Training Models

In this section, I will train models to predict whether is person's income yearly 50k+ or not depending on the data. I tried every model as hand-tuned or with grid search I reported all results after every model. You can see models and frameworks/libraries in the table below.

Model	Library - Framework
Logistic Regression	scikit-learn
Support Vector Machine	scikit-learn
K-Nearest Neighbors	scikit-learn
Decision Tree	scikit-learn
Random Forest	scikit-learn
Ada Boost	scikit-learn
Gradient Boosting	scikit-learn
eXtreme Gradient Boosting	XGBoost
Neural Network	TensorFlow

Notebook Structure

- **Load Prepare Data**
- **Logistic Regression**
- **Support Vector Machine**
- **Tree Based Models**
  - **Decision Tree Regressor**
  - **Random Forest Regressor**
  - **AdaBoost Regressor**
  - **Gradient Boosting Regressor**
  - **eXtreme Gradient Boosting**
- **Neural Network**
- **Deploying Models**

Load & Prepare Data

Firstly, I split data into two (training and test sets). In models, data will be used as both scaled and unscaled. To scale data I used a standard scaler which calculates every feature Z score with training data set statistics.

In [1]:
```
import warnings
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from src.utils import classification_scores, find_threshold

warnings.filterwarnings('ignore')
%load_ext autoreload
%autoreload 2
pd.set_option('float_format', '{:f}'.format)

processed_data_dir = Path('../data/processed/')
file_name = 'adult.csv'
file_path = processed_data_dir / file_name
models_trained_dir = Path('../models/trained/')
```

In [2]:
```
df = pd.read_csv(file_path)
df.head()
```

Out[2]:

	income	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	0	1	2	...	91
0	0	90	77053	9	0	4356	40	0.000000	0.000000	1.000000	...	0.000000
1	0	82	132870	9	0	4356	18	0.000000	0.000000	0.000000	...	0.000000
2	0	66	186061	10	0	4356	40	0.000000	0.000000	1.000000	...	0.000000
3	0	54	140359	4	0	3900	40	0.000000	0.000000	0.000000	...	0.000000
4	0	41	264663	10	0	3900	40	0.000000	0.000000	0.000000	...	0.000000

5 rows × 108 columns

In [3]:
```
X = df.drop('income', axis=1)
y = df['income']
```

In [4]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=13)
```

In [5]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

Training Models

Our data is unbalanced and have a large amount of '<=50k' labels compared with '>50k'. When training models consider a situation like our company has a kind of expensive product that buyers generally have +50k income yearly. So our company wants us which group of people should see our advertisements (so we can lower advertising money consumption) because of that, finding possible clients is more important than showing advertisements to non-buyers but still we don't want to waste our money. So we want to fit model that predict both class with close performance scores.

Note: Because of the data unbalanced I put more attention on confusion matrix and recall scores

In [6]:
```
from sklearn.model_selection import GridSearchCV
```

In [7]:
```
def grid_fit_score(model, param_grid, X_train, X_test, y_train=y_train,
                   y_test=y_test):
    '''Helper function to apply grid search and evalute scores.'''
    grid_model = GridSearchCV(model, param_grid, n_jobs=-1, verbose=3)
    grid_model.fit(X_train, y_train)
    print(f'Best Params are: \n\t{grid_model.best_params_}')
    scores = classification_scores(grid_model.best_estimator_, X_test, y_test)

    return grid_model, scores
```

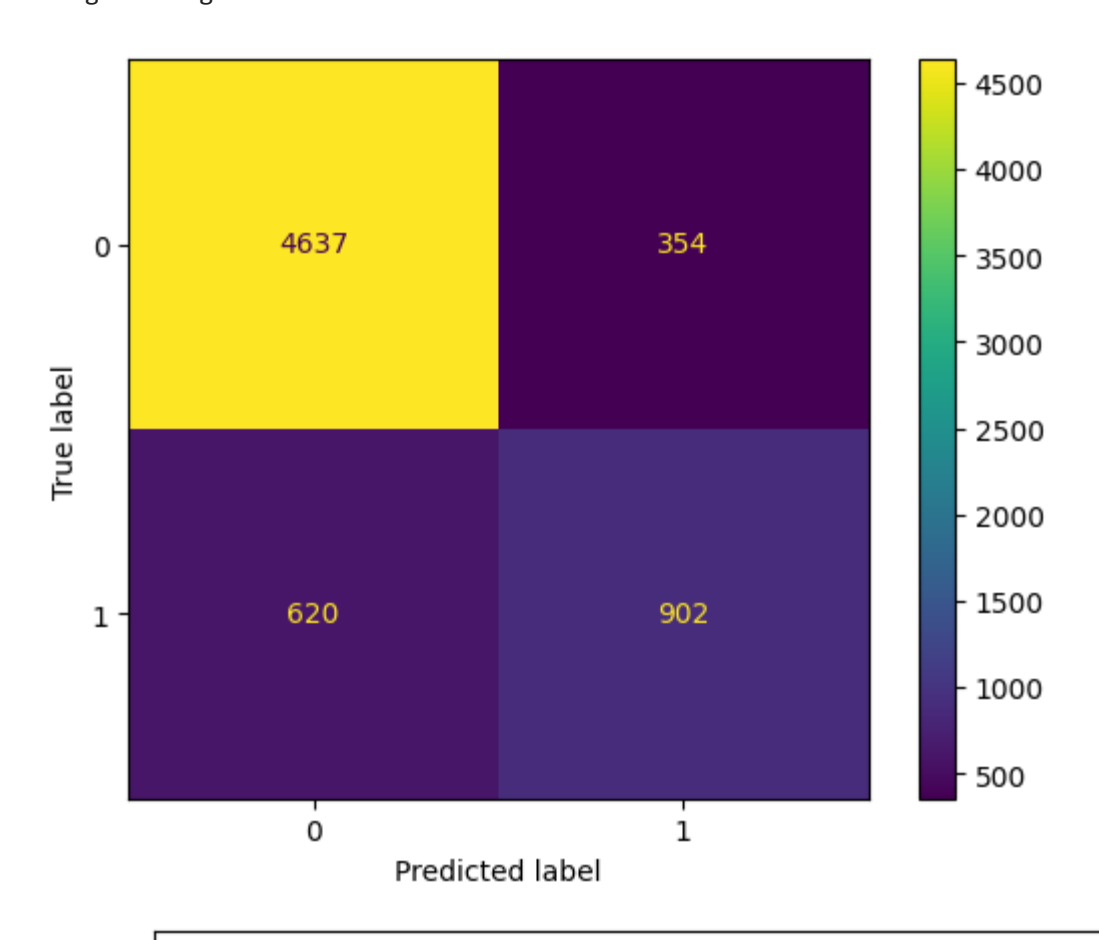
Logistic Regression

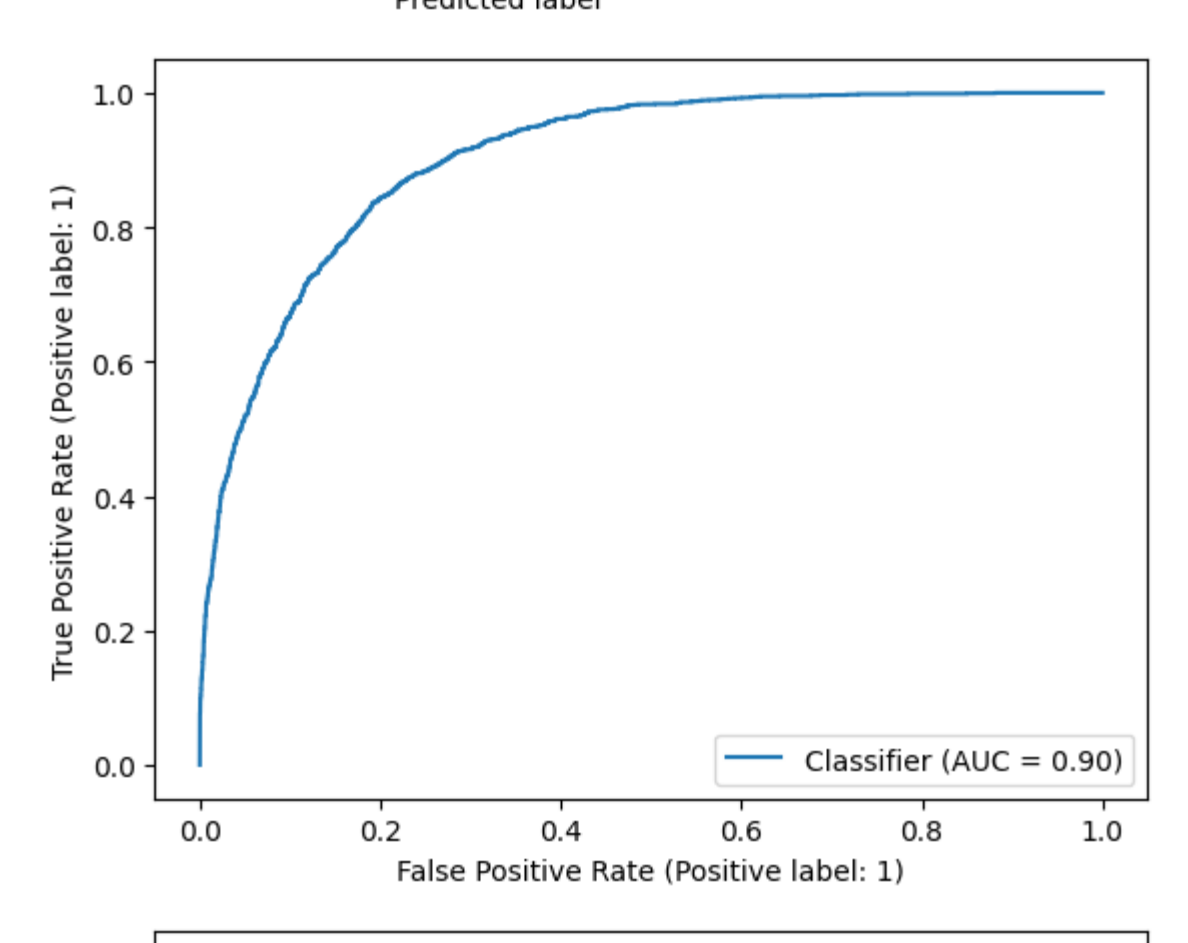
Description Logistic regression grid search with different hyperparameters.

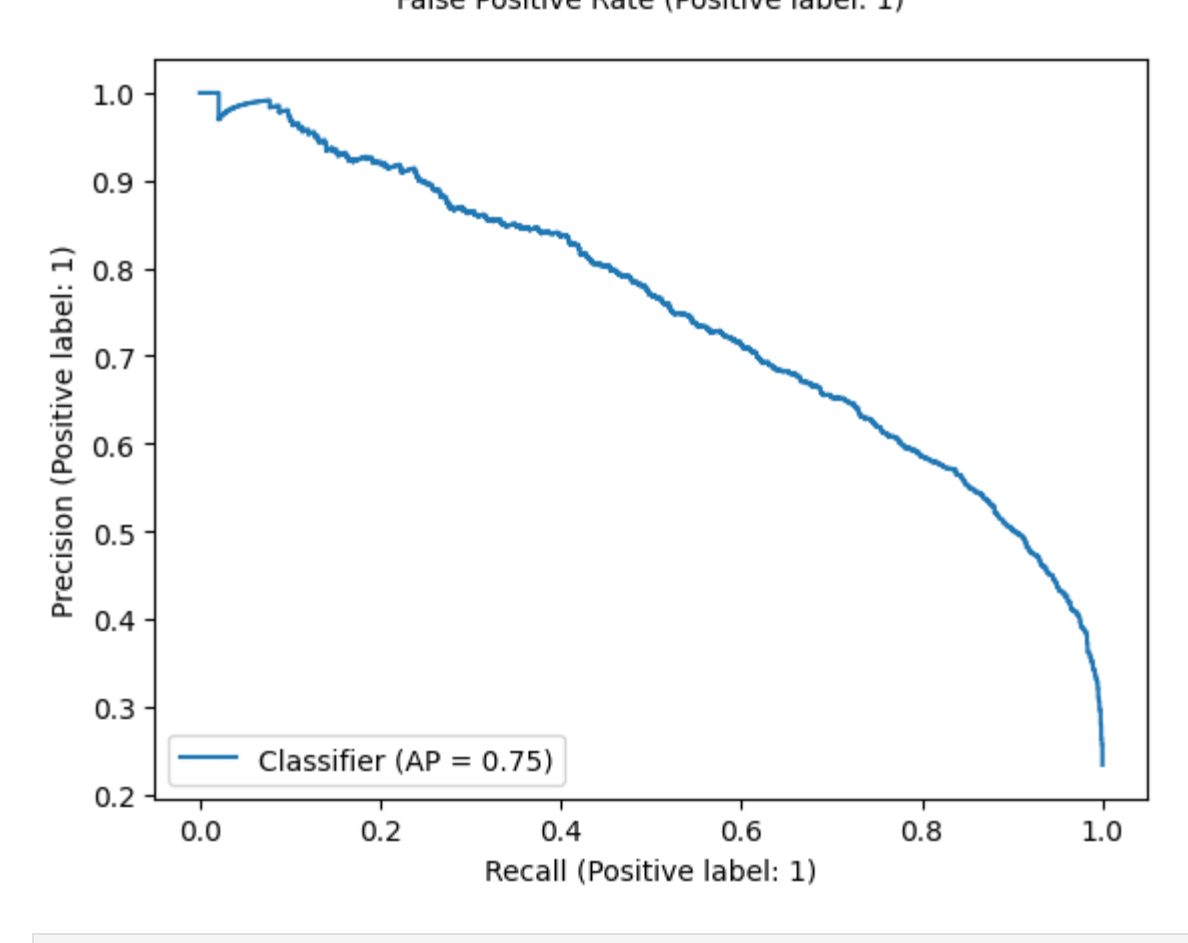
Results Because our data is unbalanced and have a large amount of <=50k' labels compared with '>50k', you can see below our model is well suited to the '<=50k'(0) label and has better performance on it, but poorer performance on '>50k'(1). By looking at different thresholds from the ROC plot we may improve the macro avg recall score to get closer to our goal. We put equal importance on predicting class so, we get an average of (1-fpr)+tpr and find a threshold to maximize that value. After getting the threshold value I get an improvement on the class 1 recall score decrease on the class 0 recall score but in total our macro avg recall score improved and recall scores became very closer to each other. And performance is quite good, considering balanced predictions accuracy.

In [8]:
```
from sklearn.linear_model import LogisticRegression

log_model = LogisticRegression(max_iter=int(1e+5))

penalty = ['l1', 'l2', 'elasticnet']
l1_ratio = np.linspace(0, 1, 4)
C = np.logspace(-1, 2, 5)
param_grid = {'penalty': penalty, 'l1_ratio': l1_ratio, 'C': C,
              'solver': ['saga']}
log_grid, log_scores = grid_fit_score(log_model, param_grid, X_train_scaled,
                                      X_test_scaled)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
Best Params are:
        {'C': 0.1, 'l1_ratio': 0.0, 'penalty': 'l1', 'solver': 'saga'}
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      4991
           1       0.72      0.59      0.65      1522

    accuracy                           0.85      6513
   macro avg       0.80      0.76      0.78      6513
weighted avg       0.84      0.85      0.85      6513
```

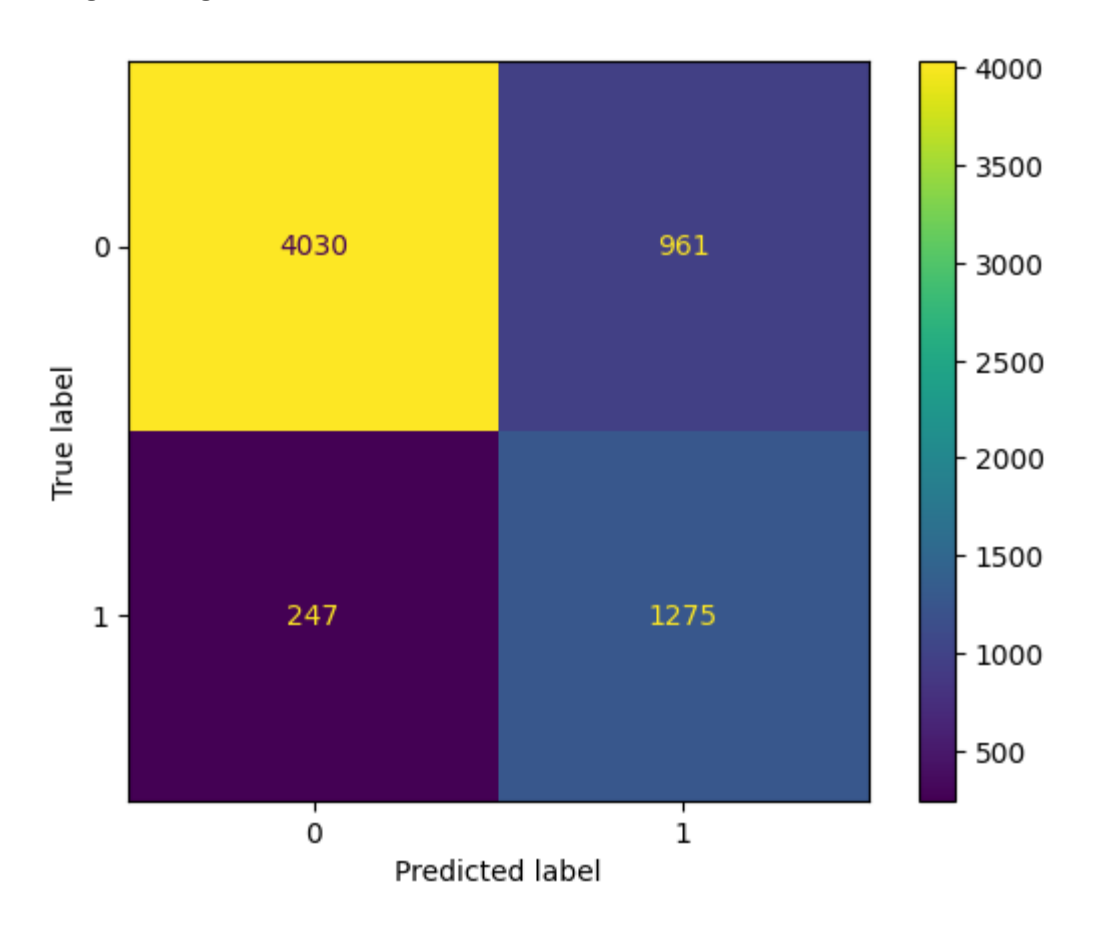






In [9]:
```
log_best_threshold = find_threshold(log_scores['roc_curve'])
log_scores = classification_scores(log_grid, X_test_scaled, y_test, log_best_threshold,
                                   plots=False)
```

```
        The best threshold for balancing recall is: 0.25186553573031456

              precision    recall  f1-score   support

           0       0.94      0.81      0.87      4991
           1       0.57      0.84      0.68      1522

    accuracy                           0.81      6513
   macro avg       0.76      0.82      0.77      6513
weighted avg       0.86      0.81      0.83      6513
```



K-Nearest Neighbors

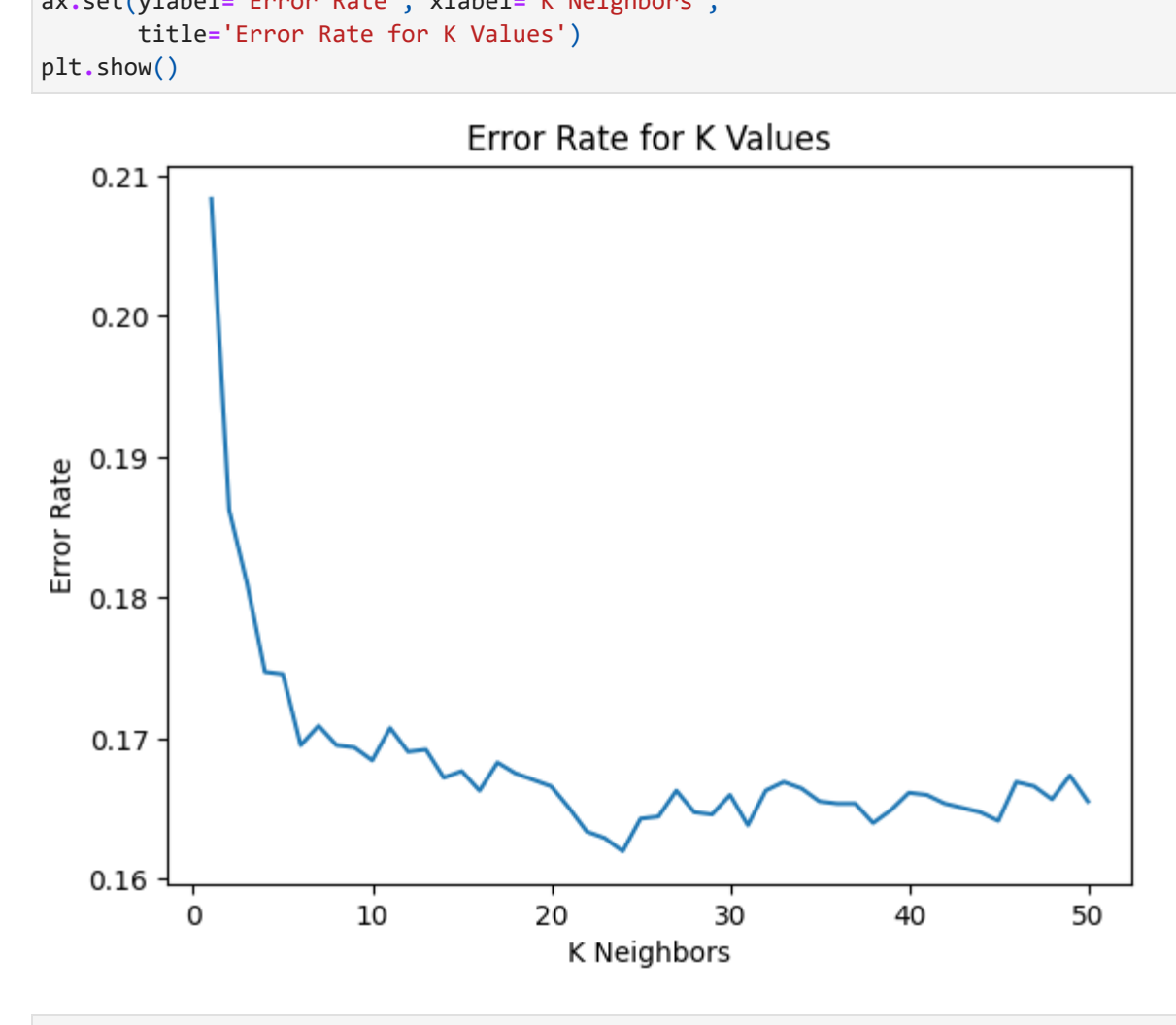
Description I implemented K-Nearest Neighbors with different neighbors values considering model complexity vs performance I decided to nearest neighbors from the plot.

Results For the accuracy error, I decided to train a model with 15 nearest neighbors. After that, I calculated the scores. We have the same problem like logistic regression. After getting the best threshold performance improved just like logistic regression did.

In [10]:
```
from sklearn.neighbors import KNeighborsClassifier

test_error_rates = []
k_values = list(range(1, 51))
for n in k_values:
    print(f'\rFitting KNN model with {n} neighbors.' + (' ' * 3), end='')
    knn_model = KNeighborsClassifier(n_neighbors=n)
    knn_model.fit(X_train_scaled, y_train)
    acc = knn_model.score(X_test_scaled, y_test)
    test_error_rates.append(acc)
```

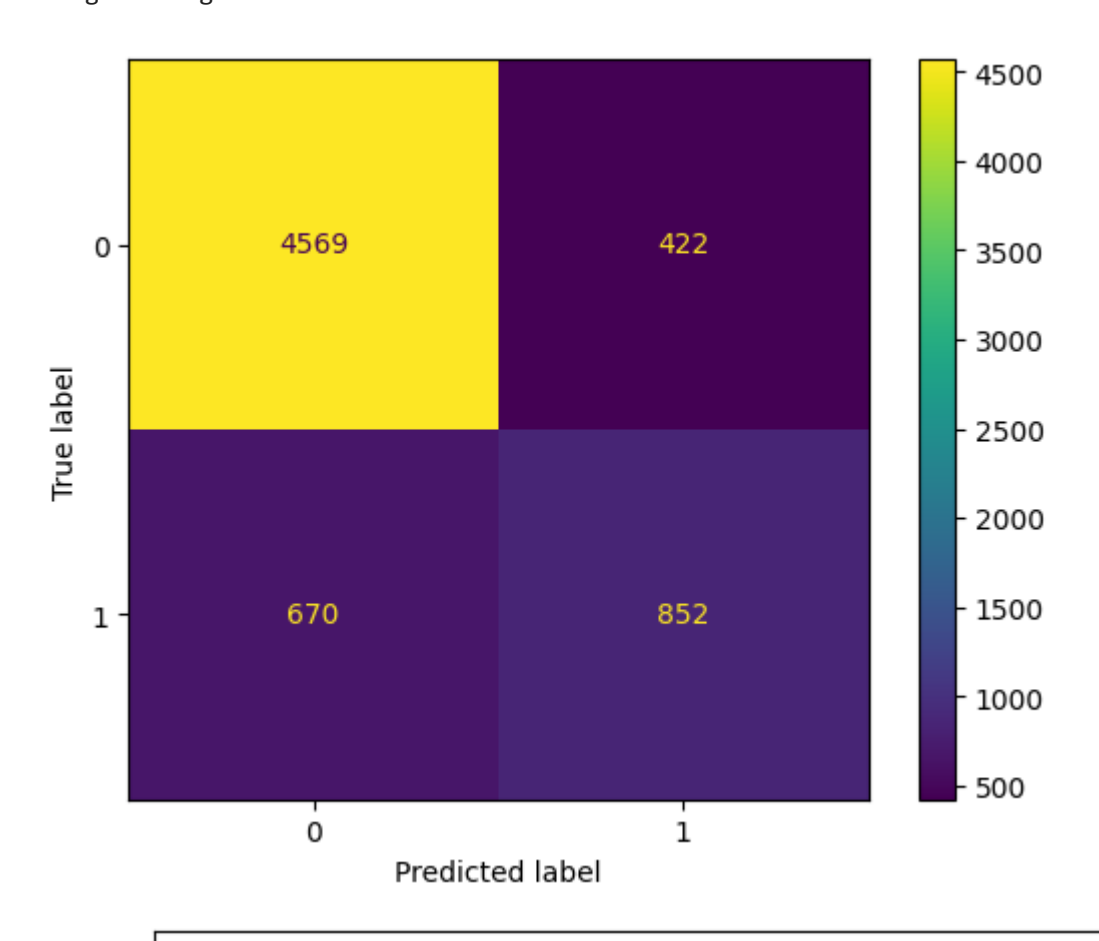
```
Fitting KNN model with 50 neighbors.
```

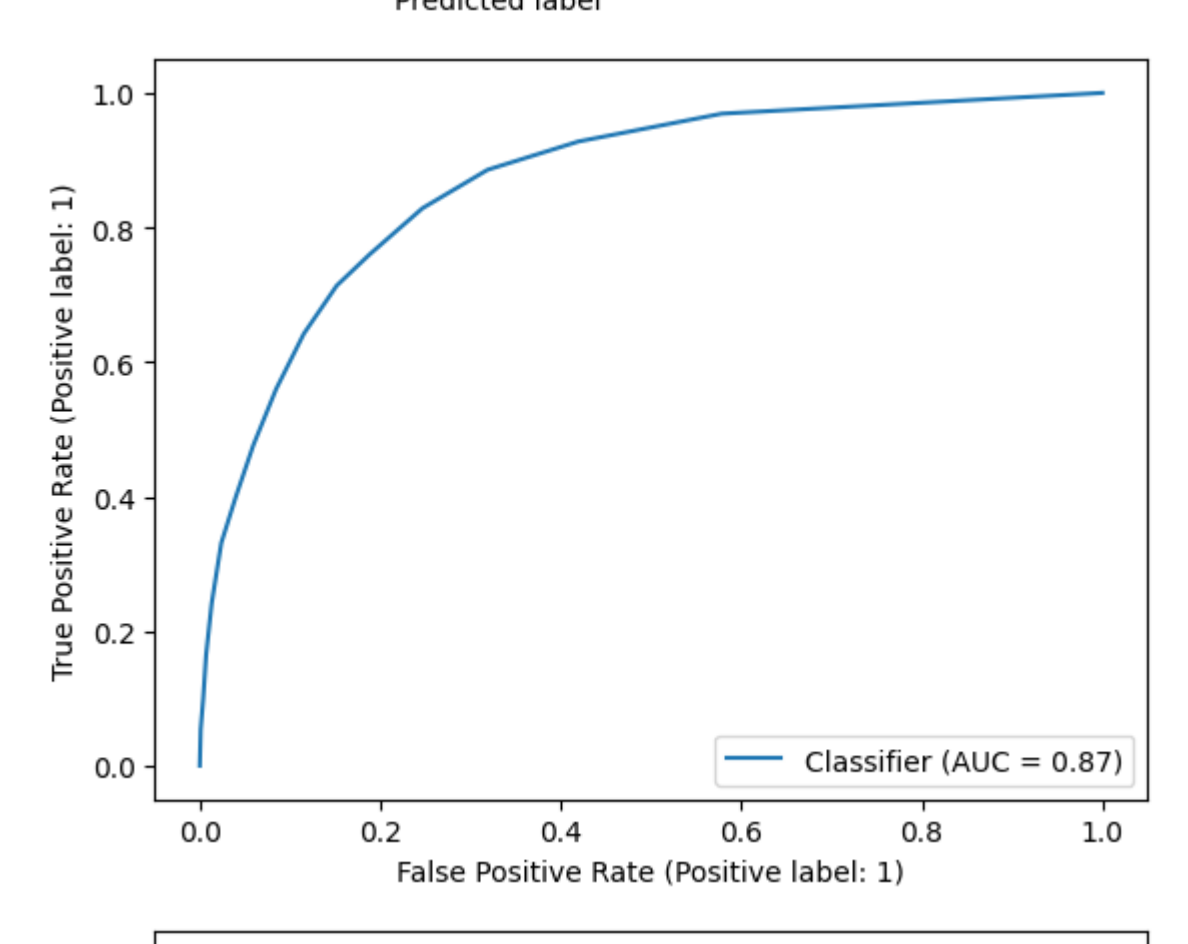
In [11]:
```
fig, ax = plt.subplots()
ax.plot(k_values, 1 - np.array(test_error_rates))
ax.set(ylabel='Error Rate', xlabel='K Neighbors',
       title='Error Rate for K Values')
plt.show()
```

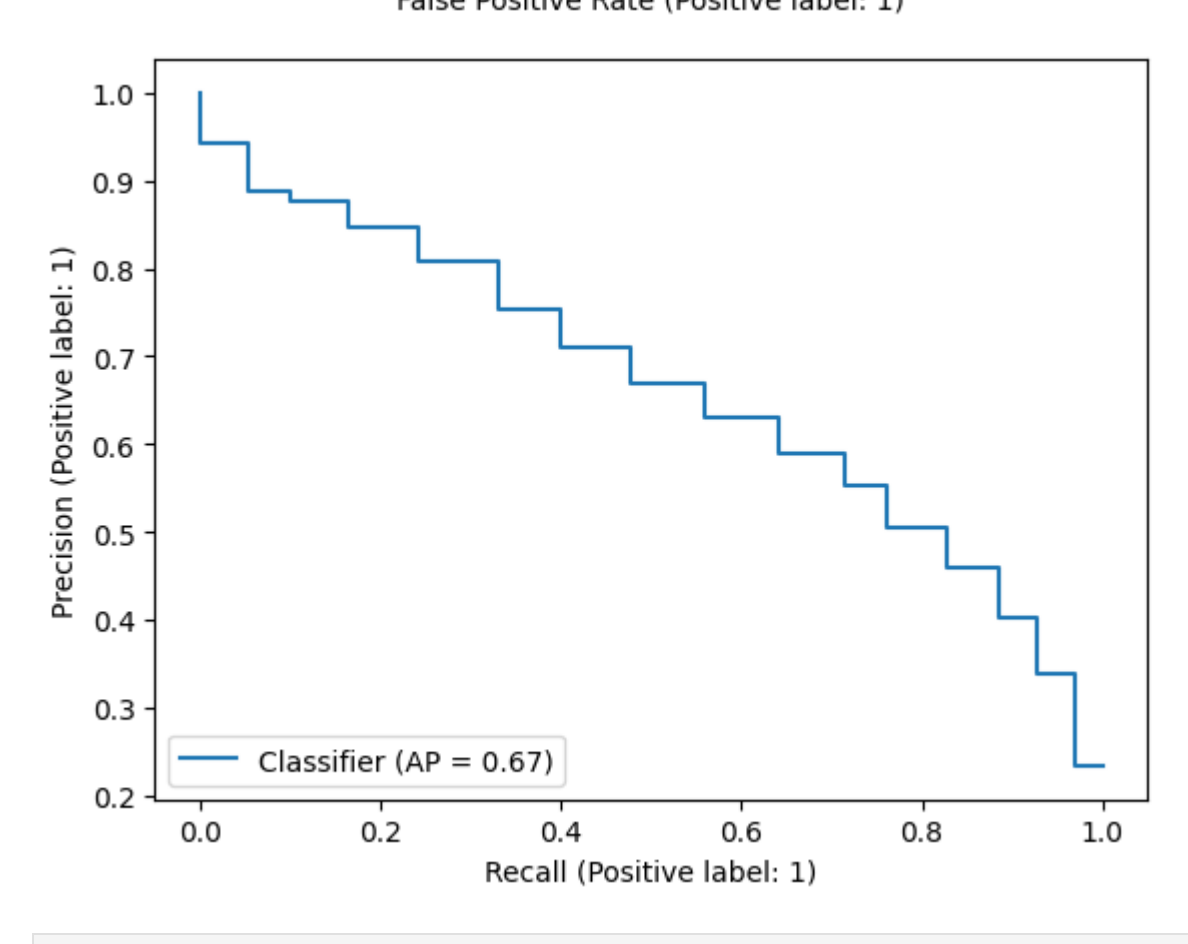


In [12]:
```
best_nn = 15
knn_model = KNeighborsClassifier(n_neighbors=best_nn)
knn_model.fit(X_train_scaled, y_train)
knn_scores = classification_scores(knn_model, X_test_scaled, y_test)
```

```
              precision    recall  f1-score   support

           0       0.87      0.92      0.89      4991
           1       0.67      0.56      0.61      1522

    accuracy                           0.83      6513
   macro avg       0.77      0.74      0.75      6513
weighted avg       0.82      0.83      0.83      6513
```

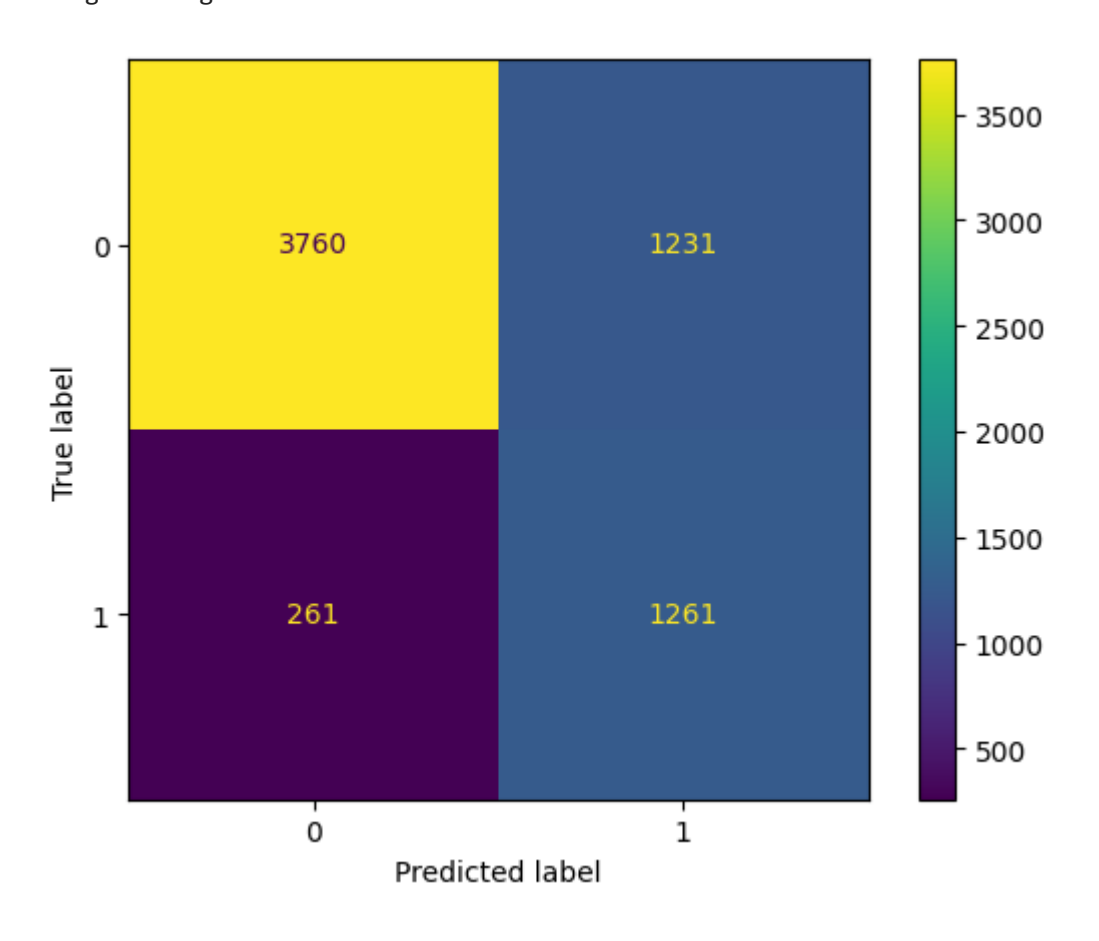






In [13]:
```
knn_best_threshold = find_threshold(knn_scores['roc_curve'])
knn_scores = classification_scores(knn_model, X_test_scaled, y_test, knn_best_threshold,
                                   plots=False)
```

```
        The best threshold for balancing recall is: 0.26666666666666666

              precision    recall  f1-score   support

           0       0.94      0.75      0.83      4991
           1       0.51      0.83      0.63      1522

    accuracy                           0.77      6513
   macro avg       0.72      0.79      0.73      6513
weighted avg       0.83      0.77      0.79      6513
```



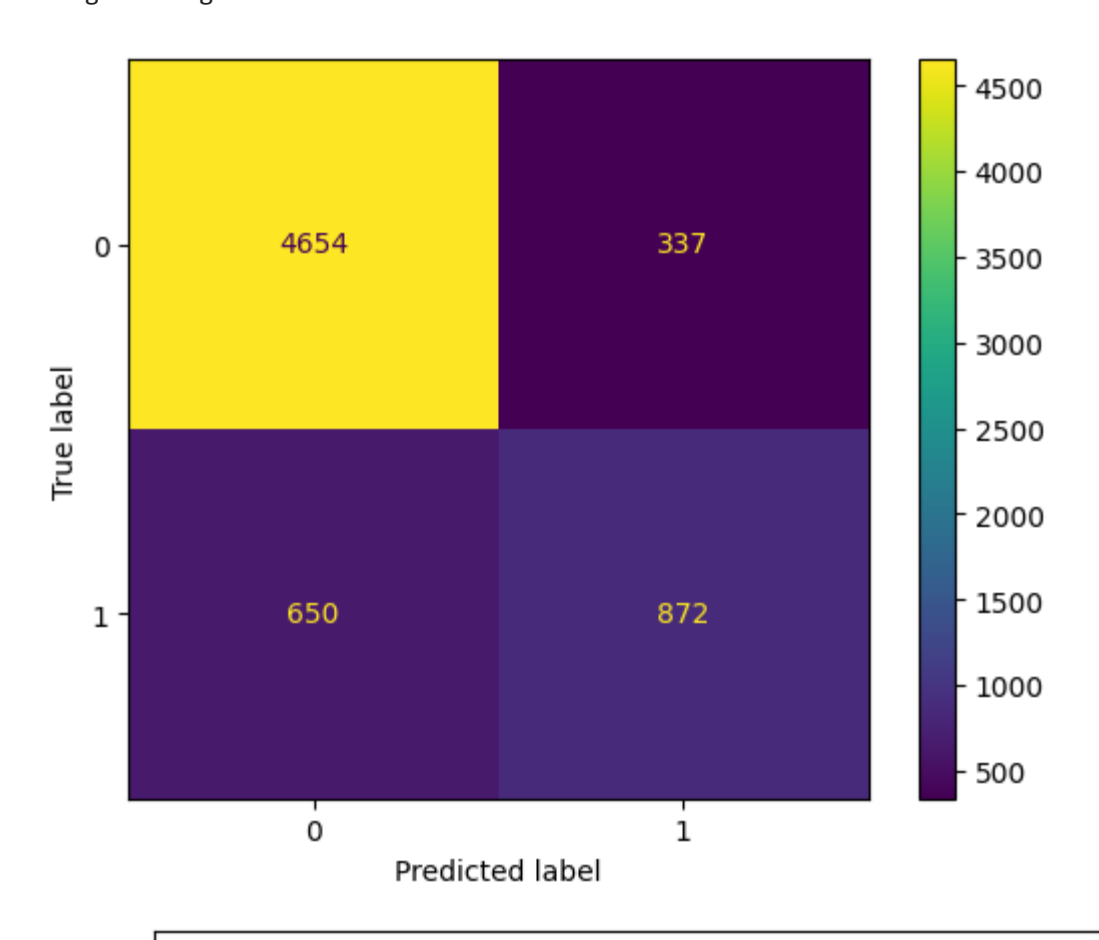
Support Vector Machine

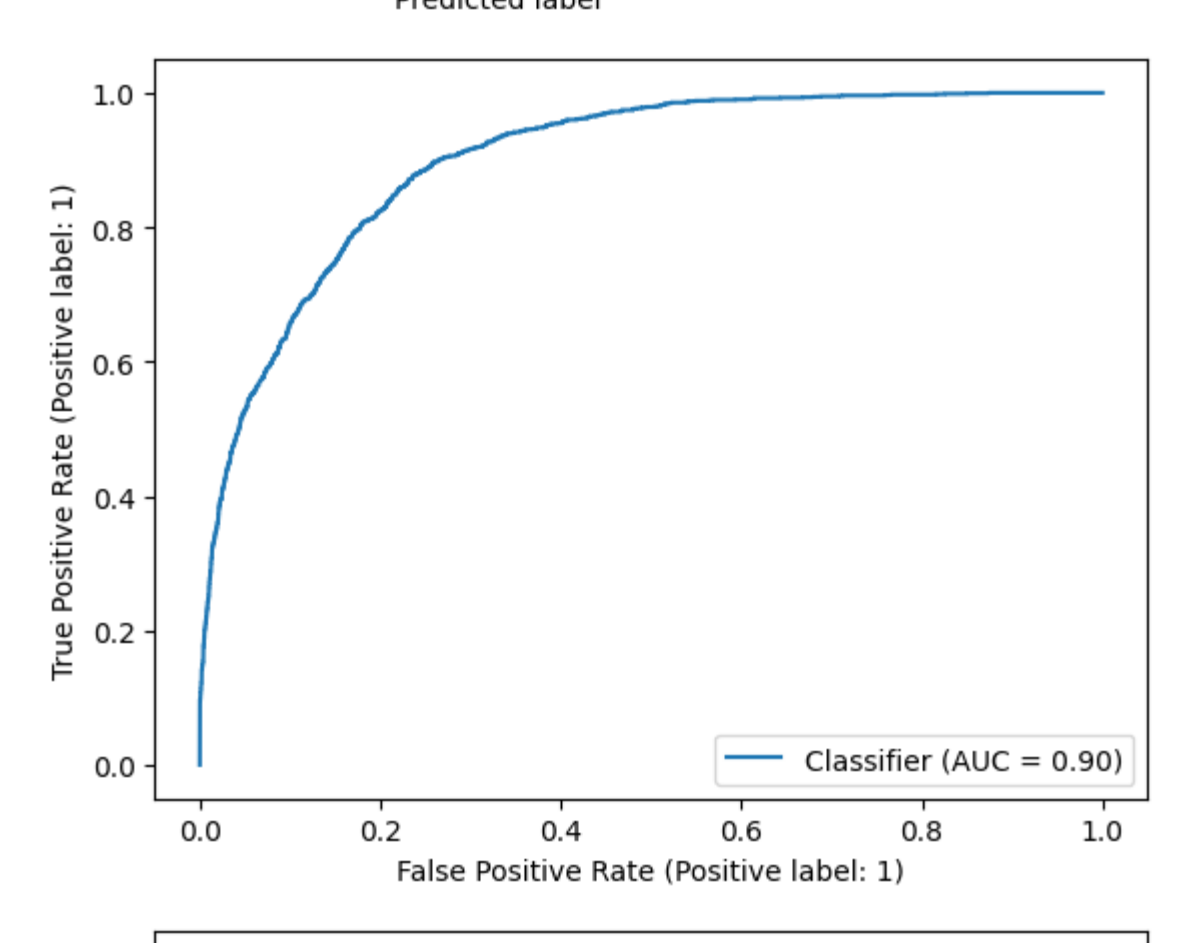
Description Support Vector Classifier grid search with different hyperparameters.

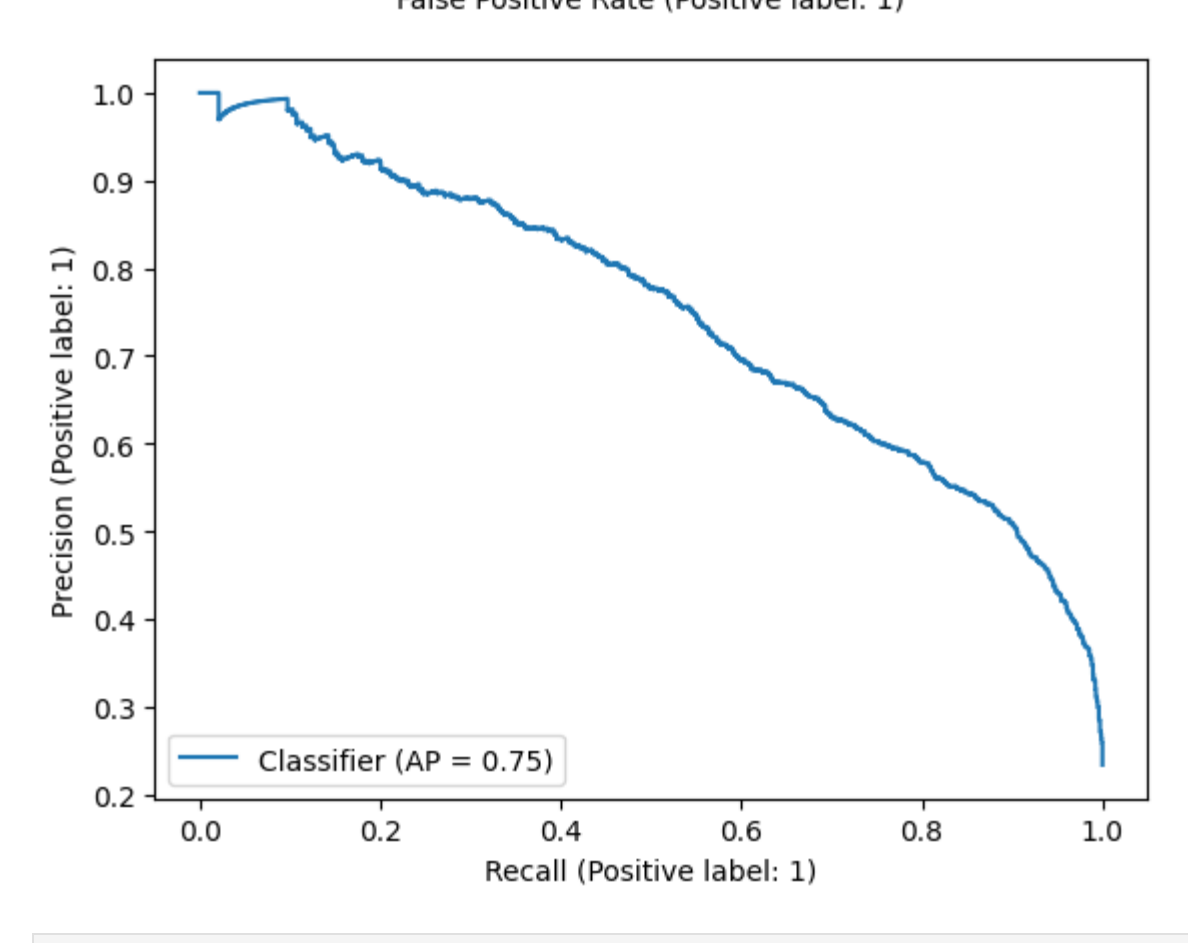
Results Performance is just like the others. After getting best threshold even though we get better avg macro recall score, performance for the 0 class decreased more than expected and results became unbalanced.

In [14]:
```
from sklearn.svm import SVC

svc_model = SVC(probability=True)
grid_params = {'C': [0.1, 1, 10], 'kernel': ['linear']}
svc_grid, svc_scores = grid_fit_score(svc_model, grid_params, X_train_scaled,
                                      X_test_scaled)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Best Params are:
        {'C': 1, 'kernel': 'linear'}
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      4991
           1       0.72      0.57      0.64      1522

    accuracy                           0.85      6513
   macro avg       0.80      0.75      0.77      6513
weighted avg       0.84      0.85      0.84      6513
```

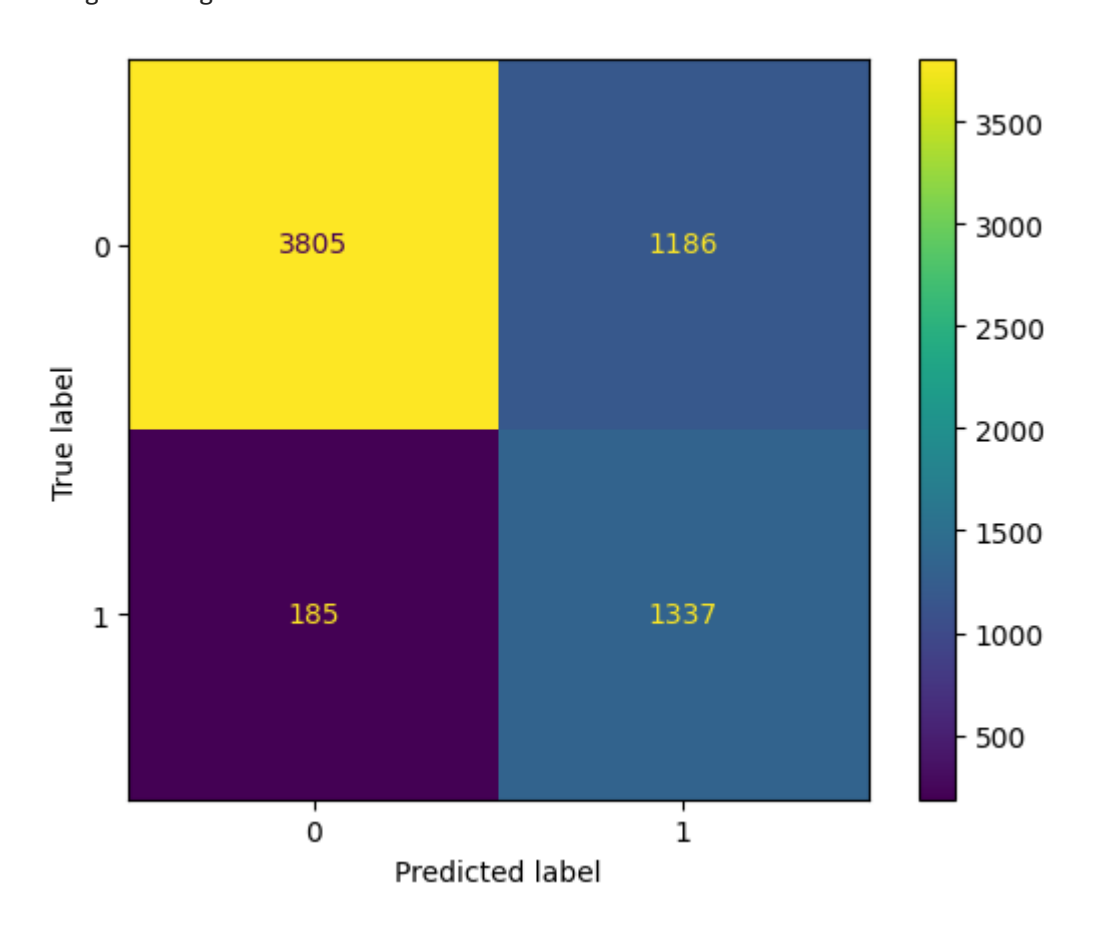






In [15]:
```
svc_best_threshold = find_threshold(svc_scores['roc_curve'])
svc_scores = classification_scores(svc_grid, X_test_scaled, y_test,
                                   svc_best_threshold,
                                   plots=False)
```

```
        The best threshold for balancing recall is: 0.19264272685801298

              precision    recall  f1-score   support

           0       0.95      0.76      0.85      4991
           1       0.53      0.88      0.66      1522

    accuracy                           0.79      6513
   macro avg       0.74      0.82      0.75      6513
weighted avg       0.85      0.79      0.80      6513
```



Tree Based Models

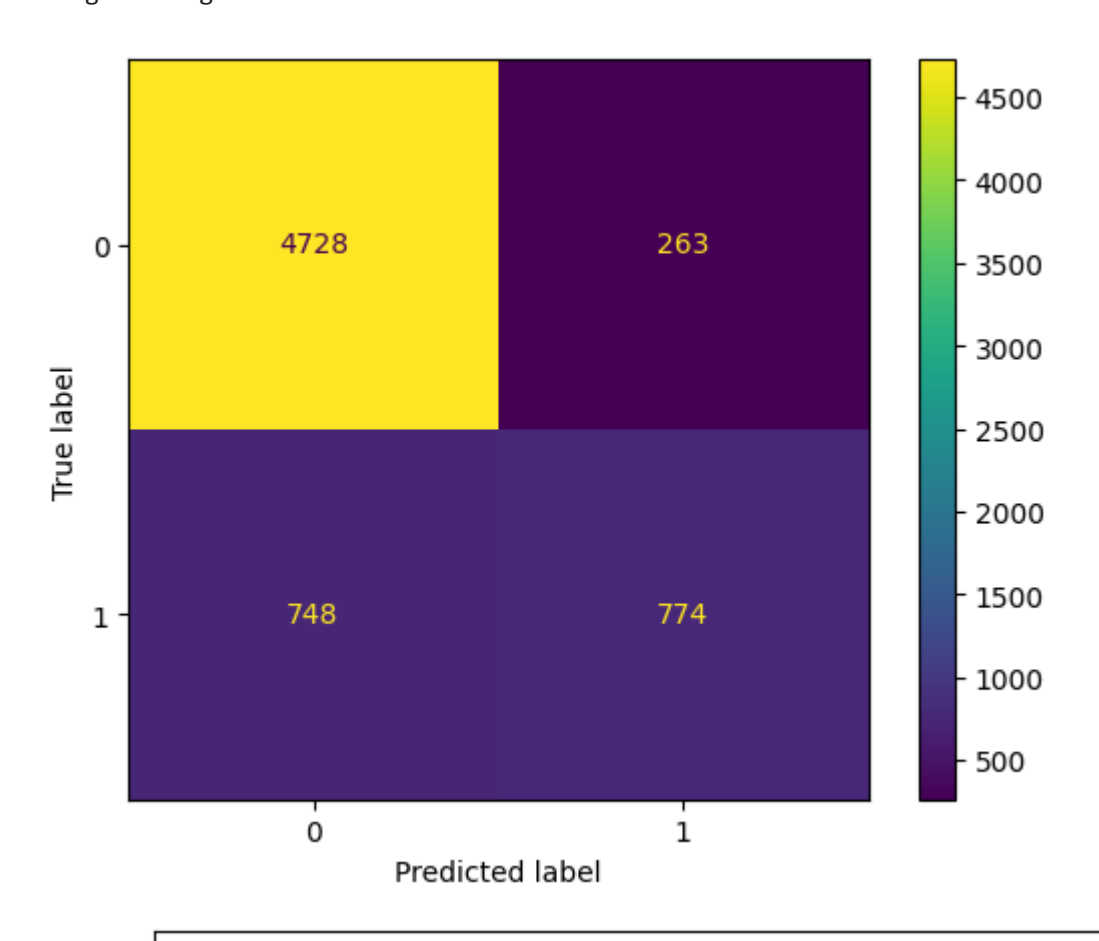
Decision Tree

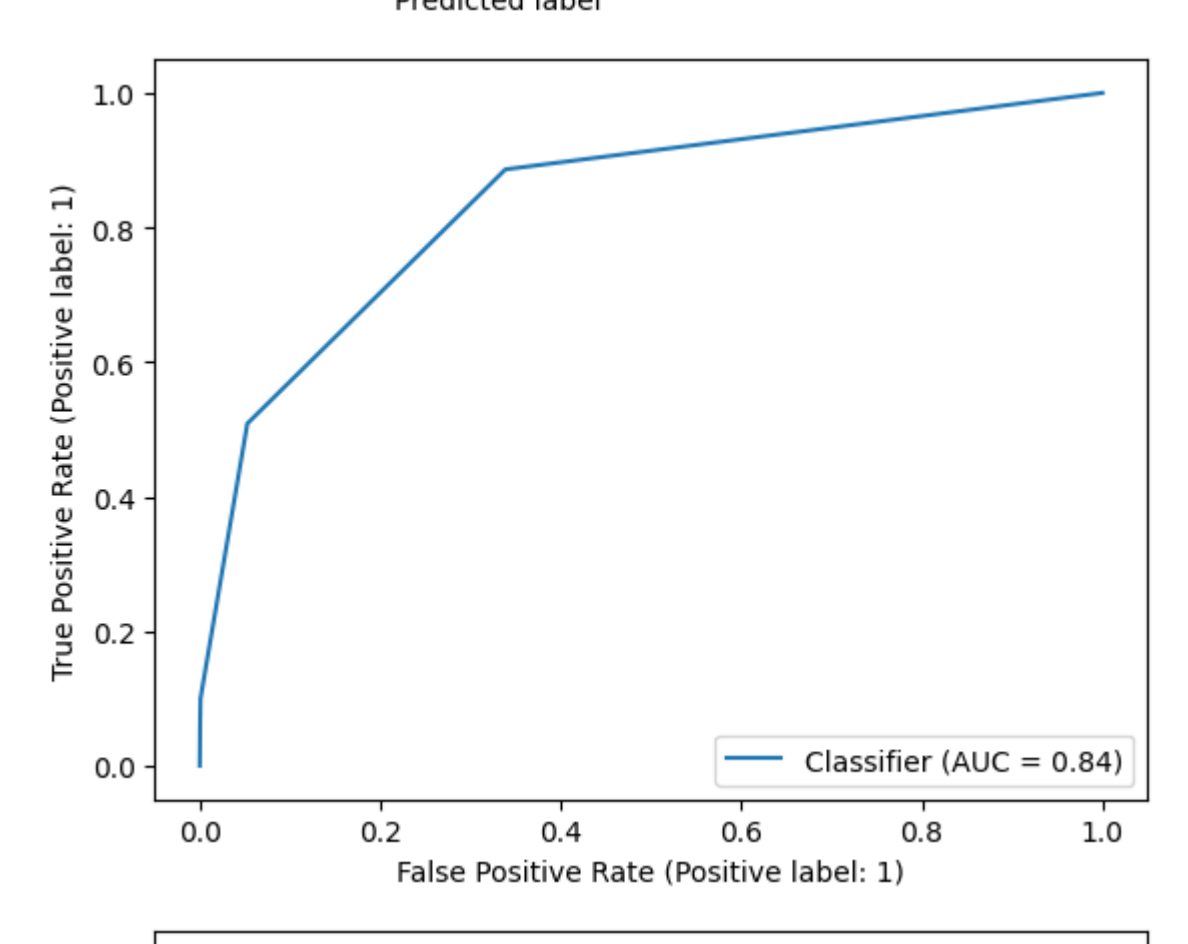
Description Decision Tree Regressor and grid search with different hyperparameters.

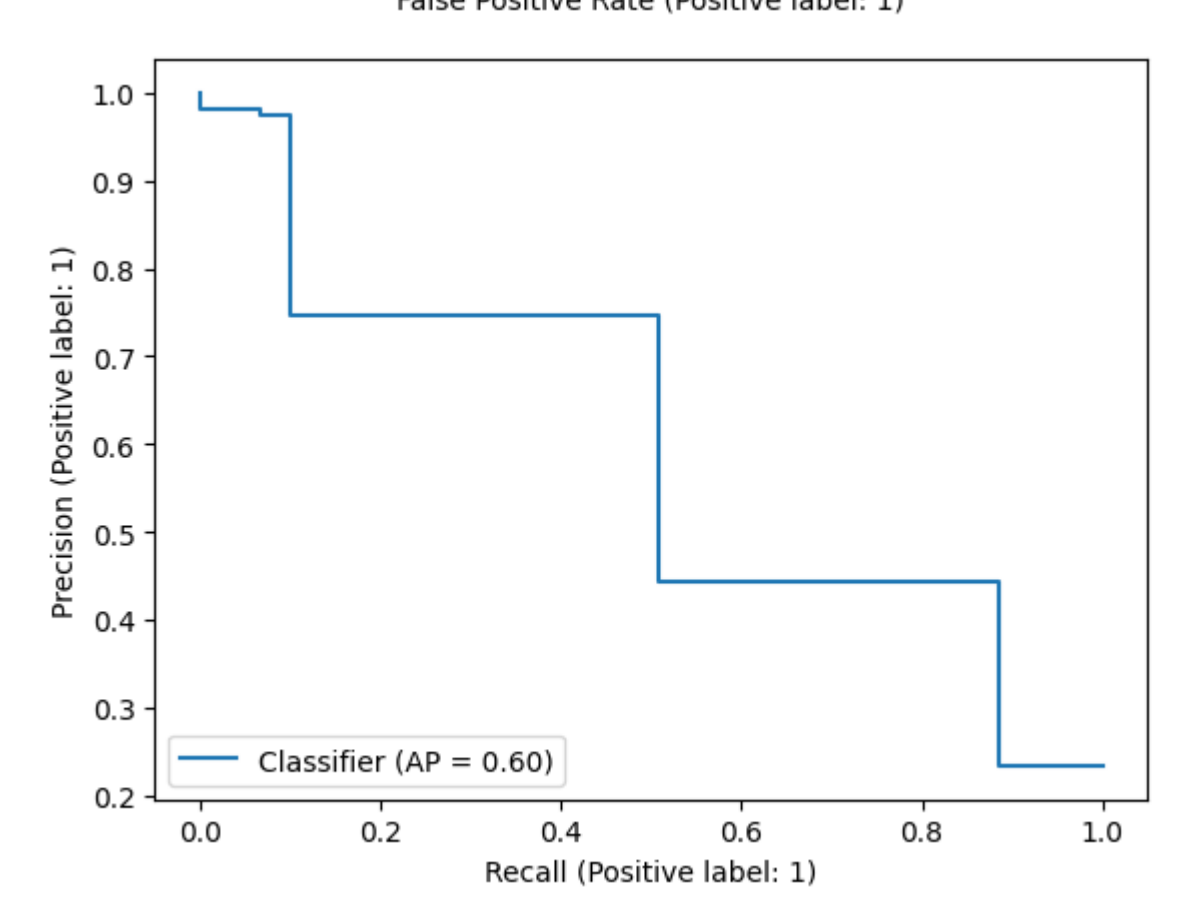
Results We have poor performance in the 1 class. For the 0 class performance is just like the others. After getting best threshold even though we get the best avg macro recall score, performance for the 0 class decreased more than expected and results became unbalanced.

In [16]:
```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier()
param_grid = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [5, 50, 100, 300],
    'min_samples_split': [3, 10],
    'min_impurity_decrease': np.logspace(-2, 0, 10)
}
tree_grid, tree_scores = grid_fit_score(tree_model, param_grid, X_train,
                                        X_test)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
Best Params are:
        {'criterion': 'gini', 'max_depth': 5, 'min_impurity_decrease': 0.01, 'min_samples_split': 3}
              precision    recall  f1-score   support

           0       0.86      0.95      0.90      4991
           1       0.75      0.51      0.60      1522

    accuracy                           0.84      6513
   macro avg       0.80      0.73      0.75      6513
weighted avg       0.84      0.84      0.83      6513
```

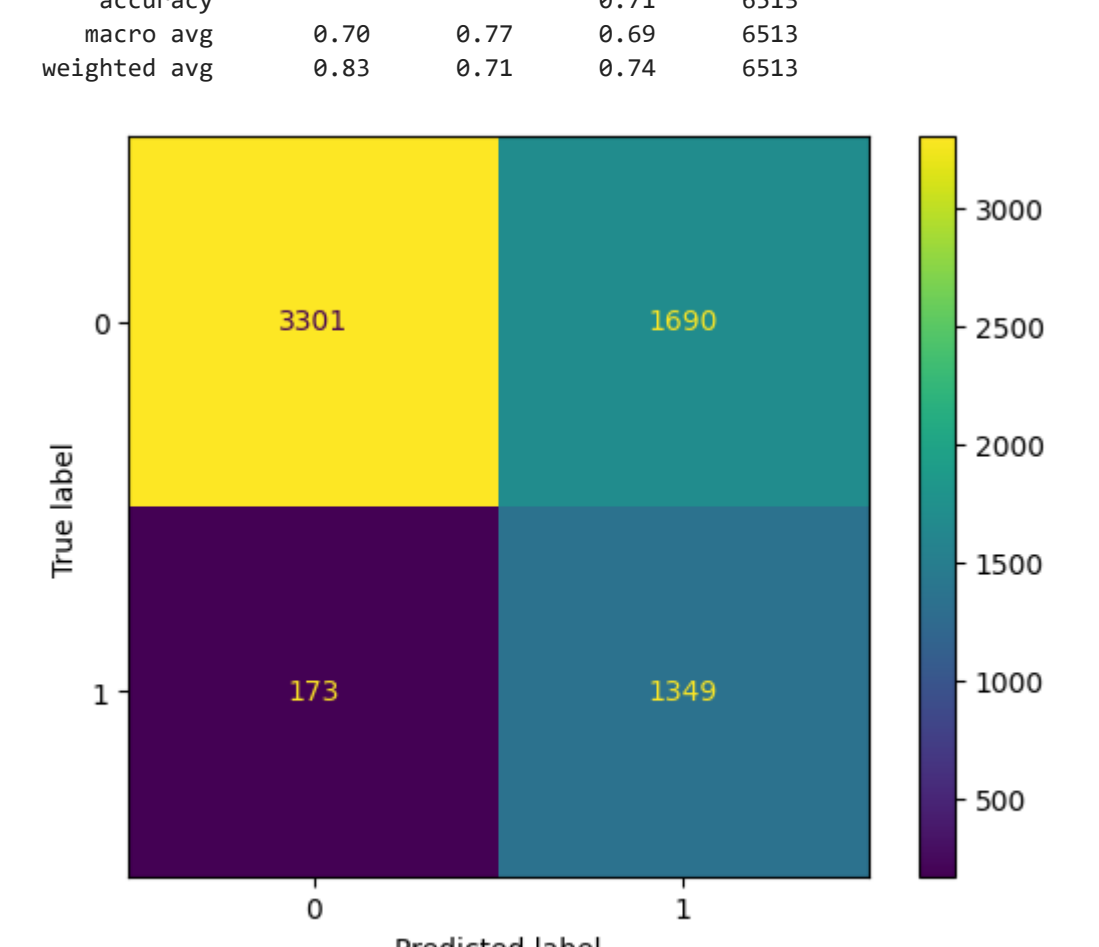






In [17]:
```
tree_best_threshold = find_threshold(tree_scores['roc_curve'])
tree_scores = classification_scores(tree_grid, X_test, y_test,
                                    tree_best_threshold,
                                    plots=False)
```

```
        The best threshold for balancing recall is: 0.29910994108060673

              precision    recall  f1-score   support

           0       0.95      0.66      0.78      4991
           1       0.44      0.89      0.59      1522

    accuracy                           0.71      6513
   macro avg       0.70      0.77      0.69      6513
weighted avg       0.83      0.71      0.74      6513
```



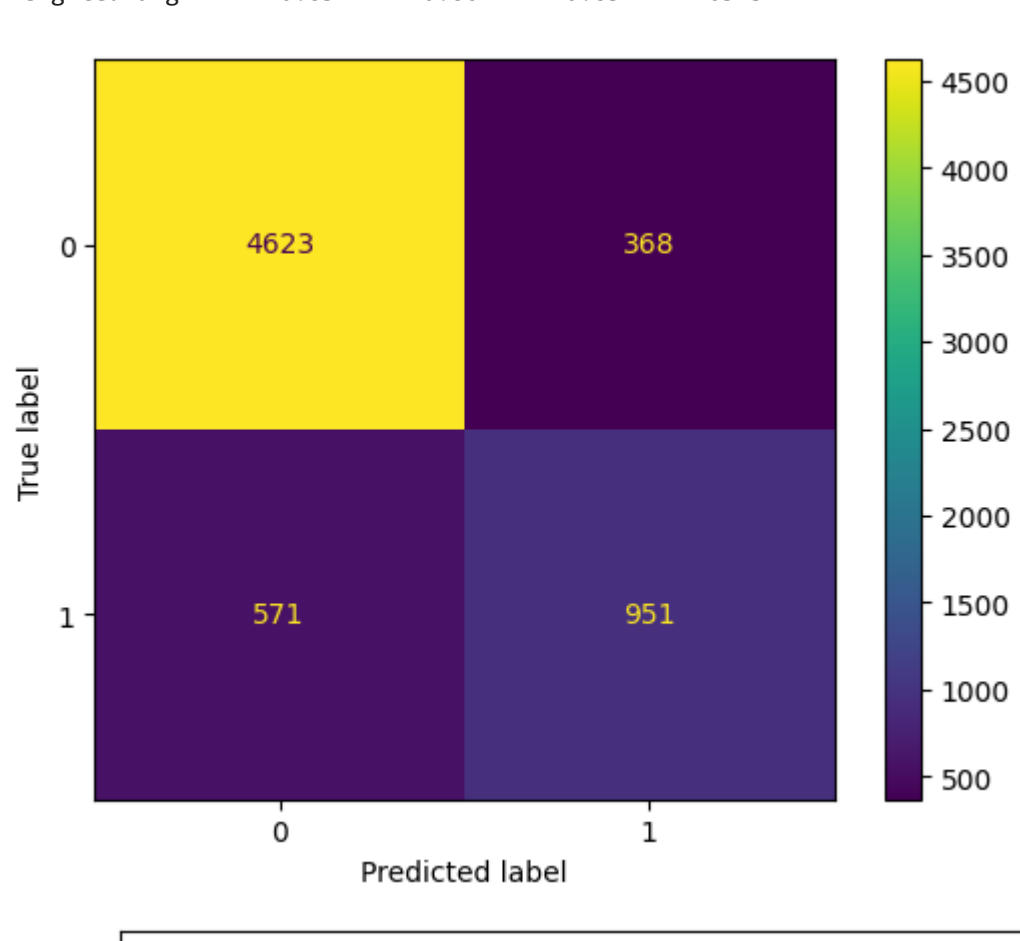
Random Forest

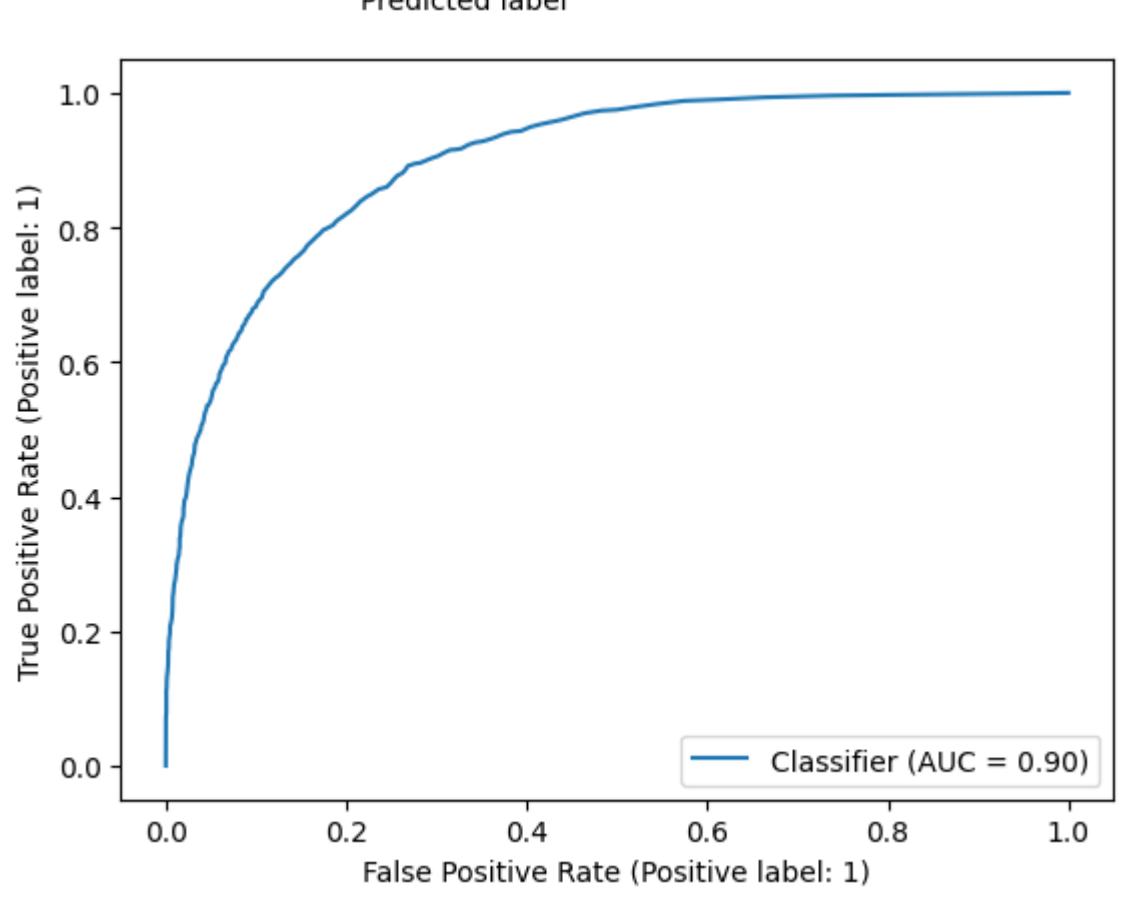
Description Random Forest Regressor and grid search with different hyperparameters.

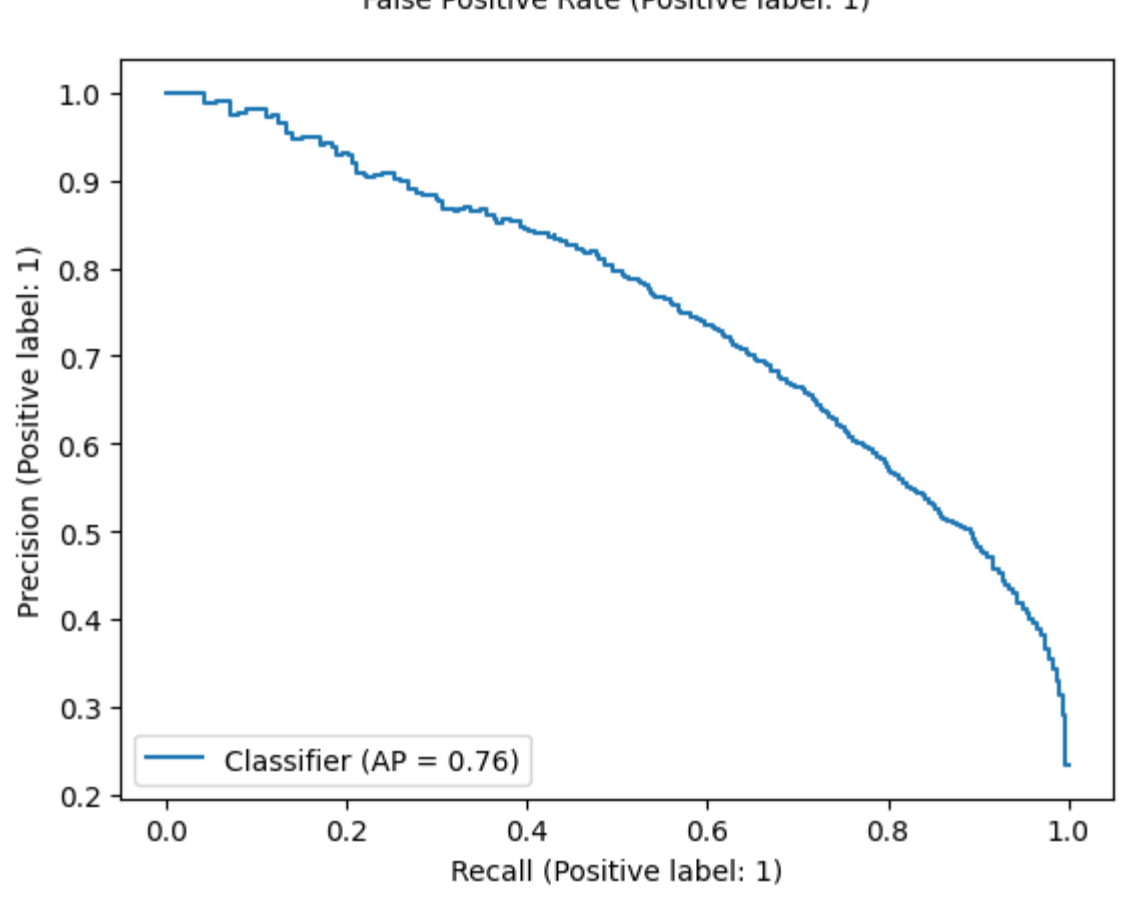
Results Its performance is almost the same as logistic regression also the best threshold performance is too.

In [18]:
```
from sklearn.ensemble import RandomForestClassifier

rfc_model = RandomForestClassifier()
param_grid = {
    'criterion': ['gini', 'entropy'],
    'n_estimators': [50, 100, 200]
}

rfc_grid, rfc_scores = grid_fit_score(rfc_model, param_grid, X_train,
                                      X_test)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best Params are:
        {'criterion': 'gini', 'n_estimators': 200}
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      4991
           1       0.72      0.62      0.67      1522

    accuracy                           0.86      6513
   macro avg       0.81      0.78      0.79      6513
weighted avg       0.85      0.86      0.85      6513
```

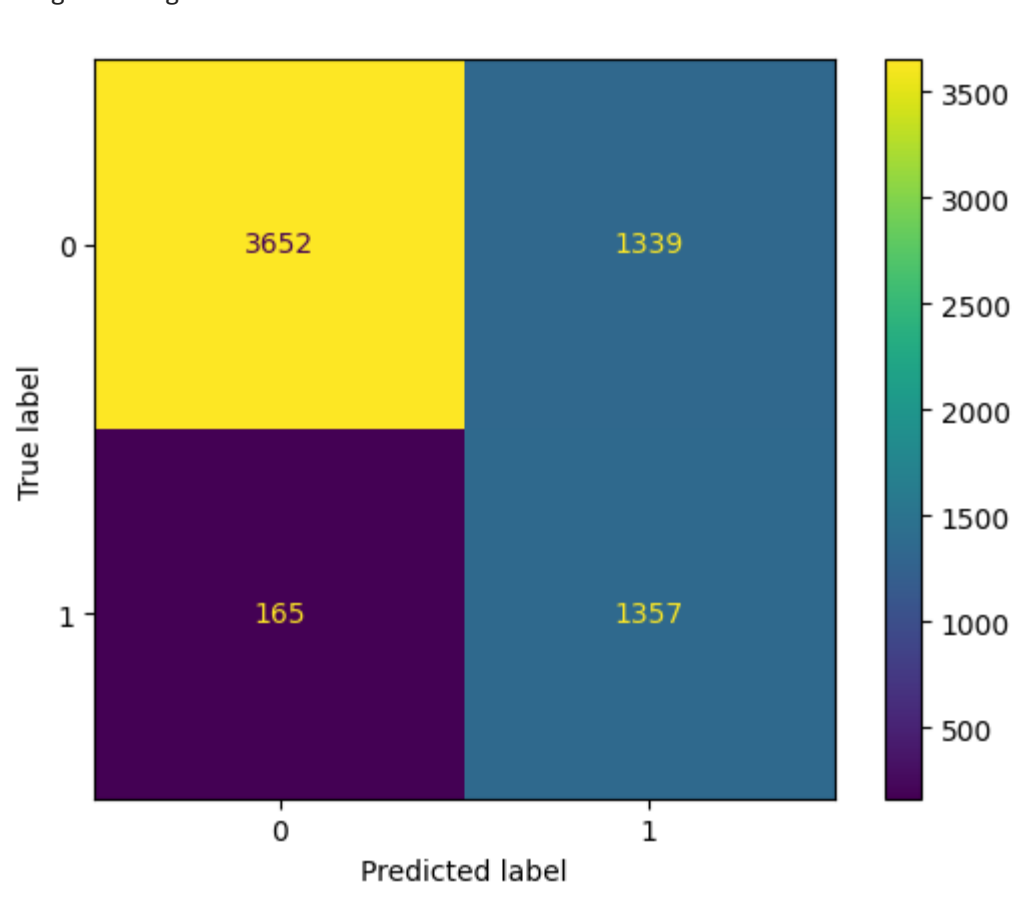






In [19]:
```python
rfc_best_threshold = find_threshold(rfc_scores['roc_curve'])
rfc_scores = classification_scores(rfc_grid, X_test, y_test,
                          rfc_best_threshold,
                          plots=False)
```

```
        The best threshold for balancing recall is: 0.16

              precision    recall  f1-score   support

           0       0.96      0.73      0.83      4991
           1       0.50      0.89      0.64      1522

    accuracy                           0.77      6513
   macro avg       0.73      0.81      0.74      6513
weighted avg       0.85      0.77      0.79      6513
```



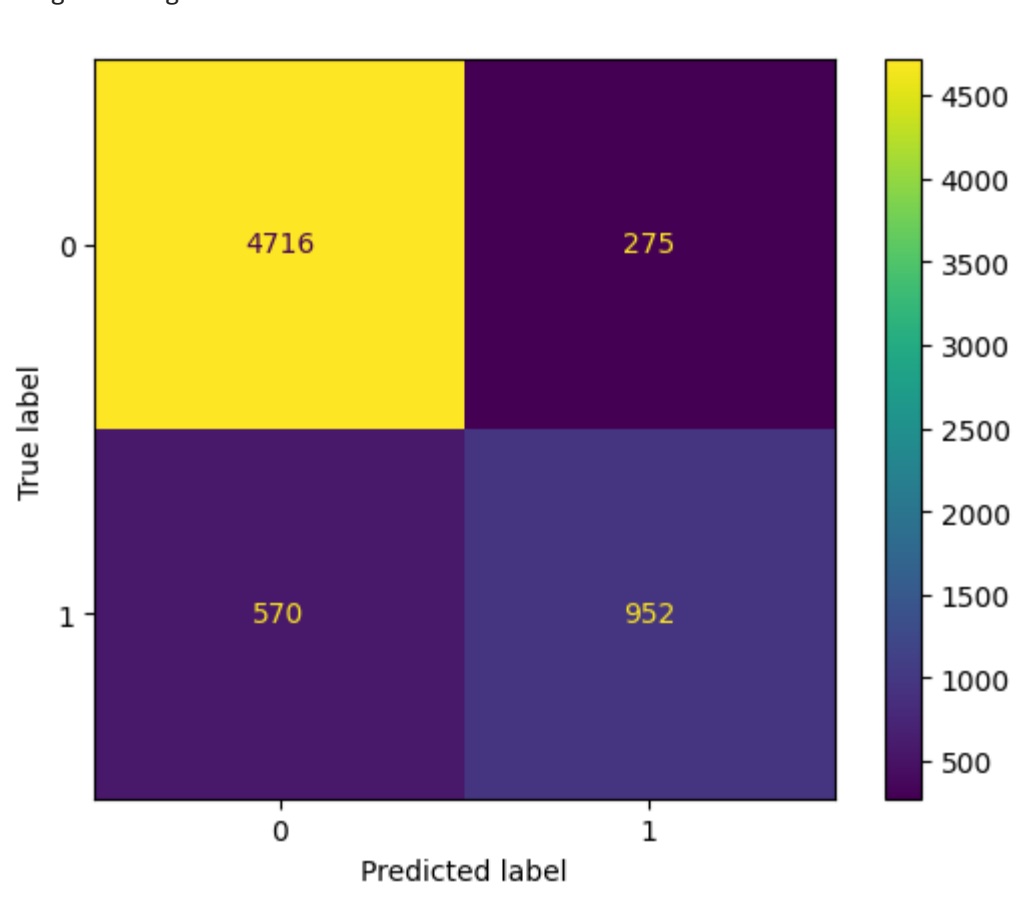
Ada Boost

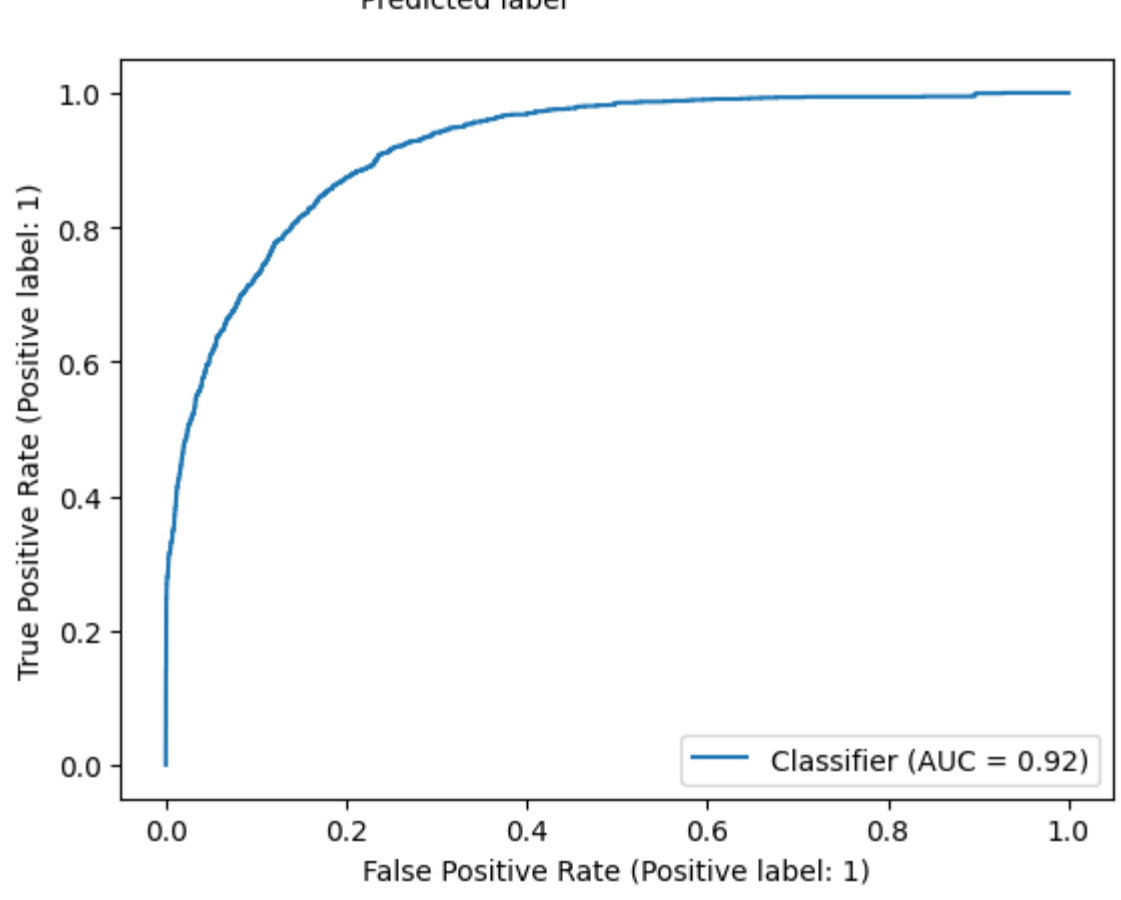
Description AdaBoost on Random Forest Regressor and grid search with different hyperparameters.

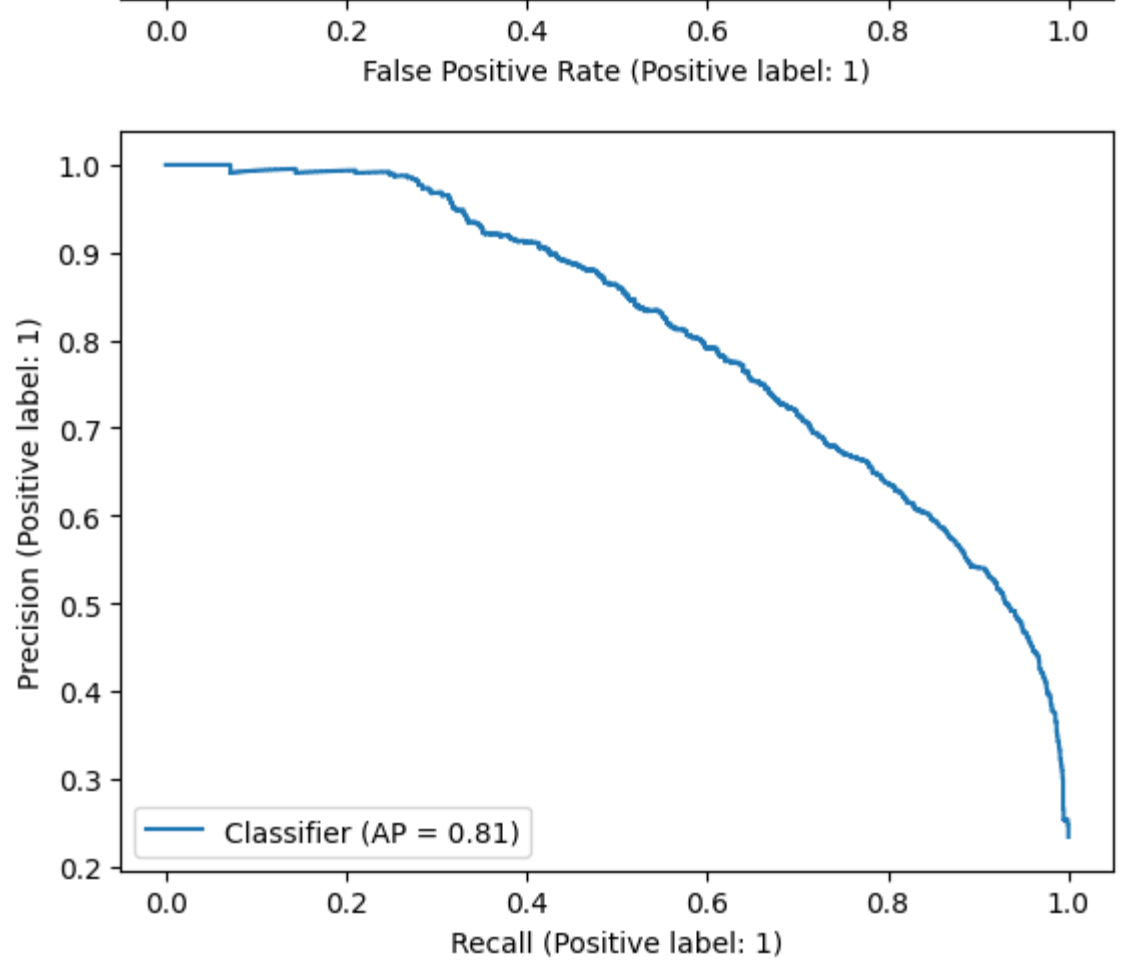
Results Its performance is almost the same as logistic regression also the best threshold performance is too. But it is slightly better on the class 1 score.

In [20]:
```python
from sklearn.ensemble import AdaBoostClassifier

ada_model = AdaBoostClassifier()
param_grid = {
    'n_estimators': [200, 500, 1000],
    'learning_rate': [0.1, 1., 10.],
}
ada_grid, ada_scores = grid_fit_score(ada_model, param_grid, X_train,
                                     X_test)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best Params are:
        {'learning_rate': 1.0, 'n_estimators': 1000}
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      4991
           1       0.78      0.63      0.69      1522

    accuracy                           0.87      6513
   macro avg       0.83      0.79      0.81      6513
weighted avg       0.86      0.87      0.87      6513
```

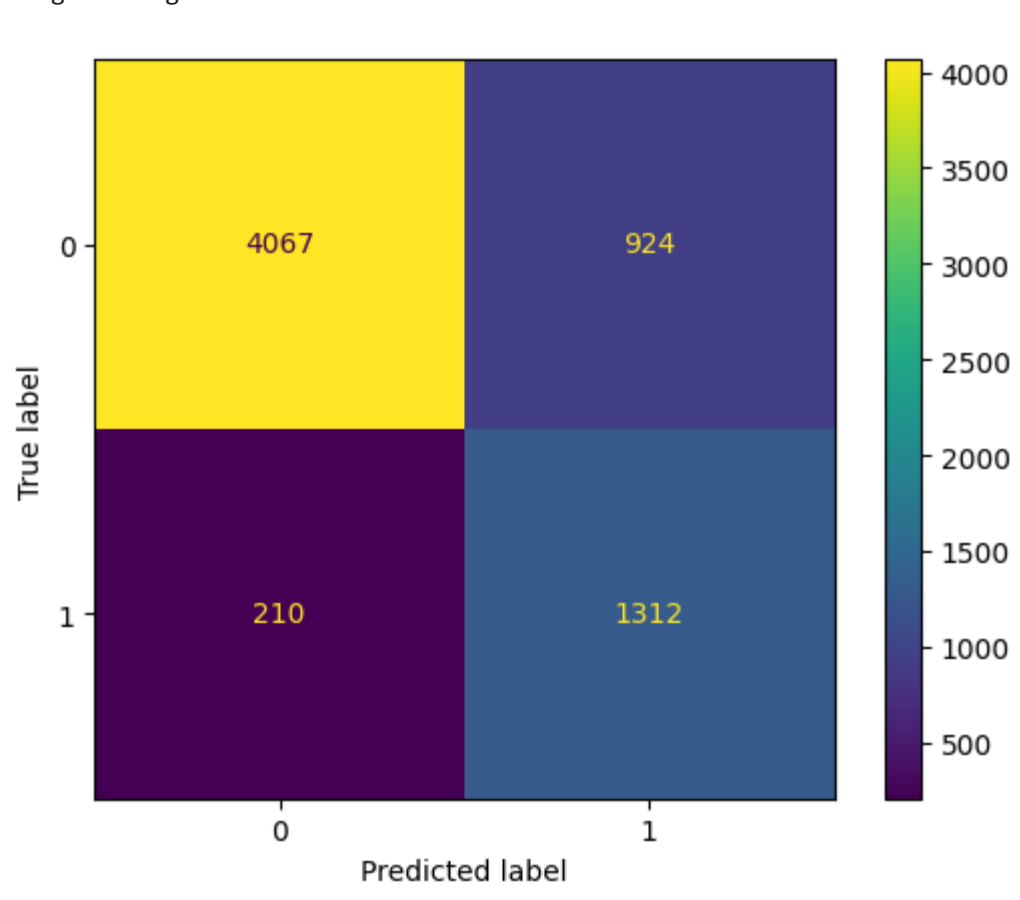






In [21]:
```python
ada_best_threshold = find_threshold(ada_scores['roc_curve'])
ada_scores = classification_scores(ada_grid, X_test, y_test,
                          ada_best_threshold,
                          plots=False)
```

```
        The best threshold for balancing recall is: 0.49966938707183634

              precision    recall  f1-score   support

           0       0.95      0.81      0.88      4991
           1       0.59      0.86      0.70      1522

    accuracy                           0.83      6513
   macro avg       0.77      0.84      0.79      6513
weighted avg       0.87      0.83      0.84      6513
```



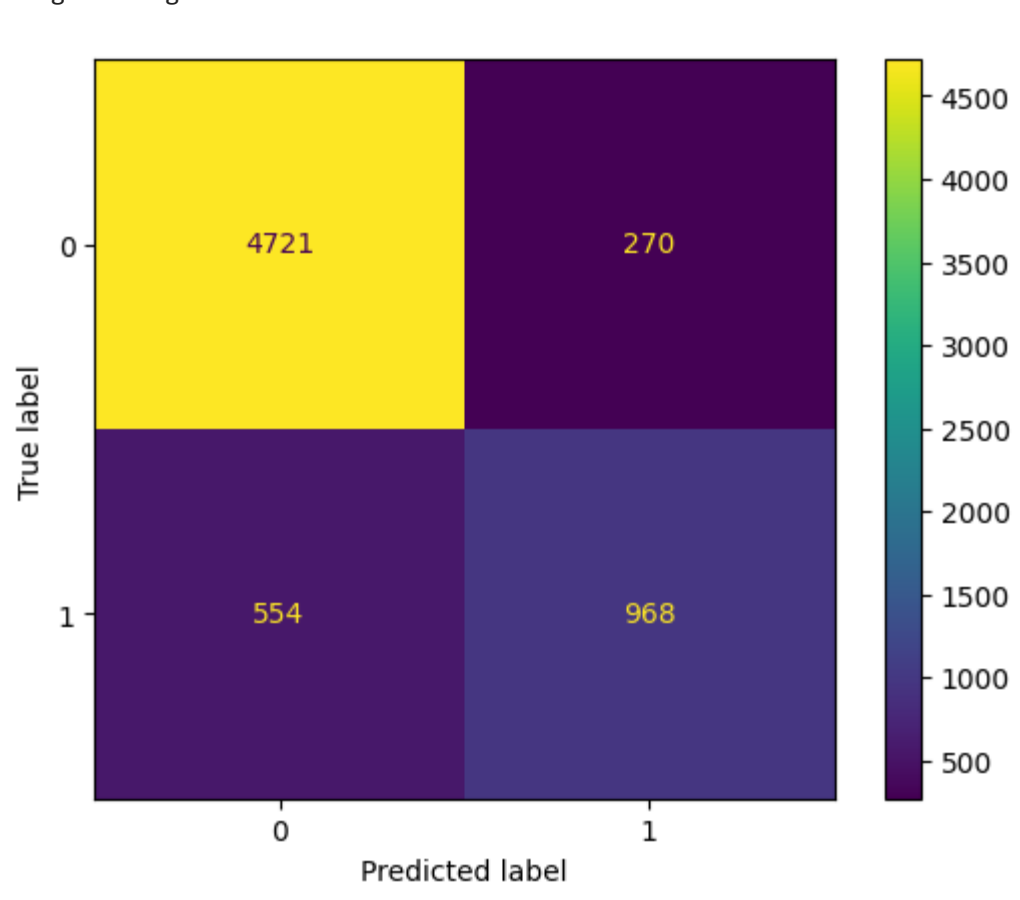
Gradient Boosting

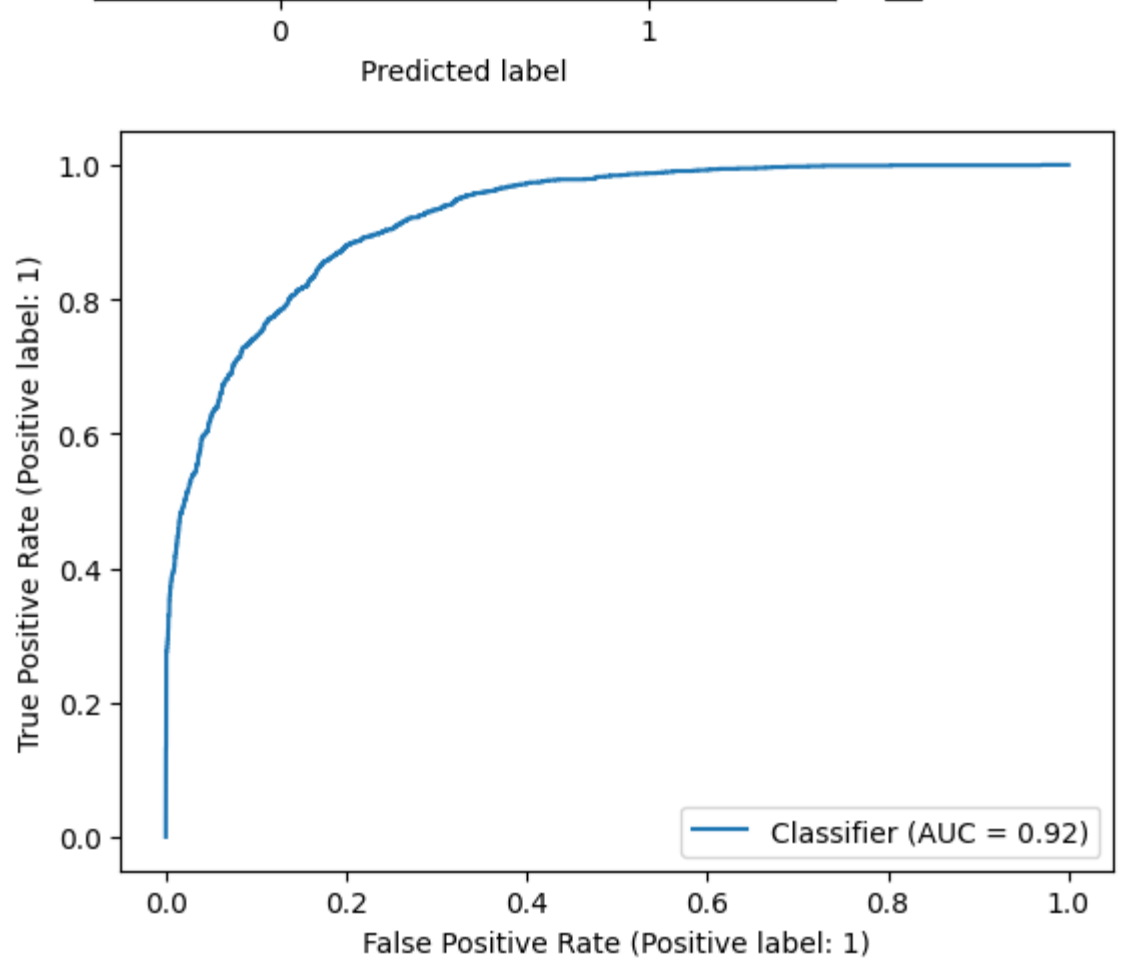
Description Gradient Boosting and grid search with different hyperparameters.

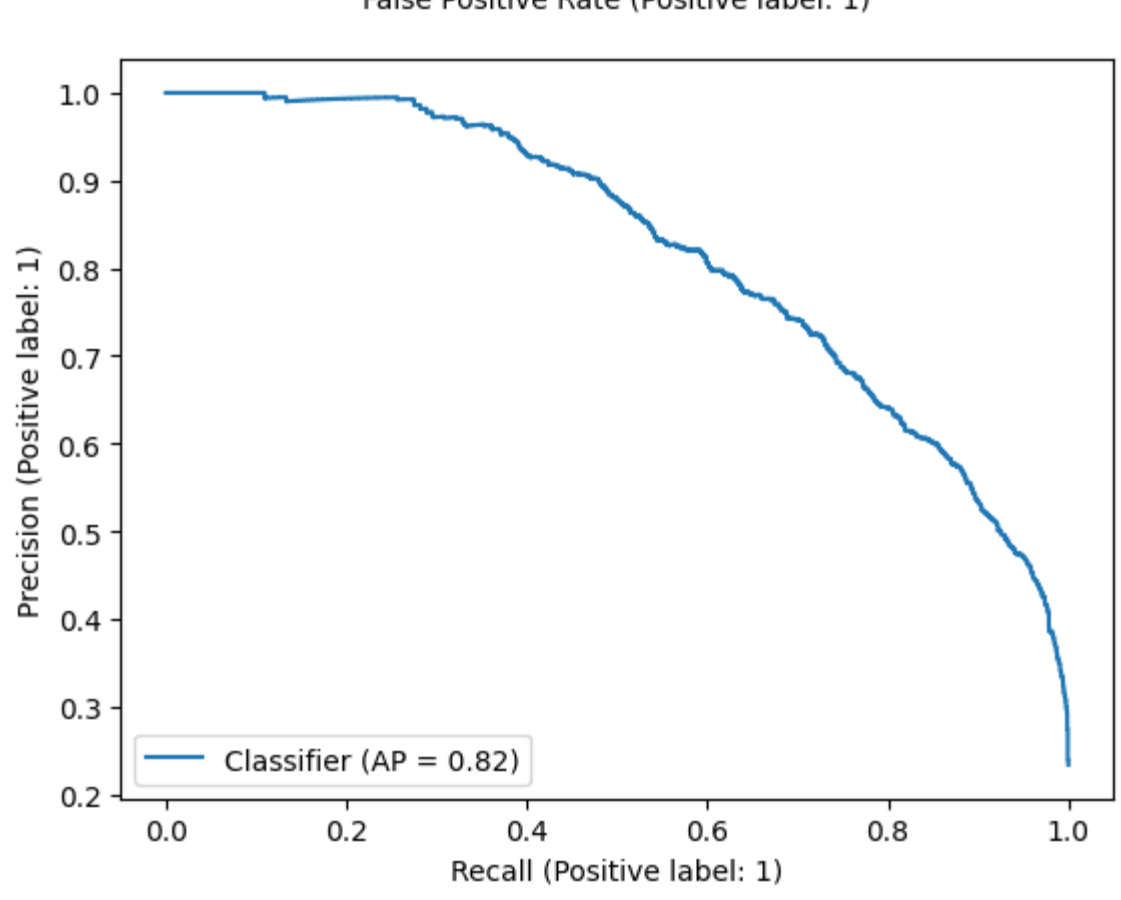
Results Its performance is almost the same as ada boost also the best threshold performance is too. But it is slightly better on the class 0 score.

In [22]:
```python
from sklearn.ensemble import GradientBoostingClassifier

gbc_model = GradientBoostingClassifier()
param_grid = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [3, 5, 10],
}
gbc_grid, gbc_scores = grid_fit_score(gbc_model, param_grid, X_train,
                                     X_test)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best Params are:
        {'max_depth': 5, 'n_estimators': 100}
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      4991
           1       0.78      0.64      0.70      1522

    accuracy                           0.87      6513
   macro avg       0.84      0.79      0.81      6513
weighted avg       0.87      0.87      0.87      6513
```

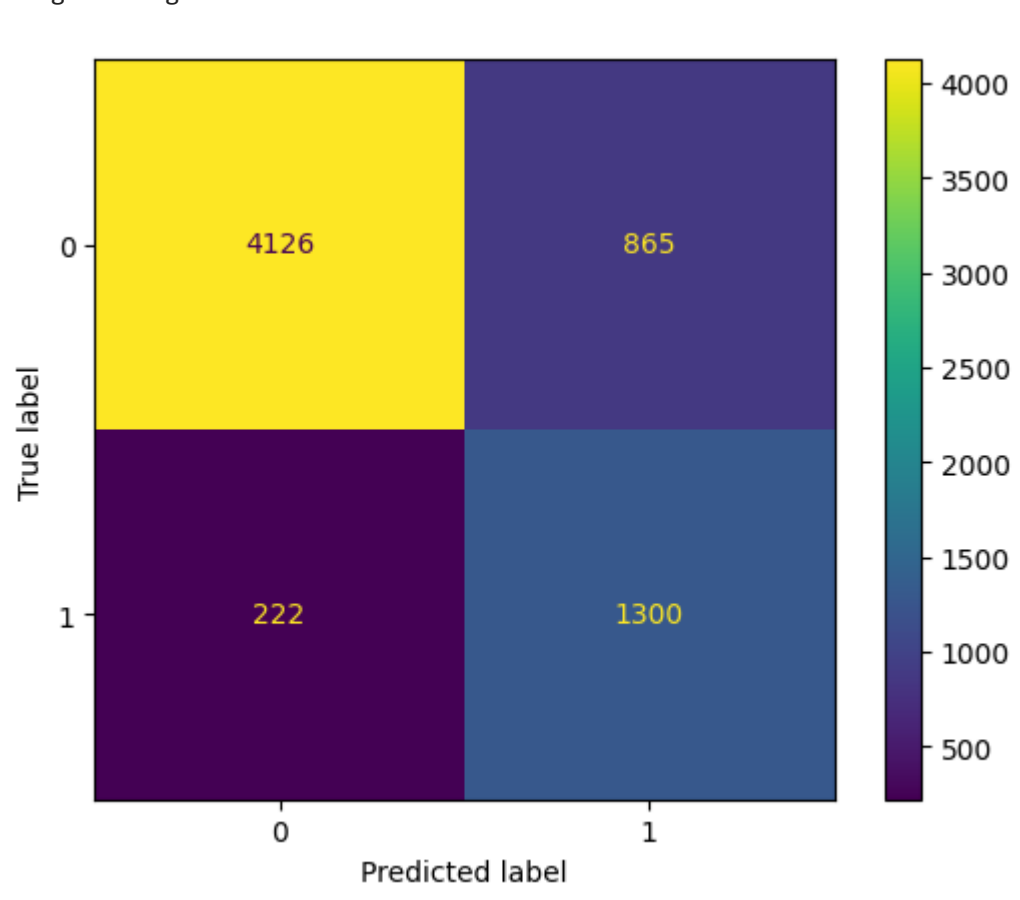






In [23]:
```python
gbc_best_threshold = find_threshold(gbc_scores['roc_curve'])
gbc_scores = classification_scores(gbc_grid, X_test, y_test,
                          gbc_best_threshold,
                          plots=False)
```

```
        The best threshold for balancing recall is: 0.23619458451169806

              precision    recall  f1-score   support

           0       0.95      0.83      0.88      4991
           1       0.60      0.85      0.71      1522

    accuracy                           0.83      6513
   macro avg       0.77      0.84      0.79      6513
weighted avg       0.87      0.83      0.84      6513
```



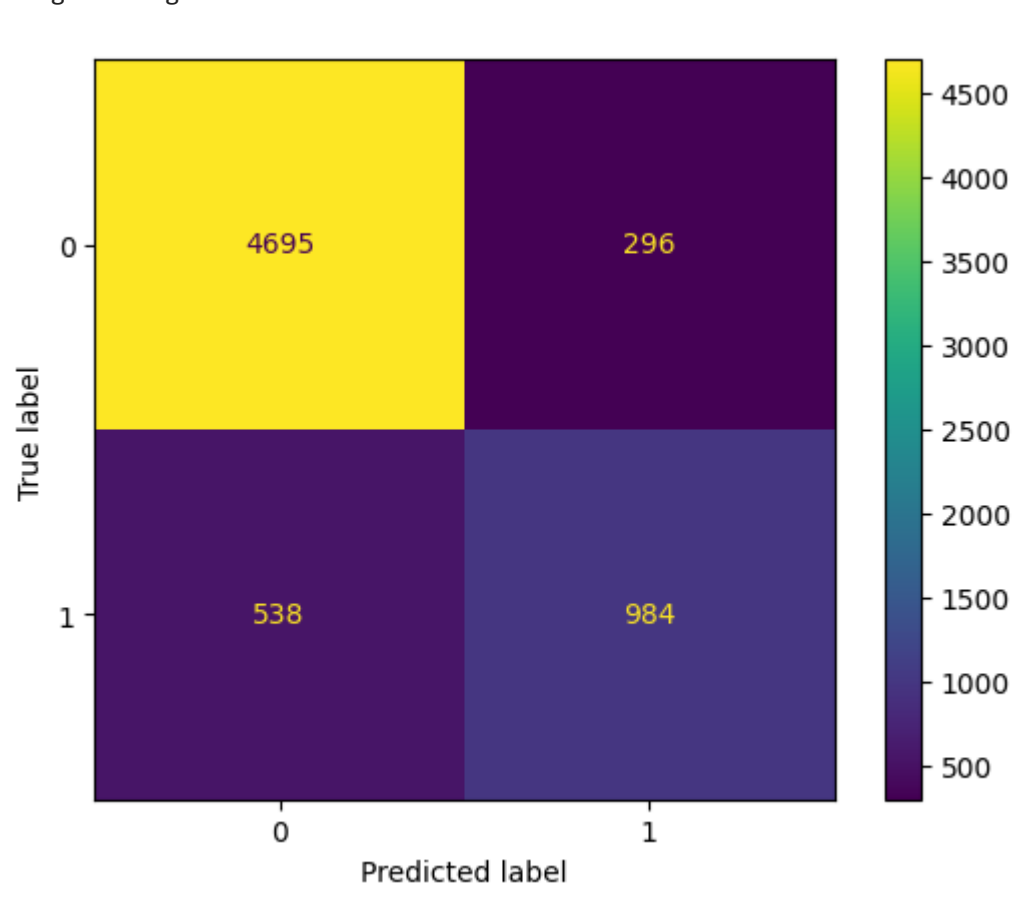
eXtrame Gradient Boosting

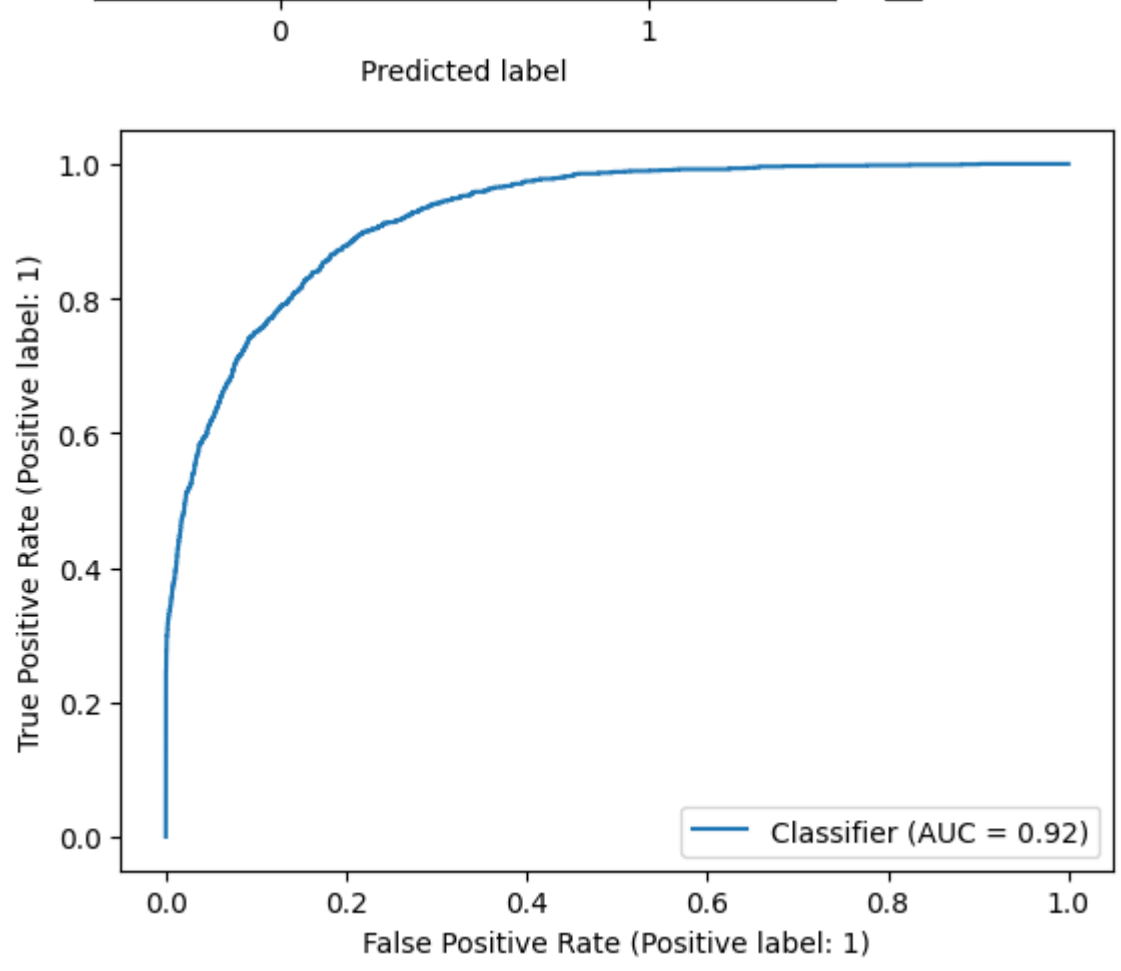
Description Extrame Gradient Boosting grid search with different hyperparameters.

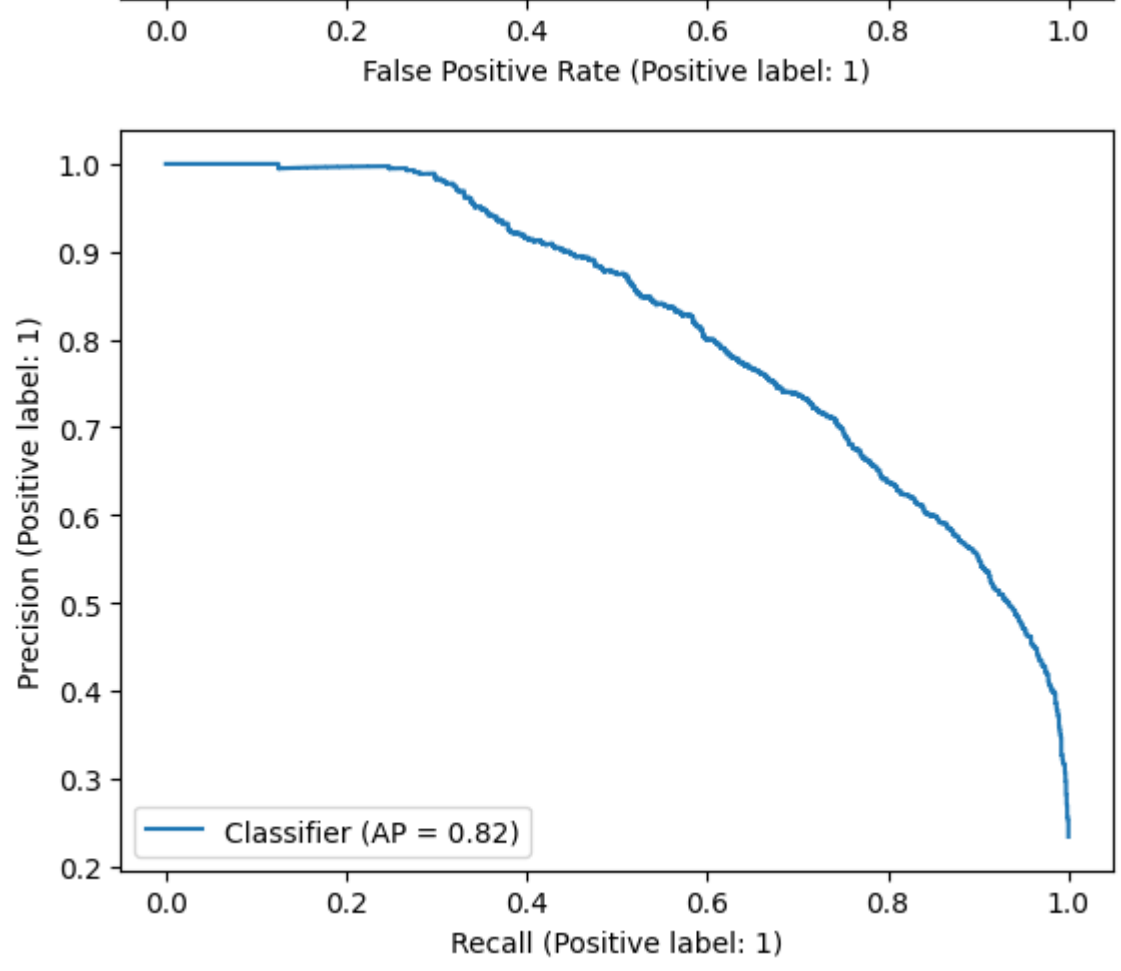
Results Its performance is almost the same as ada boost also the best threshold performance is too.

In [24]:
```python
import xgboost as xgb

xgbc_model = xgb.XGBClassifier()
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [3, 5, 10],
}
xgbc_grid, xgbc_scores = grid_fit_score(xgbc_model, param_grid, X_train,
                                       X_test)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best Params are:
        {'max_depth': 5, 'n_estimators': 100}
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      4991
           1       0.77      0.65      0.70      1522

    accuracy                           0.87      6513
   macro avg       0.83      0.79      0.81      6513
weighted avg       0.87      0.87      0.87      6513
```

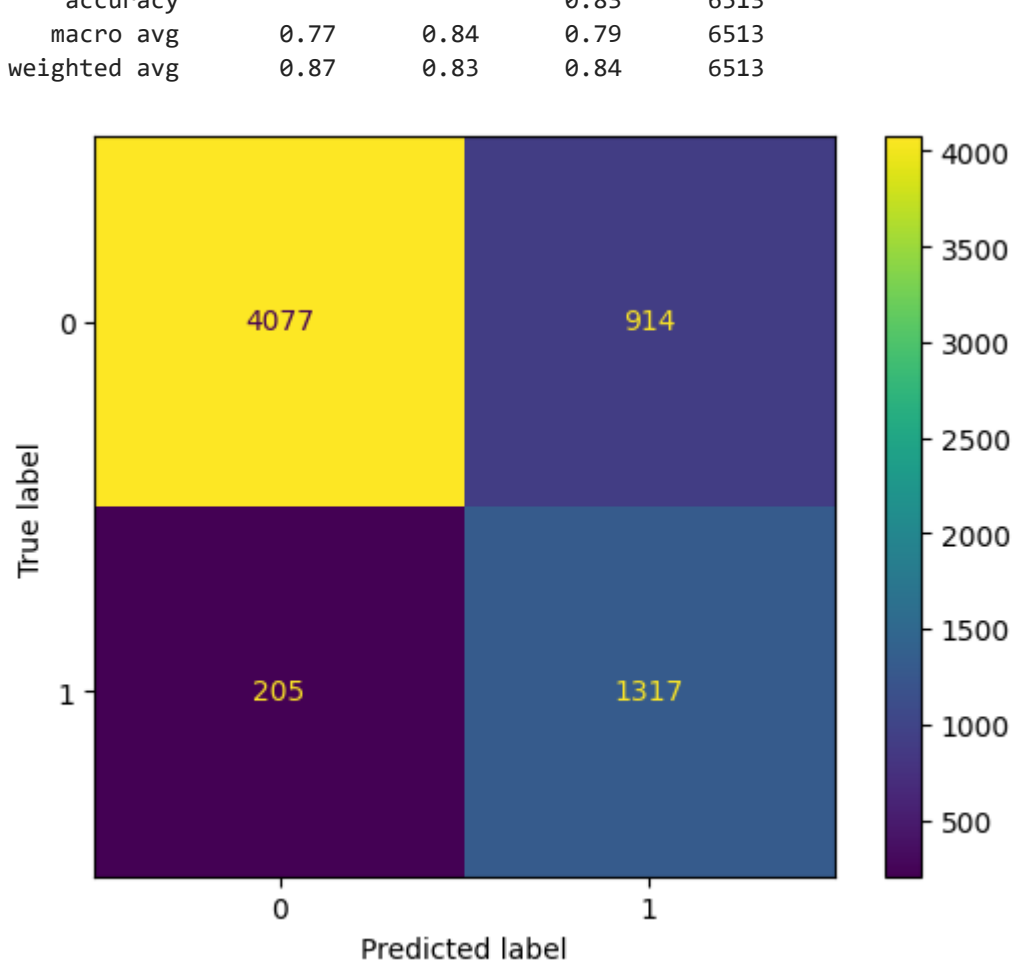






In [25]:
```python
xgbc_best_threshold = find_threshold(xgbc_scores['roc_curve'])
xgbc_scores = classification_scores(xgbc_grid, X_test, y_test,
                          xgbc_best_threshold,
                          plots=False)
```

```
        The best threshold for balancing recall is: 0.2245129644870758

              precision    recall  f1-score   support

           0       0.95      0.82      0.88      4991
           1       0.59      0.87      0.70      1522

    accuracy                           0.83      6513
   macro avg       0.77      0.84      0.79      6513
weighted avg       0.87      0.83      0.84      6513
```



Neural Network

Description I tried simple neural network model with hand tuned hyperparameters.

Results Seems like we didn't get better results with neural network more than boosting models.

In [26]:
```python
import tensorflow as tf
```

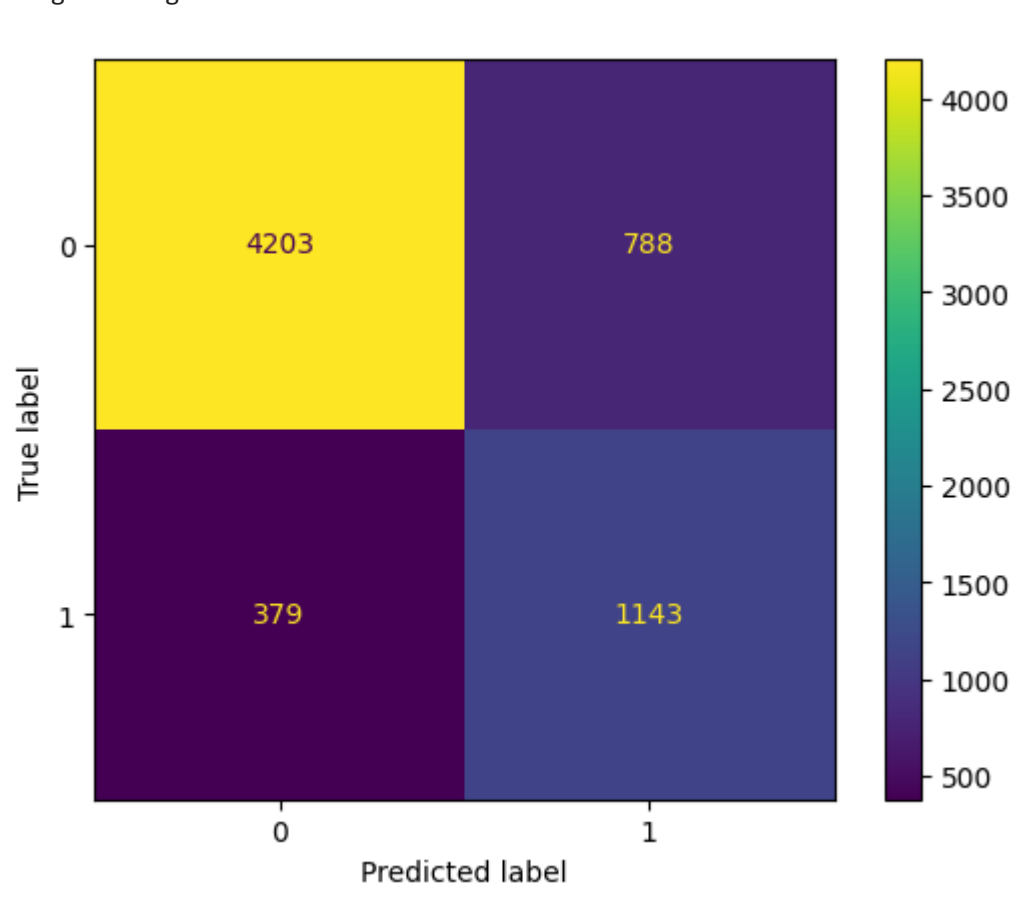
In [27]:
```python
def ann_model_init():
    inputs = tf.keras.Input(shape=(X_train_scaled.shape[1]))
    x = tf.keras.layers.Dense(10, 'relu')(inputs)
    x = tf.keras.layers.Dense(20, 'relu')(x)
    x = tf.keras.layers.Dense(10, 'relu')(x)

    outputs = tf.keras.layers.Dense(1, 'sigmoid')(x)
    return tf.keras.Model(inputs=inputs, outputs=outputs)


ann_model = ann_model_init()
ann_model.summary()
```

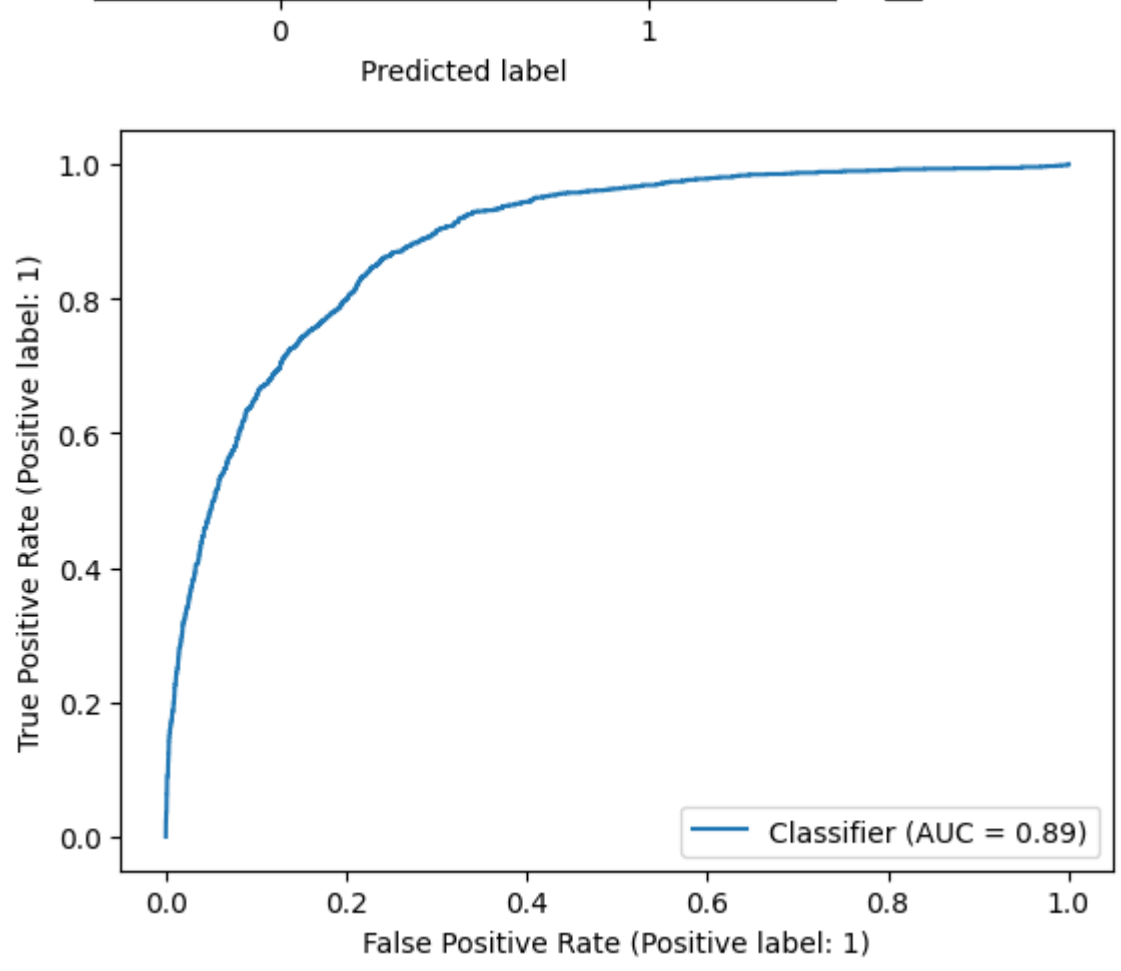
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 107)]             0

 dense (Dense)               (None, 10)                1080

 dense_1 (Dense)             (None, 20)                220

 dense_2 (Dense)             (None, 10)                210

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 1,521
Trainable params: 1,521
Non-trainable params: 0
_________________________________________________________________
```

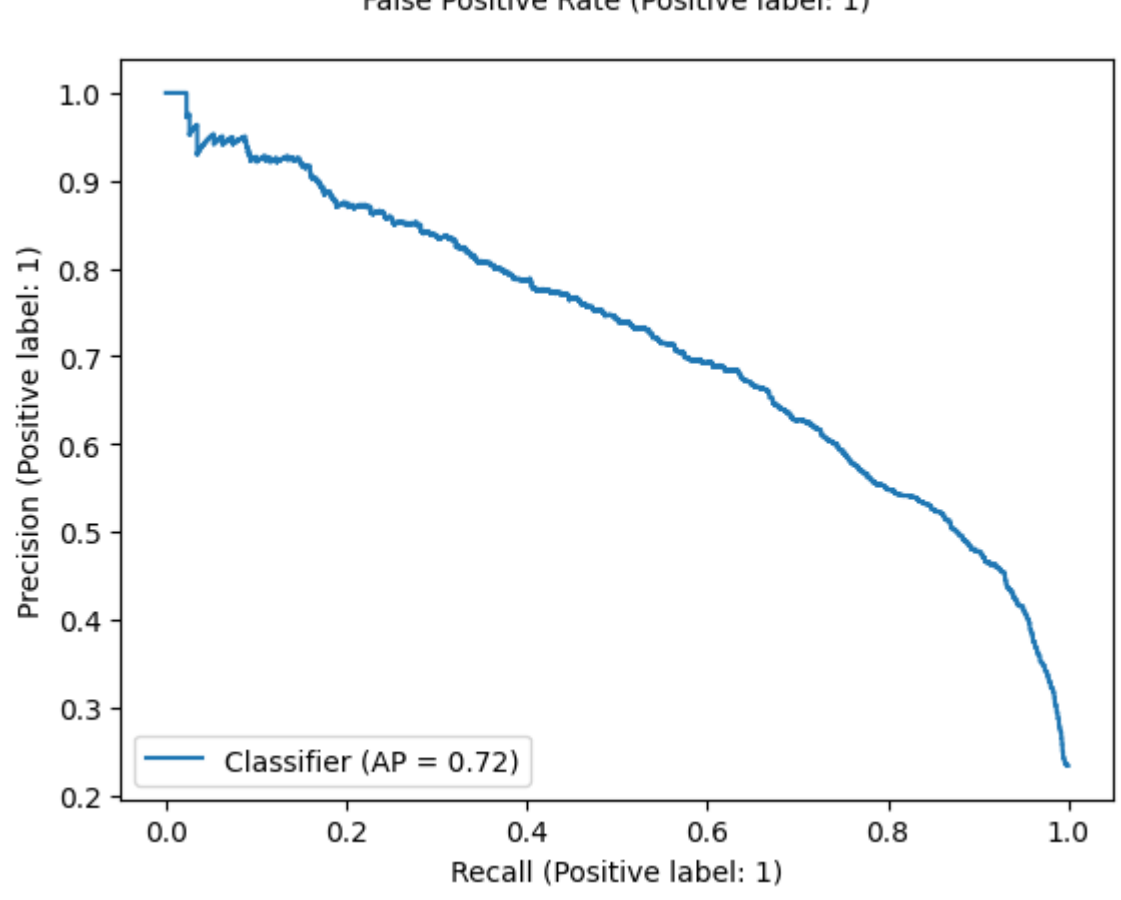
In [28]:
```python
@tf.function
def weighted_binary_crossentropy(y_true, y_pred):
    y_pred = tf.convert_to_tensor(y_pred)
    y_true = tf.cast(y_true, y_pred.dtype)
    epsilon_ = 1e-7
    return -tf.reduce_sum(
        2. * y_true * tf.math.log(y_pred + epsilon_) + (
            1. - y_true) * tf.math.log(
          1. - y_pred + epsilon_), axis=-1)


optimizer = tf.keras.optimizers.Adam(0.005)
metric = tf.keras.metrics.BinaryAccuracy(threshold=0.5)
compile_params = {'optimizer': optimizer, 'loss': weighted_binary_crossentropy,
                  'metrics': [metric]}

ann_model.compile(**compile_params)

history = ann_model.fit(x=X_train_scaled,
                        y=y_train,
                        epochs=500,
                        batch_size=X_train_scaled.shape[0],
                        verbose=0,
                        shuffle=True
                        )
```

In [29]:
```python
ann_scores = classification_scores(ann_model, X_test_scaled, y_test,
                                   tf_model=True)
```

```
204/204 [==============================] - 0s 847us/step
              precision    recall  f1-score   support

           0       0.92      0.84      0.88      4991
           1       0.59      0.75      0.66      1522

    accuracy                           0.82      6513
   macro avg       0.75      0.80      0.77      6513
weighted avg       0.84      0.82      0.83      6513
```

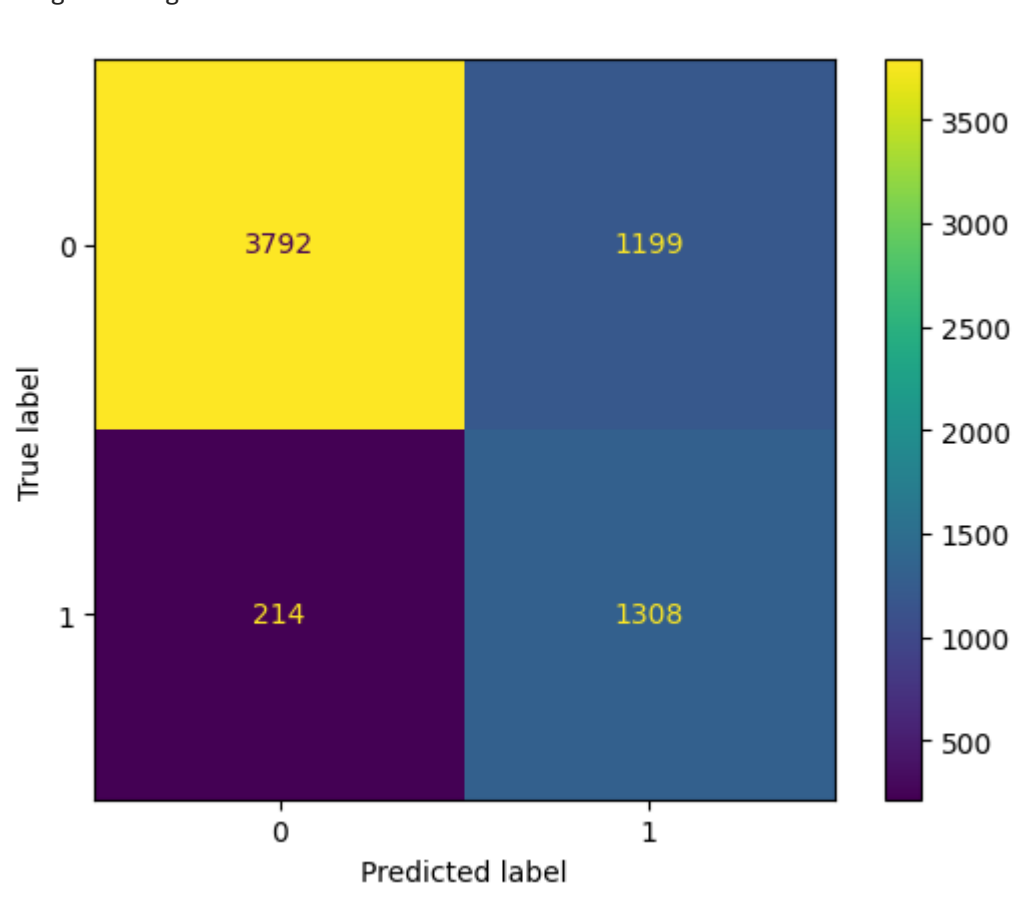






In [30]:
```python
ann_best_threshold = find_threshold(ann_scores['roc_curve'])
ann_scores = classification_scores(ann_model, X_test_scaled, y_test,
                          ann_best_threshold,
                          plots=False, tf_model=True)
```

```
        The best threshold for balancing recall is: 0.3269025683403015

204/204 [==============================] - 0s 847us/step
              precision    recall  f1-score   support

           0       0.95      0.76      0.84      4991
           1       0.52      0.86      0.65      1522

    accuracy                           0.78      6513
   macro avg       0.73      0.81      0.75      6513
weighted avg       0.85      0.78      0.80      6513
```



Deploying Models

In this section, I saved all models after retraining on the full data set because it will be used for practical purposes so we would like to get all information we can possibly get. I also saved feature transformers for transforming new data.

Performance Summary As a result, I demonstrated below table all of the model scores. There are lots of good models but considering with test set performance and because of simplicity, I would use **Gradient Boosting** for future predictions.

In [31]:
```python
# scores table code
def gs(scores):
    sc = scores['classification_reports']['macro avg'].copy()
    sc.pop('support', None)
    sc['accuracy'] = scores['classification_reports']['accuracy']
    return sc

all_scores = pd.DataFrame([gs(score) for score in [
    log_scores, knn_scores, svc_scores, tree_scores,
    rfc_scores, ada_scores, gbc_scores, xgbc_scores, ann_scores]],
                          index=['Logistic Regression','K-Nearest Neighbors','Support Vector Machine','Decis
                                 'Random Forest','Ada Boost','Gradient Boosting','eXtreme Gradient Boosting'
                                 'Neural Network']
                         )
all_scores = all_scores.round(4)
pd.set_option('display.float_format', '{:.4f}'.format)
with open('../reports/0.3-test-scores.html', 'wt') as f:
    f.write(all_scores.to_html())
all_scores.name = 'Test Set Macro Avg. Scores'
print(all_scores.name)
all_scores
```

Test Set Macro Avg. Scores

Out[31]:

	precision	recall	f1-score	accuracy
Logistic Regression	0.7562	0.8226	0.7741	0.8145
K-Nearest Neighbors	0.7206	0.7909	0.7314	0.7709
Support Vector Machine	0.7418	0.8204	0.7542	0.7895
Decision Tree	0.6970	0.7739	0.6857	0.7140
Random Forest	0.7301	0.8117	0.7363	0.7691
Ada Boost	0.7688	0.8384	0.7879	0.8259
Gradient Boosting	0.7747	0.8404	0.7944	0.8331
eXtreme Gradient Boosting	0.7712	0.8411	0.7906	0.8282
Neural Network	0.7342	0.8096	0.7461	0.7830

In [32]:
```python
final_scaler = StandardScaler()
X_scaled = final_scaler.fit_transform(X)
```

In [33]:

```python
from joblib import dump
import json
import os
from sklearn.metrics import roc_curve

threshold_file = '../models/featurebuild/' + '0.3-besttreshold.json'
if not os.path.exists(threshold_file):
    json.dump({}, open(threshold_file, 'wt'))


def save_threshold(model, model_name, tf_model=False):
    y_vals = y
    if (isinstance(y_vals, pd.Series)):
        y_vals = y_vals.values
    if tf_model:
        y_pred_proba = model.predict(X)[:, 0]
    else:
        y_pred_proba = model.predict_proba(X)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_vals, y_pred_proba)
    roc_vals = pd.DataFrame(
        {'FPR': fpr, 'TPR': tpr, 'Thresholds': thresholds})
    best_treshold_ = find_threshold(roc_vals)
    thresholds_ = json.load(open(threshold_file))
    thresholds_[model_name] = float(best_treshold_)
    json.dump(thresholds_, open(threshold_file, 'wt'))


def train_deploy(model_class, X, model_name, **kwargs):
    model = model_class(**kwargs)
    model.fit(X, y)
    save_threshold(model, model_name)
    model_path = models_trained_dir / f'{model_name}.joblib'
    dump(model, model_path)
    return model
```

In [34]:

```python
log_final = train_deploy(LogisticRegression, X_scaled, '0.1-logisticregression',
                         **log_grid.best_params_)
knn_final = train_deploy(KNeighborsClassifier, X_scaled,
                         '0.2-knearestneighbors', **{'n_neighbors': best_nn})
svc_final = train_deploy(SVC, X_scaled, '0.3-supportvector',
                         **{**svc_grid.best_params_, 'probability': True})
tree_final = train_deploy(DecisionTreeClassifier, X, '0.4-decisiontree',
                          **tree_grid.best_params_)
rfc_final = train_deploy(RandomForestClassifier, X, '0.5-randomforest',
                         **rfc_grid.best_params_)
ada_final = train_deploy(AdaBoostClassifier, X, '0.6-adaboost',
                         **ada_grid.best_params_)
gbc_final = train_deploy(GradientBoostingClassifier, X, '0.7-gradientboosting',
                         **gbc_grid.best_params_)
xgbc_final = train_deploy(xgb.XGBClassifier, X, '0.8-extremegradientboosting',
                          **xgbc_grid.best_params_)
```

```
The best threshold for balancing recall is: 2.0

The best threshold for balancing recall is: 0.26666666666666666

The best threshold for balancing recall is: 1.9999999999999698

The best threshold for balancing recall is: 0.2967231185489528

The best threshold for balancing recall is: 0.5110595238095239

The best threshold for balancing recall is: 0.4996891661000805

The best threshold for balancing recall is: 0.2581189570296502

The best threshold for balancing recall is: 0.24891822040081024
```

In [35]:

```python
ann_final = ann_model_init()
ann_final.compile(**{**compile_params, 'optimizer':tf.keras.optimizers.Adam(0.005)})
ann_final.fit(x=X_scaled, y=y,
              epochs=500,
              batch_size=X_scaled.shape[0],
              verbose=0,
              shuffle=True)
ann_final.save(models_trained_dir / f'0.9-neuralnet.h5')
save_threshold(ann_final, '0.9-neuralnet.h5', tf_model=True)
```

```
1018/1018 [==============================] - 1s 799us/step

The best threshold for balancing recall is: 2.1688528319593775e-19
```

In [36]:

```python
dump(final_scaler, '../models/featurebuild/' + '0.2-standardscaler.joblib')
```

Out[36]:

```
['../models/featurebuild/0.2-standardscaler.joblib']
```

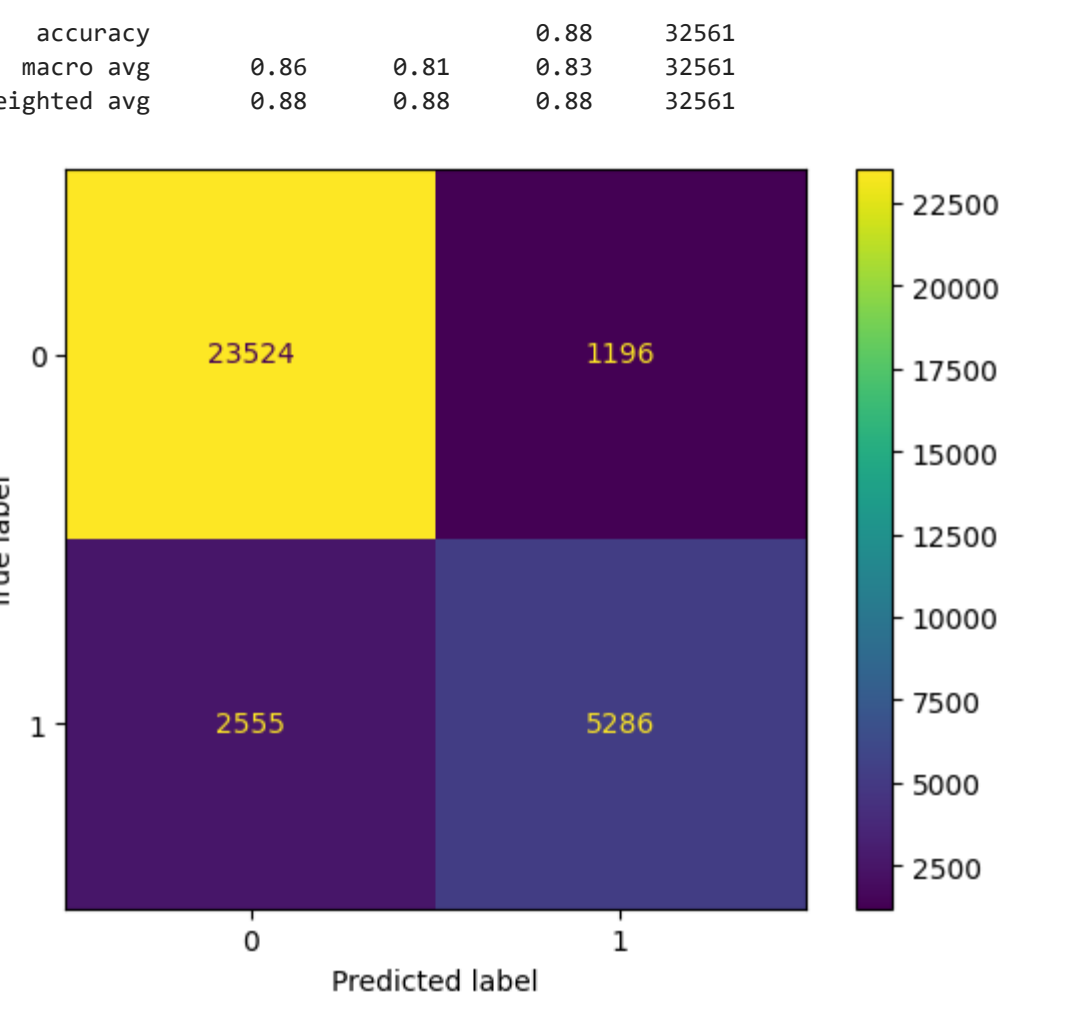
Performance on one of the Final Modals

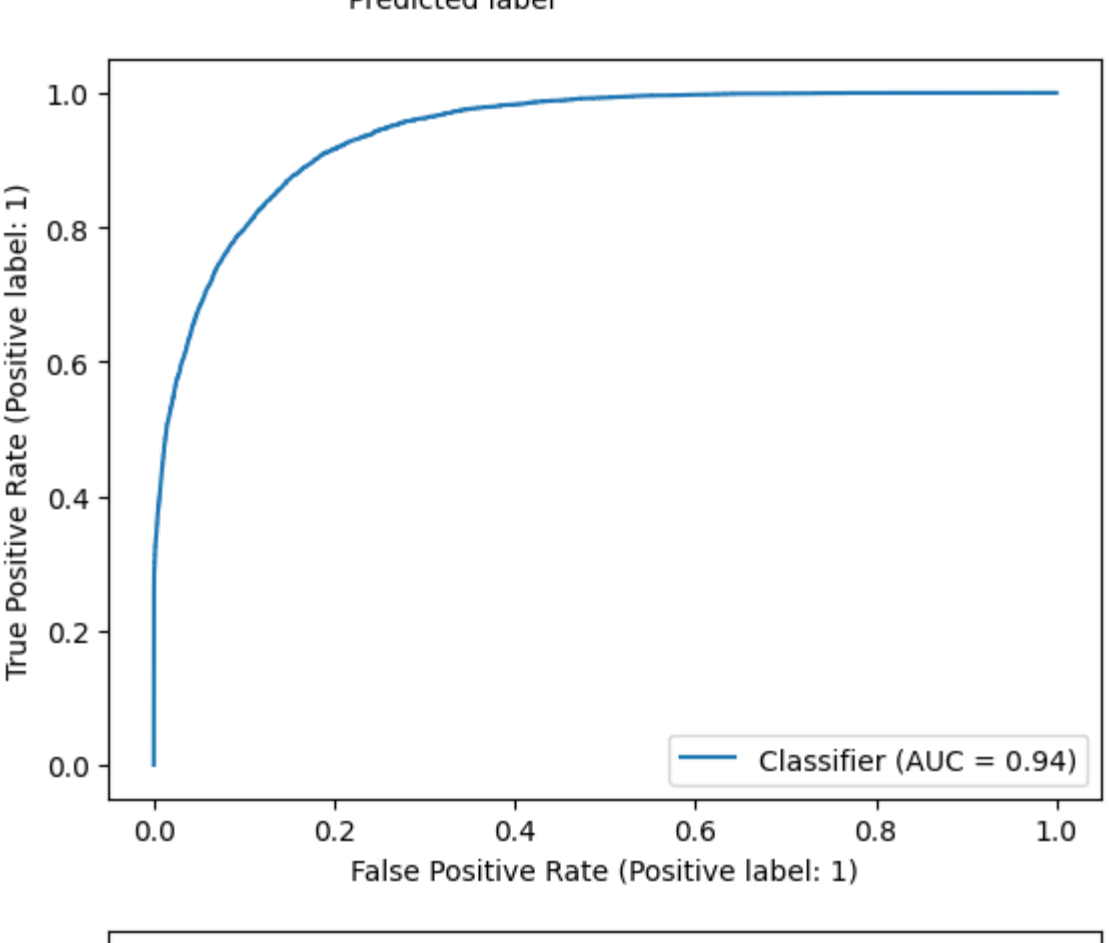
You can see Gradient Boosting Final Scores

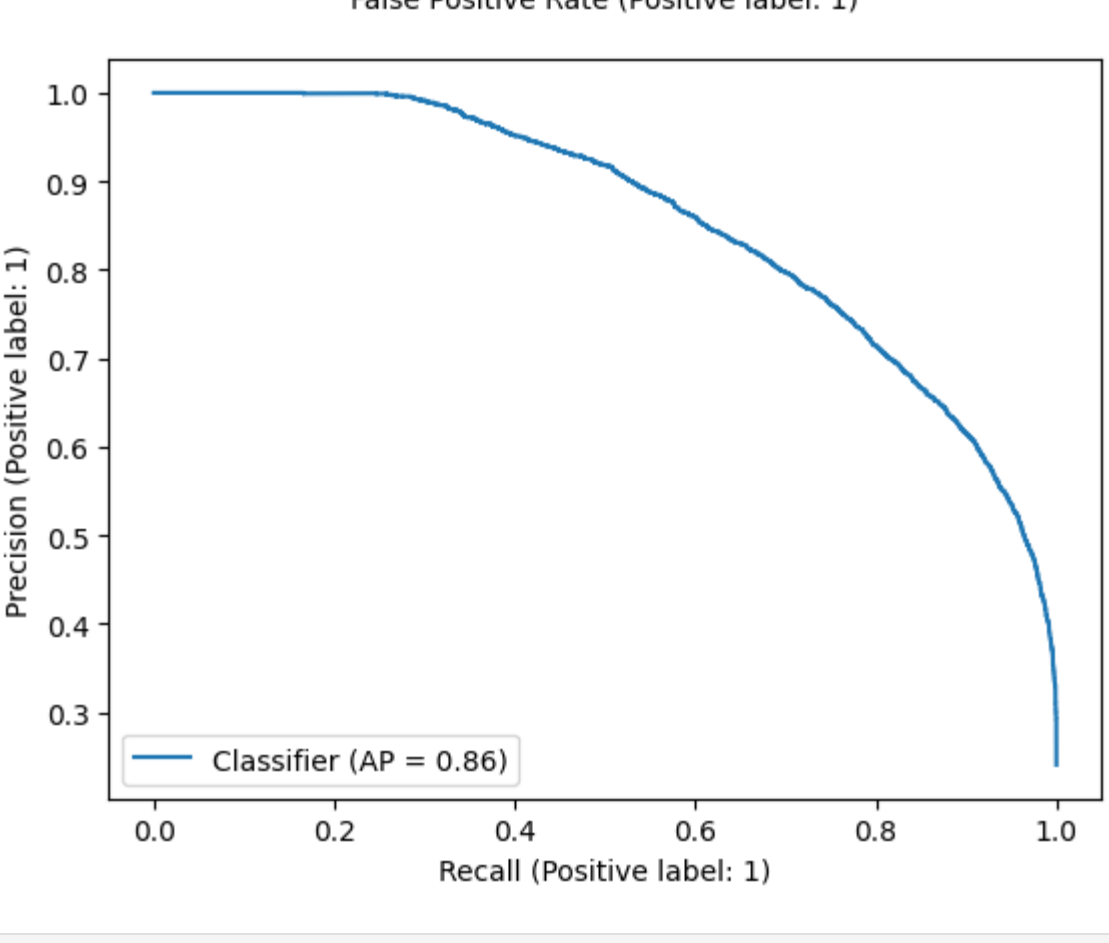
In [37]:

```python
gbc_final_scores = classification_scores(gbc_final, X, y)
```

```
              precision    recall  f1-score   support

           0       0.90      0.95      0.93     24720
           1       0.82      0.67      0.74      7841

    accuracy                           0.88     32561
   macro avg       0.86      0.81      0.83     32561
weighted avg       0.88      0.88      0.88     32561
```







In [38]:

```python
gbc_final_best_threshold = find_threshold(gbc_final_scores['roc_curve'])
_ = classification_scores(gbc_final, X, y,
                          gbc_final_best_threshold,
                          plots=False)
```

```
The best threshold for balancing recall is: 0.2581189570296502

              precision    recall  f1-score   support

           0       0.96      0.85      0.90     24720
           1       0.64      0.88      0.74      7841

    accuracy                           0.85     32561
   macro avg       0.80      0.86      0.82     32561
weighted avg       0.88      0.85      0.86     32561
```