Feature_Engineering

May 2, 2023

1 Feature Engineering

1.1 Import necessary library

[28]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import OrdinalEncoder
```

1.2 Get data from csv files

[43]:
```python
train_dataset_df = pd.read_csv('Mushroom_datasets/mushroom_train.csv')
test_dataset_df = pd.read_csv('Mushroom_datasets/mushroom_test.csv')
print(train_dataset_df)
```

```
       cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0              4.98         c           i         y                    f
1              2.84         x           y         y                    f
2             11.44         x           y         y                    f
3              8.77         s           t         r                    t
4              7.55         x           d         n                    t
…               …        …         …         …                    …
42743          3.28         f           y         p                    f
42744          8.91         x           w         p                    f
42745         45.84         o           y         y                    f
42746         10.91         f           y         n                    f
42747          2.41         f           t         w                    f

      gill-attachment gill-spacing gill-color  stem-height  stem-width  \
0                   a            c          n         6.04        6.21
1                   a            c          w         5.66        3.55
2                   a            c          w         7.03       25.29
3                   d            c          g         4.44       13.61
4                   p            c          y         8.41       18.44
…               …            …          …          …           …
42743               e            c          n         4.96        3.51
42744               s            c          p         4.61       11.12
```

```
42745                p            c          y          5.75        26.36
42746                a            c          w          7.55        24.38
42747                a            c          n          3.52         3.71

      stem-color has-ring ring-type habitat season class
0             w        f         f       d      a     p
1             y        t         r       h      u     p
2             n        t         e       d      w     e
3             r        f         f       d      a     p
4             y        f         f       d      a     e
…          …      …       …      …     …    …
42743         w        t         e       m      u     p
42744         p        f         f       d      a     p
42745         n        f         f       d      s     e
42746         n        t         e       d      w     e
42747         u        f         f       d      u     p

[42748 rows x 16 columns]
```

[46]:
```python
train_dataset_df.describe()
```

[46]:

	cap-diameter	stem-height	stem-width
count	42748.000000	42748.000000	42748.000000
mean	6.714149	6.583224	12.117692
std	5.220008	3.368333	10.004874
min	0.380000	0.000000	0.000000
25%	3.480000	4.640000	5.180000
50%	5.865000	5.960000	10.200000
75%	8.530000	7.750000	16.540000
max	62.340000	33.920000	103.910000

[47]:
```python
train_dataset_df['class'].value_counts()
```

[47]:
```
p    23595
e    19153
Name: class, dtype: int64
```

[48]:
```python
test_dataset_df['class'].value_counts()
```

[48]:
```
p    10293
e     8028
Name: class, dtype: int64
```

1.3 Feature Engineering

Most of the features of the dataset are categorical (such as cap-shape: bell, conical, convex or flat) and cannot be directly used as inputs to machine learning models (since they are not numerical). We can use such features to create extra features on both training and test datasets. The new features can reflect statistics of the original numerical features and can potentially detect patterns of poisonous or edible mushrooms and simplify the classification task.

We can use the categorical features to group all the data points with the same categorical feature value (i.e., all the mushrooms with orange cap color) and calculate statistics of the numerical data corresponding to each group (i.e., average cap-diameter of all the mushrooms with orange cap color). Then in the new feature, all data points of this group (i.e., mushrooms with orange cap-color) are assigned that calculated statistic. This could be used as an alternative to one-hot encoding of the feature.

[37]:
```
def featureEngineering():

    # Encode the "T/F" traning data
    # Copy in case o f overwrite
    encode_data_train = train_dataset_df.copy()
    enc = OrdinalEncoder() # 4, 11 from original data "f" -> 0.0, "t" -> 1.0
    encode_data_train[["does-bruise-or-bleed","has-ring"]] = enc.
 ↪fit_transform(encode_data_train[["does-bruise-or-bleed","has-ring"]])

    # Apply the same OrdinalEncoder
    encode_data_test = test_dataset_df.copy()
    encode_data_test[["does-bruise-or-bleed","has-ring"]] = enc.
 ↪fit_transform(encode_data_test[["does-bruise-or-bleed","has-ring"]])

    # Calculated the different value of the numerical data only by grouping the␣
 ↪mushrooms's non-num feature
    group_list = ["cap-shape", "cap-surface", "cap-color", "gill-attachment",␣
 ↪"gill-spacing", "gill-color", "stem-color", "ring-type", "habitat", "season"]


    # Put new features in a dictionary using the original numeric data only
    diction = {}
    for feature in group_list:
        average_feature = train_dataset_df.groupby([feature], as_index=True).
 ↪mean(numeric_only=True)
        min_feature = train_dataset_df.groupby([feature], as_index=True).
 ↪min(numeric_only=True)
        max_feature = train_dataset_df.groupby([feature], as_index=True).
 ↪max(numeric_only=True)
        median_feature = train_dataset_df.groupby([feature], as_index=True).
 ↪median(numeric_only=True)
        diction[feature] = {"average": average_feature, "min": min_feature,␣
 ↪"max": max_feature, "median": median_feature}
#     print(diction)

    # List of new features for training
    new_features = []
```

```
    for feature in diction:
#         print(feature)

        for statistic in diction[feature]:
#             print(statistic)

            for num_feature in diction[feature][statistic]:
#                 print(num_feature)
                feature_name = feature + '-' + num_feature + '-' + statistic
                new_feature = train_dataset_df[feature].
↪map(diction[feature][statistic][num_feature]).rename(feature_name)
                new_features.append(new_feature)

    # List of new features for testing using the statistics from training data
    new_features_test = []
    for feature in diction:
#         print(feature)

        for statistic in diction[feature]:
#             print(statistic)

            for num_feature in diction[feature][statistic]:
#                 print(num_feature)
                feature_name = feature + '-' + num_feature + '-' + statistic
                new_feature = test_dataset_df[feature].
↪map(diction[feature][statistic][num_feature]).rename(feature_name)
                new_features_test.append(new_feature)

    # Copy the original numerical features
    xdata_train = encode_data_train.iloc[:, [0,4,8,9,11]].copy()
    # Concat with the new features
    xdata_train = pd.concat([xdata_train] + new_features, axis = 1)
    print("New training dataset is :")
    print(xdata_train)
    print(f"The shape of training data is {xdata_train.shape}")

    # Deal with the test data
    xdata_test = encode_data_test.iloc[:, [0,4,8,9,11]].copy()
    xdata_test = pd.concat([xdata_test] + new_features_test, axis = 1)
    print("New testing dataset is :")
    print(xdata_test)
    print(f"The shape of testing data is {xdata_test.shape}")

    # Convert the training label to number
    ydata_train = train_dataset_df.iloc[:,-1:].values
    ydata_test = test_dataset_df.iloc[:,-1:].values
    ydata_train = ydata_train.reshape(-1)
```

```
    ydata_test = ydata_test.reshape(-1)
    labelencoder = preprocessing.LabelEncoder()
    labelencoder.fit(ydata_train)

    # 0 stands for edible, 1 stands for poison
    ydata_train = labelencoder.transform(ydata_train)
    ydata_test = labelencoder.transform(ydata_test)
#     print(ydata_train)
#     print(ydata_test)

    return xdata_train, ydata_train, xdata_test, ydata_test
```

[38]:
```
xdata_train, ydata_train, xdata_test, ydata_test = featureEngineering()
```

```
New training dataset is :
       cap-diameter  does-bruise-or-bleed  stem-height  stem-width  has-ring  \
0              4.98                   0.0         6.04        6.21       0.0
1              2.84                   0.0         5.66        3.55       1.0
2             11.44                   0.0         7.03       25.29       1.0
3              8.77                   1.0         4.44       13.61       0.0
4              7.55                   1.0         8.41       18.44       0.0
…               …                   …           …           …         …
42743          3.28                   0.0         4.96        3.51       1.0
42744          8.91                   0.0         4.61       11.12       0.0
42745         45.84                   0.0         5.75       26.36       0.0
42746         10.91                   0.0         7.55       24.38       1.0
42747          2.41                   0.0         3.52        3.71       0.0

       cap-shape-cap-diameter-average  cap-shape-stem-height-average  \
0                            3.781002                       6.411460
1                            6.671073                       6.849409
2                            6.671073                       6.849409
3                            7.477626                       5.541955
4                            6.671073                       6.849409
…                                 …                             …
42743                        6.939337                       6.603375
42744                        6.671073                       6.849409
42745                        9.562623                       2.983186
42746                        6.939337                       6.603375
42747                        6.939337                       6.603375

       cap-shape-stem-width-average  cap-shape-cap-diameter-min  \
0                          7.404107                        0.55
1                         12.412532                        0.38
2                         12.412532                        0.38
3                         14.113719                        1.03
4                         12.412532                        0.38
```

```
…                               …                        …
42743                   11.648403                     0.47
42744                   12.412532                     0.38
42745                   16.291891                     1.08
42746                   11.648403                     0.47
42747                   11.648403                     0.47

       cap-shape-stem-height-min  …  season-stem-width-average  \
0                           2.24  …                  12.074701
1                           1.20  …                  11.847626
2                           1.20  …                  13.734083
3                           1.92  …                  12.074701
4                           1.20  …                  12.074701
…                            …  …                        …
42743                       1.94  …                  11.847626
42744                       1.20  …                  12.074701
42745                       0.00  …                  11.767445
42746                       1.94  …                  13.734083
42747                       1.94  …                  11.847626

       season-cap-diameter-min  season-stem-height-min  season-stem-width-min  \
0                         0.38                     0.0                    0.0
1                         0.44                     0.0                    0.0
2                         0.53                     0.0                    0.0
3                         0.38                     0.0                    0.0
4                         0.38                     0.0                    0.0
…                          …                      …                     …
42743                     0.44                     0.0                    0.0
42744                     0.38                     0.0                    0.0
42745                     0.54                     0.0                    0.0
42746                     0.53                     0.0                    0.0
42747                     0.44                     0.0                    0.0

       season-cap-diameter-max  season-stem-height-max  season-stem-width-max  \
0                        30.48                   33.92                 101.69
1                        59.46                   33.03                 103.91
2                        30.34                   27.36                  70.02
3                        30.48                   33.92                 101.69
4                        30.48                   33.92                 101.69
…                          …                      …                     …
42743                    59.46                   33.03                 103.91
42744                    30.48                   33.92                 101.69
42745                    62.34                   25.78                  64.11
42746                    30.34                   27.36                  70.02
42747                    59.46                   33.03                 103.91

       season-cap-diameter-median  season-stem-height-median  \
0                            6.06                       6.01
```

```
1                          5.62                       5.91
2                          6.51                       6.00
3                          6.06                       6.01
4                          6.06                       6.01
…                         …                         …
42743                      5.62                       5.91
42744                      6.06                       6.01
42745                      4.69                       5.74
42746                      6.51                       6.00
42747                      5.62                       5.91

       season-stem-width-median
0                         10.64
1                          9.30
2                         10.81
3                         10.64
4                         10.64
…                         …
42743                      9.30
42744                     10.64
42745                      8.19
42746                     10.81
42747                      9.30

[42748 rows x 125 columns]
The shape of training data is (42748, 125)
New testing dataset is :
       cap-diameter  does-bruise-or-bleed  stem-height  stem-width  has-ring  \
0             13.95                   0.0         7.67       22.22       0.0
1             17.79                   0.0         6.39       36.42       0.0
2              1.50                   0.0         5.30        1.44       0.0
3             15.33                   1.0         5.16       26.60       0.0
4             15.96                   0.0        23.57       19.51       1.0
…            …                    …          …          …         …
18316          7.26                   0.0         7.96        7.95       1.0
18317         21.33                   0.0        17.48       38.40       0.0
18318          2.65                   0.0         8.49        3.78       0.0
18319         14.75                   0.0         8.40       22.11       0.0
18320          3.62                   0.0         5.77        4.49       0.0

       cap-shape-cap-diameter-average  cap-shape-stem-height-average  \
0                            7.477626                       5.541955
1                            7.477626                       5.541955
2                            6.671073                       6.849409
3                            6.671073                       6.849409
4                            9.038520                      11.166606
…                               …                              …
18316                        6.671073                       6.849409
```

```
18317                       6.671073                        6.849409
18318                       7.477626                        5.541955
18319                       6.939337                        6.603375
18320                       3.620942                        6.745842

       cap-shape-stem-width-average  cap-shape-cap-diameter-min  \
0                         14.113719                        1.03
1                         14.113719                        1.03
2                         12.412532                        0.38
3                         12.412532                        0.38
4                         18.907253                        0.53
…                               …                         …
18316                     12.412532                        0.38
18317                     12.412532                        0.38
18318                     14.113719                        1.03
18319                     11.648403                        0.47
18320                      5.259899                        0.47

       cap-shape-stem-height-min  …  season-stem-width-average  \
0                           1.92  …                  12.074701
1                           1.92  …                  11.847626
2                           1.20  …                  13.734083
3                           1.20  …                  12.074701
4                           2.29  …                  12.074701
…                            …  …                        …
18316                       1.20  …                  12.074701
18317                       1.20  …                  12.074701
18318                       1.92  …                  11.847626
18319                       1.94  …                  12.074701
18320                       1.80  …                  12.074701

       season-cap-diameter-min  season-stem-height-min  season-stem-width-min  \
0                         0.38                     0.0                    0.0
1                         0.44                     0.0                    0.0
2                         0.53                     0.0                    0.0
3                         0.38                     0.0                    0.0
4                         0.38                     0.0                    0.0
…                          …                      …                     …
18316                     0.38                     0.0                    0.0
18317                     0.38                     0.0                    0.0
18318                     0.44                     0.0                    0.0
18319                     0.38                     0.0                    0.0
18320                     0.38                     0.0                    0.0

       season-cap-diameter-max  season-stem-height-max  season-stem-width-max  \
0                        30.48                   33.92                 101.69
1                        59.46                   33.03                 103.91
2                        30.34                   27.36                  70.02
```

```
3                     30.48                 33.92                 101.69
4                     30.48                 33.92                 101.69
…                     …                     …                     …
18316                 30.48                 33.92                 101.69
18317                 30.48                 33.92                 101.69
18318                 59.46                 33.03                 103.91
18319                 30.48                 33.92                 101.69
18320                 30.48                 33.92                 101.69

       season-cap-diameter-median  season-stem-height-median  \
0                            6.06                       6.01
1                            5.62                       5.91
2                            6.51                       6.00
3                            6.06                       6.01
4                            6.06                       6.01
…                             …                         …
18316                        6.06                       6.01
18317                        6.06                       6.01
18318                        5.62                       5.91
18319                        6.06                       6.01
18320                        6.06                       6.01

       season-stem-width-median
0                         10.64
1                          9.30
2                         10.81
3                         10.64
4                         10.64
…                         …
18316                     10.64
18317                     10.64
18318                      9.30
18319                     10.64
18320                     10.64

[18321 rows x 125 columns]
The shape of testing data is (18321, 125)
```

1.4 Save the new data to csv file

Save the original expanded data and the data after feature selection

[39]:
```
# Concatenate the training xdata and training ydata
data_train = np.zeros((xdata_train.shape[0],xdata_train.shape[1]+1))
data_train[:, :xdata_train.shape[1]] = np.copy(xdata_train)
data_train[:,-1] = np.copy(ydata_train)

# Concatenate the training xdata and training ydata
```

```
data_test = np.zeros((xdata_test.shape[0],xdata_test.shape[1]+1))
data_test[:, :xdata_test.shape[1]] = np.copy(xdata_test)
data_test[:,-1] = np.copy(ydata_test)

print(data_train.shape)
print(data_test.shape)
# Save the data to csv file
df_train = pd.DataFrame(data_train)
df_train.to_csv('mushroom_train_encode.csv', index=False)
df_test = pd.DataFrame(data_test)
df_test.to_csv('mushroom_test_encode.csv', index=False)
```

```
(42748, 126)
(18321, 126)
```

Trivial_Baseline_System

May 2, 2023

1 Trivial system & Baseline system

Define reference systems and analysis

Report required performance measures

- F1-score • Accuracy

1.1 Import necessary library

[16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
import warnings
```

1.2 Get data from previous cvs file

[28]:
```
def getData(fname1, fname2):
    df_train = pd.read_csv(fname1)
    df_test = pd.read_csv(fname2)
    data_train = np.array(df_train)
    data_test = np.array(df_test)
    xdata_train = data_train[:,:len(data_train[0]) - 1]
    ydata_train = data_train[:, -1]
    xdata_test = data_test[:,:len(data_test[0]) - 1]
    ydata_test = data_test[:, -1]

    return xdata_train, ydata_train, xdata_test, ydata_test

xdata_train_expand, ydata_train_expand, xdata_test_expand, ydata_test_expand =␣
 ↪getData("mushroom_train_encode.csv", "mushroom_test_encode.csv")
# print(xdata_train_expand)
```

```
print(f"The shape of training xdata shape is", xdata_train_expand.shape)
print(f"The shape of training ydata shape is", ydata_train_expand.shape)
print(f"The shape of testing xdata shape is", xdata_test_expand.shape)
print(f"The shape of testing ydata shape is", ydata_test_expand.shape)
```

```
The shape of training xdata shape is (42748, 125)
The shape of training ydata shape is (42748,)
The shape of testing xdata shape is (18321, 125)
The shape of testing ydata shape is (18321,)
```

1.3 Trivial system

A system that outputs class assignments (S0, S1) at random with probability N0/N and N1/N, respectively; Ni is the number of training data points with class label Si, and N is the total number of training data points.

[18]:
```
def trival_sys(xdata_train, ydata_train, xdata_test, ydata_test):
    po = np.sum(ydata_train == 1)
    ed = np.sum(ydata_train == 0)
    N = len(ydata_train)

    ps0 = ed/N
    ps1 = po/N

    print(f"P(S0) edible = {ed/N} and P(S1) poisonous = {po/N}")

    N_test = len(ydata_test)
    ydata_test_pred = np.random.choice([0,1], size=N_test, p=[ps0, ps1])


    # Report the F1 score and the accuracy
    score = f1_score(ydata_test_pred, ydata_test)
    print(f"The F1 Score of the trivial system = {score}")

    acc = np.sum(ydata_test_pred == ydata_test) / N_test
    print(f"The accuracy of the trivial system = {acc}")


trival_sys(xdata_train_expand, ydata_train_expand, xdata_test_expand,␣
 ↪ydata_test_expand)
```

```
P(S0) edible = 0.4480443529521849 and P(S1) poisonous = 0.5519556470478151
The F1 Score of the trivial system = 0.5589720955323427
The accuracy of the trivial system = 0.5091425140549096
```

1.4 Baseline system

Nearest means classifier

[50]:

```python
def nearest_mean_classifier_unnormalize(xdata_train, ydata_train):

    # Copy the data in case overwrite
    xdata_train = np.copy(xdata_train)

    # Split data
    xdata_train, xdata_val, ydata_train, ydata_val =␣
↪train_test_split(xdata_train, ydata_train, test_size=0.2, random_state=42)

    # Calculate mean values for two class using unnormalized value
    mean_ed = np.mean(xdata_train[ydata_train == 0], axis = 0)
    mean_po = np.mean(xdata_train[ydata_train == 1], axis = 0)

    N_train = len(ydata_train)
    N_val = len(ydata_val)

    # Train on the training data
    ydata_train_pre = np.zeros(N_train)

    for i in range(N_train):
        dist_ed = np.linalg.norm(xdata_train[i] - mean_ed)
        dist_po = np.linalg.norm(xdata_train[i] - mean_po)

        # Classify the data to the nearest mean point's class
        if(dist_ed < dist_po):
            ydata_train_pre[i] = 0
        else:
            ydata_train_pre[i] = 1

    # Report the F1 score and the accuracy for the training data
    score_train = f1_score(ydata_train_pre, ydata_train)
    print(f"The training F1 Score of nearest mean system on unnormalized data =␣
↪{score_train}")

    acc_train = np.sum(ydata_train_pre == ydata_train) / N_train
    print(f"The training accuracy of nearest mean system on unnormalized data =␣
↪{acc_train}")

    # Test on the validation data
    ydata_val_pre = np.zeros(N_val)
    for i in range(N_val):
        dist_ed = np.linalg.norm(xdata_val[i] - mean_ed)
        dist_po = np.linalg.norm(xdata_val[i] - mean_po)

        # Classify the data to the nearest mean point's class
        if(dist_ed < dist_po):
            ydata_val_pre[i] = 0
```

```
        else:
            ydata_val_pre[i] = 1

    # Report the F1 score and the accuracy for the training data
    score_val = f1_score(ydata_val_pre, ydata_val)
    print(f"The validation F1 Score of nearest mean system on unnormalized data␣
↪= {score_val}")

    acc_val = np.sum(ydata_val_pre == ydata_val) / N_val
    print(f"The validation accuracy of nearest mean system on unnormalized data␣
↪= {acc_val}")

nearest_mean_classifier_unnormalize(xdata_train_expand, ydata_train_expand)
```

```
The training F1 Score of nearest mean system on unnormalized data =
0.6160428397924257
The training accuracy of nearest mean system on unnormalized data =
0.593251067313878
The validation F1 Score of nearest mean system on unnormalized data =
0.6072910824453168
The validation accuracy of nearest mean system on unnormalized data =
0.5905263157894737
```

As we can see, the accuracy indicates that the nearest mean system on unnormalized dataset doesn't perform well. To compare with the baseline system, we experiment the nearest mean on normalized data.

[51]:
```
def nearest_mean_classifier_normalize(xdata_train, ydata_train, printOrNot =␣
↪True):
#     print(xdata_train)
#     print(xdata_train.shape)

    # Copy the data in case overwrite
    xdata_train = np.copy(xdata_train)

    # Shuffle data
    p = np.random.permutation(xdata_train.shape[0])
    xdata_train = xdata_train[p]
    ydata_train = ydata_train[p]

    # Split data
    xdata_train, xdata_val, ydata_train, ydata_val =␣
↪train_test_split(xdata_train, ydata_train, test_size=0.2, random_state=42)

    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)
```

```
    # Apply scaler to data
    xdata_train_scaled = scaler.transform(xdata_train)
    xdata_val_scaled = scaler.transform(xdata_val)

    # Calculate mean values for two class using unnormalized value
    mean_ed = np.mean(xdata_train_scaled[ydata_train == 0], axis = 0)
    mean_po = np.mean(xdata_train_scaled[ydata_train == 0], axis = 0)
#     print(mean_ed.shape)
#     print(mean_ed)

    N_train = len(ydata_train)
    N_val = len(ydata_val)

    # Train on the training data
    ydata_train_pre = np.zeros(N_train)
    for i in range(N_train):
        dist_ed = np.linalg.norm(xdata_train_scaled[i] - mean_ed)
        dist_po = np.linalg.norm(xdata_train_scaled[i] - mean_po)

        # Classify the data to the nearest mean point's class
        if(dist_ed < dist_po):
            ydata_train_pre[i] = 0
        else:
            ydata_train_pre[i] = 1

    score_train = f1_score(ydata_train_pre, ydata_train)
    acc_train = np.sum(ydata_train_pre == ydata_train) / N_train

    if(printOrNot):
        # Report the F1 score and the accuracy for the training data
        print(f"The training F1 Score of nearest mean system on normalized data␣
 ↪= {score_train}")
        print(f"The training accuracy of nearest mean system on normalized data␣
 ↪= {acc_train}")

    # Test on the validation data
    ydata_val_pre = np.zeros(N_val)
    for i in range(N_val):
        dist_ed = np.linalg.norm(xdata_val_scaled[i] - mean_ed)
        dist_po = np.linalg.norm(xdata_val_scaled[i] - mean_po)

        # Classify the data to the nearest mean point's class
        if(dist_ed < dist_po):
            ydata_val_pre[i] = 0
        else:
            ydata_val_pre[i] = 1
```

```
    score_val = f1_score(ydata_val_pre, ydata_val)
    acc_val = np.sum(ydata_val_pre == ydata_val) / N_val
    if(printOrNot):
        # Report the F1 score and the accuracy for the training data
        print(f"The validation F1 Score of nearest mean system on normalized␣
 ↪data = {score_val}")
        print(f"The validation accuracy of nearest mean system on normalized␣
 ↪data = {acc_val}")

    return score_val, acc_val


nearest_mean_classifier_normalize(xdata_train_expand, ydata_train_expand)
```

```
The training F1 Score of nearest mean system on normalized data =
0.711333019312294
The training accuracy of nearest mean system on normalized data =
0.5519913445230715
The validation F1 Score of nearest mean system on normalized data =
0.7111848055471811
The validation accuracy of nearest mean system on normalized data =
0.551812865497076
```

[51]:
```
(0.7111848055471811, 0.551812865497076)
```

1.5 Test

[46]:
```
def test_nearest_mean_classifier_normalize(xdata_train, ydata_train,␣
 ↪xdata_test, ydata_test):
#     print(xdata_train)
#     print(xdata_train.shape)

    # Copy the data in case overwrite
    xdata_train = np.copy(xdata_train)
    xdata_test = np.copy(xdata_test)

    # Shuffle data
    p = np.random.permutation(xdata_train.shape[0])
    xdata_train = xdata_train[p]
    ydata_train = ydata_train[p]

    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train_scaled = scaler.transform(xdata_train)
```

```
    xdata_test_scaled = scaler.transform(xdata_test)

    # Calculate mean values for two class using unnormalized value
    mean_ed = np.mean(xdata_train_scaled[ydata_train == 0], axis = 0)
    mean_po = np.mean(xdata_train_scaled[ydata_train == 1], axis = 0)
#     print(mean_ed.shape)
#     print(mean_ed)

    N_train = len(ydata_train)
    N_test = len(ydata_test)

    # Test on the testing data
    ydata_test_pre = np.zeros(N_test)
    for i in range(N_test):
        dist_ed = np.linalg.norm(xdata_test_scaled[i] - mean_ed)
        dist_po = np.linalg.norm(xdata_test_scaled[i] - mean_po)

        # Classify the data to the nearest mean point's class
        if(dist_ed < dist_po):
            ydata_test_pre[i] = 0
        else:
            ydata_test_pre[i] = 1

    score_test = f1_score(ydata_test_pre, ydata_test)
    acc_test = np.sum(ydata_test_pre == ydata_test) / N_test

    # Report the F1 score and the accuracy for the training data
    print(f"The testing F1 Score of nearest mean system on normalized data = {score_test}")
    print(f"The testing accuracy of nearest mean system on normalized data = {acc_test}")


test_nearest_mean_classifier_normalize(xdata_train_expand, ydata_train_expand,xdata_test_expand, ydata_test_expand)
```

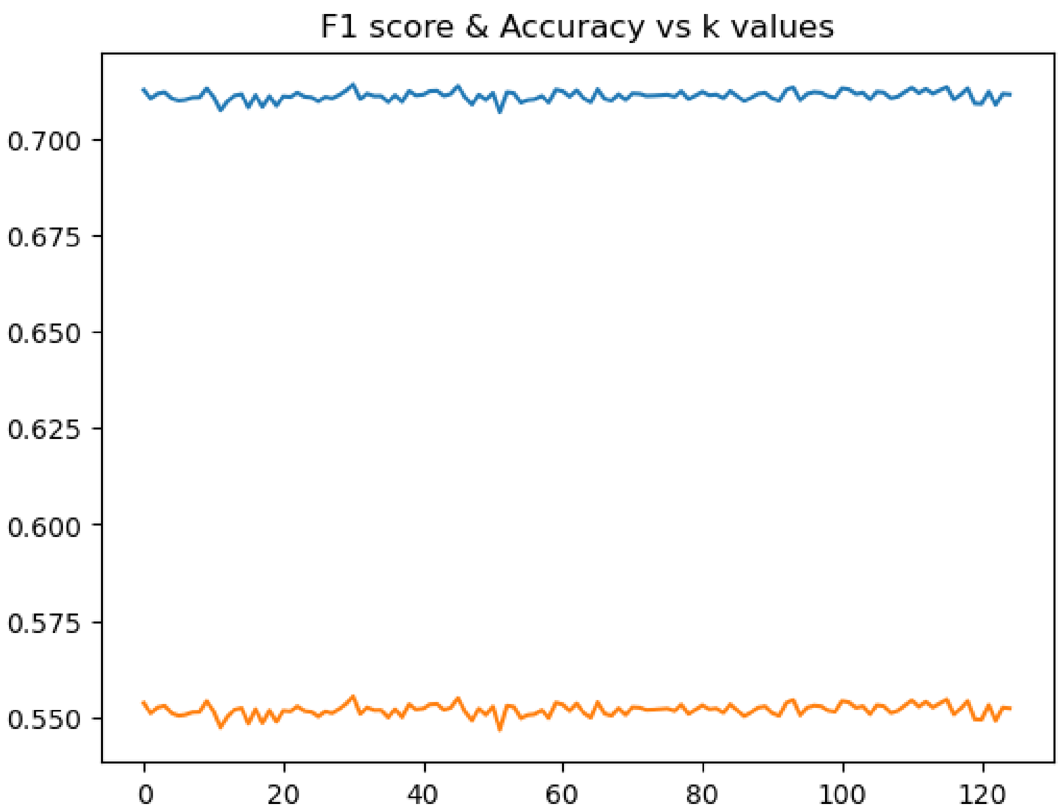
```
The testing F1 Score of nearest mean system on normalized data =
0.6989473684210525
The testing accuracy of nearest mean system on normalized data =
0.6643742153812565
```

Obviously, the F1 score of model on normalized data is much better than the previous one. But there are still too many features that possibly worsen model performance. Therefore I decide to select k features in previous numerical data on the normalized data and take the best k value.

1.6 Feature Selection or Reduction

After the feature engineering step, the dataset will have a large number of features, which can slow down model runtime or possibly worsen model performance. It is important to remove some features (or reduce dimensionality of feature space some other way) and keep $<$ number of features that are more useful for the prediction task. Here we use univariate feature selection to reduce the dimension.

[52]:
```
def feature_selection(xdata_train, ydata_train):
    D = xdata_train.shape[1]
    F1_total = np.zeros(D)
    acc_total = np.zeros(D)

    # ignore all UserWarning messages
    warnings.filterwarnings('ignore', category=UserWarning)
    warnings.filterwarnings('ignore', category=RuntimeWarning)

    epochs_num = 10

    for epoch in range(epochs_num):
        for k in range(D):
            # f_classif: Compute the ANOVA F-value for the provided sample.
            # Define feature selection
            fs = SelectKBest(score_func=f_classif, k=k+1)

            # Apply feature selection
            xdata_train_selected = fs.fit_transform(xdata_train, ydata_train)

            # Get each k value's f1 score and accuracy
            f1_score, acc =␣
 ↪nearest_mean_classifier_normalize(xdata_train_selected, ydata_train, False)

            F1_total[k] += f1_score
            acc_total[k] += acc

    for k in range(D):
        F1_total[k] =  F1_total[k] / epochs_num
        acc_total[k] = acc_total[k] / epochs_num

    plt.title("F1 score & Accuracy vs k values")
    plt.plot(np.arange(D), F1_total, label = "F1 score")
    plt.plot(np.arange(D), acc_total, label = "Accuracy")
    plt.show()
    print(f"The best performance k is {np.argmax(F1_total) + 1}")

feature_selection(xdata_train_expand, ydata_train_expand)
```




```
The best performance k is 31
```

1.7 Save the selected features as a new csv file

```
[24]: def save_selected_data(xdata_train, ydata_train, xdata_test, ydata_test):

          fs = SelectKBest(score_func=f_classif, k=32)

          # Apply feature selection
          xdata_train_selected = fs.fit_transform(xdata_train, ydata_train)
          xdata_test_selected = fs.transform(xdata_test)

          # Concatenate the training xdata and training ydata
          data_train_selected = np.zeros((xdata_train_selected.
       ↪shape[0],xdata_train_selected.shape[1]+1))
          data_train_selected[:, :xdata_train_selected.shape[1]] = np.
       ↪copy(xdata_train_selected)
          data_train_selected[:,-1] = np.copy(ydata_train)

          # Concatenate the training xdata and training ydata
```

```
    data_test_selected = np.zeros((xdata_test_selected.
↪shape[0],xdata_test_selected.shape[1]+1))
    data_test_selected[:, :xdata_test_selected.shape[1]] = np.
↪copy(xdata_test_selected)
    data_test_selected[:,-1] = np.copy(ydata_test)

    # Save the data to csv file
    df_train = pd.DataFrame(data_train_selected)
    df_train.to_csv('mushroom_train_select.csv', index=False)
    df_test = pd.DataFrame(data_test_selected)
    df_test.to_csv('mushroom_test_select.csv', index=False)
```

[25]:
```
save_selected_data(xdata_train_expand, ydata_train_expand, xdata_test_expand,␣
↪ydata_test_expand)
```

Perceptron

May 2, 2023

1 Perceptron Learning with MSE

Non-probabilistic (distribution-free) system

Report required performance measures

- F1-score
- Accuracy

1.1 Import necessary library

[8]:
```
import numpy as np
import random as rm
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import confusion_matrix, f1_score
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron, LinearRegression, Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

1.2 Get data from previous cvs file

[17]:
```
def getData(fname1, fname2):
    df_train = pd.read_csv(fname1)
    df_test = pd.read_csv(fname2)
    data_train = np.array(df_train)
    data_test = np.array(df_test)
    xdata_train = data_train[:,:len(data_train[0]) - 1]
    ydata_train = data_train[:, -1]
    xdata_test = data_test[:,:len(data_test[0]) - 1]
    ydata_test = data_test[:, -1]

    return xdata_train, ydata_train, xdata_test, ydata_test
```

```
xdata_train_select, ydata_train_select, xdata_test_select, ydata_test_select =␣
↪getData("mushroom_train_select.csv", "mushroom_test_select.csv")

print(f"The shape of training xdata shape is", xdata_train_select.shape)
print(f"The shape of training ydata shape is", ydata_train_select.shape)
print(f"The shape of testing xdata shape is", xdata_test_select.shape)
print(f"The shape of testing ydata shape is", ydata_test_select.shape)
```

```
The shape of training xdata shape is (42748, 32)
The shape of training ydata shape is (42748,)
The shape of testing xdata shape is (18321, 32)
The shape of testing ydata shape is (18321,)
```

[32]:
```
def Perceptron_experiment(xdata_train_select, ydata_train_select):
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)

    train_loss_history = []
    val_loss_history = []
    train_acc_history = []
    val_acc_history = []
    f1_score_history = []

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)

#     for degree in range(1,8):
    epochs_num = 10
    for epoch in range(epochs_num):
        MSE_train = 0.
        MSE_val = 0.
        acc_train = 0.
        acc_val = 0.
        score = 0.

        # Define the cross-validation object
        cv = KFold(n_splits=4, shuffle=True)
        for i, (train_index, val_index) in enumerate(cv.split(xdata_train)): #␣
↪i in range of 4

            D_train_xdata = xdata_train[train_index]
```

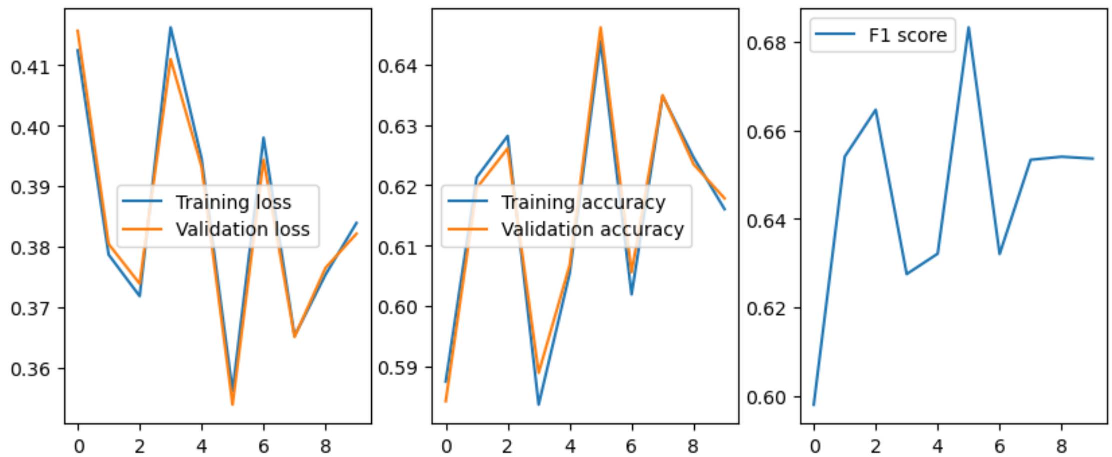
```
            D_train_ydata = ydata_train[train_index]
            D_val_xdata = xdata_train[val_index]
            D_val_ydata = ydata_train[val_index]

            N_train = len(D_train_ydata)
            N_val = len(D_val_ydata)
            print("Fold {}, train data points: {}, validation data points: {}".
↪format(i+1, len(D_train_xdata), len(D_val_xdata)))

            reg = Perceptron(tol=1e-3).fit(D_train_xdata, D_train_ydata)

            predict_train = reg.predict(D_train_xdata)
            predict_val = reg.predict(D_val_xdata)

            # Caluculate the mean squared error for training and validation data
            MSE_train += mean_squared_error(predict_train, D_train_ydata)
            MSE_val += mean_squared_error(predict_val, D_val_ydata)

            acc_train += np.sum(predict_train == D_train_ydata) / N_train
            acc_val += np.sum(predict_val == D_val_ydata) / N_val

            score += f1_score(predict_val, D_val_ydata)

        train_loss_history.append(MSE_train / 4)
        val_loss_history.append(MSE_val / 4)
        train_acc_history.append(acc_train / 4)
        val_acc_history.append(acc_val / 4)
        f1_score_history.append(score / 4)

        print(f"When epoch = {epoch}:")
        print(f"The mean of training loss = {MSE_train / 4}, the mean of␣
↪validation loss = {MSE_val / 4}")
        print(f"The mean of training accuracy = {acc_train / 4}, the mean of␣
↪validation accuracy = {acc_val / 4}")
        print(f"The mean of F1 Score for the system = {score / 4}")

    # Plot the learning curve
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(10, 4))

    ax1.plot(np.arange(epochs_num), train_loss_history, label = "Training loss")
    ax1.plot(np.arange(epochs_num), val_loss_history, label = "Validation loss")
    ax1.legend()

    ax2.plot(np.arange(epochs_num), train_acc_history, label = "Training␣
↪accuracy")
    ax2.plot(np.arange(epochs_num), val_acc_history, label = "Validation␣
↪accuracy")
```

```
    ax2.legend()

    ax3.plot(np.arange(epochs_num), f1_score_history, label = "F1 score")
    ax3.legend()


Perceptron_experiment(xdata_train_select, ydata_train_select)
```

```
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 0:
The mean of training loss = 0.41245594335797386, the mean of validation loss =
0.41569196219706184
The mean of training accuracy = 0.5875440566420261, the mean of validation
accuracy = 0.5843080378029382
The mean of F1 Score for the system = 0.5980920193424242
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 1:
The mean of training loss = 0.3786999781666199, the mean of validation loss =
0.38046224384766536
The mean of training accuracy = 0.6213000218333802, the mean of validation
accuracy = 0.6195377561523345
The mean of F1 Score for the system = 0.6540546770354567
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 2:
The mean of training loss = 0.3718536539721156, the mean of validation loss =
0.373912229811921
The mean of training accuracy = 0.6281463460278843, the mean of validation
accuracy = 0.626087770188079
The mean of F1 Score for the system = 0.6646440899006835
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 3:
The mean of training loss = 0.4162689872430678, the mean of validation loss =
0.4110133807429587
The mean of training accuracy = 0.5837310127569322, the mean of validation
accuracy = 0.5889866192570413
```

```
The mean of F1 Score for the system = 0.6275623787064764
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 4:
The mean of training loss = 0.39456816693178626, the mean of validation loss =
0.3932347712173669
The mean of training accuracy = 0.6054318330682137, the mean of validation
accuracy = 0.6067652287826331
The mean of F1 Score for the system = 0.6321635082593553
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 5:
The mean of training loss = 0.35619600137238394, the mean of validation loss =
0.3539112940956302
The mean of training accuracy = 0.6438039986276161, the mean of validation
accuracy = 0.6460887059043698
The mean of F1 Score for the system = 0.6832327820660058
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 6:
The mean of training loss = 0.3980381148435794, the mean of validation loss =
0.39438102367362216
The mean of training accuracy = 0.6019618851564206, the mean of validation
accuracy = 0.6056189763263777
The mean of F1 Score for the system = 0.6320859662724561
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 7:
The mean of training loss = 0.3652802470291008, the mean of validation loss =
0.3651164966782071
The mean of training accuracy = 0.6347197529708993, the mean of validation
accuracy = 0.6348835033217929
The mean of F1 Score for the system = 0.6533521853069424
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 8:
The mean of training loss = 0.37540157824147713, the mean of validation loss =
0.3765322354262188
```

```
The mean of training accuracy = 0.6245984217585228, the mean of validation
accuracy = 0.6234677645737813
The mean of F1 Score for the system = 0.6540430901559409
Fold 1, train data points: 32061, validation data points: 10687
Fold 2, train data points: 32061, validation data points: 10687
Fold 3, train data points: 32061, validation data points: 10687
Fold 4, train data points: 32061, validation data points: 10687
When epoch = 9:
The mean of training loss = 0.3839477870309722, the mean of validation loss =
0.382169926078413
The mean of training accuracy = 0.6160522129690278, the mean of validation
accuracy = 0.617830073921587
The mean of F1 Score for the system = 0.6536208178977508
```



According to the validation accurcay, this model using perceptron and MSE performs much better than the baseline system which has 55% accuracy. But there is still some space for us to improve this model. For example, we may convert the features to polynomial and apply the data after convert to the perceptron learning.

Although we already reduced the number of features, it is a large amount which will take a lot of time to convert to polynomial. Therefore, we should do the features selection again to avoid this situation.

[21]:
```
def Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, k,␣
↪max_degree):
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)

    # Reduce the feature numbers to k
    fs = SelectKBest(score_func=f_classif, k=k)

    # Apply feature selection
```

```
    xdata_train = fs.fit_transform(xdata_train, ydata_train)

    train_loss_history = []
    val_loss_history = []
    train_acc_history = []
    val_acc_history = []
    f1_score_history = []

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)

    for degree in range(1, max_degree):
        MSE_train = 0.
        MSE_val = 0.
        acc_train = 0.
        acc_val = 0.
        score = 0.

        # Define the cross-validation object
        cv = KFold(n_splits=4, shuffle=True)
        for i, (train_index, val_index) in enumerate(cv.split(xdata_train)): #
↪i in range of 4

            D_train_xdata = xdata_train[train_index]
            D_train_ydata = ydata_train[train_index]
            D_val_xdata = xdata_train[val_index]
            D_val_ydata = ydata_train[val_index]

            N_train = len(D_train_ydata)
            N_val = len(D_val_ydata)

            # Transfer the features to polynomial
            poly = PolynomialFeatures(degree)
            xdata_poly_train = poly.fit_transform(D_train_xdata)
            xdata_poly_val = poly.transform(D_val_xdata)

            reg = Perceptron(tol=1e-3).fit(xdata_poly_train, D_train_ydata)

            predict_train = reg.predict(xdata_poly_train)
            predict_val = reg.predict(xdata_poly_val)

            # Caluculate the mean squared error for training and validation data
```

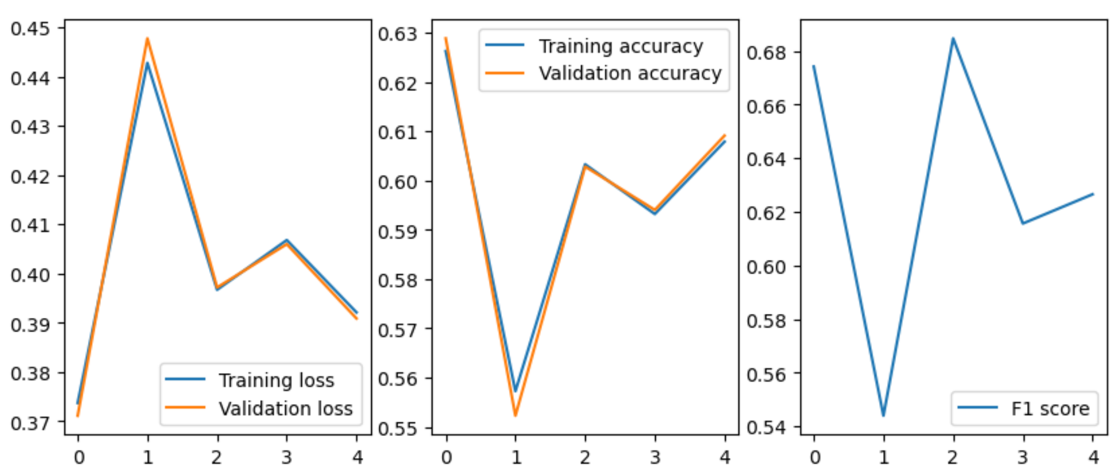
```
            MSE_train += mean_squared_error(predict_train, D_train_ydata)
            MSE_val += mean_squared_error(predict_val, D_val_ydata)

            acc_train += np.sum(predict_train == D_train_ydata) / N_train
            acc_val += np.sum(predict_val == D_val_ydata) / N_val

            score += f1_score(predict_val, D_val_ydata)

        train_loss_history.append(MSE_train / 4)
        val_loss_history.append(MSE_val / 4)
        train_acc_history.append(acc_train / 4)
        val_acc_history.append(acc_val / 4)
        f1_score_history.append(score / 4)


#         print(f"When epoch = {epoch}:")
        print(f"When degree = {degree}:")
        print(f"The mean of training loss = {MSE_train / 4}, the mean of␣
↪validation loss = {MSE_val / 4}")
        print(f"The mean of training accuracy = {acc_train / 4}, the mean of␣
↪validation accuracy = {acc_val / 4}")
        print(f"The mean of F1 Score for the system = {score / 4}")

#         print(train_loss_history)
    # Plot the learning curve
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(10, 4))

    ax1.plot(np.arange(len(train_loss_history)), train_loss_history, label =␣
↪"Training loss")
    ax1.plot(np.arange(len(val_loss_history)), val_loss_history, label =␣
↪"Validation loss")
    ax1.legend()

    ax2.plot(np.arange(len(train_acc_history)), train_acc_history, label =␣
↪"Training accuracy")
    ax2.plot(np.arange(len(val_acc_history)), val_acc_history, label =␣
↪"Validation accuracy")
    ax2.legend()

    ax3.plot(np.arange(len(f1_score_history)), f1_score_history, label = "F1␣
↪score")
    ax3.legend()
```

1.2.1 K = 5

When we reduce the feature numbers to 5, the accuracy of validation is lower than the experiment 1.

[22]:
```
Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, 5, 6)
```

```
When degree = 1:
The mean of training loss = 0.37370949128224323, the mean of validation loss =
0.3711284738467297
The mean of training accuracy = 0.6262905087177568, the mean of validation
accuracy = 0.6288715261532704
The mean of F1 Score for the system = 0.6741979983954344
When degree = 2:
The mean of training loss = 0.44276535354480523, the mean of validation loss =
0.4477402451576682
The mean of training accuracy = 0.5572346464551947, the mean of validation
accuracy = 0.5522597548423318
The mean of F1 Score for the system = 0.5440032285367746
When degree = 3:
The mean of training loss = 0.3966969214934032, the mean of validation loss =
0.3971647796388135
The mean of training accuracy = 0.6033030785065968, the mean of validation
accuracy = 0.6028352203611864
The mean of F1 Score for the system = 0.6847049343313859
When degree = 4:
The mean of training loss = 0.4067870621627523, the mean of validation loss =
0.4059605127725273
The mean of training accuracy = 0.5932129378372477, the mean of validation
accuracy = 0.5940394872274727
The mean of F1 Score for the system = 0.6156615476149195
When degree = 5:
The mean of training loss = 0.3921041140326253, the mean of validation loss =
0.3908720875830448
The mean of training accuracy = 0.6078958859673746, the mean of validation
accuracy = 0.6091279124169552
The mean of F1 Score for the system = 0.6265586090541118
```

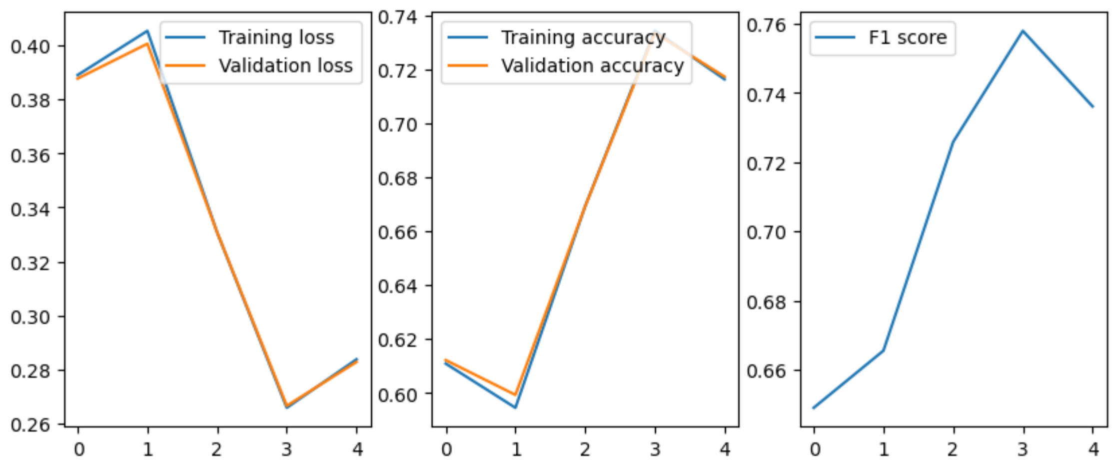


1.2.2 K = 10

When we reduce the feature numbers to 10, the accuracy of validation while degree = 10 is better than experiment 1.

[23]:
```
Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, 10, 6)
```

```
When degree = 1:
The mean of training loss = 0.38907863135897197, the mean of validation loss =
0.38778422382333677
The mean of training accuracy = 0.610921368641028, the mean of validation
accuracy = 0.6122157761766633
The mean of F1 Score for the system = 0.6489957087070615
When degree = 2:
The mean of training loss = 0.4054536664483329, the mean of validation loss =
0.40067371572939087
The mean of training accuracy = 0.5945463335516671, the mean of validation
accuracy = 0.5993262842706092
The mean of F1 Score for the system = 0.6655028824989377
When degree = 3:
The mean of training loss = 0.3307601135335766, the mean of validation loss =
0.33082249461963137
The mean of training accuracy = 0.6692398864664234, the mean of validation
accuracy = 0.6691775053803687
The mean of F1 Score for the system = 0.7258243133510512
When degree = 4:
The mean of training loss = 0.2657122360500296, the mean of validation loss =
0.2664452138111724
The mean of training accuracy = 0.7342877639499704, the mean of validation
accuracy = 0.7335547861888275
The mean of F1 Score for the system = 0.7579325232208303
When degree = 5:
The mean of training loss = 0.28366239356227196, the mean of validation loss =
0.28267989145691025
The mean of training accuracy = 0.716337606437728, the mean of validation
accuracy = 0.7173201085430898
The mean of F1 Score for the system = 0.7361108329407176
```



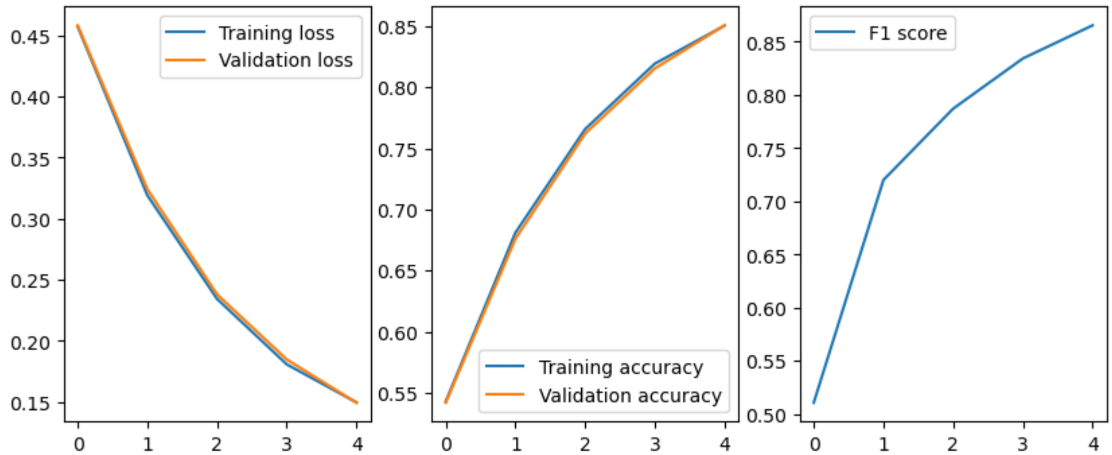
1.2.3 K = 15

When we reduce the feature numbers to 15, the average accuracy of validation is better than the experiment 1 and the highest accuracy = 85%.

[24]:
```
Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, 15, 6)
```

```
When degree = 1:
The mean of training loss = 0.4571285986089018, the mean of validation loss =
0.45805651726396557
The mean of training accuracy = 0.5428714013910982, the mean of validation
accuracy = 0.5419434827360344
The mean of F1 Score for the system = 0.5106525581390742
When degree = 2:
The mean of training loss = 0.31900127881226414, the mean of validation loss =
0.3239917656966408
The mean of training accuracy = 0.6809987211877359, the mean of validation
accuracy = 0.6760082343033593
The mean of F1 Score for the system = 0.7200538515082855
When degree = 3:
The mean of training loss = 0.23431115685724088, the mean of validation loss =
0.23799943857022549
The mean of training accuracy = 0.7656888431427592, the mean of validation
accuracy = 0.7620005614297745
The mean of F1 Score for the system = 0.7870844960109624
When degree = 4:
The mean of training loss = 0.18078818502230123, the mean of validation loss =
0.18475718162253207
The mean of training accuracy = 0.8192118149776987, the mean of validation
accuracy = 0.8152428183774679
The mean of F1 Score for the system = 0.8342741597489958
```

```
When degree = 5:
The mean of training loss = 0.1496912136240292, the mean of validation loss =
0.14945728455132404
The mean of training accuracy = 0.8503087863759708, the mean of validation
accuracy = 0.850542715448676
The mean of F1 Score for the system = 0.8654108397584024
```



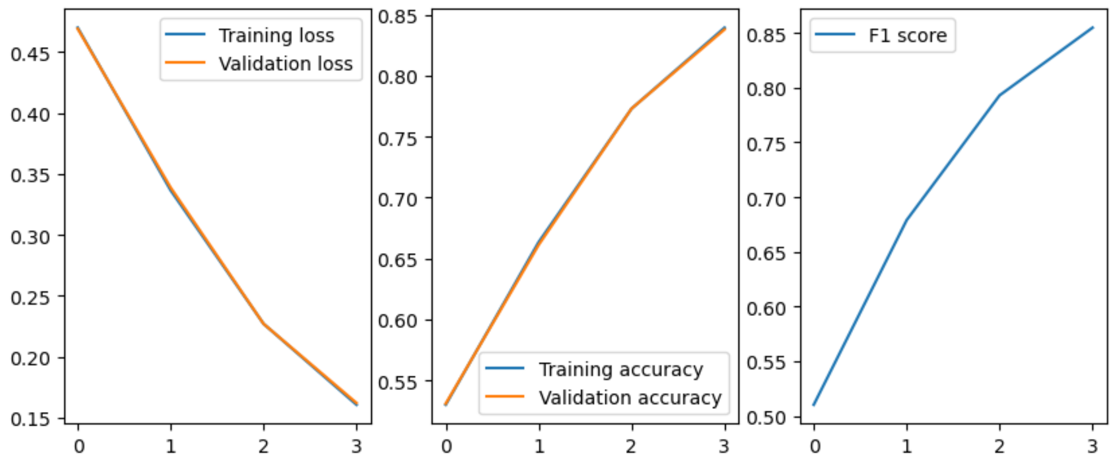
When we try to reduce the features number to 15 and the max degree = 5, the program takes nearly 10 minutes to finish the regression.

According to the preivious result, I decide to reduce the max degree while remaining and increasing the feature numbers to check the performance of model.

[25]:
```
Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, 15, 5)
```

```
When degree = 1:
The mean of training loss = 0.46980755434952115, the mean of validation loss =
0.46893421914475536
The mean of training accuracy = 0.5301924456504787, the mean of validation
accuracy = 0.5310657808552447
The mean of F1 Score for the system = 0.5101443400003552
When degree = 2:
The mean of training loss = 0.3364913758148529, the mean of validation loss =
0.33877608309160656
The mean of training accuracy = 0.6635086241851471, the mean of validation
accuracy = 0.6612239169083933
The mean of F1 Score for the system = 0.6789924272717722
When degree = 3:
The mean of training loss = 0.2270515579676242, the mean of validation loss =
0.22705155796762422
The mean of training accuracy = 0.7729484420323758, the mean of validation
accuracy = 0.7729484420323757
```

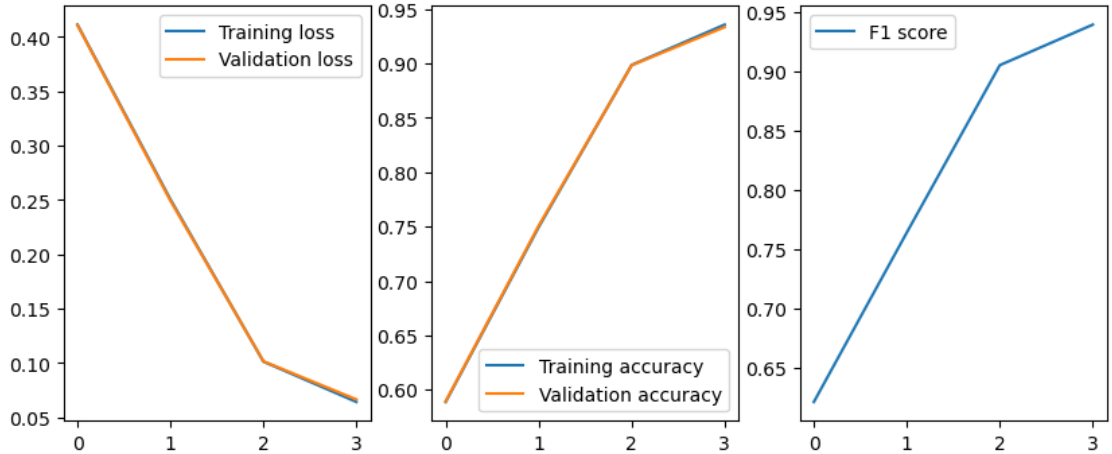
```
The mean of F1 Score for the system = 0.793080295411799
When degree = 4:
The mean of training loss = 0.16070927294844203, the mean of validation loss =
0.16215963319921398
The mean of training accuracy = 0.839290727051558, the mean of validation
accuracy = 0.837840366800786
The mean of F1 Score for the system = 0.8549268677744913
```



[26]:
```
Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, 20, 5)
```

```
When degree = 1:
The mean of training loss = 0.41125510745142074, the mean of validation loss =
0.41073266585571255
The mean of training accuracy = 0.5887448925485792, the mean of validation
accuracy = 0.5892673341442874
The mean of F1 Score for the system = 0.6212335895068034
When degree = 2:
The mean of training loss = 0.2505068463241945, the mean of validation loss =
0.2492514269673435
The mean of training accuracy = 0.7494931536758055, the mean of validation
accuracy = 0.7507485730326565
The mean of F1 Score for the system = 0.7642163032583458
When degree = 3:
The mean of training loss = 0.10140045538192821, the mean of validation loss =
0.10164218209039019
The mean of training accuracy = 0.8985995446180718, the mean of validation
accuracy = 0.8983578179096099
The mean of F1 Score for the system = 0.9053103219346219
When degree = 4:
The mean of training loss = 0.06426811390786315, the mean of validation loss =
0.06655282118461682
The mean of training accuracy = 0.9357318860921369, the mean of validation
```

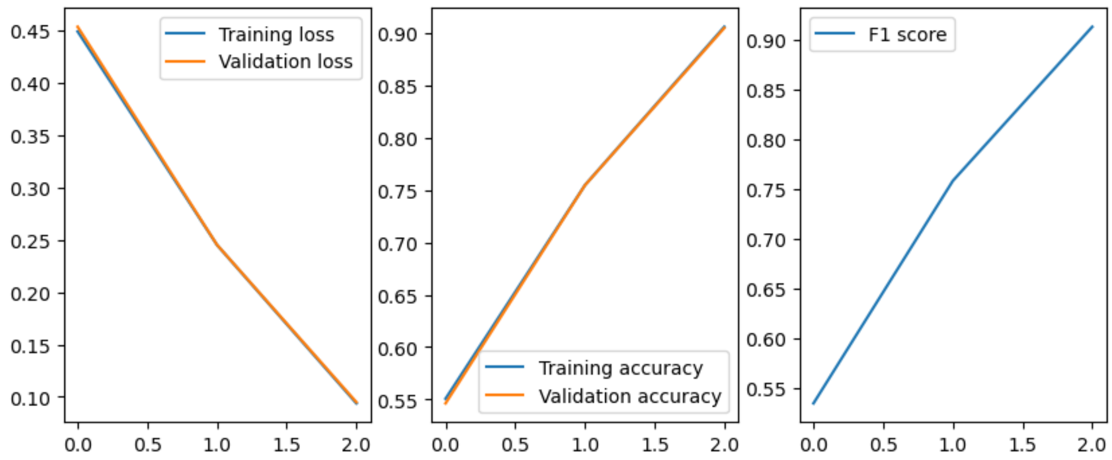
```
accuracy = 0.9334471788153831
The mean of F1 Score for the system = 0.9394831935688306
```



The accuracy and F1 score indicate that this model performs well on this dataset containing 20 features. But still cost a little time to finish. Therefore, I decided to use mode features with less degree to find if there is any improvement.

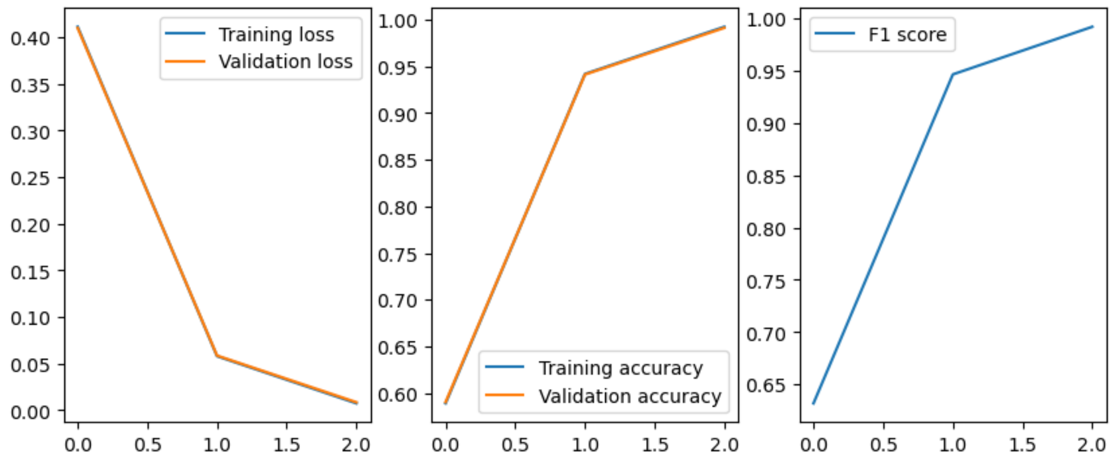
[27]:
```
Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, 20, 4)
```

```
When degree = 1:
The mean of training loss = 0.4493075699447927, the mean of validation loss =
0.45375222232619067
The mean of training accuracy = 0.5506924300552073, the mean of validation
accuracy = 0.5462477776738093
The mean of F1 Score for the system = 0.5341663505964791
When degree = 2:
The mean of training loss = 0.24512647765197593, the mean of validation loss =
0.24529802563862635
The mean of training accuracy = 0.7548735223480241, the mean of validation
accuracy = 0.7547019743613737
The mean of F1 Score for the system = 0.7583560102213616
When degree = 3:
The mean of training loss = 0.09371978416144225, the mean of validation loss =
0.09483484607467016
The mean of training accuracy = 0.9062802158385577, the mean of validation
accuracy = 0.9051651539253298
The mean of F1 Score for the system = 0.9135280424171366
```



[28]:
```
Perceptron_polynomial_experiment(xdata_train_select, ydata_train_select, 30, 4)
```

```
When degree = 1:
The mean of training loss = 0.4110055831072019, the mean of validation loss =
0.40970337793580985
The mean of training accuracy = 0.5889944168927981, the mean of validation
accuracy = 0.5902966220641901
The mean of F1 Score for the system = 0.6318506613545672
When degree = 2:
The mean of training loss = 0.058045600573905995, the mean of validation loss =
0.05869280434172359
The mean of training accuracy = 0.941954399426094, the mean of validation
accuracy = 0.9413071956582764
The mean of F1 Score for the system = 0.9466183196014107
When degree = 3:
The mean of training loss = 0.007524718505349178, the mean of validation loss =
0.008678768597361 28
The mean of training accuracy = 0.9924752814946508, the mean of validation
accuracy = 0.9913212314026387
The mean of F1 Score for the system = 0.9921575173851681
```



1.3 Test

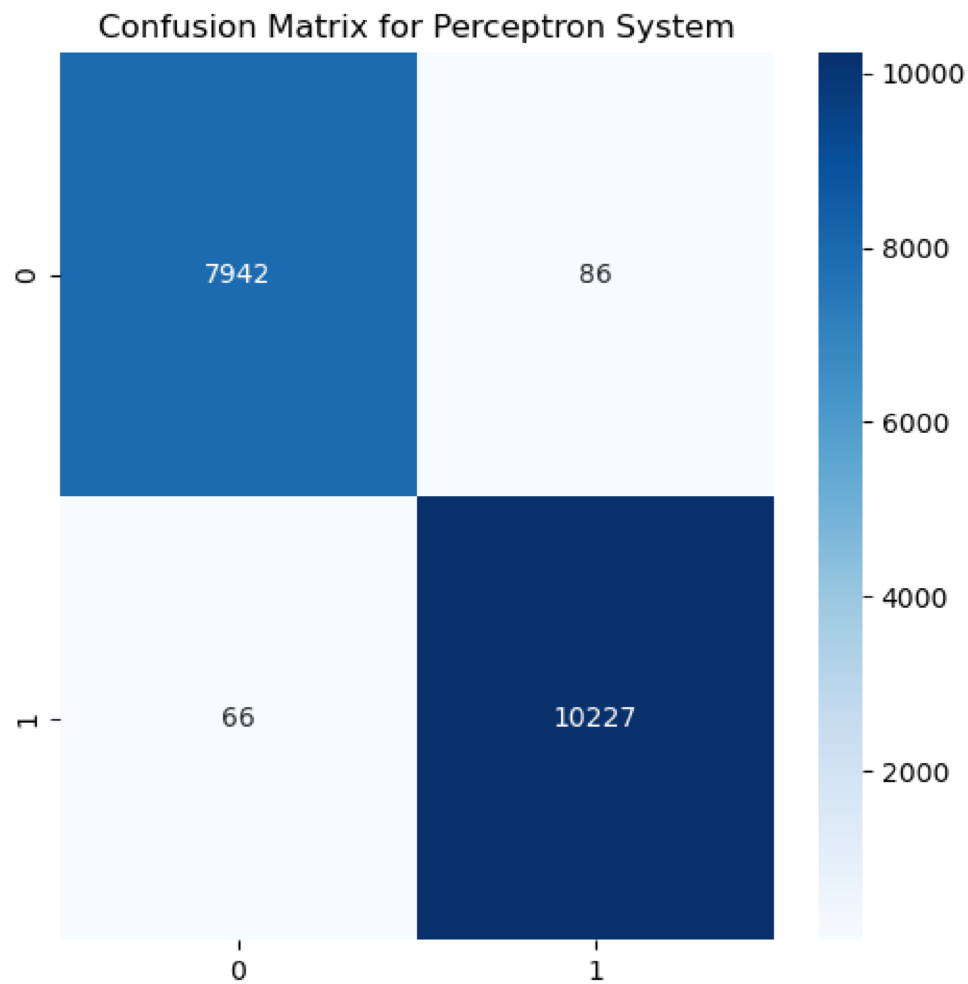
According the previous results, the best model is Perceptron with polynomial features using 30 original features and the highest degree is 3. Use this model to test on the test data and get the final result.

[29]:
```
def test_Perceptron_polynomial(xdata_train_select, ydata_train_select,␣
↪xdata_test_select, ydata_test_select, k, max_degree):
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)
    xdata_test = np.copy(xdata_test_select)
    ydata_test = np.copy(ydata_test_select)
    degree = max_degree

    # Reduce the feature numbers to k
    fs = SelectKBest(score_func=f_classif, k=k)

    # Apply feature selection
    xdata_train = fs.fit_transform(xdata_train, ydata_train)
    xdata_test = fs.transform(xdata_test)

    f1_score_history = []

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)
    xdata_test = scaler.transform(xdata_test)
```

```python
    N_train = len(xdata_train)
    N_test = len(xdata_test)

    # Transfer the features to polynomial
    poly = PolynomialFeatures(degree)
    xdata_poly_train = poly.fit_transform(xdata_train)
    xdata_poly_test = poly.transform(xdata_test)

    reg = Perceptron(tol=1e-3).fit(xdata_poly_train, ydata_train)

    predict_train = reg.predict(xdata_poly_train)
    predict_test = reg.predict(xdata_poly_test)

    # Caluculate the mean squared error for training and validation data
    MSE_train = mean_squared_error(predict_train, ydata_train)
    MSE_test = mean_squared_error(predict_test, ydata_test)

    acc_train = np.sum(predict_train == ydata_train) / N_train
    acc_test = np.sum(predict_test == ydata_test) / N_test

    score = f1_score(predict_test, ydata_test)


    print(f"Degree = {degree}:")
    print(f"The training loss = {MSE_train}")
    print(f"The testing loss = {MSE_test}")
    print(f"The training accuracy = {acc_train}")
    print(f"The testing accuracy = {acc_test}")
    print(f"The F1 Score for the system on test data = {score}")

    # Plot the confusion matrix
    conf_matrix = confusion_matrix(ydata_test, predict_test)
    plt.figure(figsize = (6, 6))
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = 'Blues')
    plt.title("Confusion Matrix for Perceptron System")
```

[30]:
```python
test_Perceptron_polynomial(xdata_train_select, ydata_train_select,␣
 ↪xdata_test_select, ydata_test_select, 30, 3)
```

```
Degree = 3:
The training loss = 0.009591091980911388
The testing loss = 0.008296490366246384
The training accuracy = 0.9904089080190887
The testing accuracy = 0.9917035096337536
The F1 Score for the system on test data = 0.9926235077161991
```


Confusion Matrix for Perceptron System
7942
86
66
10227
0
1
0
1
10000
8000
6000
4000
2000

Support_Vector_Machine

May 2, 2023

1 Support Vector Machine

Support vector system

Report required performance measures

- F1-score
- Accuracy

1.1 Import necessary library

[18]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, f1_score
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import seaborn as sns
```

[19]:

```
def getData(fname1, fname2):
    df_train = pd.read_csv(fname1)
    df_test = pd.read_csv(fname2)
    data_train = np.array(df_train)
    data_test = np.array(df_test)
    xdata_train = data_train[:,:len(data_train[0]) - 1]
    ydata_train = data_train[:, -1]
    xdata_test = data_test[:,:len(data_test[0]) - 1]
    ydata_test = data_test[:, -1]

    return xdata_train, ydata_train, xdata_test, ydata_test

xdata_train_select, ydata_train_select, xdata_test_select, ydata_test_select =␣
↪getData("mushroom_train_select.csv", "mushroom_test_select.csv")

print(f"The shape of training xdata shape is", xdata_train_select.shape)
print(f"The shape of training ydata shape is", ydata_train_select.shape)
print(f"The shape of testing xdata shape is", xdata_test_select.shape)
```

```
print(f"The shape of testing ydata shape is", ydata_test_select.shape)
```

```
The shape of training xdata shape is (42748, 32)
The shape of training ydata shape is (42748,)
The shape of testing xdata shape is (18321, 32)
The shape of testing ydata shape is (18321,)
```

1.2 Linear kernel

To start experiment on the SVM model, we have to choose the kernel and the C value. I decided to use linear kernel first and using a small value c to see how is the performance.

[20]:
```
def SVM_linear(xdata_train_select, ydata_train_select, c):

    # Copy in case of overwrite
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)

    # Shuffle data
    p = np.random.permutation(xdata_train.shape[0])
    xdata_train = xdata_train[p]
    ydata_train = ydata_train[p]

    # Split data into training and validatioin
    xdata_train, xdata_val, ydata_train, ydata_val =␣
↪train_test_split(xdata_train, ydata_train, test_size=0.2, random_state=42)

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)
    xdata_val = scaler.transform(xdata_val)

    model = SVC(C = c, kernel='linear')
    model.fit(xdata_train, ydata_train)

    train_acc = model.score(xdata_train, ydata_train)
    val_acc = model.score(xdata_val, ydata_val)

    train_score = f1_score(model.predict(xdata_train), ydata_train)
    val_score = f1_score(model.predict(xdata_val), ydata_val)
    print('Training accuracy = {}, validatioin accuracy = {}'.format(train_acc,␣
↪val_acc))
```

```
    print('Training f1 score = {}, validatioin f1 score = {}'.
↪format(train_score, val_score))
```

[21]:
```
SVM_linear(xdata_train_select, ydata_train_select, 0.01)
```

```
Training accuracy = 0.7300719340312299, validatioin accuracy = 0.723391812865497
Training f1 score = 0.7589124813915223, validatioin f1 score =
0.7530541923358045
```

[22]:
```
SVM_linear(xdata_train_select, ydata_train_select, 0.1)
```

```
Training accuracy = 0.7274986841335751, validatioin accuracy =
0.7353216374269006
Training f1 score = 0.744705914582363, validatioin f1 score = 0.7502483169628077
```

[23]:
```
SVM_linear(xdata_train_select, ydata_train_select, 1.0)
```

```
Training accuracy = 0.7299257266506813, validatioin accuracy =
0.7259649122807017
Training f1 score = 0.7457748417285989, validatioin f1 score =
0.7377140938094705
```

1.3 Use RBF kernel instead of linear kernel

[24]:
```
def SVM_rbf(xdata_train_select, ydata_train_select, c, gamma):

    # Copy in case of overwrite
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)

    # Shuffle data
    p = np.random.permutation(xdata_train.shape[0])
    xdata_train = xdata_train[p]
    ydata_train = ydata_train[p]

    # Split data into training and validatioin
    xdata_train, xdata_val, ydata_train, ydata_val =␣
↪train_test_split(xdata_train, ydata_train, test_size=0.2, random_state=42)

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)
    xdata_val = scaler.transform(xdata_val)
```

```
    model = SVC(C = c, kernel='rbf', gamma = gamma)
    model.fit(xdata_train, ydata_train)

    train_acc = model.score(xdata_train, ydata_train)
    val_acc = model.score(xdata_val, ydata_val)

    train_score = f1_score(model.predict(xdata_train), ydata_train)
    val_score = f1_score(model.predict(xdata_val), ydata_val)
    print('Training accuracy = {}, validatioin accuracy = {}'.format(train_acc,␣
↪val_acc))
    print('Training f1 score = {}, validatioin f1 score = {}'.
↪format(train_score, val_score))
```

[25]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 0.01, 1.0)
```

```
Training accuracy = 0.7499268963097258, validatioin accuracy =
0.7494736842105263
Training f1 score = 0.8142726838379013, validatioin f1 score =
0.8155991735537189
```

[26]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 0.01, 3.0)
```

```
Training accuracy = 0.6079010468448447, validatioin accuracy = 0.59953216374269
Training f1 score = 0.7381412697482767, validatioin f1 score =
0.7328339575530587
```

[27]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 0.01, 50.0)
```

```
Training accuracy = 0.5513480320486578, validatioin accuracy =
0.5543859649122806
Training f1 score = 0.7107986353269371, validatioin f1 score =
0.7133182844243792
```

[28]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 0.1, 1.0)
```

```
Training accuracy = 0.9958477103924206, validatioin accuracy =
0.9945029239766082
Training f1 score = 0.9962483487450462, validatioin f1 score =
0.9950374828423609
```

[29]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 0.1, 3.0)
```

```
Training accuracy = 0.9990057898122697, validatioin accuracy =
0.9936842105263158
```

```
Training f1 score = 0.9991001005769943, validatioin f1 score =
0.9942869234024546
```

[30]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 0.1, 50.0)
```

```
Training accuracy = 0.7462717117960115, validatioin accuracy =
0.7091228070175438
Training f1 score = 0.813088339831549, validatioin f1 score = 0.7915514206688458
```

[31]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 1.0, 1.0)
```

```
Training accuracy = 0.9996198608105737, validatioin accuracy = 0.99953216374269
Training f1 score = 0.9996556929840824, validatioin f1 score =
0.9995763609404787
```

[32]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 1.0, 3.0)
```

```
Training accuracy = 0.9999707585238903, validatioin accuracy =
0.9992982456140351
Training f1 score = 0.9999734304009352, validatioin f1 score =
0.9993719907891983
```

[33]:
```
SVM_rbf(xdata_train_select, ydata_train_select, 1.0, 50.0)
```

```
Training accuracy = 1.0, validatioin accuracy = 0.9614035087719298
Training f1 score = 1.0, validatioin f1 score = 0.9662921348314606
```

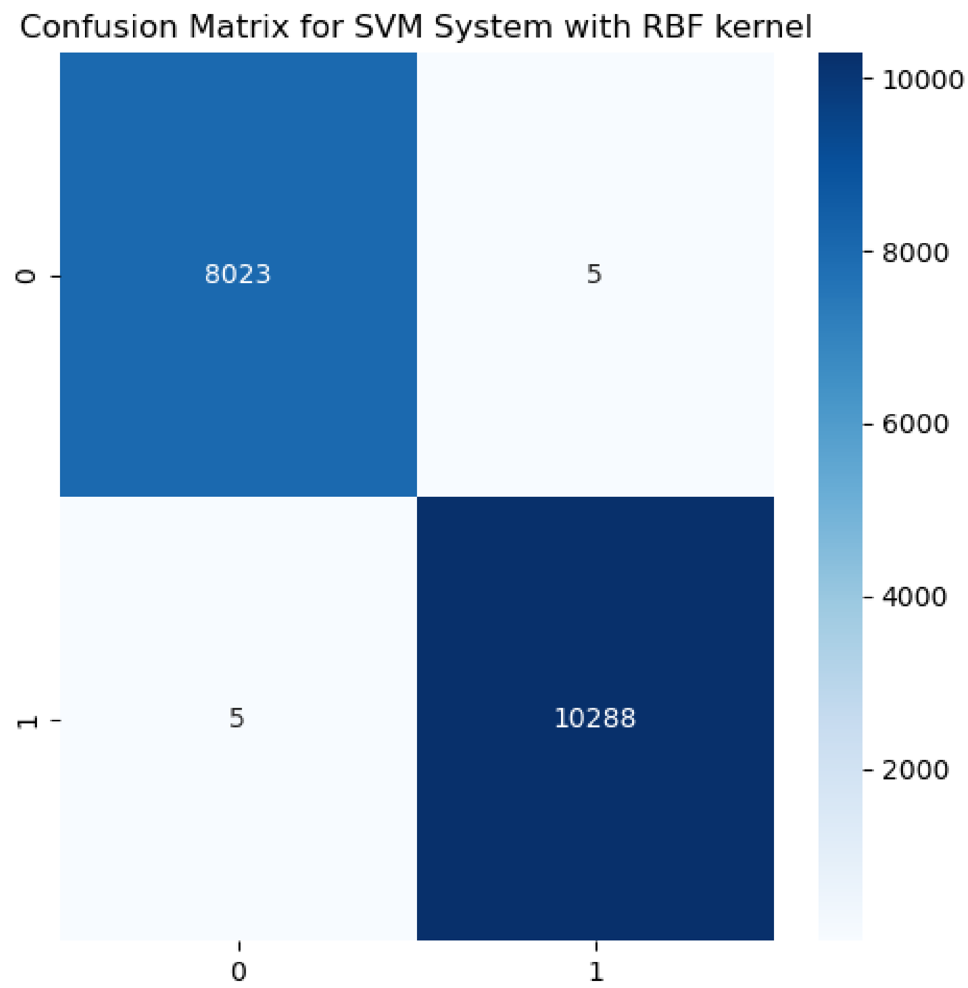
1.4 Test

[16]:
```
def test_SVM_rbf(xdata_train_select, ydata_train_select, xdata_test_select,
 ↪ydata_test_select, c, gamma):

    # Copy in case of overwrite
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)
    xdata_test = np.copy(xdata_test_select)
    ydata_test = np.copy(ydata_test_select)

    # Shuffle data
    p = np.random.permutation(xdata_train.shape[0])
    xdata_train = xdata_train[p]
    ydata_train = ydata_train[p]

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)
```

```
    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)
    xdata_test = scaler.transform(xdata_test)

    model = SVC(C = c, kernel='rbf', gamma = gamma)
    model.fit(xdata_train, ydata_train)

    train_acc = model.score(xdata_train, ydata_train)
    test_acc = model.score(xdata_test, ydata_test)

    train_score = f1_score(model.predict(xdata_train), ydata_train)
    test_score = f1_score(model.predict(xdata_test), ydata_test)

    ydata_predict = model.predict(xdata_test)

    print('Training accuracy = {}, testing accuracy = {}'.format(train_acc,␣
↪test_acc))
    print('Training f1 score = {}, testing f1 score = {}'.format(train_score,␣
↪test_score))


    # Plot the confusion matrix
    conf_matrix = confusion_matrix(ydata_test, ydata_predict)
    plt.figure(figsize = (6, 6))
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = 'Blues')
    plt.title("Confusion Matrix for SVM System with RBF kernel")
```

[34]:
```
test_SVM_rbf(xdata_train_select, ydata_train_select, xdata_test_select,␣
↪ydata_test_select, 1.0, 1.0)
```

```
Training accuracy = 0.9997192851127538, testing accuracy = 0.9994541782653785
Training f1 score = 0.9997457519386415, testing f1 score = 0.9995142329738658
```


Confusion Matrix for SVM System with RBF kernel
8023
5
5
10288
0
1
0
1
10000
8000
6000
4000
2000

```
[ ]:
```

ANN

May 2, 2023

1 Neural Network

neural network system

Report required performance measures

- F1-score • Accuracy

1.1 Import necessary library

[195]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, TensorDataset
from torchvision import datasets, transforms
import torch.nn.functional as F
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, f1_score
import seaborn as sns
```

[200]:
```
def getData(fname1, fname2):
    df_train = pd.read_csv(fname1)
    df_test = pd.read_csv(fname2)
    data_train = np.array(df_train)
    data_test = np.array(df_test)
    xdata_train = data_train[:,:len(data_train[0]) - 1]
    ydata_train = data_train[:, -1]
    xdata_test = data_test[:,:len(data_test[0]) - 1]
    ydata_test = data_test[:, -1]

    return xdata_train, ydata_train, xdata_test, ydata_test

xdata_train_select, ydata_train_select, xdata_test_select, ydata_test_select =␣
 ↪getData("mushroom_train_select.csv", "mushroom_test_select.csv")
```

```
print(f"The shape of training xdata shape is", xdata_train_select.shape)
print(f"The shape of training ydata shape is", ydata_train_select.shape)
print(f"The shape of testing xdata shape is", xdata_test_select.shape)
print(f"The shape of testing ydata shape is", ydata_test_select.shape)
```

```
The shape of training xdata shape is (42748, 32)
The shape of training ydata shape is (42748,)
The shape of testing xdata shape is (18321, 32)
The shape of testing ydata shape is (18321,)
```

[205]:
```
class Model(nn.Module):
    def __init__(self, input_features): # Define layers in the constructor
        super().__init__()
        self.fc1 = nn.Linear(input_features, 16)
        self.fc2 = nn.Linear(16, 8)
        self.fc3 = nn.Linear(8, 2)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x): # Define forward pass in the forward method
#         x = x.view(x.shape[0], -1)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        x = self.sigmoid(x)
        return x
```

[220]:
```
def train_validate_model(xdata_train_select, ydata_train_select, lr = 0.01,␣
 ↪reg_val = 1e-4, batch_size = 32):
    # Copy in case of overwrite
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)

    # Split data
    xdata_train, xdata_val, ydata_train, ydata_val =␣
 ↪train_test_split(xdata_train, ydata_train, test_size=0.2, random_state=42)

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)
    xdata_val = scaler.transform(xdata_val)
```

```
    training_loss_history = []
    val_loss_history = []
    training_acc_history = []
    val_acc_history = []
    f1_score_history = []

    # Convert your data to PyTorch tensors
    xdata_train = torch.Tensor(xdata_train)  # Shape: (num_samples,␣
↪num_features)
    ydata_train = torch.Tensor(ydata_train)  # Shape: (num_samples,)
    xdata_val = torch.Tensor(xdata_val)  # Shape: (num_samples, num_features)
    ydata_val = torch.Tensor(ydata_val)  # Shape: (num_samples,)

    # Create a PyTorch TensorDataset object
    trainset = TensorDataset(xdata_train, ydata_train)
    valset = TensorDataset(xdata_val, ydata_val)

    # Create a PyTorch DataLoader object
    trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
    valloader = DataLoader(valset, batch_size=batch_size, shuffle=False)

    model = Model(32)
    epoch_nums = 30

    device = torch.device("cpu")
    model.to(device) # Move model to device
    criterion = nn.CrossEntropyLoss()
#     criterion = nn.BCELoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, weight_decay=reg_val)
#     optimizer = optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(epoch_nums):

        # Training the model
        model.train() # set model to training mode
        running_loss = 0
        running_acc = 0
        for i, data in enumerate(trainloader):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device) # Move batch␣
↪to device
            optimizer.zero_grad()

            output = model(inputs) # Forward pass
            loss = criterion(output, labels.long()) # Compute loss
            loss.backward() # Backward pass
```

```
            optimizer.step() # Update weights
            running_loss += loss.item()
            predictions = torch.argmax(output, dim=1)
            running_acc += torch.sum(predictions == labels).item()
#         print(f"Training loss = {running_loss / len(trainset)}, accuracy =␣
↪{running_acc / len(trainset) }")
        training_loss_history.append(running_loss / len(trainset))
        training_acc_history.append(running_acc / len(trainset))

        # Eval the model on validation
        model.eval()
        running_loss = 0
        running_acc = 0
        score = 0
        with torch.no_grad():
            for i, data in enumerate(valloader):
                inputs, labels = data
                inputs, labels = inputs.to(device), labels.to(device) # Move␣
↪batch to device
#                 optimizer.zero_grad()

                output = model(inputs) # Forward pass
                loss = criterion(output, labels.long()) # Compute loss
                running_loss += loss.item()
                predictions = torch.argmax(output, dim=1)
                running_acc += torch.sum(predictions == labels).item()
                score += f1_score(predictions, labels)
#         print(f"Validation loss = {running_loss / len(valset)}, accuracy =␣
↪{running_acc / len(valset) }")
        val_loss_history.append(running_loss / len(valset))
        val_acc_history.append(running_acc / len(valset))
        f1_score_history.append(score / len(valloader))

    print(f"The mean of training loss = {np.mean(training_loss_history)}, std =␣
↪{np.std(training_loss_history)}")
    print(f"The mean of validation loss = {np.mean(val_loss_history)}, std =␣
↪{np.std(val_loss_history)}")
    print(f"The mean of training accuracy = {np.mean(training_acc_history)},␣
↪std = {np.std(training_acc_history)}")
    print(f"The mean of validation accuracy = {np.mean(val_acc_history)}, std =␣
↪{np.std(val_acc_history)}")
    print(f"The mean of f1_score = {np.mean(f1_score_history)}, std = {np.
↪std(f1_score_history)}")

    # Plot the learning curve
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
```

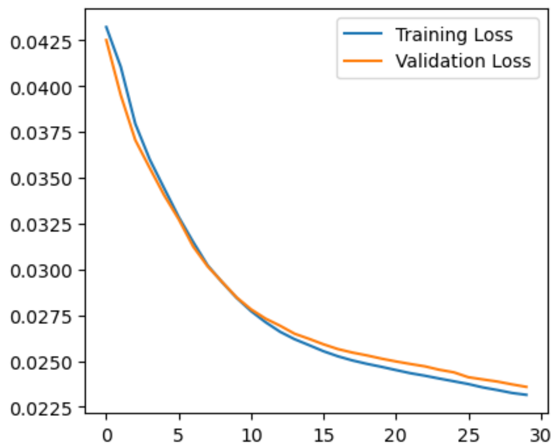
```
    ax1.plot(np.arange(len(training_loss_history)), training_loss_history,␣
↪label = "Training Loss")
    ax1.plot(np.arange(len(val_loss_history)), val_loss_history, label =␣
↪"Validation Loss")
    ax1.legend()

    ax2.plot(np.arange(len(training_acc_history)), training_acc_history, label␣
↪= "Training Accuracy")
    ax2.plot(np.arange(len(val_acc_history)), val_acc_history, label =␣
↪"Validation Accuracy")
    ax2.legend()
```

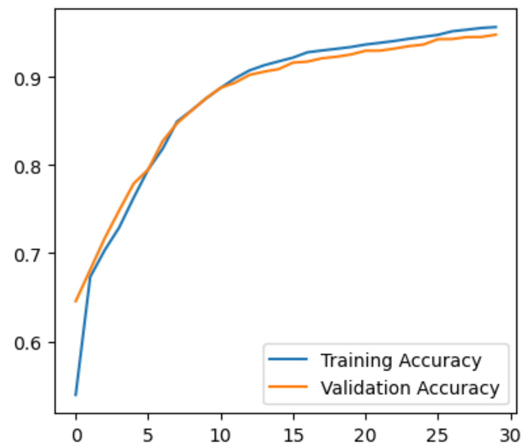
1.2 Experiment to find the best batch size

Parameter setting : lr = 0.01, reg_val = 1e-3, batch_size = 16

[221]:
```
train_validate_model(xdata_train_select, ydata_train_select, lr = 0.01, reg_val␣
↪= 1e-3, batch_size = 16)
```

```
The mean of training loss = 0.02806407604771409, std = 0.005419674325229932
The mean of validation loss = 0.02818495693959688, std = 0.004958805406917511
The mean of training accuracy = 0.8750521472990622, std = 0.09991276962751165
The mean of validation accuracy = 0.8757115009746589, std = 0.08266862883154323
The mean of f1_score = 0.8867927710321727, std = 0.06967886831726633
```



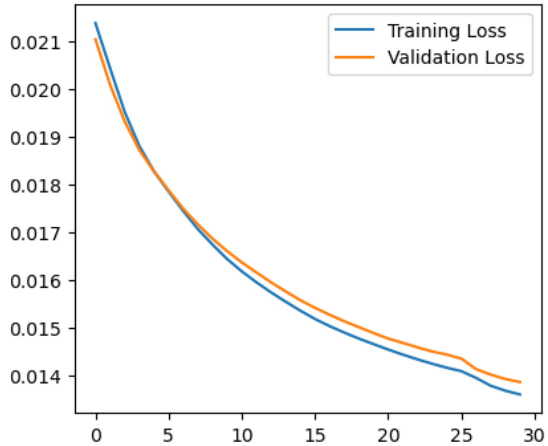


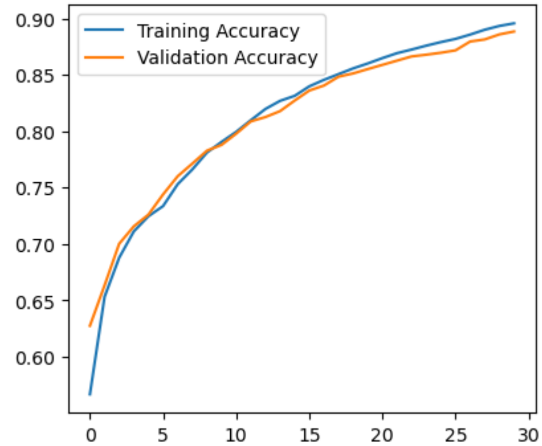
Parameter setting : lr = 0.01, reg_val = 1e-3, batch_size = 32

[208]:
```
train_validate_model(xdata_train_select, ydata_train_select, lr = 0.01, reg_val␣
↪= 1e-3, batch_size = 32)
```

```
The mean of training loss = 0.015931480860369458, std = 0.002050862596269529
```

```
The mean of validation loss = 0.016069252384452563, std = 0.0018896076273431904
The mean of training accuracy = 0.8104479794140009, std = 0.07929587844170442
The mean of validation accuracy = 0.8101598440545809, std = 0.06849869294670174
The mean of f1_score = 0.830374383938514, std = 0.05719182195475713
```

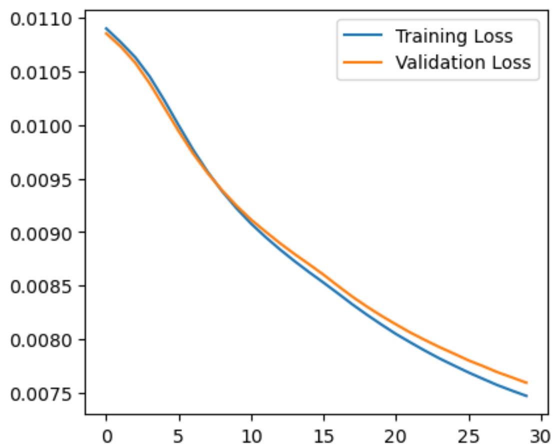


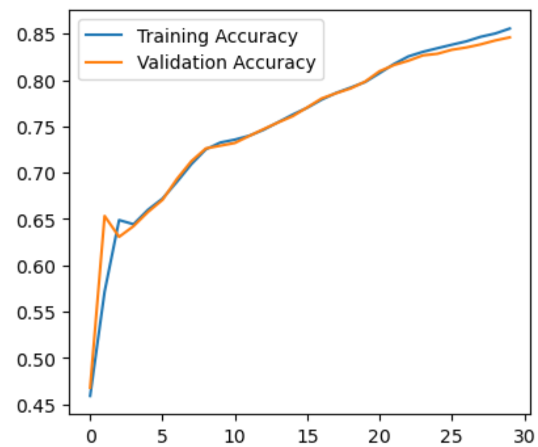


Parameter setting : lr = 0.01, reg_val = 1e-3, batch_size = 64

[209]:
```
train_validate_model(xdata_train_select, ydata_train_select, lr = 0.01, reg_val
  ↪= 1e-3, batch_size = 64)
```

```
The mean of training loss = 0.008804726006628131, std = 0.0010457066640599
The mean of validation loss = 0.00885196719636694, std = 0.0009841559074110435
The mean of training accuracy = 0.7507953681501841, std = 0.08923572772817882
The mean of validation accuracy = 0.751317738791423, std = 0.08302977539882886
The mean of f1_score = 0.7699016943966448, std = 0.12676946661504343
```





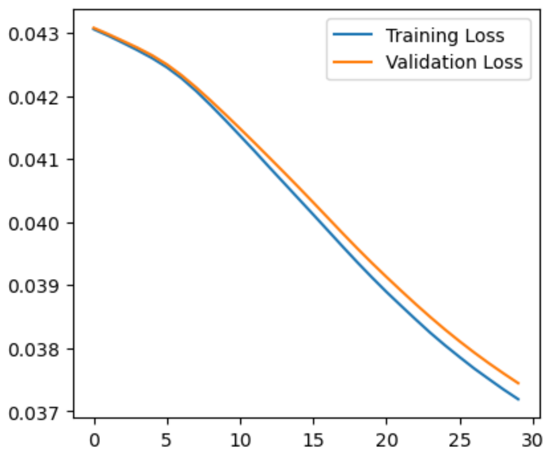
1.3 Experiment to find the best learning rate

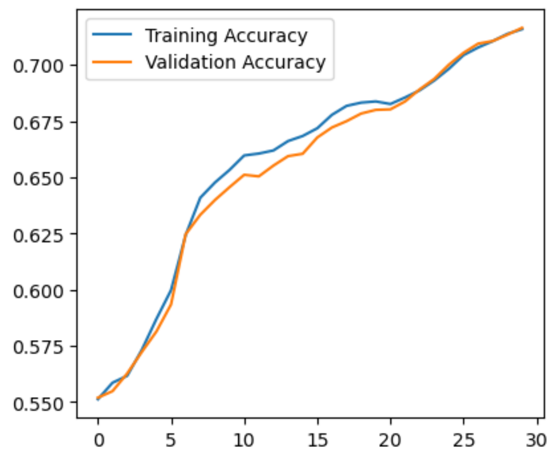
Using the best performance batch size = 16

Parameter setting : lr = 0.001, reg_val = 1e-3, batch_size = 16

[210]:
```
train_validate_model(xdata_train_select, ydata_train_select, lr = 0.001,␣
↪reg_val = 1e-3, batch_size = 16)
```

```
The mean of training loss = 0.040225672157909446, std = 0.0019115341637570985
The mean of validation loss = 0.040384307952652194, std = 0.001827021920769275
The mean of training accuracy = 0.657113476421623, std = 0.04808617528940006
The mean of validation accuracy = 0.6537465886939571, std = 0.04863427110036571
The mean of f1_score = 0.7163676567613724, std = 0.01054172798961895
```

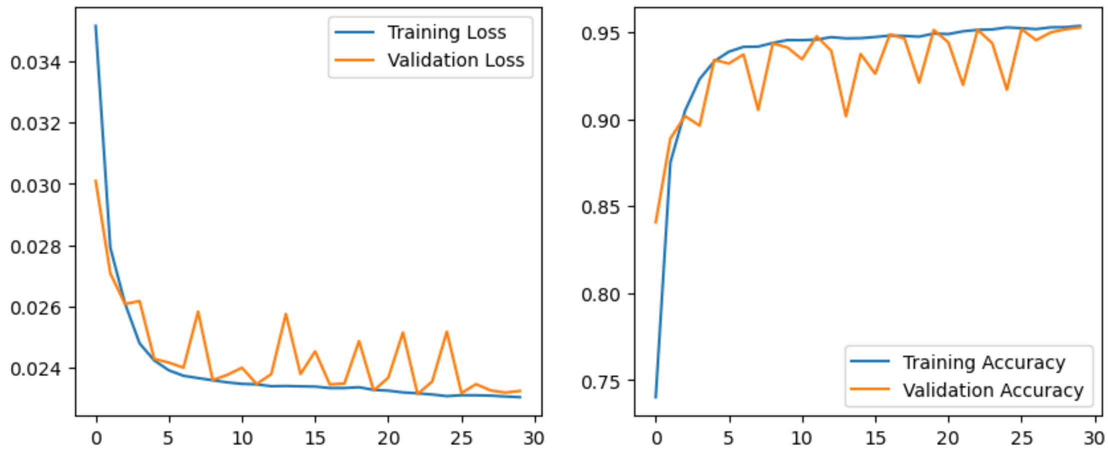




Parameter setting : lr = 0.1, reg_val = 1e-3, batch_size = 16

[211]:
```
train_validate_model(xdata_train_select, ydata_train_select, lr = 0.1, reg_val␣
↪= 1e-3, batch_size = 16)
```

```
The mean of training loss = 0.024054208303747195, std = 0.002287677135720185
The mean of validation loss = 0.02441616344370573, std = 0.0014862699659457892
The mean of training accuracy = 0.9358451761311577, std = 0.03964709215190119
The mean of validation accuracy = 0.9300272904483431, std = 0.024647633526438537
The mean of f1_score = 0.9338656536743228, std = 0.02355154985902897
```

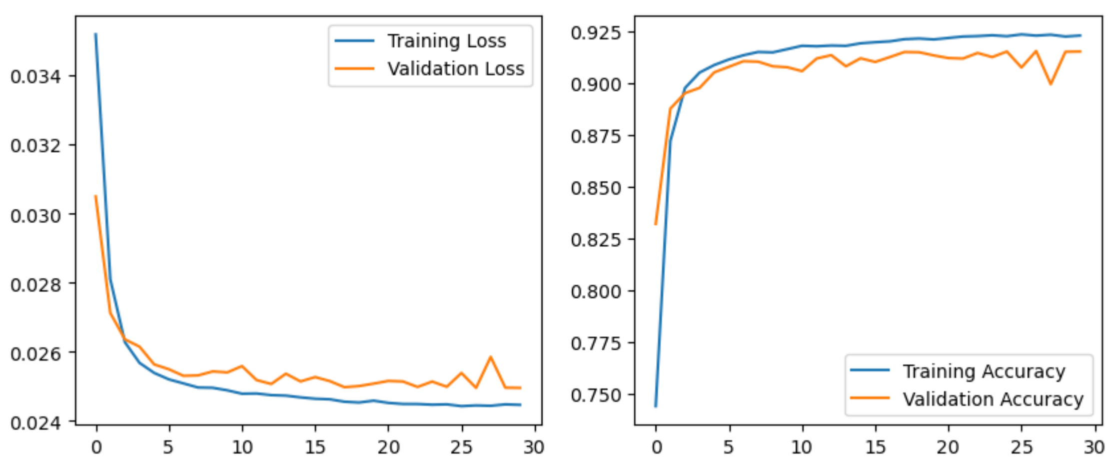


1.4 Experiment to find the best coefficient

Using the best performance batch size = 16, lr = 0.1 Parameter setting : lr = 0.1, reg_val = 1e-4, batch_size = 16

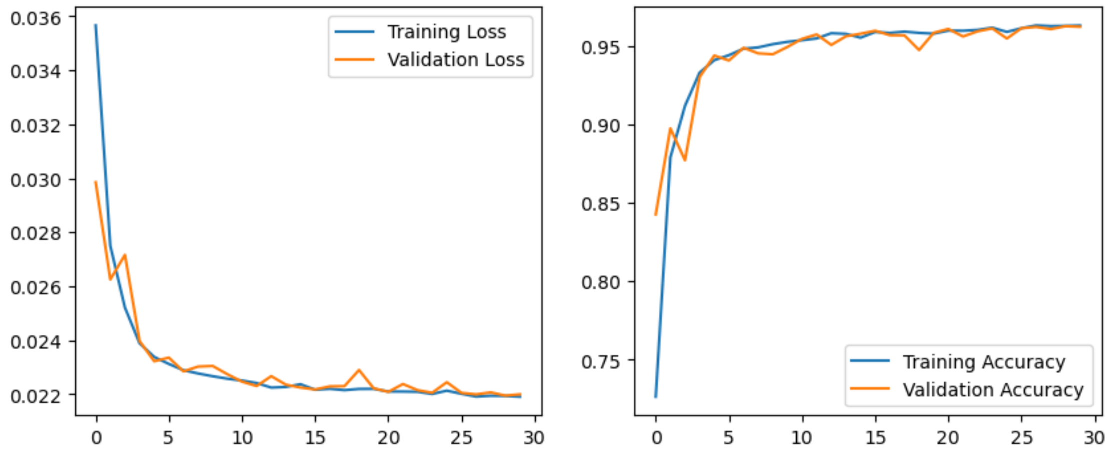
[212]:
```
train_validate_model(xdata_train_select, ydata_train_select, lr = 0.1, reg_val = 1e-4, batch_size = 16)
```

```
The mean of training loss = 0.025237053631138522, std = 0.00198371750628052
The mean of validation loss = 0.025535486472628967, std = 0.0010341412458917713
The mean of training accuracy = 0.9107871805368735, std = 0.03259039264385906
The mean of validation accuracy = 0.9066939571150099, std = 0.015304118768574543
The mean of f1_score = 0.9079169272051765, std = 0.014595463859467915
```



[213]:
```
train_validate_model(xdata_train_select, ydata_train_select, lr = 0.1, reg_val = 1e-5, batch_size = 16)
```

```
The mean of training loss = 0.02307879770789404, std = 0.0025948332765113407
The mean of validation loss = 0.023012959576257253, std = 0.001716036511888241
The mean of training accuracy = 0.9440308400101372, std = 0.04386727621096593
The mean of validation accuracy = 0.9458713450292398, std = 0.026582000266429335
The mean of f1_score = 0.949060551416899, std = 0.02506985855125152
```



1.5 Test

According to the experiments above, the best parameter setting is lr = 0.1, reg_val = 1e-5, batch_size = 16

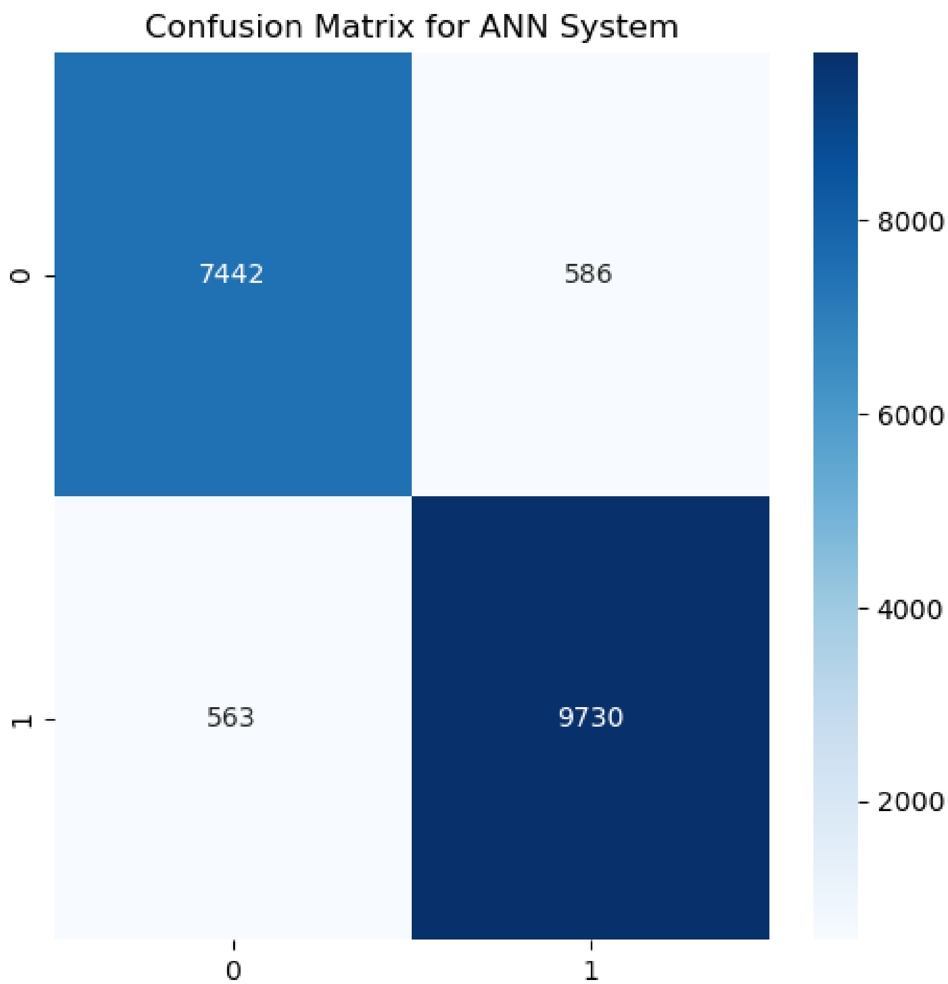
[216]:
```python
def test_model(xdata_train_select, ydata_train_select, xdata_test_select, ydata_test_select, lr = 0.1, reg_val = 1e-5, batch_size = 16):
    # Copy in case of overwrite
    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)
    xdata_test = np.copy(xdata_test_select)
    ydata_test = np.copy(ydata_test_select)

    # Normalize dataset
    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)
    xdata_test = scaler.transform(xdata_test)


    # Convert your data to PyTorch tensors
    xdata_train = torch.Tensor(xdata_train)  # Shape: (num_samples, num_features)
```

```
    ydata_train = torch.Tensor(ydata_train)  # Shape: (num_samples,)
    xdata_test = torch.Tensor(xdata_test)  # Shape: (num_samples, num_features)
    ydata_test = torch.Tensor(ydata_test)  # Shape: (num_samples,)

    # Create a PyTorch TensorDataset object
    trainset = TensorDataset(xdata_train, ydata_train)
    testset = TensorDataset(xdata_test, ydata_test)

    # Create a PyTorch DataLoader object
    trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
    testloader = DataLoader(testset, batch_size=batch_size, shuffle=False)

    model = Model(32)
    epoch_nums = 30

    device = torch.device("cpu")
    model.to(device) # Move model to device
    criterion = nn.CrossEntropyLoss()
#     criterion = nn.BCELoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, weight_decay=reg_val)
#     optimizer = optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(epoch_nums):

        # Training the model
        model.train() # set model to training mode
        for i, data in enumerate(trainloader):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device) # Move batch␣
 ↪to device
            optimizer.zero_grad()

            output = model(inputs) # Forward pass
            loss = criterion(output, labels.long()) # Compute loss
            loss.backward() # Backward pass
            optimizer.step() # Update weights

    # Eval the model on validation
    model.eval()
    running_acc = 0
    score = 0
    list_target = []
    list_pred = []
    with torch.no_grad():
        for i, data in enumerate(testloader):
            inputs, labels = data
```

```
            inputs, labels = inputs.to(device), labels.to(device) # Move batch␣
↪to device
            list_target.extend(labels)
            output = model(inputs) # Forward pass
            predictions = torch.argmax(output, dim=1)
            list_pred.extend(predictions)
            running_acc += torch.sum(predictions == labels).item()
            score += f1_score(predictions, labels)

    print(f"Testing accuracy = {running_acc / len(testset)}")
    print(f"The f1_score = {score / len(testloader)}")

    # Plot the confusion matrix
    conf_matrix = confusion_matrix(list_target, list_pred)
    plt.figure(figsize = (6, 6))
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = 'Blues')
    plt.title("Confusion Matrix for ANN System")
```

[217]:
```
test_model(xdata_train_select, ydata_train_select, xdata_test_select,␣
↪ydata_test_select, lr = 0.1, reg_val = 1e-5, batch_size = 16)
```

```
Testing accuracy = 0.9372850826919928
The f1_score = 0.9410237582397579
```


Confusion Matrix for ANN System
7442
586
563
9730
0
1
0
1
8000
6000
4000
2000

KNN

May 2, 2023

[17]:
```
import numpy as np
import random as rm
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, f1_score
import matplotlib.pyplot as plt
# from sklearn.linear_model import Perceptron, LinearRegression, Ridge
# from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
# from sklearn.feature_selection import SelectKBest
# from sklearn.feature_selection import f_classif
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

[26]:
```
def getData(fname1, fname2):
    df_train = pd.read_csv(fname1)
    df_test = pd.read_csv(fname2)
    data_train = np.array(df_train)
    data_test = np.array(df_test)
    xdata_train = data_train[:,:len(data_train[0]) - 1]
    ydata_train = data_train[:, -1]
    xdata_test = data_test[:,:len(data_test[0]) - 1]
    ydata_test = data_test[:, -1]

    return xdata_train, ydata_train, xdata_test, ydata_test

xdata_train_select, ydata_train_select, xdata_test_select, ydata_test_select =␣
 ↪getData("mushroom_train_select.csv", "mushroom_test_select.csv")

print(f"The shape of training xdata shape is", xdata_train_select.shape)
print(f"The shape of training ydata shape is", ydata_train_select.shape)
print(f"The shape of testing xdata shape is", xdata_test_select.shape)
print(f"The shape of testing ydata shape is", ydata_test_select.shape)
```

```
The shape of training xdata shape is (42748, 32)
```

```
The shape of training ydata shape is (42748,)
The shape of testing xdata shape is (18321, 32)
The shape of testing ydata shape is (18321,)
```

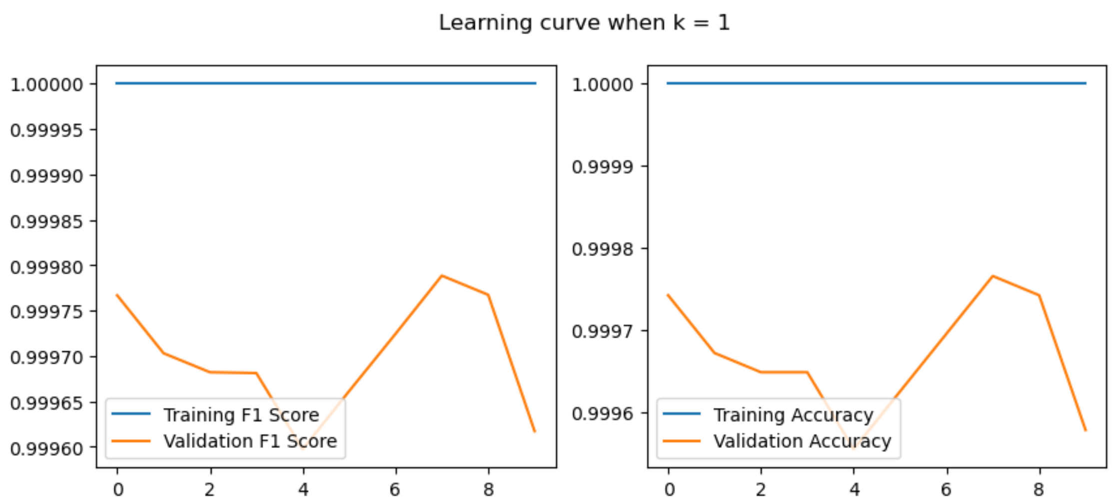
[10]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

def KNN_experiment_PCA(xdata_train_select, ydata_train_select, k, components):

    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)

    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)

    f1_score_train_history = []
    f1_score_val_history = []
    acc_train_history = []
    acc_val_history = []

    pca = PCA(n_components = components)
    xdata_train = pca.fit_transform(xdata_train)

    nums_epoch = 10
    for epoch in range(nums_epoch):
        f1_scores_train = []
        f1_scores_val = []
        acc_train_epoch = []
        acc_val_epoch = []

        # use cross-validation with 4 folds
        cv = KFold(n_splits=4,shuffle=True)
        for i, (train_index, val_index) in enumerate(cv.split(xdata_train)): #␣
 ↪i in range of 4

            D_train_xdata = xdata_train[train_index]
            D_train_ydata = ydata_train[train_index]
            D_val_xdata = xdata_train[val_index]
            D_val_ydata = ydata_train[val_index]

            # create a KNN classifier
            knn = KNeighborsClassifier(n_neighbors=k)
            knn.fit(D_train_xdata, D_train_ydata)
```

```
            # predict validation labels using KNN
            ydata_train_pred = knn.predict(D_train_xdata)
            ydata_val_pred = knn.predict(D_val_xdata)

            # calculate validation f1_score and record for each fold
            f1_train = f1_score(D_train_ydata, ydata_train_pred)
            f1_val = f1_score(D_val_ydata, ydata_val_pred)
            f1_scores_train.append(f1_train)
            f1_scores_val.append(f1_val)

            # calculate accuracy and record for each fold
            acc_train = np.sum(D_train_ydata==ydata_train_pred) /␣
↪len(D_train_ydata)
            acc_val = np.sum(D_val_ydata==ydata_val_pred) / len(D_val_ydata)
            acc_train_epoch.append(acc_train)
            acc_val_epoch.append(acc_val)


        f1_score_train_history.append(np.mean(f1_scores_train))
        f1_score_val_history.append(np.mean(f1_scores_val))
        acc_train_history.append(np.mean(acc_train_epoch))
        acc_val_history.append(np.mean(acc_val_epoch))

    print(f"The mean of training F1 Score = {np.mean(f1_score_val_history)},␣
↪std = {np.std(f1_score_val_history)}")
    print(f"The mean of training accuracy = {np.mean(acc_train_history)}, std =␣
↪{np.std(acc_train_history)}")
    print(f"The mean of validation F1 Score = {np.mean(f1_score_val_history)},␣
↪std = {np.std(f1_score_val_history)}")
    print(f"The mean of validation accuracy = {np.mean(acc_val_history)}, std =␣
↪{np.std(acc_val_history)}")

    # Plot the learning curve
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
    fig.suptitle(f"Learning curve when k = {k}")
    ax1.plot(np.arange(len(f1_score_train_history)), f1_score_train_history,␣
↪label = "Training F1 Score")
    ax1.plot(np.arange(len(f1_score_val_history)), f1_score_val_history, label␣
↪= "Validation F1 Score")
    ax1.legend()

    ax2.plot(np.arange(len(acc_train_history)), acc_train_history, label =␣
↪"Training Accuracy")
    ax2.plot(np.arange(len(acc_val_history)), acc_val_history, label =␣
↪"Validation Accuracy")
```

```
    ax2.legend()
```

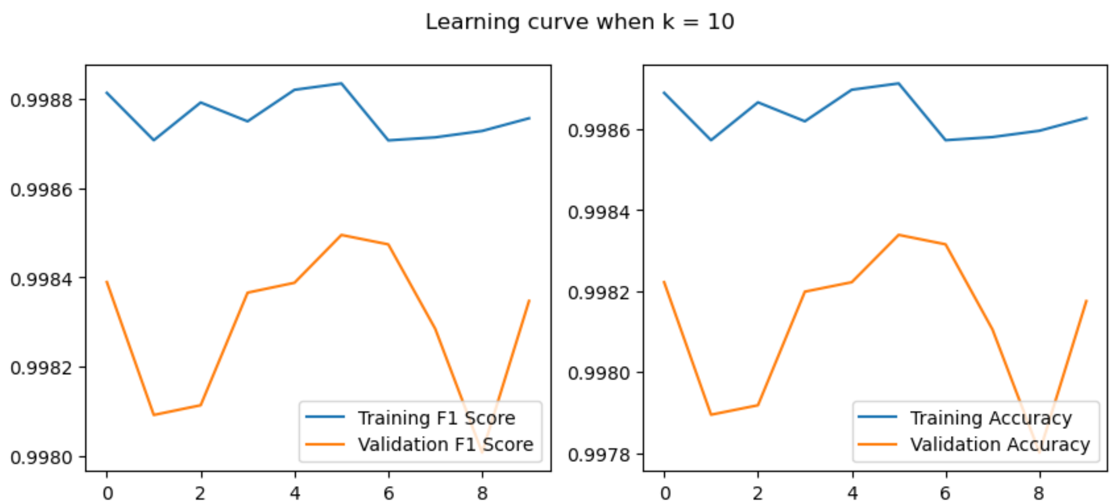
[4]:
```
KNN_experiment_PCA(xdata_train_select, ydata_train_select, 1, 30)
```

```
The mean of training F1 Score = 0.9996989055496751, std = 6.081453158572446e-05
The mean of training accuracy = 1.0, std = 0.0
The mean of validation F1 Score = 0.9996989055496751, std =
6.081453158572446e-05
The mean of validation accuracy = 0.9996678207167585, std =
6.682350920317539e-05
```



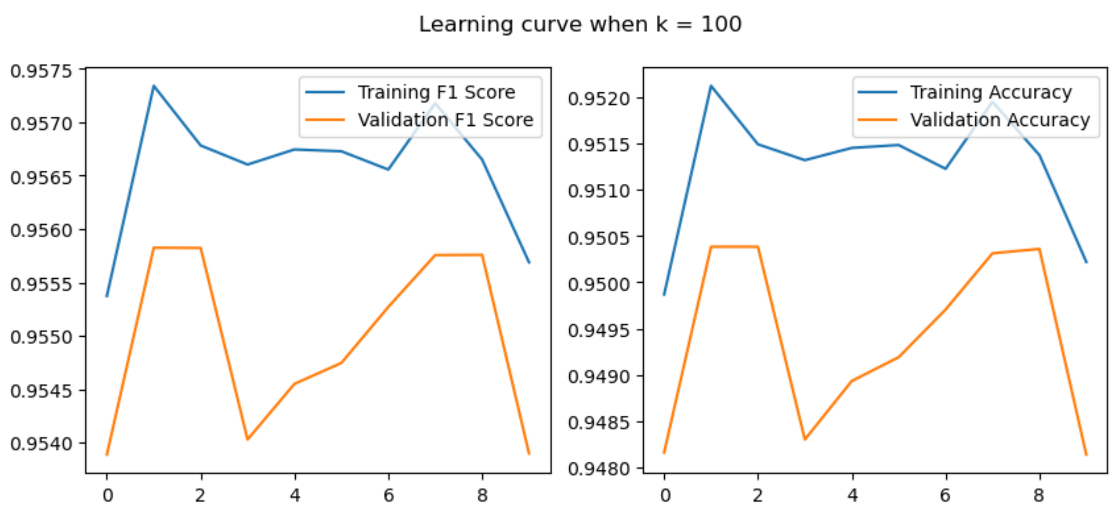
[5]:
```
KNN_experiment_PCA(xdata_train_select, ydata_train_select, 10, 30)
```

```
The mean of training F1 Score = 0.9982959674819609, std = 0.0001592041730821321
The mean of training accuracy = 0.9986338542154017, std = 5.146439599515423e-05
The mean of validation F1 Score = 0.9982959674819609, std =
0.0001592041730821321
The mean of validation accuracy = 0.9981192102554507, std =
0.0001760539695165058
```



[7]:
```
KNN_experiment_PCA(xdata_train_select, ydata_train_select, 100, 15)
```

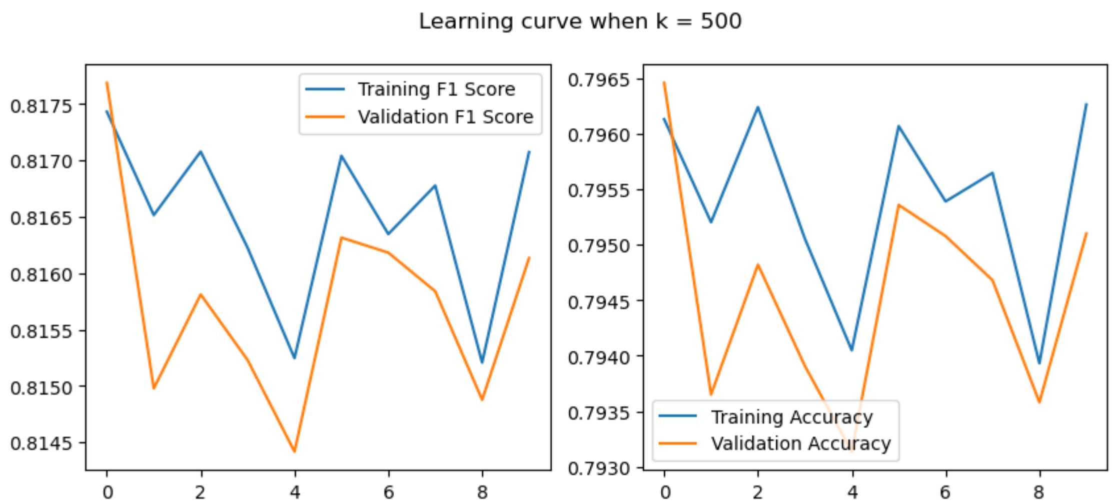
```
The mean of training F1 Score = 0.9549543294182646, std = 0.0007882684482933164
The mean of training accuracy = 0.9512507407753971, std = 0.000662934796027274
The mean of validation F1 Score = 0.9549543294182646, std =
0.0007882684482933164
The mean of validation accuracy = 0.9493871058295126, std = 0.00091532951793459
```



[13]:
```
KNN_experiment_PCA(xdata_train_select, ydata_train_select, 500, 15)
```

```
The mean of training F1 Score = 0.8157477354679628, std = 0.0008846526917039745
The mean of training accuracy = 0.7953978353763139, std = 0.0008143808376661318
The mean of validation F1 Score = 0.8157477354679628, std =
```

0.0008846526917039745
The mean of validation accuracy = 0.7945775240946945, std =
0.0009548891909279568



[23]:

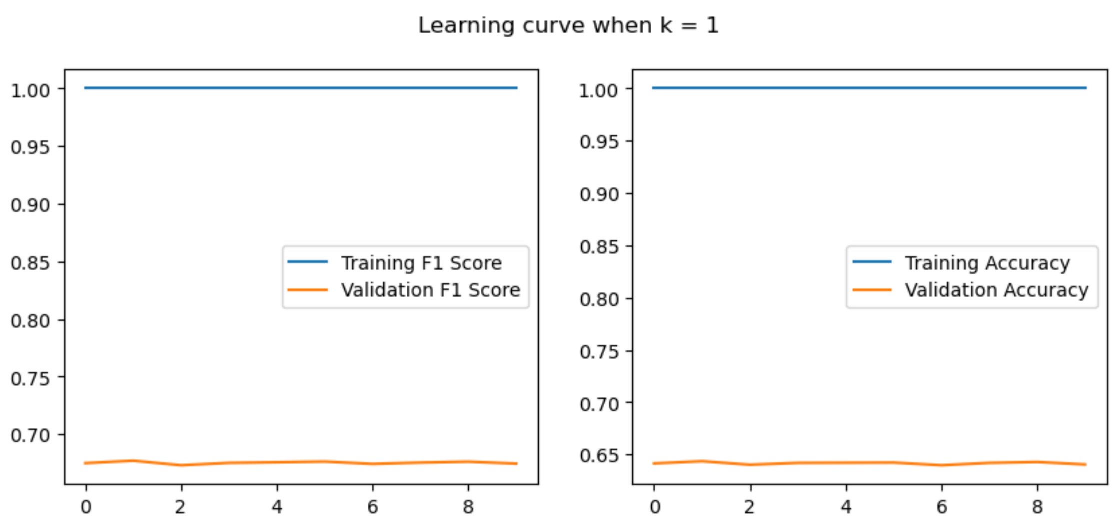
```python
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

def KNN_experiment_LDA(xdata_train_select, ydata_train_select, k, components):

    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)

    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)

    f1_score_train_history = []
    f1_score_val_history = []
    acc_train_history = []
    acc_val_history = []

    lda = LinearDiscriminantAnalysis(n_components=components)
    xdata_train = lda.fit_transform(xdata_train, ydata_train)

    nums_epoch = 10
    for epoch in range(nums_epoch):
        f1_scores_train = []
```

```
        f1_scores_val = []
        acc_train_epoch = []
        acc_val_epoch = []

        # use cross-validation with 4 folds
        cv = KFold(n_splits=4,shuffle=True)
        for i, (train_index, val_index) in enumerate(cv.split(xdata_train)): #␣
↪i in range of 4

            D_train_xdata = xdata_train[train_index]
            D_train_ydata = ydata_train[train_index]
            D_val_xdata = xdata_train[val_index]
            D_val_ydata = ydata_train[val_index]

            # create a KNN classifier
            knn = KNeighborsClassifier(n_neighbors=k)
            knn.fit(D_train_xdata, D_train_ydata)

            # predict validation labels using KNN
            ydata_train_pred = knn.predict(D_train_xdata)
            ydata_val_pred = knn.predict(D_val_xdata)

            # calculate validation f1_score and record for each fold
            f1_train = f1_score(D_train_ydata, ydata_train_pred)
            f1_val = f1_score(D_val_ydata, ydata_val_pred)
            f1_scores_train.append(f1_train)
            f1_scores_val.append(f1_val)

            # calculate accuracy and record for each fold
            acc_train = np.sum(D_train_ydata==ydata_train_pred) /␣
↪len(D_train_ydata)
            acc_val = np.sum(D_val_ydata==ydata_val_pred) / len(D_val_ydata)
            acc_train_epoch.append(acc_train)
            acc_val_epoch.append(acc_val)


        f1_score_train_history.append(np.mean(f1_scores_train))
        f1_score_val_history.append(np.mean(f1_scores_val))
        acc_train_history.append(np.mean(acc_train_epoch))
        acc_val_history.append(np.mean(acc_val_epoch))

    print(f"The mean of training F1 Score = {np.mean(f1_score_val_history)},␣
↪std = {np.std(f1_score_val_history)}")
    print(f"The mean of training accuracy = {np.mean(acc_train_history)}, std =␣
↪{np.std(acc_train_history)}")
    print(f"The mean of validation F1 Score = {np.mean(f1_score_val_history)},␣
↪std = {np.std(f1_score_val_history)}")
```

```
    print(f"The mean of validation accuracy = {np.mean(acc_val_history)}, std =␣
↪{np.std(acc_val_history)}")

    # Plot the learning curve
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
    fig.suptitle(f"Learning curve when k = {k}")
    ax1.plot(np.arange(len(f1_score_train_history)), f1_score_train_history,␣
↪label = "Training F1 Score")
    ax1.plot(np.arange(len(f1_score_val_history)), f1_score_val_history, label␣
↪= "Validation F1 Score")
    ax1.legend()

    ax2.plot(np.arange(len(acc_train_history)), acc_train_history, label =␣
↪"Training Accuracy")
    ax2.plot(np.arange(len(acc_val_history)), acc_val_history, label =␣
↪"Validation Accuracy")
    ax2.legend()
```

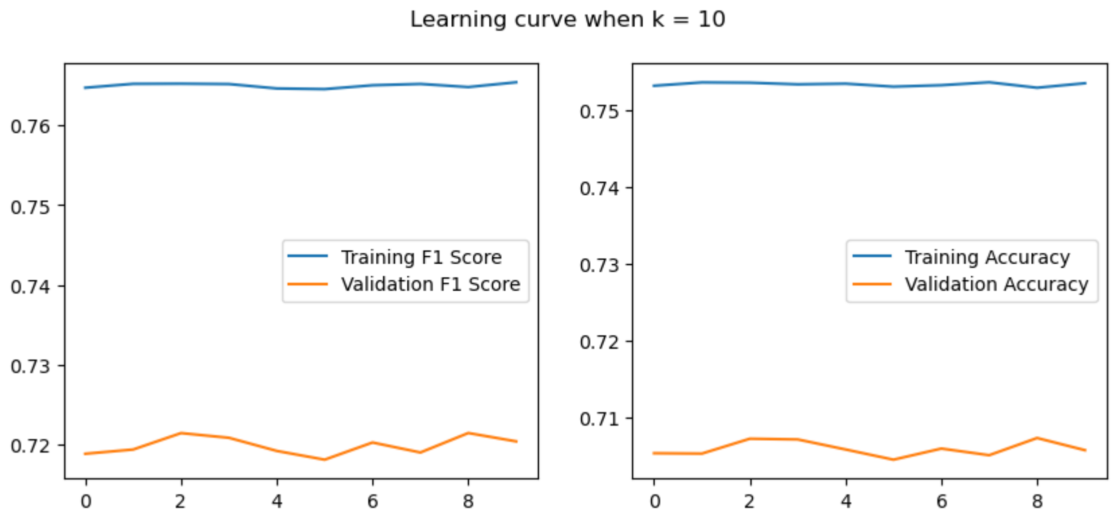
[30]:
```
KNN_experiment_LDA(xdata_train_select, ydata_train_select, 1, 1)
```

```
The mean of training F1 Score = 0.6751173786252431, std = 0.0010832110253885562
The mean of training accuracy = 1.0, std = 0.0
The mean of validation F1 Score = 0.6751173786252431, std =
0.0010832110253885562
The mean of validation accuracy = 0.6415902498362496, std =
0.0011460161157862577
```



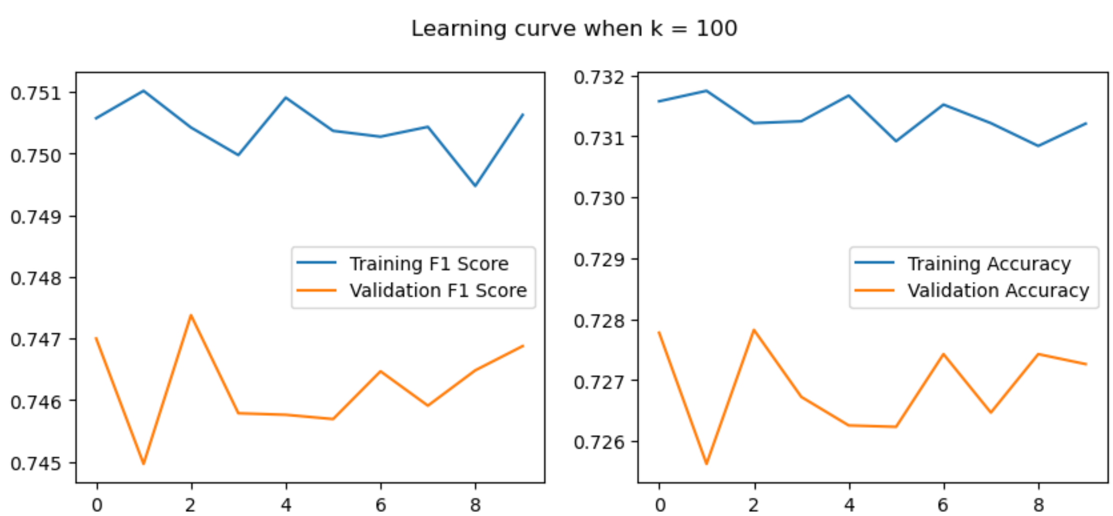
[31]:
```
KNN_experiment_LDA(xdata_train_select, ydata_train_select, 10, 1)
```

```
The mean of training F1 Score = 0.7199241383437412, std = 0.0010905799838560892
The mean of training accuracy = 0.7534130251707681, std = 0.00023273167028294925
The mean of validation F1 Score = 0.7199241383437412, std =
0.0010905799838560892
The mean of validation accuracy = 0.7058762983063536, std =
0.0009323651040886391
```



[33]:
```
KNN_experiment_LDA(xdata_train_select, ydata_train_select, 100, 1)
```

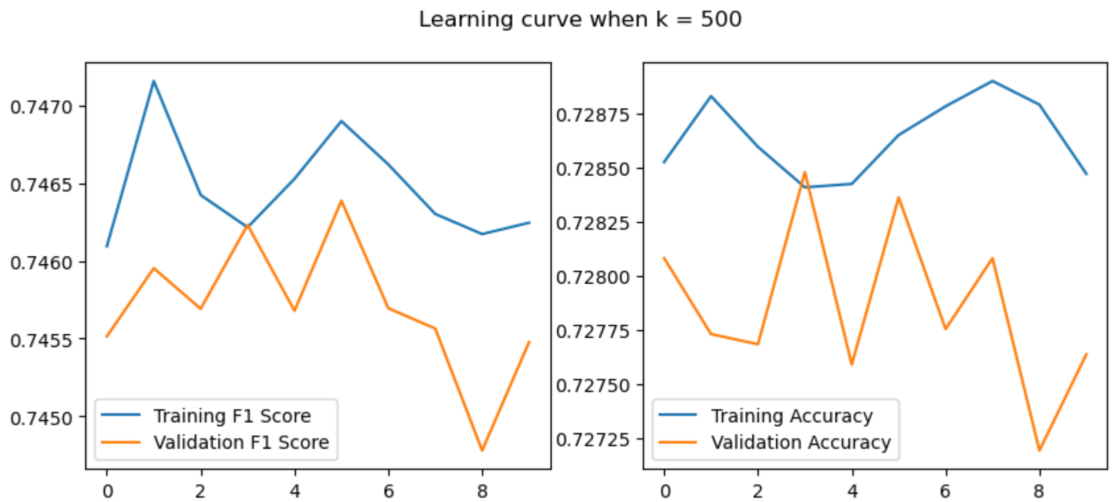
```
The mean of training F1 Score = 0.7462341856364791, std = 0.0006969860923836956
The mean of training accuracy = 0.7313153051994636, std = 0.00028916089495242017
The mean of validation F1 Score = 0.7462341856364791, std =
0.0006969860923836956
The mean of validation accuracy = 0.726901843361093, std = 0.0007078587056464987
```



[34]:

```
KNN_experiment_LDA(xdata_train_select, ydata_train_select, 500, 1)
```

```
The mean of training F1 Score = 0.7456977772213532, std = 0.00042078165332564793
The mean of training accuracy = 0.7286375970805652, std = 0.00017041714661095235
The mean of validation F1 Score = 0.7456977772213532, std =
0.00042078165332564793
The mean of validation accuracy = 0.727858613268457, std =
0.00036887432777640524
```



[35]:

```
KNN_experiment_LDA(xdata_train_select, ydata_train_select, 1000, 1)
```

```
The mean of training F1 Score = 0.744675413096523, std = 0.00018921100767441116
The mean of training accuracy = 0.727513178004429, std = 8.702664563366052e-05
The mean of validation F1 Score = 0.744675413096523, std =
0.00018921100767441116
The mean of validation accuracy = 0.7272667727145129, std =
0.00018472992441658496
```

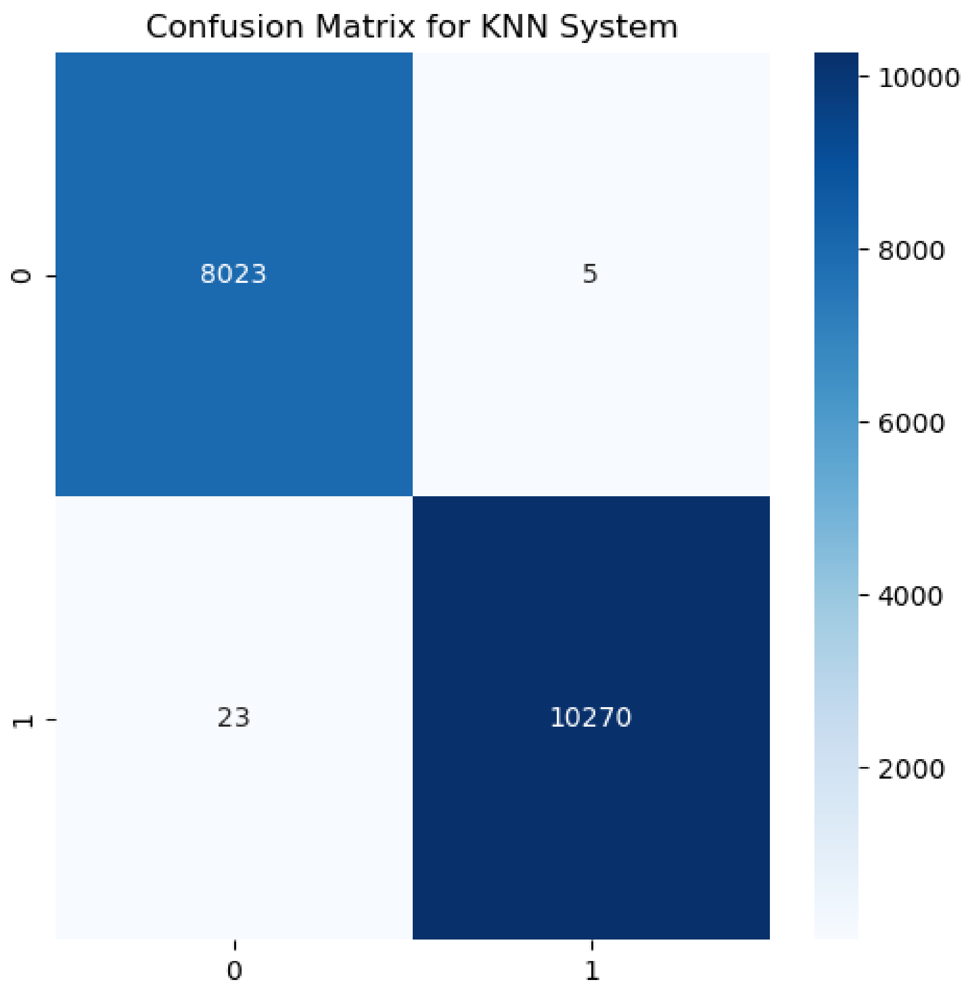
0.1 Test

[41]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

def test_KNN_experiment_PCA(xdata_train_select, ydata_train_select,␣
↪xdata_test_select, ydata_test_select, k, components):

    xdata_train = np.copy(xdata_train_select)
    ydata_train = np.copy(ydata_train_select)
    xdata_test = np.copy(xdata_test_select)
    ydata_test = np.copy(ydata_test_select)

    # Create scaler object and fit to data
    scaler = StandardScaler()
    scaler.fit(xdata_train)

    # Apply scaler to data
    xdata_train = scaler.transform(xdata_train)
    xdata_test = scaler.transform(xdata_test)

    f1_score_train_history = []
    f1_score_val_history = []
    acc_train_history = []
    acc_val_history = []

    pca = PCA(n_components = components)
    xdata_train = pca.fit_transform(xdata_train)
    xdata_test = pca.transform(xdata_test)
```

```
    # Create a KNN model
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xdata_train, ydata_train)

    # Predict test labels using KNN
    ydata_train_pred = knn.predict(xdata_train)
    ydata_test_pred = knn.predict(xdata_test)

    # Calculate F1 Score
    f1_train = f1_score(ydata_train, ydata_train_pred)
    f1_test = f1_score(ydata_test, ydata_test_pred)

    # Calculate accurcay
    acc_train = np.sum(ydata_train==ydata_train_pred) / len(ydata_train)
    acc_test = np.sum(ydata_test==ydata_test_pred) / len(ydata_test)

    print(f"The training F1 Score = {f1_train}")
    print(f"The testing F1 Score = {f1_test}")
    print(f"The training accuracy = {acc_train}")
    print(f"The testing accuracy = {acc_test}")

    # Plot the confusion matrix
    conf_matrix = confusion_matrix(ydata_test, ydata_test_pred)
    plt.figure(figsize = (6, 6))
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = 'Blues')
    plt.title("Confusion Matrix for KNN System")
```

[42]:
```
test_KNN_experiment_PCA(xdata_train_select, ydata_train_select,␣
↪xdata_test_select, ydata_test_select,  10, 30)
```

```
The training F1 Score = 0.9990248865900708
The testing F1 Score = 0.9986386619992221
The training accuracy = 0.9989239262655563
The testing accuracy = 0.9984716991430599
```


Confusion Matrix for KNN System
8023
5
23
10270
0
1
0
1
10000
8000
6000
4000
2000