ML Part 1

March 1, 2020

1 Nathan Timmerman and Micah Thompkins

2 Task 1. Regression

2.0.1 a.

[19]:
```
from sklearn import datasets
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import KFold

cal = datasets.fetch_california_housing()
X = cal['data']
y = cal['target']

kf = KFold(n_splits=5)

r2_scores_reg = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    reg = LinearRegression().fit(X_train, y_train)
    r2_scores_reg.append(reg.score(X_test, y_test))

print(f'Linear Regression r2 Score: {np.mean(r2_scores_reg)}')

r2_scores_GBR = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    GBR = GradientBoostingRegressor().fit(X_train, y_train)
    r2_scores_GBR.append(GBR.score(X_test, y_test))

print(f'GBTree Regression r2 Score: {np.mean(r2_scores_GBR)}')
```

```
Linear Regression r2 Score: 0.5530311140279232
GBTree Regression r2 Score: 0.6698645135097733
```

2.0.2 b.

[20]:
```
from sklearn import datasets
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import KFold

cal = datasets.fetch_california_housing()
X = cal['data']
y = cal['target']

kf = KFold(n_splits=5)

learning_rate = [0.01, 0.1, 0.5]
n_estimators = [50, 100, 200]
max_depth = [1, 3, 5]

for l in learning_rate:
    for n in n_estimators:
        for m in max_depth:
            r2_scores = []
            GBR = GradientBoostingRegressor(learning_rate=l, n_estimators=n,␣
 ↪max_depth=m)
            for train_index, test_index in kf.split(X):
                X_train, X_test = X[train_index], X[test_index]
                y_train, y_test = y[train_index], y[test_index]
                GBR.fit(X_train, y_train)
                r2_scores.append(GBR.score(X_test, y_test))
            print(f'LR: {l} \t n_est: {n} \t depth: {m} \t r2: {np.
 ↪mean(r2_scores)}')
```

```
LR: 0.01         n_est: 50       depth: 1        r2: 0.13716778094240847
LR: 0.01         n_est: 50       depth: 3        r2: 0.2803357862690098
LR: 0.01         n_est: 50       depth: 5        r2: 0.32040091949643623
LR: 0.01         n_est: 100      depth: 1        r2: 0.2556720337734954
LR: 0.01         n_est: 100      depth: 3        r2: 0.4350438301683216
LR: 0.01         n_est: 100      depth: 5        r2: 0.486592980599929
LR: 0.01         n_est: 200      depth: 1        r2: 0.3635632700884985
LR: 0.01         n_est: 200      depth: 3        r2: 0.5445304370452579
LR: 0.01         n_est: 200      depth: 5        r2: 0.5982635276657959
LR: 0.1          n_est: 50       depth: 1        r2: 0.46718980922217296
LR: 0.1          n_est: 50       depth: 3        r2: 0.6330041985338623
LR: 0.1          n_est: 50       depth: 5        r2: 0.6661536552866457
LR: 0.1          n_est: 100      depth: 1        r2: 0.5365069651471427
LR: 0.1          n_est: 100      depth: 3        r2: 0.6698649765200339
LR: 0.1          n_est: 100      depth: 5        r2: 0.6455269342496381
LR: 0.1          n_est: 200      depth: 1        r2: 0.5867343590454117
LR: 0.1          n_est: 200      depth: 3        r2: 0.6785686943002426
```

```
LR: 0.1         n_est: 200      depth: 5        r2: 0.6426375176076486
LR: 0.5         n_est: 50       depth: 1        r2: 0.5851024032093719
LR: 0.5         n_est: 50       depth: 3        r2: 0.6358382512504972
LR: 0.5         n_est: 50       depth: 5        r2: 0.6328315118905145
LR: 0.5         n_est: 100      depth: 1        r2: 0.6061969370253413
LR: 0.5         n_est: 100      depth: 3        r2: 0.6458062421238864
LR: 0.5         n_est: 100      depth: 5        r2: 0.6139414990504313
LR: 0.5         n_est: 200      depth: 1        r2: 0.6164129421863059
LR: 0.5         n_est: 200      depth: 3        r2: 0.6326880480652705
LR: 0.5         n_est: 200      depth: 5        r2: 0.5902868528095094
```

2.0.3 c.

Performance and Conclusions Based on the r2 scores from part (a), Gradient Boosting Tree Regression performs significantly better than Linear Regression (0.66 versus 0.55), and thus it more accurately fits the data.

Based on part (b), the Gradient Boosting Tree Regression is more accurate with a medium learning-rate (0.1) than small (0.01) or large (0.5) ones. This is likely because a small learning-rate can often lead to overfitting, whereas a large learning-rate often misses the best-fitting regression line. There is also a tradeoff between the number of n_estimators and depth: if the number of n_est is low, the regressor performs better with a larger depth (5), while if the number of n_est is high, the algorithm performs better with a lower depth (3).

3 Task 2. Classification

3.0.1 a.

[16]:
```python
from sklearn import datasets
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

cal = datasets.fetch_california_housing()
X = cal['data']
y = cal['target']

kf = KFold(n_splits=5)

accuracy_log = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    log = LogisticRegression(solver='liblinear').fit(X_train, np.where(y_train␣
 ↪> 2, 1, 0))
    y_pred = log.predict(X_test)
```

```
    accuracy_log.append(accuracy_score(y_pred, np.where(y_test > 2, 1, 0)))

print(f'Logistic Regression Classification Accuracy Score: {np.
 ↪mean(accuracy_log)}')

accuracy_GBC = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    GBC = GradientBoostingClassifier().fit(X_train, np.where(y_train > 2, 1, 0))
    y_pred = GBC.predict(X_test)
    accuracy_GBC.append(accuracy_score(y_pred, np.where(y_test > 2, 1, 0)))

print(f'GBTree Classification Accuracy Score: {np.mean(accuracy_GBC)}')
```

```
Logistic Regression Classification Accuracy Score: 0.7922965116279069
GBTree Classification Accuracy Score: 0.8382267441860465
```

3.0.2 b.

[18]:

```
from sklearn import datasets
import numpy as np
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

cal = datasets.fetch_california_housing()
X = cal['data']
y = cal['target']

kf = KFold(n_splits=5)

learning_rate = [0.01, 0.1, 0.5]
n_estimators = [50, 100, 200]
max_depth = [1, 3, 5]

for l in learning_rate:
    for n in n_estimators:
        for m in max_depth:
            accuracy_scores = []
            GBC = GradientBoostingClassifier(learning_rate=l, n_estimators=n,␣
 ↪max_depth=m)
            for train_index, test_index in kf.split(X):
                X_train, X_test = X[train_index], X[test_index]
                y_train, y_test = y[train_index], y[test_index]
                GBC.fit(X_train, np.where(y_train > 2, 1, 0))
                y_pred = GBC.predict(X_test)
```

```
                accuracy_scores.append(accuracy_score(y_pred, np.where(y_test >␣
↪2, 1, 0)))
            print(f'LR: {l} \t n_est: {n} \t depth: {m} \t accuracy: {np.
↪mean(accuracy_scores)}')
```

```
LR: 0.01        n_est: 50       depth: 1        accuracy: 0.749563953488372
LR: 0.01        n_est: 50       depth: 3        accuracy: 0.7650678294573644
LR: 0.01        n_est: 50       depth: 5        accuracy: 0.7953488372093023
LR: 0.01        n_est: 100      depth: 1        accuracy: 0.7474806201550388
LR: 0.01        n_est: 100      depth: 3        accuracy: 0.7871124031007752
LR: 0.01        n_est: 100      depth: 5        accuracy: 0.8078488372093023
LR: 0.01        n_est: 200      depth: 1        accuracy: 0.7624031007751938
LR: 0.01        n_est: 200      depth: 3        accuracy: 0.8065406976744187
LR: 0.01        n_est: 200      depth: 5        accuracy: 0.8163275193798448
LR: 0.1         n_est: 50       depth: 1        accuracy: 0.7859011627906975
LR: 0.1         n_est: 50       depth: 3        accuracy: 0.8268895348837211
LR: 0.1         n_est: 50       depth: 5        accuracy: 0.8366279069767442
LR: 0.1         n_est: 100      depth: 1        accuracy: 0.804360465116279
LR: 0.1         n_est: 100      depth: 3        accuracy: 0.838953488372093
LR: 0.1         n_est: 100      depth: 5        accuracy: 0.8421996124031008
LR: 0.1         n_est: 200      depth: 1        accuracy: 0.8243701550387597
LR: 0.1         n_est: 200      depth: 3        accuracy: 0.8425872093023257
LR: 0.1         n_est: 200      depth: 5        accuracy: 0.844331395348837
LR: 0.5         n_est: 50       depth: 1        accuracy: 0.8232558139534885
LR: 0.5         n_est: 50       depth: 3        accuracy: 0.8435077519379846
LR: 0.5         n_est: 50       depth: 5        accuracy: 0.8377422480620155
LR: 0.5         n_est: 100      depth: 1        accuracy: 0.8343023255813954
LR: 0.5         n_est: 100      depth: 3        accuracy: 0.8387112403100774
LR: 0.5         n_est: 100      depth: 5        accuracy: 0.8347383720930234
LR: 0.5         n_est: 200      depth: 1        accuracy: 0.8377906976744185
LR: 0.5         n_est: 200      depth: 3        accuracy: 0.8387112403100776
LR: 0.5         n_est: 200      depth: 5        accuracy: 0.8319767441860465
```

3.0.3 c.

[30]:
```
from sklearn import datasets
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score

cal = datasets.fetch_california_housing()
X = cal['data']
y = cal['target']

kf = KFold(n_splits=5)
```

```
roc_auc_score_log = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    log = LogisticRegression(solver='liblinear').fit(X_train, np.where(y_train␣
↪> 2, 1, 0))
    y_pred = log.predict(X_test)
    roc_auc_score_log.append(roc_auc_score(y_pred, np.where(y_test > 2, 1, 0)))

print(f'Logistic Regression Classification ROC AUC Score: {np.
↪mean(roc_auc_score_log)}')

roc_auc_score_GBC = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    GBC = GradientBoostingClassifier().fit(X_train, np.where(y_train > 2, 1, 0))
    y_pred = GBC.predict(X_test)
    roc_auc_score_GBC.append(roc_auc_score(y_pred, np.where(y_test > 2, 1, 0)))

print(f'GBTree Classification ROC AUC Score: {np.mean(roc_auc_score_GBC)}')

print('\n')

cal = datasets.fetch_california_housing()
X = cal['data']
y = cal['target']

kf = KFold(n_splits=5)

learning_rate = [0.01, 0.1, 0.5]
n_estimators = [50, 100, 200]
max_depth = [1, 3, 5]

for l in learning_rate:
    for n in n_estimators:
        for m in max_depth:
            roc_auc_scores = []
            GBC = GradientBoostingClassifier(learning_rate=l, n_estimators=n,␣
↪max_depth=m)
            for train_index, test_index in kf.split(X):
                X_train, X_test = X[train_index], X[test_index]
                y_train, y_test = y[train_index], y[test_index]
                GBC.fit(X_train, np.where(y_train > 2, 1, 0))
                y_pred = GBC.predict(X_test)
```

```
                roc_auc_scores.append(roc_auc_score(y_pred, np.where(y_test >␣
↪2, 1, 0)))
            print(f'LR: {l} \t n_est: {n} \t depth: {m} \t ROC AUC Score: {np.
↪mean(roc_auc_scores)}')
```

```
Logistic Regression Classification ROC AUC Score: 0.7862541394388711
GBTree Classification ROC AUC Score: 0.8396247238334131
```

```
LR: 0.01        n_est: 50       depth: 1        ROC AUC Score:
0.7674944226613825
LR: 0.01        n_est: 50       depth: 3        ROC AUC Score:
0.7943428033861013
LR: 0.01        n_est: 50       depth: 5        ROC AUC Score:
0.8071434308801783
LR: 0.01        n_est: 100      depth: 1        ROC AUC Score:
0.7626207905690827
LR: 0.01        n_est: 100      depth: 3        ROC AUC Score:
0.7925264844254658
LR: 0.01        n_est: 100      depth: 5        ROC AUC Score:
0.811116405455676
LR: 0.01        n_est: 200      depth: 1        ROC AUC Score:
0.770935513988583
LR: 0.01        n_est: 200      depth: 3        ROC AUC Score:
0.8057625831192674
LR: 0.01        n_est: 200      depth: 5        ROC AUC Score:
0.8196742623761853
LR: 0.1         n_est: 50       depth: 1        ROC AUC Score:
0.7908372815235094
LR: 0.1         n_est: 50       depth: 3        ROC AUC Score:
0.8299240808897315
LR: 0.1         n_est: 50       depth: 5        ROC AUC Score:
0.8374154761207245
LR: 0.1         n_est: 100      depth: 1        ROC AUC Score:
0.8099027263005365
LR: 0.1         n_est: 100      depth: 3        ROC AUC Score:
0.8395841894912822
LR: 0.1         n_est: 100      depth: 5        ROC AUC Score:
0.8417581011852884
LR: 0.1         n_est: 200      depth: 1        ROC AUC Score:
0.8275194898599546
LR: 0.1         n_est: 200      depth: 3        ROC AUC Score:
0.843362067375994
LR: 0.1         n_est: 200      depth: 5        ROC AUC Score:
0.841990771053586
LR: 0.5         n_est: 50       depth: 1        ROC AUC Score:
0.8247026705242597
```

```
LR: 0.5         n_est: 50      depth: 3       ROC AUC Score:
0.8377747557449666
LR: 0.5         n_est: 50      depth: 5       ROC AUC Score:
0.8343662363183568
LR: 0.5         n_est: 100     depth: 1       ROC AUC Score:
0.8369906641539886
LR: 0.5         n_est: 100     depth: 3       ROC AUC Score:
0.8363449179307377
LR: 0.5         n_est: 100     depth: 5       ROC AUC Score:
0.8308563275961479
LR: 0.5         n_est: 200     depth: 1       ROC AUC Score:
0.8392080023100785
LR: 0.5         n_est: 200     depth: 3       ROC AUC Score:
0.8344215515480908
LR: 0.5         n_est: 200     depth: 5       ROC AUC Score:
0.8332924224580015
```

3.0.4 d.

[1]:
```python
from sklearn import datasets
import numpy as np
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score, accuracy_score

cal = datasets.fetch_california_housing()
X = cal['data']
y = cal['target']

kf = KFold(n_splits=5)

roc_auc_score_nb = []
accuracy_score_nb = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    nb = DummyClassifier().fit(X_train, np.where(y_train > 2, 1, 0))
    y_pred = nb.predict(X_test)
    roc_auc_score_nb.append(roc_auc_score(y_pred, np.where(y_test > 2, 1, 0)))
    accuracy_score_nb.append(accuracy_score(y_pred, np.where(y_test > 2, 1, 0)))

print(f'Dummy Classification ROC AUC Score: {np.mean(roc_auc_score_nb)}')
print(f'Dummy Classification Accuracy Score: {np.mean(accuracy_score_nb)}')
```

```
Dummy Classification ROC AUC Score: 0.5005488077071869
Dummy Classification Accuracy Score: 0.508672480620155
```

Performance and Conclusions Based on part (a) and (c), the Gradient Boosting Tree Classification algorithm performs better than Logistic Regression Classification by both metrics, accuracy score and ROC score. Both classifiers perform significantly better than a trivial dummy classifier, also by both metrics. Based on part (c), we see similar relationships between learning-rate, n estimators, depth and ROC score as we did with the regressors. However, the difference between learning-rates is not as large as we saw with regression, and the larger learning-rate does not suffer as much with the accuracy and ROC AUC metric, though a learning-rate of 0.1 remains highest among the three tested.