Stroke Prediction

Dataset

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. This dataset is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status. Each row in the data provides relavant information about the patient.

Attribute Information

1. id: unique identifier
2. gender: "Male", "Female" or "Other"
3. age: age of the patient
4. hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5. heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6. ever_married: "No" or "Yes"
7. work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8. Residence_type: "Rural" or "Urban"
9. avg_glucose_level: average glucose level in blood
10. bmi: body mass index
11. smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12. stroke: 1 if the patient had a stroke or 0 if not

*Note: "Unknown" in smoking_status means that the information is unavailable for this patient

Experiment

In this program we will use different classification algorithms to determine which of those has the best accuracy.

The classification algorithms used in this program:

1. Decision Tree (with all nodes and only some of them)
2. Naive Bayes
3. K-nearest neighbour (for k = 3 & k = 5 & k = 11)
4. Neural networks

In the first part of experiment our dataset in only preprocessed partly (only with the most important preprocessing techniques). However in the second part we used optimization techniques like normalizing or principal component analysis algorithm (PCA)

Author of experiment: Oskar Lewna

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import  ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.decomposition import PCA
from keras.models import Sequential
from keras.layers import Dense
```

There we read the file and delete rows with missing values and remove ID column. Below we can see how the amount of data changed

In [ ]:
```
df = pd.read_csv('healthcare-dataset-stroke-data.csv')

# before removing rows with null and id column
print(df.shape)

gender = {"Female": 0, "Male": 1, "Other": 2}
married = {"No": 0, "Yes": 1}
work_type = {"children": 0,"Govt_job": 1, "Never_worked": 2, "Private": 3,"Self-
residence_type = {"Rural": 0, "Urban": 1}
smoking_status = {"formerly smoked": 0, "never smoked": 1, "smokes": 2}

df['Gender'] = df['Gender'].map(gender)
df['Ever_Married'] = df['Ever_Married'].map(married)
df['Work_Type'] = df['Work_Type'].map(work_type)
df['Residence_Type'] = df['Residence_Type'].map(residence_type)
df['Smoking_Status'] = df['Smoking_Status'].map(smoking_status)

df = df.dropna()
df = df.drop(columns=["ID"])
# after
print(df.shape)
```

```
(43400, 12)
(29072, 11)
```

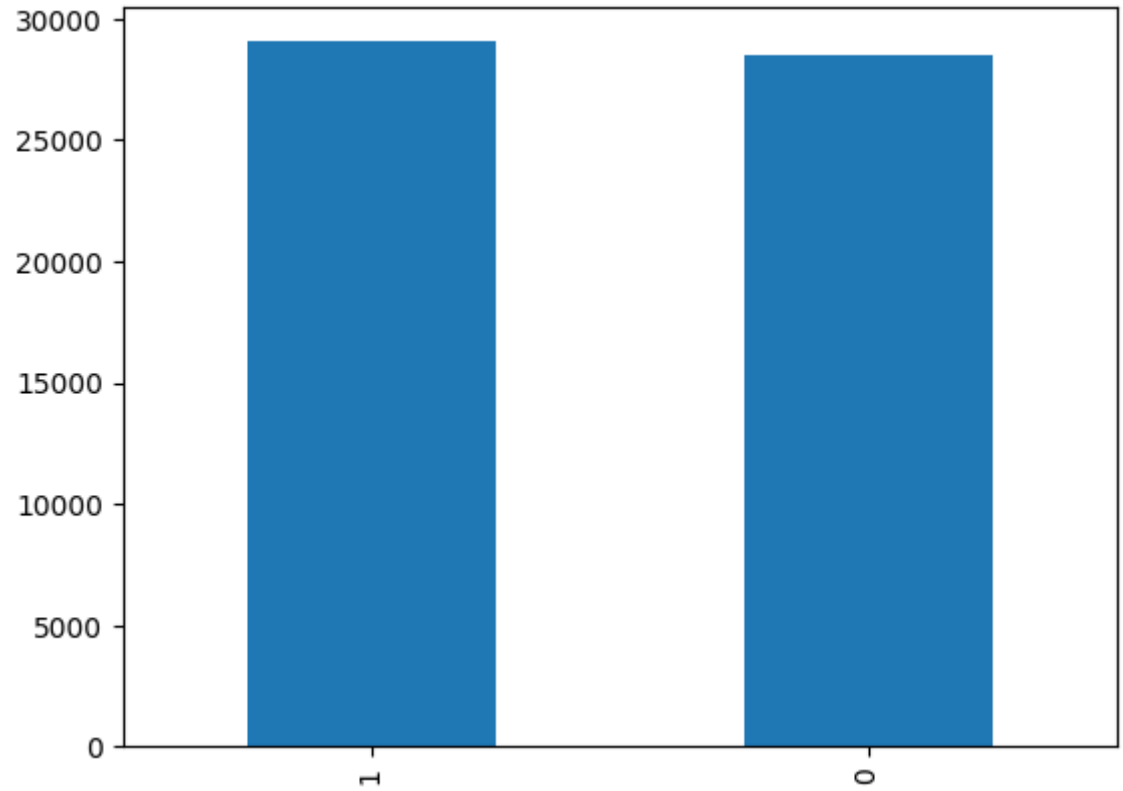
Due to imbalanced data we need to use over-sampling to add samples which are less

In [ ]:
```
true = df["Stroke"].sum()
rows, cols = df.shape
new_true_amount = (rows-true)//true
new_rows = df[df["Stroke"] == 1]
for i in range(new_true_amount):
    df = pd.concat([df,new_rows])

# after adding more TRUE samples
print(df.shape)
```

```
df = df.sample(frac=1)

df['Stroke'].value_counts().plot.bar()
```

```
(57568, 11)
```

Out[ ]: <AxesSubplot:>



Here we split data to train set and test set

In [ ]:
```
inputs = df.values[:, :-1]
classes = df.values[:, -1]

(train_inputs, test_inputs, train_labels, test_labels) = train_test_split(inputs
```

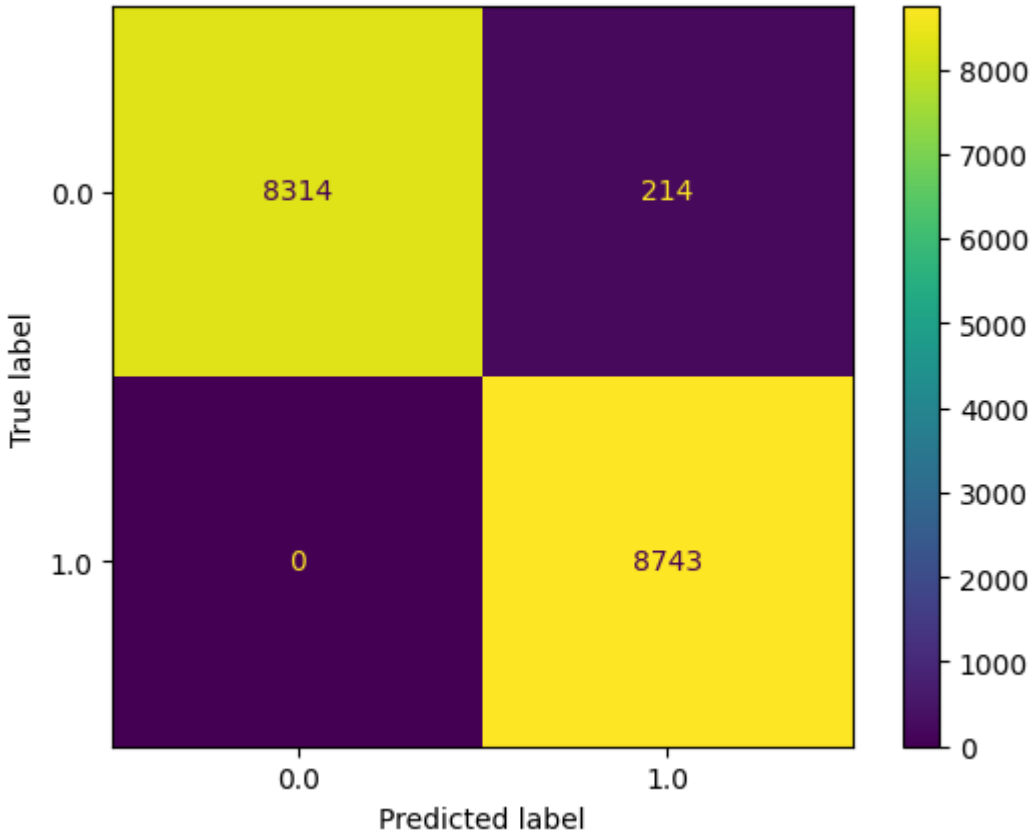
Firstly we will classify dataset with a bit of preprocessing

Decision Tree with all nodes

In [ ]:
```
dtc = DecisionTreeClassifier(random_state=278779)

dtc = dtc.fit(train_inputs, train_labels)

print(dtc.score(test_inputs, test_labels))

predictions = dtc.predict(test_inputs)
cm = confusion_matrix(test_labels, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dtc.classes_)
disp.plot()
```

```
0.9876092872445139
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219d36c7190>

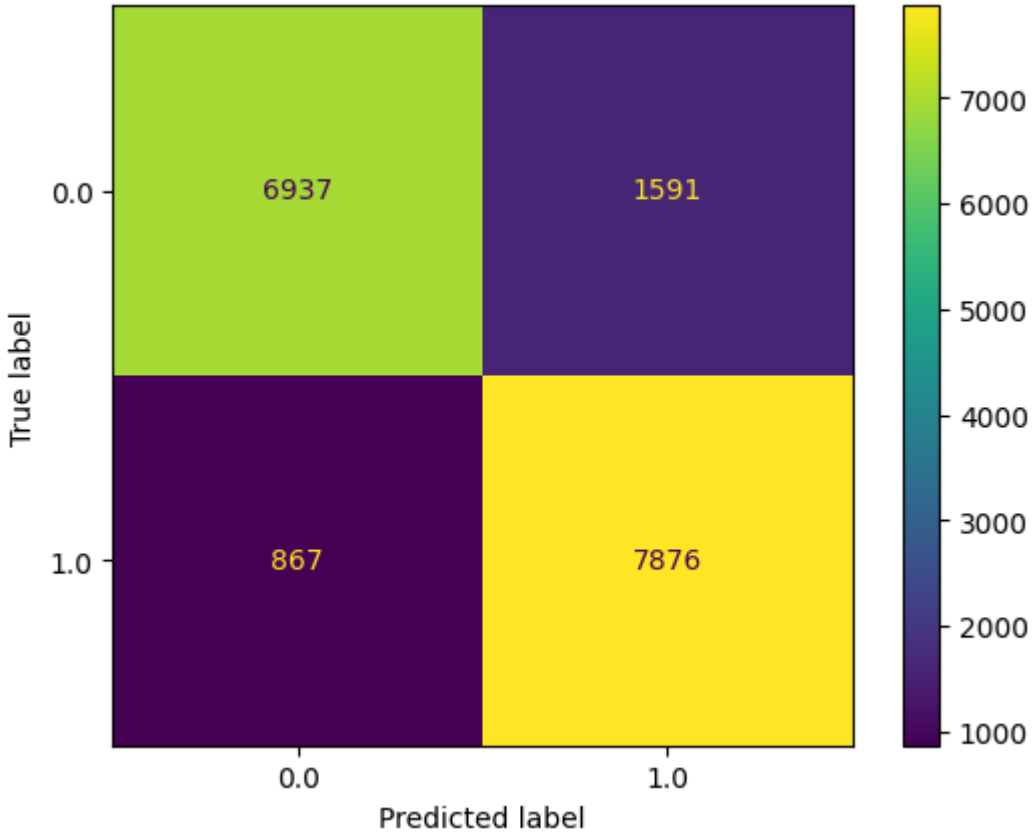


Decision Tree with 10 layers of nodes

In [ ]:
```
dtc = DecisionTreeClassifier(random_state=278779, max_depth=10)

dtc = dtc.fit(train_inputs, train_labels)

print(dtc.score(test_inputs, test_labels))

predictions = dtc.predict(test_inputs)
cm = confusion_matrix(test_labels, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dtc.classes_)
disp.plot()
```

```
0.8576805048925945
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219cfbd48b0>

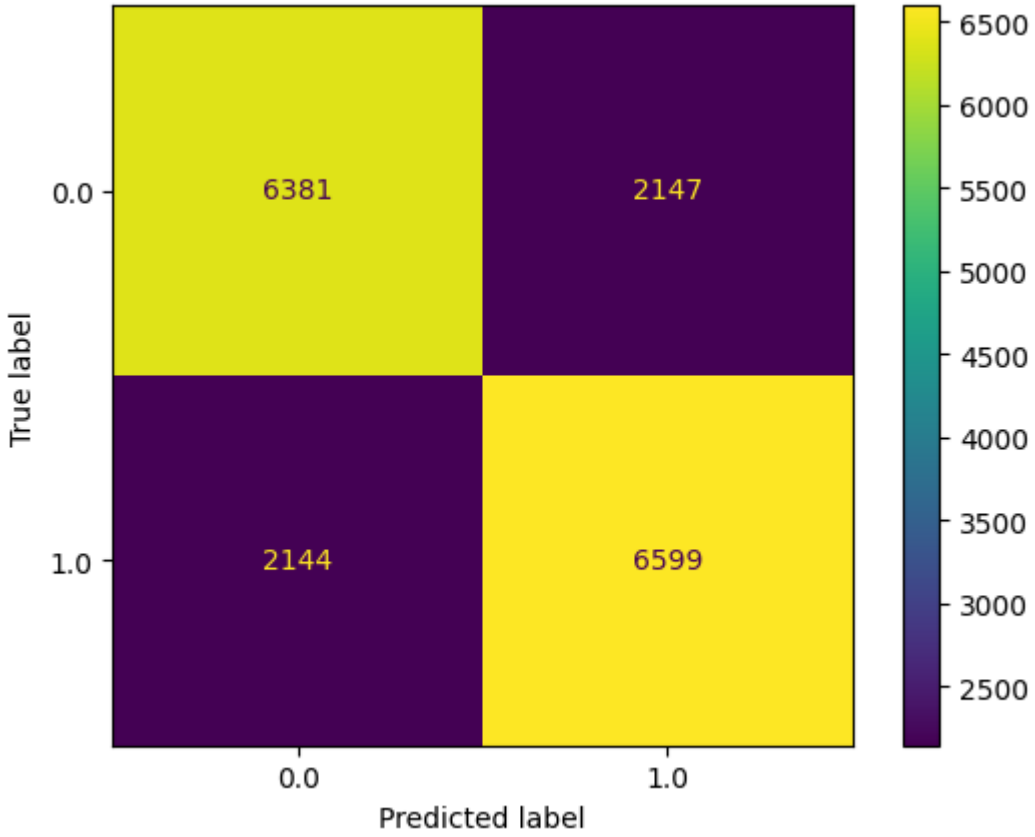


Naive Bayes

In [ ]:
```
bayes = GaussianNB()
bayes.fit(train_inputs, train_labels)
predictions = bayes.predict(test_inputs)

print(bayes.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=bayes.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=bayes.classes_)
disp.plot()
```

```
0.7515488390944357
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219c9f767d0
>
```

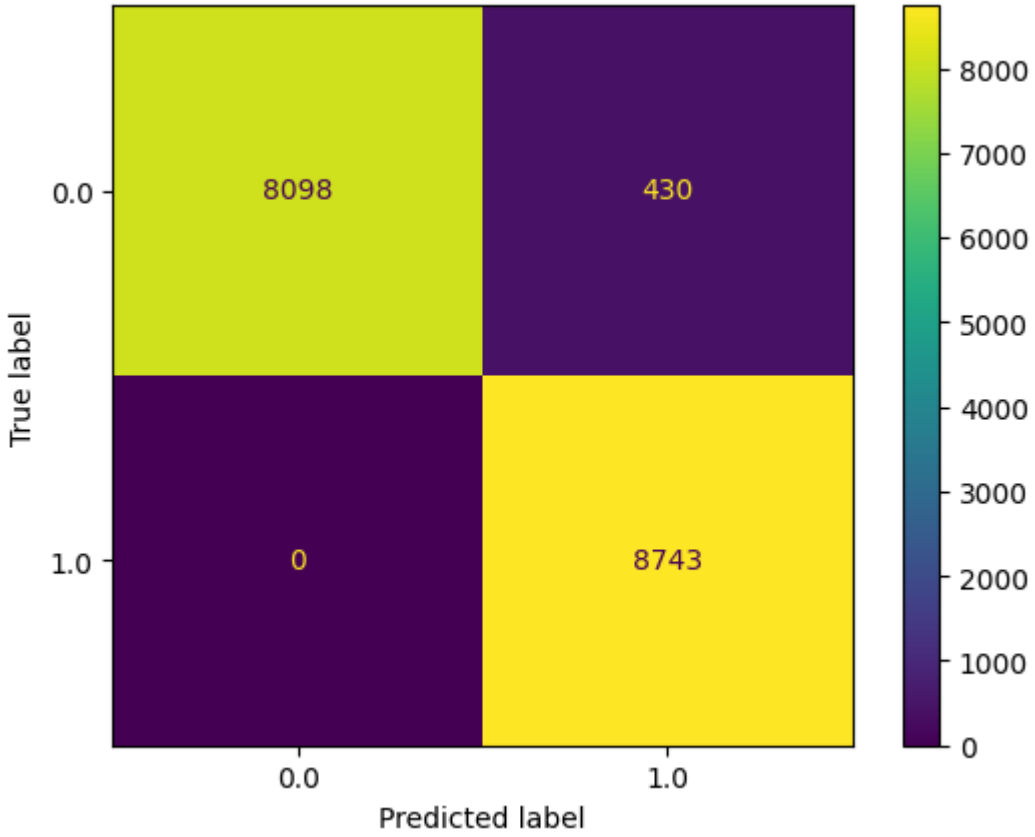


K-nearest neighbors (KNN) where k=3

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn.fit(train_inputs, train_labels)
predictions = knn.predict(test_inputs)

print(knn.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=knn.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=knn.classes_)
disp.plot()
```

```
0.9751027734352382
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219d0fd5bd0
>
```

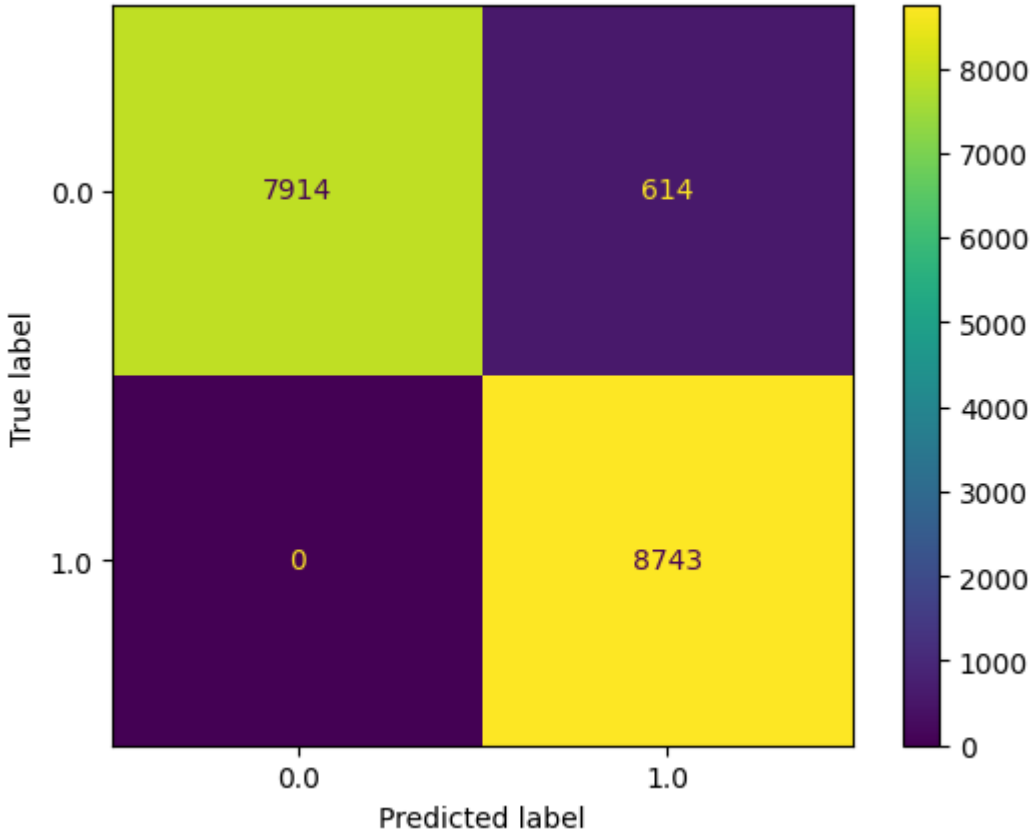


K-nearest neighbors (KNN) where k=5

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(train_inputs, train_labels)
predictions = knn.predict(test_inputs)

print(knn.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=knn.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=knn.classes_)
disp.plot()
```

```
0.9644490764865961
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219c9df2ce0
>
```

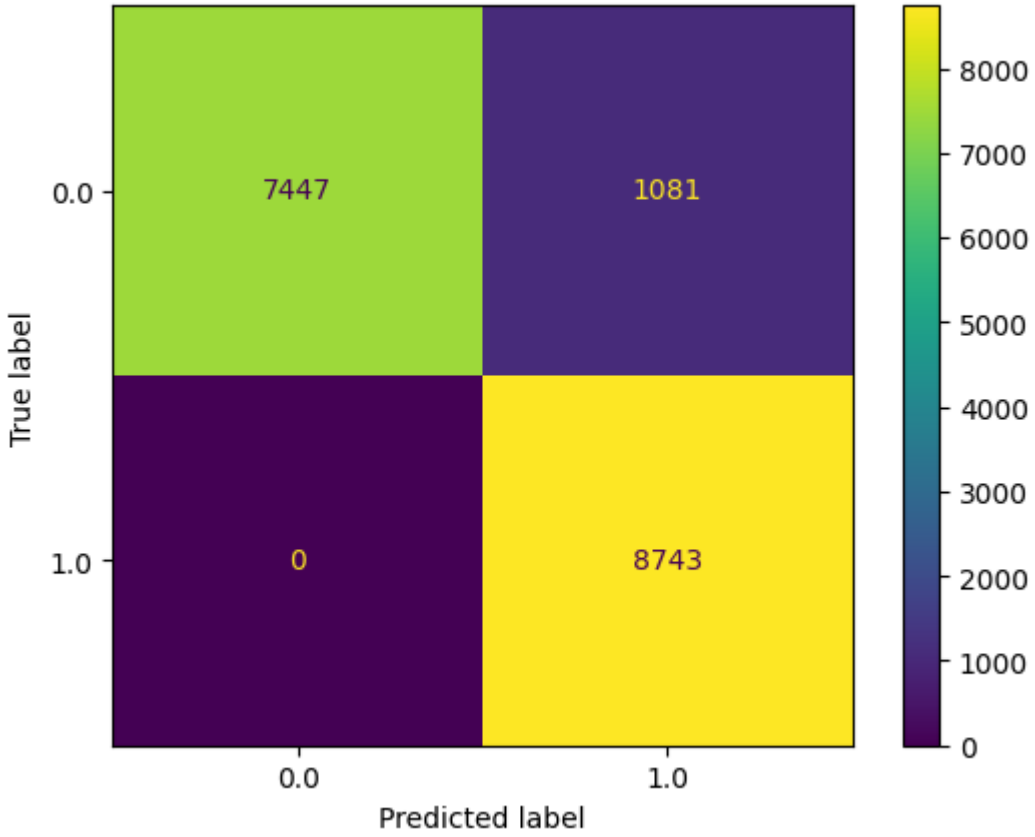


K-nearest neighbors (KNN) where k=11

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=11, metric='euclidean')
knn.fit(train_inputs, train_labels)
predictions = knn.predict(test_inputs)

print(knn.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=knn.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=knn.classes_)
disp.plot()
```

```
0.9374095304267269
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219c18ba770
>
```



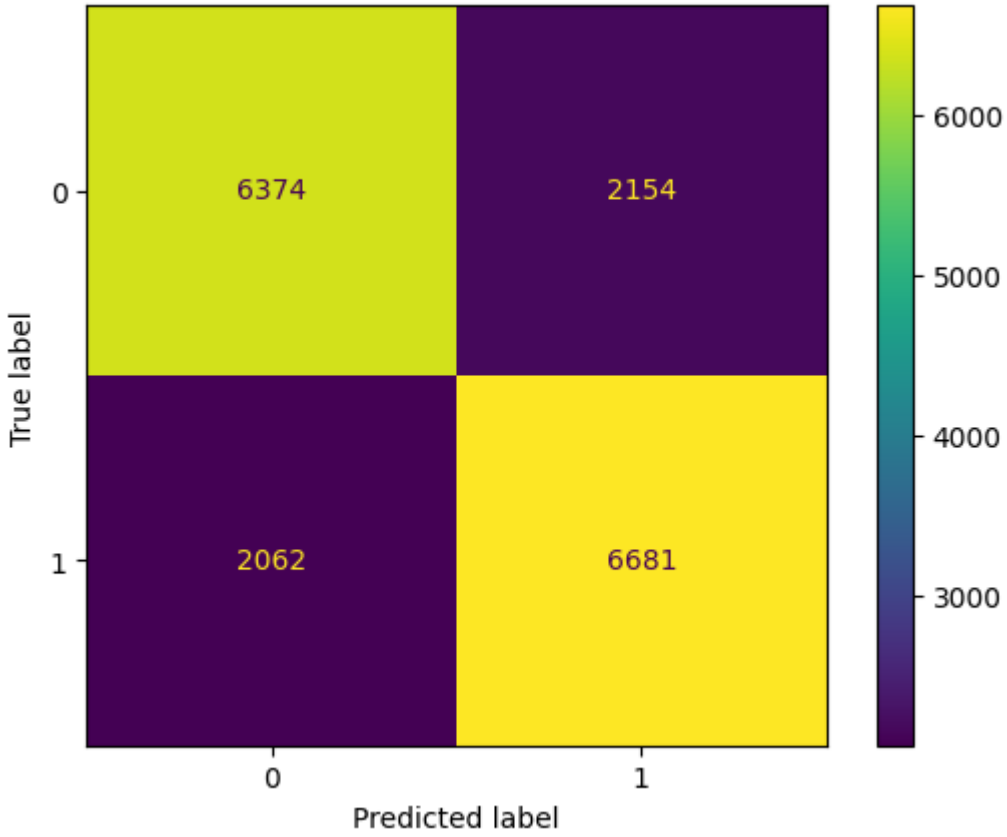
Neural network

In [ ]:
```
train_inputs_n = tf.convert_to_tensor(train_inputs, dtype=tf.float32)
test_inputs_n = tf.convert_to_tensor(test_inputs, dtype=tf.float32)
train_labels_n = tf.convert_to_tensor(train_labels, dtype=tf.float32)
test_labels_n = tf.convert_to_tensor(test_labels, dtype=tf.float32)
```

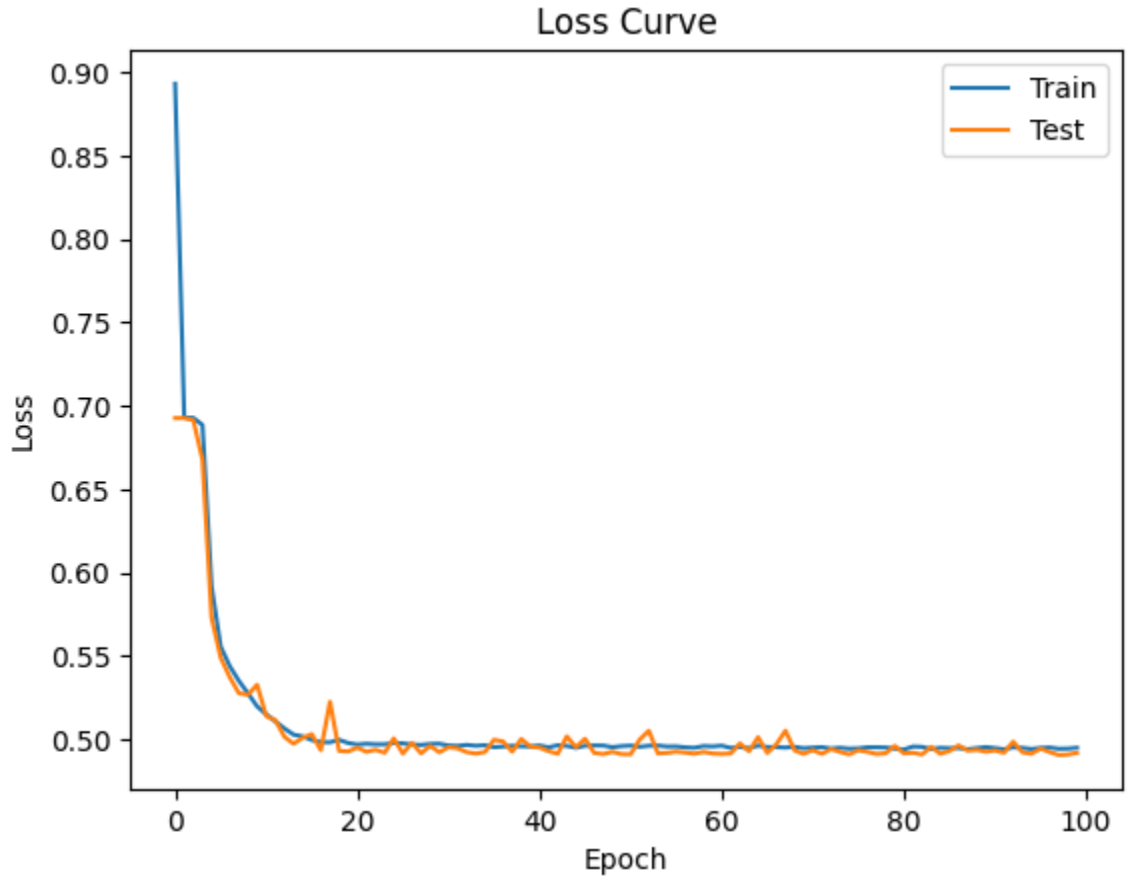
In [ ]:
```
model = Sequential()
model.add(Dense(6, activation='relu', input_dim=train_inputs_n.shape[1]))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(train_inputs_n, train_labels_n, epochs=100, verbose=0, valic

predictions = model.predict(test_inputs_n).round()
print(accuracy_score(predictions, test_labels_n))

cm = confusion_matrix(test_labels_n,predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
disp.plot()
```

```
540/540 [==============================] - 1s 867us/step
0.7558913786115453
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219c76f8a30
>
```



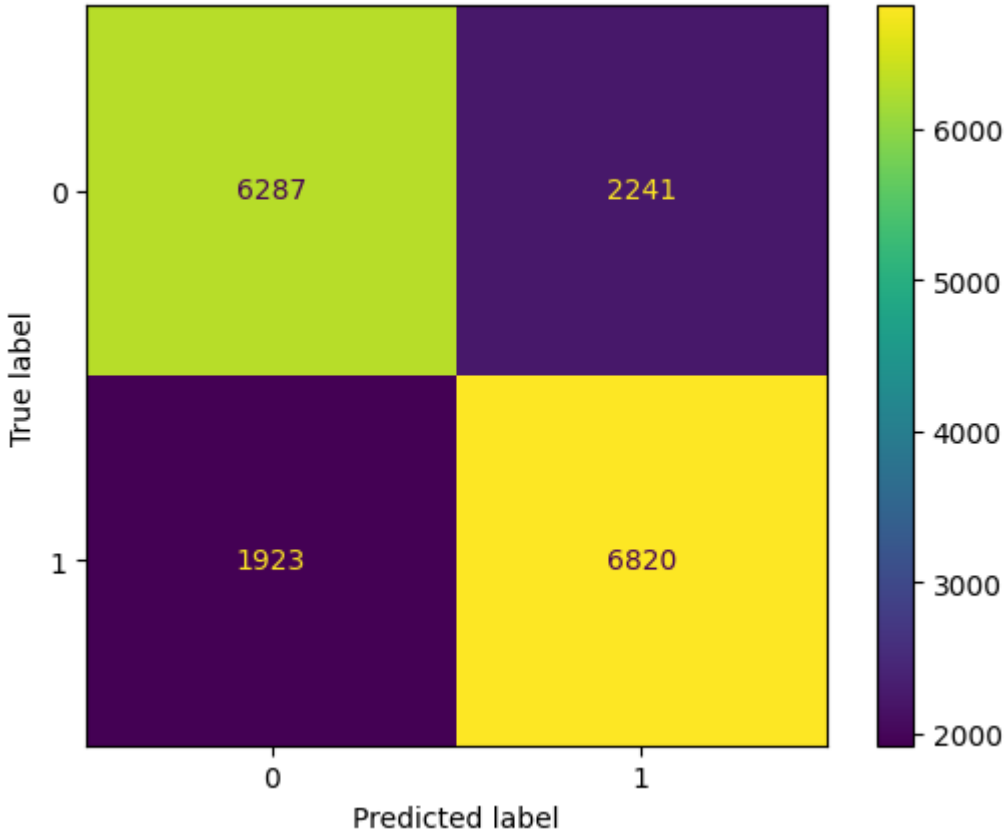
In [ ]:
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(train_loss, label='Train')
plt.plot(val_loss, label='Test')
plt.title('Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



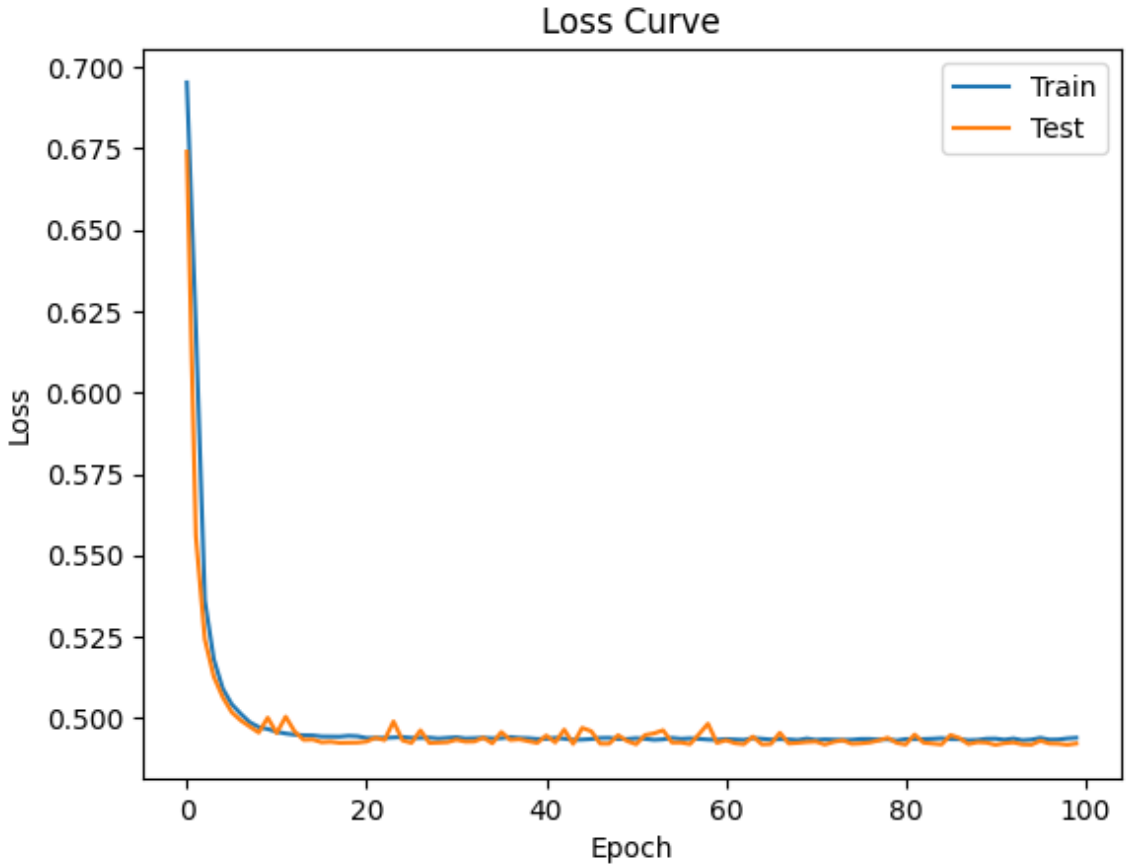
In [ ]:
```
model = Sequential()
model.add(Dense(4, activation='relu', input_dim=train_inputs_n.shape[1]))
model.add(Dense(3, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(train_inputs_n, train_labels_n, epochs=100, verbose=0, valic

predictions = model.predict(test_inputs_n).round()
print(accuracy_score(predictions, test_labels_n))

cm = confusion_matrix(test_labels_n,predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
disp.plot()
```

```
540/540 [==============================] - 0s 783us/step
0.7589022060100747
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219c8c1be50
>
```



In [ ]:
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(train_loss, label='Train')
plt.plot(val_loss, label='Test')
plt.title('Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Now we will use preprocessing and normalize our data

We scale our data with min-max scaling to reduce data in column to values from 0 to 1. Then we use PCA algorithm to reduce it even more and at the end we split data to train set and test set

In [ ]:
```
scaler = MinMaxScaler()
scaler.fit(df)
df_scaler = scaler.transform(df)
df = pd.DataFrame(df_scaler, columns=df.columns)
print(df.head())

inputs = df.values[:, :-1]
classes = df.values[:, -1]

pca= PCA(n_components=10).fit(inputs)
# print(pca)
# print(pca.explained_variance_ratio_)
# print(pca.components_)
print(pca.transform(inputs))
inputs = pca.transform(inputs)

(train_inputs, test_inputs, train_labels, test_labels) = train_test_split(inputs
```

```
   Gender       Age  Hypertension  Heart_Disease  Ever_Married  Work_Type  \
0     0.0  0.375000           0.0            0.0           1.0       1.00
1     0.0  0.944444           0.0            1.0           1.0       1.00
2     0.0  0.736111           1.0            0.0           1.0       0.75
3     0.5  0.375000           0.0            0.0           0.0       0.25
4     0.0  0.625000           0.0            0.0           1.0       0.75

   Residence_Type  Avg_Glucose_Level       BMI  Smoking_Status  Stroke
0             0.0           0.150483  0.333333             1.0     0.0
1             0.0           0.177766  0.301587             0.5     1.0
2             1.0           0.619598  0.188034             0.5     0.0
3             0.0           0.208312  0.216117             0.5     0.0
4             0.0           0.106931  0.189255             0.5     0.0
[[-0.53876789 -0.29001842 -0.15986911 ... -0.1469411  -0.28864933
   0.08350469]
 [-0.54709108  0.37871009 -0.2918493  ... -0.2850844  -0.01690297
   0.08863556]
 [ 0.52833797  0.57994913  0.46619749 ...  0.25077652 -0.14738718
  -0.10020555]
 ...
 [-0.49812259  0.07560228 -0.29117866 ... -0.12061771  0.11852752
  -0.09167947]
 [ 0.48202358 -0.00904687 -0.31958704 ... -0.03144266  0.17096538
   0.0941825 ]
 [-0.48956489 -0.89622949  0.48623093 ... -0.12697871 -0.21895356
   0.10237225]]
```

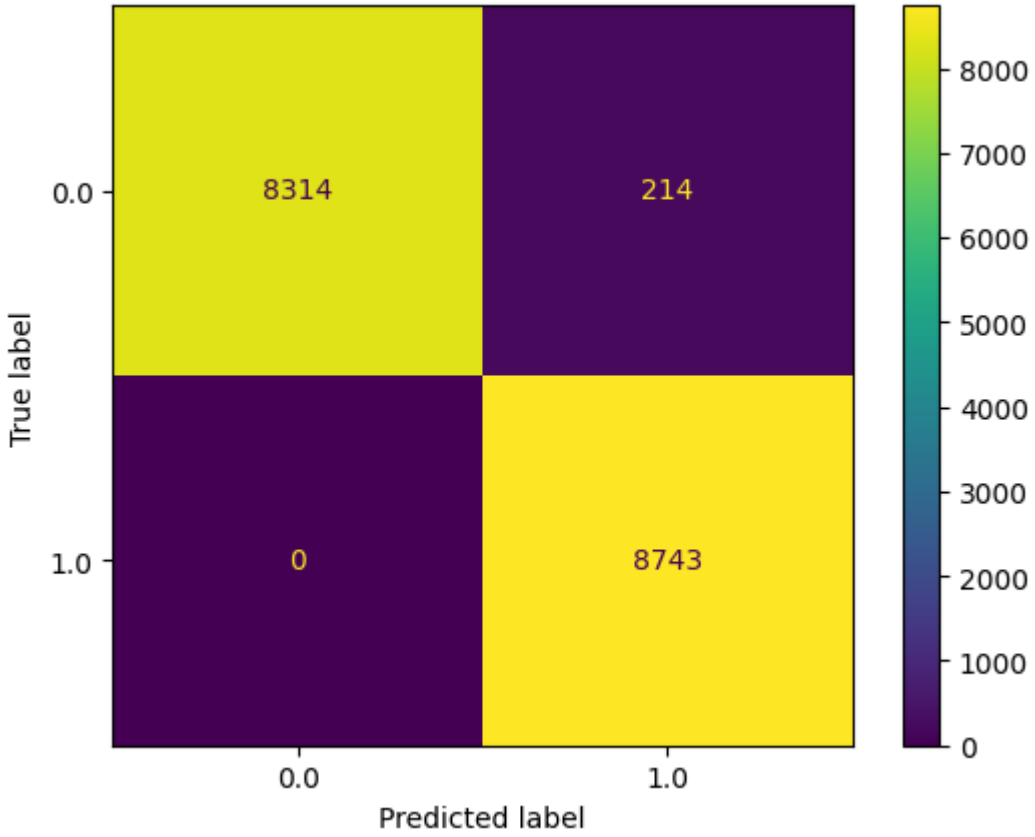
Now we use the same classifiers as above but this time with **normalized** data

Decision Tree with all nodes after preprocessing

In [ ]:
```
dtc = DecisionTreeClassifier(random_state=278779)

dtc = dtc.fit(train_inputs, train_labels)

print(dtc.score(test_inputs, test_labels))

predictions = dtc.predict(test_inputs)
cm = confusion_matrix(test_labels, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dtc.classes_)
disp.plot()
```

```
0.9876092872445139
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219bc5f9510
>
```

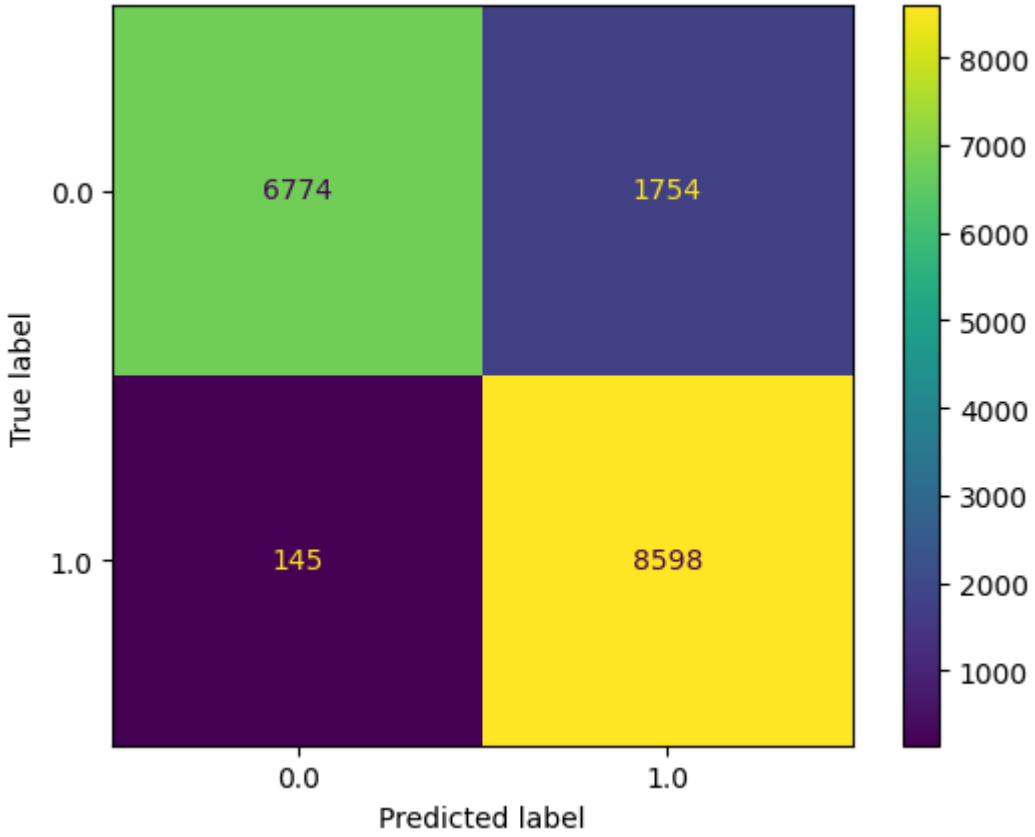


Decision tree with 10 layers of nodes after preprocessing

In [ ]:
```
dtc = DecisionTreeClassifier(random_state=278779, max_depth=10)

dtc = dtc.fit(train_inputs, train_labels)

print(dtc.score(test_inputs, test_labels))

predictions = dtc.predict(test_inputs)
cm = confusion_matrix(test_labels, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dtc.classes_)
disp.plot()
```

```
0.8900468994267848
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219bbb97580
>
```

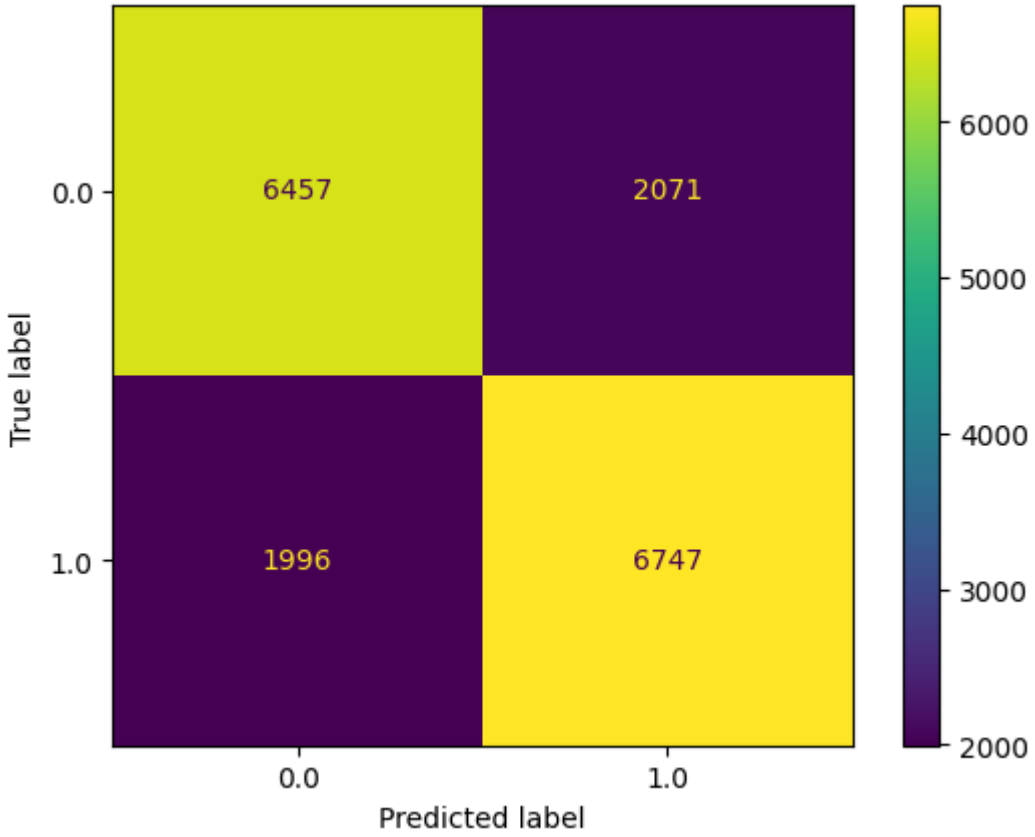


Naive Bayes after preprocessing

In [ ]:
```
bayes = GaussianNB()
bayes.fit(train_inputs, train_labels)
predictions = bayes.predict(test_inputs)

print(bayes.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=bayes.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=bayes.classes_)
disp.plot()
```

```
0.7645185571188697
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219c2c85870
>
```

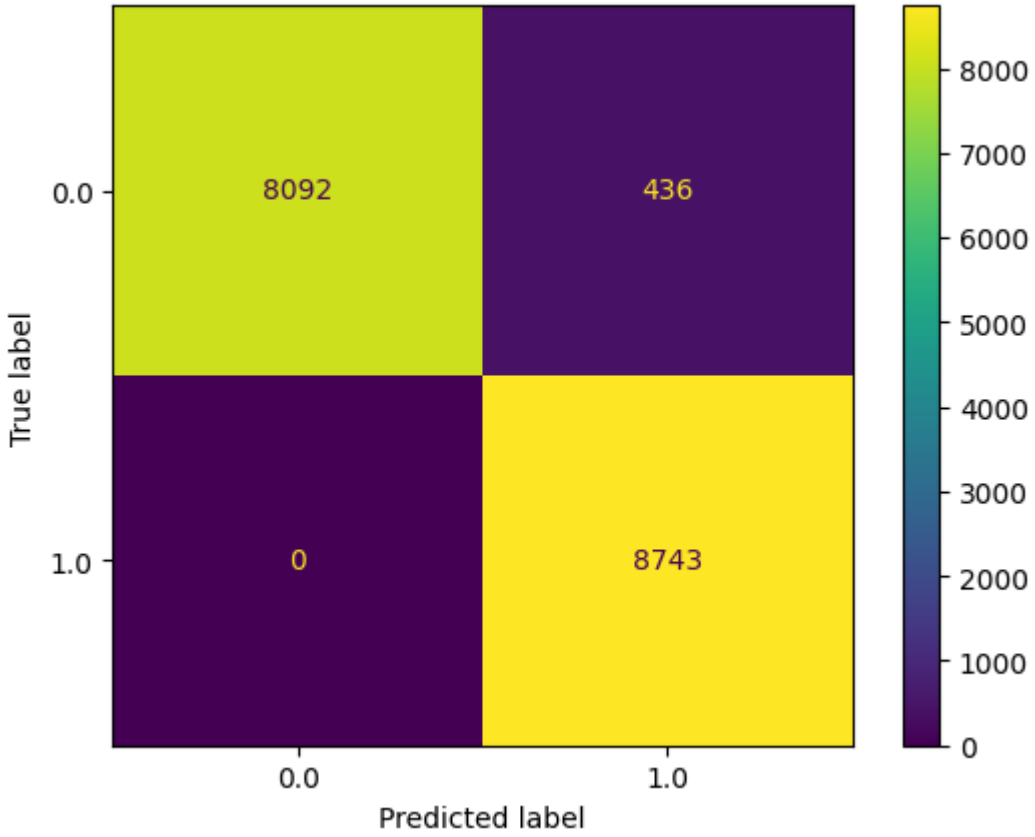


K-nearest neighbors (KNN) where k=3 after preprocessing

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn.fit(train_inputs, train_labels)
predictions = knn.predict(test_inputs)

print(knn.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=knn.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=knn.classes_)
disp.plot()
```

```
0.9747553702738695
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219c9cbde10
>
```

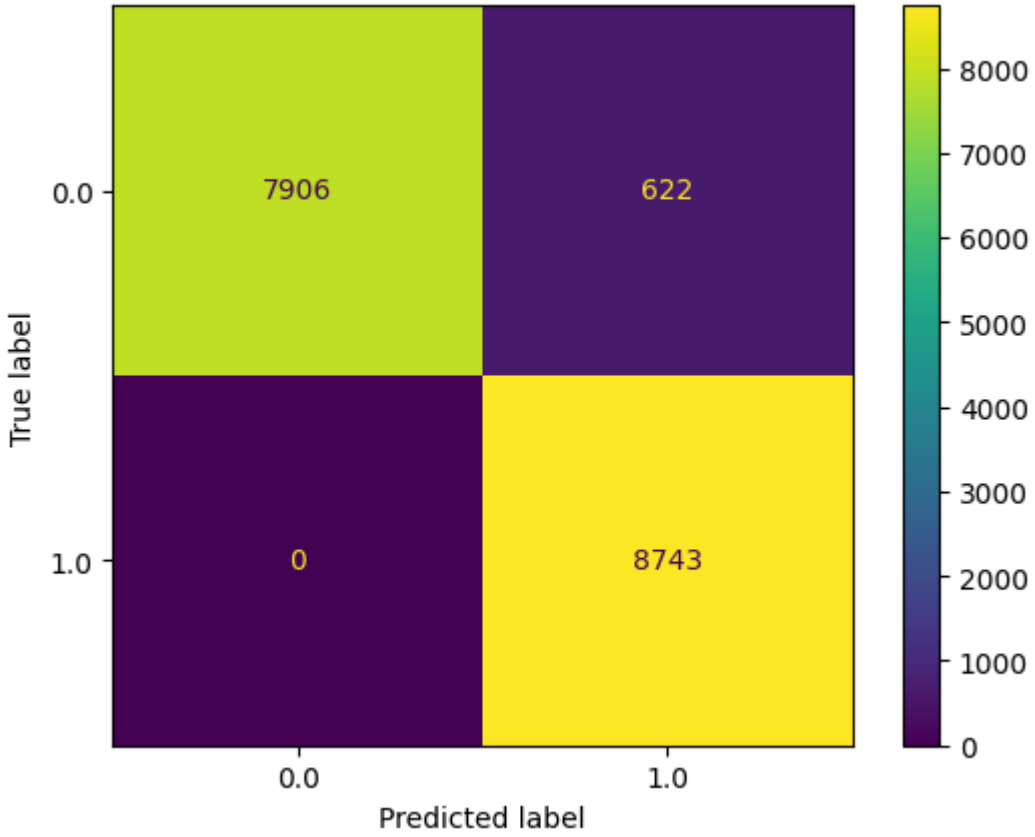


K-nearest neighbors (KNN) where k=5 after preprocessing

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(train_inputs, train_labels)
predictions = knn.predict(test_inputs)

print(knn.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=knn.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=knn.classes_)
disp.plot()
```

```
0.9639858722714376
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219bbb37520
>
```

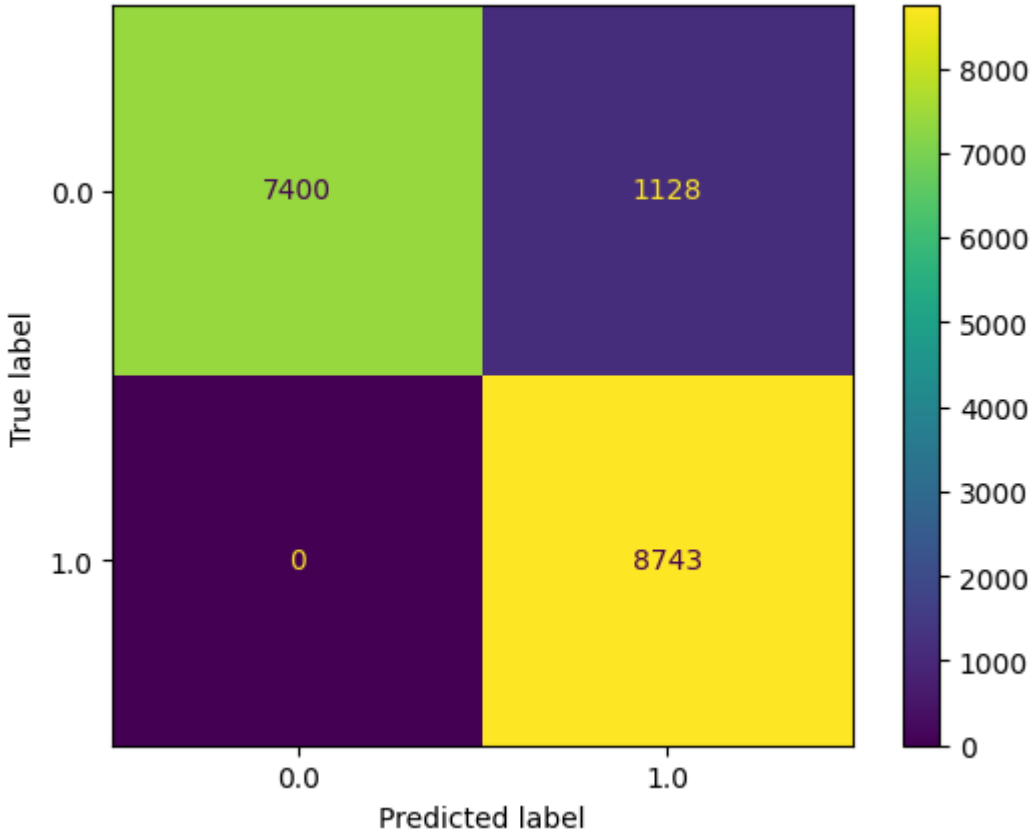


K-nearest neighbors (KNN) where k=11 after preprocessing

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=11, metric='euclidean')
knn.fit(train_inputs, train_labels)
predictions = knn.predict(test_inputs)

print(knn.score(test_inputs, test_labels))

cm = confusion_matrix(test_labels,predictions,labels=knn.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=knn.classes_)
disp.plot()
```

```
0.9346882056626715
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219bd0dceb0
>
```



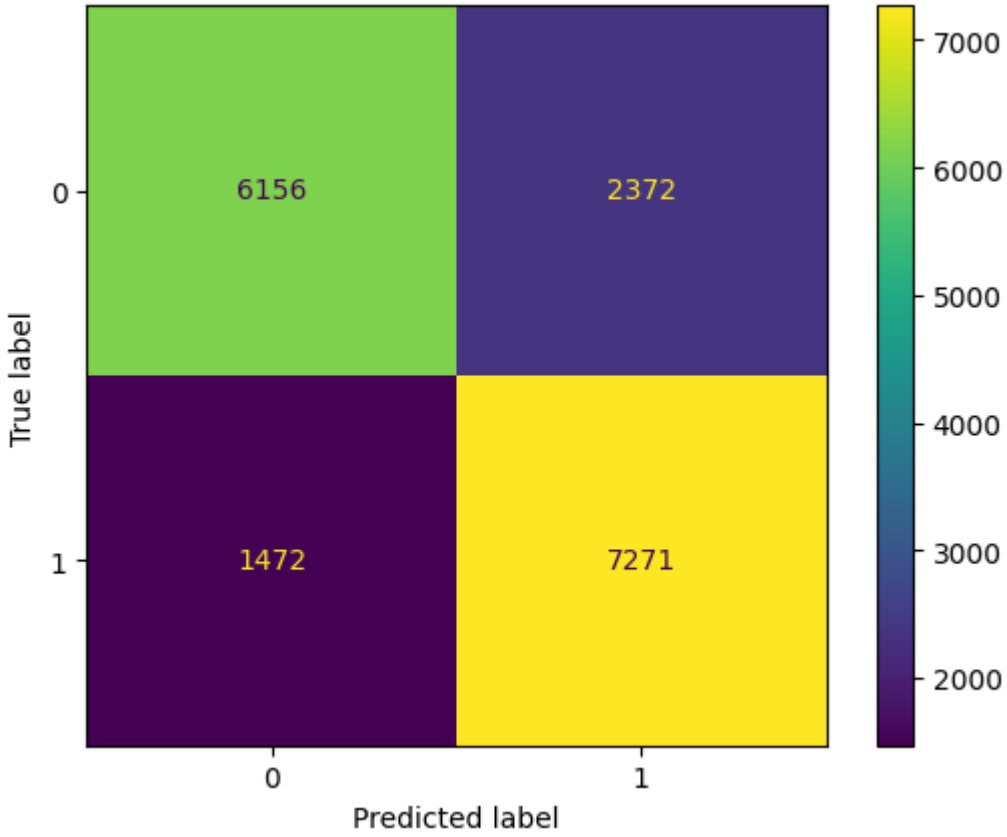
Neural network after preprocessing

In [ ]:
```
train_inputs_n = tf.convert_to_tensor(train_inputs, dtype=tf.float32)
test_inputs_n = tf.convert_to_tensor(test_inputs, dtype=tf.float32)
train_labels_n = tf.convert_to_tensor(train_labels, dtype=tf.float32)
test_labels_n = tf.convert_to_tensor(test_labels, dtype=tf.float32)
```

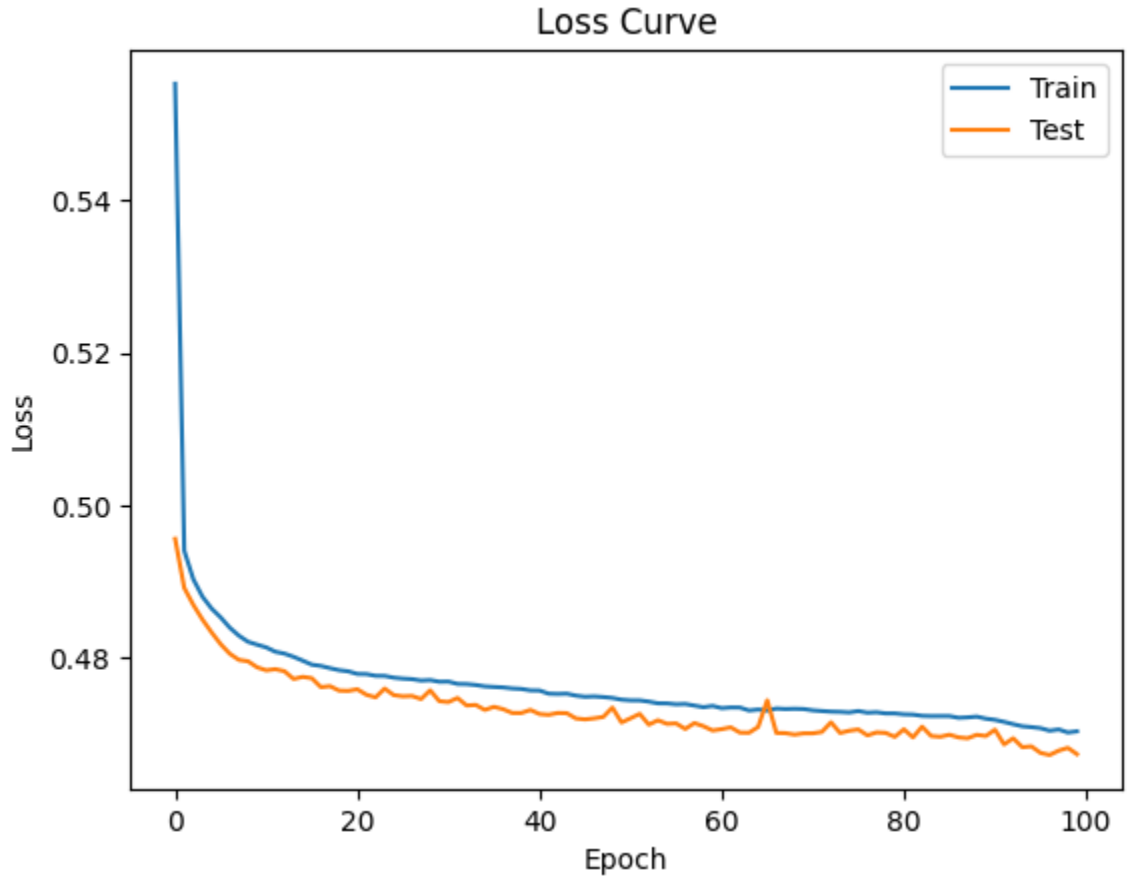
In [ ]:
```
model = Sequential()
model.add(Dense(6, activation='relu', input_dim=train_inputs_n.shape[1]))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(train_inputs_n, train_labels_n, epochs=100, verbose=0, valic

predictions = model.predict(test_inputs_n).round()
print(accuracy_score(predictions, test_labels_n))

cm = confusion_matrix(test_labels_n,predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
disp.plot()
```

```
540/540 [==============================] - 0s 750us/step
0.777430374616409
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219bd348e20
>
```



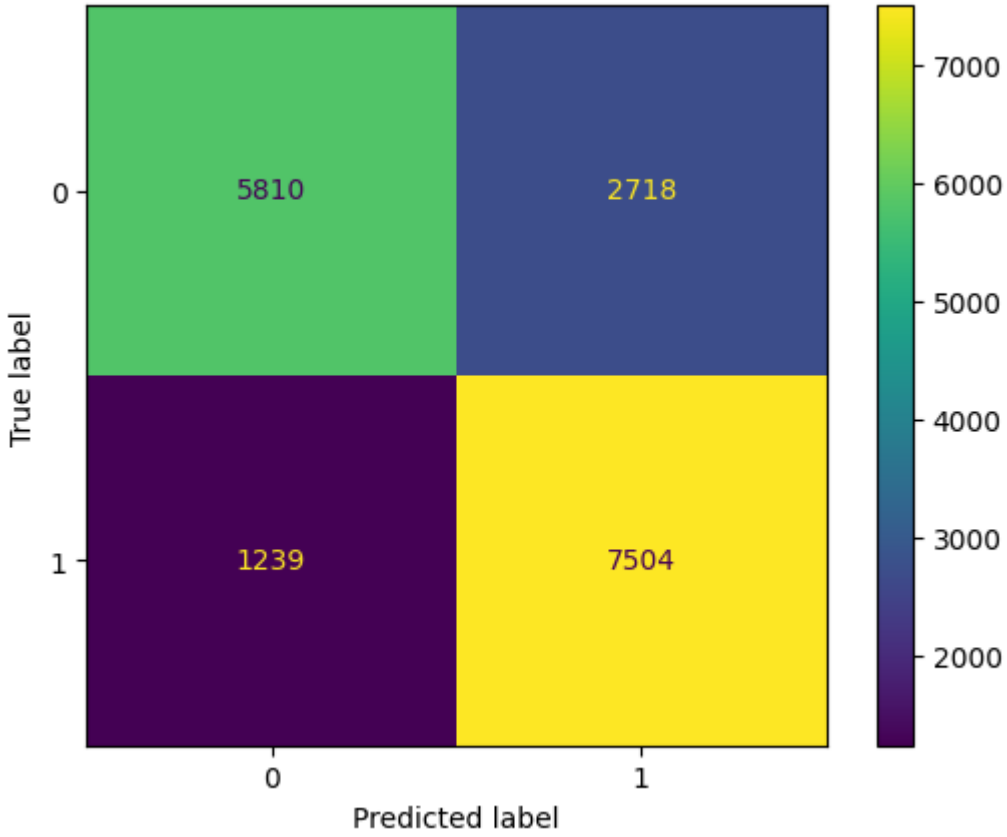
In [ ]:
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(train_loss, label='Train')
plt.plot(val_loss, label='Test')
plt.title('Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



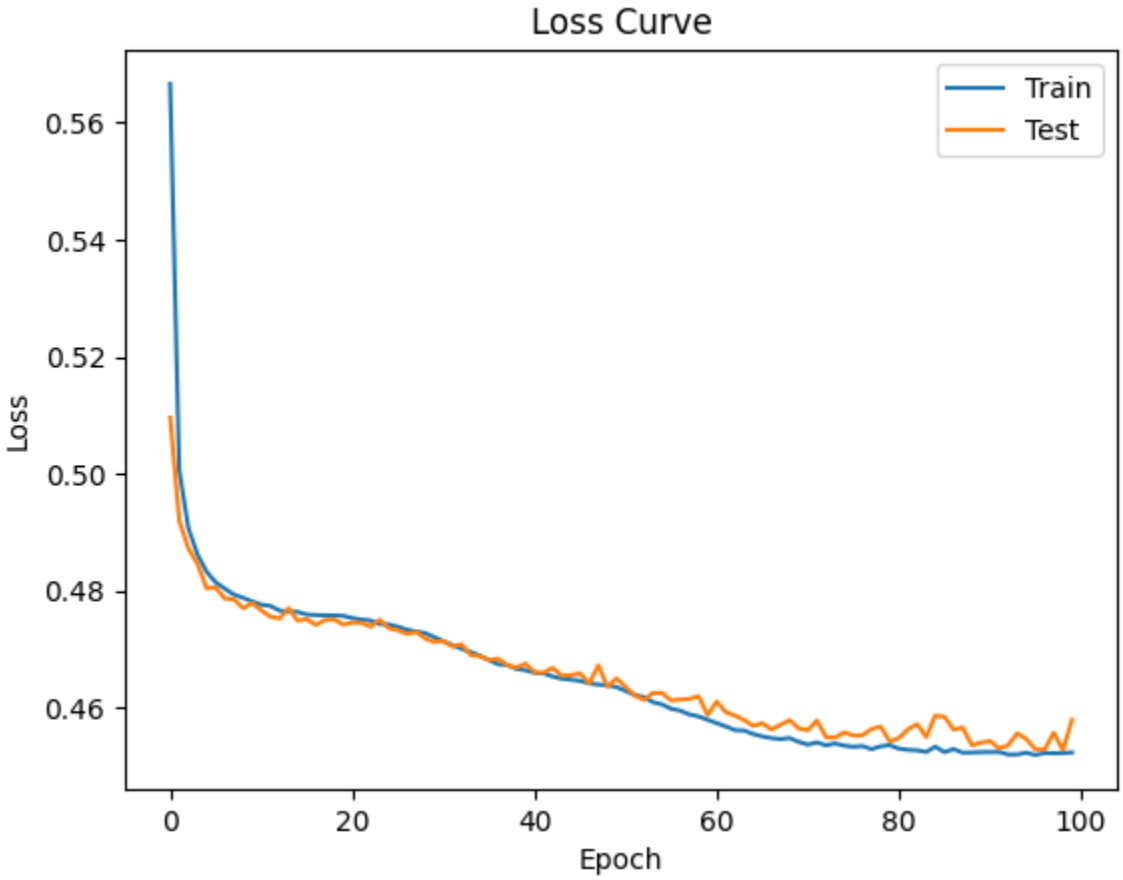
In [ ]:
```
model = Sequential()
model.add(Dense(4, activation='relu', input_dim=train_inputs_n.shape[1]))
model.add(Dense(3, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(train_inputs_n, train_labels_n, epochs=100, verbose=0, valic

predictions = model.predict(test_inputs_n).round()
print(accuracy_score(predictions, test_labels_n))

cm = confusion_matrix(test_labels_n,predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
disp.plot()
```

```
540/540 [==============================] - 0s 825us/step
0.7708876150772972
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x219d2730880
>
```



In [ ]:
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(train_loss, label='Train')
plt.plot(val_loss, label='Test')
plt.title('Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



As we can see both of neural network are overtrained because test set loss is greater than train set

To sum up

Classifier	Normal dataset	Preprocessed dataset
Decision tree (all nodes)	98.76%	98.76%
Decision tree (10 nodes)	85.76%	89.00%
Naive Bayes	75.15%	76.45%
KNN k=3	97.51%	97.48%
KNN k=5	96.44%	96.40%
KNN k=11	93.74%	93.47%
Neural network 1	75.58%	77.74%
Neural network 2	75.89%	77.09%

In the table above we have gathered all accuracies. Preprocessed dataset improved all classify algorithms except that K-nearest neighbours is slightly worse. The best accuracy has Decision Tree with all nodes. Second best is KNN algorithm with k=3. Unfortunatelly, neural networks

have really bad accuracy. It may be due to the low number of hidden layers.

Bibliography

Original dataset - https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset Enlarged dataset - https://www.kaggle.com/datasets/swetha0406/heart-stroke-prediction sklearn documentation - https://scikit-learn.org/stable/supervised_learning.html#supervised-learning pandas documentation - https://pandas.pydata.org/docs/reference/general_functions.html and lectures