L at a point $\mathbf{a}$ is a vector $\mathbf{v}$ such that

$$L(\mathbf{b}) - L(\mathbf{a}) \geq \mathbf{v}^t(\mathbf{b} - \mathbf{a}) \tag{6}$$

- For the ℓ_1 norm, the subgradient at nondifferentiable points can be given by $(sign(a_1), sign(a_2), \ldots, sign(a_d))$.

Gradient Descent for Lasso Regression

- The Lasso cost function is not differentiable when $a_i = 0$ for some i. But it has a *subgradient*. A **subgradient** for a convex function L at a point $\mathbf{a}$ is a vector $\mathbf{v}$ such that

$$L(\mathbf{b}) - L(\mathbf{a}) \geq \mathbf{v}^t(\mathbf{b} - \mathbf{a}) \tag{6}$$

- For the ℓ_1 norm, the subgradient at nondifferentiable points can be given by $(sign(a_1), sign(a_2), \ldots, sign(a_d))$.
- In Scikit-Learn, Lasso regression is provided by `SGDRegressor(penalty="l1")`

Elastic Net Regularization

- It is preferable to regularize regression models such as Ridge Regression.

Elastic Net Regularization

- It is preferable to regularize regression models such as Ridge Regression.
- If you it seems that only a few of the features are important, use Lasso or Elastic Net.

Elastic Net Regularization

- It is preferable to regularize regression models such as Ridge Regression.
- If you it seems that only a few of the features are important, use Lasso or Elastic Net.
- Lasso's behavior may be erratic if we have more features than training instances.

Elastic Net Regularization

- It is preferable to regularize regression models such as Ridge Regression.
- If you it seems that only a few of the features are important, use Lasso or Elastic Net.
- Lasso's behavior may be erratic if we have more features than training instances.
- Elastic Net regularization uses a linear combination of Ridge and LASSO penalty terms:

$$\alpha \cdot ||\mathbf{a}||_{\ell_1} + \frac{1-r}{2}\alpha \cdot ||\mathbf{a}||_{\ell_2}^2 \tag{7}$$

where $0 \leq r \leq 1$