Stacking in Machine Learning

Last Updated : 11 Sep, 2025

Stacking is a ensemble learning technique where the final model known as the “stacked model" combines the predictions from multiple base models. The goal is to create a stronger model by using different models and combining them.

Architecture of Stacking

Stacking architecture is like a team of models working together in two layers to improve prediction accuracy. Each layer has a specific job and the process is designed to make the final result more accurate than any single model alone. It has two parts:

1. Base Models (Level-0)

These are the first models that directly learn from the original training data. You can think of them as the “helpers” that try to make predictions in their own way.

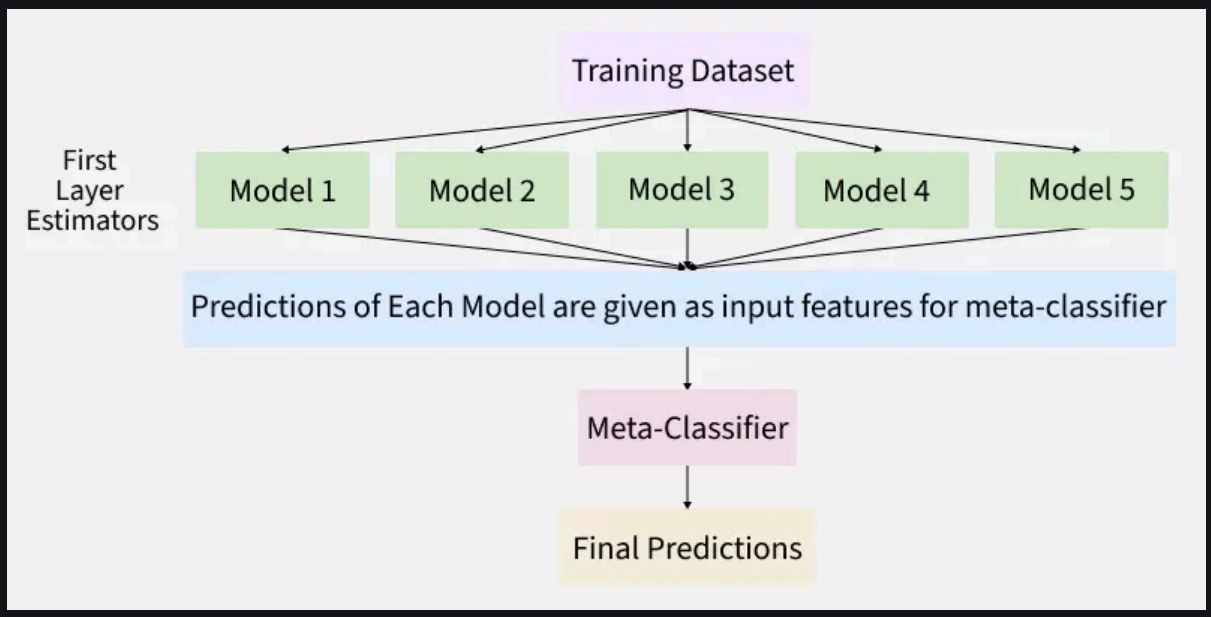
- Base models can be Decision Tree, Logistic Regression, Random Forest, etc.
- Each model is trained separately using the same training data.

2. Meta-Model (Level-1)

This is the final model that learns from the output of the base models instead of the raw data. Its job is to combine the base models predictions in a smart way to make the final prediction.

- A simple Linear Regression or Logistic Regression can act as a meta-model.

- It looks at the outputs of the base models and finds patterns in how they make mistakes or agree.



Stacking in Machine Learning

Working of Stacking

The process can be summarized in the following steps:

- **Start with training data**: We begin with the usual training data that contains both input features and the target output.
- **Train base models**: The base models are trained on this training data. Each model tries to make predictions based on what it learns.
- **Generate predictions**: After training the base models make predictions on new data called validation data or out-of-fold data. These predictions are collected.
- **Train meta-model**: The meta-model is trained using the predictions from the base models as new features. The target output stays the same and the meta-model learns how to combine the base model predictions.
- **Final prediction**: When testing the base models make predictions on new, unseen data. These predictions are passed to the meta-model which then gives the final prediction.

With stacking we can improve our models performance and its accuracy.

Implementation of Stacking

Lets see its implementation step by step:

Step 1: Importing the required Libraries

We will import pandas, matplotlib and scikit learn for data handling, visualization and modeling.

```
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

Step 2: Loading the Dataset

We will load the dataset into a pandas DataFrame and separate features from the target variable.

- **pd.read_csv():** Reads the dataset from a CSV file.
- **drop():** Removes the target column from features.
- **df['target']:** Selects the target column for prediction.

You can Download the dataset from this link Heart Dataset.

```
df = pd.read_csv('heart.csv')

X = df.drop('target', axis = 1)
y = df['target']

df.head()
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

Step 3: Splitting the Data into Training and Testing Sets

We will split the dataset into training and testing sets so we can train models and evaluate their performance.

- **train_test_split():** Splits data into train and test sets.
- **test_size = 0.2:** Specifies that 20% of the data should be used for testing, leaving 80% for training.
- **random_state = 42:** Ensures reproducibility by setting a fixed seed for random number generation.

```
X_train, X_test, y_train, y_test = train_test_sp
y, test_size = 0.2, random_state = 42)
```

Step 4: Standardizing the Data

We will standardize numerical features so they have a mean of 0 and standard deviation of 1. This helps some models perform better.

- **StandardScaler():** Standardizes features.
- **fit_transform():** Learns scaling parameters from training data and applies them.
- **transform()**: Applies learned scaling to test data.
- **var_transform:** Specifies the list of feature columns that need to be standardized.
- **X_train[var_transform]:** Applies the **fit_transform** method to standardize the selected columns in the training data.
- **X_test[var_transform]:** Applies the transform method to standardize the corresponding columns in the test data using the scaling parameters from the training data.

```
sc = StandardScaler()

var_transform = ['thalach', 'age', 'trestbps', 'oldpeak', '
X_train[var_transform] = sc.fit_transform(X_train[var_trans
X_test[var_transform] = sc.transform(X_test[var_transform])

X_train.head()
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
835	-0.585840	1	2	-0.779454	-1.935031	0	0	-1.019094	0	-0.210661	2	3	2
137	1.051477	0	0	2.741732	1.610634	0	1	0.202882	1	-0.912152	2	0	2
534	-0.040068	0	2	-1.347387	0.442176	0	0	0.770228	0	-0.912152	2	0	2
495	0.505705	1	0	0.186033	-0.222636	0	1	0.508376	0	-0.473720	1	0	3
244	-0.367531	1	2	-0.381900	-0.001032	1	0	0.726586	0	1.192321	1	0	2

Step 5: Building First Layer Estimators

We will create base models that will form the first layer of our stacking model. For this example we'll use K-Nearest Neighbors classifier and Naive Bayes classifier.

- **KNeighborsClassifier():** A model based on nearest neighbors.
- **GaussianNB():** A Naive Bayes classifier assuming Gaussian distribution.

```
KNC = KNeighborsClassifier()
NB = GaussianNB()
```

Step 6: Training and Evaluating KNeighborsClassifier

We will Train the KNN model and check its accuracy on the test set.

- **fit():** Trains the model.
- **predict():** Makes predictions on test data.


```
model_kNeighborsClassifier = KNC.fit(X_train, y_t i
pred_knc = model_kNeighborsClassifier.predict(X_test)
```

```
acc_knc = accuracy_score(y_test, pred_knc)
print('Accuracy Score of KNeighbors Classifier:', acc_knc *
100)
```

Output:

Accuracy Score of KNeighbors Classifier: 86.88524590163934

Step 7: Training and Evaluating Naive Bayes Classifier

Similarly, we will train the Naive Bayes model and check its accuracy.

```
model_NaiveBayes = NB.fit(X_train, y_train)
pred_nb = model_NaiveBayes.predict(X_test)

acc_nb = accuracy_score(y_test, pred_nb)
print('Accuracy of Naive Bayes Classifier:', acc_nb *
100)
```

Output:

Accuracy of Naive Bayes Classifier: 86.88524590163934

Step 8: Implementing the Stacking Classifier

Now, we will combine the base models using a Stacking Classifier. The meta-model will be a logistic regression model which will take the predictions of KNN and Naive Bayes as input.

- **StackingClassifier():** Combines base models and a meta-model.
- **classifiers:** List of base learners.
- **meta_classifier:** Model that learns from base learners' predictions.
- **use_probas=True:** Passes probability outputs to the meta-model instead of class labels.

```
base_learners = [
    KNeighborsClassifier(),
    GaussianNB()
]
```

```
meta_model = LogisticRegression()

stacking_model =
StackingClassifier(classifiers=base_learners,
meta_classifier=meta_model, use_probas=True)
```

Step 9: Training Stacking Classifier

Next we will rain the stacking classifier and evaluate its accuracy.

```
model_stack = stacking_model.fit(X_train, y_train)
pred_stack = model_stack.predict(X_test)

acc_stack = accuracy_score(y_test, pred_stack)
print('Accuracy Score of Stacked Model:', acc_stack *
100)
```

Output:

Accuracy Score of Stacked Model: 88.52459016393442

Both individual models (KNN and Naive Bayes) achieved an accuracy of approximately 86.88%, while the stacked model achieved an accuracy of around 88.52%. This shows that combining the predictions of multiple models using stacking can slightly improve overall performance compared to using a single model.

Advantages of Stacking

Here are some of the key advantages of stacking:

- **Better Performance:** Stacking often results in higher accuracy by combining predictions from multiple models making the final output more reliable.
- **Combines Different Models:** It allow to use various types of models like decision trees, logistic regression, SVM made from each model's unique strengths.
- **Reduces Overfitting:** When implemented with proper cross-validation it can reduce the risk of overfitting by balancing out the weaknesses of individual models.

- **Learns from Mistakes:** The meta-model is trained to recognize where base models go wrong and improves the final prediction by correcting those errors.
- **Customizable:** We can choose any combination of base and meta-models depending on our dataset and problem type making it highly flexible.

Limitations of Stacking

Stacking also have some limitations as well:

- **Complex to Implement**: Compared to simple models or even bagging or boosting, stacking requires more steps and careful setup.
- **Slow Training Time**: Since you're training multiple models plus a meta-model it can be slow and computationally expensive.
- **Hard to Interpret**: With multiple layers of models it becomes difficult to explain how the final prediction was made.
- **Risk of Overfitting**: If the meta-model is too complex or if there's data leakage it can overfit the training data.
- **Needs More Data**: It performs better when you have enough data, especially for training both base and meta-models effectively

Related Articles:

- *Ensemble Learning*
- *Types of Ensemble Learning*
- *Bagging vs Boosting in Machine Learning*.