Linear Model 1 - Nearest-neighbor

In [23]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from auxiliars import *
import pickle
```

In [2]:
```
np.random.seed(1234)
```

Data

Standarized data loading:

In [3]:
```
data = pd.read_csv("./data/stdHTRU_2.csv")
```

We split a separate test set of relative size 20%:

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(data[data.colum
ns[0:8]],
                                                    data['class'],
                                                    test_size = 0.2
,
                                                    random_state =
1234)
```

I order to improve the performance of k-NN, we will analyze the performance of the method with no-correlated standarized data:

In [5]:
```
noCorrData = pd.read_csv("./data/noCorrStdHTRU_2.csv")
```

In [6]:
```
X_train_NC, X_test_NC, y_train_NC, y_test_NC = train_test_split(noC
orrData[noCorrData.columns[0:6]],
                                                    noCorrData['cla
ss'],
                                                    test_size = 0.2
,
                                                    random_state =
1234)
```

Model Training

Scikit-learn library offers two options of Supervised Nearest Neighbors:

- KNeighborsClassifier: Algorithm based on the k number of classes.
- RadiusNeighborsClassifier: Algorithm based on the number of neighbors within a fixed radius of each training point.

We will use the first one because we know the number of classes and it is more useful.

In [7]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [8]:
```
kNC = KNeighborsClassifier(n_jobs = -1)
```

KNeighborsClassifier allow us to hypertuning the following parameters:

- Weights:
  - Uniform: All points in each neighborhood are weighted equally.
  - Distance: Weight points by the inverse of their distance.
- Algorithm to compute the nearest neighbors:
  - BallTree
  - KDTree
  - Brute-force Search
- Power parameter for the Minkowski metric:
  - Manhattan Distance (p = 1)
  - Euclidean Distance (p = 2)

In order to hypertuning model parameters and get a better idea on how the model performs on unseen data, we will use GridSearchCV.

In [9]:
```
from sklearn.model_selection import GridSearchCV
```

Values of the 10-Fold CV Grid to test:

In [10]:
```
grid = {'n_neighbors': np.arange(2, 51),
        'weights': ['uniform', 'distance'],
        'algorithm': ['ball_tree', 'kd_tree', 'brute'],
        'p': [1,2]}
```

In [11]:
```
grid
```

Out[11]:
```
{'n_neighbors': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
13, 14, 15, 16, 17, 18,
        19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33
, 34, 35,
        36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50
]),
 'weights': ['uniform', 'distance'],
 'algorithm': ['ball_tree', 'kd_tree', 'brute'],
 'p': [1, 2]}
```

Grid Search 10-Fold CV:

In [12]:
```
gs10cv = GridSearchCV(kNC, param_grid = grid, cv = 10, n_jobs = -1)
```

Normal Data Training

In [13]:
```
gs10cv.fit(X_train, y_train)
```

Out[13]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf
_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_
jobs=-1,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=-1,
             param_grid={'algorithm': ['ball_tree', 'kd_tree', 'br
ute'],
                         'n_neighbors': array([ 2,  3,  4,  5,  6,
7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]
),
                         'p': [1, 2], 'weights': ['uniform', 'dist
ance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=False,
             scoring=None, verbose=0)
```

In [14]:
```
gs10cv.best_params_
```

Out[14]:
```
{'algorithm': 'ball_tree', 'n_neighbors': 18, 'p': 2, 'weights': '
distance'}
```

In [15]:
```
pd.DataFrame(gs10cv.cv_results_).iloc[gs10cv.best_index_]
```

Out[15]:
```
mean_fit_time                                                    0.092
2411
std_fit_time                                                     0.083
5155
mean_score_time                                                   0.36
1971
std_score_time                                                   0.080
9577
param_algorithm                                                  ball_
tree
param_n_neighbors
18
param_p
2
param_weights                                                     dist
ance
params               {'algorithm': 'ball_tree', 'n_neighbors': 18,
...
split0_test_score                                                 0.9
7905
split1_test_score                                                 0.9
7905
split2_test_score                                                0.97
7654
split3_test_score                                                0.97
8352
split4_test_score                                                0.97
3464
split5_test_score                                                 0.9
8324
split6_test_score                                                0.98
0447
split7_test_score                                                0.97
8352
split8_test_score                                                0.98
1132
split9_test_score                                                0.97
7638
mean_test_score                                                   0.97
8838
std_test_score                                                  0.0024
3966
rank_test_score
1
Name: 67, dtype: object
```

In [16]:
```
# Save model
kNCFile = open('./models/kNC_BestCV_STDData_pickle_file', 'wb')
pickle.dump(gs10cv, kNCFile)
```

No-correlated Data Training

Grid Search 10-Fold CV:

In [17]:
```
gs10cv_nc = GridSearchCV(kNC, param_grid = grid, cv = 10, n_jobs =
-1)
```

Training:

In [18]:
```
gs10cv_nc.fit(X_train_NC, y_train_NC)
```

Out[18]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf
_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_
jobs=-1,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=-1,
             param_grid={'algorithm': ['ball_tree', 'kd_tree', 'br
ute'],
                         'n_neighbors': array([ 2,  3,  4,  5,  6,
7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]
),
                         'p': [1, 2], 'weights': ['uniform', 'dist
ance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=False,
             scoring=None, verbose=0)
```

In [19]:
```
pd.DataFrame(gs10cv_nc.cv_results_).iloc[gs10cv_nc.best_index_]
```

Out[19]:
```
mean_fit_time                                                    0.081
2808
std_fit_time                                                     0.050
1152
mean_score_time                                                   0.20
7251
std_score_time                                                    0.04
6478
param_algorithm                                                 ball_
tree
param_n_neighbors
17
param_p
2
param_weights                                                    dist
ance
params               {'algorithm': 'ball_tree', 'n_neighbors': 17,
...
split0_test_score                                                 0.9
7905
split1_test_score                                                0.97
9749
split2_test_score                                                0.97
6955
split3_test_score                                                0.97
8352
split4_test_score                                                 0.9
7486
split5_test_score                                                0.98
1844
split6_test_score                                                0.97
9749
split7_test_score                                                 0.9
7905
split8_test_score                                                0.98
0433
split9_test_score                                                0.97
8337
mean_test_score                                                  0.97
8838
std_test_score                                                 0.0018
2156
rank_test_score
1
Name: 63, dtype: object
```

In [20]:
```
# Save model
kNCFileNC = open('./models/kNC_BestCV_NCorrSTDData_pickle_file', 'w
b')
pickle.dump(gs10cv_nc, kNCFile)
```

Testing

Normal Data Model Testing

In [21]:
```
y_pred = gs10cv.predict(X_test)
```

In [24]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3249
           1       0.93      0.82      0.87       331

    accuracy                           0.98      3580
   macro avg       0.96      0.91      0.93      3580
weighted avg       0.98      0.98      0.98      3580
```

In [25]:
```
print("Confusion Matrix:")
confusionMatrix(y_test, y_pred, classes = [0,1])
```

```
Confusion Matrix:
```

Out[25]:

Predicted	0	1
Real		
0	3230	19
1	60	271

In [26]:
```
print("Test Error:")
(1-accuracy_score(y_test, gs10cv.predict(X_test)))*100
```

```
Test Error:
```

Out[26]:
```
2.206703910614527
```

No-correlated Data Model Testing

In [27]:
```
y_pred_NC = gs10cv_nc.predict(X_test_NC)
```

In [28]:
```
print(classification_report(y_test_NC, y_pred_NC))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3249
           1       0.94      0.81      0.87       331

    accuracy                           0.98      3580
   macro avg       0.96      0.90      0.93      3580
weighted avg       0.98      0.98      0.98      3580
```

In [29]:
```
print("Confusion Matrix:")
confusionMatrix(y_test_NC, y_pred_NC, classes = [0,1])
```

```
Confusion Matrix:
```

Out[29]:

Predicted	0	1
Real		
0	3231	18
1	62	269

In [30]:
```
print("Test Error:")
(1-accuracy_score(y_test_NC, gs10cv_nc.predict(X_test_NC)))*100
```

```
Test Error:
```

Out[30]:
```
2.2346368715083775
```