In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import utils
```

```
2023-12-22 23:05:02.712488: I tensorflow/core/platform/cpu_feature_guard.cc:
182] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 FMA, in other o
perations, rebuild TensorFlow with the appropriate compiler flags.
```

Decision Trees and Random Forest

Chronic kidney disease

In [2]:
```python
from sklearn.model_selection import train_test_split, cross_val_score

ckd_data = pd.read_csv("Data/kidney_disease_cleaned.csv").set_index("id")
X_labels = ckd_data.drop("classification", axis="columns").columns

X, Y = utils.clean_normalize_dataset("Data/kidney_disease.csv", "ckd")

# Train/Test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, ra
```

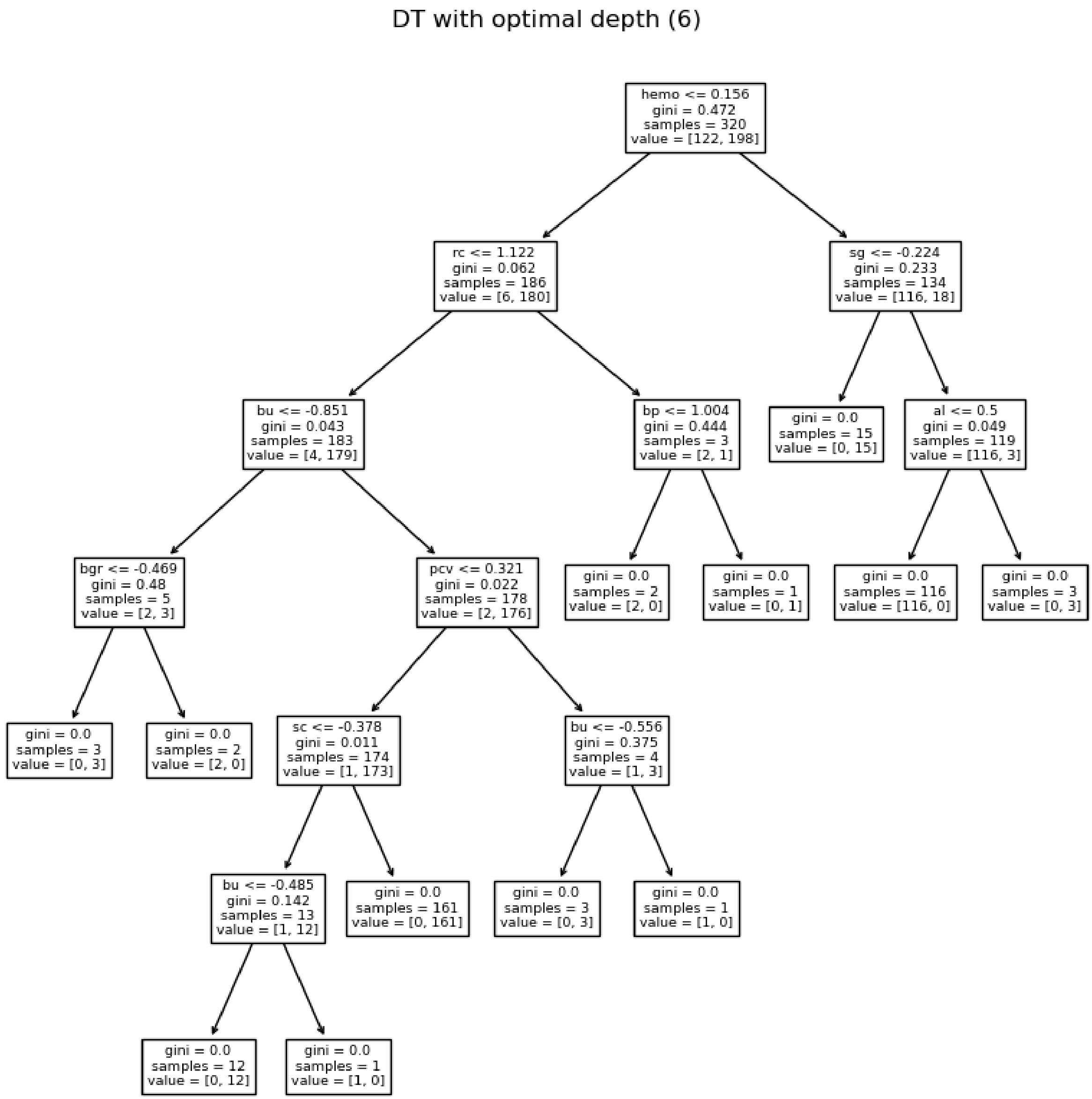
Single decision tree

In [3]:
```python
from sklearn import tree

max_depth = 8
mean_scores = pd.DataFrame(index=pd.Index(range(2, max_depth+1), name="depth
                           columns=["score"], dtype=np.float64)

# Crossvalidation on multiple depths to find optimal one
for depth in mean_scores.index:
    clf = tree.DecisionTreeClassifier(max_depth=depth)
    scores = cross_val_score(clf, X_train, Y_train, cv = 5) # Use training d
    mean_scores.loc[depth] = scores.mean()

opt_depth = mean_scores.idxmax()[0]
```

In [4]:
```python
# Train and test
clf = tree.DecisionTreeClassifier(max_depth=opt_depth)
clf.fit(X_train, Y_train)
acc = clf.score(X_test, Y_test)
print(f"Accuracy: {acc}")

fig, ax = plt.subplots(figsize=(10,10))
```

```
tree.plot_tree(clf, feature_names=list(X_labels), ax=ax);
ax.set_title(f"DT with optimal depth ({opt_depth})");
```

```
Accuracy: 0.9875
```



Usually 100% accuracy is a sign that something's wrong, but in training we didn't get 100% so it's just that we got lucky with the split

Random forest classifier

We will assume for the sake of simplicity that the optimal `max_depth` is the same as for single trees.

In [5]:
```
from sklearn.ensemble import RandomForestClassifier

n_trees_list = [5, 10, 20, 40, 60, 80, 100, 120, 150]
```

```
mean_scores = pd.DataFrame(index=pd.Index(n_trees_list, name="n_trees"),\
                           columns=["score"], dtype=np.float64)

# Crossvalidation on multiple depths to find optimal one
for n_trees in mean_scores.index:
    clf = RandomForestClassifier(n_estimators=n_trees, max_depth=opt_depth,
    scores = cross_val_score(clf, X_train, Y_train, cv = 5) # Use training c
    mean_scores.loc[n_trees] = scores.mean()

opt_n_trees= mean_scores.idxmax()[0]
print(f"Optimal n_trees: {opt_n_trees}")

clf = RandomForestClassifier(n_estimators=opt_n_trees, max_depth=opt_depth,
clf.fit(X_train, Y_train)
acc = clf.score(X_test, Y_test)
print(f"Accuracy: {acc}")
```

```
Optimal n_trees: 60
Accuracy: 1.0
```

Banknote authentication

We apply the same workflow to the other dataset.

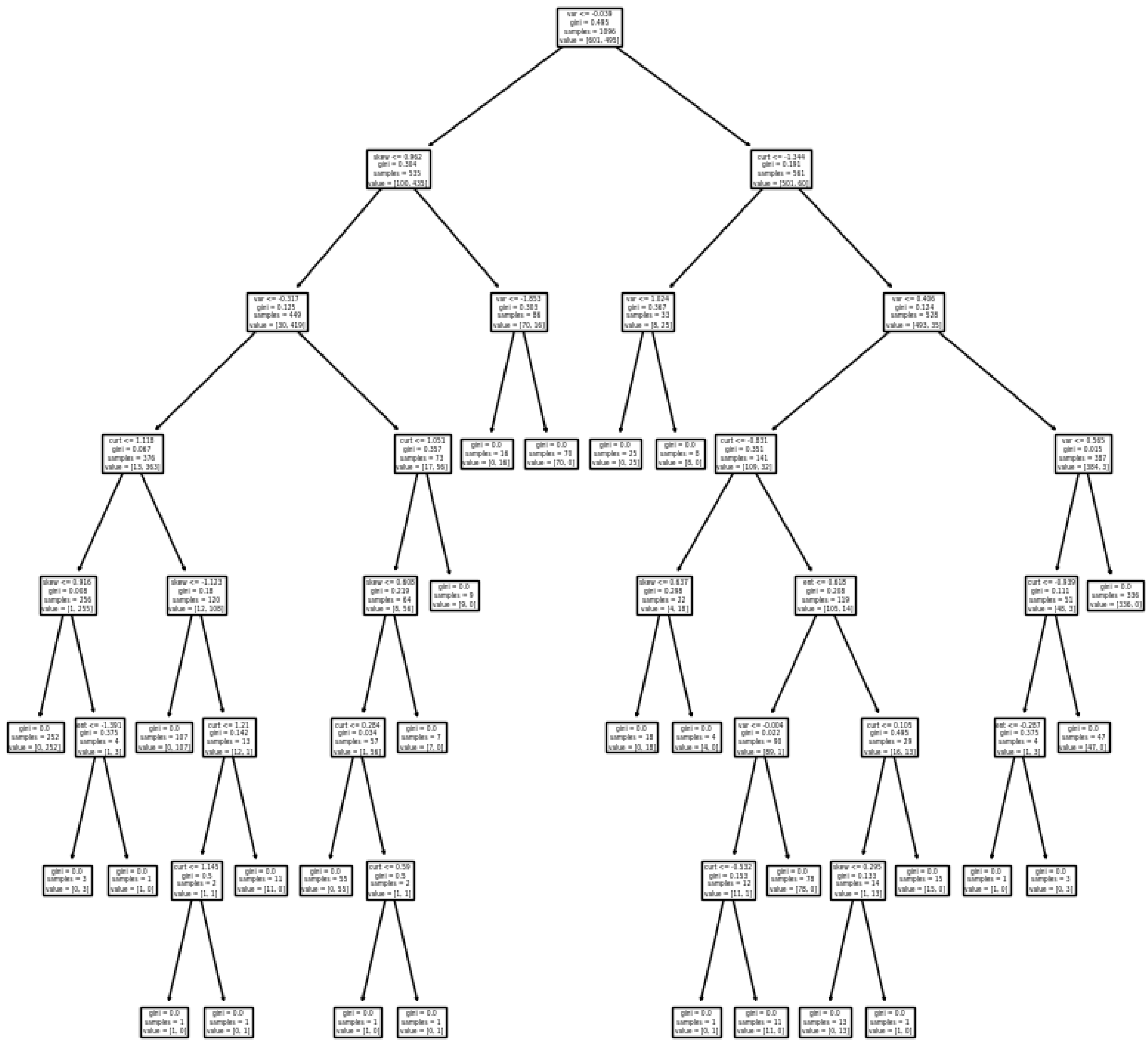
In [6]:
```
X, Y = utils.clean_normalize_dataset("Data/data_banknote_authentication.txt"
X_labels = ["var", "skew", "curt", "ent"]

# Train/Test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, ra
```

Single decision tree

In [7]:
```
max_depth = 8
mean_scores = pd.DataFrame(index=pd.Index(range(2, max_depth+1), name="depth
                           columns=["score"], dtype=np.float64)

# Crossvalidation on multiple depths to find optimal one
for depth in mean_scores.index:
    clf = tree.DecisionTreeClassifier(max_depth=depth)
    scores = cross_val_score(clf, X_train, Y_train, cv = 5) # Use training c
    mean_scores.loc[depth] = scores.mean()

opt_depth = mean_scores.idxmax()[0]

# Train and test
clf = tree.DecisionTreeClassifier(max_depth=opt_depth)
clf.fit(X_train, Y_train)
acc = clf.score(X_test, Y_test)
print(f"Accuracy: {acc}")

fig, ax = plt.subplots(figsize=(10,10))
tree.plot_tree(clf, feature_names=list(X_labels), ax=ax);
ax.set_title(f"DT with optimal depth ({opt_depth})");
```

```
Accuracy: 0.9927272727272727
```

DT with optimal depth (8)



Random forest classifier

In [8]:
```python
n_trees_list = [5, 10, 20, 40, 60, 80, 100, 120, 150]

mean_scores = pd.DataFrame(index=pd.Index(n_trees_list, name="n_trees"),\
                           columns=["score"], dtype=np.float64)

# Crossvalidation on multiple depths to find optimal one
for n_trees in mean_scores.index:
    clf = RandomForestClassifier(n_estimators=n_trees, max_depth=opt_depth,
    scores = cross_val_score(clf, X_train, Y_train, cv = 5) # Use training d
    mean_scores.loc[n_trees] = scores.mean()

opt_n_trees= mean_scores.idxmax()[0]
print(f"Optimal n_trees: {opt_n_trees}")
```

```
clf = RandomForestClassifier(n_estimators=opt_n_trees, max_depth=opt_depth,
clf.fit(X_train, Y_train)
acc = clf.score(X_test, Y_test)
print(f"Accuracy: {acc}")
```

```
Optimal n_trees: 60
Accuracy: 0.9890909090909091
```

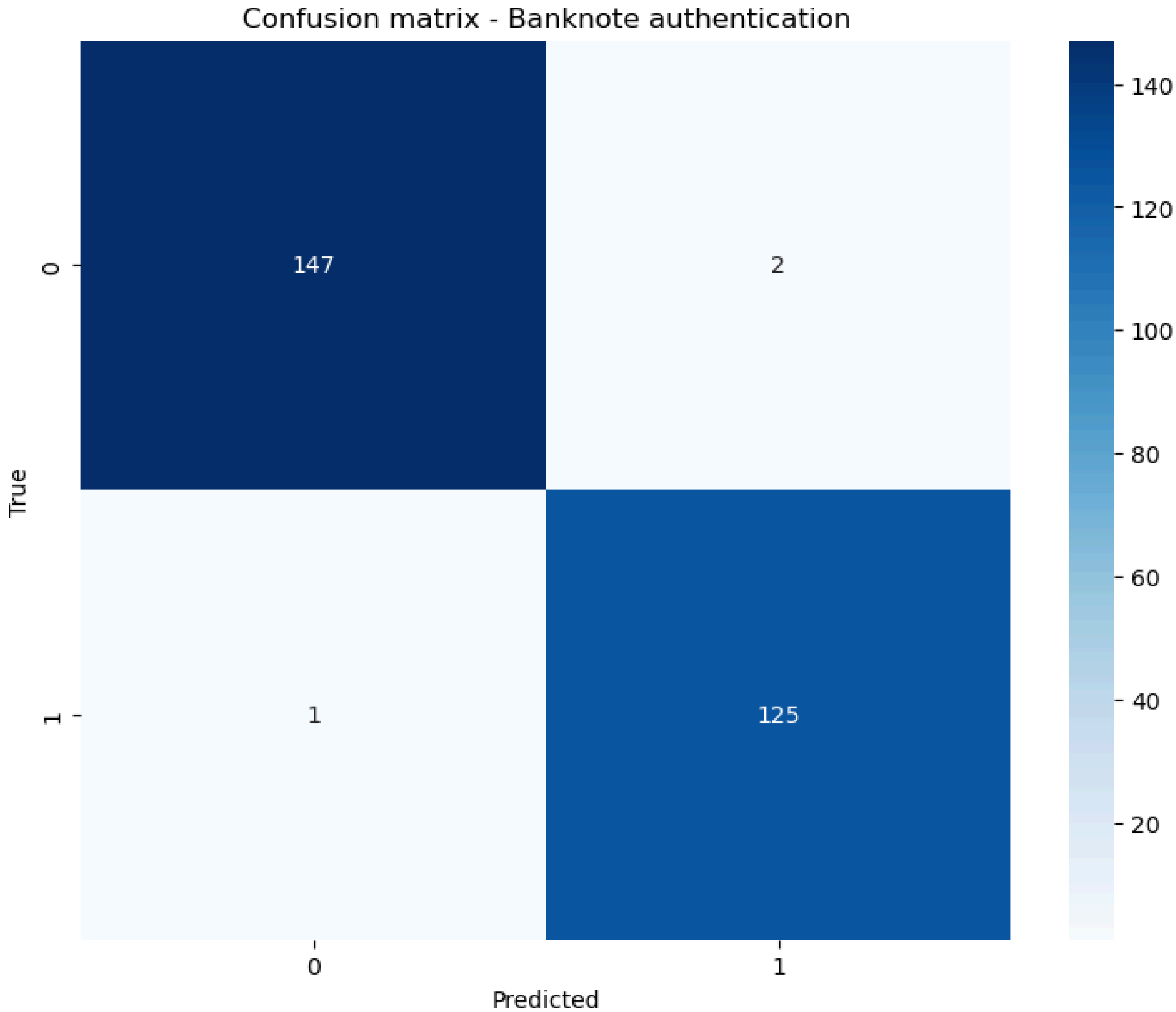
Neural Networks

In [9]:
```
# Load, process and split the banknote authentication data
ba, y_ba = utils.clean_normalize_dataset("Data/data_banknote_authentication.
X_train_ba, X_test_ba, y_train_ba, y_test_ba = train_test_split(ba, y_ba, te
```

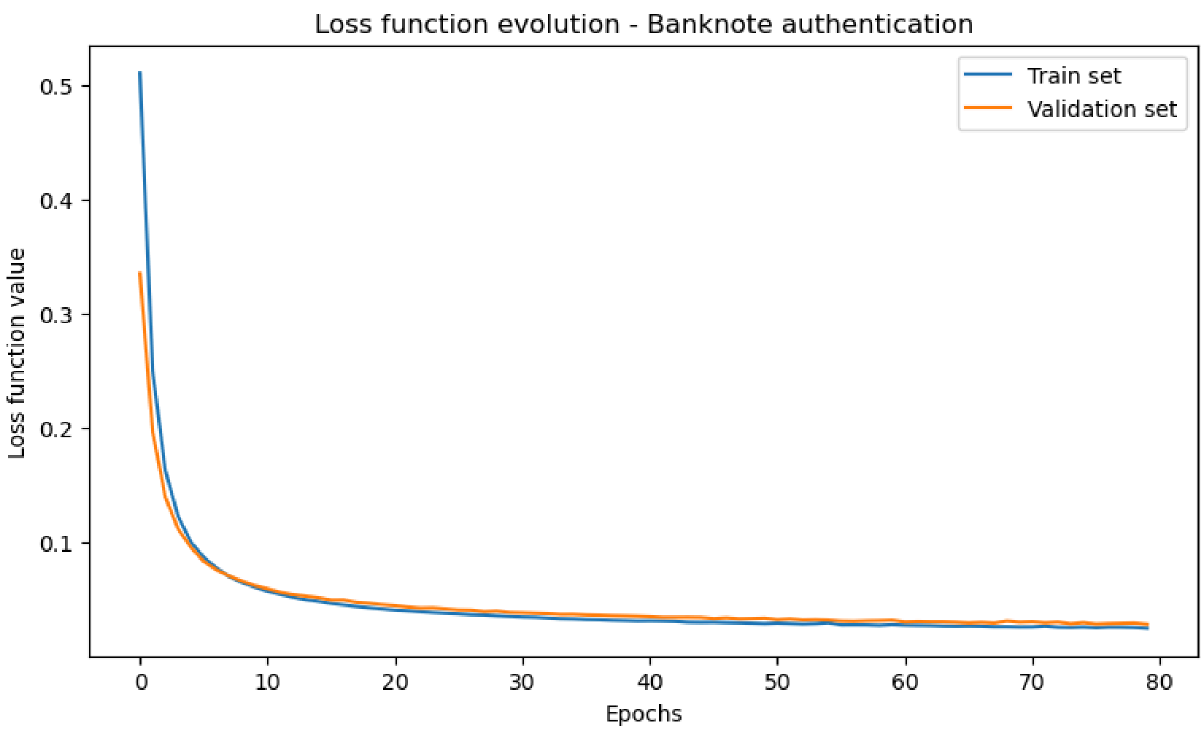
In [10]:
```
# Load, process and split the chronic kidney disease data
ckd, y_ckd = utils.clean_normalize_dataset("Data/kidney_disease.csv", "ckd")
X_train_ckd, X_test_ckd, y_train_ckd, y_test_ckd = train_test_split(ckd, y_c
```

In [11]:
```
# Banknote authentication
# Train the model
model_ba, history_ba = utils.fit_NN(X_train_ba, X_test_ba, y_train_ba, y_tes

#Make predictions
y_pred, cm_ba = utils.predict_NN(model_ba, X_test_ba, y_test_ba)
utils.plot_cm(cm_ba, 'Banknote authentication')

# Plot the loss function evolution
utils.plot_loss(history_ba, 'Banknote authentication')
```

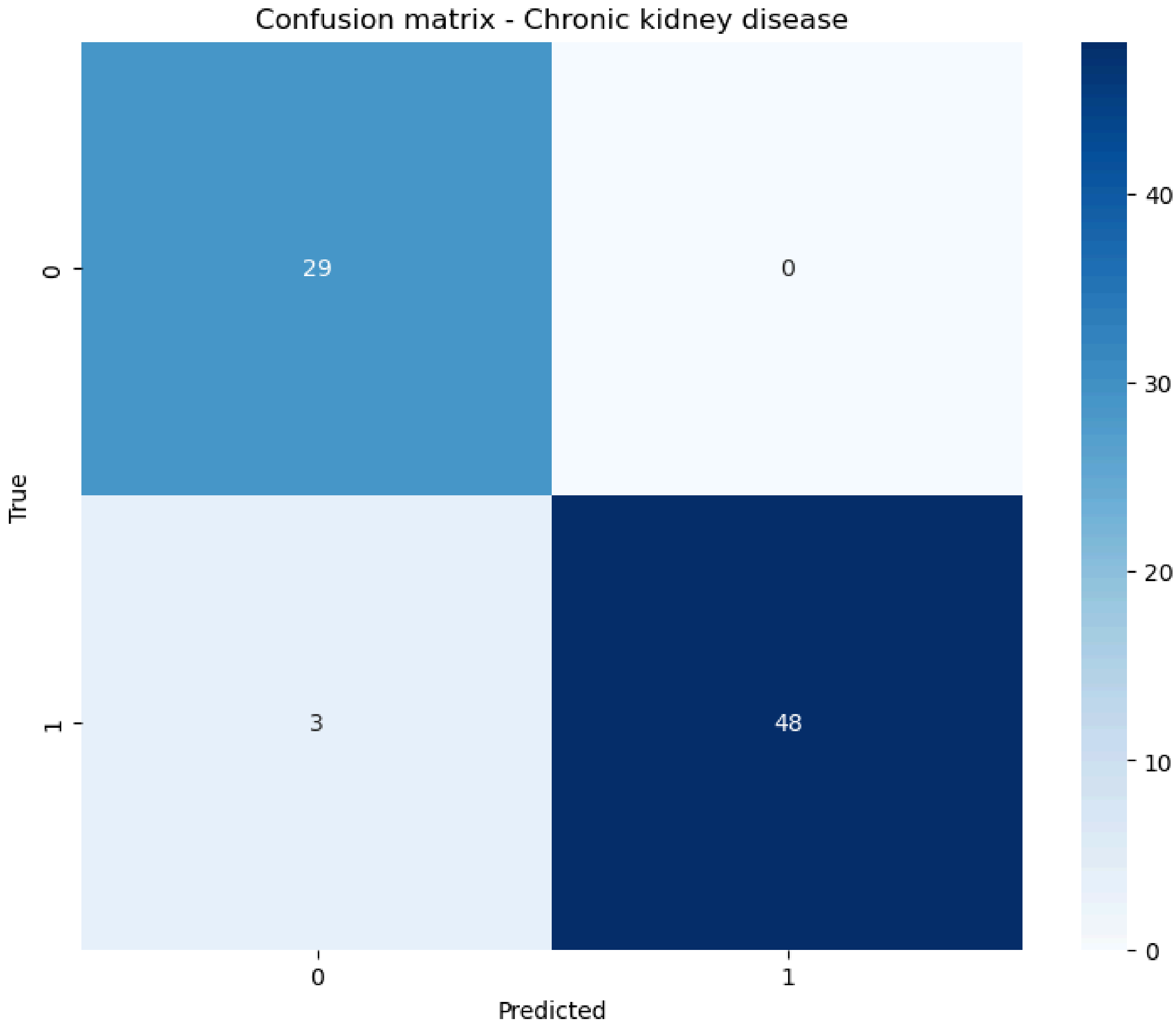
```
2023-12-22 23:05:19.168963: I tensorflow/core/common_runtime/process_util.c
c:146] Creating new thread pool with default inter op setting: 2. Tune using
inter_op_parallelism_threads for best performance.
```



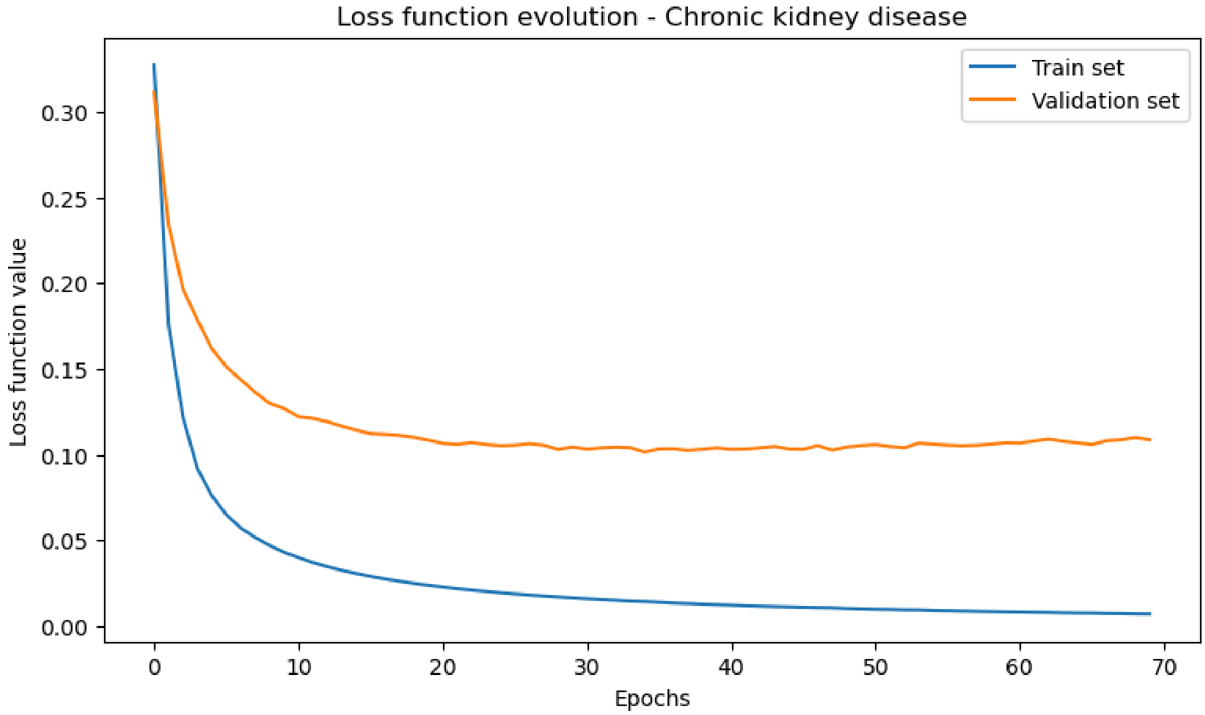
Training loss: 0.024390550330281258 , Validation loss: 0.027814155444450283



In [12]:
```
# Chronic kidney disease
# Train the model
model_ckd, history_ckd = utils.fit_NN(X_train_ckd, X_test_ckd, y_train_ckd,

#Make predictions
y_pred, cm_ckd = utils.predict_NN(model_ckd, X_test_ckd, y_test_ckd)
utils.plot_cm(cm_ckd, 'Chronic kidney disease')

# Plot the loss function evolution
utils.plot_loss(history_ckd, 'Chronic kidney disease')
```




```
Training loss: 0.007106180302798748 , Validation loss: 0.10872243344783783
```



The models perform well. However, there are still improvements and verifications that could be made:

- use cross-validation
- check how categorical variables are handled
- check the metrics
- use different validation set instead of test set

Support Vector Machine

In [13]:
```
random_state = 42
test_size    = 0.20
```

Banknote authentication

In [14]:
```
X, y = utils.clean_normalize_dataset("Data/data_banknote_authentication.txt"
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_siz
```

In [15]:
```
y_pred = utils.fit_pred_SVM(X_train, y_train, X_test, y_test, "ba")
```

```
----------------------------------------
Accuracy for SVM on 'ba' dataset:
0.9927272727272727
----------------------------------------
```

Chronic Kidney Disease

In [16]:
```python
X, y = utils.clean_normalize_dataset("Data/kidney_disease.csv", "ckd")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_siz
```

In [17]:
```python
y_pred = utils.fit_pred_SVM(X_train, y_train, X_test, y_test, "ckd")
```

```
----------------------------------------
Accuracy for SVM on 'ckd' dataset:
0.9875
----------------------------------------
```

K-Nearest Neighbours

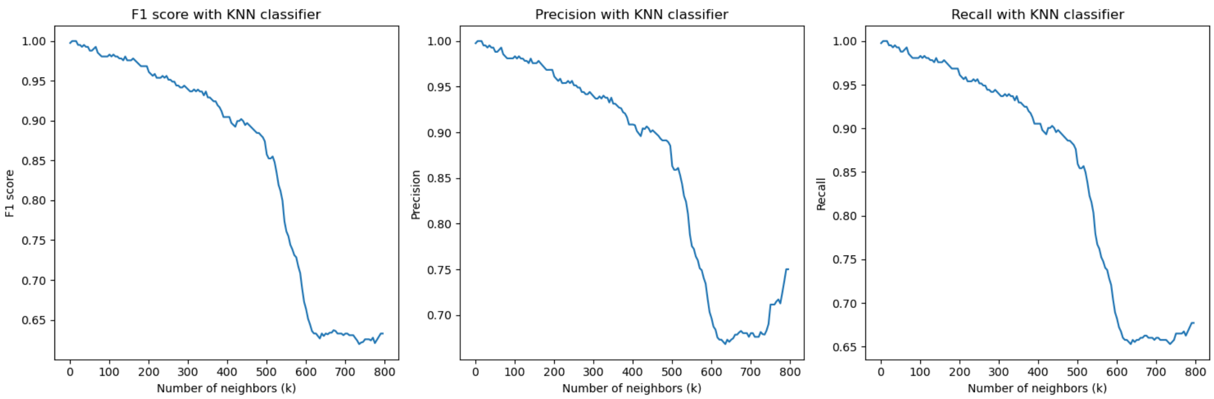
In [18]:
```python
from sklearn.metrics import accuracy_score, classification_report
```

First approach

In [19]:
```python
ba = pd.read_csv("Data/data_banknote_authentication.txt")
ba, y_ba = utils.clean_normalize_ba(ba, normalize = False)
X = np.array(ba.to_numpy())
y = np.array(y_ba.to_numpy())
```

In [21]:
```python
# Evaluate KNN for different values of k
k_values, f1_score_list, precision_list, recall_list, conf_matrix_list = uti

# Create a figure with 1 row and 3 columns
plt.figure(figsize=(15, 5))  # Adjust the figsize as needed

# Plot for the first subplot
plt.subplot(1, 3, 1)
plt.plot(k_values, f1_score_list)
plt.title('F1 score with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('F1 score')

# Plot for the second subplot
plt.subplot(1, 3, 2)
plt.plot(k_values, precision_list)
plt.title('Precision with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('Precision')

# Plot for the third subplot
plt.subplot(1, 3, 3)
plt.plot(k_values, recall_list)
plt.title('Recall with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('Recall')

# Adjust layout to prevent overlapping
plt.tight_layout()
```

```
# Show the plots
plt.show()

print("Confusion matrix for k = 1")
print(conf_matrix_list[0])
```




```
Confusion matrix for k = 1
[[228   1]
 [  0 183]]
```

F1 score, precision and recall show very similar behaviors, so F1 score will be considered from now on as it contains both the recall and precision through the next formula:

$$\text{F1 score} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

Taking $k = 1$ seems to be the best choice given the F1 score (even for $k = 1$ no overfitting seems to be taking place).

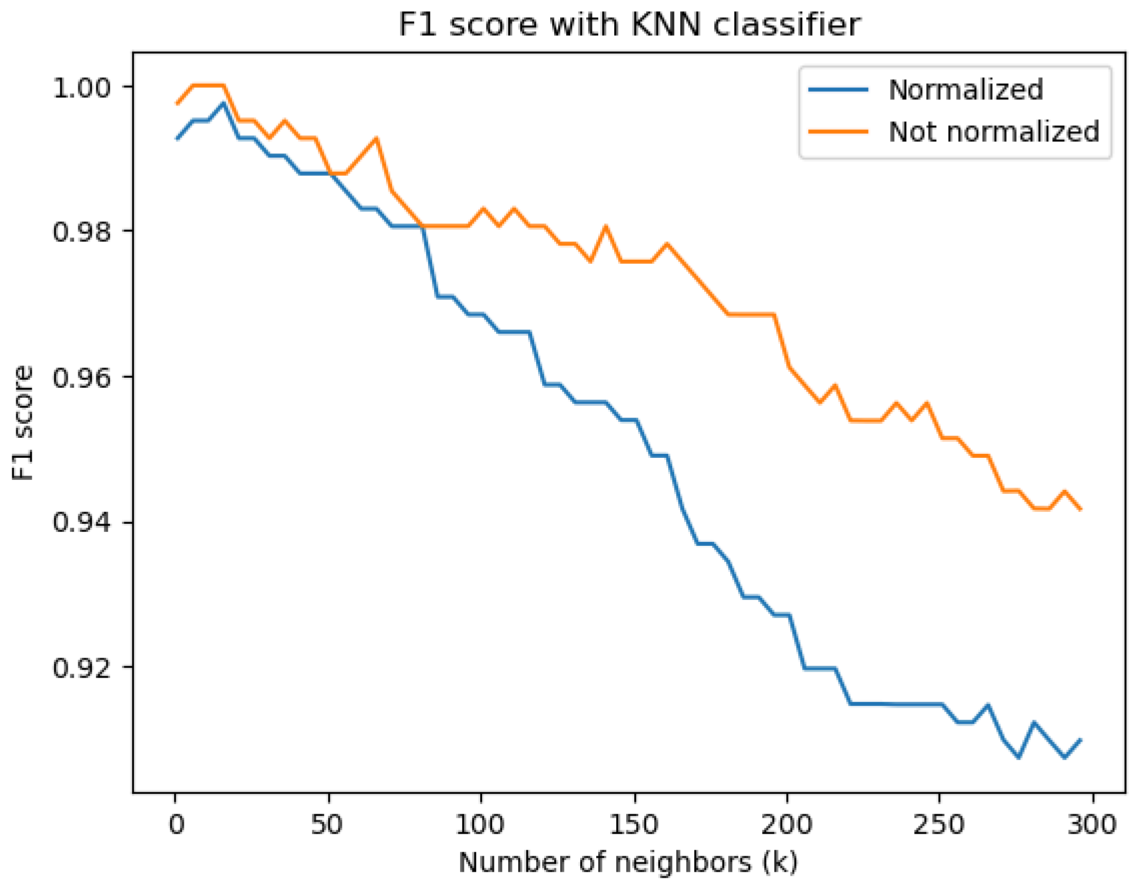
Data normalization

In [22]:
```
X_normalized = utils.normalize(X)
```

```
Mean of the unscaled features:
[-1.24383849e-16  4.14612829e-17 -5.18266036e-17 -1.16609858e-17]
Standard deviation of the unscaled features:
[1. 1. 1. 1.]
```

In [23]:
```
k_values, f1_score_list_normalized, _, _, conf_matrix_list_normalized = util
k_values, f1_score_list, precision_list, recall_list, conf_matrix_list = uti

plt.plot(k_values, f1_score_list_normalized, label='Normalized')
plt.plot(k_values, f1_score_list, label='Not normalized')
plt.title('F1 score with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('F1 score')
plt.legend(loc = 'upper right')
```

Out[23]: <matplotlib.legend.Legend at 0x7fb33331e710>



Low values of k are more attractive for KNN implementation, since the offer a better perfomance but also because of the small size of the dataset. For low values of k it is observed that feature normalization does not actually improve the performance of the model. For values of k higher than 100, the unnormalized case outperforms the use of normalized features. This last phenomenon is quite logical, since unnormalized data will give more weight to some variables in the implementation of KNN algorithm, which in this case seems to improve the predicition (probably due to the fact that the features with the higher values were, by mere luck, more relevant).

Feature selection

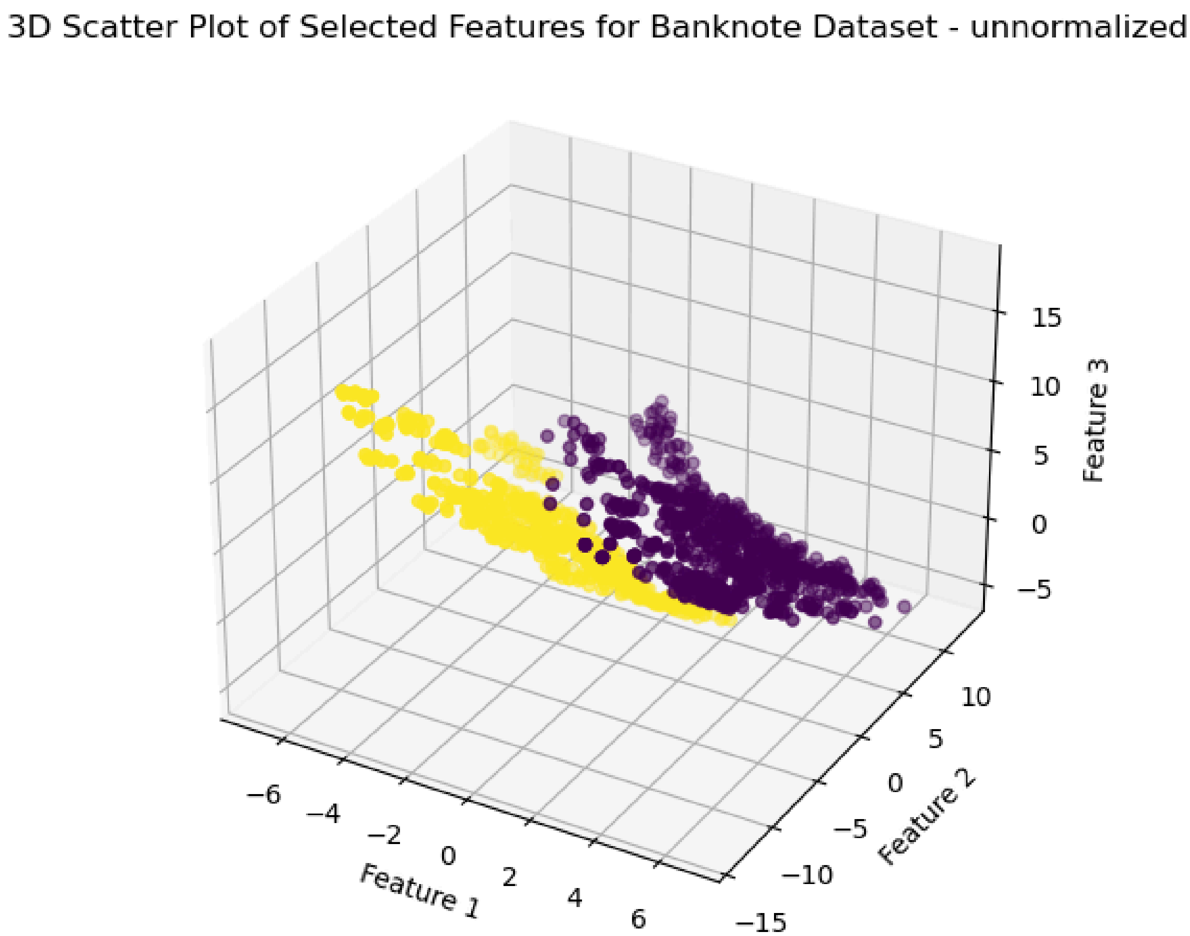
In [24]:
```
from sklearn.feature_selection import SelectKBest, f_classif

def select_k_best_features(X, y, k):

    # Initialize SelectKBest with the f_classif statistical test
    selector = SelectKBest(score_func = f_classif, k = k)
    X_selected = selector.fit_transform(X, y)

    # Get the indices of the selected features
    selected_indices = selector.get_support(indices=True)
```

```
    # Get feature scores
    feature_scores = selector.scores_
    print("Indices of selected features:", selected_indices)
    print("Feature scores for normalized data:", feature_scores)

    # Now X_selected contains the selected features

    return X_selected, selected_indices
```

In [25]:
```
X_selected, indices = select_k_best_features(X, y, 3)
X_selected_normalized, indices_normalized = select_k_best_features(X_normali
```

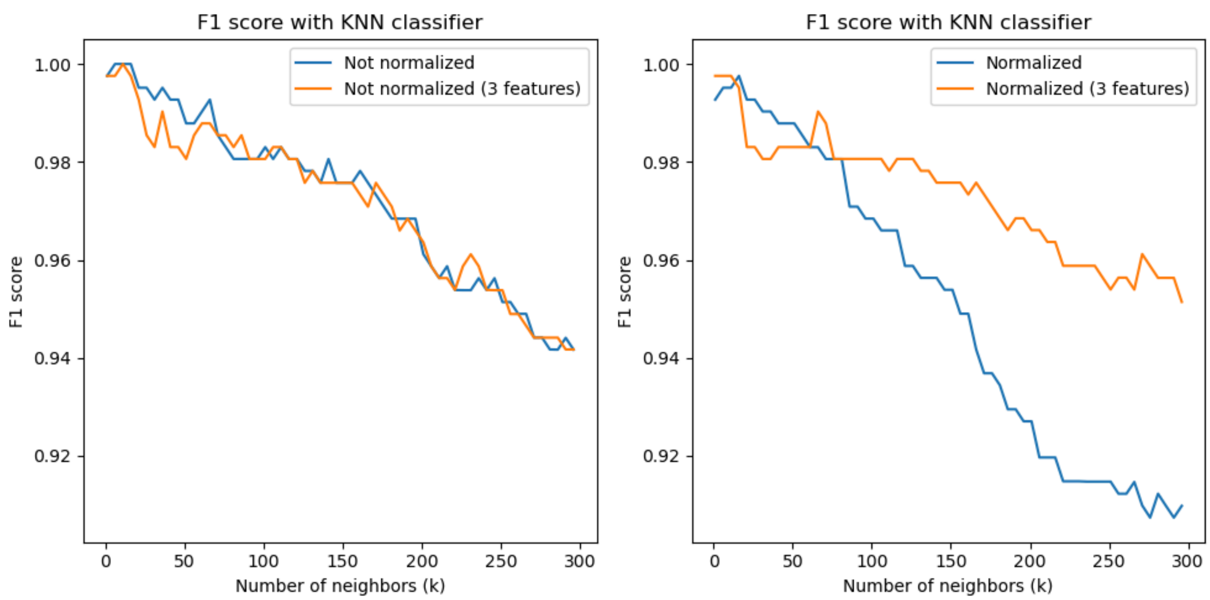
```
Indices of selected features: [0 1 2]
Feature scores for normalized data: [1.51386490e+03 3.36676646e+02 3.3854045
7e+01 7.37255535e-01]
Indices of selected features: [0 1 2]
Feature scores for normalized data: [1.51386490e+03 3.36676646e+02 3.3854045
7e+01 7.37255535e-01]
```

In [26]:
```
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D plot
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
ax.scatter(X_selected[:, 0], X_selected[:, 1], X_selected[:, 2], c=y, cmap='

# Set labels
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('3D Scatter Plot of Selected Features for Banknote Dataset - ur

# Show the plot
plt.show()
```



In [27]:
```
k_values, f1_score_list_3_features, _, _, conf_matrix_list_normalized_3_feat
k_values, f1_score_list_normalized_3_features, _,_, conf_matrix_list_3_featu

# Set the same y-axis limits for both subplots
y_min = min(min(f1_score_list), min(f1_score_list_3_features), min(f1_score_
y_max = max(max(f1_score_list), max(f1_score_list_3_features), max(f1_score_

# Create a figure with 1 row and 3 columns
plt.figure(figsize=(10, 5))  # Adjust the figsize

plt.subplot(1, 2, 1)

plt.plot(k_values, f1_score_list, label='Not normalized')
plt.plot(k_values, f1_score_list_3_features, label='Not normalized (3 featur

plt.title('F1 score with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('F1 score')
plt.legend(loc = 'upper right')
plt.ylim(y_min, y_max)

plt.subplot(1, 2, 2)

plt.plot(k_values, f1_score_list_normalized, label='Normalized')
plt.plot(k_values, f1_score_list_normalized_3_features, label='Normalized (3
```

```
plt.title('F1 score with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('F1 score')
plt.legend(loc = 'upper right')
plt.ylim(y_min, y_max)

plt.tight_layout()

plt.show()
```



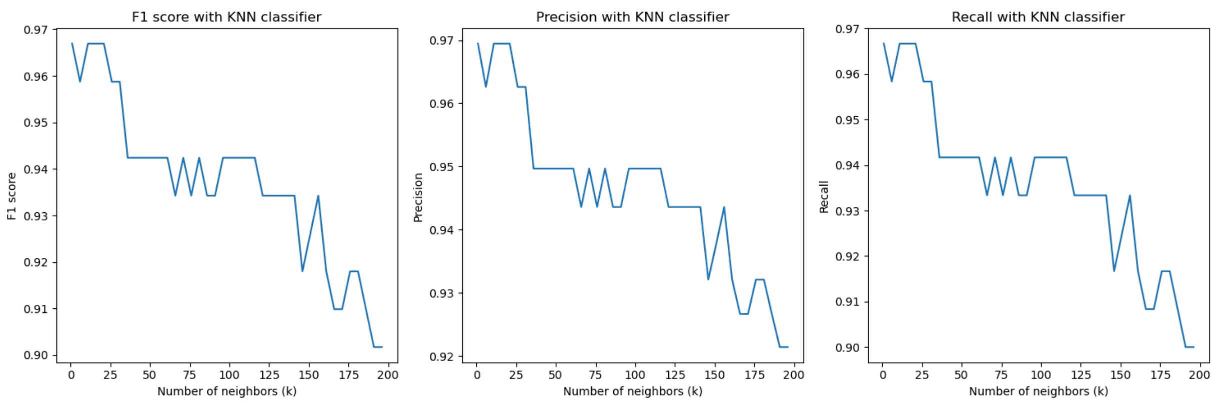
For unnormalized data, erasing one feature (by choosing the 3 more relevant features) does not seem to improve the model for most values of k. For normalized data, it is clear that for $k < 70$ the model works better when considering the 4 features, but for $k > 100$ using 3 features works the best.

KNN for Kidney Disease dataset

In [28]:
```
ckd = pd.read_csv("Data/kidney_disease.csv")
ckd, y_ckd = utils.clean_normalize_ckd(ckd)
X = np.array(ckd.to_numpy())
y = np.array(y_ckd.to_numpy())
```

In [29]:
```
# Evaluate KNN for different values of k
k_values, f1_score_list, precision_list, recall_list, conf_matrix_list = uti

# Create a figure with 1 row and 3 columns
plt.figure(figsize=(15, 5))  # Adjust the figsize as needed

# Plot for the first subplot
plt.subplot(1, 3, 1)
plt.plot(k_values, f1_score_list)
plt.title('F1 score with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('F1 score')
```

```
# Plot for the second subplot
plt.subplot(1, 3, 2)
plt.plot(k_values, precision_list)
plt.title('Precision with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('Precision')

# Plot for the third subplot
plt.subplot(1, 3, 3)
plt.plot(k_values, recall_list)
plt.title('Recall with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('Recall')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Show the plots
plt.show()

print("Confusion matrix for k = 1")
print(conf_matrix_list[0])
```




```
Confusion matrix for k = 1
[[44  0]
 [ 4 72]]
```

In [30]:
```
X_normalized = utils.normalize(X)
X_selected, indices = select_k_best_features(X, y, 3)
X_selected_normalized, indices_normalized = select_k_best_features(X_normali
```

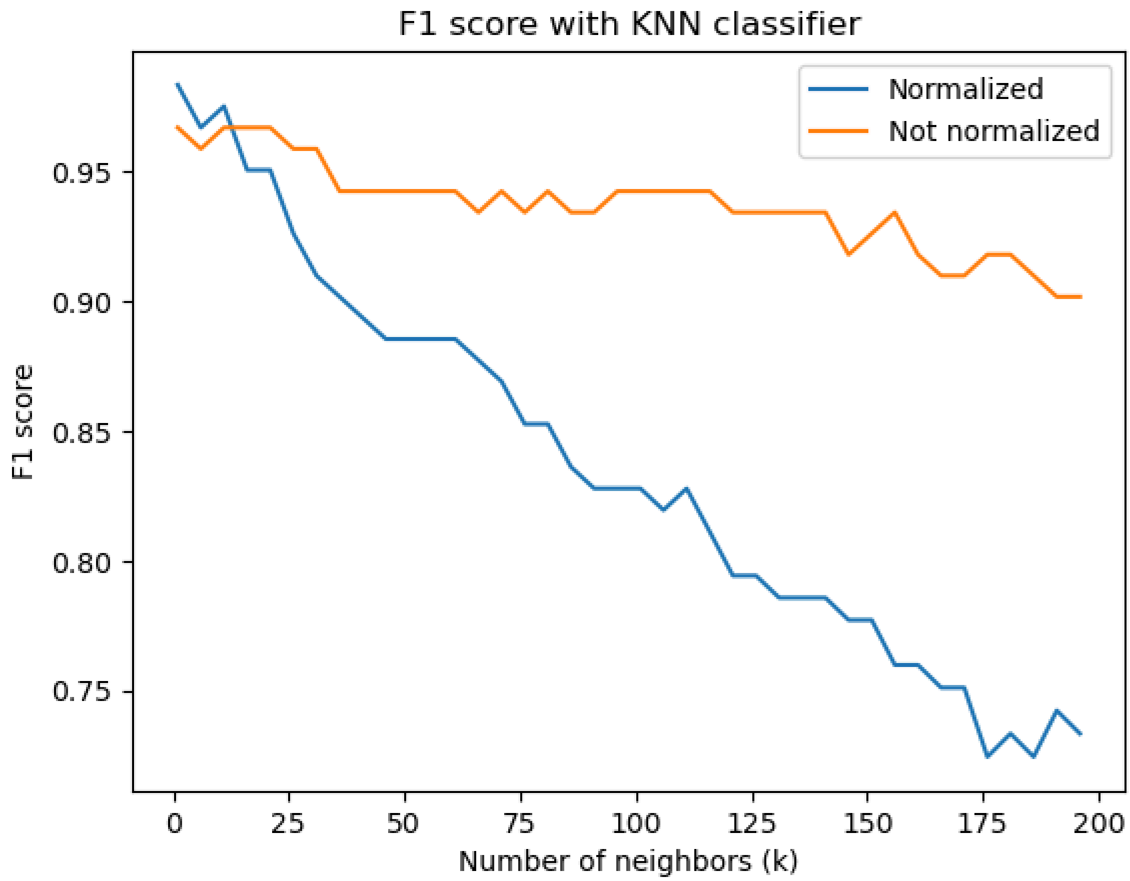
```
Mean of the unscaled features:
[ 8.88178420e-18  0.00000000e+00 -7.10542736e-17  3.55271368e-17
  1.77635684e-17  2.13162821e-16 -1.42108547e-16  1.77635684e-17
 -5.32907052e-17  1.77635684e-17  1.77635684e-17  0.00000000e+00
  1.77635684e-17  0.00000000e+00  0.00000000e+00  3.55271368e-17
 -1.77635684e-17 -7.10542736e-17  3.55271368e-17 -7.10542736e-17
 -9.76996262e-17  0.00000000e+00 -1.77635684e-17  0.00000000e+00]
Standard deviation of the unscaled features:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Indices of selected features: [ 2 14 15]
Feature scores for normalized data: [ 21.30380871  36.71059968 380.16675662
156.74878444  47.91879344
  34.55541872  65.18965517  30.13701923  14.40131579  76.43119693
  63.93581681  37.67863422  52.81829989   2.36894798 453.07789236
 361.80745153  17.50848727 213.53495655 213.00728155 180.94911504
  23.49305556  72.84821429  65.18965517  47.13157895]
Indices of selected features: [ 2 14 15]
Feature scores for normalized data: [ 21.30380871  36.71059968 380.16675662
156.74878444  47.91879344
  34.55541872  65.18965517  30.13701923  14.40131579  76.43119693
  63.93581681  37.67863422  52.81829989   2.36894798 453.07789236
 361.80745153  17.50848727 213.53495655 213.00728155 180.94911504
  23.49305556  72.84821429  65.18965517  47.13157895]
```

In [31]:

```
k_values, f1_score_list_normalized, _, _, conf_matrix_list_normalized = util
k_values, f1_score_list, precision_list, recall_list, conf_matrix_list = uti

plt.plot(k_values, f1_score_list_normalized, label='Normalized')
plt.plot(k_values, f1_score_list, label='Not normalized')
plt.title('F1 score with KNN classifier')
plt.xlabel('Number of neighbors (k)')
plt.ylabel('F1 score')
plt.legend(loc = 'upper right')
```

Out[31]:

```
<matplotlib.legend.Legend at 0x7fb333378a90>
```



In [32]:
```
X_selected, indices = select_k_best_features(X_normalized,y, 3)

# Create a 3D plot
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
ax.scatter(X_selected[:, 0], X_selected[:, 1], X_selected[:, 2], c=y, cmap='

# Set labels
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('3D Scatter Plot of Selected Features for Banknote Dataset - ur

# Show the plot
plt.show()
```

```
Indices of selected features: [ 2 14 15]
Feature scores for normalized data: [ 21.30380871  36.71059968 380.16675662
156.74878444  47.91879344
  34.55541872  65.18965517  30.13701923  14.40131579  76.43119693
  63.93581681  37.67863422  52.81829989   2.36894798 453.07789236
 361.80745153  17.50848727 213.53495655 213.00728155 180.94911504
  23.49305556  72.84821429  65.18965517  47.13157895]
```

3D Scatter Plot of Selected Features for Banknote Dataset - unnormalized