assignment08

April 14, 2021

[ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
```

[ ]:

```
# Hidden parts create data x and y



x = np.linspace(-3,3, 30).reshape(-1, 1)
np.random.seed(42)
y_clean= x * -0.8545 + x**2*0.35+ x**3*0.25
y = y_clean + np.random.normal(np.zeros_like(x),1)



plt.plot(x, y, "x")
plt.xlabel("x")
plt.ylabel("y")
```

0.1 Regularisation - Ridge Regression

1. Create a polynomial input X_poly of degree 10 (without a bias term) from the sampling locations x
2. For alpha = 0, 1, 5, 10 fit a Ridge Regressor to X_poly and y
3. Plot the predictions of the respective models over x and create a legend to distinguish them

You can check X_poly in the cell below.

[ ]:

```
from sklearn.linear_model import Ridge

plt.plot(x, y, "x")
plt.xlabel("x")
plt.ylabel("y")



# YOUR CODE HERE
raise NotImplementedError()
```

[ ]:

```
# Hidden part checks X_poly to avoid simple mistakes


if is_correct
    print("X_poly is correct!")
else:
    print("X_poly is not correct!")
```

0.2 Regularisation - Decision Tree

Tune the following parameters by hand to get a feel for their effect: * max_depth: Max depth of the tree * min_samples_split: The minimum number of samples required to split an internal node * min_samples_leaf: The minimum number of samples required to be at a leaf node.

The goal is to get a reasonable regressor. The original settings are below if you want to reset your try:

`regressor = DecisionTreeRegressor(max_depth=None, min_samples_split=2, min_samples_leaf=1, ranc`

[ ]:

```
from sklearn.tree import DecisionTreeRegressor
plt.plot(x, y, "x")
plt.xlabel("x")
plt.ylabel("y")
regressor = DecisionTreeRegressor(max_depth=None, min_samples_split=2,␣
 ↪min_samples_leaf=1, random_state=0)
regressor.fit(x,y)
y_pred = regressor.predict(x)
plt.plot(x, y_pred, label="alpha=")

# YOUR CODE HERE
raise NotImplementedError()
```

0.3 Cross Validation

To get to know cross validation you should see it in comparison to “traditional” train / test splitting. Therefore, implement the following steps:

- For a random seed from 1 to 5 split the X_poly and y into a train and test set, with a test_size of 0.33 and respective random_state. Please use sklearn’s train_test_split for this”: X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.33, random_state=seed). Please use X_poly from the last exercise.
- For each random seed train a ridge regressor with an alpha of your choice on the training set and calculate and print its MSE with respect to the test set
- Plot the predictions of the model for the whole range of x (not just X_train or X_test)

[ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.utils import shuffle
```

```
scorer = make_scorer(mean_squared_error)

plt.plot(x, y, "x")
plt.xlabel("x")
plt.ylabel("y")


# YOUR CODE HERE
raise NotImplementedError()

model = Ridge(alpha=10)
X_poly_, y_ = shuffle(X_poly, y)
scores = cross_val_score(model, X_poly_, y_, cv=10, scoring=scorer)
print("\n")
print(scores.mean())
```

0.4 Toy Problem

In this last part of today's assignment you can try out everything you've learned so far. In the next hidden cell a dataset is created that you should analyse and build a model for. Please use standard sklearn functions as the evaluation cell uses model.predict(test_data) to calculate the root mean square error.

Your goal is to build a model with a RMSE below 4, any model that can be evaluated with model.predict is fair game! Feel free to try different models, hyperparameter search, cross validation and other techniques to solve this exercise!

[ ]:

```
# Hidden code generates X_train and y_train

from sklearn.datasets import load_boston
X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=711)
y_test = 5
X_test = 5
```

[ ]:

```
# YOUR CODE HERE
raise NotImplementedError()
```

[ ]:

```
# Hidden code uses model on test set and returns the RMSE.


print(rmse)
assert rmse < 4
```

[ ]:

```

```