Lazy Predict

- While building machine learning models we are not sure which algorithm should work well with the given dataset, hence we end up trying many models and keep iterating until we get proper accuracy. Have you ever thought about getting all the basic algorithms at once to predict for model performance ?
- Ans is LazyPredict. It is a module helpful for this purpose. LazyPredict will generate all the basic machine learning algorithms' performances on your model. Along with the accuracy score, LazyPredict provides certain evaluation metrics and the time taken by each model.

What is Lazy Predict ?

- LazyPredict is an open-source Python library that automates the model training pipeline and speeds up the workflow. LazyPredict trains around 30 classification models for a classification dataset and trains around 40 regression models for a regression dataset.
- LazyPredict returns with the trained models along with its performance metric without writing much code. One can compare the performance metrics of each model and tune the best model to further improve the performance.

Parameters used in LazyRegressor ():

```
    verbose  – by default 0
    ignore_warning – by default set to True, to avoid warning messages for any kin
d of discrepancy in generating models
    custom_metric – by default None, can be set to custom metrics if defined
    predictions – by default False, if set to True it'll return predictions based
 on each model.
    random_state by default is set to 42.
    Note that all of these parameters are optional, if not defined they will take
 the default values.
```

In [1]:

```
# ! pip install --user lazypredict
```

Classification

In [2]:

```
# Import Libraries
import warnings
warnings.simplefilter('ignore')

from lazypredict.Supervised import LazyClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

In [3]:

```
# Load dataset
data = load_breast_cancer()
X = data.data
y= data.target
```

In [4]:

```
# Data split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=.5,random_state =123)
```

In [5]:

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[5]:

```
((284, 30), (285, 30), (284,), (285,))
```

In [6]:

```
# Defines and builds the lazyclassifier
clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
models,predictions = clf.fit(X_train, X_test, y_train, y_test)
```

```
100%|██████████████████████████████████████████████████████████████████████
███████████| 29/29 [00:02<00:00, 11.85it/s]
```

In [7]:

```
# Prints the model performance
models
```

Out[7]:

	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
Model					
LinearSVC	0.99	0.99	0.99	0.99	0.03
Perceptron	0.99	0.98	0.98	0.99	0.02
LogisticRegression	0.99	0.98	0.98	0.99	0.03
SVC	0.98	0.98	0.98	0.98	0.03
XGBClassifier	0.98	0.98	0.98	0.98	0.57
LabelPropagation	0.98	0.97	0.97	0.98	0.03
LabelSpreading	0.98	0.97	0.97	0.98	0.03
BaggingClassifier	0.97	0.97	0.97	0.97	0.11
PassiveAggressiveClassifier	0.98	0.97	0.97	0.98	0.03
SGDClassifier	0.98	0.97	0.97	0.98	0.03
RandomForestClassifier	0.97	0.97	0.97	0.97	0.35
CalibratedClassifierCV	0.98	0.97	0.97	0.98	0.06
QuadraticDiscriminantAnalysis	0.96	0.97	0.97	0.97	0.03
ExtraTreesClassifier	0.97	0.96	0.96	0.97	0.25
RidgeClassifierCV	0.97	0.96	0.96	0.97	0.03
LGBMClassifier	0.96	0.96	0.96	0.96	0.10
RidgeClassifier	0.97	0.96	0.96	0.97	0.03
AdaBoostClassifier	0.96	0.96	0.96	0.96	0.40
KNeighborsClassifier	0.96	0.96	0.96	0.96	0.06
BernoulliNB	0.95	0.95	0.95	0.95	0.03
LinearDiscriminantAnalysis	0.96	0.95	0.95	0.96	0.03
GaussianNB	0.95	0.95	0.95	0.95	0.03
NuSVC	0.95	0.94	0.94	0.95	0.03
ExtraTreeClassifier	0.94	0.93	0.93	0.94	0.02
NearestCentroid	0.95	0.93	0.93	0.95	0.02
DecisionTreeClassifier	0.93	0.93	0.93	0.93	0.02
DummyClassifier	0.53	0.50	0.50	0.53	0.02

Regression

In [8]:

```
from lazypredict.Supervised import LazyRegressor
from sklearn import datasets
from sklearn.utils import shuffle
import numpy as np
```

In [9]:

```
boston = datasets.load_boston()
X, y = shuffle(boston.data, boston.target, random_state=13)
X = X.astype(np.float32)
```

In [10]:

```
offset = int(X.shape[0] * 0.9)

X_train, y_train = X[:offset], y[:offset]
X_test, y_test = X[offset:], y[offset:]
```

In [11]:

```
reg = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric=None)
models, predictions = reg.fit(X_train, X_test, y_train, y_test)

print(models)
```

```
100%|██████████████████████████████████████████████████████████████████████
███████████| 42/42 [00:04<00:00,  9.56it/s]

                               Adjusted R-Squared  R-Squared  RMSE  Time T
aken
Model
SVR                                          0.83       0.88  2.62
0.03
BaggingRegressor                             0.83       0.88  2.63
0.07
NuSVR                                        0.82       0.86  2.76
0.02
RandomForestRegressor                        0.81       0.86  2.78
0.55
XGBRegressor                                 0.81       0.86  2.79
0.12
GradientBoostingRegressor                    0.81       0.86  2.84
0.24
ExtraTreesRegressor                          0.79       0.84  2.98
0.34
AdaBoostRegressor                            0.78       0.83  3.04
0.18
HistGradientBoostingRegressor                0.77       0.83  3.06
0.64
PoissonRegressor                             0.77       0.83  3.11
0.02
LGBMRegressor                                0.77       0.83  3.11
0.10
KNeighborsRegressor                          0.77       0.83  3.12
0.02
DecisionTreeRegressor                        0.65       0.74  3.79
0.02
MLPRegressor                                 0.65       0.74  3.80
1.05
HuberRegressor                               0.64       0.74  3.84
0.04
GammaRegressor                               0.64       0.73  3.88
0.03
LinearSVR                                    0.62       0.72  3.96
0.02
RidgeCV                                      0.62       0.72  3.97
0.02
BayesianRidge                                0.62       0.72  3.97
0.02
Ridge                                        0.62       0.72  3.97
0.01
TransformedTargetRegressor                   0.62       0.72  3.97
0.01
LinearRegression                             0.62       0.72  3.97
0.02
ElasticNetCV                                 0.62       0.72  3.98
0.16
LassoCV                                      0.62       0.72  3.98
0.11
```

```
LassoLarsIC                                 0.62        0.72  3.98
0.02
LassoLarsCV                                 0.62        0.72  3.98
0.04
Lars                                        0.61        0.72  3.99
0.02
LarsCV                                      0.61        0.71  4.02
0.05
SGDRegressor                                0.60        0.70  4.07
0.02
TweedieRegressor                            0.59        0.70  4.12
0.02
GeneralizedLinearRegressor                  0.59        0.70  4.12
0.03
ElasticNet                                  0.58        0.69  4.16
0.02
Lasso                                       0.54        0.66  4.35
0.02
RANSACRegressor                             0.53        0.65  4.41
0.14
OrthogonalMatchingPursuitCV                 0.45        0.59  4.78
0.03
PassiveAggressiveRegressor                  0.37        0.54  5.09
0.02
GaussianProcessRegressor                    0.23        0.43  5.65
0.06
OrthogonalMatchingPursuit                   0.16        0.38  5.89
0.01
ExtraTreeRegressor                          0.08        0.32  6.17
0.02
DummyRegressor                             -0.38       -0.02  7.56
0.02
LassoLars                                  -0.38       -0.02  7.56
0.02
KernelRidge                               -11.50       -8.25 22.74
0.03
```

Conclusion

LazyPredict would be very handy for selecting the more accurate model for the dataset being used from a variety of different models along with evaluation metrics within some seconds. Thereafter the best model could be tested against hyperparameters. Easy to implement and use as it performs all the preprocessing.