DSP 556 Assignment_4

Sheikh-Sedat Touray

Importing all the libraries

In [14]:
```
# Importing Libraries

import pandas as pd
import numpy as np
np.set_printoptions(formatter={'float_kind':"{:3.2f}".format})
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder as ohc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import StackingClassifier
```

Forest Cover Type Classification Problem

In [15]:
```
# Reading the forest cover type data set as a data frame
covert = pd.read_csv("covtype.csv")
```

In [16]:
```
# Splitting the dataset into features and target variables
x = covert.drop(["Cover_Type"], axis=1)
y = covert["Cover_Type"]
y = covert["Cover_Type"].astype("category")
```

In [17]:
```
# I am choosing 10000 observations and 25 columns
x,y = make_classification(n_samples=10000, n_features=25, random_state
=42)
# SPlit the training and testing sets
X_traint, X_testt, y_traint, y_testt = train_test_split(x, y, train_si
ze=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_traint = sc.fit_transform(X_traint)

# Use the same scaler to transform the test data
X_testt = sc.transform(X_testt)
```

In [18]:
```
#Initialize my random forest model
Rforest = RandomForestClassifier(min_samples_split=2, random_state=0)
# grid search
param_grid = {'max_depth': list(range(1,7)), 'n_estimators': list(rang
e(1,11)) }
#Grid search with 5 fold CV on the Rforest model
grid = GridSearchCV(Rforest, param_grid, cv=5)
#fit the grid on the training set
%time grid.fit(X_traint, y_traint)
#Print out the best Parameters
print("Grid Search: best parameters: {}".format(grid.best_params_))
```

```
CPU times: user 22.5 s, sys: 35 ms, total: 22.5 s
Wall time: 23 s
Grid Search: best parameters: {'max_depth': 6, 'n_estimators': 10}
```

In [26]:
```
# accuracy of best model
best_model = grid.best_estimator_
#Use our best model to predict on unseen data
predictt_y = best_model.predict(X_testt)
#Compute the accuracy score and Print it out
acct = accuracy_score(y_testt, predictt_y)
print("Accuracy of RForest Model: {:3.2f}".format(acct))
```

```
Accuracy of RForest Model: 0.92
```

For the Random forest we trained our model for about 23 secs and had an accuracy score of 92.There's possible not that much of data leakage because the model is great at shuffling and randomizing the samples. However, the only leakages we might incur is in the scaling of the data

Bagging Classifier

In [20]:
```
# I am choosing 5000 observations and 20 columns
x,y = make_classification(n_samples=5000, n_features=20, random_state=
42)
# SPlit the training and testing sets
X_trainb, X_testb, y_trainb, y_testb = train_test_split(x, y, train_si
ze=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_trainb = sc.fit_transform(X_trainb)

# Use the same scaler to transform the test data
X_testb = sc.transform(X_testb)
```

In [23]:
```
# Initialize the bagging classifier
bag = BaggingClassifier(random_state=3)
# grid search
param_gridb = {'base_estimator': [KNeighborsClassifier(n_neighbors=1),
DecisionTreeClassifier()], 'n_estimators': list(range(1,11)) }
#Perform gridsearch with 5-fold cross validation of the bag model
grid1 = GridSearchCV(bag, param_gridb, cv=5)
# fit the grid on the training set
%time grid1.fit(X_trainb, y_trainb)
#Print out the best parameters
print("Grid Search: best parameters: {}".format(grid1.best_params_))
```

```
CPU times: user 27.2 s, sys: 18.2 s, total: 45.4 s
Wall time: 31.5 s
Grid Search: best parameters: {'base_estimator': DecisionTreeClassifie
r(), 'n_estimators': 9}
```

In [28]:
```
# accuracy of best model
best_bag = grid1.best_estimator_
#Use our best bag model to predict on unseen data
predictb_y = best_bag.predict(X_testb)
#Compute the accuracy score of our best model
accb = accuracy_score(y_testb, predictb_y)
#Print out the score
print("Accuracy Score of Bagging Classifier: {:3.2f}".format(accb))
```

```
Accuracy Score of Bagging Classifier: 0.90
```

When I used a sample size of 3000 it trained for 11 secs and gave an Accuracy of 98%. So I decided to use a larger sample size of 5000 observations and it trained for 27 secs with a accuracy of 90%.

Data leakage scenario: Data leakage could happen If the base models in the ensemble are trained on the same dataset or if there is any overlap in the samples used to train different models, it can lead to data leakage.

My Prevention: I split the data into test and train subsets before any preprocessing. I also did on use fit and fit_transform on the test subset.

AdaBoost Classifier

In [30]:
```
# I am choosing 1000 observations and 20 columns
x,y = make_classification(n_samples=1000, n_features=20, random_state=
42)
# SPlit the training and testing sets
X_traina, X_testa, y_traina, y_testa = train_test_split(x, y, train_si
ze=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_traina = sc.fit_transform(X_traina)

# Use the same scaler to transform the test data
X_testa = sc.transform(X_testa)
```

In [31]:
```
# Initialize the Adaboost classifier
boost = AdaBoostClassifier(random_state=3)
# grid search
param_grida = {'base_estimator': [SVC(probability=True, kernel='linea
r'),DecisionTreeClassifier(random_state=3)], 'n_estimators': list(rang
e(1,11))}
# Do a grid search on the adabosst with 5 fold CV
grida = GridSearchCV(boost, param_grida, cv=5)
#fit the grid on the training set and time it
%time grida.fit(X_traina, y_traina)
#Print out the best parameters and the timing
print("Grid Search: best parameters: {}".format(grida.best_params_))
```

```
CPU times: user 32.2 s, sys: 0 ns, total: 32.2 s
Wall time: 33.6 s
Grid Search: best parameters: {'base_estimator': DecisionTreeClassifie
r(random_state=3), 'n_estimators': 1}
```

In [32]:
```
# accuracy of best model
best_boost = grida.best_estimator_
# Use our best model to predict on test data
predicta_y = best_boost.predict(X_testa)
#Compute the accuracy score
acca = accuracy_score(y_testa, predicta_y)
#Print out the score
print("Accuracy Score of Adaboost: {:3.2f}".format(acca))
```

```
Accuracy Score of Adaboost: 0.83
```

I had an Accuracy of 89% with a sample size of 2000, however, it 2 minutes to train. when I reduced the sample size to 1000 with the same size of features the training time was reduced to 32 secs, however, the accuracy score went down to 84%.

Data leakage Scenario: In boosting, each base model corrects the errors of its predecessor. If there is information leakage between the models, where the next model learns from the mistakes of the previous ones on the same set of data, it can lead to poor generalization and overfitting.

Algorithm leakage Prevention: The algorithm depends on re-weighting of misclassified samples. Also our careful splitting before processing and adhering to the scaling rule of not applying fit or fit_transform to the test subset could help prevent leakage.

GradientBoostingClassifier

In [5]:
```
# I am choosing 8000 observations and 25 columns
x,y = make_classification(n_samples=2000, n_features=20, random_state=
42)
# SPlit the training and testing sets
X_trainga, X_testga, y_trainga, y_testga = train_test_split(x, y, trai
n_size=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_trainga = sc.fit_transform(X_trainga)

# Use the same scaler to transform the test data
X_testga = sc.transform(X_testga)
```

In [6]:
```
gboost = GradientBoostingClassifier(random_state=0)
# grid search
param_gridga = {'criterion': ['friedman_mse','mse'], 'n_estimators': l
ist(range(1,11)), 'max_depth': [1,3,5,7]}
#5 fold CV with grid search
gridga = GridSearchCV(gboost, param_gridga, cv=5)
#Fit the grid on my training set and time it
%time gridga.fit(X_trainga, y_trainga)
#Print out my best parameters
print("Grid Search: best parameters: {}".format(gridga.best_params_))
```

```
CPU times: user 29.3 s, sys: 0 ns, total: 29.3 s
Wall time: 31.7 s
Grid Search: best parameters: {'criterion': 'friedman_mse', 'max_dept
h': 3, 'n_estimators': 5}
```

In [33]:
```
# accuracy of best model
best_gboost = gridga.best_estimator_
#using our best model to predict on unseen data
predictga_y = best_gboost.predict(X_testga)
#Compute and print the accuracy score of the classification
accga = accuracy_score(y_testga, predictga_y)
print("Accuracy Score of GradientBoost: {:3.2f}".format(accga))
```

```
Accuracy Score of GradientBoost: 0.90
```

With the gradient boosting I got a 90% accuracy from training of about 29 secs and some of the best parameters include max_depth of 3 meaning the best tree had 3 terminal nodes, the number of base estmators in the final model is 5 and the best criterion is 'friedman_mse' and this was expected because it is generally the best as it can provide a better approximation in most cases.

StackingClassifier

In [10]:
```
# I am choosing 500 observations and 20 columns
x,y = make_classification(n_samples=500, n_features=20, random_state=4
2)
# SPlit the training and testing sets
X_trainst, X_testst, y_trainst, y_testst = train_test_split(x, y, trai
n_size=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_trainst = sc.fit_transform(X_trainst)

# Use the same scaler to transform the test data
X_testst = sc.transform(X_testst)
```

In [11]:
```
#estimators =
# Define base models
base_modelc = [
    ('forestc', RandomForestClassifier(random_state=0)),
    ('gradbc', GradientBoostingClassifier(random_state=0))
]

# Define the final-model
final_modelc = LogisticRegression()

# Define the StackingRegressor
stkc = StackingClassifier(estimators=base_modelc, final_estimator=fina
l_modelc, cv=5)

# Define the hyperparameters to tune
param_gridstkc = {
    'forestc__n_estimators': [10, 20],

    'final_estimator__C': [0.1, 1.0, 4.0]
}
```

In [12]:
```
#Perform gridsearch on the stacking classifier with 5 fold CV
gridstkc = GridSearchCV(stkc, param_gridstkc, cv=5)
#fit the grid on the training set and time it
%time gridstkc.fit(X_trainst, y_trainst)
#Print out the best parameters
print("Grid Search: best parameters: {}".format(gridstkc.best_params
_))
```

```
CPU times: user 55.2 s, sys: 0 ns, total: 55.2 s
Wall time: 57.1 s
Grid Search: best parameters: {'final_estimator__C': 4.0, 'forestc__n_
estimators': 10}
```

In [34]:
```
# accuracy of best model
best_stk = gridstkc.best_estimator_
#Use our best model to predict on test data
predictstk_y = best_stk.predict(X_testst)
#Compute the Accuracy score
accstk = accuracy_score(y_testst, predictstk_y)
#Print the score
print("Accuracy Score of Stacking Classifier: {:3.2f}".format(accstk))
```

```
Accuracy Score of Stacking Classifier: 0.93
```

Here i used a smaller sample size because of the number of hyperparameters it was taking a long time to train hence I played with the tuning parameters also until I was able to capture enough of them (parameters) in the model and a somewhat bearable runtime of 55 secs.

Housing Data Ensemble Regression Problems

In [37]:
```
# Importing more libraries from Sci-kit learn
from sklearn.datasets import make_regression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
```

In [38]:
```
# Reading the housing data as a data frame
housing = pd.read_csv('ames.csv')
```

In [39]:
```
# Looking at the first 5 rows and all the columns
housing.head()
```

Out[39]:

	MS_SubClass	MS_Zoning	Lot_Frontage	Lot_Area	Street	
0	One_Story_1946_and_Newer_All_Styles	Residential_Low_Density	141	31770	Pave	N
1	One_Story_1946_and_Newer_All_Styles	Residential_High_Density	80	11622	Pave	N
2	One_Story_1946_and_Newer_All_Styles	Residential_Low_Density	81	14267	Pave	N
3	One_Story_1946_and_Newer_All_Styles	Residential_Low_Density	93	11160	Pave	N
4	Two_Story_1946_and_Newer	Residential_Low_Density	74	13830	Pave	N

5 rows × 81 columns

In [40]:
```
# Selecting just the numerical data types to avoid encoding
housing2 = housing.select_dtypes(include='number')
#This is to see more of what is in the dataset
housing2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 35 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Lot_Frontage        2930 non-null   int64  
 1   Lot_Area            2930 non-null   int64  
 2   Year_Built          2930 non-null   int64  
 3   Year_Remod_Add      2930 non-null   int64  
 4   Mas_Vnr_Area        2930 non-null   int64  
 5   BsmtFin_SF_1        2930 non-null   int64  
 6   BsmtFin_SF_2        2930 non-null   int64  
 7   Bsmt_Unf_SF         2930 non-null   int64  
 8   Total_Bsmt_SF       2930 non-null   int64  
 9   First_Flr_SF        2930 non-null   int64  
 10  Second_Flr_SF       2930 non-null   int64  
 11  Low_Qual_Fin_SF     2930 non-null   int64  
 12  Gr_Liv_Area         2930 non-null   int64  
 13  Bsmt_Full_Bath      2930 non-null   int64  
 14  Bsmt_Half_Bath      2930 non-null   int64  
 15  Full_Bath           2930 non-null   int64  
 16  Half_Bath           2930 non-null   int64  
 17  Bedroom_AbvGr       2930 non-null   int64  
 18  Kitchen_AbvGr       2930 non-null   int64  
 19  TotRms_AbvGrd       2930 non-null   int64  
 20  Fireplaces          2930 non-null   int64  
 21  Garage_Cars         2930 non-null   int64  
 22  Garage_Area         2930 non-null   int64  
 23  Wood_Deck_SF        2930 non-null   int64  
 24  Open_Porch_SF       2930 non-null   int64  
 25  Enclosed_Porch      2930 non-null   int64  
 26  Three_season_porch  2930 non-null   int64  
 27  Screen_Porch        2930 non-null   int64  
 28  Pool_Area           2930 non-null   int64  
 29  Misc_Val            2930 non-null   int64  
 30  Mo_Sold             2930 non-null   int64  
 31  Year_Sold           2930 non-null   int64  
 32  Sale_Price          2930 non-null   int64  
 33  Longitude           2930 non-null   float64
 34  Latitude            2930 non-null   float64
dtypes: float64(2), int64(33)
memory usage: 801.3 KB
```

In [41]:
```
# Splitting the data set into predictors and a response
feat = housing2.drop(['Sale_Price'], axis = 1)
targ = housing2['Sale_Price']
```

In [42]:
```
# making a subset of the data to avoid long run times so we random sam
ple
#But here I used all the features since it is not as much as the fores
t data
feat,targ = make_regression(n_samples=2000, n_features=35, random_stat
e=42)
#split into 70% training and 30% testing
X_trainh, X_testh, y_trainh, y_testh = train_test_split(feat,targ, tra
in_size=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_trainh = sc.fit_transform(X_trainh)

# Use the same scaler to transform the test data
X_testh = sc.transform(X_testh)
```

In [43]:
```
# decision trees
Rforest = RandomForestRegressor(random_state=0)

# grid search
param_gridf = {'max_depth': list(range(1,7)), 'n_estimators': list(ran
ge(1,11)) }
gridf = GridSearchCV(Rforest, param_gridf, cv=5)
#time the output of the best parameters and what they are
%time gridf.fit(X_trainh, y_trainh)
#Print the best parameters
print("Grid Search: best parameters: {}".format(gridf.best_params_))
```

```
CPU times: user 24.8 s, sys: 0 ns, total: 24.8 s
Wall time: 1min 15s
Grid Search: best parameters: {'max_depth': 6, 'n_estimators': 10}
```

In [44]:
```
# accuracy of best model
best_modelfr = gridf.best_estimator_
# use our best model to predict on test data
predictfr_y = best_modelfr.predict(X_testh)
#compute the mean squared error and r2  score
msef = mean_squared_error(y_testh, predictfr_y)
r2f = r2_score(y_testh, predictfr_y)
#Print out the metrics
print("MSE of RForest: {:3.2f}".format(msef))
print("R2 Score of RForest: {:3.2f}".format(r2f))
```

```
MSE of RForest: 11830.80
R2 Score of RForest: 0.71
```

The mean squared error measures the squared distance between what my predictor predicted and the actual value and by looking at our MSE value of 11830, it is not a bad prediction considering the price of the houses are in the hundred thousands. Our coefficient of determination of 0.71 is very good also because it how well the data fit my model.

Bagging Regressor

In [46]:
```
# making a subset of the data to avoid long run times so we random sam
ple
#But here I used all the features since it is not as much as the fores
t data
feat,targ = make_regression(n_samples=2930, n_features=35, random_stat
e=42)
#split into 70% training and 30% testing
X_trainbg, X_testbg, y_trainbg, y_testbg = train_test_split(feat,targ,
train_size=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_trainbg = sc.fit_transform(X_trainbg)

# Use the same scaler to transform the test data
X_testbg = sc.transform(X_testbg)
```

In [47]:
```
bagr = BaggingRegressor(base_estimator = DecisionTreeRegressor(),rando
m_state=0)
# grid search
param_gridbr = {
    'n_estimators': [5, 10, 20],
    'max_features': [0.5, 0.7, 1.0],
    'base_estimator__max_depth': [None, 5, 10]
}
#Perform grid search and 5-fold CV
gridbr = GridSearchCV(bagr, param_gridbr, cv=5)
#Fit the grid on training set and time it
%time gridbr.fit(X_trainbg, y_trainbg)
#Print out the best parameters and time
print("Grid Search: best parameters: {}".format(gridbr.best_params_))
```

```
CPU times: user 43.1 s, sys: 597 ms, total: 43.7 s
Wall time: 45 s
Grid Search: best parameters: {'base_estimator__max_depth': None, 'max
_features': 1.0, 'n_estimators': 20}
```

In [48]:
```
# accuracy of best model
best_bagr = gridbr.best_estimator_
#use our best model to predict on test data
predictbr_y = best_bagr.predict(X_testbg)
#Compute the metrics
msebr = mean_squared_error(y_testbg, predictbr_y)
r2br = r2_score(y_testbg, predictbr_y)
#Print out the metrics
print("MSE of Bagging Regressor: {:3.2f}".format(msebr))
print("R2 Score of Bagging Regressor: {:3.2f}".format(r2br))
```

```
MSE of Bagging Regressor: 4045.64
R2 Score of Bagging Regressor: 0.86
```

The mean squared error measures the squared distance between what my predictor predicted and the actual value and by looking at our MSE value of 4046, it is not a bad prediction considering the price of the houses are in the hundred thousands. Our coefficient of determination of 0.86 is very good also because it how well the data fit my model. This is my best performing model so far on this dataset!

AdaBoost Regressor

In [50]:
```
# making a subset of the data to avoid long run times so we random sam
ple
#But here I used all the features since it is not as much as the fores
t data
feat,targ = make_regression(n_samples=2000, n_features=35, random_stat
e=0)
#split into 70% training and 30% testing
X_trainarg, X_testarg, y_trainarg, y_testarg = train_test_split(feat,t
arg, train_size=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_trainarg = sc.fit_transform(X_trainarg)

# Use the same scaler to transform the test data
X_testarg = sc.transform(X_testarg)
```

In [51]:
```
rboost = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(),rand
om_state=0)
# grid search
param_gridarg = {
    'n_estimators': [5, 10, 20],
    'learning_rate': [0.00001, 0.0001, 0.001]

}
#Perform gridsearch and 5-fold CV
gridada = GridSearchCV(rboost, param_gridarg, cv=5)
#fit the grid on the training set and time it
%time gridada.fit(X_trainarg, y_trainarg)
#Print out the best parameters
print("Grid Search: best parameters: {}".format(gridada.best_params_))
```

```
CPU times: user 20.8 s, sys: 174 ms, total: 21 s
Wall time: 21.9 s
Grid Search: best parameters: {'learning_rate': 0.0001, 'n_estimator
s': 20}
```

In [52]:
```
# accuracy of best model
best_boostarg = gridada.best_estimator_
#USe our best model to predict on test data
predictarg_y = best_boostarg.predict(X_testarg)
#Compute the metrics
msearg = mean_squared_error(y_testarg, predictarg_y)
r2arg = r2_score(y_testarg, predictarg_y)
#Print out the metrics
print("MSE of Adaboost: {:3.2f}".format(msearg))
print("R2 Score of Adaboost: {:3.2f}".format(r2arg))
```

```
MSE of Adaboost: 5478.63
R2 Score of Adaboost: 0.76
```

This model performed better than the forest, however, it did not do as well as the Bagging Regressor.

Gradient Boosting Regressor

In [53]:
```
# making a subset of the data to avoid long run times so we random sam
ple
#But here I used all the features since it is not as much as the fores
t data
feat,targ = make_regression(n_samples=2000, n_features=35, random_stat
e=42)
#split into 70% training and 30% testing
X_traingb, X_testgb, y_traingb, y_testgb = train_test_split(feat,targ,
train_size=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_traingb = sc.fit_transform(X_traingb)

# Use the same scaler to transform the test data
X_testgb = sc.transform(X_testgb)
```

In [54]:
```
rgboost = GradientBoostingRegressor(random_state=0)
# Define the parameters
param_gridgb = {'criterion': ['friedman_mse','mse'], 'n_estimators': l
ist(range(1,11)), 'max_depth': [1,3,5,7]}
# Grid search with 5-fold CV
gridgb = GridSearchCV(rgboost, param_gridgb, cv=5)
#fit the grid on the training set and time it
%time gridgb.fit(X_traingb, y_traingb)
#Print out the best parameters
print("Grid Search: best parameters: {}".format(gridgb.best_params_))
```

```
CPU times: user 47.9 s, sys: 0 ns, total: 47.9 s
Wall time: 59 s
Grid Search: best parameters: {'criterion': 'mse', 'max_depth': 7, 'n_
estimators': 10}
```

In [55]:
```
# accuracy of best model
best_gboost = gridgb.best_estimator_
#Use our best model to predict on test data
predictgb_y = best_gboost.predict(X_testgb)
#Compute the metrics
accga = mean_squared_error(y_testgb, predictgb_y)
r2ga = r2_score(y_testgb, predictgb_y)
#Print out the metrics
print("MSE of GradientBoost: {:3.2f}".format(accga))
print("R2 of GradientBoost: {:3.2f}".format(r2ga))
```

```
MSE of GradientBoost: 16075.36
R2 of GradientBoost: 0.61
```

This ensemble above did not perform as well as the previous models. However, the results are somewhat decent and could possibly be improved through further tuning of the hyperparameters.

Stacking Regressor

In [57]:
```
# making a subset of the data to avoid long run times so we random sam
ple
#But here I used all the features since it is not as much as the fores
t data
feat,targ = make_regression(n_samples=500, n_features=35, random_state
=42)
#split into 70% training and 30% testing
X_trainstr, X_teststr, y_trainstr, y_teststr = train_test_split(feat,t
arg, train_size=0.7, test_size=0.3, random_state=3)

# Initialize the scaler
sc = StandardScaler()

# Fit the scaler on the training data and transform it
X_trainstr = sc.fit_transform(X_trainstr)

# Use the same scaler to transform the test data
X_teststr = sc.transform(X_teststr)
```

In [61]:
```
#estimators =
# Define base models
base_models = [
    ('forest', RandomForestRegressor(random_state=42)),
    ('gradb', GradientBoostingRegressor(random_state=42))
]

# Define the meta-model
final_model = LinearRegression()

# Define the StackingRegressor
stkr = StackingRegressor(estimators=base_models, final_estimator=final
_model, cv=5)

# Define the hyperparameters to tune
param_gridstkr = {
    'forest__n_estimators': [10,20],
    'final_estimator__normalize': [True, False]
}
```

In [62]:
```
#Grid Search and 5-fold CV
gridstkr = GridSearchCV(stkr, param_gridstkr, cv=5)
#Fit the grid on training set and time it
%time gridstkr.fit(X_trainstr, y_trainstr)
#Print out the best parameters
print("Grid Search: best parameters: {}".format(gridstkr.best_params
_))
```

```
CPU times: user 51.2 s, sys: 0 ns, total: 51.2 s
Wall time: 53.7 s
Grid Search: best parameters: {'final_estimator__normalize': True, 'fo
rest__n_estimators': 20}
```

In [63]:
```
# accuracy of best model
best_stkr = gridstkr.best_estimator_
#Use our best model to predict on test data
predictstkr_y = best_stkr.predict(X_teststr)
#Compute metrics
msestkr = mean_squared_error(y_teststr, predictstkr_y)
r2stkr = r2_score(y_teststr, predictstkr_y)
#Print out the metrics
print("MSE of Stacking Regressor: {:3.2f}".format(msestkr))
print("R2 of Stacking Regressor: {:3.2f}".format(r2stkr))
```

```
MSE of Stacking Regressor: 5407.31
R2 of Stacking Regressor: 0.85
```

The stacking regressor is my best ensemble considering we sampled only 1/6 of the data and it almost performed as well as the ensemble in which we used the entire dataset. The only reason I could not sample a larger size was due to the amount of time it took to train the samples was very long when a larger sample size was chosen.

Summary: I was split the data into training and testing sets before any processing steps to prevent data leakage. Also, I am very cautious not to apply fit or fit_transform to the test set. Also any information derived from entire data including the test set is calculated solely based on the training data. And FInally, When using cross-validation, make sure that each fold's training set is used exclusively for training, and the validation set is kept separate.

- Also outwardly looking at our ensembles and results we can observe that there is hardly any bias in the models because there are no overfittings and underfittings and the variances are not very bad and could perhaps be reduced by sampling more data and fine tuning the parameters.Maybe this is so because mixture models and ensemble learning mitigates this issue of variance bias tradeoff. Because boosting combines many "Weak" (high bias) models in an ensemble to collectively lower the bias of the individual models while bagging combines "Strong" learners in a way that reduces variance like we have seen in my bagging regressor.
  One must also be cautious not to overly fine tune parameters to avoid overfitting.

In [ ]: