

Supervised Machine Learning Algorithms



www.educba.com



ECOM SCHOOL

המכללה למקצועות הדיגיטל וההייטק

Last lecture reminder



We learned about:

- Introduction to SciPy library
- Introduction to Scikit-Learn library
- Installation and import for Scipy and Scikit-Learn
- Supervised machine learning framework - theory
- Supervised machine learning framework - Python example

Evaluating Regression

Until now we learned how to split our data set into train / test set and how to train our model on the train set and predict the label values of the test set.

Once we have the predictions of the test set we can now evaluate our model results.

Most common evaluation metrics for regression:

- **Mean Absolute Error (MAE)** → The average of the absolute differences between predicted and actual values, showing the absolute deviation from true values.
- **Mean Squared Error (MSE)** → The average of the squared differences between predicted and actual values, heavily penalizing large errors.
- **Root Mean Square Error (RMSE)** → The square root of MSE, bringing the errors back to same unit as the original data, and is especially useful when large errors are particularly undesirable.

Evaluating Regression - MAE

The **Mean Absolute Error (MAE)** formula is the following:

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

We are calculating the error of our model predictions by taking the real label value (y_i) and subtract from it the corresponding prediction value (\hat{y}_i).

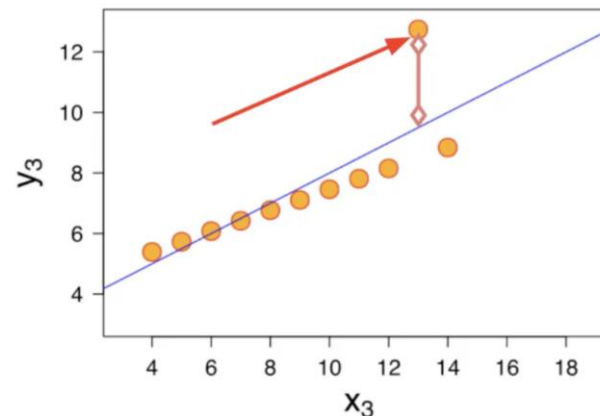
By calculating the sum of all errors and divide by the total amount of predictions we will get the mean error. Because the error can be positive (the model prediction is above the actual value) or negative (the model prediction is below the actual value) and that can affect the sum calculation, we need to use absolut so it will eliminate the negative part.



Evaluating Regression - MAE

The problem with MAE is that the metric not punish us on large errors.

For example → If we will take a look at the following regression line:



MAE won't allow us to know that we have a prediction point that is very far from the actual value.

MAE handle all error values the same so it makes no different if the error is small or large.



Evaluating Regression - MSE

The **Mean Squared Error (MSE)** formula is the following:

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

We are calculating the mean of the squared error, meaning we are taking the model error ($y_i - \hat{y}_i$) and square it to eliminate the error negative values. Then we calculate the sum of all squared errors and divide by the total predictions to get the mean squared error.

MSE allowing us to punish the model more on larger errors, because the error value is been squared the larger the distance between the predication and the actual result, the larger the squared error will be.

Evaluating Regression - MSE

The problem with MSE is that the metric units been squared and are no longer matched to the original model units.

For example → Let's say you have a dataset of housing prices where prices are in units of thousands of dollars (a house worth \$150,000 is represented as 150 in your dataset). You build a model to predict these housing prices and evaluate it using Mean Squared Error (MSE).

If you get an MSE of 4, this doesn't mean your predictions are "off" by \$4,000. Because the MSE metric squares the errors before averaging them, the units also become squared, the MSE would actually be in terms of "thousand dollars squared".



Evaluating Regression - RMSE

The **Root Mean Squared Error (RMSE)** formula is the following:

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

We are calculating the mean of the squared error, same formula as MSE. The only difference is that once

we calculate the sum of the squared errors we execute a root on the result.

This action allowing us to get back the original model units and not deal with squared units as MSE.

Because RMSE allowing us to punish the model on larger errors and get the error metric result with the original model units, it's the most popular regression error metric.



Evaluating Regression - RMSE

When using RSME what will be a good value for RMSE?

This question is the most common question when evaluation a model using RMSE.

The answer to this question is that there is no specific RMSE value that can determine if a model is "good" or "bad".

A good value for the Root Mean Squared Error (RMSE) depends heavily on the context.

Generally, a lower RMSE indicates a better fit to the data. However, the value of RMSE itself doesn't provide a clear indication of "good" or "bad" performance because it depends on the scale of the dependent variable.

Error Metrics - Python Example

In our previous Python example we wanted to train our model on previous campaign expenses in order to predict future campaign sales.

We applied linear regression modeling and train / test split and we got the model predictions to the test set.

```
In [6]: X = df.drop('sales', axis=1)
        y = df['sales']

        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Applying train / test split on our campaign data set

```
In [ ]: from sklearn.linear_model import LinearRegression
        model = LinearRegression()
        model.fit(X_train, y_train)
        model.predict(X_test)
```

Applying linear regression training on our train set and predict() on our test set

```
Out[16]: array([16.5653963, 21.18822792, 21.55107058, 10.88923816, 22.20231988,
                13.35556872, 21.19692502,  7.35028523, 13.27547079, 15.12449511,
                9.01443026,  6.52542825, 14.30205991,  8.97026042,  9.45679576,
                12.00454351,  8.91549403, 16.15619251, 10.29582883, 18.72473553,
                19.76821818, 13.77469028, 12.49638908, 21.53501762,  7.60860741,
                5.6119801,  20.91759483, 11.80627665,  9.08076637,  8.51412012,
                12.17604891,  9.9691939, 21.73008956, 12.77770578, 18.1011362,
                20.07590796, 14.26202556, 20.93826535, 10.83938827,  4.38190607,
                9.51332406, 12.40486324, 10.17045434,  8.09081363, 13.16388427,
                5.2243552,  9.28893833, 14.09330719,  8.69024497, 11.66119763,
                15.71848432, 11.63156862, 13.35360735, 11.1531472,  6.33636845,
                9.76157954,  9.4195714, 24.25516546,  7.69519137, 12.15317572])
```

The model sales prediction for the test set campaign expenses

Error Metrics - Python Example

Now let's evaluate our model predictions by using the error metrics that we learned before.

First, let's see what is the MAE of our model predictions:

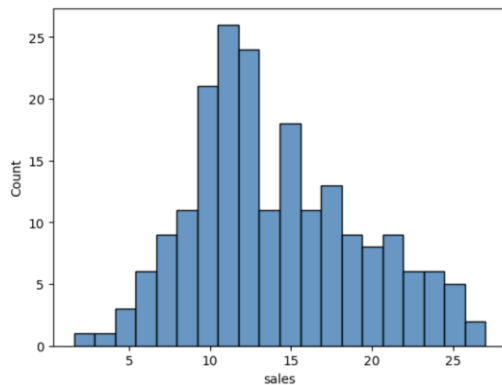
```
In [18]: from sklearn.metrics import mean_absolute_error, mean_squared_error
         test_predictions = model.predict(X_test)
         mean_absolute_error(y_test, test_predictions)

Out[18]: 1.5116692224549093
```

We got that on average, our model will have an error of 1.511 from the real sales value of the campaign

By checking the campaigns sales mean value and the sales distribution we can see that our model prediction error is on average a little above 10% of the real sale amount.

```
In [23]: sns.histplot(data=df, x='sales', bins=20)
         plt.show()
```




```
In [19]: df['sales'].mean()
```

```
Out[19]: 14.0225
```

Error Metrics - Python Example

Next, let's see what is the MSE of our model predictions:

```
In [24]: from sklearn.metrics import mean_absolute_error, mean_squared_error  
         mean_squared_error(y_test, test_predictions)  
  
Out[24]: 3.7967972367152245
```



We got that on average, our model will have an error of 3.79 (squared dollars) from the real sales value of the campaign

The MSE error is higher than the MAE error, that's because the MSE “punish” the model more on grater errors and in addition returns the value in squared units.

If we want to compare the error to the mean sales we need to bring it back to the original units, meaning we need to evaluate the RMSE:

```
In [25]: from sklearn.metrics import mean_absolute_error, mean_squared_error  
         np.sqrt(mean_squared_error(y_test, test_predictions))  
  
Out[25]: 1.9485372043446398
```

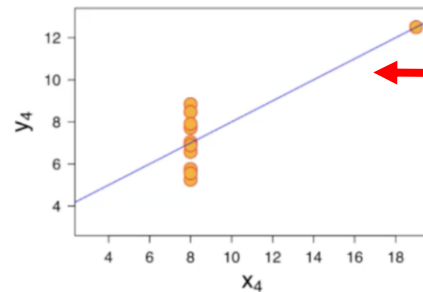
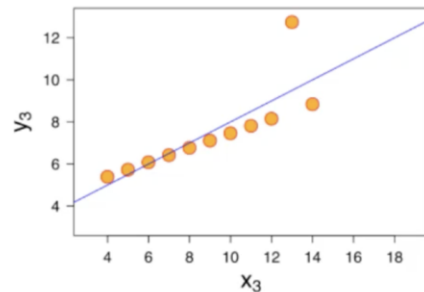
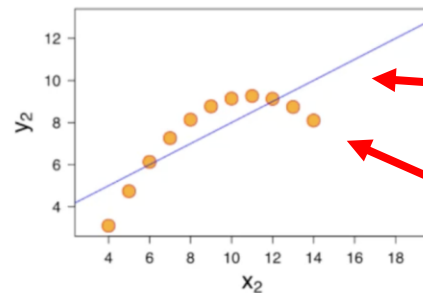
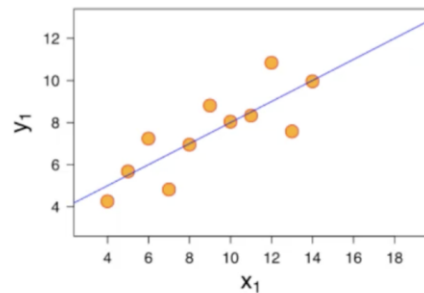
Residual Investigation

Residual → The difference between the observed value of the dependent variable (y) and the predicted value (\hat{y}), given by the model. It is an estimate of the error in the prediction.

Mathematically, it can be represented as: $\text{Residual} = \text{Observed value } (y) - \text{Predicted value } (\hat{y})$.

As we discuss before, not all continuous data fit to linear regression modeling.

For example → let's look at the following observations and their corresponding linear regression line:



All of the following chart has the same mean, same variance and the same linear regression line

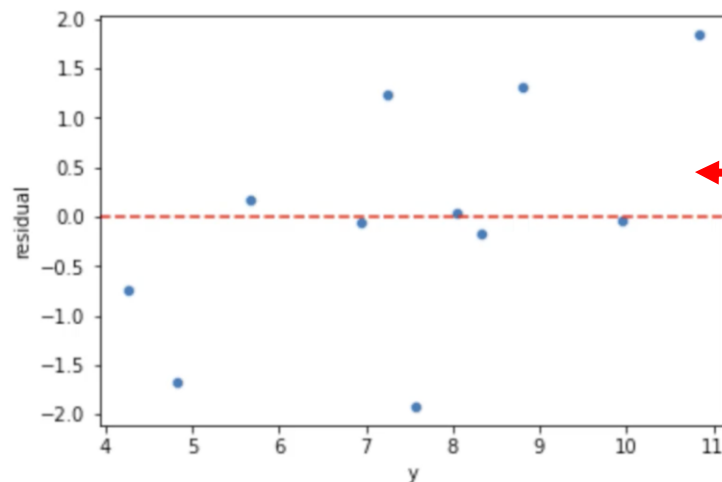
We clearly can see that the following data sets are not suitable for linear regression modeling

Residual Investigation

When dealing with 1 feature (like in the examples) it's easy to tell from the chart itself rather the data set is fit for linear regression or not. But when it's come to multiple features it becomes a harder task.

How can we determine if multiple features data set is suitable for linear regression modeling?

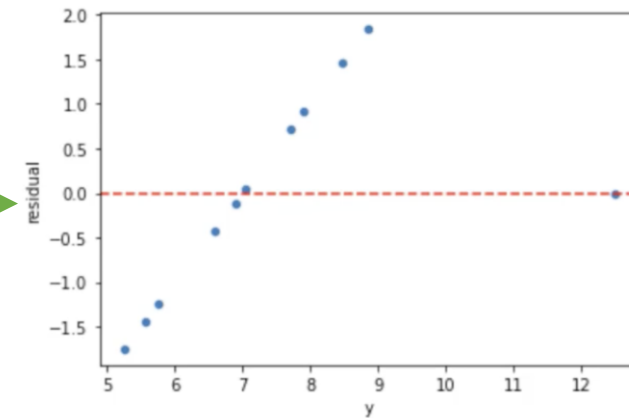
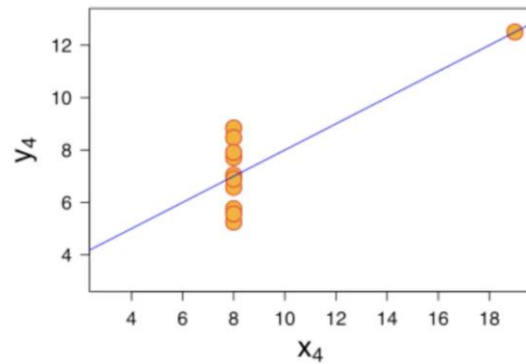
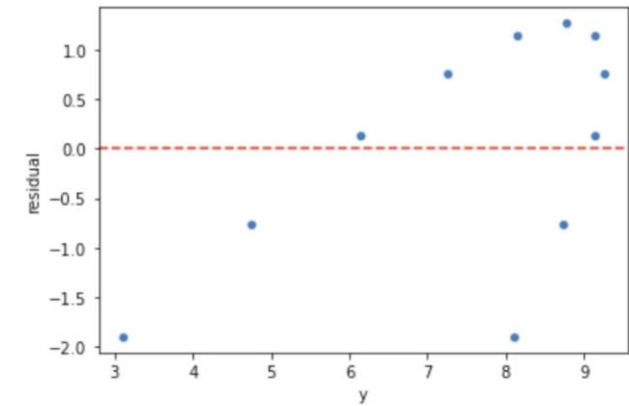
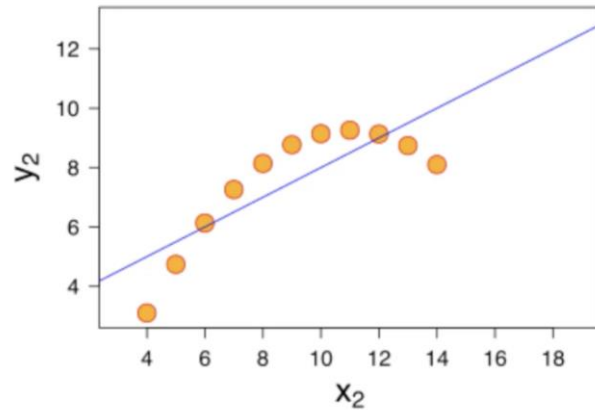
For that we can use the model residuals. What we can do is plot the residual error against the true y values. In case of a linear regression fit data set we will get random points without any pattern.



We plot y vs residual scatter plot and got random points meaning the data set is fit for linear regression modeling

Residual Investigation

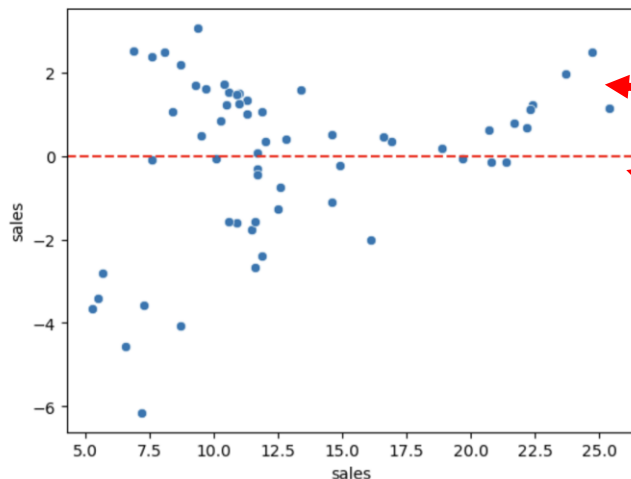
If we will take a non valid data set and check it's y vs residual scatter plot we will get the following chart:



Residual Investigation - Python Example

Now, let's take a look at the y vs residual scatter plot in our campaign sales example and see if the data was fit for linear regression modeling.

```
In [38]: test_residuals = y_test - test_predictions
sns.scatterplot(x=y_test, y=test_residuals)
plt.axhline(y=0, color='r', ls='--')
plt.show()
```



We are plotting the y_{test} values vs the residuals and see that there is no clear pattern in the scatter plot

The red line indicate the “0 error” line. It's where the residual = 0 meaning the prediction was with the same value as the real result

As we can see, there is no clear line or pattern, the dots look randomly distributed what indicating that our data set was indeed fit for linear regression modeling.



Class Exercise - Evaluating Regression

Instructions:

Given the following simple data set that shows the number of hours spent calling customers and the associated money earned:

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\$]

- Perform train test split on the following data points
- Train a linear regression model according to the train set and predict the money earned according to the test set.
- Evaluate your model prediction using MAE, MSE, and RMSE
- Find what is your model average % of error according to each error metric
- Plot the y_{test} vs residuals scatter plot and determine if your data set was a good fit for linear regression modeling



Class Exercise Solution - Evaluating Regression



Model Deployment

Once we finish to examine our model preddication and we are happy with the model result we can move on to the model deployment process.

Model deployment meaning creating a dedicated file (joblib file) that has the model training algorithm. Once we created this file anyone could just import it to its own working area, provide a data set and run the predict() function to get the model results.

The deployment process:

- Save the final model to a new variable
- Fit the final model on the entire data set (no need for train / test split)
- Import from joblib package the dump & load methods
- Execute dump() method on the final model and provide the file name



Model Deployment - Python Example

Let's deploy our final model from the previous Python example:

```
In [44]: final_model = LinearRegression()
         final_model.fit(X, y)
         final_model.coef_

Out[44]: array([ 0.04576465,  0.18853002, -0.00103749])
```

The TV, Radio and Newspaper beta coefficient values according to the final model results

By running `coef_` we can see the corresponding coefficient values for each feature we examine.

Note: positive coefficient value meaning the feature has positive effect on the result.

The more the coefficient closer to 0 the less the effect it has in the result (newspaper has almost no effect on the result).

Model Deployment - Python Example

Now let's export our final model to a joblib file:

```
In [45]: from joblib import dump, load
         dump(final_model, 'final_sales_model.joblib')

Out[45]: ['final_sales_model.joblib']
```

We export our final model to joblib file called 'final_sales_model.joblib'

In order to use the model we can simply import the joblib file and use it again:

```
In [47]: loaded_model = load('final_sales_model.joblib')
         loaded_model.coef_

Out[47]: array([ 0.04576465,  0.18853002, -0.00103749])
```

We loaded the model from the joblib file and print out the model beta coefficients. We can see that it's the same as we got in the 'final_model'

Now, let's predict a new sales campaign using our import model:

The campaign has the following expenses:

- TV = 149K\$
- Radio = 22K\$
- Newspaper = 12K\$

Model Deployment - Python Example

In order to successfully provide this new campaign data to our model we need it to match the shape of the model train data which is a dataframe with rows and each row has 3 columns:

```
In [49]: X.shape  
Out[49]: (200, 3)
```

Let's create a new campaign data and use the loaded_model to predict the sales amount:

```
In [50]: new_campaign = [[149, 22, 12]]  
         loaded_model.predict(new_campaign)  
Out[50]: array([13.893032])
```

We got that the predicted sales amount for the new campaign expenses is 13.89K\$

Remember that our model has an RMSE value of 1.948K\$.

Class Exercise - Deploying Model

Instructions:

Use the data set provided in the previous class exercise and use your linear regression 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\$]

- Train your final model on the entire data set
- Print the beta coefficient values of your final model
- Export your final model into a joblib file
- Import your final model from the joblib file and load it back to your working area
- Use the import model to predict how much money will be earned with the following new hours of calling customers [5.5, 15]

Class Exercise Solution - Deploying Model

