

Machine Learning Basics



Evaluation Metrics For Regression Models

Learn different metrics to evaluate regression models: R^2 , MSE, RMSE, MAE, MAPE, MedAE... The Intuition Behind, Practice with Python to compute them step by step and Interpret the Results

Hanane D.

Contents

1. Loading the dataset and fitting the model:	3
1.1. Dataset	3
1.2. Model to fit	4
2. Evaluation Metrics:	5
2.1. R2 score	5
2.1.1. Definition	5
2.1.2. Recomputed	6
2.1.3. sklearn	6
2.1.4. Both	6
2.1.5. Let's Visualize	6
2.2. Explained Variable score	7
2.2.1. Definition	7
2.2.2. Recomputed	7
2.2.3. sklearn	8
2.2.4. Both	8
2.3. MSE: Mean Squared Error	8
2.3.1. Definition	8
2.3.2. Recomputed	9
2.3.3. sklearn	9
2.3.4. Both	9
2.4. RMSE: Root Mean Squared Error	9
2.4.1. Definition	9
2.4.2. Recomputed	10
2.4.3. sklearn	10
2.4.4. Both	10
2.5. MAE: Mean Absolute Error	10
2.5.1. Definition	10
2.5.2. Recomputed	10
2.5.3. sklearn	11
2.5.4. Both	11
2.6. MAPE: Mean Absolute Percentage Error	11
2.6.1. Definition	11
2.6.2. Recomputed	11
2.6.3. sklearn	12
2.6.4. Both	12
2.7. MedAE: Median Absolute Value	12
2.7.1. Definition	12
2.7.2. Recomputed	12
2.7.3. sklearn	12
2.7.4. Both	13
3. Evaluation metrics to compare among models:	13

Evaluation Metrics For Regression Models

To be able to measure the performance of a regression model and compare several models, different metrics could be used: R^2 score, MSE, RMSE, MAE, MAPE.

In the following, you will find a definition of each metric. Furthermore, based on the famous dataset “Diabetes” used in LARS paper, you will also find a practical example in Python, showing how you compute each metric step by step, and how to use scikit-learn methods. Also, using 2 regression models, you will find how to exploit those measures to choose the best one.

In all the following metrics formulas, we define:

- y_i as the true dependent variable for the observation i
- \hat{y}_i as the predicted value for the observation i
- N as the number of observations in our dataset
- p as the number of independent variables in our dataset

1. Loading the dataset and fitting the model

1.1. Dataset

We will use the Diabetes data used in the "Least Angle Regression" paper: $N=442$ patients, $p=10$ predictors. One row per patient, and the last column is the response variable.

You can find the dataset (raw) in: <https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.txt>

```
df=pd.read_csv("https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.txt", sep="\t")
print(df.shape)
df.head()
```



(442, 11)

	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	59	2	32.1	101.0	157	93.2	38.0	4.0	4.8598	87	151
1	48	1	21.6	87.0	183	103.2	70.0	3.0	3.8918	69	75

We will also normalize data, to have the same order of magnitude among the independent variables:

```
#### Scaled data: X-mean: # Start by subtracting the mean with StandardScaler, without
dividing by the std(with_std=False)
df_sc=StandardScaler(with_std=False).fit_transform(df)
```

```
df_sc=pd.DataFrame(data=df_sc, columns=df.columns)

#### Normalize data: divide each feature by its L2 norm
# If axis=0 no need to transpose, this available in "normalize" method but not in
"Normalizer"
df_norm=normalize(df_sc.iloc[:, :-1], axis=0, norm='l2')
#or transpose the dataframe: (as axis=1 is the default value )
#df_norm=normalize(df_sc.iloc[:, :-1].T, norm='l2')
#(not to forget to transpose the results too, to go back to the initial shape)

df_norm=pd.DataFrame(data=df_norm, columns=df.columns[:-1])

df_norm['Y']=df_sc['Y']
print("Normalized data: Scaled data/L2 norm")
df_norm.head()
```

	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	0.03808	0.05068	0.06170	0.02187	-0.04422	-0.03482	-0.04340	-0.00259	0.01991	-0.01765	-1.13348
1	-0.00188	-0.04464	-0.05147	-0.02633	-0.00845	-0.01916	0.07441	-0.03949	-0.06833	-0.09220	-77.13348
2	0.08530	0.05068	0.04445	-0.00567	-0.04560	-0.03419	-0.03236	-0.00259	0.00286	-0.02593	-11.13348
3	-0.08906	-0.04464	-0.01160	-0.03666	0.01219	0.02499	-0.03604	0.03431	0.02269	-0.00936	53.86652
4	0.00538	-0.04464	-0.03638	0.02187	0.00393	0.01560	0.00814	-0.00259	-0.03199	-0.04664	-17.13348

1.2. Model to fit

To illustrate the computation of the different metrics, step by step, let's take the predicted values given by the OLS model "y_predict_ols". In this example we will not be splitting the dataset to train and test sets. It's not the objective of this notebook.

```
features=df.columns[:-1]
X=df_norm[features]
y=df_norm['Y']

#### OLS: Ordinary Least Square
reg_ols=LinearRegression(fit_intercept=False)
reg_ols.fit(X,y)
#Predict values
y_predict_ols=reg_ols.predict(X)
```

2. Evaluation Metrics:

2.1. R2 score

2.1.1. Definition

R2 score or the coefficient of determination measures how much the explanatory variables explain the variance of the dependent variable. It indicates if the fitted model is a good one, and if it could be used to predict the unseen values.

The best value of R^2 is 1, meaning that the model is a perfect fit of our dataset. It could be 0, meaning that the model is a constant and it will always predict the expected average value of the dependent variable y , regardless of the input variables. It could also be negative, the model could be arbitrarily worse.

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

With \hat{y}_i the predicted value for the observation i , and \bar{y} is the average value of the dependent variable:

$$\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$$

Let's introduce another measure RSS "Residual Sum of Square" which as its name indicates, compute the square of the residual error (difference between the real and the predicted value):

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

We can then, write the formula of the R^2 as:

$$R^2 = 1 - \frac{RSS}{\sum_{i=1}^N (y_i - \bar{y})^2}$$

By adding more and more independent variables to the dataset, the R2 score will be mechanically improved, which does not mean that those variables are really improving the accuracy of the prediction. Therefore, the adjusted R2 is introduced to provide a more precise accuracy by taking into account how many independent variables are used:

$$R_{adj}^2 = 1 - \frac{N-1}{N-p-1} (1 - R^2)$$

2.1.2. Computed

```
df_errors=pd.DataFrame()
df_errors['y_true']=y
df_errors['yhat']=y_predict_ols

df_errors['y_yhat']=df_errors['y_true']-df_errors['yhat']
df_errors['(y_yhat)**2']=df_errors['y_yhat']**2

mean_y=df_errors['y_true'].mean()
df_errors['(y_ymean)**2']=(df_errors['y_true']-mean_y)**2

r2_recomp=1-(np.sum(df_errors['(y_yhat)**2'])/np.sum(df_errors['(y_ymean)**2']))
r2_recomp
```



0.5177484222203499

2.1.3. sklearn

```
r2_sk_learn=r2_score(y, y_predict_ols)
r2_sk_learn
```



0.5177484222203499

2.1.4. Both

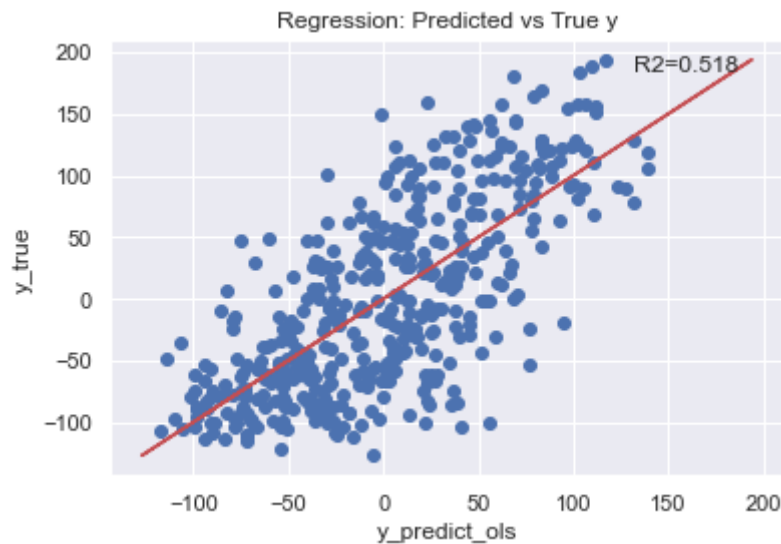
```
print("r2 recomputed:",r2_recomp, ", r2_sk_learn:",r2_sk_learn)
```



r2 recomputed: 0.5177484222203499 , r2_sk_learn: 0.5177484222203499

2.1.5. Let's Visualize

```
plt.scatter(y_predict_ols,y)
plt.plot(y,y, '-r')
plt.annotate(r"R2={0}".format(round(r2_recomp,3)), xy=(200, 200),xytext=(-65, -10),
textcoords='offset points',
          fontsize=12)
plt.xlabel("y_predict_ols")
plt.ylabel("y_true")
plt.title("Regression: Predicted vs True y")
```



2.2. Explained Variable score

2.2.1. Definition

The explained variance score is computed:

$$\text{explained_variance}(y, \hat{y}) = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)}$$

With:

$$\text{Var}(y) = \frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2$$

and:

$$\text{Var}(y - \hat{y}) = \frac{1}{N} \sum_{i=1}^N ((y_i - \hat{y}_i) - (\bar{y} - \bar{\hat{y}}))^2$$

When the residuals have 0 mean, the explained variable score becomes equal to R^2 score:

$$(\bar{y} - \bar{\hat{y}}) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i) = 0$$

2.2.2. Computed

```
N=y.shape[0]

df_errors=pd.DataFrame()
df_errors['y_true']=y
df_errors['yhat']=y_predict_ols

mean_y=df_errors['y_true'].mean()
df_errors['(y_ymean)**2']=(df_errors['y_true']-mean_y)**2
```

```


var_y=np.sum(df_errors['(y_ymean)**2']/N

df_errors['y_yhat']=df_errors['y_true']-df_errors['yhat']
mean_yyhat=df_errors['y_yhat'].mean()
df_errors['(y_yhat_yyhatmean)**2']=(df_errors['y_yhat']-mean_yyhat)**2

var_yyhat=np.sum(df_errors['(y_yhat_yyhatmean)**2']/N

explained_variance_recomp=1-var_yyhat/var_y
explained_variance_recomp

```


 0.5177484222203499

2.2.3. sklearn

```

explained_variance_sklearn=explained_variance_score(y, y_predict_ols)
explained_variance_sklearn

```


 0.5177484222203499

2.2.4. Both

```

print("Explained variance recomputed:",explained_variance_recomp, ", Explained variance from sklearn:",explained_variance_sklearn)

```


 Explained variance recomputed: 0.5177484222203499 , Explained variance from sklearn: 0.5177484222203499

One can see that the Explained Variance and the R2 score are equal. That's because the mean of the residual is ~0:

```

mean_yyhat=df_errors['y_yhat'].mean()
mean_yyhat

```

 -4.4810811672653376e-14

2.3. MSE: Mean Squared Error

2.3.1. Definition

Mean square Error compute the average squared error between the true value and the predicted one:

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

We can recall here that the RSS (Residual Sum of Square):

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

Then:

$$MSE = \frac{RSS}{N}$$

It gives more importance to the highest errors, thus it's more sensitive to outliers.

2.3.2. Computed

```
N=y.shape[0]

df_errors=pd.DataFrame()
df_errors['y_true']=y
df_errors['yhat']=y_predict_ols
df_errors['y_yhat']=df_errors['y_true']-df_errors['yhat']
df_errors['(y_yhat)**2']=df_errors['y_yhat']**2

mse_recomp=np.sum(df_errors['(y_yhat)**2'])/N
mse_recomp
```



2859.69634758675

2.3.3. sklearn

```
mse_sklearn=mean_squared_error(y, y_predict_ols)
mse_sklearn
```



2859.69634758675

2.3.4. Both

```
print("MSE recomputed:",mse_recomp, ", MSE from sklearn:",mse_sklearn)
```



MSE recomputed: 2859.69634758675 , MSE from sklearn: 2859.69634758675

2.4. RMSE: Root Mean Squared Error

2.4.1. Definition

RMSE is the root squared of MSE

$$RMSE = \sqrt{MSE}$$

2.4.2. Computed

```
rmse_recomp=np.sqrt(mse_recomp)
rmse_recomp
```



53.47612876402657

2.4.3. sklearn

```
rmse_sklearn=mean_squared_error(y, y_predict_ols,squared=False)
rmse_sklearn
```



53.47612876402657

2.4.4. Both

```
print("RMSE recomputed:",rmse_recomp, ", RMSE from sklearn:",rmse_sklearn)
```



RMSE recomputed: 53.47612876402657 , RMSE from sklearn: 53.47612876402657

2.5. MAE: Mean Absolute Error

2.5.1. Definition

MAE is the mean absolute error, is more robust to outliers than the MSE:

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

2.5.2. Computed

```
N=y.shape[0]

df_errors=pd.DataFrame()
df_errors['y_true']=y
df_errors['yhat']=y_predict_ols
df_errors['y_yhat']=df_errors['y_true']-df_errors['yhat']
df_errors['abs(y_yhat)']=df_errors['y_yhat'].abs()

mae_recomp=np.sum(df_errors['abs(y_yhat)'])/N
mae_recomp
```



43.27745202531506

2.5.3. sklearn

```
mae_sklearn=mean_absolute_error(y, y_predict_ols)
mae_sklearn
```



43.27745202531506

2.5.4. Both

```
print("MAE recomputed:",mae_recomp, ", MAE sklearn:",mae_sklearn)
```



MAE recomputed: 43.27745202531506 , MAE sklearn: 43.27745202531506

2.6. MAPE: Mean Absolute Percentage Error

2.6.1. Definition

MAPE also known as Mean Absolute Percentage Deviation (MAPD). This measure is sensitive to relative errors. It's also insensitive to a global scaling of the target value. It's also more robust to outliers than MAE.

$$MAPE = \frac{1}{N} \sum_{i=1}^N \frac{|y_i - \hat{y}_i|}{\max(\epsilon, |y_i|)}$$

With ϵ a very small strict positive value, to avoid undefined value when dividing by a value of $y=0$. The score could be high when the y is small, or when the difference of the absolute value is high.

2.6.2. Computed

```
N=y.shape[0]
epsilon=0.0001

df_errors=pd.DataFrame()
df_errors['y_true']=y
df_errors['yhat']=y_predict_ols
df_errors['y_yhat']=df_errors['y_true']-df_errors['yhat']
df_errors['abs(y_yhat)']=df_errors['y_yhat'].abs()
df_errors['abs(y_true)']= df_errors['y_true'].apply(lambda x: max(epsilon,np.abs(x)))
df_errors['(y_yhat)/y']=df_errors['abs(y_yhat)']/df_errors['abs(y_true)']

mape_recomp=np.sum(df_errors['(y_yhat)/y'])/N
mape_recomp
```



3.2636897018293114

2.6.3. sklearn

```
mape_sklearn=mean_absolute_percentage_error(y, y_predict_ols)
mape_sklearn
```



3.2636897018293114

2.6.4. Both

```
print("MAPE recomputed:", mape_recomp, ", MAPE sklearn:", mape_sklearn)
```



MAPE recomputed: 3.2636897018293114 , MAPE sklearn: 3.2636897018293114

2.7. MedAE: Median Absolute Value

2.7.1. Definition

MedAE metric computes the median of all absolute residual values:

$$\text{MedAE}(y, \hat{y}) = \text{median}(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \dots, |y_N - \hat{y}_N|)$$

This metric does not take into consideration the outliers.

2.7.2. Computed

```
N=y.shape[0]
epsilon=0.0001

df_errors=pd.DataFrame()
df_errors['y_true']=y
df_errors['yhat']=y_predict_ols
df_errors['y_yhat']=df_errors['y_true']-df_errors['yhat']
df_errors['abs(y_yhat)']=df_errors['y_yhat'].abs()

medae_recomp=np.median(df_errors['abs(y_yhat)'])
medae_recomp
```



38.5247645673831

2.7.3. sklearn

```
medae_sklearn=median_absolute_error(y, y_predict_ols)
medae_sklearn
```



38.5247645673831

2.7.4. Both

```
print("MAPE recomputed:", mape_recomp, ", MAPE sklearn:", mape_sklearn)
```



MAPE recomputed: 3.2636897018293114 , MAPE sklearn: 3.2636897018293114

3. Evaluation metrics to compare among models

We will use 2 regression models, to fit the data and evaluate their performance. As explained at the beginning, we will not be splitting the dataset to train and test sets. It's not the objective of this notebook.

```
features=df.columns[:-1]
X=df_norm[features]
y=df_norm['Y']

#### OLS: Ordinary Least Square
reg_ols=LinearRegression(fit_intercept=False)
reg_ols.fit(X,y)
#Predict values
y_predict_ols=reg_ols.predict(X)

####LASSO
reg_lasso=Lasso(alpha=1,fit_intercept=False) #Without cross-validation to find the best
alpha, it's not the objective here
reg_lasso.fit(X,y)
#Predict values
y_predict_lasso=reg_lasso.predict(X)
```

Now that we fit the models, let's compute the different metrics for each of them:

```
predicted_values=[y_predict_ols,y_predict_lasso]
models=['OLS','LASSO']
measures_list=[]
i=0

for y_predict in predicted_values:
    r2=r2_score(y, y_predict)
    explained_variance=explained_variance_score(y, y_predict)
    mse=mean_squared_error(y, y_predict)
    rmse=mean_squared_error(y, y_predict,squared=False)
    mae=mean_absolute_error(y, y_predict)
    mape=mean_absolute_percentage_error(y, y_predict)
    medae=median_absolute_error(y, y_predict)
    measures_list.append([models[i],r2,explained_variance,mse,rmse,mae,mape,medae])
    i+=1

df_results=pd.DataFrame(data=measures_list,
columns=['model','r2','explained_var','mse','rmse','mae','mape','medae'])
```

df_results



	model	r2	explained_var	mse	rmse	mae	mape	medae
0	OLS	0.517748	0.517748	2859.696348	53.476129	43.277452	3.263690	38.524765
1	LASSO	0.357379	0.357379	3810.670115	61.730625	52.544928	2.069498	48.658992

As shown in the results, the R^2 is higher for the OLS (0.52) than the Lasso model (0.36) (even if the value in absolute terms is not that high). At this stage, we can assume that the OLS is a better model than the Lasso for our dataset.

Furthermore, the MSE and RMSE are lower for the OLS than the Lasso. Once again, OLS is a good fit for our dataset.

MAE and MedAE are also showing better results for OLS than Lasso.

However, MAPE is lower for the Lasso than the OLS, showing that there are some values of the true y that could be high, making the relative value of the residual lower. It could be interesting to study the outliers in the dataset, and remove them if any.

Globally, the OLS model is showing better metrics than the Lasso model. Thus, between these 2 models, OLS will be a good fit for our dataset.

I hope you enjoy it.

If you want to learn more about data science, have a look on my blog:

<https://machinelearning-basic.blogspot.com/>

Download for free an ebook on the most famous regression models, simply explained:

<https://sites.google.com/view/machinelearning-sample/download-free-ebook>

