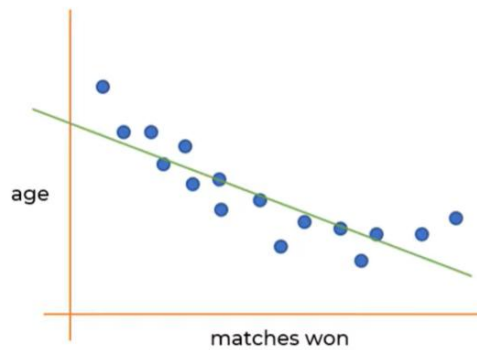


Ridge and Lasso Regression

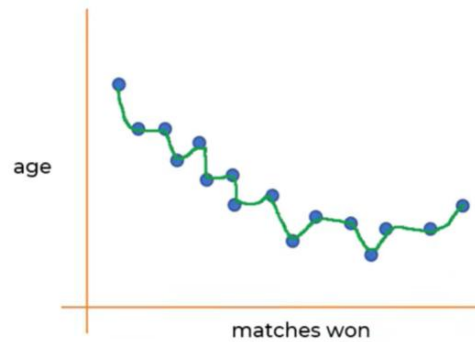
Dr. Dipak Kumar Mohanty
KIIT University, Bhubaneswr

underfit



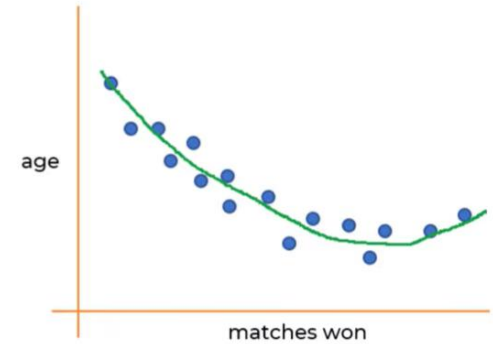
$$\text{match won} = \theta_0 + \theta_1 * \text{age}$$

overfit



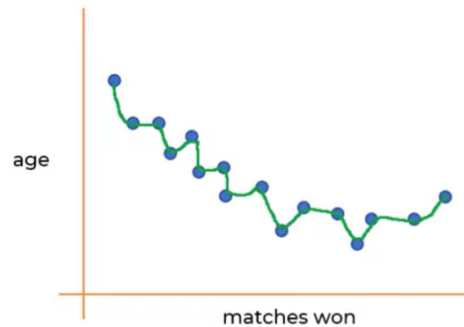
$$\text{match won} = \theta_0 + \theta_1 * \text{age} + \theta_2 * \text{age}^2 + \theta_3 * \text{age}^3 + \theta_4 * \text{age}^4$$

balanced fit



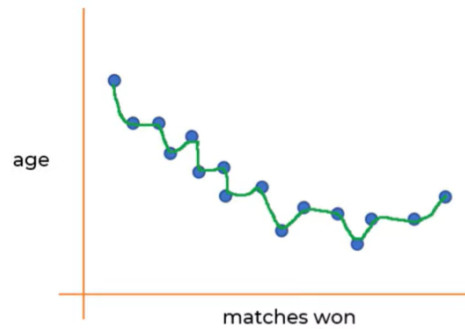
$$\text{match won} = \theta_0 + \theta_1 * \text{age} + \theta_2 * \text{age}^2$$

How to reduce overfitting?



$$\text{match won} = \theta_0 + \theta_1 * \text{age} + \theta_2 * \text{age}^2 + \theta_3 * \text{age}^3 + \theta_4 * \text{age}^4$$

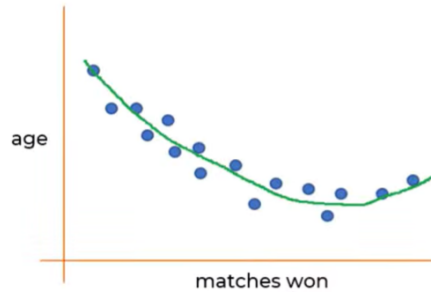
4



$$\text{match won} = \theta_0 + \theta_1 * \text{age} + \theta_2 * \text{age}^2 + \theta_3 * \text{age}^3 + \theta_4 * \text{age}^4$$



Try to make θ_3 and θ_4 almost close to zero



$$\text{match won} = \theta_0 + \theta_1 * \text{age} + \theta_2 * \text{age}^2$$



From Linear Regression we know, MSE...

Mean Squared Error

$$mse = \frac{1}{n} \sum_{i=1}^n (y_i - y_{predicted})^2$$

Mean Squared Error

$$mse = \frac{1}{n} \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2$$

$$h_{\theta}(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

Regularization

- Regularizations are of two types
 - 1) Ridge Regression or L2- Regularization
 - 2) Lasso Regression or L1- Regularization

1) Ridge Regression or L2-Regularization

Ridge Regression

$$R = \text{Loss} + \lambda \|\theta\|_2^2$$

where λ = penalty,

$$\text{where } \|\theta\|_2^2 = \theta_1^2 + \theta_2^2 + \dots + \theta_n^2$$

Here, When θ is bigger, Error become bigger, so model will not converge

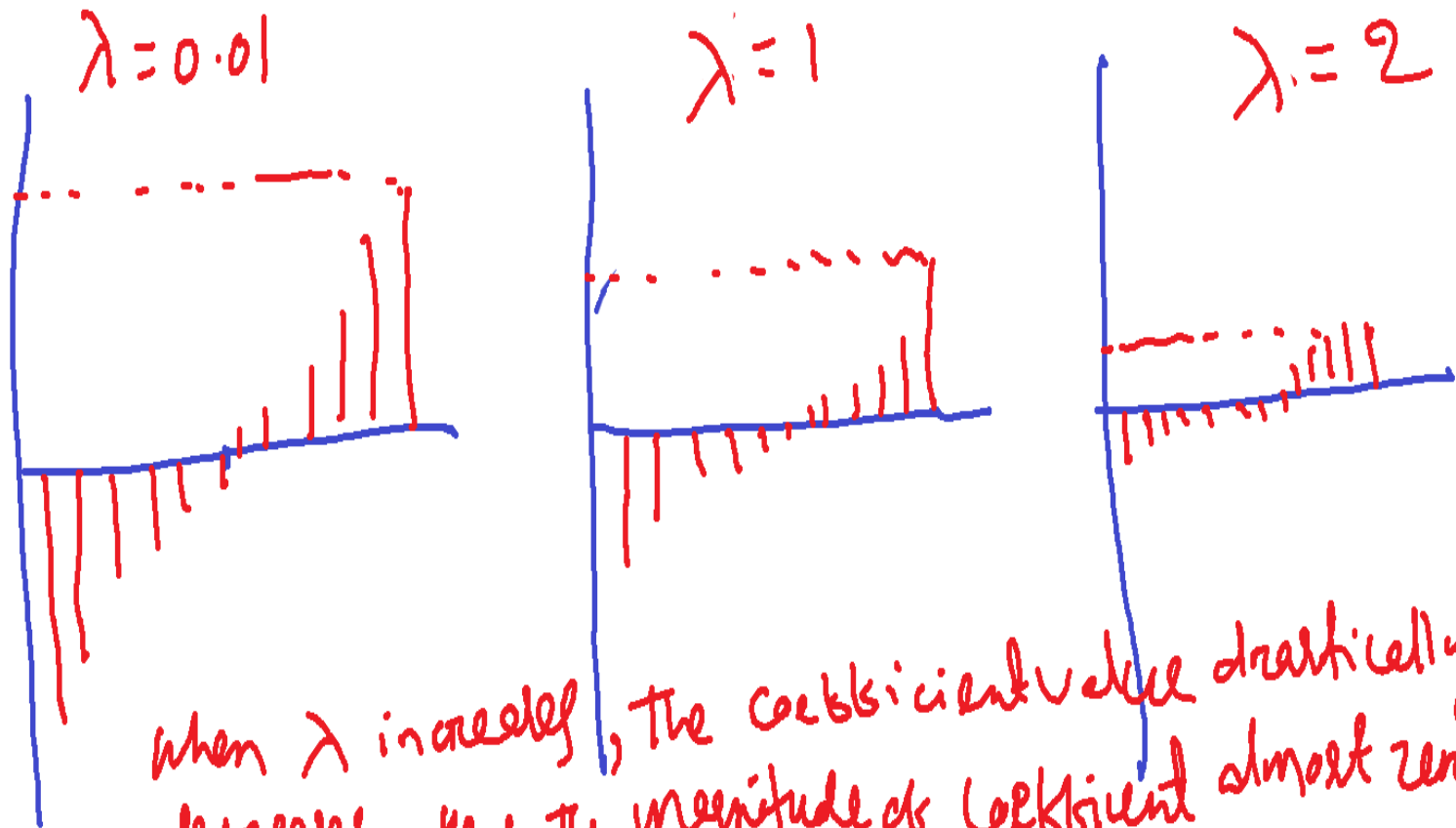
L2 Regularization

$$mse = \frac{1}{n} \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$

$$h_{\theta}(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

- Here, Penalty Lambda is used to ensure that θ value doesn't go too high.
- When Lambda is bigger, θ is smaller and vice versa.
- This approach is called **L2-Regularization**

Ridge: Example



when λ increases, the coefficient value drastically decreases, here the magnitude of coefficient almost zero.

L1-Regularization or Lasso Regression

$$\text{Lasso } R = \text{Loss} + \lambda \|\theta\|_1$$

$$\|\theta\|_1 = |\theta_1| + |\theta_2| + \dots + |\theta_n|$$

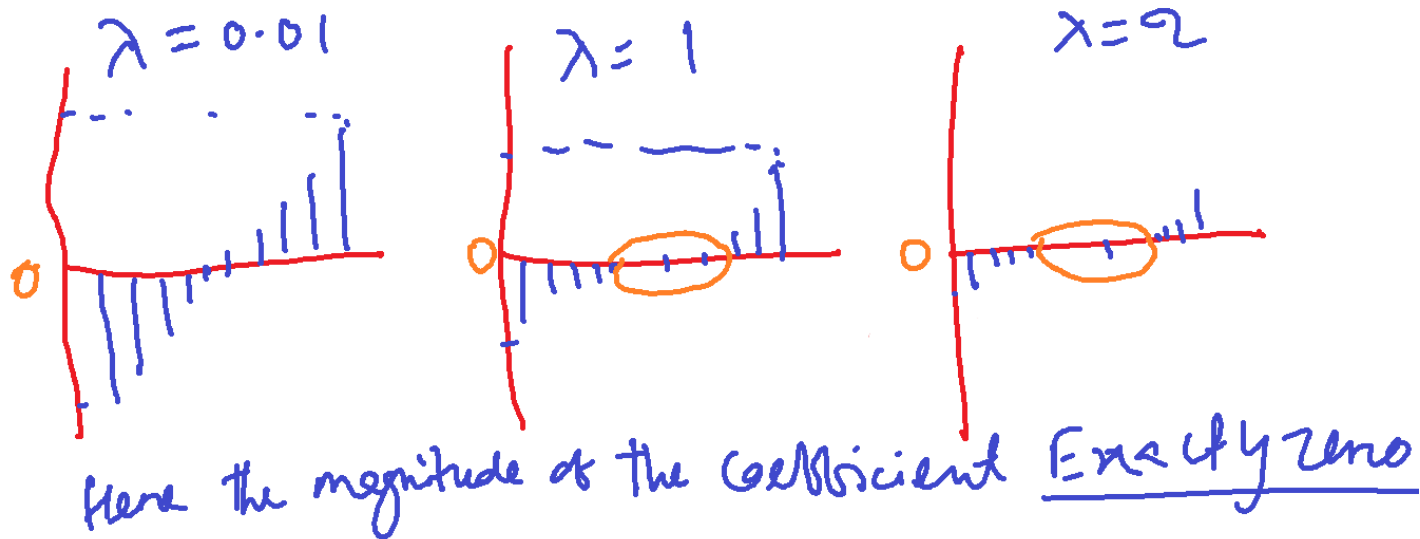
L1-Regularization

L1 Regularization

$$mse = \frac{1}{n} \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{i=1}^n |\theta_i|$$

$$h_{\theta}(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

Lasso Regression: Example



Example

$$\text{Let } y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

$$y = 0.8 + 1.2x_1 + 30x_2 + 49x_3$$

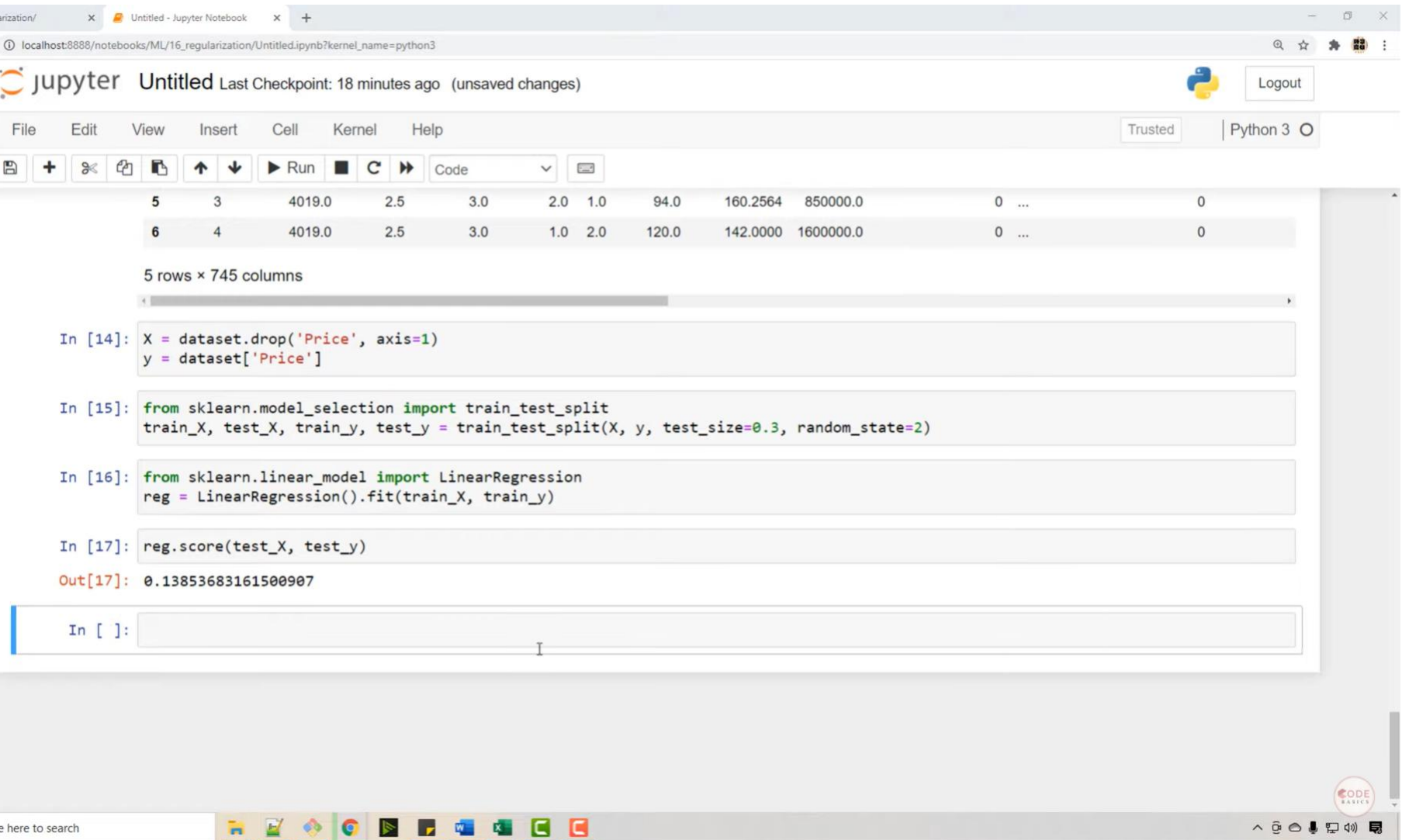
After Ridge ↑

$$y = 0.8 + 1.2x_1 + 0.2x_2 + 0.3x_3$$

After Lasso

$$y = 0.8 + 1.2x_1 + 0.2x_2 + 0 \cdot x_3$$

reg: Regular Linear Regression



The screenshot displays a Jupyter Notebook environment. At the top, the browser address bar shows the local host URL. The notebook's title bar indicates it is 'Untitled' and shows the last checkpoint was 18 minutes ago. The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. A toolbar with icons for file operations and execution is visible. The main area shows a dataset preview with 5 rows and 745 columns. The first two rows are highlighted. Below the preview, four code cells are shown, demonstrating the process of dropping the 'Price' column, splitting the data into training and testing sets, fitting a LinearRegression model, and finally calculating the score on the test set. The output of the last cell is 0.13853683161500907.

	5	3	4019.0	2.5	3.0	2.0	1.0	94.0	160.2564	850000.0	0	...	0
	6	4	4019.0	2.5	3.0	1.0	2.0	120.0	142.0000	1600000.0	0	...	0

5 rows × 745 columns

```
In [14]: X = dataset.drop('Price', axis=1)
y = dataset['Price']

In [15]: from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.3, random_state=2)

In [16]: from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(train_X, train_y)

In [17]: reg.score(test_X, test_y)

Out[17]: 0.13853683161500907

In [ ]:
```


Overfitting case: train accu=0.68, test accu= 0.13,

The screenshot shows a Jupyter Notebook interface with the following components:

- Browser Tab:** ML16_regularization/
- Page Title:** Untitled - Jupyter Notebook
- Address Bar:** localhost:8888/notebooks/ML16_regularization/Untitled.ipynb?kernel_name=python3
- Page Header:** jupyter Untitled Last Checkpoint: 19 minutes ago (unsaved changes)
- Page Actions:** Logout
- Page Info:** Trusted Python 3
- Menu Bar:** File Edit View Insert Cell Kernel Help
- Toolbar:** Includes icons for saving, opening, and running code.
- Data View:** A table with 5 rows and 745 columns. The first two rows are visible:

5	3	4019.0	2.5	3.0	2.0	1.0	94.0	160.2564	850000.0	0	...	0	
6	4	4019.0	2.5	3.0	1.0	2.0	120.0	142.0000	1600000.0	0	...	0	
- Code Cells:**
 - In [14]: `X = dataset.drop('Price', axis=1)`
`y = dataset['Price']`
 - In [15]: `from sklearn.model_selection import train_test_split`
`train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.3, random_state=2)`
 - In [16]: `from sklearn.linear_model import LinearRegression`
`reg = LinearRegression().fit(train_X, train_y)`
 - In [17]: `reg.score(test_X, test_y)`
Out[17]: 0.13853683161500907
 - In [18]: `reg.score(train_X, train_y)`
Out[18]: 0.6827792395792723

Lasso: L1 regularization

[Install](#) [User Guide](#) [API](#) [Examples](#) [More ▾](#)

[Prev](#) [Up](#) [Next](#)

scikit-learn 0.23.2
[Other versions](#)

Please [cite us](#) if you use the software.

[sklearn.linear_model.Lasso](#)
Examples using
[sklearn.linear_model.Lasso](#)

sklearn.linear_model.Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')
```

Linear Model trained with **L1 prior as regularizer** (aka the Lasso)

The optimization objective for Lasso is:

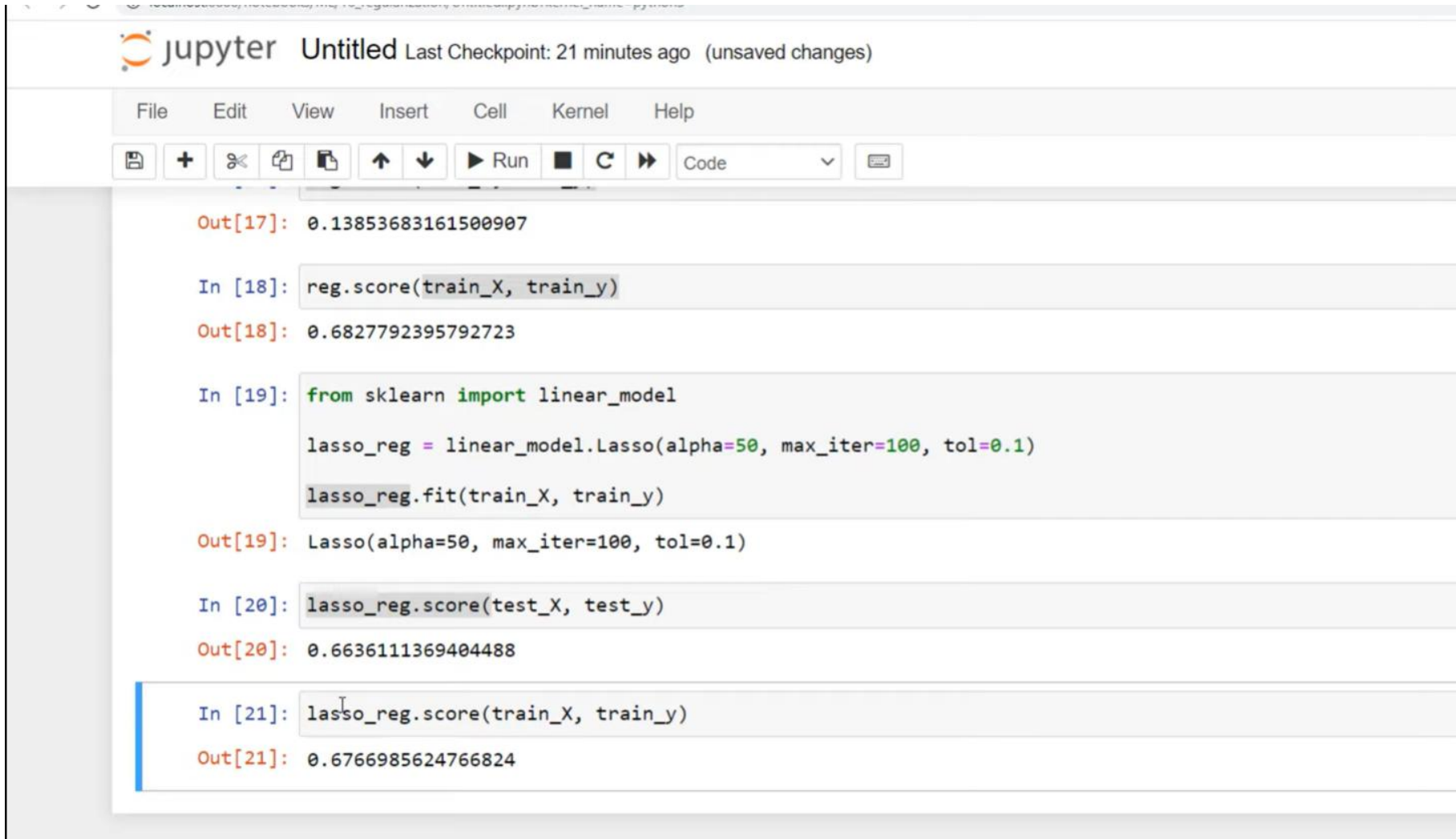
$$\left(\frac{1}{2} \cdot \frac{1}{n_{\text{samples}}} \right) * ||y - Xw||^2_2 + \alpha * ||w||_1$$

Technically the Lasso model is optimizing the same objective function as the Elastic Net with **l1_ratio=1**. (no L2 penalty).

Read more in the [User Guide](#).

Parameters:	alpha : float, default=1.0 Constant that multiplies the L1 term. Defaults to 1.0. alpha = 0 is equivalent to an ordinary least square, solved by the LinearRegression object. For numerical reasons,
--------------------	---

Result in Lasso: Good with 66%,67% accuracy



The image shows a Jupyter Notebook interface with the title "Untitled" and a status bar indicating "Last Checkpoint: 21 minutes ago (unsaved changes)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook contains five code cells. The first cell shows the output of a previous cell: "Out[17]: 0.13853683161500907". The second cell contains the code "reg.score(train_X, train_y)" and its output "Out[18]: 0.6827792395792723". The third cell contains the code to import "linear_model" from "sklearn" and create a "Lasso" object with parameters "alpha=50", "max_iter=100", and "tol=0.1". It also shows the "fit" method being called on "train_X" and "train_y". The output of this cell is "Out[19]: Lasso(alpha=50, max_iter=100, tol=0.1)". The fourth cell contains the code "lasso_reg.score(test_X, test_y)" and its output "Out[20]: 0.6636111369404488". The fifth cell contains the code "lasso_reg.score(train_X, train_y)" and its output "Out[21]: 0.6766985624766824".

```
jupyter Untitled Last Checkpoint: 21 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Out[17]: 0.13853683161500907

In [18]: reg.score(train_X, train_y)
Out[18]: 0.6827792395792723

In [19]: from sklearn import linear_model
         lasso_reg = linear_model.Lasso(alpha=50, max_iter=100, tol=0.1)
         lasso_reg.fit(train_X, train_y)
Out[19]: Lasso(alpha=50, max_iter=100, tol=0.1)

In [20]: lasso_reg.score(test_X, test_y)
Out[20]: 0.6636111369404488

In [21]: lasso_reg.score(train_X, train_y)
Out[21]: 0.6766985624766824
```

Ridge Regression: L2 Regularization

The screenshot shows the scikit-learn documentation page for the `sklearn.linear_model.Ridge` class. The page layout includes a top navigation bar with links for 'Install', 'User Guide', 'API', 'Examples', and 'More'. A left sidebar contains navigation links for 'Prev', 'Up', and 'Next', along with version information for 'scikit-learn 0.23.2' and a 'Toggle Menu' button at the bottom. The main content area features a large blue header for the class name, followed by the class signature in code. Below this, a description states it is 'Linear least squares with l2 regularization' and 'Minimizes the objective function:'. The objective function is displayed in a code block as
$$\|y - Xw\|^2_2 + \alpha * \|w\|^2_2$$
. A paragraph explains that this model solves a regression problem using the linear least squares function with l2-norm regularization, also known as Ridge Regression or Tikhonov regularization. It notes that the estimator supports multi-variate regression. A 'Parameters' section is partially visible at the bottom, detailing the `alpha` parameter.

scikit-learn

Install User Guide API Examples More ▾

Prev Up Next

scikit-learn 0.23.2
Other versions

Please [cite us](#) if you use the software.

`sklearn.linear_model.Ridge`
Examples using
`sklearn.linear_model.Ridge`

`sklearn.linear_model.Ridge`

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

[source]

Linear least squares with l2 regularization.

Minimizes the objective function:

$$\|y - Xw\|^2_2 + \alpha * \|w\|^2_2$$

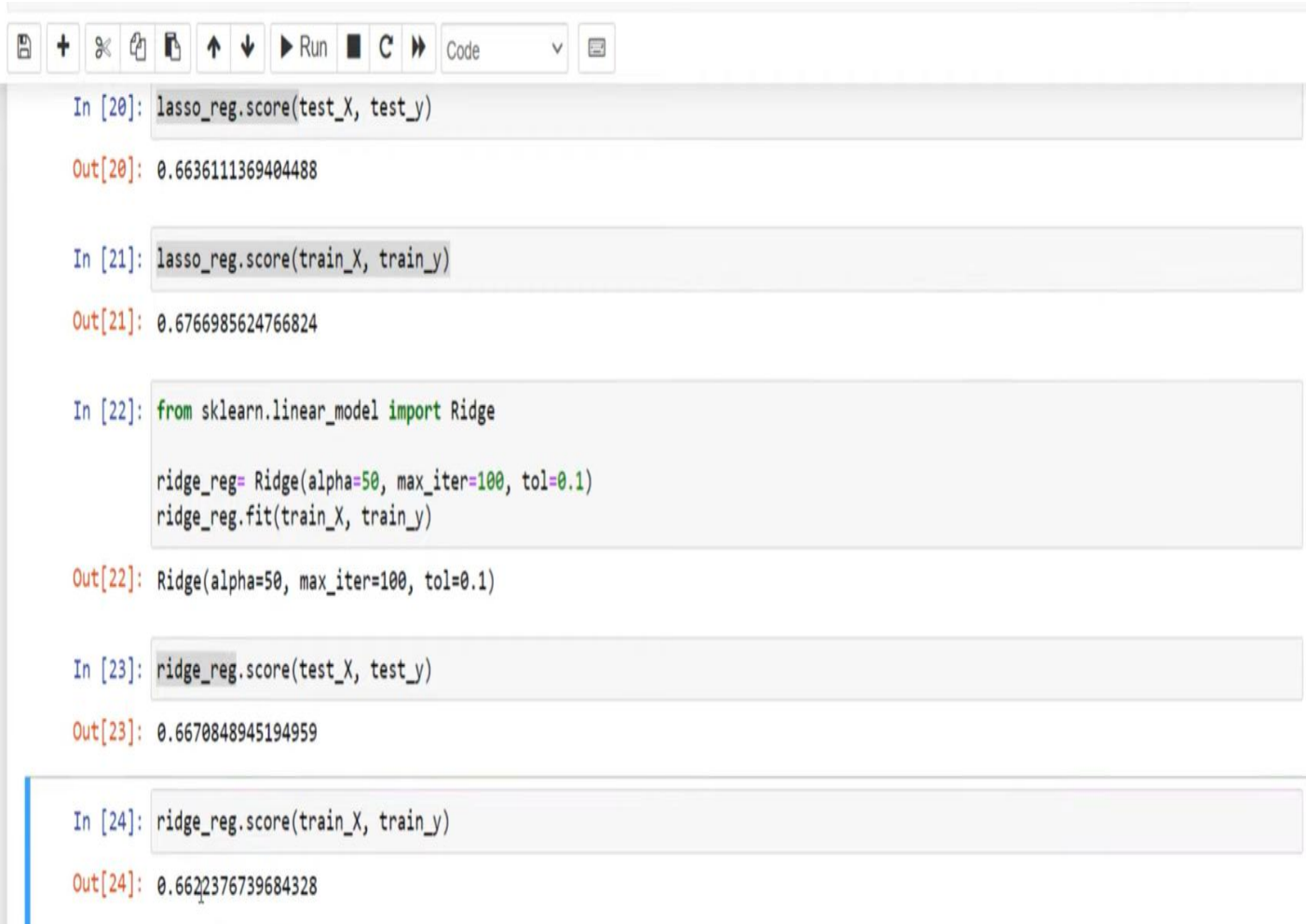
This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when `y` is a 2d-array of shape `(n_samples, n_targets)`).

Read more in the [User Guide](#).

Parameters:

`alpha` : `{float, ndarray of shape (n_targets,,), default=1.0}`
Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values

Result in Ridge: Good with 66%,66% accuracy



The image shows a Jupyter Notebook interface with a toolbar at the top containing icons for file operations, execution, and code management. The notebook contains four input-output pairs. The first two pairs use `lasso_reg.score()` to calculate accuracy on test and training data. The third pair imports `Ridge` from `sklearn.linear_model`, creates a `Ridge` object with `alpha=50`, `max_iter=100`, and `tol=0.1`, and fits it to the training data. The fourth pair uses `ridge_reg.score()` to calculate accuracy on test and training data.

```
In [20]: lasso_reg.score(test_X, test_y)
Out[20]: 0.6636111369404488

In [21]: lasso_reg.score(train_X, train_y)
Out[21]: 0.6766985624766824

In [22]: from sklearn.linear_model import Ridge

        ridge_reg= Ridge(alpha=50, max_iter=100, tol=0.1)
        ridge_reg.fit(train_X, train_y)

Out[22]: Ridge(alpha=50, max_iter=100, tol=0.1)

In [23]: ridge_reg.score(test_X, test_y)
Out[23]: 0.6670848945194959

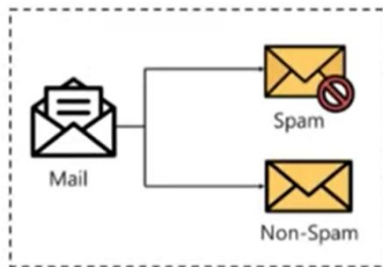
In [24]: ridge_reg.score(train_X, train_y)
Out[24]: 0.6622376739684328
```

Recent Reference Papers

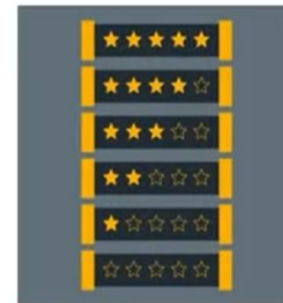
- Looking for interesting machine learning papers to read for the break or the new year? Here is a nicely curated list by Louis-François Bouchard.
- https://github.com/louisfb01/Best_AI_paper_2020
-

Thank You

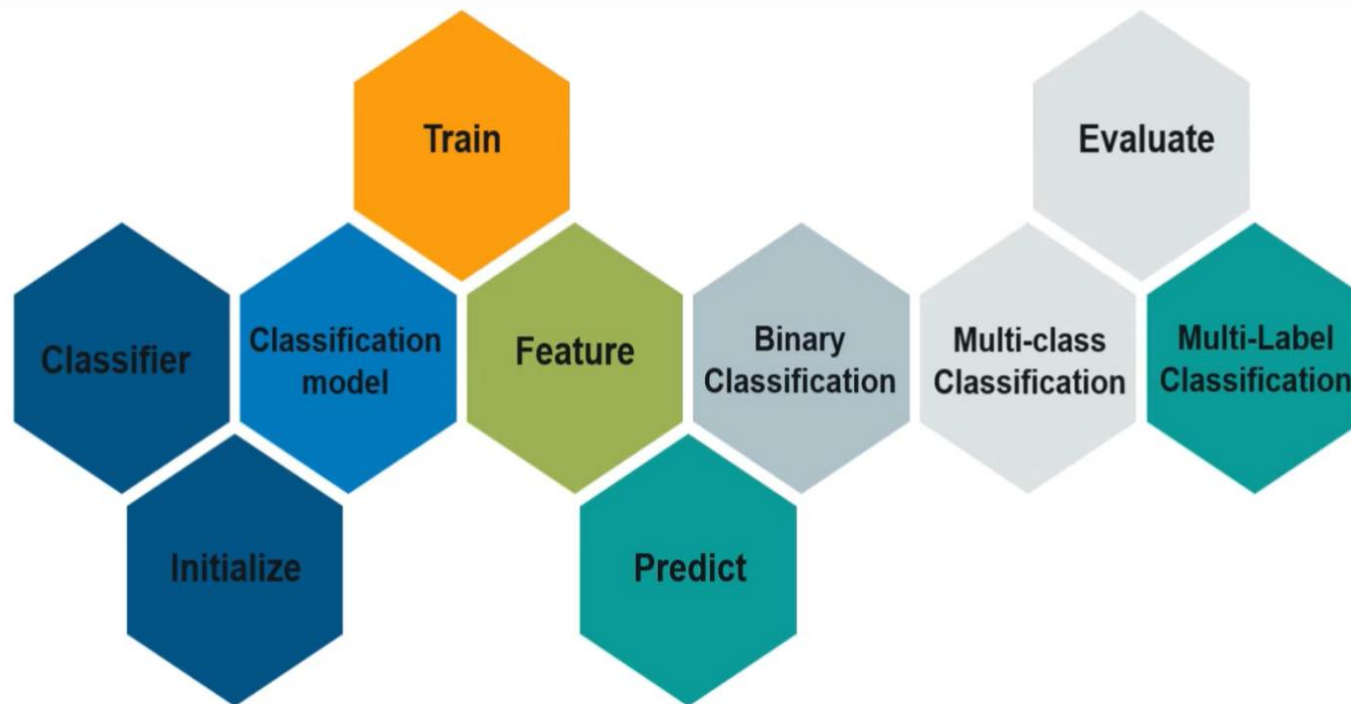
What is Classification In Machine Learning?




Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.



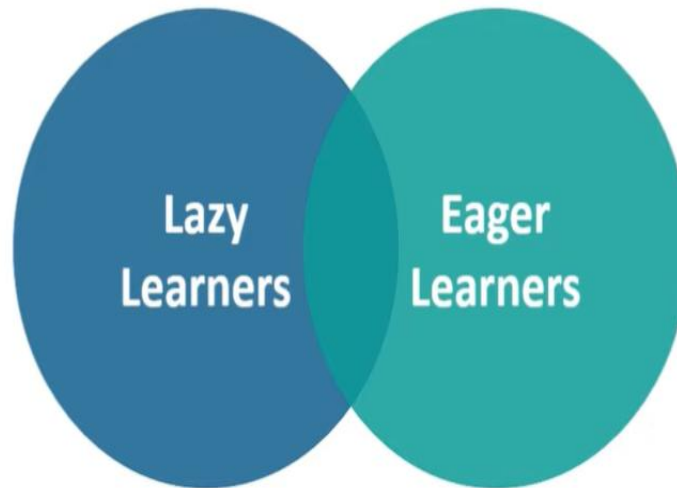
Classification Terminologies




Types Of Learners In Classification



Lazy learners simply store the training data and wait until a testing data appears.



Eager Learners



Eager learners construct a classification model based on the given training data before getting data for predictions