

•

Data Science Lab

Scikit-learn Regression

> Andrea Pasini Flavio Giobergia

DataBase and Data Mining Group





 Linear model to predict a single real value based on some input features

$$f(\mathbf{x}) = w_0 + \mathbf{w}^T \mathbf{x} = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

Simple linear regression (1 input feature)

$$f(\mathbf{x}) = w_1 x_1 + w_0$$







Regression with Scikit-learn

```
from sklearn.linear_model import LinearRegression
```

```
reg = LinearRegression(fit_intercept = True)
```

```
reg.fit(X_train, y_train)
```

```
y_test_pred = reg.predict(X_test)
```

The hyperparameter fit_intercept specifies whether the intercept will be computed during training









- Evaluation metrics for regression:
 - MAE (Mean Absolute Error)
 - MSE (Mean Squared Error)
 - R²
- Evaluated by comparing the two vectors
 - y_test (y): the expected result (ground truth)
 - y_test_pred (\hat{y}) : the prediction made by your model





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

MSE (Mean Squared Error)

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

- Both positive numbers
 - MSE tends to penalize less errors close to 0





- R² (R squared)
 - It represents the proportion of variance explained by the predictions

$$R^2 = 1 - \frac{MSE}{std^2}$$

- R² is close to 1 when you have good predictions
- R² negative or close to 0 means wrong predictions







Evaluating regression with Scikit-learn

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
```

```
from sklearn.metrics import mean_squared_error
```

```
# Compute R2, MAE and MSE:
```

```
r2 = r2_score(y_test, y_test_pred)
```

```
mae = mean_absolute_error(y_test, y_test_pred)
```

```
mse = mean_squared_error(y_test, y_test_pred)
```





Evaluation with cross_val_score()

```
from sklearn.model_selection import cross_val_score
```

```
reg = LinearRegression()
```

```
r2 = cross_val_score(reg, X, y, cv=5, scoring='r2')
```

Parameters:

- cv = number of partitions for cross-validation
- scoring = scoring function for the evaluation

E.g. 'r2', 'neg_mean_squared_error'

Similarly, we can use cross_val_predict()



- 4b-Scikitlearn-Linear-Regression.ipynb
 - 1. Simple linear regression
 - 2. Linear regression with multiple input features







- Polynomial regression
 - Useful when the data does not follow a linear trend

It consists of:

- Computing new features that are power functions of the input features
- Applying **linear** regression on the new features







Extracting polynomial features

from sklearn.preprocessing import PolynomialFeatures

```
poly = PolynomialFeatures(5)
```

```
X_poly = poly.fit_transform(X)
```

Input

- degree
 - Integer: the maximal degree of the computed features
 - Tuple: min and max degrees of the computed features
- interaction_only: if True, only include interactions
- include_bias: if True, add a bias column (column of ones)
- Output (of fit_transform())
 - A 2D NumPy array with the new feature matrix





 Building a **pipeline** with polynomial features and linear regression

```
from sklearn.pipeline import make_pipeline
reg = make_pipeline(PolynomialFeatures(5), LinearRegression())
reg.fit(X_train, y_train)
y_test_pred = ret.predict(X_test)
```

Pipelines are objects that allow concatenating multiple Scikit-learn models



- Higher polynomial degree means higher capacity of your model, but ...
 - Pay attention to not overfit your data
 - Overfitting occurs in these cases when you have few samples and a model that has high capacity







- To avoid this form of overfitting
 - Use more training data (if possible)
 - Use lower model complexity (capacity)
 - Use regularization techniques
 - E.g. Ridge, Lasso



- Ridge and Lasso are two techniques for training a linear regression (or a linear regression with polynomial features)
- They try to assign values closer to zero to the coefficients assigned to features that are not useful for the regression
- This effect can decrease the complexity of the model when necessary



- When training normal linear regression you minimize the MSE to compute the coefficients
- When training Ridge you minimize

$$MSE + \alpha(\sum_i w_i^2)$$

When training Lasso you minimize

$$MSE + \alpha(\sum_{i} |w_i|)$$







Ridge:

from sklearn.linear_model import Ridge

```
reg = Ridge(alpha=0.5)
```

Lasso:

from sklearn.linear_model import Lasso
reg = Lasso(alpha=0.5)



- 4c-Scikitlearn-Polynomial-Regression.ipynb
 - 1. Polynomial regression
 - 2. Overfitting and regularization





PoliTo DMG

- Hyperparameters vs parameters
 - Hyperparameters are selected by the user
 - Parameters are computed by the algorithm during training
- Important: hyperparameters cannot be set by finding the values that give the best results on the test set
 - This methodology will overfit the test set
 - We would be using information from the test data to select some training hyperparameters





- There are two valid methodologies
- I. Use hold-out to divide training data in 2 parts
 - Fit different model configurations on the training set
 - Pick the best one by evaluating the performance on the validation set





Finally test the selected model on the actual test set to have a measure of how well the selected hyperparameters work with new data





- 2. Use cross-validation (k-fold) on training data
 - At each iteration 1 partition of the training data is used as validation set, the others are used to train the models





- 2. Use cross-validation (k-fold) on training data
 - For a given configuration we train k models on the training partitions and evaluate them on the validation partition





- 2. Use cross-validation (k-fold) on training data
 - For each model configuration compute the overall scores on the validation partitions
 - Select the configuration with the highest overall score





- This second methodology can be easily performed in Scikit-learn
 - First define a dictionary with the parameter values that you want to tune
 - E.g. for Ridge regression:

With this grid Scikit-learn will try all the combinations:

- {alpha=0.1,fit_intercept=True}
- {alpha=0.1,fit_intercept=False}
- {alpha=0.2,fit_intercept=True}
- {alpha=0.2,fit_intercept=False}
- ParameterGrid() enumerates all these combinations





Then define a model and call GridSearchCV

```
from sklearn.model_selection import GridSearchCV
reg = Ridge()
gridsearch = GridSearchCV(reg, param_grid, scoring='r2', cv=5)
gridsearch.fit(X_train, y_train)
```

- This code will pick the best configuration of the param grid, for Ridge model,
 - According to the R² score
 - Using a cross validation with k=5 partitions



- The best parameter configuration can be found in the *best_params_* attribute of the gridsearch object
- An instance of the model with the best configuration is available in best estimator
 - Important: it is trained on the whole dataset!

```
...
gridsearch.fit(X_train, y_train)
print(gridsearch.best_params_['alpha'])
print(gridsearch.best_params_['fit_intercept')
```

best_configured_model = gridsearch.best_estimator_



4c-Scikitlearn-Polynomial-Regression.ipynb

 3. Grid-search to select model hyperparameters

