Machine Learning with Python

By Dr. Maryam Rahbaralam

جلسه دوم

Train/test split for regression

- Train and test sets are vital to ensure that your supervised learning model is able **to generalize well to new data**.

1. from **sklearn.model_selection** import **train_test_split**
2. **train_test_split()** to create training and test sets.
    - Pass in the arguments X and y, and specify the test set size using test_size (or the training set size using train_size)
    - the random state using random_state.

Create training and test sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=42)
```


KEEP
CALM
AND
LET'S
CODING
KeepCalmAndPosters.com
Maryam Rahbaralam

- In this exercise, you will split the Gapminder dataset into training and testing sets, and then fit and predict a linear regression over **all features**. In addition to computing the **R2 score**, you will also compute the **Root Mean Squared Error (RMSE)**, which is another commonly used metric to evaluate regression models.

To create the regressor

1. use **LinearRegression().**
2. use **.fit()** to fit it to X_train and y_train
3. use **.predict()** to evaluate it over X_test

To compute the RMSE

1. Compute the mean squared error inside the provided **np.sqrt()** function using the **mean_squared_error()** function.

```
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

Excellent!

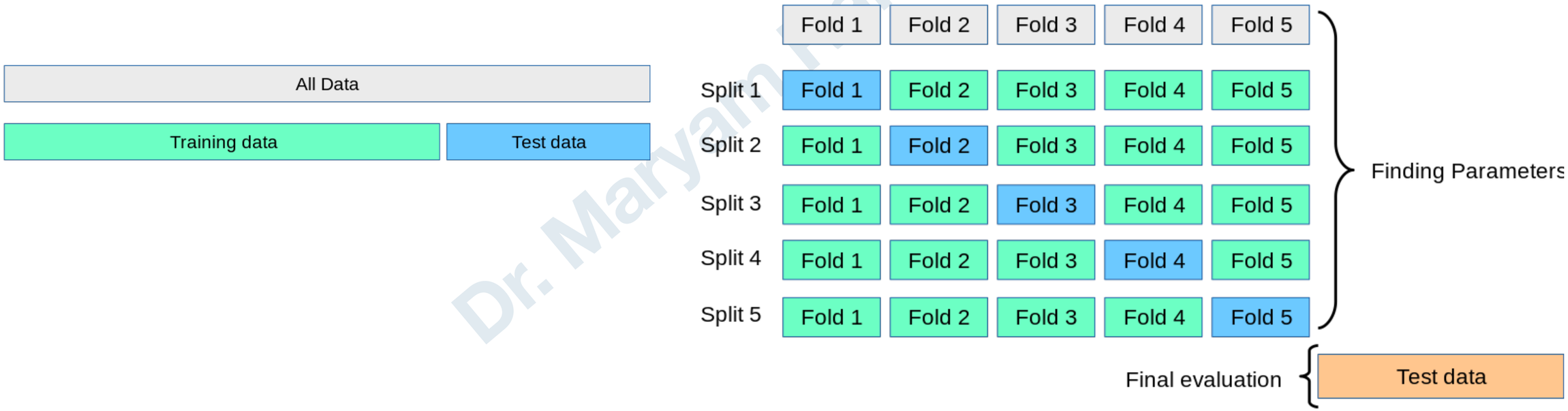
Using all features has improved the model score.
This makes sense, as the model has more information to learn from.

However, there is one potential pitfall to this process.
Can you spot it?
You'll learn about this as well how to better validate your models in the next slide!

Cross-validation motivation

1. Model performance is dependent on way the data is split
2. Not representative of the model's ability to generalize

Solution: Cross-validation!



5-fold cross-validation

1. from **sklearn.model_selection** import **cross_val_score**
2. Create a linear regression regressor LinearRegression()
3. Use the **cross_val_score()** function to perform 5-fold cross-validation on **X** and **y**.

```
cv_scores = cross_val_score(reg, X, y, cv=5)
```

4. Compute and print the average cross-validation score. You can use NumPy's **mean()** function to compute the average.

```
np.mean(cv_scores)
```


KEEP
CALM
AND
LET'S
CODING
KeepCalmAndPosters.com
Maryam Rahbaralam

Regularized regression

1. Recall: What fitting a Linear regression does?

   - Linear regression **minimizes a loss function**

   - It chooses a coefcient for each feature variable

Overfitting

   - Large coefcients can lead to overtting

Solution: Penalizing large coefcients: **Regularization**

Regularized regression

1. A simple relation for linear regression looks like this. Here Y represents the learned relation and *β represents the coefficient estimates for different variables or predictors(X).*

$$Y \approx \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp$$

The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

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

Now, this will adjust the coefficients based on your training data.

Regularized regression

Overfitting

1. If there is

   - noise in the training data
   - your data is high-dimensional space with large coefficients,
   - it gets easy to predict nearly anything then the estimated coefficients won't generalize well to the future data. = **Overfitting**

Solution: This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

Regularized regression

Ridge regression

1. where the ***RSS is modified by adding the shrinkage quantity.***

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

L2 norm.

- Now, the coefficients are estimated by minimizing this function. Here, ***λ is the tuning parameter that decides how much we want to penalize the flexibility of our model.***
- When **λ = 0**, the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares => lead **overfitting**.
- However, as **λ→∞**, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero => lead to a simple model => **Underfitting**

Regularized regression

Ridge regression

```
1. from sklearn.linear_model import Ridge

2. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
   random_state=42)

3. ridge = Ridge(alpha=0.1, normalize=True)

4. ridge.fit(X_train, y_train)

5. ridge_pred = ridge.predict(X_test)

6. ridge.score(X_test, y_test)
```


KEEP
CALM
AND
LET'S
CODING
KeepCalmAndPosters.com
Maryam Rahbaralam

Regularized regression

Lasso regression

1. where the ***RSS is modified by adding the shrinkage quantity.***

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

L1 norm.

- Now, the coefficients are estimated by minimizing this function. Here, ***λ is the tuning parameter that decides how much we want to penalize the flexibility of our model.***
- When **λ = 0**, the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares => lead **overfitting**.
- However, as **λ→∞**, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero => lead to a simple model => **Underfitting**

Regularized regression

Ridge regression
Lasso regression

1. **This sheds light on the obvious disadvantage of ridge regression, which is model interpretability.**

2. It will shrink the coefficients for least important predictors, very close to zero. But it will **never make them exactly zero**. In other words, the final model will **include all predictors.**

3. However, in the case of the lasso, the L1 penalty has the effect of forcing some of the coefficient estimates to be **exactly equal to zero** when the tuning parameter λ is sufficiently large.

Therefore, the lasso method also performs variable selection and is said to yield sparse models.

Regularized regression

Lasso regression

```
1. from sklearn.linear_model import Lasso

2. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
   random_state=42)

3. lasso = Lasso(alpha=0.1, normalize=True)

4. lasso.fit(X_train, y_train)

5. lasso_pred = lasso.predict(X_test)

6. lasso.score(X_test, y_test)
```