# 26-Stacking_Regressor

October 20, 2024

## 1 Stacking Regression

A **Stacking Regressor** is an ensemble machine learning technique that combines the predictions of multiple regression models (base models) by training a meta-model (stacker) to make final predictions based on the outputs of those base models. Stacking, also known as stacked generalization, aims to leverage the strengths of several models to improve predictive performance compared to using any single model.

The key idea of stacking is that different models may excel in capturing different aspects of the data. By training a second-level model (meta-model) on the outputs of base models, stacking allows the meta-model to learn how to best combine the predictions of the base models.

### 1.0.1 Why is Stacking Regressor Important?

- **Boosts model performance**: Since stacking combines the predictive power of several models, it often yields better performance than individual models. It uses the strengths of each model while compensating for their weaknesses.
- **Improves generalization**: Stacking helps in reducing overfitting by leveraging a blend of multiple models. The meta-model is trained to understand which base models perform best under different circumstances.
- **Flexibility**: You can use a wide range of base learners (e.g., decision trees, linear regression, support vector machines, etc.) and combine their predictions in a meaningful way, allowing for diverse model architectures.

### 1.0.2 How does Stacking Regressor work?

Stacking involves two layers:

1. **Base Models (Level 0)**: These are the individual regression models that make predictions on the dataset. Each base model is trained on the original features of the data.
2. **Meta-model (Level 1)**: The meta-model (also called the blender or stacker) is trained on the predictions of the base models. It learns to combine these predictions to generate the final output.

Here's a step-by-step breakdown of how stacking works:

1. **Train base models**: Each base model is trained on the training data. These models could be decision trees, random forests, gradient boosting, linear regression, etc.

2. **Generate base model predictions**: Once trained, the base models are used to make predictions on both the training data (for cross-validation) and the test data. These predictions are then used as inputs to the meta-model.

3. **Train the meta-model**: A second model (meta-model), often a simple linear regression or another powerful algorithm, is trained using the predictions from the base models as features. This model learns to combine the outputs of the base models to improve predictive performance.

4. **Final prediction**: During the prediction phase, the base models make predictions on new data, and these predictions are fed into the meta-model, which then produces the final prediction.

#### 1.0.3 When should you use Stacking Regressor?

- **When base models have different strengths**: If your base models have different inductive biases and capture different patterns in the data (e.g., decision trees might capture non-linear relationships, while linear models capture linear trends), stacking can help by learning how to best combine them.
- **High complexity datasets**: Stacking can be useful when the dataset is complex and no single model is able to fully capture the data's underlying patterns. By combining multiple models, stacking often performs better in these cases.
- **To reduce overfitting**: Stacking helps in preventing overfitting by blending multiple models. If individual models tend to overfit, the meta-model can act as a stabilizing factor by learning to trust the better-performing models.
- **When you want to combine models with different architectures**: Stacking gives the flexibility to combine a variety of different models (e.g., tree-based models, linear models, neural networks, etc.) in one powerful ensemble.

#### 1.0.4 Who uses Stacking Regressor?

- **Data Scientists and Machine Learning Engineers**: Stacking is a common technique used in competitions like Kaggle and real-world applications where predictive performance is critical. It's widely adopted when optimizing models for predictive accuracy.
- **Financial Analysts and Economists**: Stacking is used in scenarios such as stock market predictions or economic forecasting, where combining models that capture different signals (e.g., time-series patterns, economic indicators, etc.) can enhance the overall predictive power.
- **Researchers and AI Developers**: Stacking is often used in research when building complex models that need to integrate multiple types of predictions (e.g., biological data analysis, medical imaging, etc.).

#### 1.0.5 Key Points to Remember:

- **Base Models and Meta-model**: The base models can be diverse (linear models, tree-based models, SVM, etc.), and the meta-model is trained on their predictions to combine them effectively.

- **Cross-Validation**: Stacking usually involves cross-validation to avoid overfitting. The meta-model should be trained on out-of-sample predictions to ensure it generalizes well.
- **Customizability**: You can choose any combination of models for the base learners and meta-model depending on the problem and data structure.

### 1.0.6 How does Stacking differ from Bagging and Boosting?

- **Bagging**: Bagging (e.g., Random Forest) trains multiple models in parallel on different subsets of the data and averages their predictions to reduce variance.
- **Boosting**: Boosting (e.g., Gradient Boosting) trains models sequentially, where each subsequent model attempts to correct the mistakes of the previous one.
- **Stacking**: Stacking trains multiple models in parallel (like Bagging) but uses another model to learn how to best combine the outputs of the base models.
- Lasso Regression captures important features by shrinking irrelevant ones.
- Ridge Regression handles multicollinearity and smooths out feature effects.
- Support Vector Regression captures complex, non-linear relationships.

[1]:
```
import pandas as pd
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import StackingRegressor, RandomForestRegressor,␣
 ↪GradientBoostingRegressor, VotingRegressor
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.metrics import mean_absolute_error, root_mean_squared_error,␣
 ↪r2_score
from sklearn.svm import SVR

mpg = sns.load_dataset('mpg')
mpg
```

[1]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | \ |
|---|---|---|---|---|---|---|---|
| 0 | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | |
| 1 | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | |
| 2 | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | |
| 3 | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | |
| 4 | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | |
| .. | … | … | … | … | … | … | |
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790 | 15.6 | |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130 | 24.6 | |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295 | 11.6 | |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625 | 18.6 | |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720 | 19.4 | |

```
     model_year  origin                       name
0            70     usa  chevrolet chevelle malibu
1            70     usa          buick skylark 320
2            70     usa         plymouth satellite
3            70     usa              amc rebel sst
4            70     usa                ford torino
..          …     …                          …
393          82     usa            ford mustang gl
394          82  europe                  vw pickup
395          82     usa              dodge rampage
396          82     usa                ford ranger
397          82     usa                 chevy s-10

[398 rows x 9 columns]
```

[2]:
```
mpg = mpg.drop('name', axis=1)
mpg = pd.get_dummies(mpg)
mpg
```

[2]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
0    18.0          8         307.0       130.0    3504          12.0
1    15.0          8         350.0       165.0    3693          11.5
2    18.0          8         318.0       150.0    3436          11.0
3    16.0          8         304.0       150.0    3433          12.0
4    17.0          8         302.0       140.0    3449          10.5
..    …         …           …          …      …           …
393  27.0          4         140.0        86.0    2790          15.6
394  44.0          4          97.0        52.0    2130          24.6
395  32.0          4         135.0        84.0    2295          11.6
396  28.0          4         120.0        79.0    2625          18.6
397  31.0          4         119.0        82.0    2720          19.4

     model_year  origin_europe  origin_japan  origin_usa
0            70          False         False        True
1            70          False         False        True
2            70          False         False        True
3            70          False         False        True
4            70          False         False        True
..          …           …           …         …
393          82          False         False        True
394          82           True         False       False
395          82          False         False        True
396          82          False         False        True
397          82          False         False        True

[398 rows x 10 columns]
```

[3]:
```
pd.DataFrame(mpg.isnull().sum().sort_values(ascending=False))
```

[3]:

| | 0 |
|---|---|
| horsepower | 6 |
| mpg | 0 |
| cylinders | 0 |
| displacement | 0 |
| weight | 0 |
| acceleration | 0 |
| model_year | 0 |
| origin_europe | 0 |
| origin_japan | 0 |
| origin_usa | 0 |

[4]:
```
mpg['horsepower'].fillna(mpg['horsepower'].mean(), inplace=True)
```

```
C:\Users\ikiga\AppData\Local\Temp\ipykernel_26612\370553803.py:1: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  mpg['horsepower'].fillna(mpg['horsepower'].mean(), inplace=True)
```

[5]:
```
mpg
```

[5]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | \ |
|---|---|---|---|---|---|---|---|
| 0 | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | |
| 1 | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | |
| 2 | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | |
| 3 | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | |
| 4 | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | |
| .. | … | … | … | … | … | … | |
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790 | 15.6 | |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130 | 24.6 | |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295 | 11.6 | |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625 | 18.6 | |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720 | 19.4 | |

| | model_year | origin_europe | origin_japan | origin_usa |
|---|---|---|---|---|
| 0 | 70 | False | False | True |
| 1 | 70 | False | False | True |

```
2            70          False          False         True
3            70          False          False         True
4            70          False          False         True
..          …            …              …             …
393          82          False          False         True
394          82           True          False        False
395          82          False          False         True
396          82          False          False         True
397          82          False          False         True

[398 rows x 10 columns]
```

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

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,␣
↪random_state=19)
```

### 1.1 Linear Regression

[7]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

[7]:
```
LinearRegression()
```

[8]:
```
y_pred = lr.predict(X_test)
y_pred
```

[8]:
```
array([24.30115472, 24.73892314, 24.96215822, 17.38881632, 15.86675499,
       27.76426172, 17.05954093, 10.49344079, 20.347386  , 25.11254933,
       19.36068674, 28.56363082, 26.2720292 , 17.73036858, 20.66471276,
       28.7669855 , 28.98129852, 29.25304272, 16.8393709 ,  8.76273604,
       16.24941413, 13.29303025, 27.03902874, 23.9878045 ,  8.31184254,
       31.52160011, 13.42789179, 35.09654981, 11.22026997, 23.39755039,
       33.81737584, 29.59696005, 15.57771451, 23.05695715, 31.56588099,
       36.11360599, 27.33151931, 12.33155235, 28.54718257, 15.3977963 ,
       31.21498729, 23.50190767, 14.44084076, 27.66645708, 26.78275774,
       25.05523353, 24.20075891, 16.30434811, 15.63453382, 12.61013012,
       17.36880625, 20.27285977, 30.10153781, 16.80607781, 33.29417373,
       11.53912186, 32.05759396, 21.35355061, 18.08124065, 30.24876403,
       20.89162341, 20.33755243, 10.89210702, 35.17824516, 32.51214434,
       15.01604073, 25.98942786, 15.12905345, 32.04280232, 24.17805781,
       29.82643316, 22.55608296, 14.09889069, 14.59105924, 26.25541638,
       23.19242234, 12.07920811, 33.66219498, 29.56124407, 20.90794551])
```

[9]:
```
mean_absolute_error(y_test, y_pred)
```

[9]: 2.3241334520650594

[10]:
```
root_mean_squared_error(y_test, y_pred)
```

[10]: 8.30859663913005

[11]:
```
r2_score(y_test, y_pred)
```

[11]: 0.8325409560733986

### 1.2 Random Forest Regressor

[12]:
```
rfr = RandomForestRegressor(random_state=13)
rfr.fit(X_train, y_train)
```

[12]: RandomForestRegressor(random_state=13)

[13]:
```
y_pred_rfr = rfr.predict(X_test)
y_pred_rfr
```

[13]:
```
array([24.91 , 26.604, 24.747, 15.894, 15.851, 25.264, 18.296, 13.44 ,
       18.43 , 25.158, 17.164, 32.449, 26.397, 16.445, 19.475, 26.181,
       28.641, 33.474, 15.54 , 12.64 , 15.985, 14.75 , 24.137, 22.409,
       12.15 , 30.689, 13.895, 37.527, 13.52 , 21.971, 35.155, 27.763,
       15.234, 26.289, 33.153, 36.126, 23.992, 13.88 , 35.015, 14.225,
       31.179, 25.104, 15.482, 31.032, 25.78 , 26.102, 21.299, 16.106,
       14.604, 14.18 , 16.227, 18.722, 35.506, 16.804, 34.797, 13.82 ,
       32.936, 18.86 , 18.847, 36.382, 15.164, 21.129, 13.38 , 36.05 ,
       33.419, 16.248, 24.139, 16.082, 32.642, 23.887, 30.994, 20.967,
       14.33 , 15.03 , 24.663, 21.33 , 14.09 , 38.027, 31.285, 19.26 ])
```

[14]:
```
mean_absolute_error(y_test, y_pred_rfr)
```

[14]: 1.6326000000000005

[15]:
```
root_mean_squared_error(y_test, y_pred_rfr)
```

[15]: 5.647237075000005

[16]:
```
r2_score(y_test, y_pred_rfr)
```

[16]: 0.8861804270347423

### 1.3 Ridge Regression

[17]:
```
ridge = Ridge()
param_grid = {
    'alpha': [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30 ,50,75]
}

ridge_cv = GridSearchCV(ridge, param_grid, cv=5, n_jobs=-1)
ridge_cv.fit(X_train, y_train)
```

[17]:
```
GridSearchCV(cv=5, estimator=Ridge(), n_jobs=-1,
             param_grid={'alpha': [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50,
                                   75]})
```

[18]:
```
ridge_cv.best_estimator_
```

[18]:
```
Ridge(alpha=10)
```

[19]:
```
y_pred_ridge = ridge_cv.predict(X_test)
y_pred_ridge
```

[19]:
```
array([24.50784329, 25.1453431 , 24.53292975, 17.28315713, 15.76458563,
       27.56386969, 17.08360929, 10.4757291 , 20.27551626, 25.32880598,
       19.37647572, 28.86733796, 26.62154492, 17.73752567, 20.67788725,
       28.61052096, 29.25178742, 29.5422219 , 16.87281192,  8.53318336,
       16.27511627, 13.2104424 , 26.88340032, 24.009771  ,  8.54411519,
       31.26528627, 13.51952031, 34.90097167, 10.98124741, 23.16980535,
       33.58843945, 29.44372653, 15.49893771, 23.46053976, 31.4803678 ,
       35.94195329, 27.1668034 , 12.13433213, 28.84816147, 15.45947453,
       30.97278809, 23.81928018, 14.51031251, 27.2552616 , 26.63822582,
       25.25480835, 24.33315835, 16.31946596, 15.52974001, 12.67976987,
       17.40627283, 20.26742196, 30.05946769, 16.89475666, 33.16874796,
       11.53698617, 31.9061312 , 21.40546978, 18.10722736, 30.55085517,
       20.62056802, 20.53397226, 10.65600453, 35.01686624, 32.35642928,
       15.05582844, 25.86631482, 15.21926104, 31.87832831, 24.08914757,
       29.75506395, 22.43481848, 14.10345592, 14.38519282, 26.3607398 ,
       23.43599278, 12.1534485 , 33.50094882, 29.46266978, 20.95613101])
```

[20]:
```
mean_absolute_error(y_test, y_pred_ridge)
```

[20]:
```
2.298757602037677
```

[21]:
```
root_mean_squared_error(y_test, y_pred_ridge)
```

[21]:
```
8.045449532074198
```

[22]:
```
r2_score(y_test, y_pred_ridge)
```

[22]:
```
0.8378446631702253
```

### 1.4 Gradient Boosting Regressor

[23]:
```
gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)
```

[23]:
```
GradientBoostingRegressor()
```

[24]:
```
y_pred_gbr = gbr.predict(X_test)
y_pred_gbr
```

[24]:
```
array([24.81614709, 26.92370305, 24.05991901, 17.30771176, 16.19740785,
       25.28778472, 18.47327473, 13.4293537 , 17.87010961, 25.32283075,
       17.3784293 , 32.80854588, 26.43231189, 16.71346622, 19.89542764,
       26.93072095, 29.09128776, 34.5437197 , 16.2287741 , 12.9976502 ,
       16.15260843, 14.0218489 , 23.430844  , 23.554742  , 12.15018212,
       29.3384603 , 13.78280808, 37.46136379, 13.83317576, 21.92720442,
       35.35308293, 27.59229457, 15.38298242, 25.77062188, 32.41174551,
       36.29002888, 22.94838594, 13.1043108 , 35.75852911, 14.27426203,
       31.12414823, 24.07273405, 15.53471207, 33.44820815, 26.03408155,
       25.8403547 , 20.59553436, 16.25817053, 15.70183123, 13.21764247,
       16.88855731, 18.84642478, 31.8104251 , 16.10117025, 36.1970594 ,
       13.4293537 , 33.0030821 , 18.74768476, 18.59036535, 35.9761289 ,
       19.47705059, 20.92402884, 14.42364406, 36.40972479, 32.92852343,
       16.49541674, 24.5359267 , 14.67871467, 31.81852565, 23.62370365,
       29.16526417, 20.5589327 , 14.29479101, 15.56229952, 23.83009838,
       21.90942175, 13.86561836, 34.69275483, 29.33933   , 19.03518476])
```

[25]:
```
mean_absolute_error(y_test, y_pred_gbr)
```

[25]:
```
1.7513204620962235
```

[26]:
```
root_mean_squared_error(y_test, y_pred_gbr)
```

[26]:
```
5.225603103128156
```

[27]:
```
r2_score(y_test, y_pred_gbr)
```

[27]:
```
0.8946784231324356
```

### 1.5 Stacking Regression

[28]:
```
estimators = [
    ('rfr', rfr),
    ('ridge', ridge_cv.best_estimator_),
    ('lr', lr)
]
```

```
sr = StackingRegressor(
    estimators=estimators,
    final_estimator=gbr
)

sr.fit(X_train, y_train)
```

[28]:
```
StackingRegressor(estimators=[('rfr', RandomForestRegressor(random_state=13)),
                              ('ridge', Ridge(alpha=10)),
                              ('lr', LinearRegression())],
                  final_estimator=GradientBoostingRegressor())
```

[29]:
```
y_pred_sr = sr.predict(X_test)
y_pred_sr
```

[29]:
```
array([23.53378874, 25.71455216, 24.09668026, 15.40149474, 15.40615392,
       27.36438654, 17.98058559, 13.43407508, 18.52080362, 24.24343196,
       16.93869117, 30.90969533, 26.90396953, 16.73109235, 18.98478987,
       26.63577215, 25.55108187, 29.9583349 , 15.33757724, 12.2164736 ,
       15.40615392, 15.17557021, 24.92925791, 22.1997565 , 12.13294436,
       33.13916835, 13.79983485, 34.80103745, 13.56035943, 20.45624966,
       36.40332172, 27.01318555, 15.40615392, 23.05829459, 31.64665183,
       36.72001283, 27.15296778, 13.79983485, 32.15462099, 14.16801969,
       34.11696062, 23.53378874, 15.87656151, 32.40313431, 25.59016802,
       25.45964114, 20.68403814, 15.81340739, 15.40615392, 14.3074484 ,
       15.9730475 , 18.43689213, 31.85386936, 16.50287556, 39.40124297,
       13.56035943, 33.96965856, 17.56922769, 18.4320498 , 36.70733073,
       16.3237033 , 20.25150774, 13.56035943, 37.78515789, 34.28042863,
       15.87161528, 25.40213556, 15.87161528, 32.20626472, 23.6224426 ,
       30.78302652, 20.5972072 , 15.00843971, 15.46436181, 25.59016802,
       20.68403814, 14.3074484 , 37.76093249, 30.96411227, 18.98478987])
```

[30]:
```
mean_absolute_error(y_test, y_pred_sr)
```

[30]:
```
1.9308715384123967
```

[31]:
```
root_mean_squared_error(y_test, y_pred_sr)
```

[31]:
```
6.580558650525768
```

[32]:
```
r2_score(y_test, y_pred_sr)
```

[32]:
```
0.8673694117090569
```

### 1.6 Voting Regressor

[33]:
```
vr = VotingRegressor([
    ('rfr', rfr),
    ('gbr', gbr),
    ('lr', lr)
], weights=[2,3,1])

vr.fit(X_train, y_train)
```

[33]:
```
VotingRegressor(estimators=[('rfr', RandomForestRegressor(random_state=13)),
                            ('gbr', GradientBoostingRegressor()),
                            ('lr', LinearRegression())],
                weights=[2, 3, 1])
```

[34]:
```
y_pred_vc = vr.predict(X_test)
y_pred_vc
```

[34]:
```
array([24.76159933, 26.45300538, 24.43931921, 16.84999193, 16.02682976,
       25.69260264, 18.17856085, 12.94358365, 18.46961914, 25.23284026,
       17.63732911, 31.98121141, 26.39382748, 16.79346121, 19.88349928,
       26.98685806, 28.9228603 , 33.30536697, 16.10094887, 12.17261444,
       16.11287324, 14.14309616, 24.26759346, 23.24500508, 11.51039815,
       30.15249683, 13.76105267, 37.08910687, 13.29329954, 22.18686061,
       35.0311041 , 27.98330729, 15.36577696, 25.49113713, 32.49028417,
       36.20594877, 24.02677952, 13.23408079, 34.30879498, 14.44509707,
       31.157572  , 24.32135164, 15.3348295 , 31.67918025, 26.07416706,
       25.79671627, 21.43089366, 16.21514328, 15.32467125, 13.43717625,
       16.7480797 , 19.04268902, 32.84100406, 16.45293143, 35.24655866,
       13.24453049, 32.82314004, 19.21943415, 18.59105612, 35.15685845,
       18.27006274, 20.89460649, 13.48210674, 36.08456992, 33.0226191 ,
       16.16638182, 24.64586799, 15.22153291, 32.13039654, 23.51922136,
       29.75720817, 21.02781351, 14.27387728, 15.22299296, 24.51195192,
       21.9301146 , 13.6426772 , 35.63240991, 30.02487235, 19.42224996])
```

[35]:
```
root_mean_squared_error(y_test, y_pred_vc)
```

[35]:
```
5.0571779581339475
```

[36]:
```
r2_score(y_test, y_pred_vc)
```

[36]:
```
0.8980730173074735
```

### 1.7 Final Estimator

[37]:
```
estimators2 = [
    ('rfr', rfr),
    ('ridge', ridge_cv.best_estimator_),
    ('gbr', gbr)
]


sr2 = StackingRegressor(
    estimators=estimators2,
    final_estimator=vr
)
sr2.fit(X_train, y_train)
```

[37]:
```
StackingRegressor(estimators=[('rfr', RandomForestRegressor(random_state=13)),
                              ('ridge', Ridge(alpha=10)),
                              ('gbr', GradientBoostingRegressor())],
                  final_estimator=VotingRegressor(estimators=[('rfr',
RandomForestRegressor(random_state=13)),
                                                              ('gbr',
GradientBoostingRegressor()),
                                                              ('lr',
LinearRegression())],
                                                  weights=[2, 3, 1]))
```

[38]:
```
y_pred_sr2 = sr2.predict(X_test)
y_pred_sr2
```

[38]:
```
array([24.3491363 , 27.11568059, 23.39061009, 17.05006586, 15.40976879,
       25.52060588, 18.04750935, 13.39328651, 18.64617906, 24.98338014,
       17.1792968 , 30.97858195, 28.10750468, 15.68440833, 19.64604119,
       27.63643889, 26.05606814, 30.67613107, 15.67300316, 12.33635145,
       15.40526841, 14.087848  , 25.02953828, 22.15264521, 10.96638522,
       33.17943801, 13.99024305, 35.98606902, 13.3395992 , 21.31222876,
       36.64232503, 28.26352216, 15.0655988 , 24.74528493, 32.06494113,
       37.00372476, 25.66198544, 13.83742911, 32.91868345, 13.83196459,
       33.0385216 , 23.3559947 , 15.62901099, 30.6175818 , 27.05611273,
       25.57012826, 21.10867599, 15.50163291, 15.26869649, 13.67442293,
       15.7415394 , 18.49101327, 34.04061254, 16.02565044, 38.78773653,
       13.52816628, 33.1378723 , 17.90159211, 18.37792041, 35.9752136 ,
       18.70332375, 20.73245054, 14.26120238, 37.42491304, 33.21332751,
       15.71597323, 25.43355876, 15.75085825, 32.73059787, 23.50424327,
       30.53905967, 20.94953693, 14.23724779, 15.62513511, 24.19958196,
       20.9596028 , 13.96885838, 37.38444735, 30.55691623, 19.2151501 ])
```

[39]:
```
root_mean_squared_error(y_test, y_pred_sr2)
```

[39]:
```
6.414412415181024
```

[40]:
```
r2_score(y_test, y_pred_sr2)
```

[40]:
```
0.8707180746579599
```

[41]:
```
estimators3 = [
    ('rfr', rfr),
    ('ridge', ridge_cv.best_estimator_),
    ('svr', SVR(C=1.0, kernel='linear')),
    ('random_forest',RandomForestRegressor())
]

sr3 = StackingRegressor(
    estimators=estimators3,
    final_estimator=Ridge(alpha=1.0)
)

param_grid_sr = {
    'random_forest__n_estimators':[50,100,250],
    'svr__C': [0.1, 1.0, 10.0],
    'final_estimator__alpha': [0.1,1.0,10.0]
}

sr_cv = RandomizedSearchCV(sr3, param_grid_sr, n_iter=5, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1)
sr_cv.fit(X_train, y_train)
```

[41]:
```
RandomizedSearchCV(cv=3,
                   estimator=StackingRegressor(estimators=[('rfr',
RandomForestRegressor(random_state=13)),
                                                           ('ridge',
                                                            Ridge(alpha=10)),
                                                           ('svr',
SVR(kernel='linear')),
                                                           ('random_forest',
RandomForestRegressor())],
                                               final_estimator=Ridge()),
                   n_iter=5, n_jobs=-1,
                   param_distributions={'final_estimator__alpha': [0.1, 1.0,
                                                                   10.0],
                                        'random_forest__n_estimators': [50, 100,
                                                                        250],
                                        'svr__C': [0.1, 1.0, 10.0]},
                   scoring='neg_mean_squared_error')
```

[42]:
```
y_pred_8 = sr_cv.predict(X_test)
y_pred_8
```

[42]:
```
array([24.45505141, 26.08064778, 25.377915  , 15.93300187, 15.24263626,
       25.45247833, 17.30640013, 12.365641  , 18.28087359, 25.39590536,
       17.09670375, 30.33733978, 26.29055207, 16.56544171, 19.26286061,
       27.02489251, 29.40600026, 32.61210673, 15.07380717, 11.0614695 ,
       15.61011049, 13.4696746 , 24.67148938, 23.53833227, 11.43036111,
       31.19528332, 13.48103115, 37.06619408, 12.66684219, 21.79112221,
       34.90552177, 28.92212115, 14.95776505, 25.75974317, 32.5812645 ,
       36.34849374, 24.79244634, 12.77105095, 32.95633545, 14.24149635,
       31.54489825, 24.29351523, 14.88776827, 30.01059792, 26.08003928,
       25.61700765, 21.68020634, 15.65541277, 14.90706114, 13.46447027,
       15.61558932, 18.61179576, 34.77888606, 16.10115523, 34.68386065,
       13.07459897, 31.41977482, 19.17668167, 17.95115591, 34.99862448,
       16.6652014 , 20.35133345, 12.5105491 , 35.28992113, 33.31986247,
       15.64511174, 24.53707771, 15.51719415, 31.25388316, 23.41547812,
       30.63741646, 21.42362772, 14.42679083, 14.19939658, 25.45697562,
       22.11579612, 13.25555966, 37.29781596, 30.58671278, 19.00212692])
```

[43]:
```
root_mean_squared_error(y_test, y_pred_8)
```

[43]:
```
5.580300867802961
```

[44]:
```
r2_score(y_test, y_pred_8)
```

[44]:
```
0.8875295204795366
```

[45]:
```
sr_cv.best_estimator_
```

[45]:
```
StackingRegressor(estimators=[('rfr', RandomForestRegressor(random_state=13)),
                              ('ridge', Ridge(alpha=10)),
                              ('svr', SVR(C=10.0, kernel='linear')),
                              ('random_forest', RandomForestRegressor())],
                  final_estimator=Ridge(alpha=0.1))
```