What is Stacking in Machine Learning?

Stacking in machine learning is an ensemble algorithm used for prediction models where we can get efficient outputs.

Thus, some of the significant advantages of Stacking are enhanced accuracy and layered models having diversified trends.

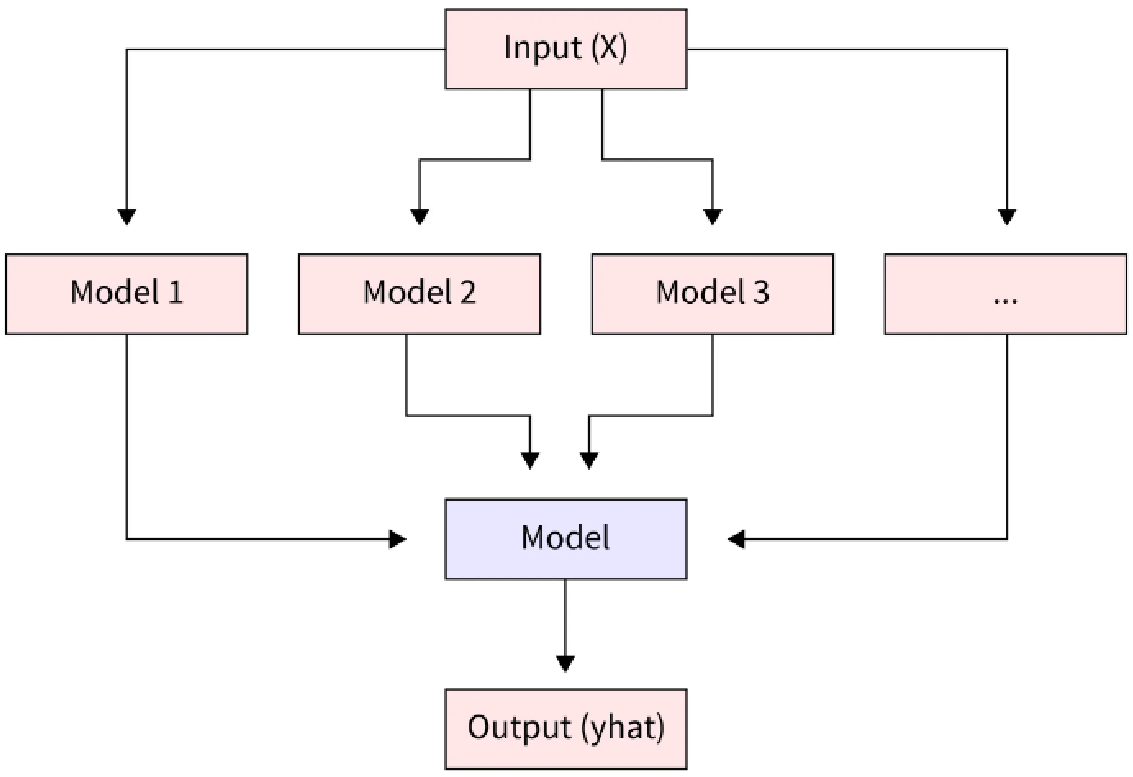
Stacking in Machine Learning

Stacking in machine learning is also known as **Stacking Generalisation**, which is a technique where all models aggregated are utilized according to their weights for producing an output which is a new model. As a result, this model has better accuracy and is stacked with other models to be used.

We can envision it as a **two-layer** model where the first layer incorporates all the models, and the second one is the prediction layer which renders output.

The principle is that you can always tackle a learning issue with various models that can learn a subset of the problem but not the whole problem space. This is where Stacking is used.

Architecture of a Stacking Model



We can use ensemble models for feature selection and data fusion and to improve function **approximation**, **Classification**, **regression**, **prediction**, etc.

There are three ensemble learning models - **Bagging, Boosting, and Stacking**. In this article, we will ponder over stacking.

To know more, check out the article on ensemble learning.

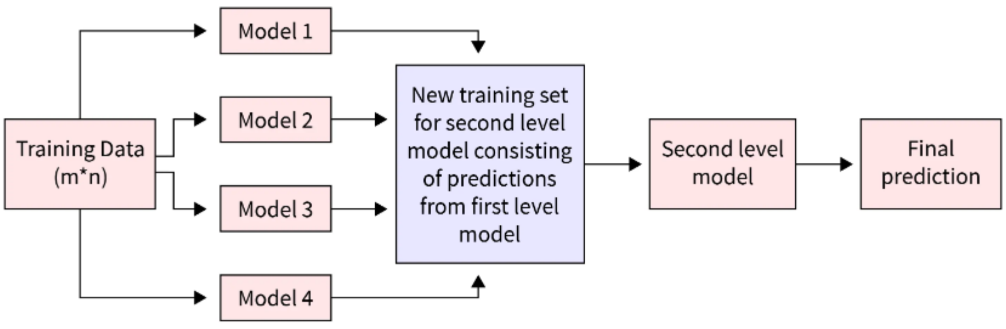
Stacking in Machine Learning

Stacking in machine learning is also known as **Stacking Generalisation**, which is a technique where all models aggregated are utilized according to their weights for producing an output which is a new model. As a result, this model has better accuracy and is stacked with other models to be used.

We can envision it as a **two-layer** model where the first layer incorporates all the models, and the second one is the prediction layer which renders output.

The principle is that you can always tackle a learning issue with various models that can learn a subset of the problem but not the whole problem space. This is where Stacking is used.

Architecture of a Stacking Model


Training Data (m*n)
Model 1
Model 2
Model 3
Model 4
New training set for second level model consisting of predictions from first level model
Second level model
Final prediction

Given below is the **architecture** of a Stacking model in machine learning -

- **Dataset** - The Dataset we work on. This Dataset is divided into **training** and **test** data.
- **Level 0 Models** - These are prediction-based models.
- **Level 0 Prediction** - The prediction that is the resultant of the above models is the level 0 prediction. This is based on the training data.
- **Level 1 Model** - This is the model which combines the prediction of all the level 0 or **base** models. It is also known as a **meta-model**.
- **Level 1 Prediction** - The level 1 model is trained on multiple predictions given by several base models, and then it optimally integrates the predictions of the base models on the testing data.

In stacking, an algorithm takes the outputs of sub-models as input and attempts to learn how to best combine the input predictions to make a better output prediction.

It may be helpful to think of the stacking procedure as having two levels: level 0 and level 1.

- **Level 0**: The level 0 data is the training dataset inputs and level 0 models learn to make predictions from this data.
- **Level 1**: The level 1 data takes the output of the level 0 models as input and the single level 1 model, or meta-learner, learns to make predictions from this data.

Steps to Implement Stacking Models

Now that we understand the architecture of the Stacking model in machine learning, we can understand how it works.

1. **Data** - The Dataset to be used is split into training and testing data into n folds. This is achieved by repeated **n-fold** cross-validation, a technique to guarantee the efficient performance of the model.
2. **Fitting data to base model** - Based on the above n-fold data, the first fold is assigned to the base model, and the output is generated. This is done for all n folds for all the base models.
3. **Level 1 model** - Now that we have the results for the base models, we train the level 1 model.
4. **Final prediction** - Predictions based on the level 1 model are used as the features for the model, and then they can be tested on the test data for the final results of the stack model.

How to Implement Classification with Stacking

Let us look at an example to work with Classification using Stacking on the **Wine Quality dataset**.

```
# Importing Pandas

import pandas as pd

# Importing libraries from SciKit Learn
# Importing models - Decision Tree, Random Forest, KNN, Logistic
Regression, Stacking
# Importing validation techniques like cross-validation

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbours import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# Importing Extreme Gradient Boosting Library for distributed gradient-
boosted decision tree
```

```
import xgboost

# Loading and printing the Dataset

df = pd.read_csv("/kaggle/input/wine-quality/winequalityN.csv")
df.head()
```

Output:-

```
# Understanding the dependent and independent attributes
# the number of unique values in "quality"

df.quality.unique()
```

Output:-

```
array([6, 5, 7, 8, 4, 3, 9])
# Splitting our Dataset into train and test

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

# Importing Preprocessing a package consisting of utility functions

from sklearn import preprocessing

# Label Encoding for 'type' feature

label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'

X['type']= label_encoder.fit_transform(X['type'])
X['type'].unique()
```

Output:-

```
array([1, 0])
# Removing null values from our Dataset

X['volatile acidity'].fillna(np.mean(X['volatile acidity']),
inplace=True)
X['citric acid'].fillna(np.mean(X['citric acid']), inplace=True)
X['residual sugar'].fillna(np.mean(X['residual sugar']), inplace=True)
X['chlorides'].fillna(np.mean(X['chlorides']), inplace=True)
X['pH'].fillna(np.mean(X['pH']), inplace=True)
X['sulphates'].fillna(np.mean(X['sulphates']), inplace=True)

# Checking for null values in our Dataset

X.isnull().sum()
```

Output:-

```
type                      0
```

```
fixed acidity           10
volatile acidity         0
citric acid              0
residual sugar           0
chlorides                0
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
pH                       0
sulphates                0
alcohol                  0
dtype: int64
# Understanding the dataset and column values

X.describe()
```

Output:-

```
# Cross Validation using N Fold technique

dtc =  DecisionTreeClassifier()
rfc = RandomForestClassifier()
knn =  KNeighborsClassifier()
xgb = xgboost.XGBClassifier()

# Printing the individual accuracy of the models

classifications = [decisiontree,randomforest,knn,xgb]
for algo in classifications:
    score = cross_val_score( algo,X,y,cv = 5,scoring = 'accuracy')
    print("The accuracy score of {} is:".format(algo),score.mean())
```

Output:-

```
The accuracy score of DecisionTreeClassifier() is: 0.3952607331083081
The accuracy score of RandomForestClassifier() is: 0.5077670397347072
The accuracy score of KNeighborsClassifier() is: 0.39741209214188433
The accuracy score of XGBClassifier(base_score=None, booster=None, 
callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, gamma=None,
              gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, 
max_bin=None,
              max_cat_to_onehot=None, max_delta_step=None, 
max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, predictor=None, random_state=None,
              reg_alpha=None, reg_lambda=None, ...)
# Implementing Stacking with the final layer being logistic regression

decisiontree =  DecisionTreeClassifier()
randomforest = RandomForestClassifier()
knn =  KNeighborsClassifier()
xgb = xgboost.XGBClassifier()
```

```
#list of the estimator

classificationres =
[('decisiontree',decisiontree),('randomforest',randomforest),('knn',knn),
('xgb',xgb)]

# Logistic Regression

lr = LogisticRegression()
stack_model = StackingClassifier( estimators =
classificationres,final_estimator = lr)
score = cross_val_score(stack_model,X,y,cv = 5,scoring = 'accuracy')

# Printing the final accuracy of the stacked model

print("The accuracy score of is:",score.mean())
```

Output:-

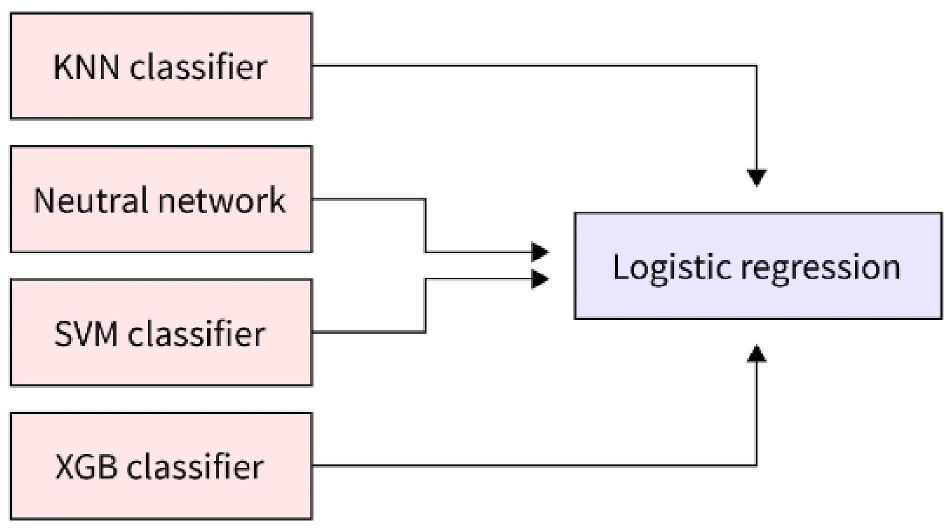
```
The accuracy score is: 0.907651447859299
```

Stacking for Regression

Regression is one of the most common and straightforward techniques to predict continuous outcomes. It maps out dependent and independent variables to compute the output.

When using Stacking, the **last layer** is always a computer with a regressor for final prediction. This is to compare the individual strength of the model with the final regressor using cross-validation.

The above example depicts using **logistic regression** as the **meta-layer** predictor.



Commonly Used Ensemble Techniques Related to Stacking

Various **ensemble techniques** can be used with **stacking** -

1. **Voting Ensembles -**

   This is the most fundamental style of stacking ensembling method. **This method works with individual models and then, based on the number of votes from individual prediction models, determines the result**. In simpler terms, every base model has an output that can be either positive or negative depending on the result since not all models accurately fit the algorithm. We get an aggregate positive or a negative for the final model based on these results.

   Compared to the simple stacking ensemble, this method uses statistics to combine the predictions.

2. **Weighted Average Ensemble -**

As the name suggests, this method uses the average weights of each model based on their predictions. This is slightly superior to the voting ensemble as it considers a wide range of models. We can also use a different dataset for **cross-validation** to get more efficiency.

3. **Blending Ensemble -**

   Similar to Stacking, **blending** is another type of machine learning ensemble that uses a third type of testing set - hold out Dataset, a small fragment of the primary Dataset to predict values using n-fold cross-validation.

4. **Super Learner Ensemble -**

   This can be considered a higher level of blending ensemble, where the only change is the change of the hold-out Dataset. Although, this Dataset is used explicitly with the meta or the level 1 model.