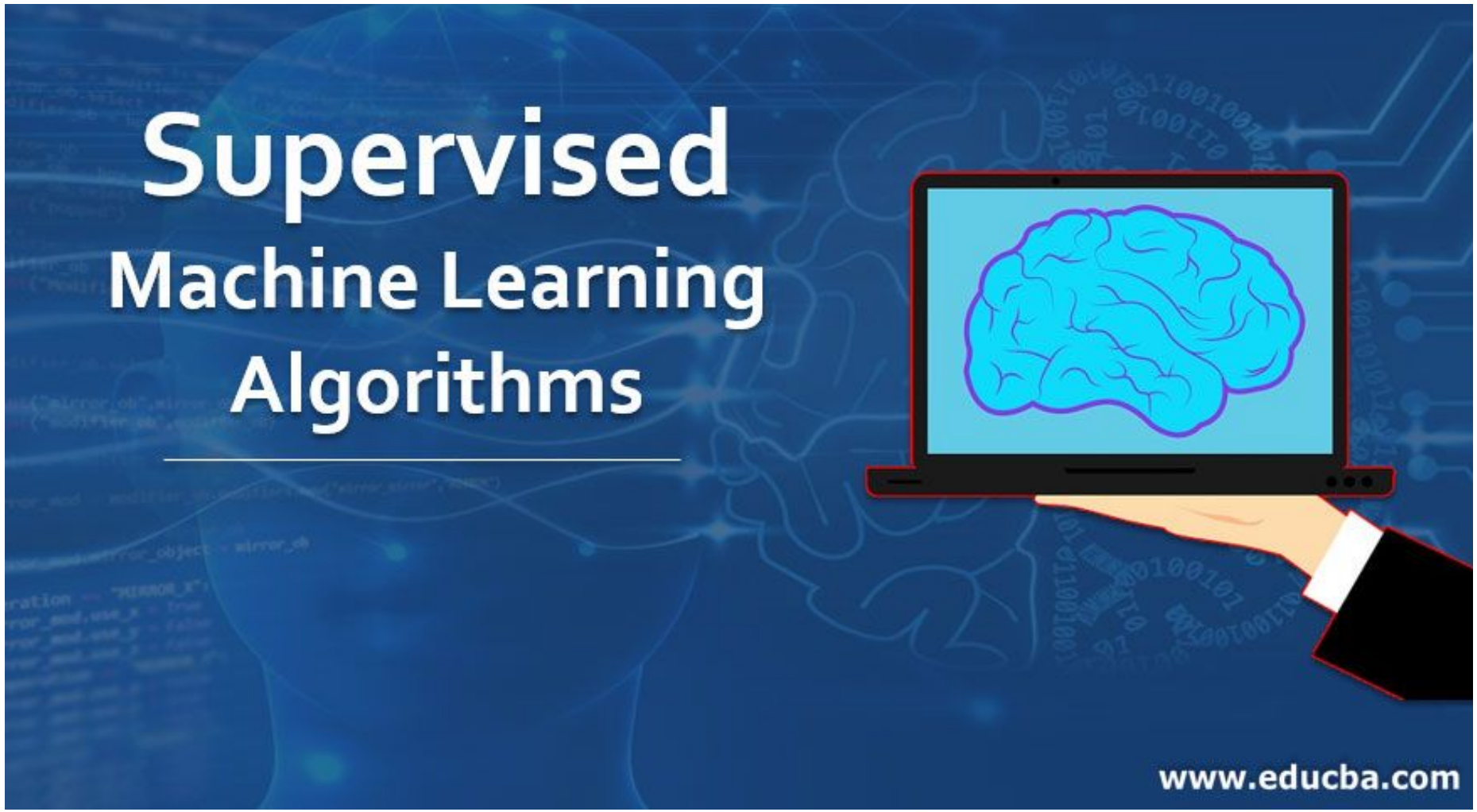

Supervised
Machine Learning
Algorithms
www.educba.com

Last lecture reminder

We learned about:

- Introduction to Feature Scaling
- Feature scaling methods - Standardization and Normalization
- Introduction to Regularization
- L2 Regularization - Ridge Regression
- Ridge Regression - Python example
- Ridge Regression parameters - Lambda value and scoring metric
- RidgeCV - Python example

L1 Regularization - Lasso Regression

L1 Regularization (also known as Lasso regularization) → This regression type adds a term equal to the absolute value of the magnitude of coefficients to the loss function. L1 can result in sparse models where some feature coefficients can become zero, effectively excluding those features from the model.

The name Lasso regression comes from **least absolute shrinkage and selection operator (Lasso)**.

The penalty term in Lasso regression is the absolute value of the magnitude of coefficients.
The lambda term determine the severity of the palenty.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|$$

Lasso Regression vs Ridge Regression

The difference between L1 and L2 regularization:

The key difference is that while L1 regularization can result in zero coefficients, effectively performing feature selection, L2 regularization instead keeps all of the features but might limit their contribution by reducing the magnitude of their coefficients.

When should we use L1 and L2:

L1 Regularization (Lasso) → Typically used when we have a lot of features and we believe that many of them are irrelevant or redundant. With its ability to shrink coefficients to zero and effectively perform feature selection, it's good for feature selection when you have a high number of features and you expect only a few of them to be significant.

Lasso Regression vs Ridge Regression

L2 Regularization (Ridge) → Typically used when we think all features should be included in the model because they are all relevant or when we want to avoid overfitting. It can't shrink coefficients to exactly zero, but rather, it decreases all of them equally and proportionally, thus prevents any single feature from dominating the model.

Finally, if you aren't sure, many practitioners will start with Ridge regularization as a default, and then use Lasso regularization (or Elastic Net, which is a combination of the two) as needed.

Lasso Regression - Python Example

Let’s now see a real Python example appling LassoCV regression model.

Same as we have with Ridge regression Scikit-Learn provide regular Lasso regression model and the cross-validation Lasso regression model which help us find the optimized alpha to use.

For this example we will use the **‘Advertising.csv’** file, applying polynomial regression to increase the model complexity and apply feature scaling to deal with Lasso regression (When applying any type of regularization we have to perform feature scaling).

In [84]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

df = pd.read_csv('/Users/ben.meir/Downloads/Advertising (1).csv')
X = df.drop('sales', axis=1)
y = df['sales']

polynomial_converter = PolynomialFeatures(degree=3, include_bias=False)
polynomial_features = polynomial_converter.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(polynomial_features, y, test_size=0.3, random_state=101)
```

Lasso Regression - Python Example

Now let's apply feature scaling same as we did in the Ridge regression example.

Remember to fit only on the train data and not exposed it to the test data.

```
In [86]: from sklearn.preprocessing import StandardScaler

         scaler = StandardScaler()
         scaler.fit(X_train)
         scaled_X_train = scaler.transform(X_train)
         scaled_X_test = scaler.transform(X_test)
```

Standardization feature scaling

Now let's perform LassoCV regression and see the error metrics it provides:

```
In [125]: from sklearn.linear_model import LassoCV
          lasso_cv_model = LassoCV(eps=0.1, n_alphas=100)
          lasso_cv_model.fit(scaled_X_train, y_train)

          test_predictions = lasso_cv_model.predict(scaled_X_test)
```

We apply Lasso regression modeling on our data and provide the additional parameters to find the optimal alpha value

```
In [117]: from sklearn.metrics import mean_absolute_error, mean_squared_error
          MAE = mean_absolute_error(y_test, test_predictions)
          MSE = mean_squared_error(y_test, test_predictions)
          RMSE = np.sqrt(MSE)
          print(f'MAE: {MAE}')
          print(f'MSE: {MSE}')
          print(f'RMSE: {RMSE}')

          MAE: 0.6541723161252856
          MSE: 1.2787088713079862
          RMSE: 1.130800102276254
```

LassoCV model error metrics values

LassoCV Regression - Optimal Lambda Value

Let’s understand how LassoCV find the optimal alpha (equal to lambda in our formula) value by providing the ‘eps’, ‘n_alphas’ and ‘cv’ parameters.

As we discussed the main difference between Lasso regression and Ridge regression is that Lasso can eliminate features from our model by making their coefficient to 0.
The alpha (or lambda) value helps determine the severity of the penalty. A larger alpha means a more severe penalty, leading to smaller coefficient estimates. The maximum alpha which makes all feature coefficients zero is computed automatically by the LassoCV model.

The next step is to choose the minimum alpha value, indicated by the `eps` parameter. The chosen `eps` value multiplies the maximum alpha to get the minimum alpha.

```
`min_alpha = eps * max_alpha`
```

LassoCV Regression - Optimal Lambda Value

Finally, the model uses cross-validation to find the optimal alpha value from the range of possible alpha values between this minimum and maximum. The number of different alpha values that will be generated in this range is controlled by the `n_alphas` parameter.

So in calculation:

`eps` parameter → Determines the lower bound for alpha values (smallest penalty strength) using the formula: `min_alpha = eps * max_alpha`

`n_alphas` parameter → Determines how many different alpha values will be used between this minimum and the automatically calculated maximum alpha values.

Class Exercise - Lasso Regression

Instructions:

Use the data set provided in the previous class exercise and use **polynomial regression & Lasso** model training.

Hours of calling customers: [2, 3, 4, 5, 6, 1.5, 5, 7, 8, 10]

Money earned: [50K$, 70K$, 90K$, 100K$, 110K$, 40K$, 110K$, 130K$, 145K$, 180K$]

- Convert your model data to be with polynomial regression of 3 degree
- Use feature scaling of type Standardization to prepare your data set
- Use basic LassoCV model to eliminate overfitting, for the parameters use 2 different values for `eps` and `n_alphas`
- Print the MAE and RMSE results for each LassoCV model
- Print the beta coefficients you got for each model, see if some coefficients got 0 value
- Print the different optimal alpha values for for each LassoCV model
- Compare the result of the different LassoCV models and select the more accurate one

Class Exercise Solution - Lasso Regression

Cross-Validation - Introduction

Until now we saw that Scikit-Learn provide us cross validation build-in model such as RidgeCV and LassoCV but cross validation is a very important technique in supervised learning that we can also implement manually when needed.

Cross validation → A technique in supervised learning used to assess how well the model will generalize to an independent data set. It helps in understanding the model's effectiveness and to minimize overfitting.

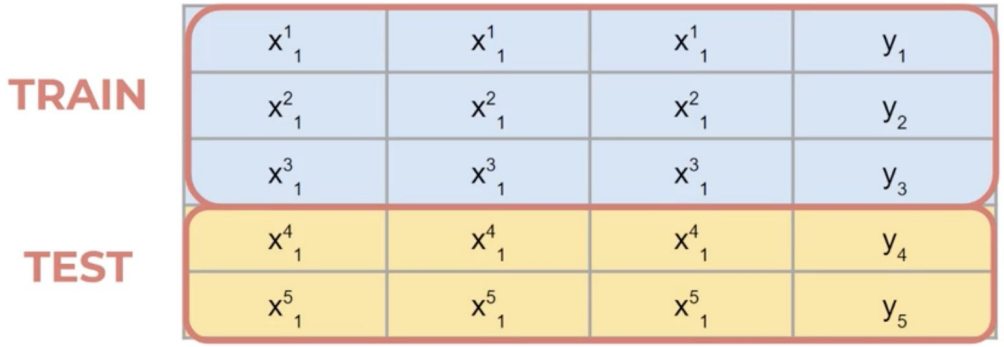
The process of cross-validation involves partitioning the data set into two subsets multiple times. Training occurs on one subset, known as the training set, and the model is tested on the other subset, known as the validation set or testing set.

Cross-Validation - Introduction

K-fold cross-validation → The most common method of cross-validation, known as k-fold cross-validation, involves splitting the data into 'k' folds or groups. The model is trained on (k-1) folds and validated on the remaining fold. This process is repeated k times with each fold serving as the validation set once. The average error across all k trials is computed to get the final result.

By using this technique, every data point gets to be in the validation set at least once which helps in making the model validation more robust and less prone to errors.

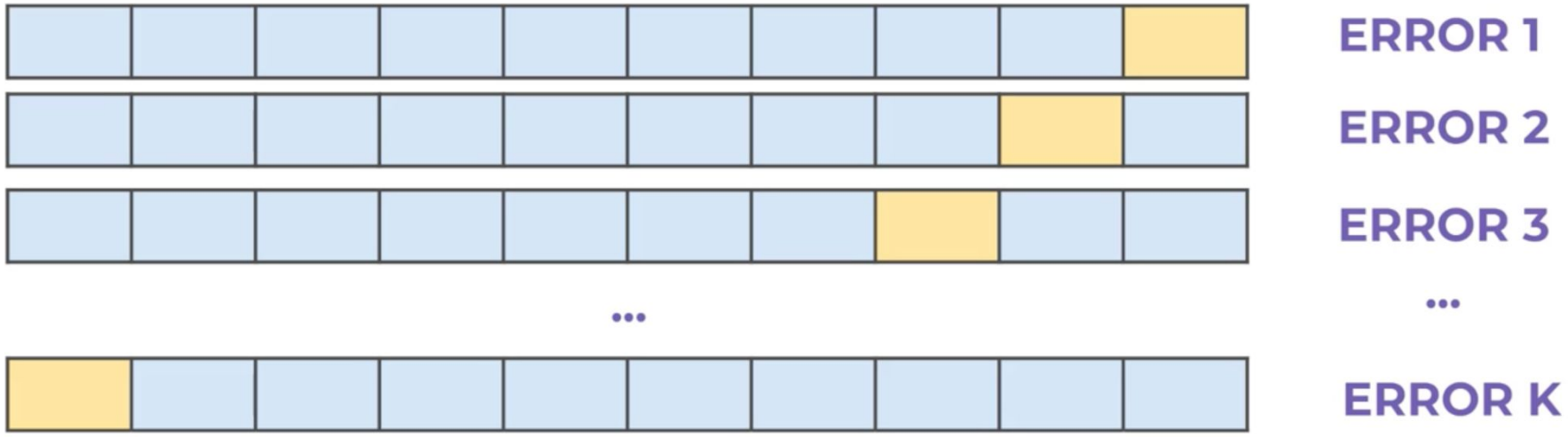
To better visualize how cross validation works let's say we have the following dataset:



Cross-Validation - Introduction

In regular train / test split we split the dataset into training set and test set and perform model fit on the train set and predict the test set.

However, in cross validation we will perform this process k times, each time we will take different part of the dataset as the train set and as the test set.

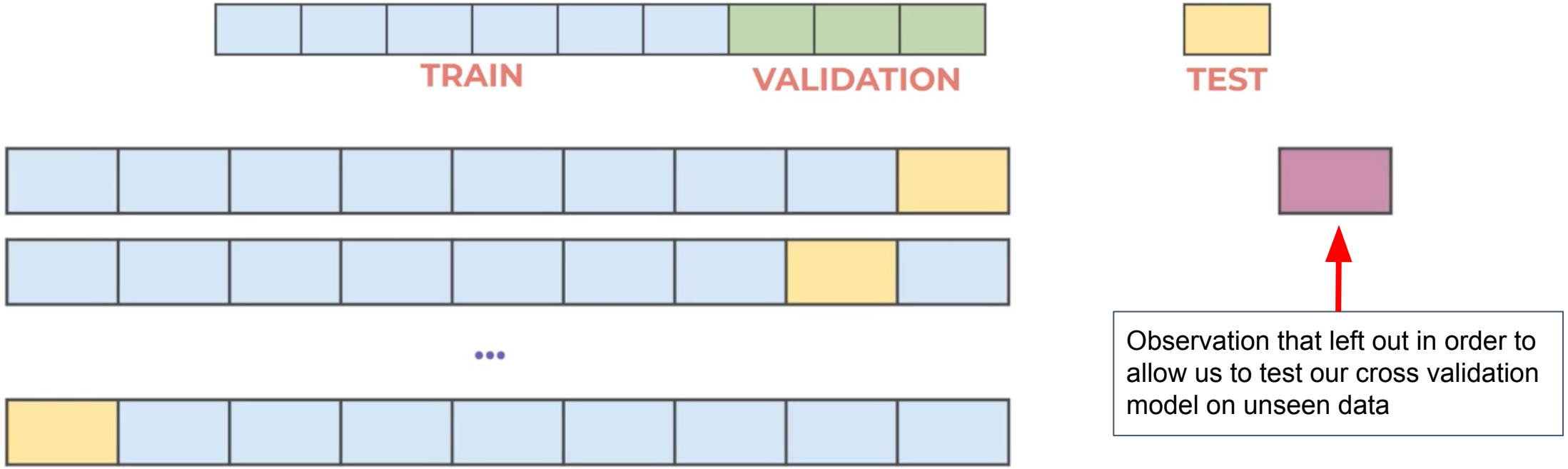


For each iteration we will get corresponding error value. The average error across all k trials is computed to get the final result.

Cross-Validation - Introduction

Leave-one-out cross-validation (LOOCV) → A particular case of k-fold cross-validation where k is set to the total number of observations in the data set.

In LOOCV, the model is trained on all the data except for one observation and then tested on that single left-out observation.



Cross-Validation - Python Example

Scikit-Learn provide us 2 different methods to implement our own cross validation and it can be with any model and with any dataset we want.

corss_val_score() → Method that provides a score based on cross-validation. It splits the input dataset into 'k' number of subsets, used training set as (k-1) subsets and the remaining 1 subset as the test set, and it calculates the accuracy of the model for k times.

The score metric will be from SCORES Scikit-Learn object (as we saw earlier in RidgeCV).

For example → Let's use the **'Advertising.csv'** and simple Ridge regression with fixed alpha value

In [176]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

df = pd.read_csv('/Users/ben.meir/Downloads/Advertising (1).csv')
X = df.drop('sales', axis=1)
y = df['sales']

polynomial_converter = PolynomialFeatures(degree=3, include_bias=False)
polynomial_features = polynomial_converter.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(polynomial_features, y, test_size=0.3, random_state=101)
```

Cross-Validation - Python Example

Next, let's apply standardization feature scaling (because we want to perform Ridge regression)

```
In [177]: from sklearn.preprocessing import StandardScaler

          scaler = StandardScaler()
          scaler.fit(X_train)
          scaled_X_train = scaler.transform(X_train)
          scaled_X_test = scaler.transform(X_test)
```

Finally, let's create our simple Ridge regression model and execute 'cross_val_score':

```
In [178]: from sklearn.linear_model import Ridge
          from sklearn.model_selection import cross_val_score

          ridge_model = Ridge(alpha=10)
          scores = cross_val_score(ridge_model, scaled_X_train, y_train, scoring='neg_mean_squared_error', cv=5)
          scores

Out[178]: array([-0.64743623, -0.5376752 , -2.2331385 , -0.76281271, -0.64660644])
```

The result is the score according to the selected score metric for each fold.
Because we choose cv=5, we got back 5 different score values (score for each fold)

The parameters we are passing to the 'cross_val_score' method →
The model, the X_train, y_train data sets, the scoring metric and the number of folds (cv)

Cross-Validation - Python Example

We can also calculate the mean score value and that's will give us the model result

```
In [182]: abs(scores.mean())

Out[182]: 0.9655338188817334
```

The mean score value in absolute term

In addition, we can adjust our Ridge model alpha value and see if it improved our model accuracy or not.

```
In [186]: from sklearn.linear_model import Ridge
          from sklearn.model_selection import cross_val_score

          ridge_model = Ridge(alpha=1)
          scores = cross_val_score(ridge_model, scaled_X_train, y_train, scoring='neg_mean_squared_error', cv=5)
          abs(scores.mean())

Out[186]: 0.5573176172692851
```

We lowered the alpha value in our model and we got better score mean so our model accuracy improved

Note: higher score meaning better result (this is why we using ‘negative values’) but when we switch to absolute terms the lower the MSE the more aqurete the model.

ECOM SCHOOL

המכללה למקצועות הדיגיטל וההייטק

Cross-Validation - Python Example

corss_validate() → Method that provides automatically cross-validation processing on a given model and given dataset.

Unlike `cross_val_score` which returns only the scoring metric for each testing set, `cross_validate` allows to return multiple scoring metrics and also returns information on fit times and score times for each fold for more extensive evaluation of the model.

For example → Again let's use the 'Advertising.csv' and simple Ridge regression with fixed alpha value

```
In [176]: from sklearn.preprocessing import PolynomialFeatures
          from sklearn.model_selection import train_test_split

          df = pd.read_csv('/Users/ben.meir/Downloads/Advertising (1).csv')
          X = df.drop('sales', axis=1)
          y = df['sales']

          polynomial_converter = PolynomialFeatures(degree=3, include_bias=False)
          polynomial_features = polynomial_converter.fit_transform(X)

          X_train, X_test, y_train, y_test = train_test_split(polynomial_features, y, test_size=0.3, random_state=101)
```

Cross-Validation - Python Example

Next, let's apply standardization feature scaling (because we want to perform Ridge regression)

```
In [177]: from sklearn.preprocessing import StandardScaler

          scaler = StandardScaler()
          scaler.fit(X_train)
          scaled_X_train = scaler.transform(X_train)
          scaled_X_test = scaler.transform(X_test)
```

Finally, let's create our simple Ridge regression model and execute 'cross_validate':

```
In [209]: from sklearn.linear_model import Ridge
          from sklearn.model_selection import cross_validate

          ridge_model = Ridge(alpha=10)
          scores = cross_validate(ridge_model, scaled_X_train, y_train,
                                  scoring=['neg_mean_squared_error', 'neg_mean_absolute_error'], cv=5)
          scores = pd.DataFrame(scores)
          scores
```

Out[209]:

	fit_time	score_time	test_neg_mean_squared_error	test_neg_mean_absolute_error
0	0.001919	0.001634	-3.435384	-1.560038
1	0.001842	0.001454	-1.403711	-1.025212
2	0.001224	0.000675	-5.778437	-1.516873
3	0.001155	0.000341	-2.181487	-1.056032
4	0.000531	0.000300	-4.650883	-1.515156

The parameters we are passing to the 'cross_validate' method →
The model, the X_train, y_train data sets, the scoring metrics as array and the number of folds (cv)

'cross_validate' return its result as a dictionary that we can easily transform into a Dataframe for better visualization

Class Exercise - Cross-Validation

Instructions:

Use the data set provided in the previous class exercise and use **polynomial regression & Ridge** model training.

Hours of calling customers: [2, 3, 4, 5, 6, 1.5, 5, 7, 8, 10]

Money earned: [50K$, 70K$, 90K$, 100K$, 110K$, 40K$, 110K$, 130K$, 145K$, 180K$]

- Convert your model data to be with polynomial regression of 3 degree
- Use feature scaling of type Standardization to prepare your data set
- Use basic Ridge model to eliminate overfitting
- Apply ‘cross_val_score’ for MAE score metric with ‘alpha’=10 and ‘alpha’=100
- Calculate the mean MAE for each alpha and decide which performed better
- Apply ‘cross_validate’ for MAE and SME with ‘alpha’=1, ‘alpha’=10, ‘alpha’=100
- Calculate the mean MAE and SME for each alpha and decide which performed better

Class Exercise Solution - Cross-Validation