In [53]:
```
import pandas as pd
import numpy as np
import imblearn
from imblearn.under_sampling import NearMiss
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_tes
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.metrics import accuracy_score, classification_report, confusion_ma
from sklearn.svm import SVC
from xgboost import XGBClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
```

In [5]:
```
#balancing the dataset here:
df = pd.read_csv('diabetes.csv')
undersample = NearMiss(version=1)
X = df.loc[:, df.columns != 'Diabetes_binary']
y = df.loc[:, df.columns == 'Diabetes_binary']
X, y = undersample.fit_resample(X, y)

#splitting the balanced dataset into train and testing samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, rand
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

#putting the balanced datasets into individual dataframes for the train and tes
df_undersampled_train = pd.DataFrame(X_train_scaled, columns = X.columns)
df_undersampled_train['Diabetes_binary'] = y_train
df_undersampled_train.head()

df_undersampled_test = pd.DataFrame(X_test_scaled, columns = X.columns)
df_undersampled_test['Diabetes_binary'] = y_test
df_undersampled_test.head()
```

Out[5]:

	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysA
0	-1.212894	0.876922	0.074482	-1.061978	1.158253	-0.225623	-0.384172	0.5
1	-1.212894	-1.140353	0.074482	0.377975	1.158253	-0.225623	-0.384172	0.5
2	0.824475	0.876922	0.074482	1.017954	-0.863369	-0.225623	-0.384172	0.5
3	-1.212894	0.876922	0.074482	0.377975	-0.863369	-0.225623	-0.384172	0.5
4	0.824475	-1.140353	0.074482	2.777896	-0.863369	-0.225623	-0.384172	0.5

5 rows × 22 columns

Features Choosing From LassoCV Selection Technique

In [51]:
```
#Creating the dataframe that will contain the features that I selected through
X_selected_train = df_undersampled_train.loc[:, ['HighBP', 'BMI', 'Smoker','Hea
                                                 'GenHlth','MentHlth', 'PhysHlt

X_selected_test = df_undersampled_test.loc[:, ['HighBP', 'BMI', 'Smoker','Heart
                                               'GenHlth','MentHlth', 'PhysHlth
```

Feature Selection From SFS Method

In [37]:
```
sfs_selected_train = df_undersampled_train.iloc[:, [3, 5, 6, 7, 9, 13, 14, 15,
sfs_selected_test = df_undersampled_test.iloc[:, [3,  5,  6, 7, 9, 13, 14, 15,
```

XGBoost Model Creation and Parameter Tuning

In [38]:
```
xgb_gs = GridSearchCV(estimator = XGBClassifier(),
                      param_grid={'objective': ['binary:logistic', 'binary:logit
                                  'learning_rate':[0.3, 0.6, 0.9],
                                  'max_depth': [6, 12, 18],
                                  'min_child_weight': [1, 5, 10],
                                  'subsamples': [0.5, 0.75, 1],
                                  'colsample_bytree': [0.5, 0.7, 1],
                                  'seed':[42]},
                      cv = 5,
                      verbose = 3,
                      n_jobs = -1)
xgb_gs.fit(X_selected_train, y_train.values.ravel())

print(xgb_gs.best_params_)
print(xgb_gs.best_score_)
```

```
Fitting 5 folds for each of 729 candidates, totalling 3645 fits
{'colsample_bytree': 0.5, 'learning_rate': 0.3, 'max_depth': 6, 'min_child_we
ight': 1, 'objective': 'binary:logistic', 'seed': 42, 'subsamples': 0.5}
0.8569840047721933

C:\Users\Felipe\anaconda3\lib\site-packages\xgboost\core.py:160: UserWarning:
[16:02:00] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscali
ng-group-i-0750514818a16474a-1\xgboost\xgboost-ci-windows\src\learner.cc:742:
Parameters: { "subsamples" } are not used.

  warnings.warn(smsg, UserWarning)
```

XGBoost Model Performance

Important to note what is a XGBoost model

```
* It is an implementation of gradient boosting decision trees
* Creates a series of models and combines them to make an overall mode
l that is more accurate
```

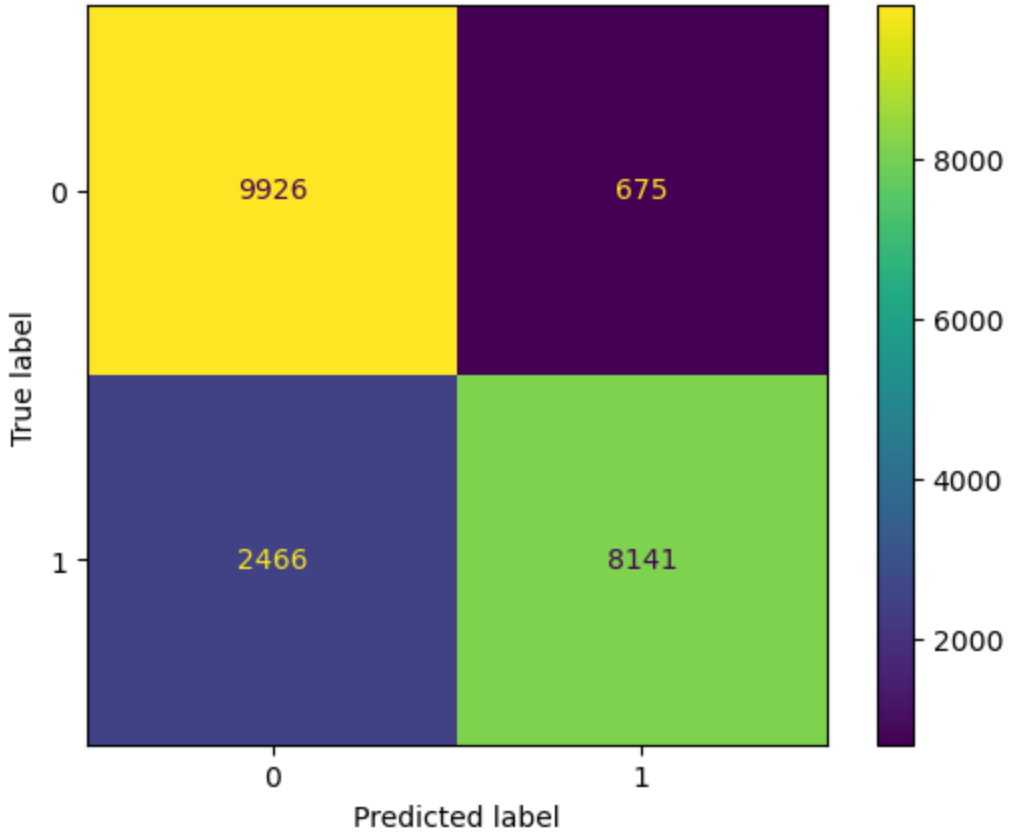
Why to use XGBOost

```
* It is a robust machine learning model
* Easy to use
* Fast on large datasets
* There isnt much of a need for optimization but still encouraged
* OFfers built in regularization
* Handles sparse data sets using weighted quantile sketch algorithm
    * Meaning that it deals with non-zero entries and maintains its co
mputational complexity like stochastic gradient descent
* Easy to scale up on multicore machines or clusters
* uses disk based data structures instead of in-memory ones during com
putation
```

In [54]:
```
xgb = XGBClassifier(colsample_bytree = 0.5, learning_rate = 0.3, max_depth=6, r
                    objective='binary:hinge', seed=1, subsamples=0.5)
xgb.fit(X_selected_train, y_train.values.ravel())

xbg_train_pred = xgb.predict(X_selected_train)
xbg_train_score = accuracy_score(y_train.values.ravel(), xbg_train_pred)

xbg_test_pred = xgb.predict(X_selected_test)
xbg_test_score = accuracy_score(y_test.values.ravel(), xbg_test_pred)

print(f'Train Accuracy: {xbg_train_score}')
print(f'Classification Report for Train: \n{classification_report(y_train.value
print(f'Confusion Matrix:\n{confusion_matrix(y_train.values.ravel(), xbg_train_

print(f'Test Accuracy: {xbg_test_score}')
print(f'Classification Report for Test: \n{classification_report(y_test.values
print(f'Confusion Matrix:\n{confusion_matrix(y_test.values.ravel(), xbg_test_pr

disp = ConfusionMatrixDisplay(confusion_matrix(y_test.values.ravel(), xbg_test_
disp.plot()
plt.show()
```

```
C:\Users\Felipe\anaconda3\lib\site-packages\xgboost\core.py:160: UserWarning:
[02:02:34] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscali
ng-group-i-0750514818a16474a-1\xgboost\xgboost-ci-windows\src\learner.cc:742:
Parameters: { "subsamples" } are not used.

  warnings.warn(smsg, UserWarning)
```

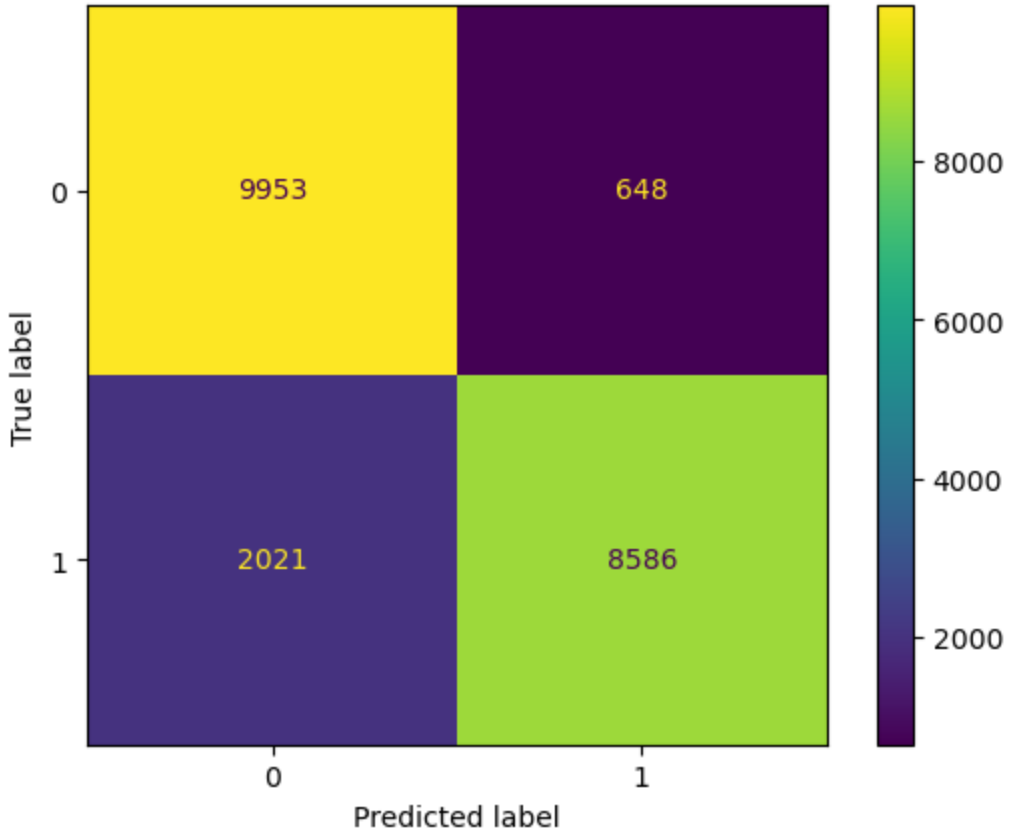
```
Train Accuracy: 0.8570447013175976
Classification Report for Train:
              precision    recall  f1-score   support

         0.0       0.81      0.94      0.87     24745
         1.0       0.93      0.77      0.84     24739

    accuracy                           0.86     49484
   macro avg       0.87      0.86      0.86     49484
weighted avg       0.87      0.86      0.86     49484


Confusion Matrix:
[[23271  1474]
 [ 5600 19139]]

Test Accuracy: 0.8518955111278763
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.80      0.94      0.86     10601
         1.0       0.92      0.77      0.84     10607

    accuracy                           0.85     21208
   macro avg       0.86      0.85      0.85     21208
weighted avg       0.86      0.85      0.85     21208


Confusion Matrix:
[[9926  675]
 [2466 8141]]
```


9926
675
2466
8141
0
1
True label
0
1
Predicted label
8000
6000
4000
2000

In [55]:
```
xgb = XGBClassifier(colsample_bytree = 0.5, learning_rate = 0.3, max_depth=6, r
                    objective='binary:hinge', seed=1, subsamples=0.5)
xgb.fit(sfs_selected_train, y_train.values.ravel())

xbg_train_pred = xgb.predict(sfs_selected_train)
xbg_train_score = accuracy_score(y_train.values.ravel(), xbg_train_pred)

xbg_test_pred = xgb.predict(sfs_selected_test)
xbg_test_score = accuracy_score(y_test.values.ravel(), xbg_test_pred)

print(f'Train Accuracy: {xbg_train_score}')
print(f'Classification Report for Train: \n{classification_report(y_train.value
print(f'Confusion Matrix:\n{confusion_matrix(y_train.values.ravel(), xbg_train_

print(f'Test Accuracy: {xbg_test_score}')
print(f'Classification Report for Test: \n{classification_report(y_test.values
print(f'Confusion Matrix:\n{confusion_matrix(y_test.values.ravel(), xbg_test_pr

disp = ConfusionMatrixDisplay(confusion_matrix(y_test.values.ravel(), xbg_test_
disp.plot()
plt.show()
```

```
C:\Users\Felipe\anaconda3\lib\site-packages\xgboost\core.py:160: UserWarning:
[02:02:55] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscali
ng-group-i-0750514818a16474a-1\xgboost\xgboost-ci-windows\src\learner.cc:742:
Parameters: { "subsamples" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Train Accuracy: 0.8809514186403686
Classification Report for Train:
              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     24745
         1.0       0.94      0.82      0.87     24739

    accuracy                           0.88     49484
   macro avg       0.89      0.88      0.88     49484
weighted avg       0.89      0.88      0.88     49484


Confusion Matrix:
[[23412  1333]
 [ 4558 20181]]

Test Accuracy: 0.8741512636740852
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.83      0.94      0.88     10601
         1.0       0.93      0.81      0.87     10607

    accuracy                           0.87     21208
   macro avg       0.88      0.87      0.87     21208
weighted avg       0.88      0.87      0.87     21208


Confusion Matrix:
[[9953  648]
 [2021 8586]]
```



Extreme Machine Learning Model

- While it is a feed forward neural network it has a few differences
  - First, is that it does not use back propogation
  - Second, the weights for the hidden layers are randomly generated
  - Lastly, the output is linear rather than sigmoidal

In [41]:
```
input_length = X_selected_train.shape[1]
hidden_units = 1000

win = np.random.normal(size = [input_length, hidden_units])

def input_to_hidden(x):
    a = np.dot(x, win)
    a = np.maximum(a, 0, a)
    return a

x_h_v = input_to_hidden(X_selected_train)
x_h_t = np.transpose(x_h_v)
w_out = np.dot(np.linalg.inv(np.dot(x_h_t, x_h_v)), np.dot(x_h_t, y_train))

def predict(x):
    x = input_to_hidden(x)
    y = np.dot(x, w_out)
    return y

extreme_pred = predict(X_selected_test)
num_correct = 0
total = extreme_pred.shape[0]
for i in range(total):
    predicted = np.argmax(extreme_pred[i])
    test = np.argmax(y_test.values.ravel()[i])
    num_correct = num_correct + (1 if predicted == test else 0)

print('Accuracy of test set: {:f}'.format(num_correct/total))

extreme_pred = predict(X_selected_train)
num_correct = 0
total = extreme_pred.shape[0]
for i in range(total):
    predicted = np.argmax(extreme_pred[i])
    train = np.argmax(y_train.values.ravel()[i])
    num_correct = num_correct + (1 if predicted == train else 0)

print('Accuracy of train set: {:f}'.format(num_correct/total))
```

```
Accuracy of test set: 1.000000
Accuracy of train set: 1.000000
```

In [42]:
```
input_length = sfs_selected_train.shape[1]
hidden_units = 1000

win = np.random.normal(size = [input_length, hidden_units])

def input_to_hidden(x):
    a = np.dot(x, win)
    a = np.maximum(a, 0, a)
    return a

x_h_v = input_to_hidden(sfs_selected_train)
x_h_t = np.transpose(x_h_v)
w_out = np.dot(np.linalg.inv(np.dot(x_h_t, x_h_v)), np.dot(x_h_t, y_train))

def predict(x):
    x = input_to_hidden(x)
    y = np.dot(x, w_out)
    return y

extreme_pred = predict(sfs_selected_test)
num_correct = 0
total = extreme_pred.shape[0]
for i in range(total):
    predicted = np.argmax(extreme_pred[i])
    test = np.argmax(y_test.values.ravel()[i])
    num_correct = num_correct + (1 if predicted == test else 0)

print('Accuracy of test set: {:f}'.format(num_correct/total))

extreme_pred = predict(sfs_selected_train)
num_correct = 0
total = extreme_pred.shape[0]
for i in range(total):
    predicted = np.argmax(extreme_pred[i])
    train = np.argmax(y_train.values.ravel()[i])
    num_correct = num_correct + (1 if predicted == train else 0)

print('Accuracy of train set: {:f}'.format(num_correct/total))
```

```
Accuracy of test set: 1.000000
Accuracy of train set: 1.000000
```

Basic Deep Learning Model

- Basic feed forward neural network
  - This neural network has 2 hidden layers, 1 input and 1 output layer

In [72]:
```
nn = Sequential()
nn.add(Dense(20, input_shape=(9,), activation='relu'))
nn.add(Dense(40, activation='relu'))
nn.add(Dense(10, activation = 'relu'))
nn.add(Dense(1, activation='sigmoid'))

nn.compile(loss = 'binary_crossentropy', optimizer='adam', metrics='accuracy')
nn.fit(X_selected_train, y_train.values.ravel(), epochs=50, batch_size=10)

_, train_accuracy = nn.evaluate(X_selected_train, y_train)
_2, test_accuracy = nn.evaluate(X_selected_test, y_test)

print(f'Train Accuracy: {train_accuracy}')
print(f'Test Accuracy: {test_accuracy}')
```

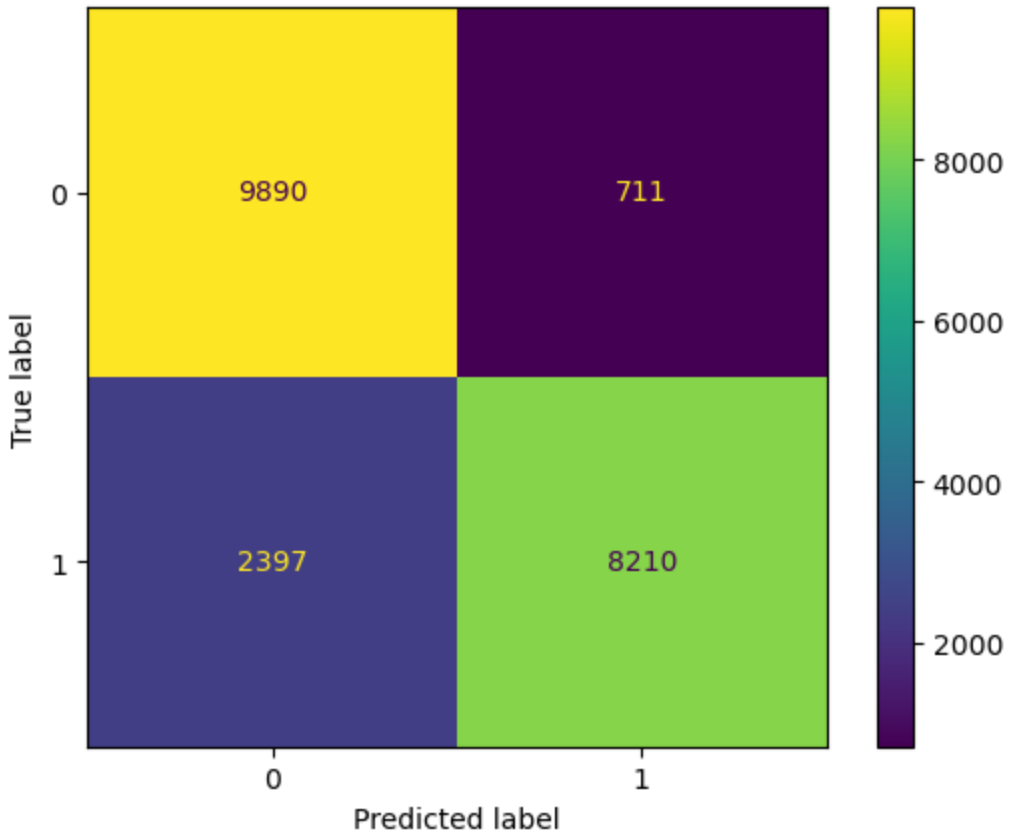
```
Epoch 1/50
4949/4949 [==============================] - 4s 722us/step - loss: 0.3511 - a
ccuracy: 0.8416
Epoch 2/50
4949/4949 [==============================] - 4s 734us/step - loss: 0.3299 - a
ccuracy: 0.8520
Epoch 3/50
4949/4949 [==============================] - 4s 763us/step - loss: 0.3263 - a
ccuracy: 0.8543
Epoch 4/50
4949/4949 [==============================] - 4s 757us/step - loss: 0.3235 - a
ccuracy: 0.8544
Epoch 5/50
4949/4949 [==============================] - 4s 769us/step - loss: 0.3231 - a
ccuracy: 0.8538
Epoch 6/50
4949/4949 [==============================] - 4s 734us/step - loss: 0.3225 - a
ccuracy: 0.8542
Epoch 7/50
4949/4949 [==============================] - 4s 734us/step - loss: 0.3216 - a
ccuracy: 0.8554
Epoch 8/50
4949/4949 [==============================] - 4s 736us/step - loss: 0.3209 - a
ccuracy: 0.8557
Epoch 9/50
4949/4949 [==============================] - 4s 733us/step - loss: 0.3204 - a
ccuracy: 0.8564
Epoch 10/50
4949/4949 [==============================] - 4s 726us/step - loss: 0.3199 - a
ccuracy: 0.8554
Epoch 11/50
4949/4949 [==============================] - 4s 723us/step - loss: 0.3199 - a
ccuracy: 0.8562
Epoch 12/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.3199 - a
ccuracy: 0.8562
Epoch 13/50
4949/4949 [==============================] - 4s 729us/step - loss: 0.3193 - a
ccuracy: 0.8559
Epoch 14/50
4949/4949 [==============================] - 4s 733us/step - loss: 0.3196 - a
ccuracy: 0.8556
Epoch 15/50
4949/4949 [==============================] - 4s 740us/step - loss: 0.3192 - a
ccuracy: 0.8567
Epoch 16/50
4949/4949 [==============================] - 4s 736us/step - loss: 0.3191 - a
ccuracy: 0.8561
Epoch 17/50
4949/4949 [==============================] - 4s 732us/step - loss: 0.3184 - a
ccuracy: 0.8568
Epoch 18/50
4949/4949 [==============================] - 4s 728us/step - loss: 0.3184 - a
ccuracy: 0.8566
Epoch 19/50
4949/4949 [==============================] - 4s 728us/step - loss: 0.3186 - a
ccuracy: 0.8571
```

```
Epoch 20/50
4949/4949 [==============================] - 4s 733us/step - loss: 0.3183 - a
ccuracy: 0.8570
Epoch 21/50
4949/4949 [==============================] - 4s 735us/step - loss: 0.3186 - a
ccuracy: 0.8570
Epoch 22/50
4949/4949 [==============================] - 4s 731us/step - loss: 0.3182 - a
ccuracy: 0.8572
Epoch 23/50
4949/4949 [==============================] - 4s 738us/step - loss: 0.3178 - a
ccuracy: 0.8566
Epoch 24/50
4949/4949 [==============================] - 4s 745us/step - loss: 0.3176 - a
ccuracy: 0.8567
Epoch 25/50
4949/4949 [==============================] - 4s 729us/step - loss: 0.3178 - a
ccuracy: 0.8575
Epoch 26/50
4949/4949 [==============================] - 4s 740us/step - loss: 0.3176 - a
ccuracy: 0.8573
Epoch 27/50
4949/4949 [==============================] - 4s 731us/step - loss: 0.3175 - a
ccuracy: 0.8567
Epoch 28/50
4949/4949 [==============================] - 4s 725us/step - loss: 0.3176 - a
ccuracy: 0.8577
Epoch 29/50
4949/4949 [==============================] - 4s 747us/step - loss: 0.3175 - a
ccuracy: 0.8576
Epoch 30/50
4949/4949 [==============================] - 4s 738us/step - loss: 0.3174 - a
ccuracy: 0.8567
Epoch 31/50
4949/4949 [==============================] - 4s 733us/step - loss: 0.3173 - a
ccuracy: 0.8572
Epoch 32/50
4949/4949 [==============================] - 4s 744us/step - loss: 0.3173 - a
ccuracy: 0.8569
Epoch 33/50
4949/4949 [==============================] - 4s 743us/step - loss: 0.3173 - a
ccuracy: 0.8568
Epoch 34/50
4949/4949 [==============================] - 4s 722us/step - loss: 0.3175 - a
ccuracy: 0.8572
Epoch 35/50
4949/4949 [==============================] - 4s 729us/step - loss: 0.3169 - a
ccuracy: 0.8571
Epoch 36/50
4949/4949 [==============================] - 4s 738us/step - loss: 0.3172 - a
ccuracy: 0.8579
Epoch 37/50
4949/4949 [==============================] - 4s 720us/step - loss: 0.3170 - a
ccuracy: 0.8574
Epoch 38/50
4949/4949 [==============================] - 4s 736us/step - loss: 0.3169 - a
ccuracy: 0.8573
```

```
Epoch 39/50
4949/4949 [==============================] - 4s 732us/step - loss: 0.3167 - a
ccuracy: 0.8579
Epoch 40/50
4949/4949 [==============================] - 4s 764us/step - loss: 0.3168 - a
ccuracy: 0.8573
Epoch 41/50
4949/4949 [==============================] - 4s 738us/step - loss: 0.3165 - a
ccuracy: 0.8574
Epoch 42/50
4949/4949 [==============================] - 4s 721us/step - loss: 0.3169 - a
ccuracy: 0.8573
Epoch 43/50
4949/4949 [==============================] - 4s 736us/step - loss: 0.3165 - a
ccuracy: 0.8574
Epoch 44/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.3163 - a
ccuracy: 0.8571
Epoch 45/50
4949/4949 [==============================] - 4s 759us/step - loss: 0.3166 - a
ccuracy: 0.8575
Epoch 46/50
4949/4949 [==============================] - 4s 761us/step - loss: 0.3164 - a
ccuracy: 0.8582
Epoch 47/50
4949/4949 [==============================] - 4s 757us/step - loss: 0.3163 - a
ccuracy: 0.8576
Epoch 48/50
4949/4949 [==============================] - 4s 735us/step - loss: 0.3163 - a
ccuracy: 0.8573
Epoch 49/50
4949/4949 [==============================] - 4s 754us/step - loss: 0.3162 - a
ccuracy: 0.8572
Epoch 50/50
4949/4949 [==============================] - 4s 765us/step - loss: 0.3162 - a
ccuracy: 0.8572
1547/1547 [==============================] - 1s 632us/step - loss: 0.3139 - a
ccuracy: 0.8587
663/663 [==============================] - 0s 613us/step - loss: 0.3279 - acc
uracy: 0.8535
Train Accuracy: 0.8587422370910645
Test Accuracy: 0.8534515500068665
```

In [73]:
```
nn_test_pred = nn.predict(X_selected_test)
nn_test_pred = np.round(nn_test_pred)

cm = confusion_matrix(y_test, nn_test_pred)

disp = ConfusionMatrixDisplay(confusion_matrix(y_test.values.ravel(), nn_test_p
disp.plot()
plt.show()

print(f'Test Accuracy: {test_accuracy}')
print(f'Classification Report for Test: \n{classification_report(y_test.values
```

```
663/663 [==============================] - 0s 621us/step
```




```
Test Accuracy: 0.8534515500068665
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.80      0.93      0.86     10601
         1.0       0.92      0.77      0.84     10607

    accuracy                           0.85     21208
   macro avg       0.86      0.85      0.85     21208
weighted avg       0.86      0.85      0.85     21208
```

In [68]:
```
nn = Sequential()
nn.add(Dense(20, input_shape=(9,), activation='relu'))
nn.add(Dense(40, activation='relu'))
nn.add(Dense(10, activation = 'relu'))
nn.add(Dense(1, activation='sigmoid'))

nn.compile(loss = 'binary_crossentropy', optimizer='adam', metrics='accuracy')
nn.fit(sfs_selected_train, y_train.values.ravel(), epochs=50, batch_size=10)

_, train_accuracy = nn.evaluate(sfs_selected_train, y_train)
_2, test_accuracy = nn.evaluate(sfs_selected_test, y_test)

print(f'Train Accuracy: {train_accuracy}')
print(f'Test Accuracy: {test_accuracy}')
```

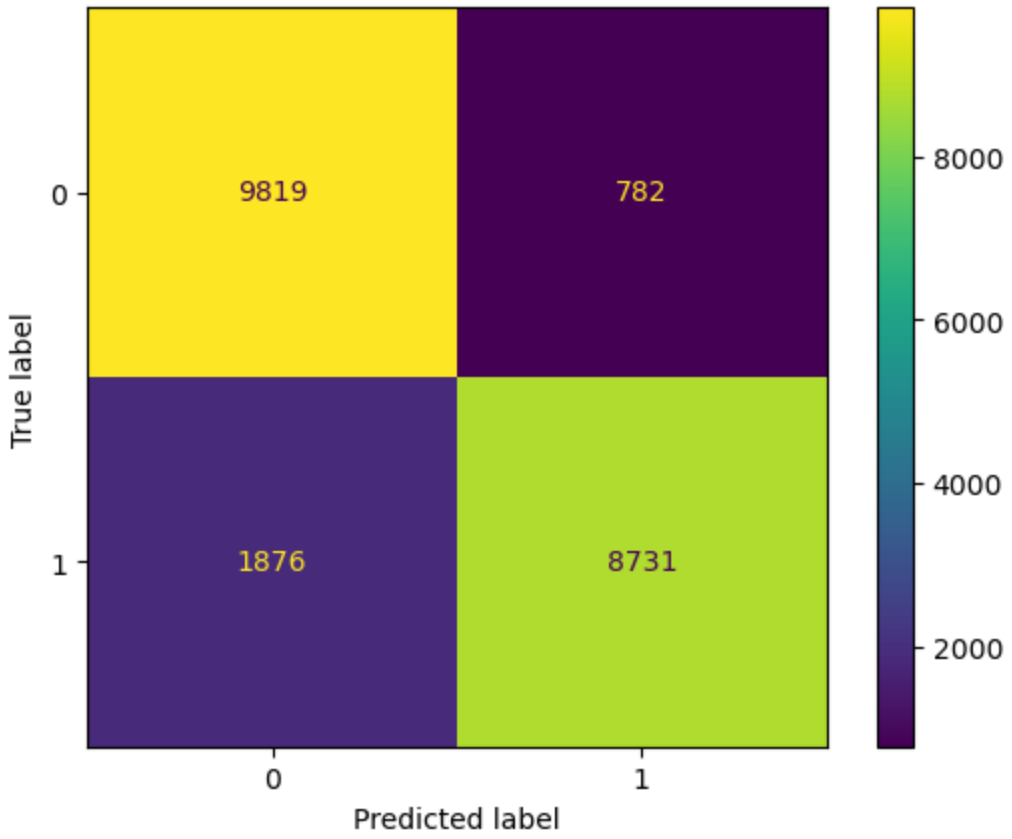
```
Epoch 1/50
4949/4949 [==============================] - 4s 729us/step - loss: 0.3124 - a
ccuracy: 0.8634
Epoch 2/50
4949/4949 [==============================] - 4s 728us/step - loss: 0.2928 - a
ccuracy: 0.8752
Epoch 3/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.2896 - a
ccuracy: 0.8757
Epoch 4/50
4949/4949 [==============================] - 4s 743us/step - loss: 0.2883 - a
ccuracy: 0.8760
Epoch 5/50
4949/4949 [==============================] - 4s 735us/step - loss: 0.2871 - a
ccuracy: 0.8759
Epoch 6/50
4949/4949 [==============================] - 4s 745us/step - loss: 0.2861 - a
ccuracy: 0.8761
Epoch 7/50
4949/4949 [==============================] - 4s 748us/step - loss: 0.2857 - a
ccuracy: 0.8776
Epoch 8/50
4949/4949 [==============================] - 4s 738us/step - loss: 0.2850 - a
ccuracy: 0.8778
Epoch 9/50
4949/4949 [==============================] - 4s 724us/step - loss: 0.2850 - a
ccuracy: 0.8775
Epoch 10/50
4949/4949 [==============================] - 4s 725us/step - loss: 0.2839 - a
ccuracy: 0.8779
Epoch 11/50
4949/4949 [==============================] - 4s 724us/step - loss: 0.2843 - a
ccuracy: 0.8783
Epoch 12/50
4949/4949 [==============================] - 4s 728us/step - loss: 0.2839 - a
ccuracy: 0.8775
Epoch 13/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.2834 - a
ccuracy: 0.8772
Epoch 14/50
4949/4949 [==============================] - 4s 728us/step - loss: 0.2836 - a
ccuracy: 0.8780
Epoch 15/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.2832 - a
ccuracy: 0.8775
Epoch 16/50
4949/4949 [==============================] - 4s 727us/step - loss: 0.2829 - a
ccuracy: 0.8786
Epoch 17/50
4949/4949 [==============================] - 4s 725us/step - loss: 0.2828 - a
ccuracy: 0.8779
Epoch 18/50
4949/4949 [==============================] - 4s 723us/step - loss: 0.2822 - a
ccuracy: 0.8785
Epoch 19/50
4949/4949 [==============================] - 4s 725us/step - loss: 0.2828 - a
ccuracy: 0.8782
```

```
Epoch 20/50
4949/4949 [==============================] - 4s 724us/step - loss: 0.2821 - a
ccuracy: 0.8786
Epoch 21/50
4949/4949 [==============================] - 4s 751us/step - loss: 0.2824 - a
ccuracy: 0.8783
Epoch 22/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.2817 - a
ccuracy: 0.8788
Epoch 23/50
4949/4949 [==============================] - 4s 733us/step - loss: 0.2824 - a
ccuracy: 0.8785
Epoch 24/50
4949/4949 [==============================] - 4s 738us/step - loss: 0.2818 - a
ccuracy: 0.8777
Epoch 25/50
4949/4949 [==============================] - 4s 737us/step - loss: 0.2818 - a
ccuracy: 0.8779
Epoch 26/50
4949/4949 [==============================] - 4s 752us/step - loss: 0.2810 - a
ccuracy: 0.8796
Epoch 27/50
4949/4949 [==============================] - 4s 733us/step - loss: 0.2814 - a
ccuracy: 0.8787
Epoch 28/50
4949/4949 [==============================] - 4s 722us/step - loss: 0.2815 - a
ccuracy: 0.8785
Epoch 29/50
4949/4949 [==============================] - 4s 721us/step - loss: 0.2810 - a
ccuracy: 0.8779
Epoch 30/50
4949/4949 [==============================] - 4s 729us/step - loss: 0.2810 - a
ccuracy: 0.8785
Epoch 31/50
4949/4949 [==============================] - 4s 727us/step - loss: 0.2810 - a
ccuracy: 0.8783
Epoch 32/50
4949/4949 [==============================] - 4s 727us/step - loss: 0.2810 - a
ccuracy: 0.8788
Epoch 33/50
4949/4949 [==============================] - 4s 726us/step - loss: 0.2805 - a
ccuracy: 0.8787
Epoch 34/50
4949/4949 [==============================] - 4s 729us/step - loss: 0.2805 - a
ccuracy: 0.8796
Epoch 35/50
4949/4949 [==============================] - 4s 736us/step - loss: 0.2806 - a
ccuracy: 0.8789
Epoch 36/50
4949/4949 [==============================] - 4s 735us/step - loss: 0.2804 - a
ccuracy: 0.8785
Epoch 37/50
4949/4949 [==============================] - 4s 737us/step - loss: 0.2803 - a
ccuracy: 0.8784
Epoch 38/50
4949/4949 [==============================] - 4s 732us/step - loss: 0.2809 - a
ccuracy: 0.8790
```

```
Epoch 39/50
4949/4949 [==============================] - 4s 735us/step - loss: 0.2805 - a
ccuracy: 0.8793
Epoch 40/50
4949/4949 [==============================] - 4s 735us/step - loss: 0.2800 - a
ccuracy: 0.8794
Epoch 41/50
4949/4949 [==============================] - 4s 732us/step - loss: 0.2804 - a
ccuracy: 0.8797
Epoch 42/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.2801 - a
ccuracy: 0.8788
Epoch 43/50
4949/4949 [==============================] - 4s 741us/step - loss: 0.2800 - a
ccuracy: 0.8797
Epoch 44/50
4949/4949 [==============================] - 4s 737us/step - loss: 0.2801 - a
ccuracy: 0.8797
Epoch 45/50
4949/4949 [==============================] - 4s 724us/step - loss: 0.2797 - a
ccuracy: 0.8798
Epoch 46/50
4949/4949 [==============================] - 4s 734us/step - loss: 0.2797 - a
ccuracy: 0.8787
Epoch 47/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.2799 - a
ccuracy: 0.8800
Epoch 48/50
4949/4949 [==============================] - 4s 730us/step - loss: 0.2798 - a
ccuracy: 0.8797
Epoch 49/50
4949/4949 [==============================] - 4s 748us/step - loss: 0.2795 - a
ccuracy: 0.8797
Epoch 50/50
4949/4949 [==============================] - 4s 728us/step - loss: 0.2796 - a
ccuracy: 0.8795
1547/1547 [==============================] - 1s 592us/step - loss: 0.2776 - a
ccuracy: 0.8804
663/663 [==============================] - 0s 589us/step - loss: 0.3010 - acc
uracy: 0.8747
Train Accuracy: 0.880446195602417
Test Accuracy: 0.8746699094772339
```

In [70]:
```
nn_test_pred = nn.predict(sfs_selected_test)
nn_test_pred = np.round(nn_test_pred)

cm = confusion_matrix(y_test, nn_test_pred)

disp = ConfusionMatrixDisplay(confusion_matrix(y_test.values.ravel(), nn_test_p
disp.plot()
plt.show()

print(f'Test Accuracy: {test_accuracy}')
print(f'Classification Report for Test: \n{classification_report(y_test.values
```

```
663/663 [==============================] - 0s 544us/step
```



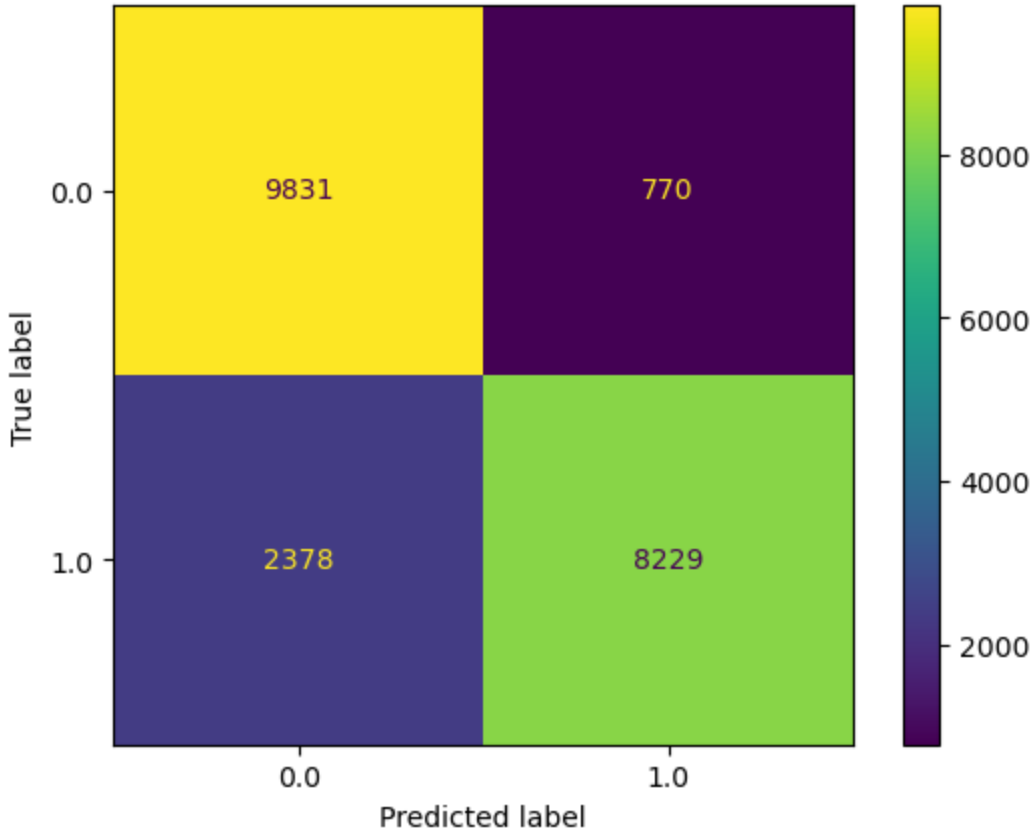

```
Test Accuracy: 0.8746699094772339
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.84      0.93      0.88     10601
         1.0       0.92      0.82      0.87     10607

    accuracy                           0.87     21208
   macro avg       0.88      0.87      0.87     21208
weighted avg       0.88      0.87      0.87     21208
```

Ensemble Model

* This model combines a n number of models into one and decides final prediction through various methods
* This model uses the voting method
    * With hard voting so the predictions are based on majority vote

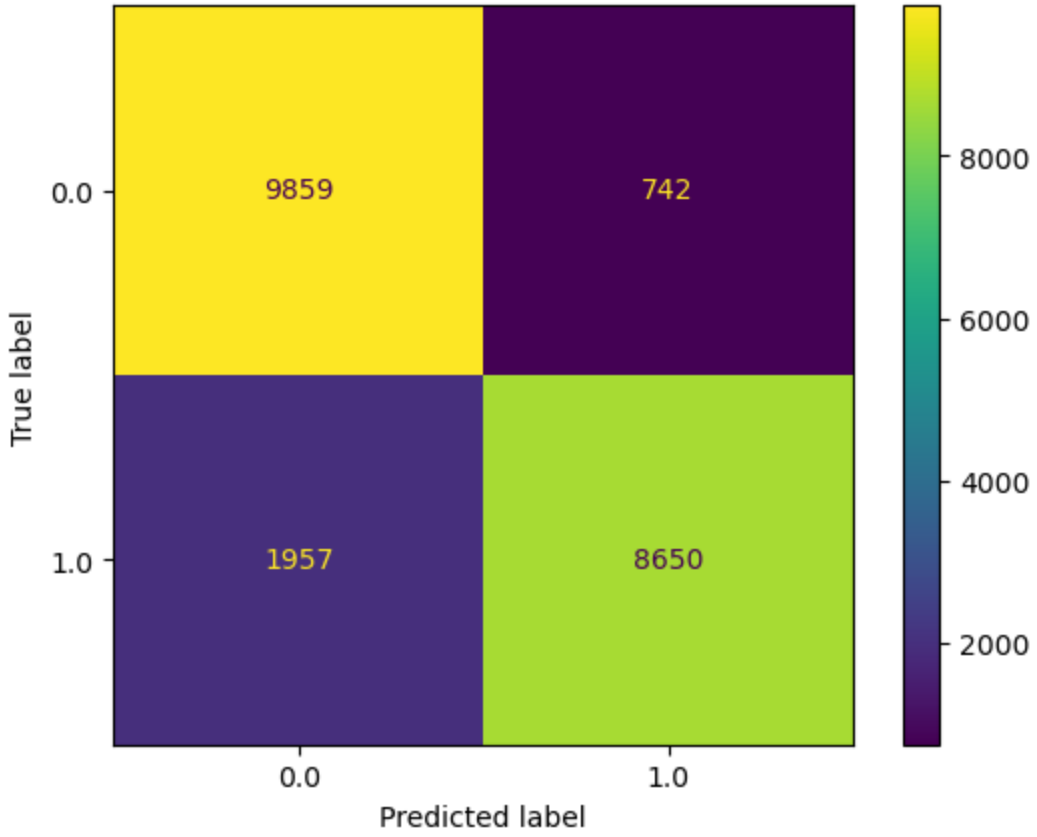
In [66]:
```
model_1 = RandomForestClassifier(criterion = 'entropy',max_features = 'sqrt', r
model_2 = SVC(C = 5, degree = 3, gamma = 'auto', kernel = 'rbf')
model_3 = LogisticRegression(C = 8, l1_ratio = 0, penalty = 'l1', random_state=

ensemble_model = VotingClassifier(estimators = [('rf', model_1), ('svm', model_
                                  voting = 'hard',
                                  n_jobs = -1,
                                  verbose = True)
ensemble_model.fit(X_selected_train, y_train.values.ravel())
ensemble_pred_train = ensemble_model.predict(X_selected_train)
ensemble_pred_test = ensemble_model.predict(X_selected_test)

ensemble_train_score = accuracy_score(y_train.values.ravel(), ensemble_pred_tra
ensemble_test_score = accuracy_score(y_test.values.ravel(), ensemble_pred_test)

print(f'Train Accuracy: {ensemble_train_score}')
print(f'Classification Report for Train: \n{classification_report(y_train.value

print(f'Test Accuracy: {ensemble_test_score}')
print(f'Classification Report for Test: \n{classification_report(y_test.values

disp = ConfusionMatrixDisplay(confusion_matrix(y_test.values.ravel(), ensemble_
disp.plot()
plt.show()
```

```
Train Accuracy: 0.8555290598981489
Classification Report for Train:
              precision    recall  f1-score   support

         0.0       0.81      0.93      0.87     24745
         1.0       0.92      0.78      0.84     24739

    accuracy                           0.86     49484
   macro avg       0.86      0.86      0.85     49484
weighted avg       0.86      0.86      0.85     49484


Test Accuracy: 0.8515654470011317
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.81      0.93      0.86     10601
         1.0       0.91      0.78      0.84     10607

    accuracy                           0.85     21208
   macro avg       0.86      0.85      0.85     21208
weighted avg       0.86      0.85      0.85     21208
```


9831
770
2378
8229
0.0
1.0
True label
0.0
1.0
Predicted label
8000
6000
4000
2000

In [67]:
```
model_1 = RandomForestClassifier(criterion = 'entropy',max_features = 'sqrt', r
model_2 = SVC(C = 5, degree = 3, gamma = 'auto', kernel = 'rbf')
model_3 = LogisticRegression(C = 8, l1_ratio = 0, penalty = 'l1', random_state=

ensemble_model = VotingClassifier(estimators = [('rf', model_1), ('svm', model_
                                  voting = 'hard',
                                  n_jobs = -1,
                                  verbose = True)
ensemble_model.fit(sfs_selected_train, y_train.values.ravel())
ensemble_pred_train = ensemble_model.predict(sfs_selected_train)
ensemble_pred_test = ensemble_model.predict(sfs_selected_test)

ensemble_train_score = accuracy_score(y_train.values.ravel(), ensemble_pred_tra
ensemble_test_score = accuracy_score(y_test.values.ravel(), ensemble_pred_test)

print(f'Train Accuracy: {ensemble_train_score}')
print(f'Classification Report for Train: \n{classification_report(y_train.value

print(f'Test Accuracy: {ensemble_test_score}')
print(f'Classification Report for Test: \n{classification_report(y_test.values

disp = ConfusionMatrixDisplay(confusion_matrix(y_test.values.ravel(), ensemble_
disp.plot()
plt.show()
```

```
Train Accuracy: 0.8776574246221001
Classification Report for Train:
              precision    recall  f1-score   support

         0.0       0.84      0.94      0.88     24745
         1.0       0.93      0.82      0.87     24739

    accuracy                           0.88     49484
   macro avg       0.88      0.88      0.88     49484
weighted avg       0.88      0.88      0.88     49484


Test Accuracy: 0.872736703130894
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.83      0.93      0.88     10601
         1.0       0.92      0.82      0.87     10607

    accuracy                           0.87     21208
   macro avg       0.88      0.87      0.87     21208
weighted avg       0.88      0.87      0.87     21208
```



Once Again The Wrapper Method Seems To Offer The Best Results