In [1]:
```
from imblearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE, RandomOverSampler
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold
from yellowbrick.classifier import ConfusionMatrix
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_validate, cross_val_predict
from sklearn.metrics import recall_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import plotly.express as px
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from xgboost import plot_tree as plot_tree_xgb
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
```

In [2]:
```
# read the hotel reservation file

hotel = pd.read_csv('Hotel Reservations.csv')
pd.options.display.max_colwidth = 100
```

In [3]:
```
hotel = hotel.drop('Booking_ID', axis = 1)
```

In [4]:
```
label_encoder_type_of_meal_plan = LabelEncoder()
label_encoder_room_type_reserved = LabelEncoder()
label_encoder_market_segment_type = LabelEncoder()
label_encoder_booking_status = LabelEncoder()

hotel['type_of_meal_plan'] = label_encoder_type_of_meal_plan.fit_transform(hotel['type_o
hotel['room_type_reserved'] = label_encoder_room_type_reserved.fit_transform(hotel['room
hotel['market_segment_type'] = label_encoder_market_segment_type.fit_transform(hotel['ma
hotel['booking_status'] = label_encoder_booking_status.fit_transform(hotel['booking_stat
```

In [5]:
```
le_name_mapping = dict(zip(label_encoder_booking_status.classes_, label_encoder_booking_
print(le_name_mapping)
```

```
{'Canceled': 0, 'Not_Canceled': 1}
```

In [6]:
```
X_not_altered = hotel.drop('booking_status', axis = 1)
```

In [7]:
```
X = hotel.drop('booking_status', axis = 1)
X_copy = X.copy()
X = X.values
X_standard = StandardScaler().fit_transform(X)
y = hotel['booking_status']
```

In [8]:
```
x_train, x_test, y_train, y_test = train_test_split(X_standard, y, test_size=0.2, random
```

In [10]:
```
x_train_balanced, y_train_balanced = SMOTE().fit_resample(x_train, y_train)
```

```
In [9]: scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

        kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

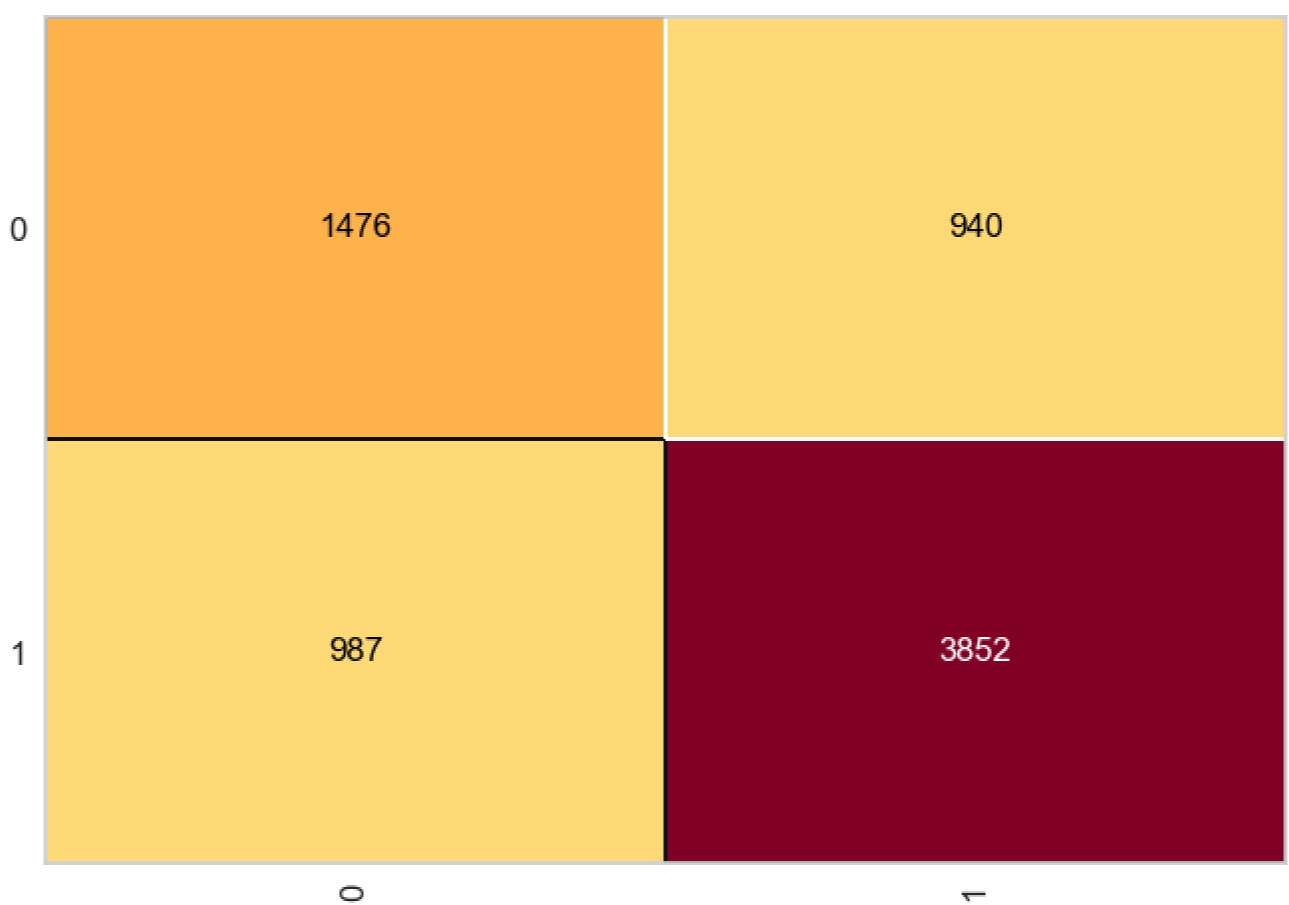
Naive Bayes

```
In [15]: classifier = make_pipeline(SMOTE(random_state=100), GaussianNB(var_smoothing=1e-03))
         nb_scores = cross_validate(classifier, X, y, scoring=scoring, cv=kf)
         print(nb_scores)
         print("Mean score for accuracy: ", nb_scores['test_accuracy'].mean())
         print("Mean score for precision: ", nb_scores['test_precision'].mean())
         print("Mean score for recall: ", nb_scores['test_recall'].mean())
         print("Mean score for f1: ", nb_scores['test_f1'].mean())
         print("Mean score for roc_auc: ", nb_scores['test_roc_auc'].mean())
```

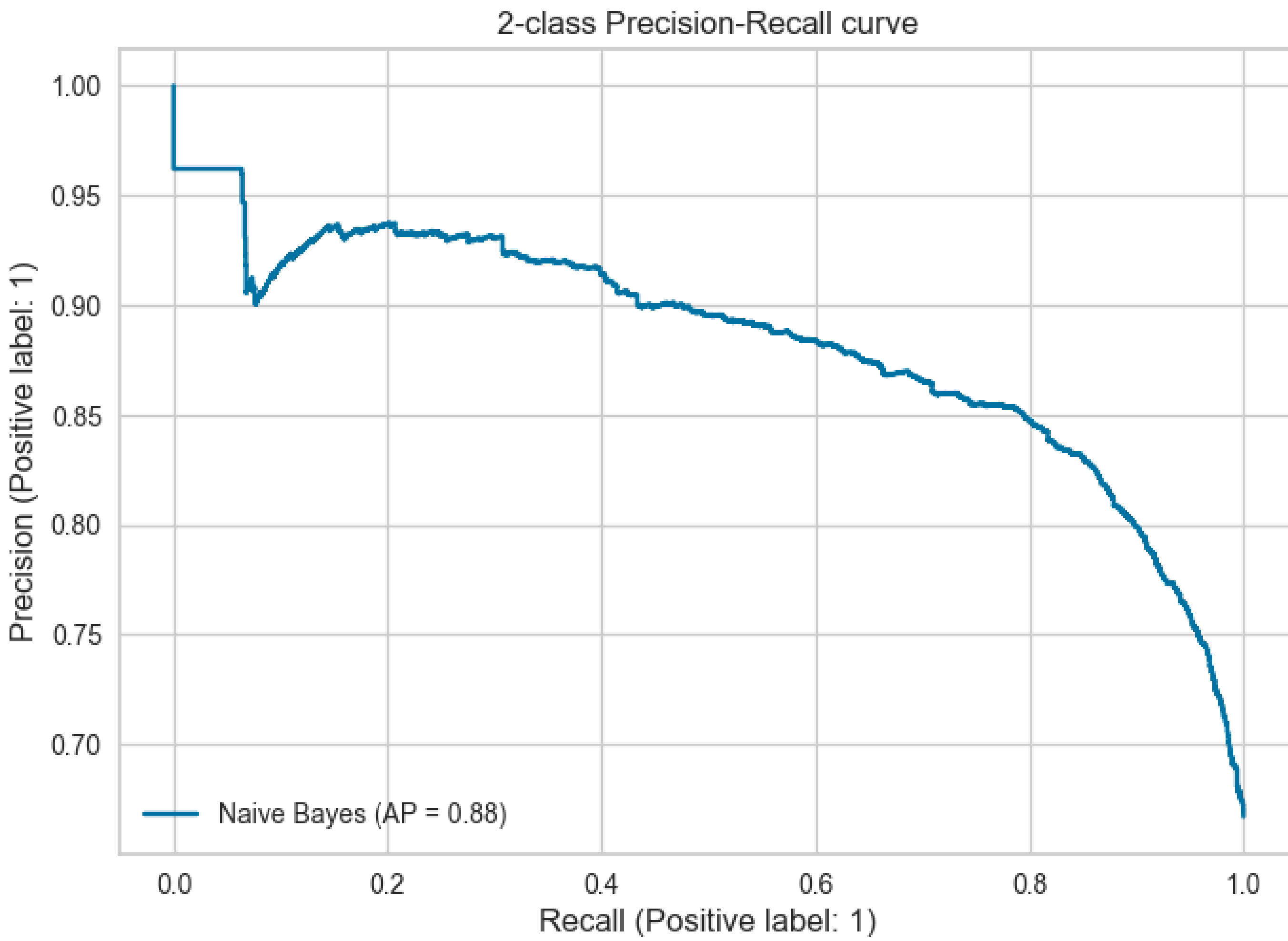
```
{'fit_time': array([0.44772625, 0.21080518, 0.20424461, 0.33565331, 0.22131705]), 'score
_time': array([0.02854943, 0.02813101, 0.02661753, 0.03220463, 0.03520608]), 'test_accur
acy': array([0.73494142, 0.72984149, 0.72929014, 0.72170917, 0.73618194]), 'test_precisi
on': array([0.80591691, 0.80535381, 0.80104822, 0.8059126 , 0.81132853]), 'test_recall':
array([0.79375904, 0.79155619, 0.7901158 , 0.77169231, 0.79785295]), 'test_f1': array
([0.79979178, 0.79839539, 0.79554445, 0.78843131, 0.80453431]), 'test_roc_auc': array
([0.78576227, 0.77168619, 0.77467742, 0.77310291, 0.78035137])}
Mean score for accuracy:  0.7303928325292902
Mean score for precision:  0.8059120122043858
Mean score for recall:  0.788995256843682
Mean score for f1:  0.797339448606072
Mean score for roc_auc:  0.7771160301919415
```

```
In [149…  cm = ConfusionMatrix(classifier)
          cm.fit(x_train_balanced, y_train_balanced)
          cm.score(x_test, y_test)
```

```
Out[149]: 0.7343900758097863
```



In [16]:
```
classifier.fit(x_train, y_train)
display = PrecisionRecallDisplay.from_estimator(
    classifier, x_test, y_test, name="Naive Bayes"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [17]:
```
y_pred = classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.37      0.98      0.53      2416
Not Canceled       0.94      0.15      0.26      4839

    accuracy                           0.43      7255
   macro avg       0.65      0.57      0.40      7255
weighted avg       0.75      0.43      0.35      7255
```

KNN

In [12]:
```
from sklearn.metrics import accuracy_score
scores = []

#Determine the best number of neighbours
knn = KNeighborsClassifier()
classifier = make_pipeline(StandardScaler(), SMOTE(random_state=100), knn)

no_neighbours_list = list(range(1,10))
# k_values = dict(n_neighbors = no_neighbours_list)
```

```
params = {'kneighborsclassifier__n_neighbors': no_neighbours_list}
# perform a new split inside method
grid = GridSearchCV(classifier, param_grid=params, cv = kf, scoring =  scoring, n_jobs =
```

In [ ]:
```
grid.fit(X, y)
```

In [94]:
```
accuracies = []
precisions = []
recalls = []
f1s = []
roc_aucs = []

for i in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    classifier = make_pipeline(StandardScaler(), SMOTE(random_state=100), knn)
    score = cross_validate(classifier, X, y, scoring=scoring, cv=kf)
    # take scores
    test_accuracy = score['test_accuracy'].mean()
    test_precision = score['test_precision'].mean()
    test_recall = score['test_recall'].mean()
    test_f1 = score['test_f1'].mean()
    test_roc_auc = score['test_roc_auc'].mean()

    accuracies.append(test_accuracy)
    precisions.append(test_precision)
    recalls.append(test_recall)
    f1s.append(test_f1)
    roc_aucs.append(test_roc_auc)

    print("Iteration: ", i)
```

```
Iteration:  1
Iteration:  2
Iteration:  3
Iteration:  4
Iteration:  5
Iteration:  6
Iteration:  7
Iteration:  8
Iteration:  9
```

In [95]:
```
# create a dataframe of the scores
# scores_df = pd.DataFrame(scores, columns=['neighbors', 'score'])
# scores_df

scores = pd.DataFrame({'neighbors': no_neighbours_list, 'accuracy': accuracies, 'precisi
scores.sort_values(by=['accuracy'], ascending=False)
```

Out[95]:

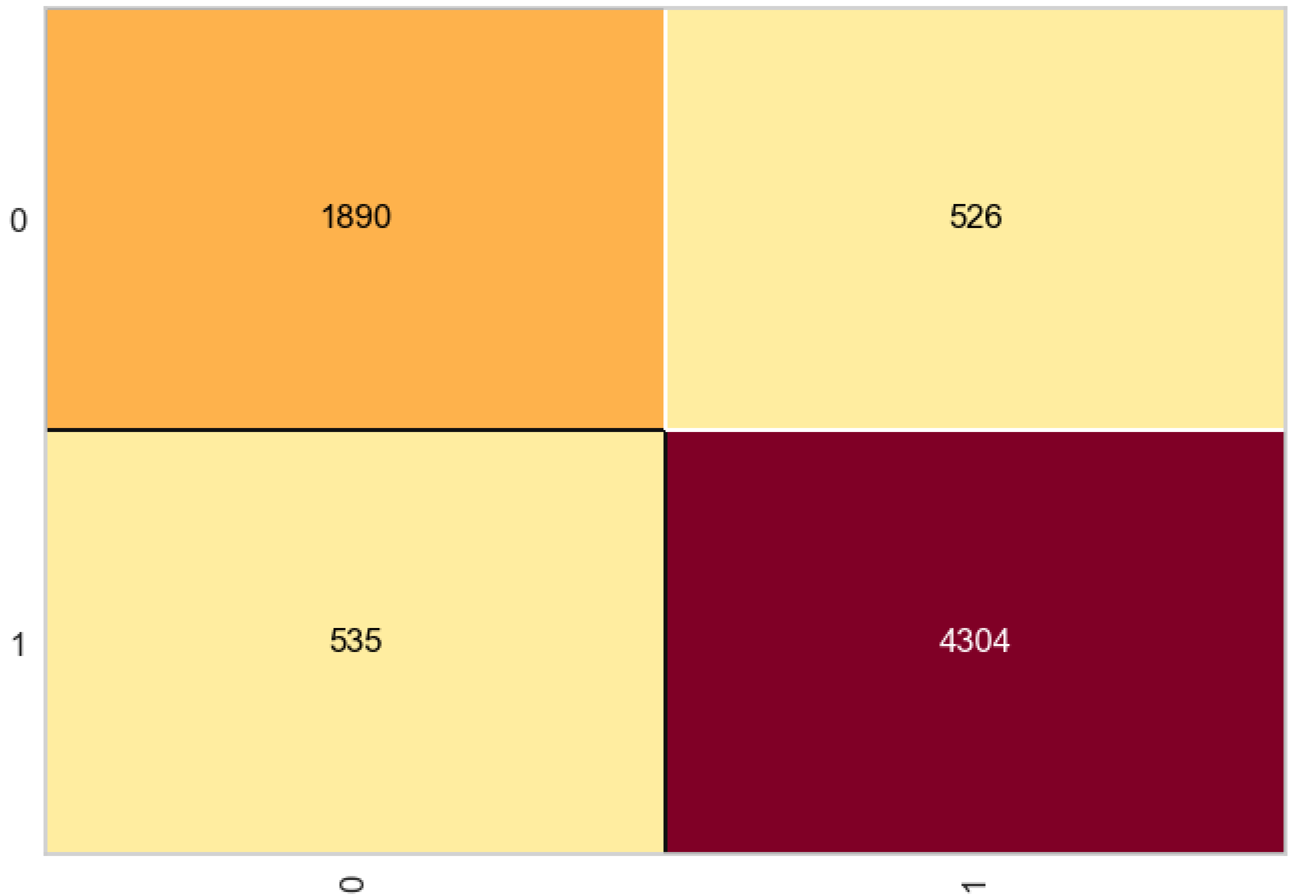
	neighbors	accuracy	precision	recall	f1	roc_auc
0	1	0.847636	0.892689	0.879110	0.885832	0.831078
2	3	0.834045	0.903133	0.843678	0.872387	0.884430
4	5	0.832143	0.910208	0.832483	0.869609	0.899706
6	7	0.830737	0.912086	0.828102	0.868060	0.905474
8	9	0.826823	0.912448	0.821240	0.864439	0.908474
5	6	0.815245	0.924263	0.789960	0.851846	0.902840
1	2	0.814087	0.922032	0.790329	0.851110	0.865962
7	8	0.813728	0.921827	0.789958	0.850805	0.907307
3	4	0.812984	0.925190	0.785368	0.849559	0.894269

In [12]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=1)
```

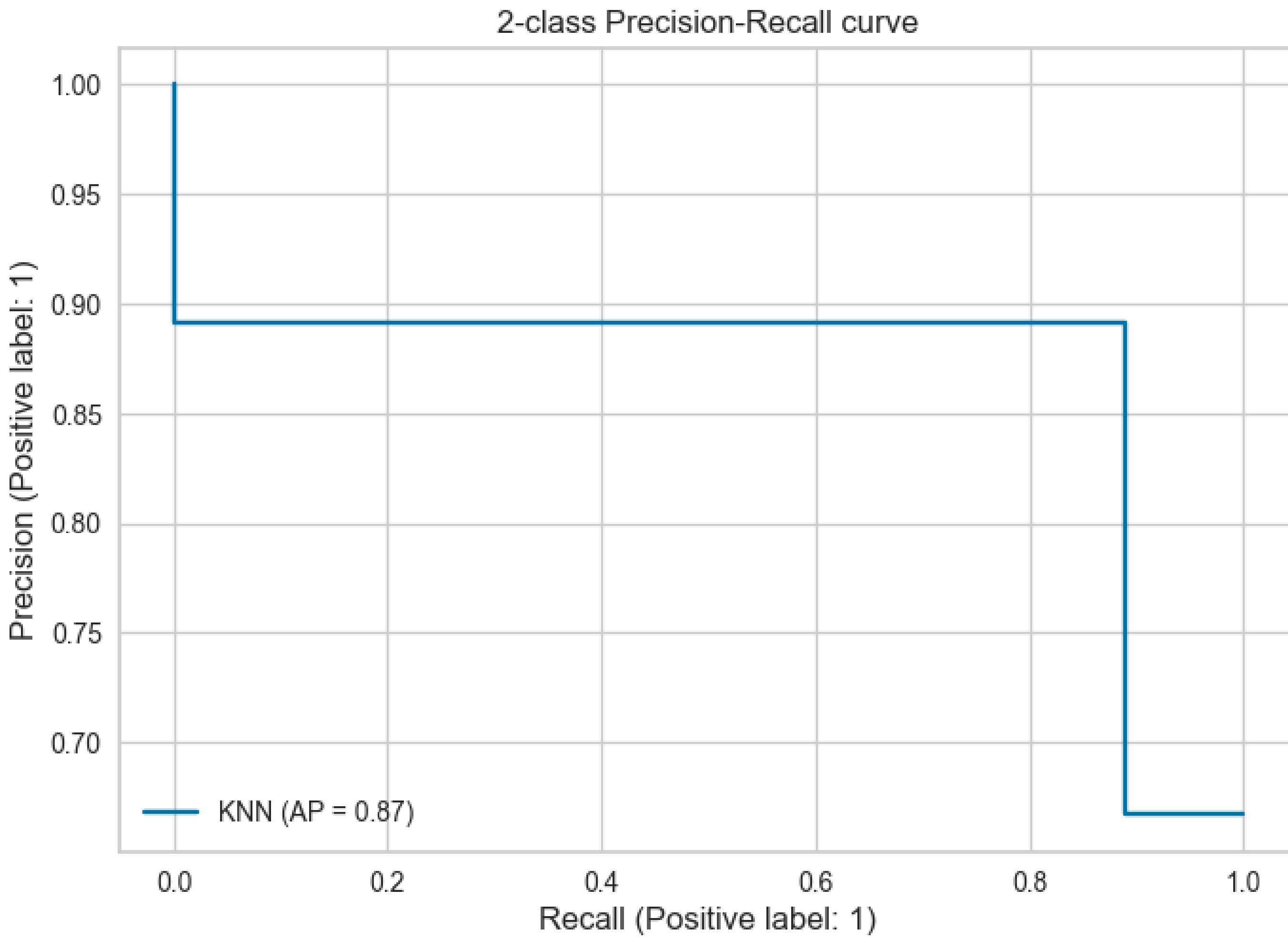
In [ ]:
```
knn_scores = cross_validate(knn_classifier, X, y, scoring=scoring, cv=kf)
```

In [155…
```
knn_classifier = KNeighborsClassifier(n_neighbors=1)

cm = ConfusionMatrix(knn_classifier)
cm.fit(x_train_balanced, y_train_balanced)
cm.score(x_test, y_test)
```

Out[155]:
```
0.8537560303239146
```



In [13]:
```
knn_classifier.fit(x_train, y_train)
display = PrecisionRecallDisplay.from_estimator(
    knn_classifier, x_test, y_test, name="KNN"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [14]:
```
y_pred = knn_classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.78      0.78      0.78      2416
Not Canceled       0.89      0.89      0.89      4839

    accuracy                           0.85      7255
   macro avg       0.84      0.84      0.84      7255
weighted avg       0.85      0.85      0.85      7255
```

Neural networks

In [10]:
```
from keras import backend as K

def check_units(y_true, y_pred):
    if y_pred.shape[1] != 1:
      y_pred = y_pred[:,1:2]
      y_true = y_true[:,1:2]
    return y_true, y_pred

def precision(y_true, y_pred):
    y_true, y_pred = check_units(y_true, y_pred)
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def recall(y_true, y_pred):
```

```python
    y_true, y_pred = check_units(y_true, y_pred)
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    y_true, y_pred = check_units(y_true, y_pred)
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

In [13]:
```python
from keras.utils import np_utils

y_convert = np_utils.to_categorical(y)
X_standard = StandardScaler().fit_transform(X)

# train using Neural Network
from tensorflow.keras.callbacks import CSVLogger
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.metrics import AUC, Precision, Recall, CategoricalAccuracy

metrics = [precision, recall, f1, AUC(), CategoricalAccuracy()]

def test_model(optimizer, learning_rate):
    accuracies = []
    precisions = []
    recalls = []
    roc_aucs = []

    # implement k-fold cross validation
    fold_no = 1
    for train, test in kf.split(X_standard, y_convert):
        train_data_x, train_data_y = SMOTE(random_state=100).fit_resample(X_standard[tra
        train_data_y = np_utils.to_categorical(train_data_y)

        model = Sequential()
        model.add(Dense(32, input_dim = 17, kernel_initializer = 'uniform', activation =
        model.add(Dropout(0.2))
        model.add(Dense(64, kernel_initializer = 'uniform', activation = 'relu'))
        model.add(Dropout(0.2))
        model.add(Dense(128, kernel_initializer = 'uniform', activation = 'relu'))
        model.add(Dropout(0.2))
        model.add(Dense(256, kernel_initializer = 'uniform', activation = 'relu'))
        model.add(Dropout(0.2))
        model.add(Dense(2, kernel_initializer = 'uniform', activation = 'softmax'))

        to_add_optimizer = optimizer(learning_rate=learning_rate)

        logger = CSVLogger(f'rn_stats\\{to_add_optimizer._name}_{str(learning_rate)}_{st
```

```
        model.compile(loss='binary_crossentropy', optimizer = to_add_optimizer, metrics=
        history = model.fit(train_data_x, train_data_y, epochs = 50, batch_size = 600, v
        scores = model.evaluate(X_standard[test], y_convert[test], verbose=0)

        print('Fold: ', fold_no)
        accuracies.append(scores[3])
        precisions.append(scores[1])
        recalls.append(scores[2])
        roc_aucs.append(scores[4])
        fold_no = fold_no + 1

    return tuple([np.mean(accuracies), np.mean(precisions), np.mean(recalls), np.mean(ro
```

In [14]:

```
# try different learning_rate
learning_rate = [0.001, 0.01, 0.02, 0.03, 0.1]
optimizers = [Adam, SGD, RMSprop]

histories = []
final_results = []

for optimizer in optimizers:
    for lr in learning_rate:
        print("Optimizer: ", optimizer.__name__)
        print("Learning rate: ", lr)
        results, history = test_model(optimizer, lr)
        final_results.append((optimizer.__name__, lr, *results))
        histories.append(history)
        print("Accuracy: ", results[0])
        print("Precision: ", results[1])
        print("Recall: ", results[2])
        print("ROC AUC: ", results[3])
        print("")
```

```
Optimizer:  Adam
Learning rate:  0.001
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.8821262717247009
Precision:  0.9142017483711242
Recall:  0.857167637348175
ROC AUC:  0.9239423155784607

Optimizer:  Adam
Learning rate:  0.01
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.8796829342842102
Precision:  0.9097523808479309
Recall:  0.856605613231659
ROC AUC:  0.9215418100357056

Optimizer:  Adam
Learning rate:  0.02
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
```

```
Accuracy:  0.8818481087684631
Precision:  0.9006226778030395
Recall:  0.8693264842033386
ROC AUC:  0.9209715485572815

Optimizer:  Adam
Learning rate:  0.03
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.8740932583808899
Precision:  0.8984219551086425
Recall:  0.8567670941352844
ROC AUC:  0.909711217880249

Optimizer:  Adam
Learning rate:  0.1
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.5682575702667236
Precision:  0.7003242254257203
Recall:  0.49581193923950195
ROC AUC:  0.6239896535873413

Optimizer:  SGD
Learning rate:  0.001
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.5807163376361132
Precision:  0.6652085423469544
Recall:  0.5880172632634639
ROC AUC:  0.5

Optimizer:  SGD
Learning rate:  0.01
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.7400710940361023
Precision:  0.8350904703140258
Recall:  0.6840677499771118
ROC AUC:  0.5

Optimizer:  SGD
Learning rate:  0.02
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.6028191208839416
Precision:  0.8401505351066589
Recall:  0.5674682319164276
ROC AUC:  0.5000275611877442

Optimizer:  SGD
```

```
Learning rate:  0.03
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.7297757267951965
Precision:  0.8518028855323792
Recall:  0.6809546709060669
ROC AUC:  0.5055759906768799

Optimizer:  SGD
Learning rate:  0.1
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.848986029624939
Precision:  0.8946496486663819
Recall:  0.8139476299285888
ROC AUC:  0.88998361825943

Optimizer:  RMSprop
Learning rate:  0.001
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.8857917189598083
Precision:  0.9012309432029724
Recall:  0.8762413382530212
ROC AUC:  0.9284339070320129

Optimizer:  RMSprop
Learning rate:  0.01
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.8759007930755616
Precision:  0.9114988327026368
Recall:  0.8492431044578552
ROC AUC:  0.9180360794067383

Optimizer:  RMSprop
Learning rate:  0.02
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.8754438281059265
Precision:  0.8998964071273804
Recall:  0.8578574538230896
ROC AUC:  0.9098972678184509

Optimizer:  RMSprop
Learning rate:  0.03
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
```

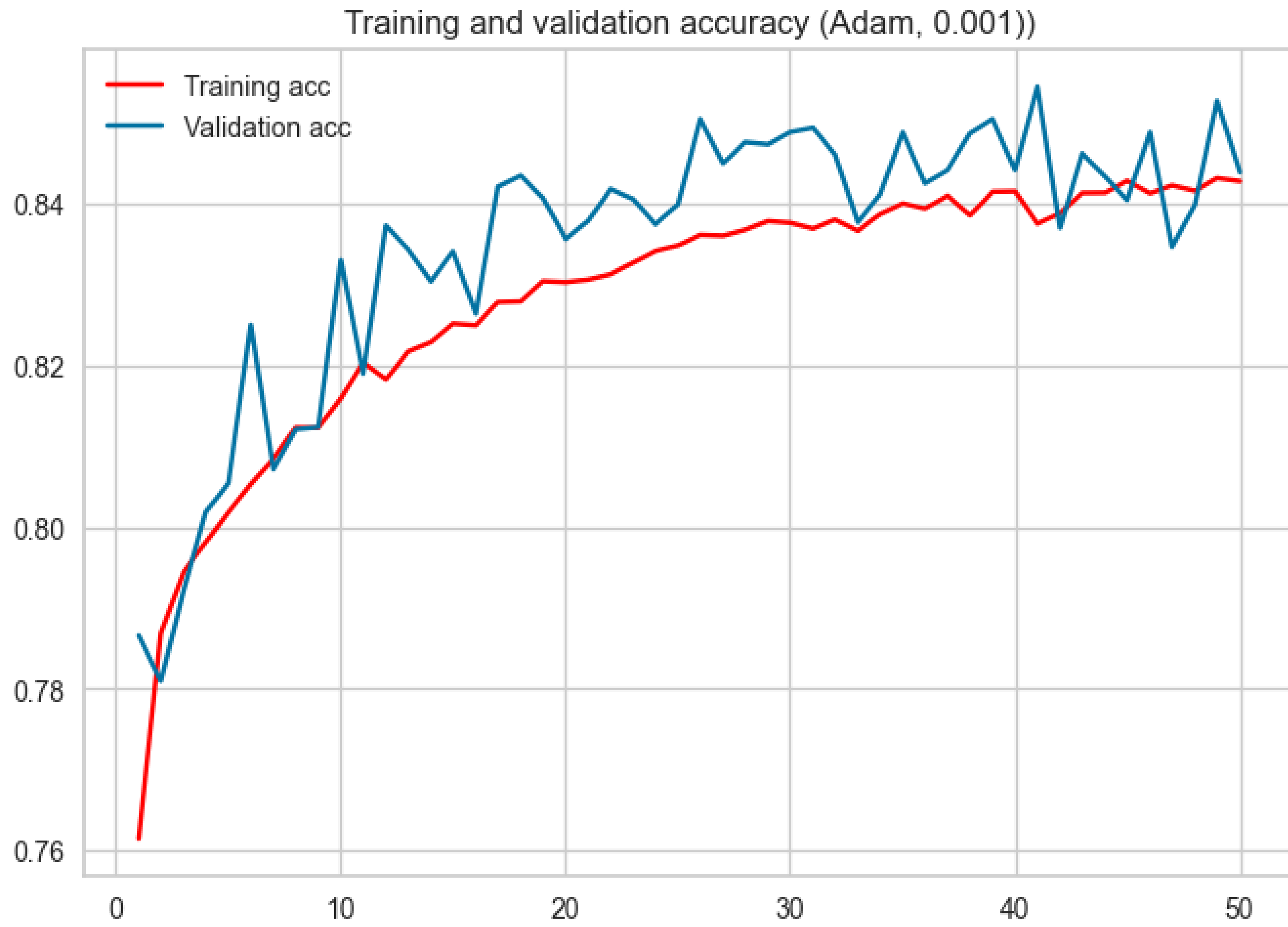
```
Accuracy:  0.876719868183136
Precision:  0.8850917816162109
Recall:  0.8746797919273377
ROC AUC:  0.9136934757232666

Optimizer:  RMSprop
Learning rate:  0.1
Fold:  1
Fold:  2
Fold:  3
Fold:  4
Fold:  5
Accuracy:  0.826692807674408
Precision:  0.8577847361564637
Recall:  0.8085249543190003
ROC AUC:  0.8036921620368958
```

In [45]:
```python
learning_rate = [0.001, 0.01, 0.02, 0.03, 0.1]
optimizers = [Adam, SGD, RMSprop]
```

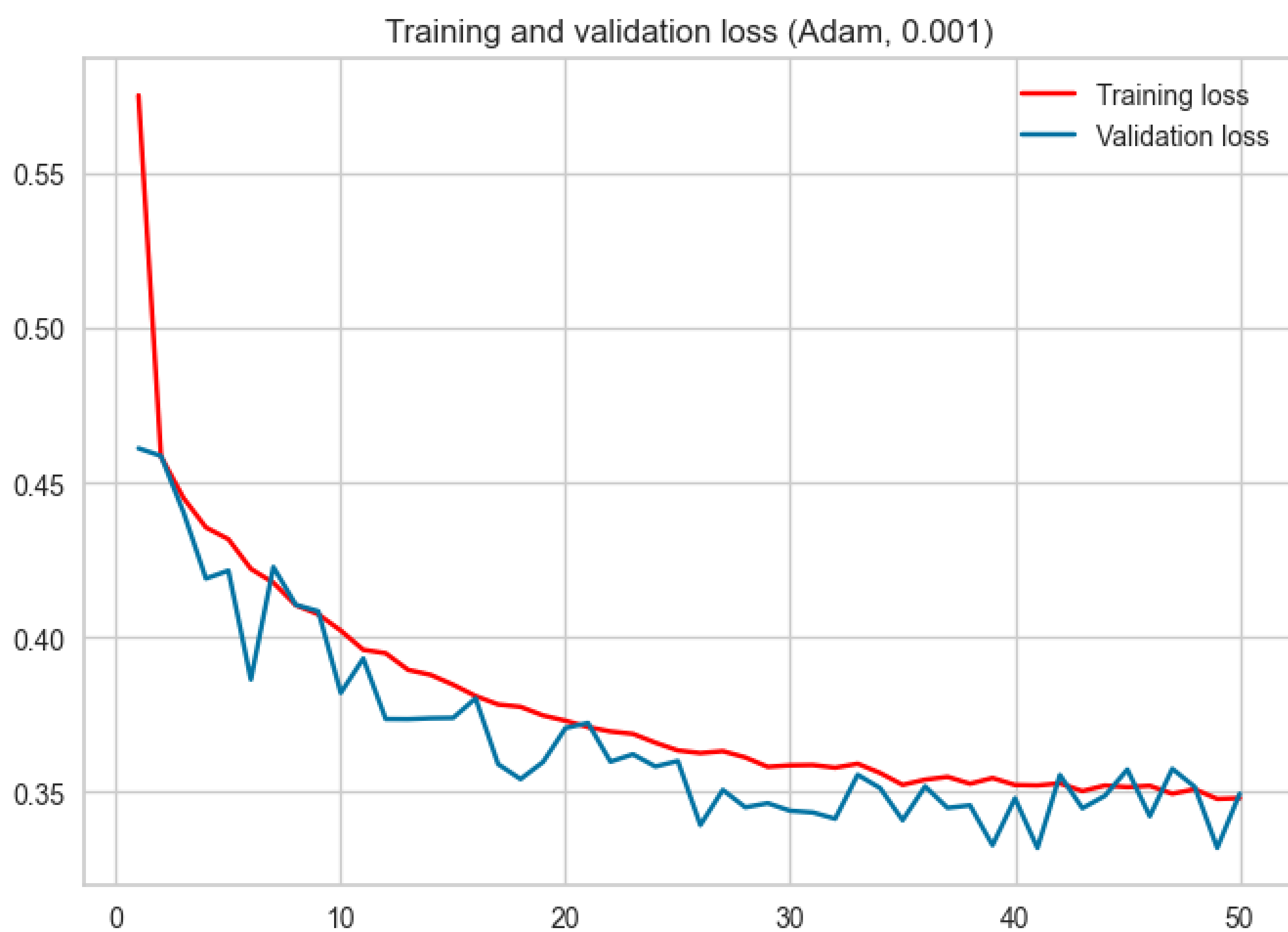
In [46]:
```python
# vizualize the results
def display_statistics(history, index):
    global learning_rate, optimizers
    optimizer_name = optimizers[index // len(learning_rate)].__name__
    learning_rate_index = str(learning_rate[index % len(learning_rate)])

    acc = history.history['categorical_accuracy']
    val_acc = history.history['val_categorical_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.plot(epochs, acc, 'b', label='Training acc', color='red')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title(f'Training and validation accuracy ({optimizer_name}, {learning_rate_index
    plt.legend()
    plt.savefig(f'rn-graph/{index}-acc.png')
    plt.figure()
    plt.plot(epochs, loss, 'b', label='Training loss', color='red')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title(f'Training and validation loss ({optimizer_name}, {learning_rate_index})')
    plt.legend()
    plt.show()
    plt.savefig(f'rn-graph/{index}-loss.png')
```

In [47]:
```python
for i in range(len(histories)):
    display_statistics(histories[i], i)
```

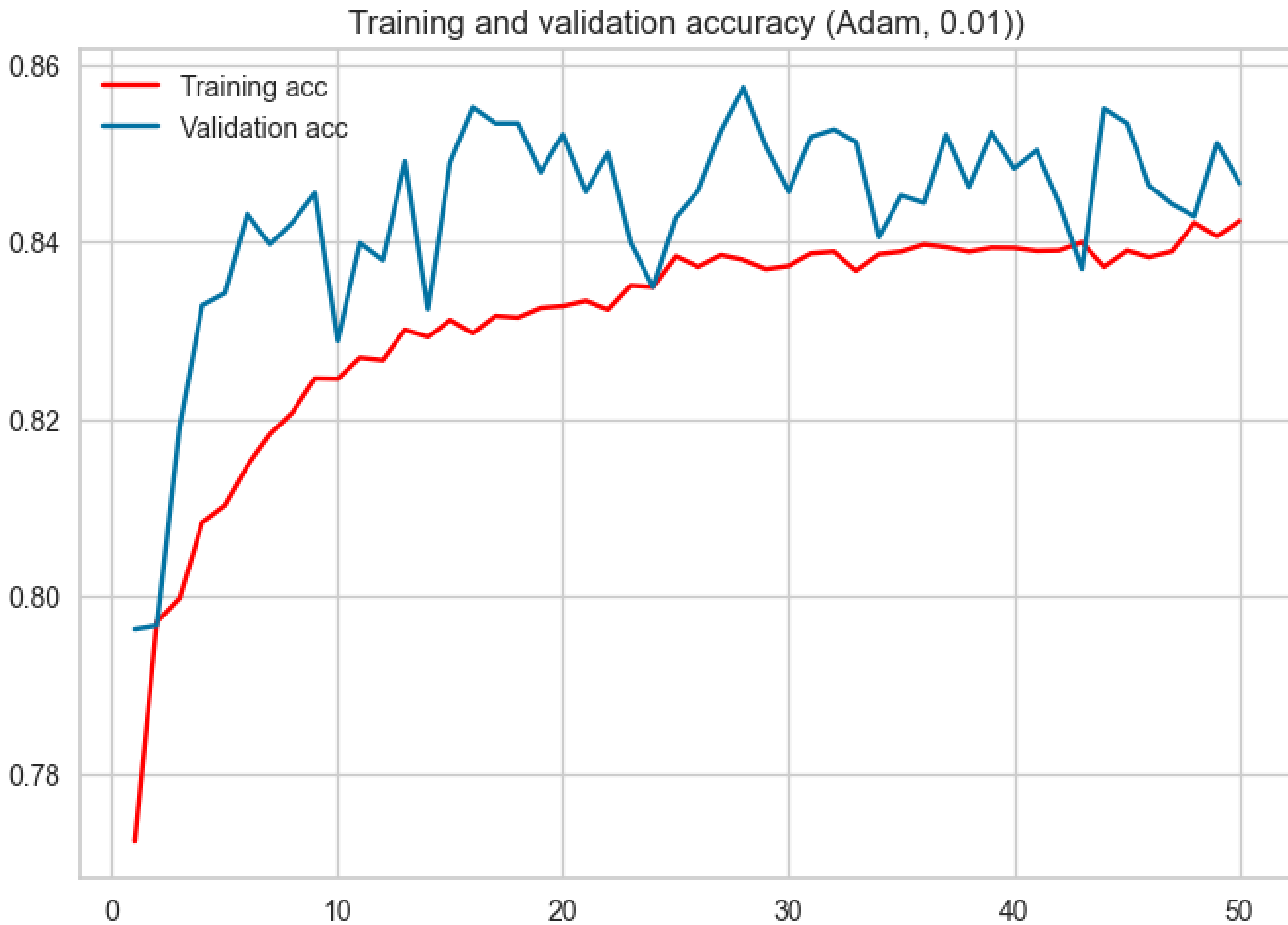
```
C:\Users\cezar\AppData\Local\Temp\ipykernel_11804\221855988.py:13: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string "b" (-> color=(0.
00784313725490196, 0.4470588235294118, 0.6352941176470588, 1)). The keyword argument wil
l take precedence.
  plt.plot(epochs, acc, 'b', label='Training acc', color='red')
C:\Users\cezar\AppData\Local\Temp\ipykernel_11804\221855988.py:19: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string "b" (-> color=(0.
00784313725490196, 0.4470588235294118, 0.6352941176470588, 1)). The keyword argument wil
l take precedence.
  plt.plot(epochs, loss, 'b', label='Training loss', color='red')
```

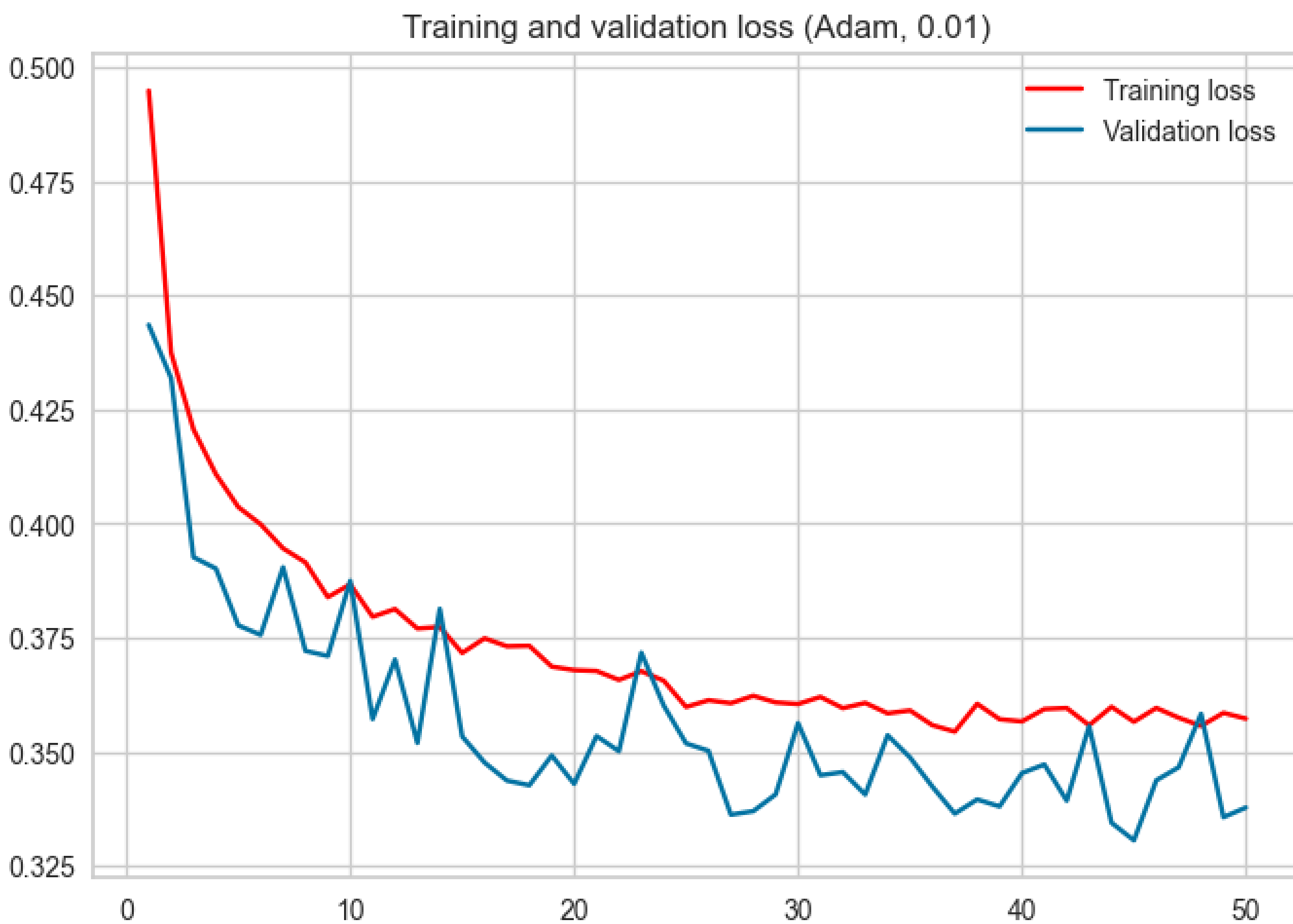


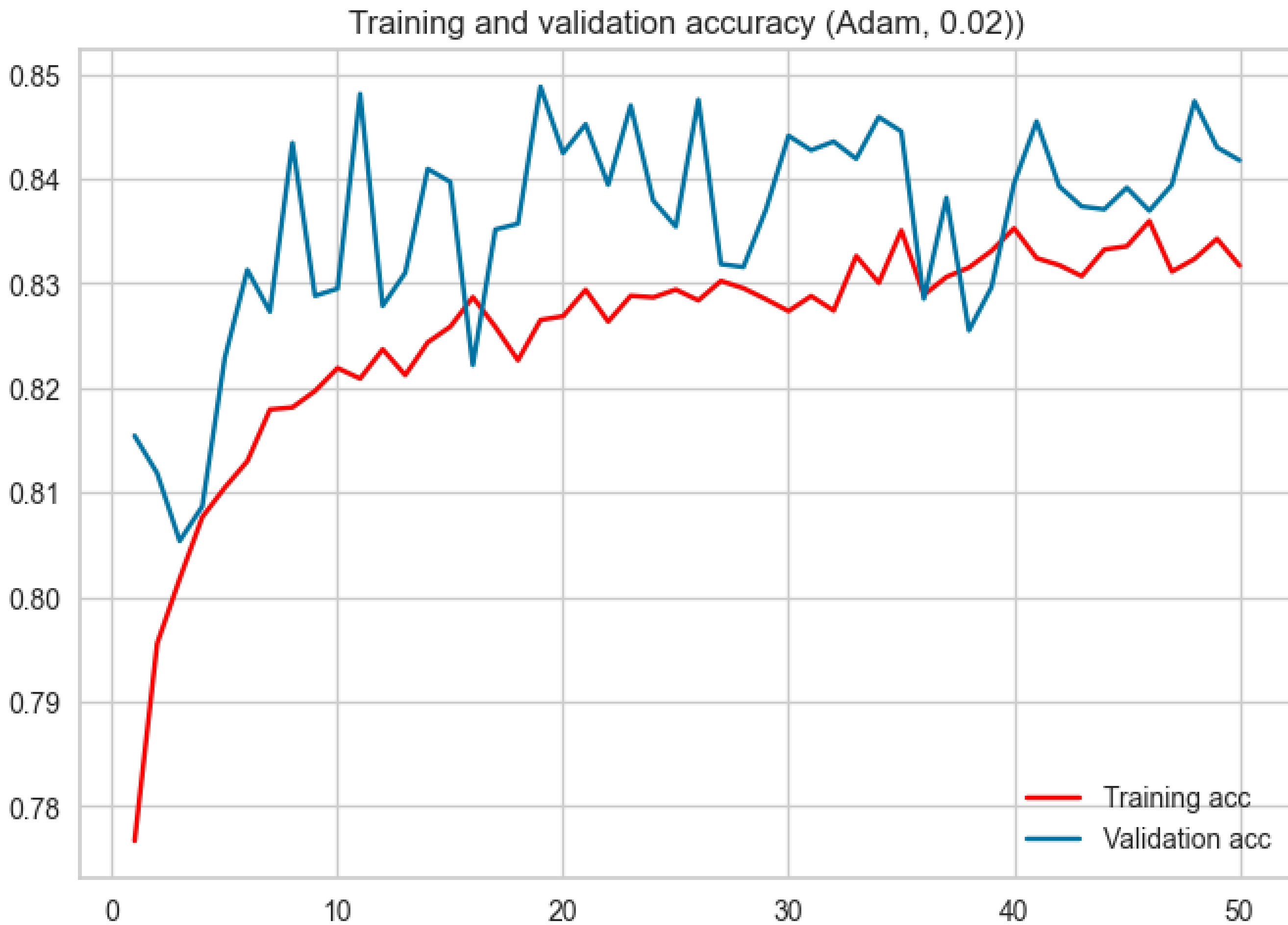
Training and validation accuracy (Adam, 0.001))
Training acc
Validation acc
0.84
0.82
0.80
0.78
0.76
0
10
20
30
40
50

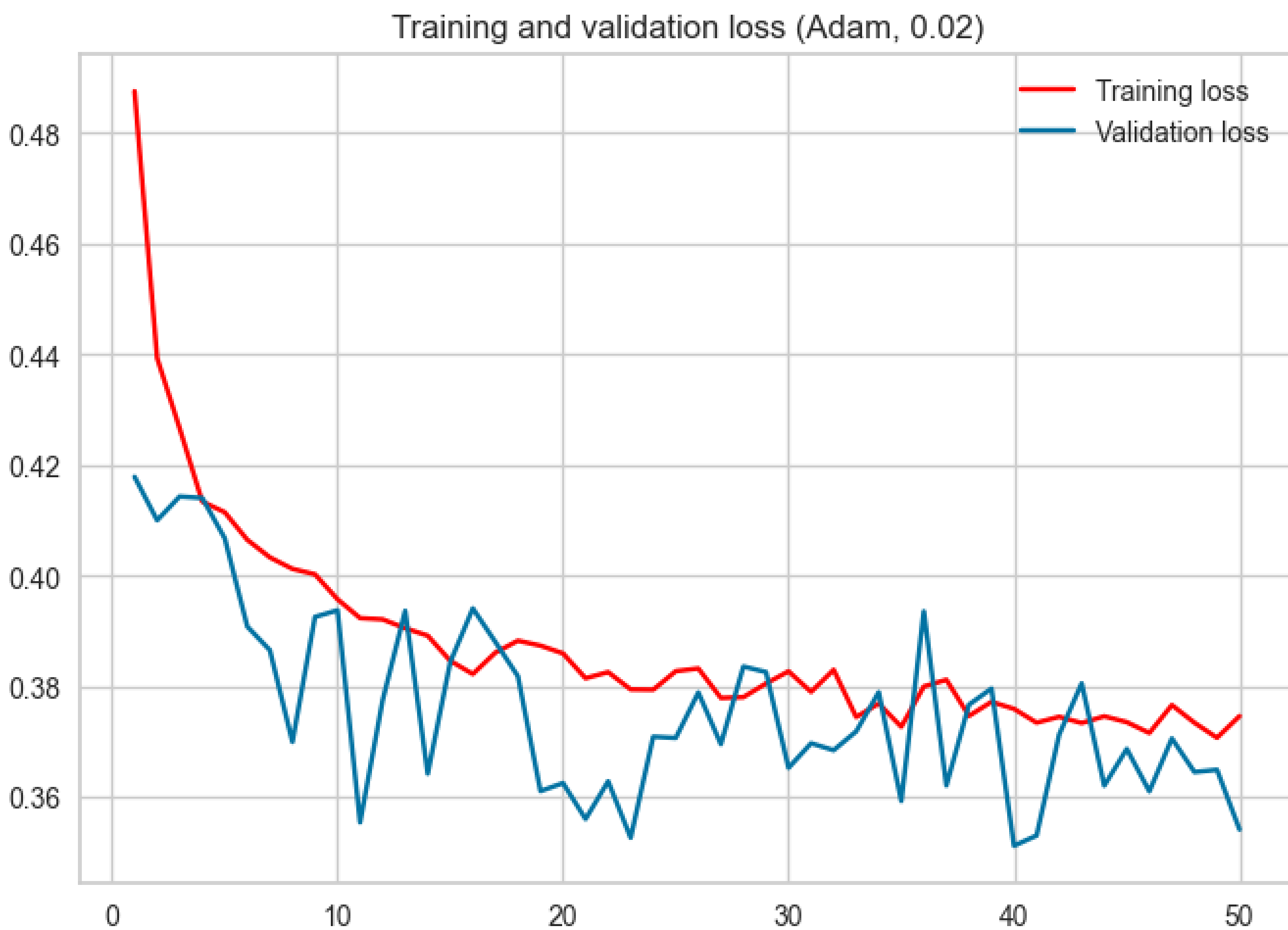


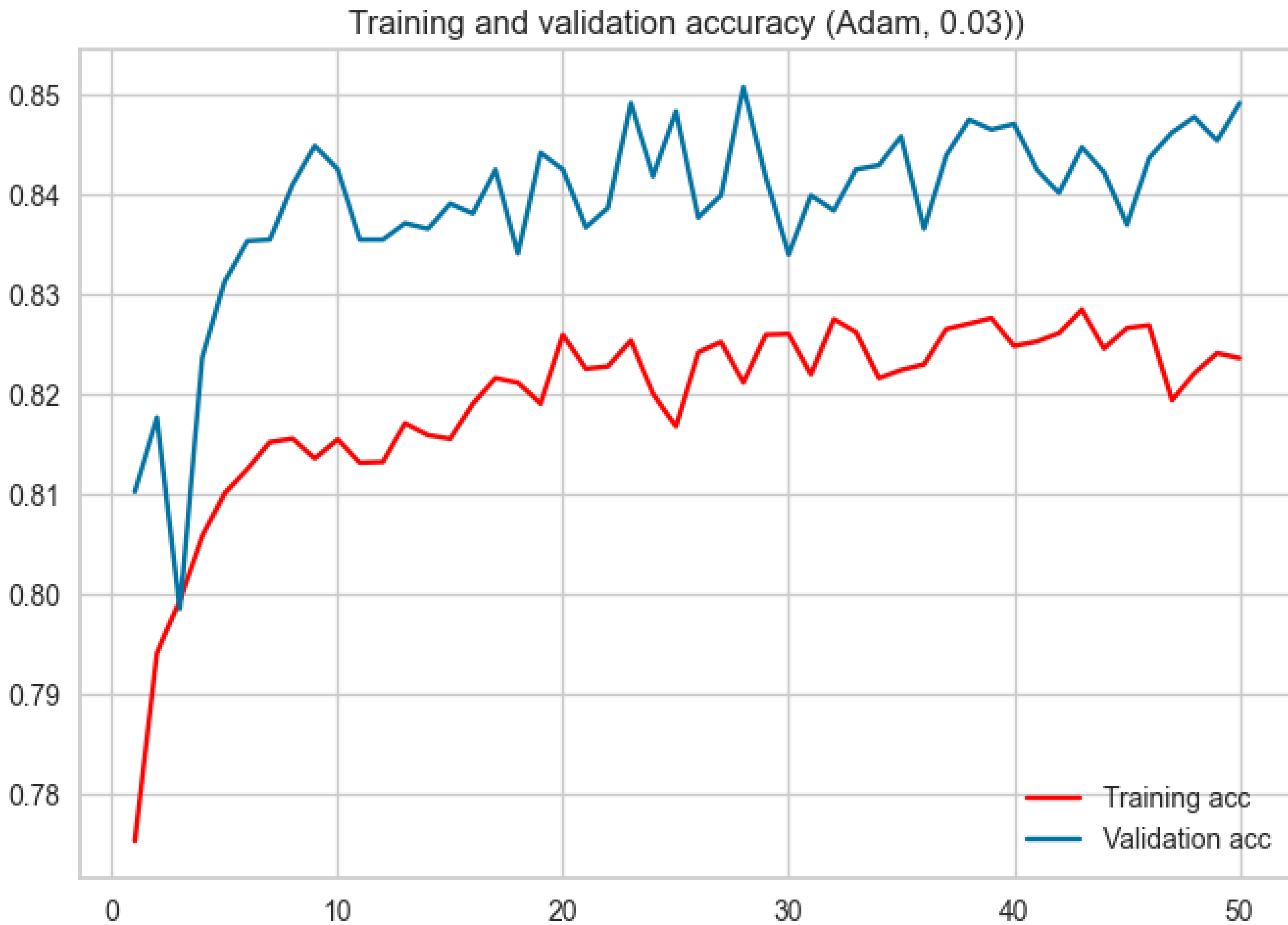
Training and validation loss (Adam, 0.001)
Training loss
Validation loss
0.55
0.50
0.45
0.40
0.35
0
10
20
30
40
50

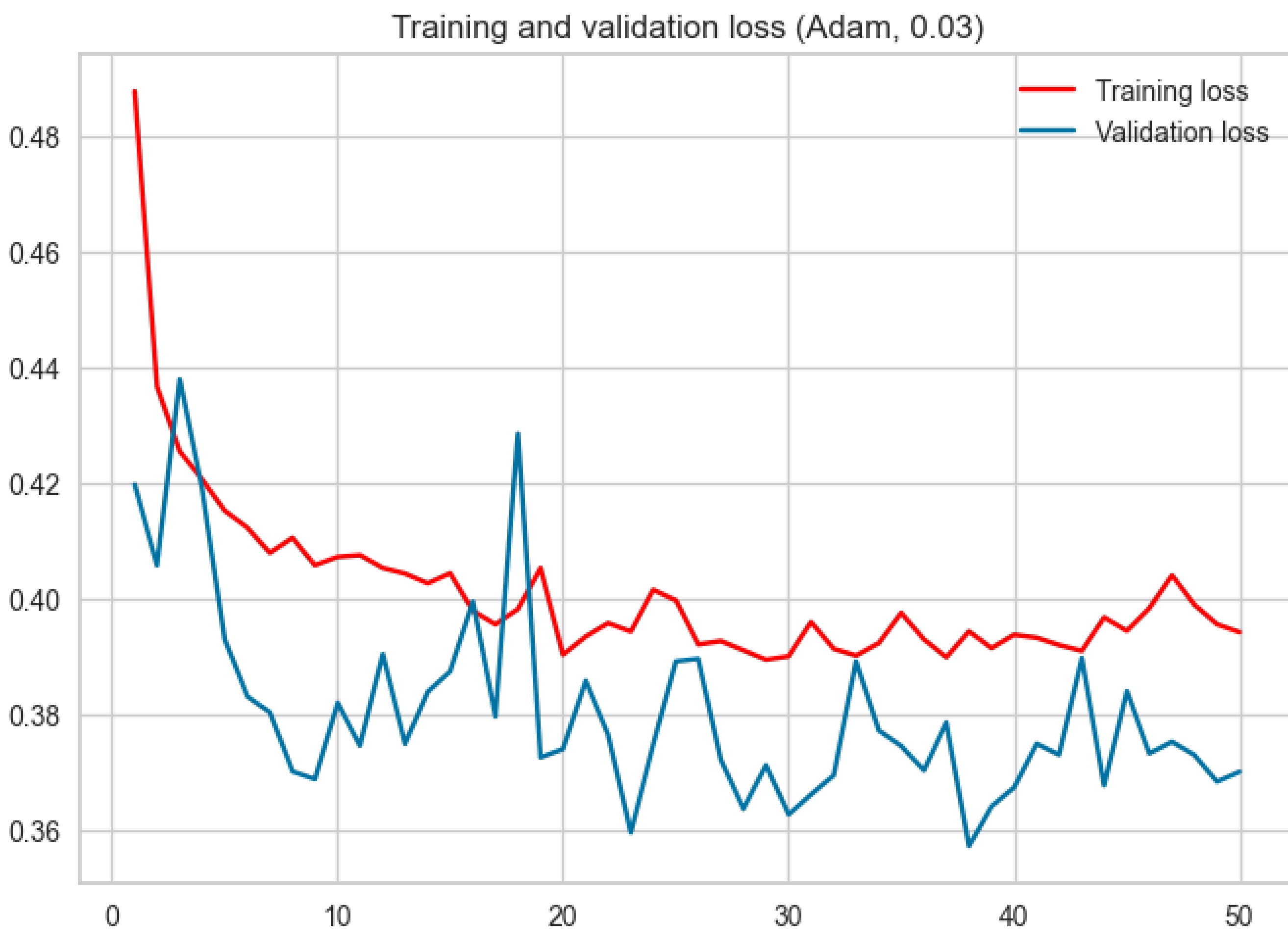

Training and validation accuracy (Adam, 0.01))
Training acc
Validation acc
0.86
0.84
0.82
0.80
0.78
0
10
20
30
40
50

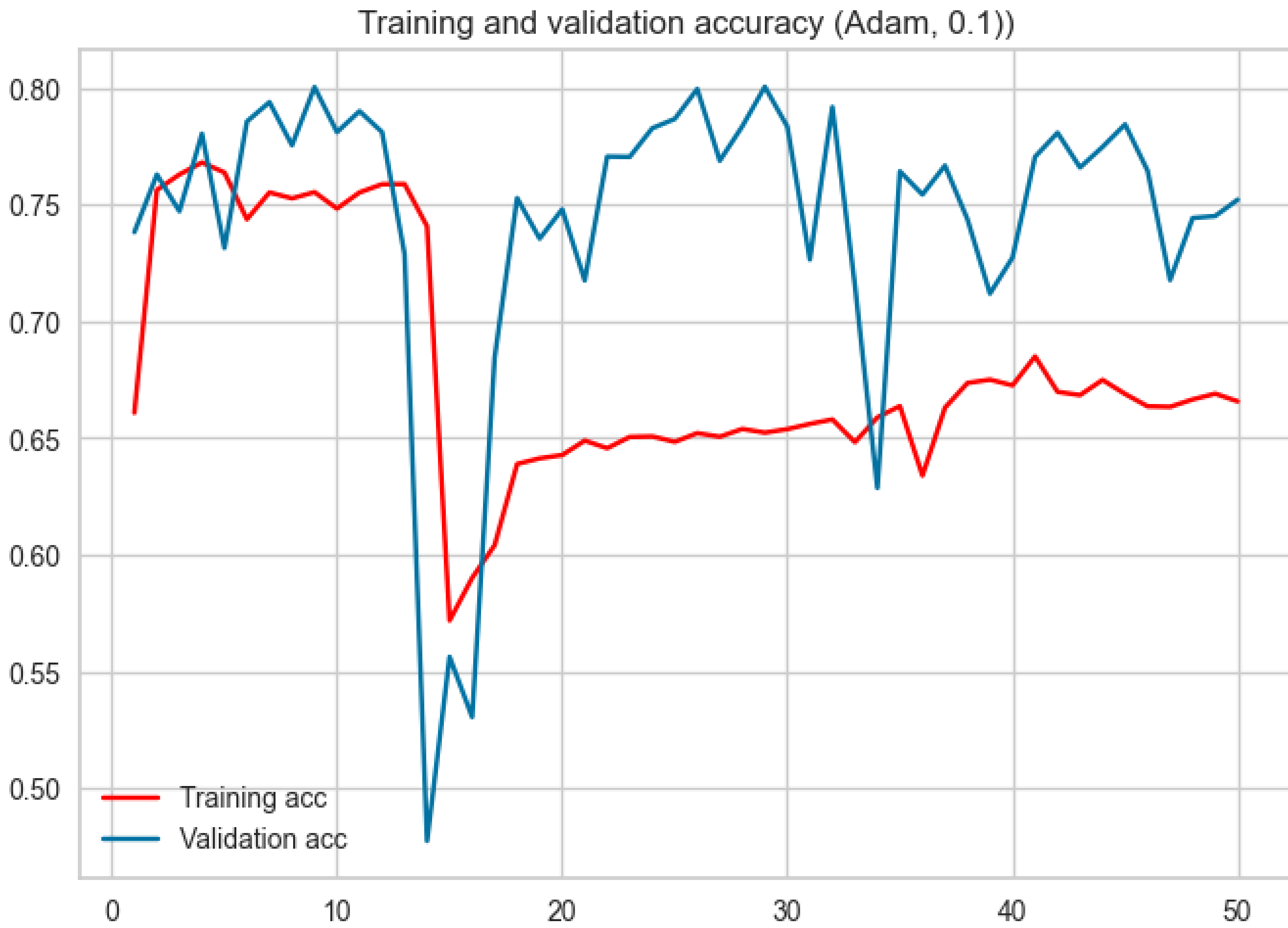

Training and validation loss (Adam, 0.01)
Training loss
Validation loss
0.500
0.475
0.450
0.425
0.400
0.375
0.350
0.325
0
10
20
30
40
50

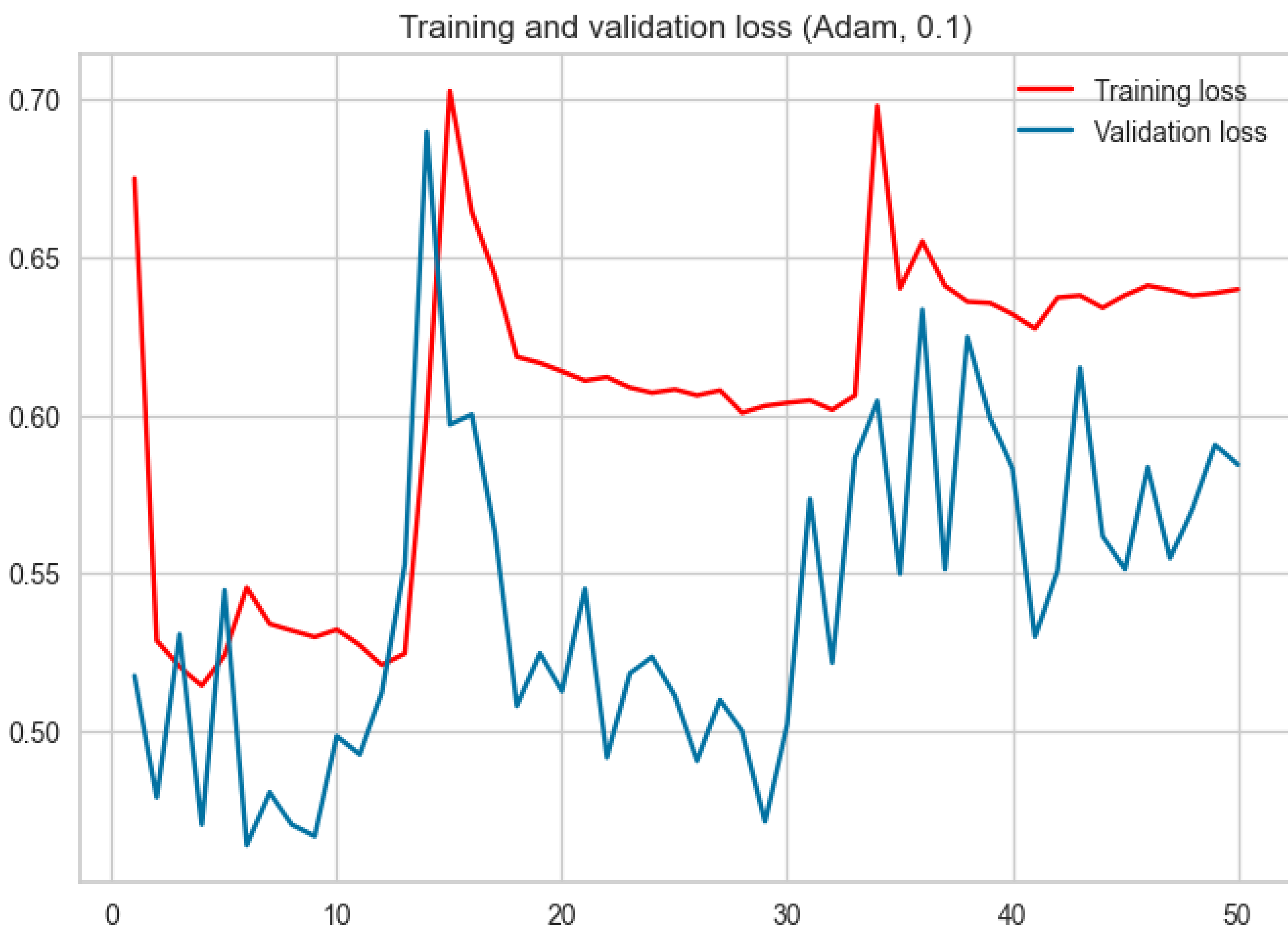

Training and validation accuracy (Adam, 0.02))
0.85
0.84
0.83
0.82
0.81
0.80
0.79
0.78
0
10
20
30
40
50
Training acc
Validation acc

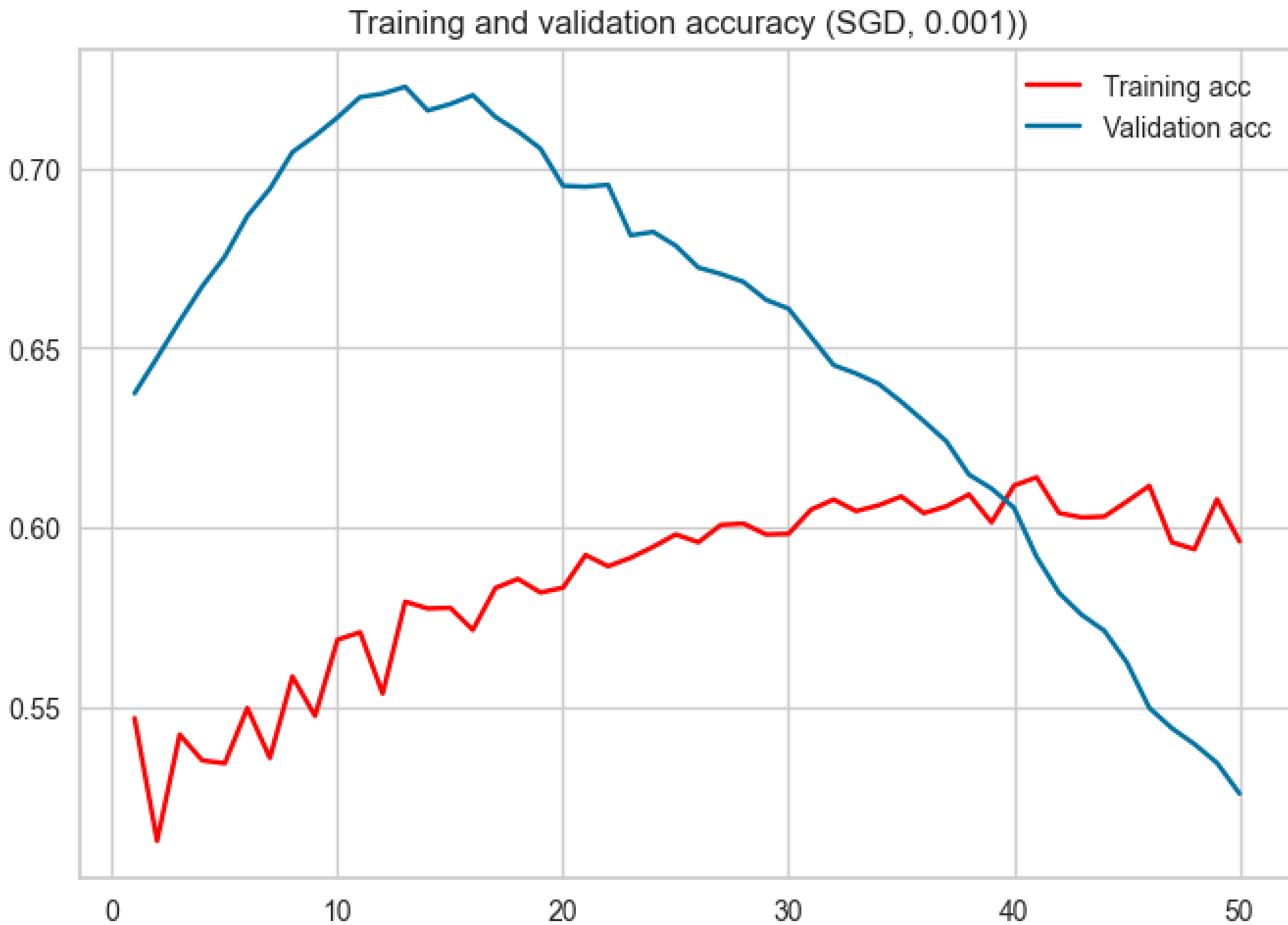

Training and validation loss (Adam, 0.02)
0.48
0.46
0.44
0.42
0.40
0.38
0.36
0
10
20
30
40
50
Training loss
Validation loss

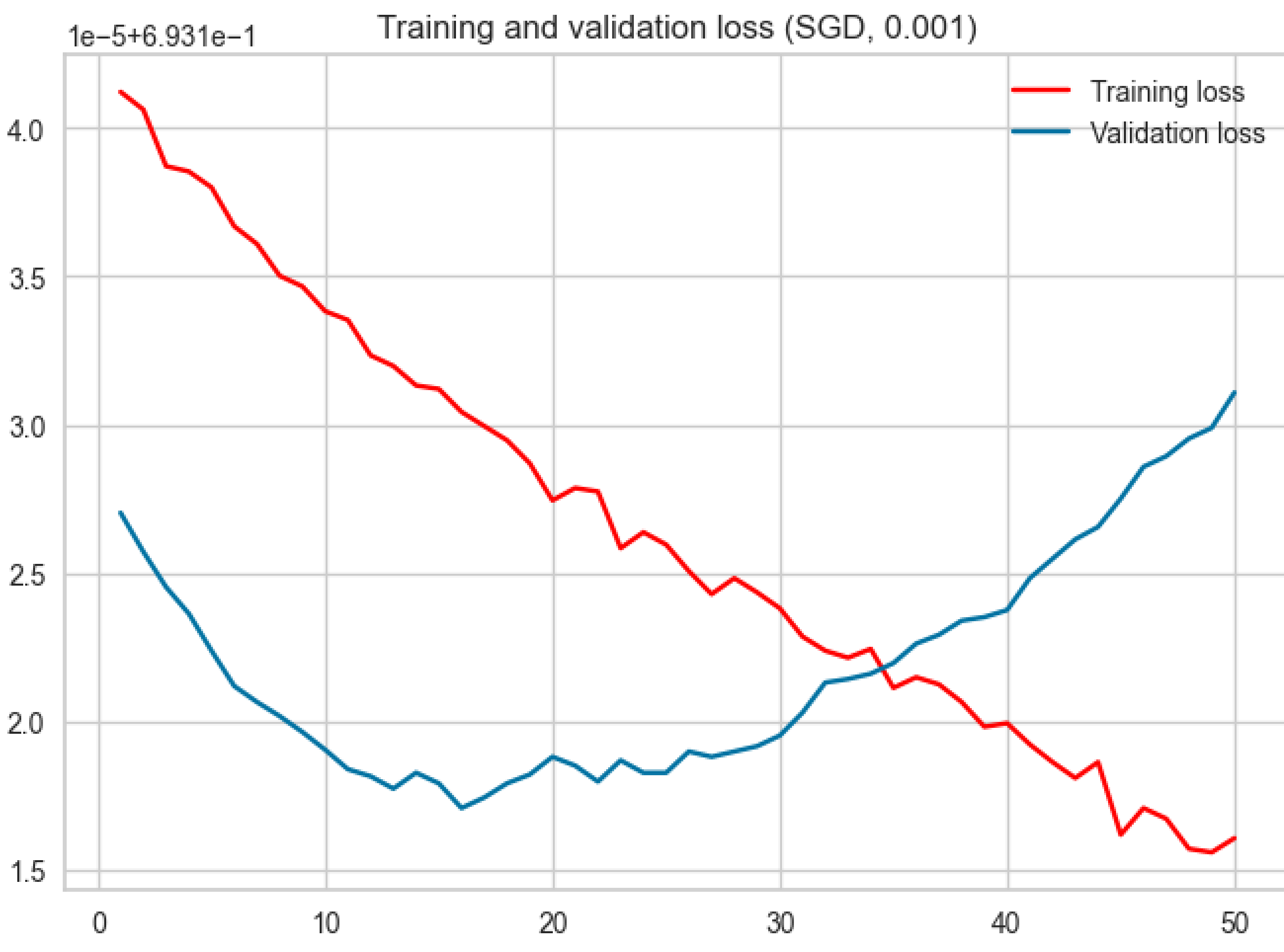

Training and validation accuracy (Adam, 0.03))
0.85
0.84
0.83
0.82
0.81
0.80
0.79
0.78
0
10
20
30
40
50
Training acc
Validation acc

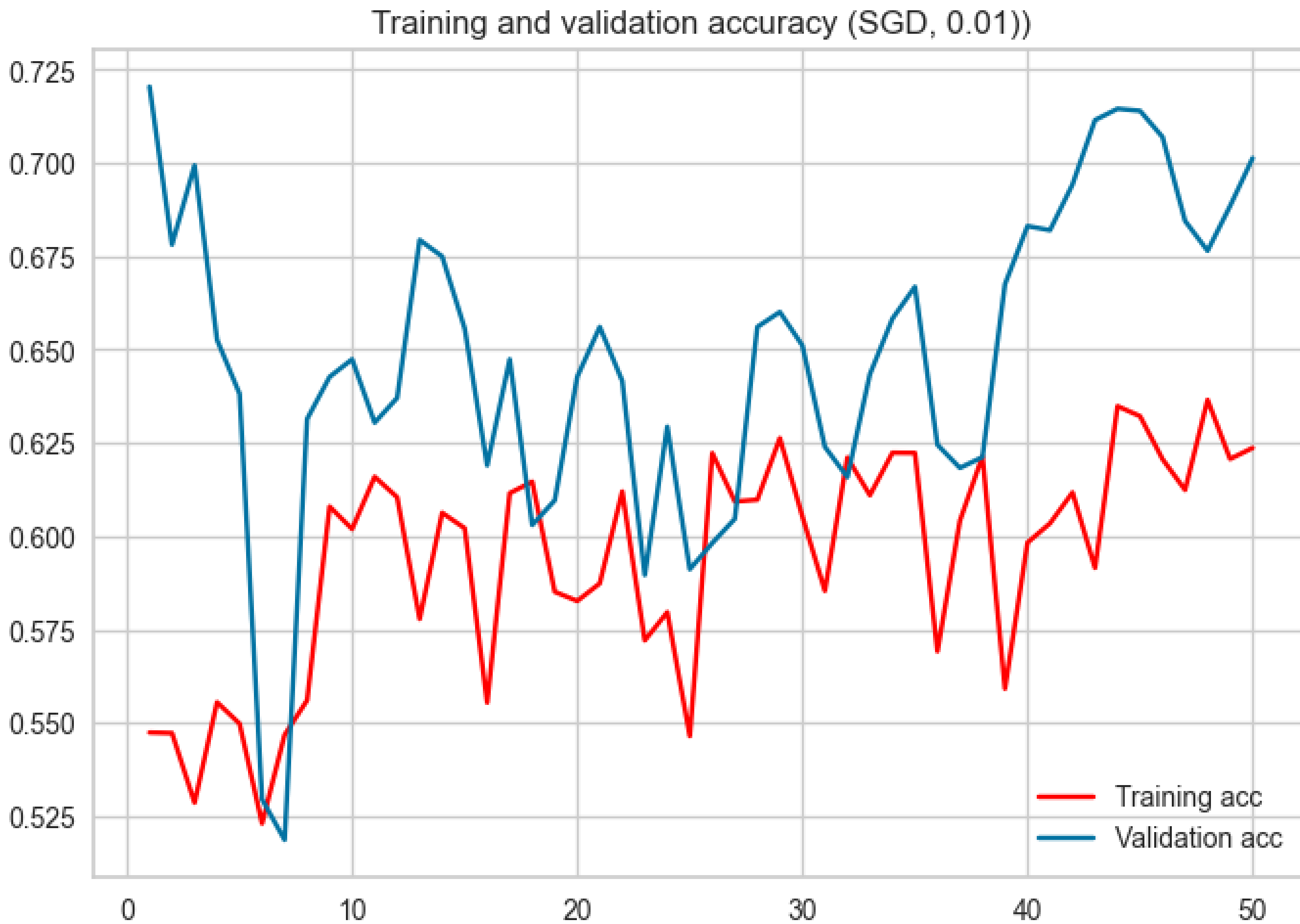

Training and validation loss (Adam, 0.03)
0.48
0.46
0.44
0.42
0.40
0.38
0.36
0
10
20
30
40
50
Training loss
Validation loss

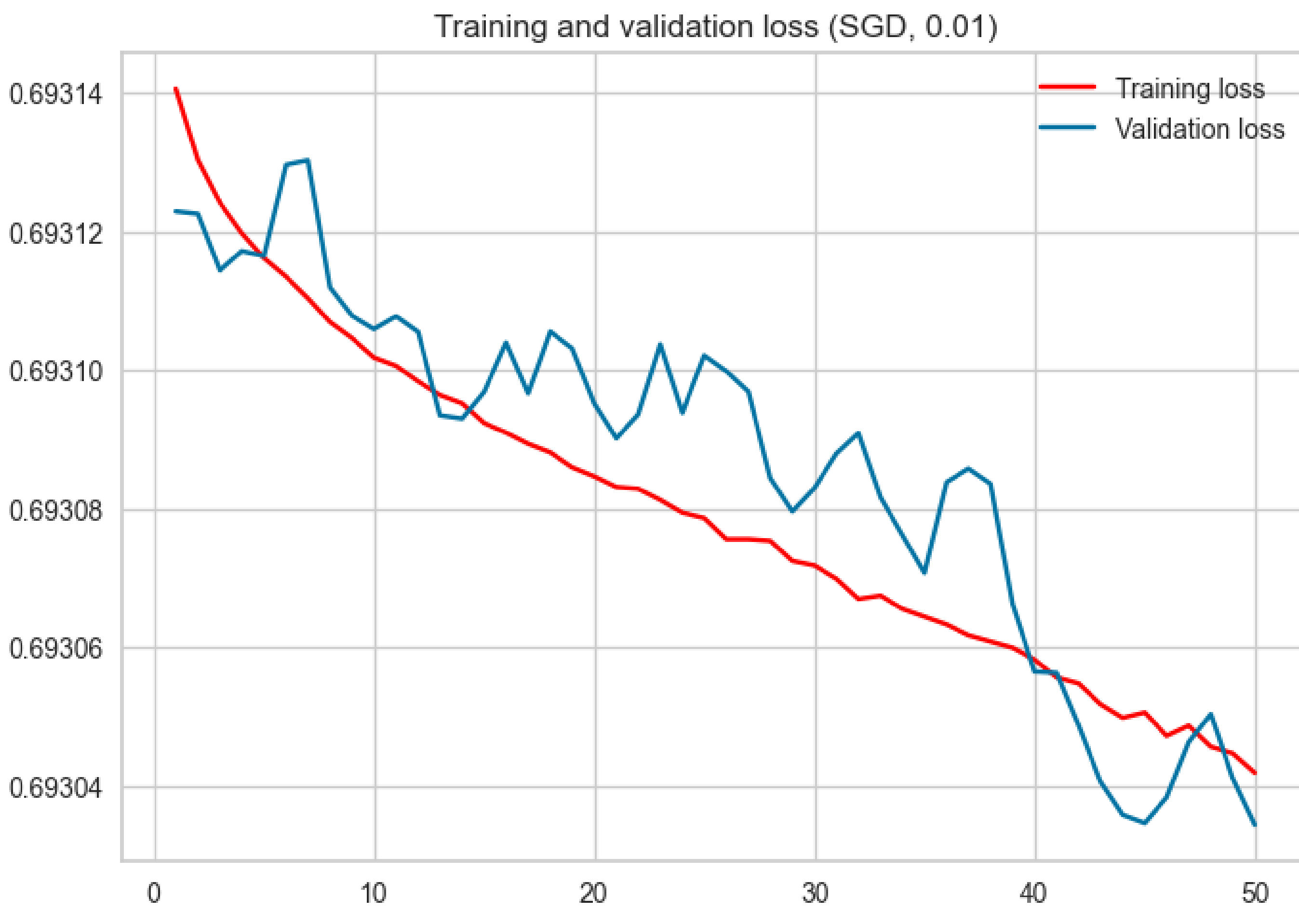

Training and validation accuracy (Adam, 0.1))
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0 10 20 30 40 50
Training acc
Validation acc

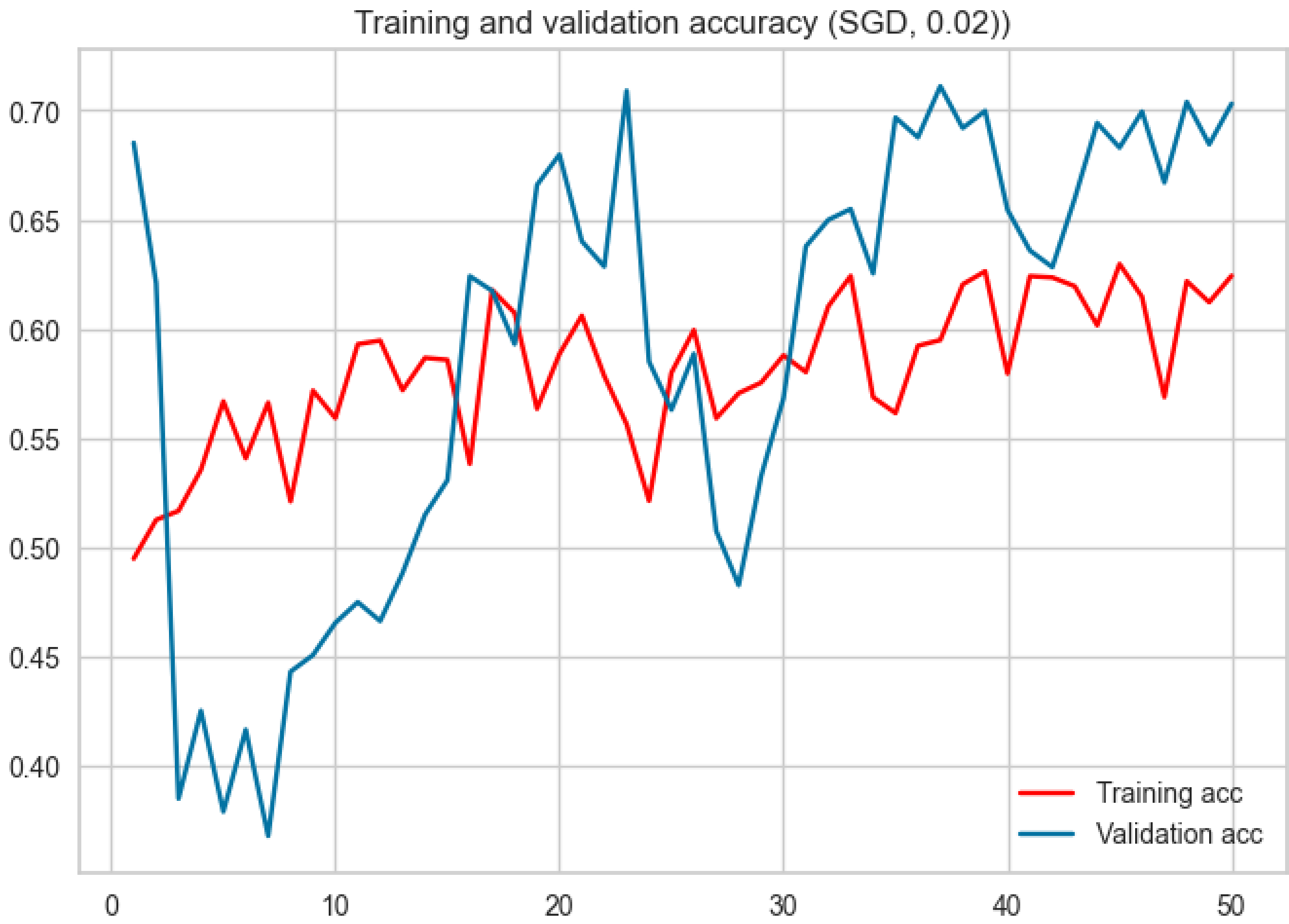

Training and validation loss (Adam, 0.1)
0.70
0.65
0.60
0.55
0.50
0 10 20 30 40 50
Training loss
Validation loss


Training and validation accuracy (SGD, 0.001))
Training acc
Validation acc
0.70
0.65
0.60
0.55
0
10
20
30
40
50

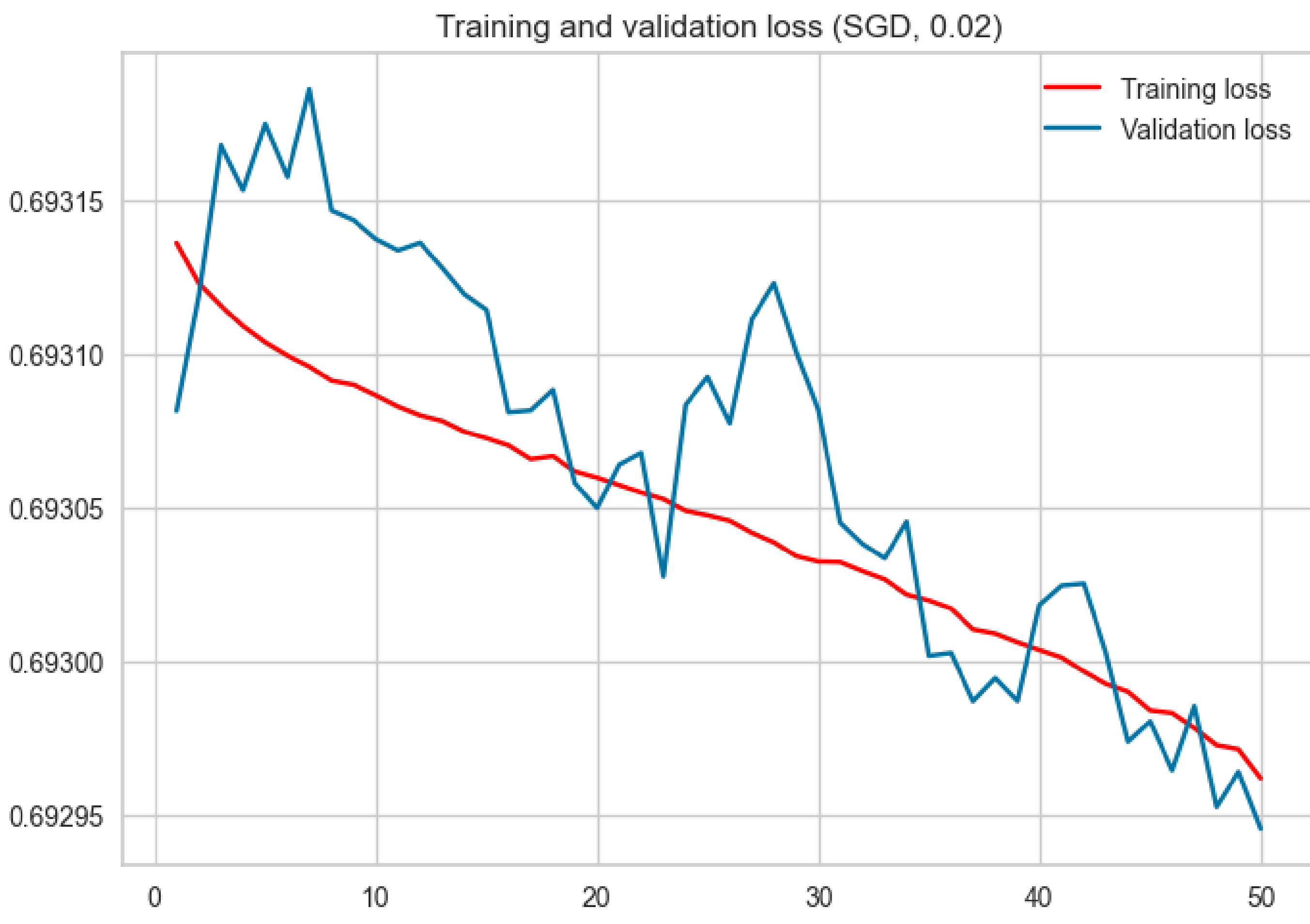

Training and validation loss (SGD, 0.001)
1e−5+6.931e−1
Training loss
Validation loss
4.0
3.5
3.0
2.5
2.0
1.5
0
10
20
30
40
50


Training and validation accuracy (SGD, 0.01))
0.725
0.700
0.675
0.650
0.625
0.600
0.575
0.550
0.525
0
10
20
30
40
50
Training acc
Validation acc

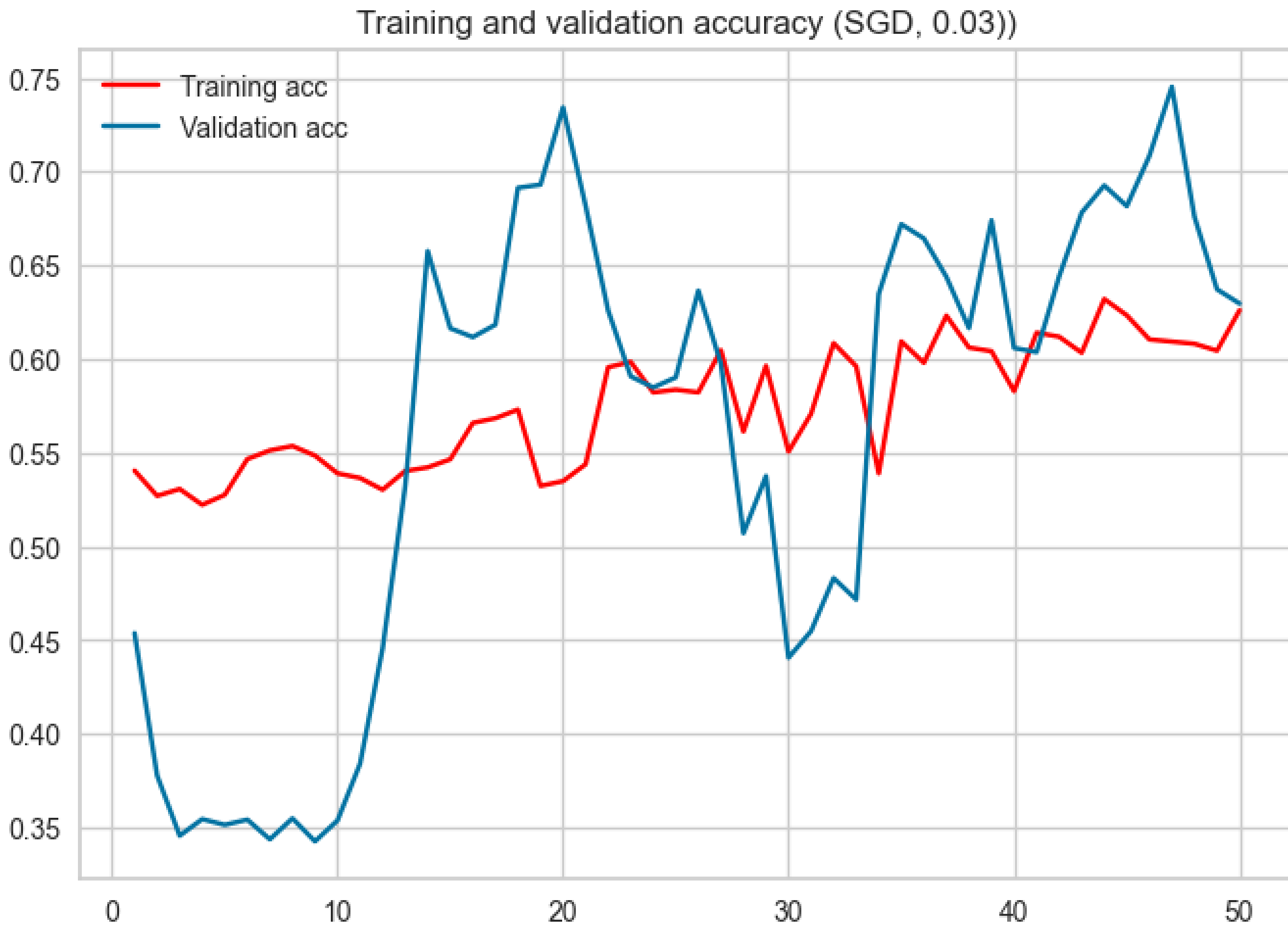

Training and validation loss (SGD, 0.01)
0.69314
0.69312
0.69310
0.69308
0.69306
0.69304
0
10
20
30
40
50
Training loss
Validation loss



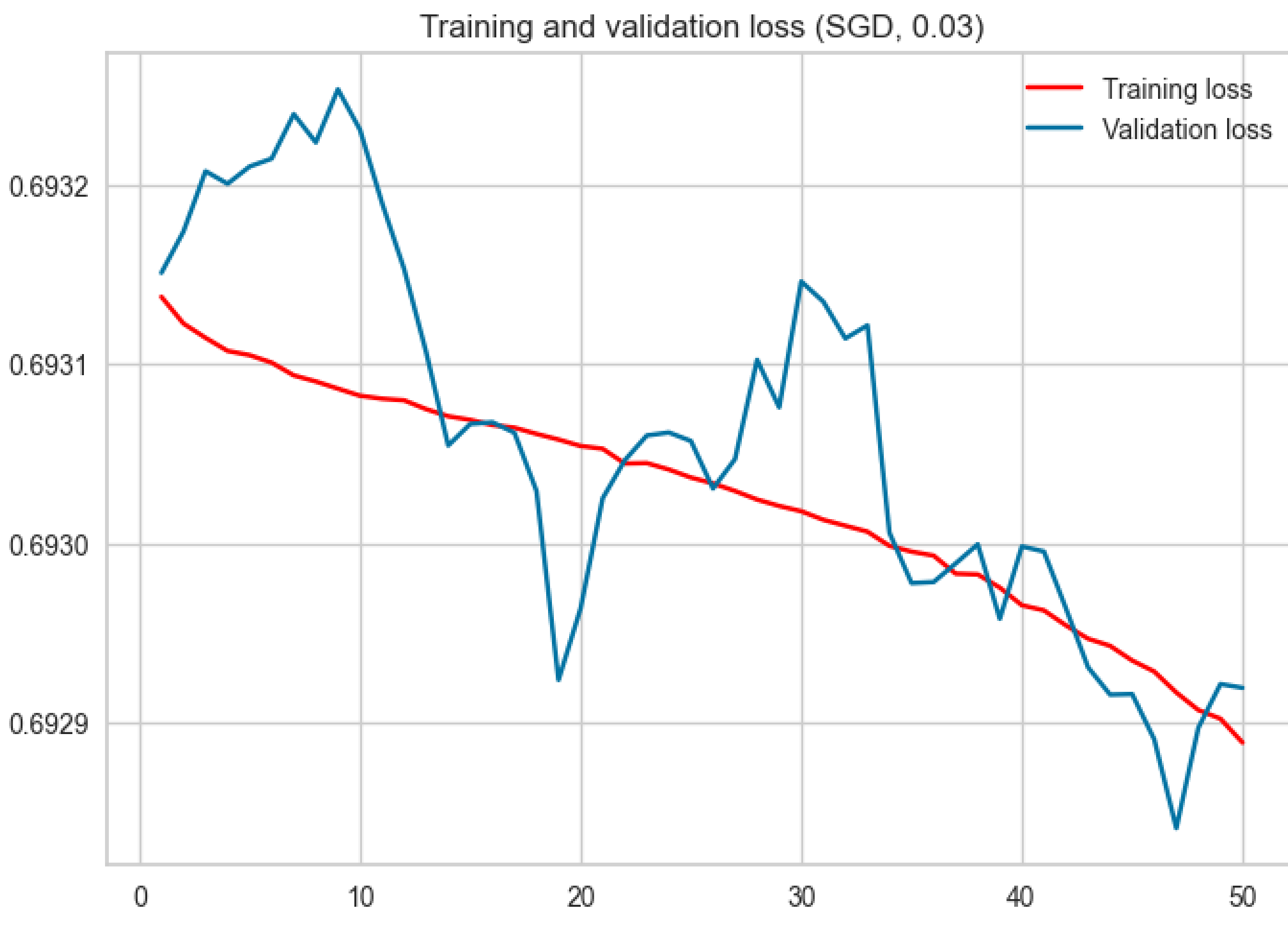
Training and validation accuracy (SGD, 0.02))
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0 10 20 30 40 50
Training acc
Validation acc



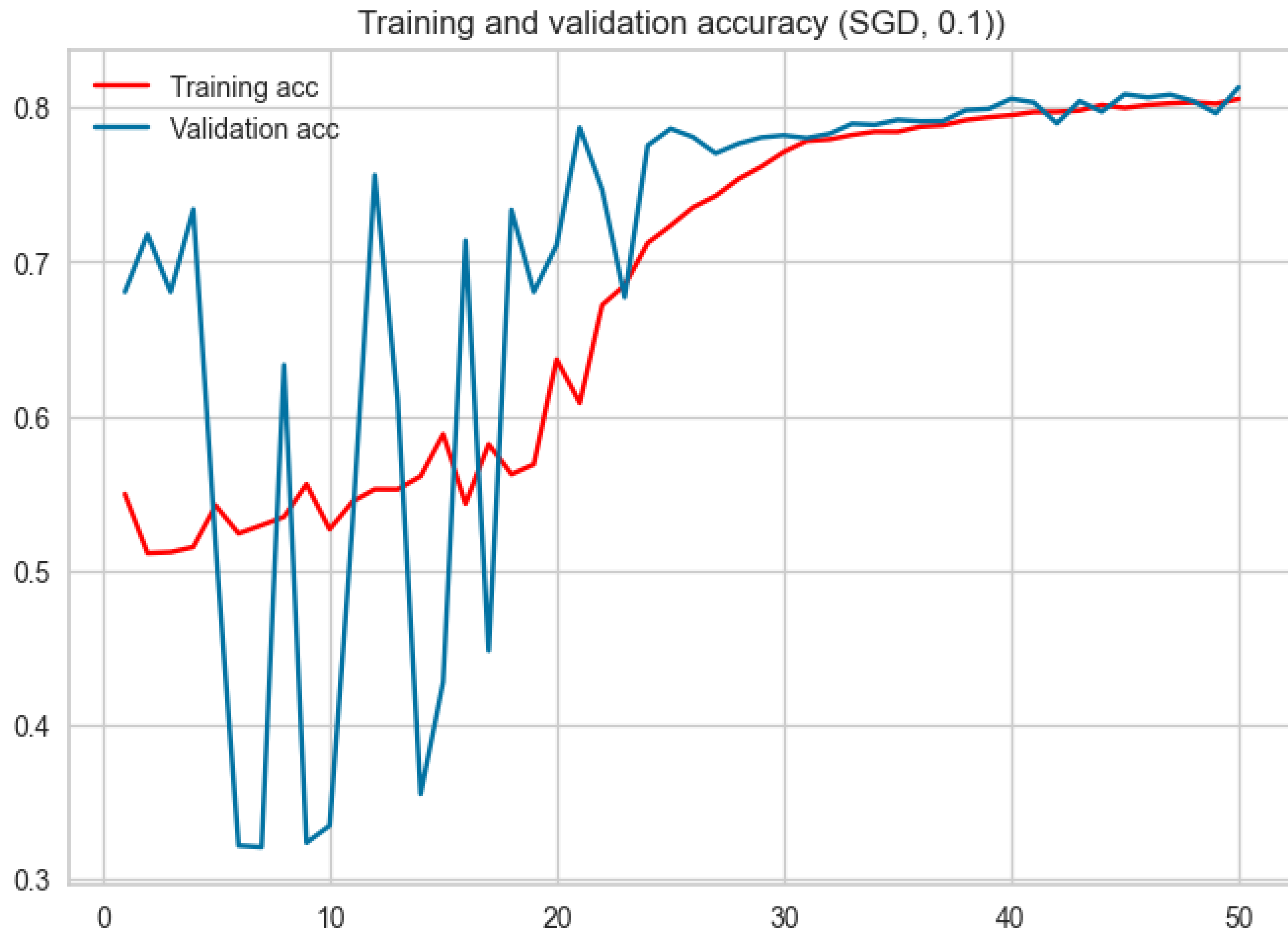
Training and validation loss (SGD, 0.02)
0.69315
0.69310
0.69305
0.69300
0.69295
0 10 20 30 40 50
Training loss
Validation loss

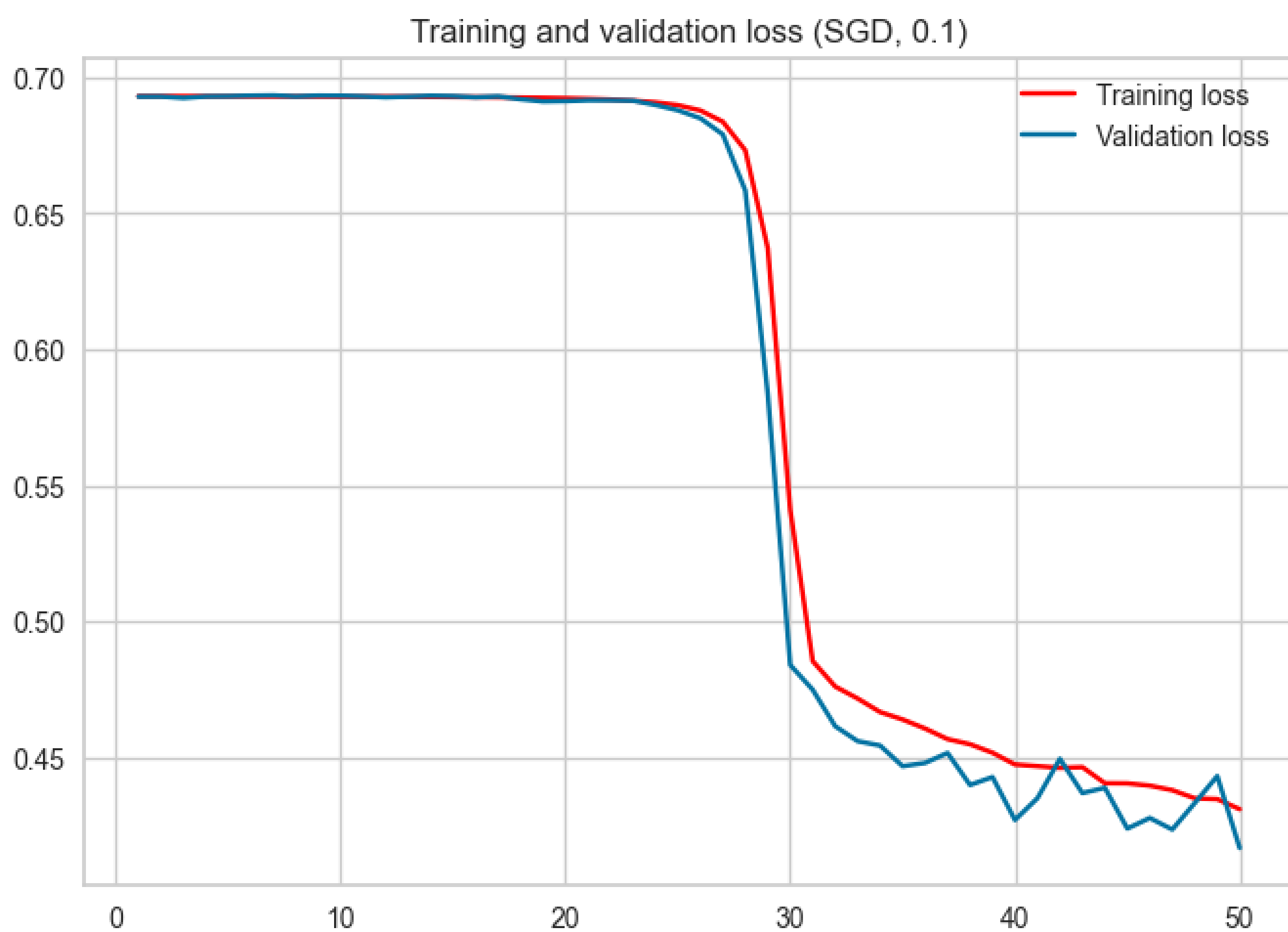


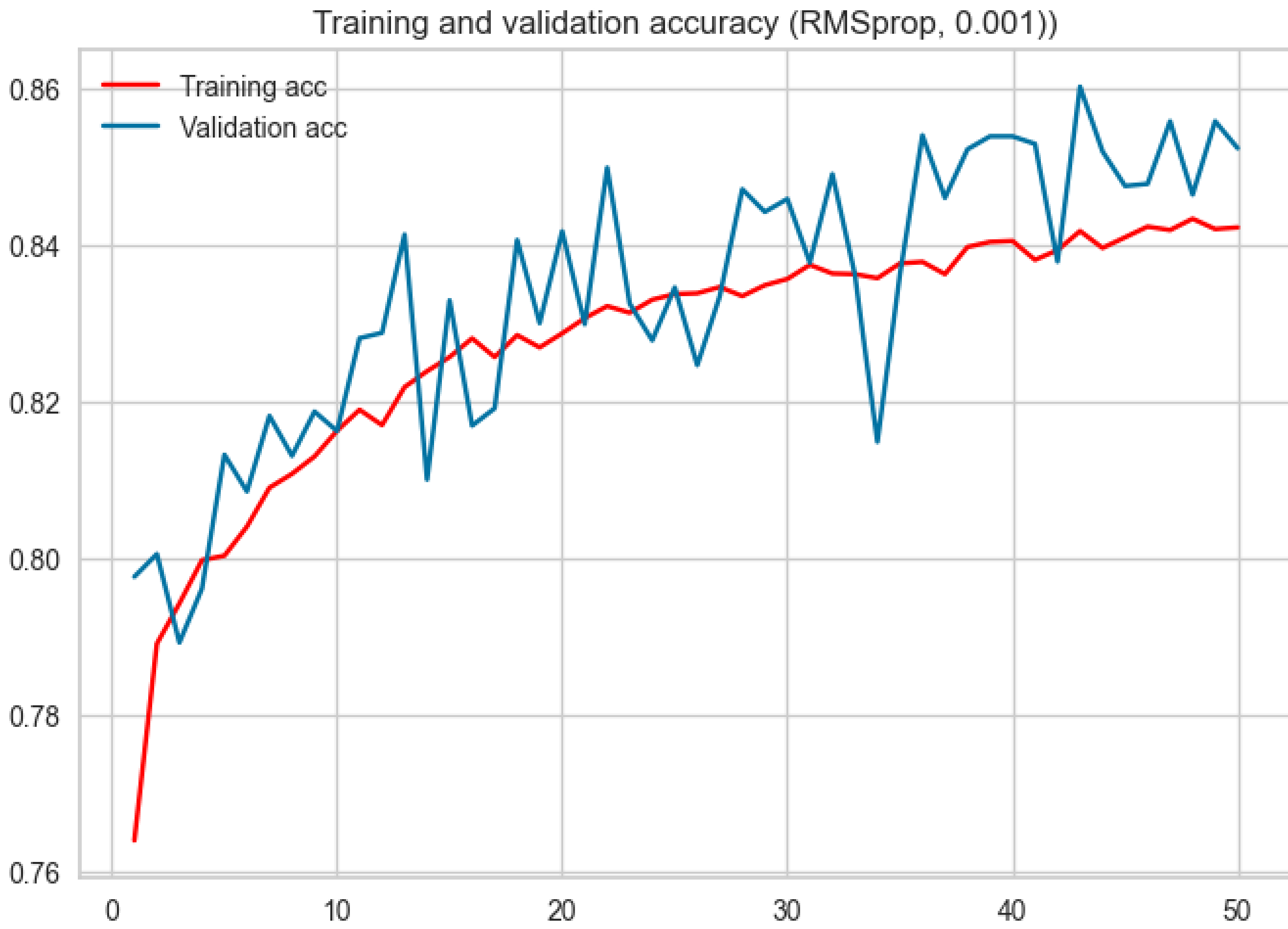
Training and validation accuracy (SGD, 0.03))
Training acc
Validation acc
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
10
20
30
40
50

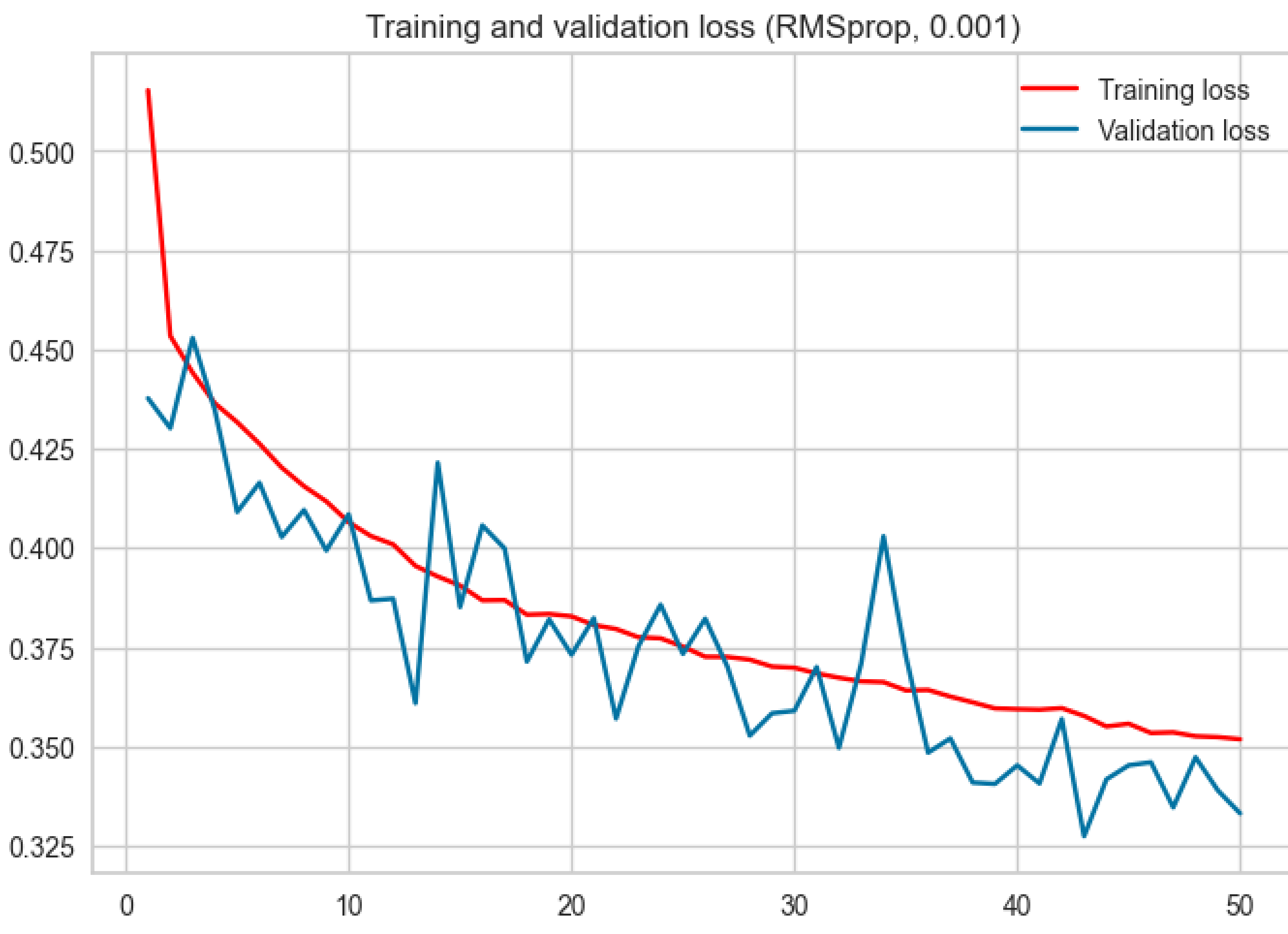


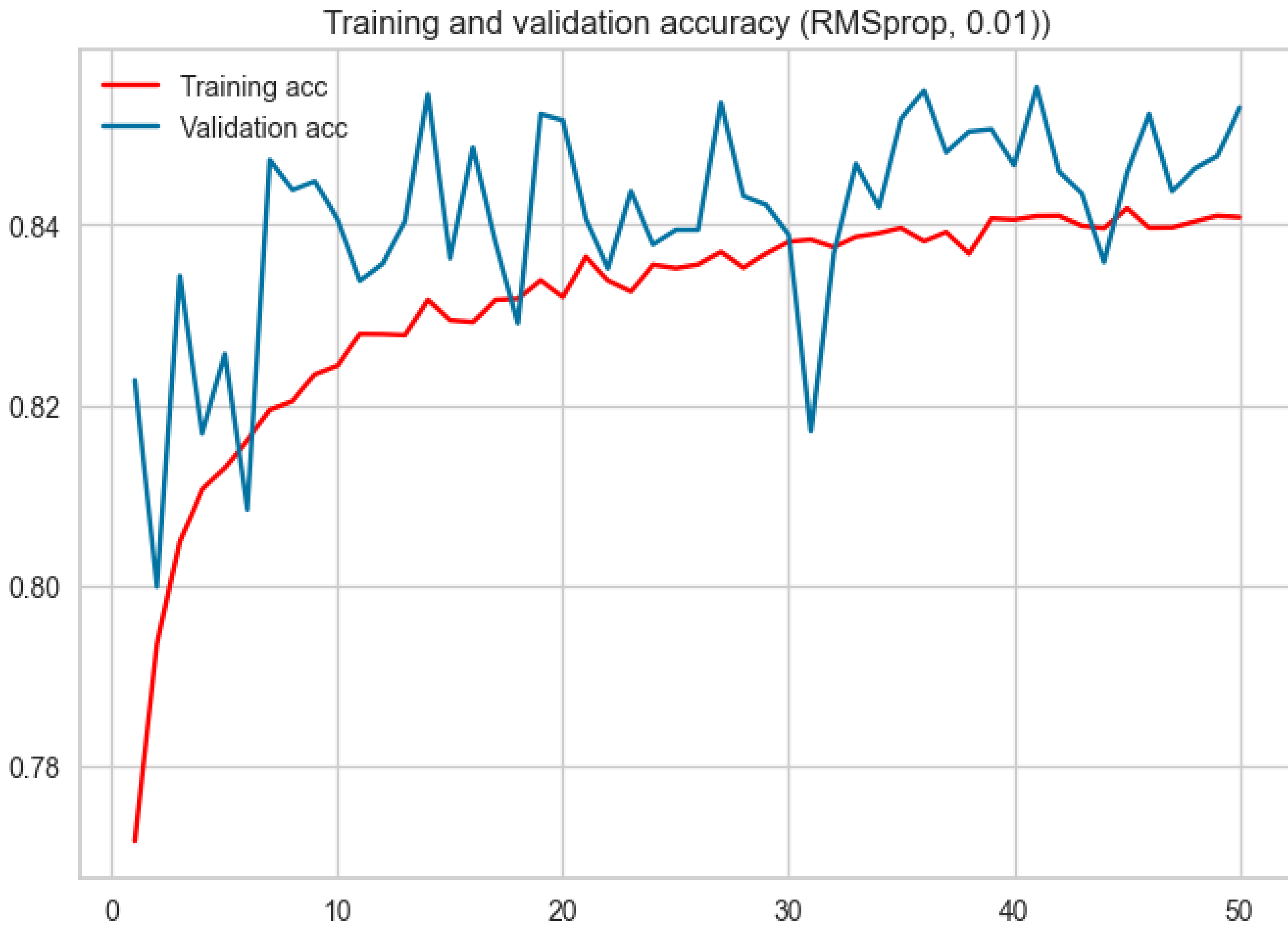
Training and validation loss (SGD, 0.03)
Training loss
Validation loss
0.6932
0.6931
0.6930
0.6929
0
10
20
30
40
50


Training and validation accuracy (SGD, 0.1))
Training acc
Validation acc
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50

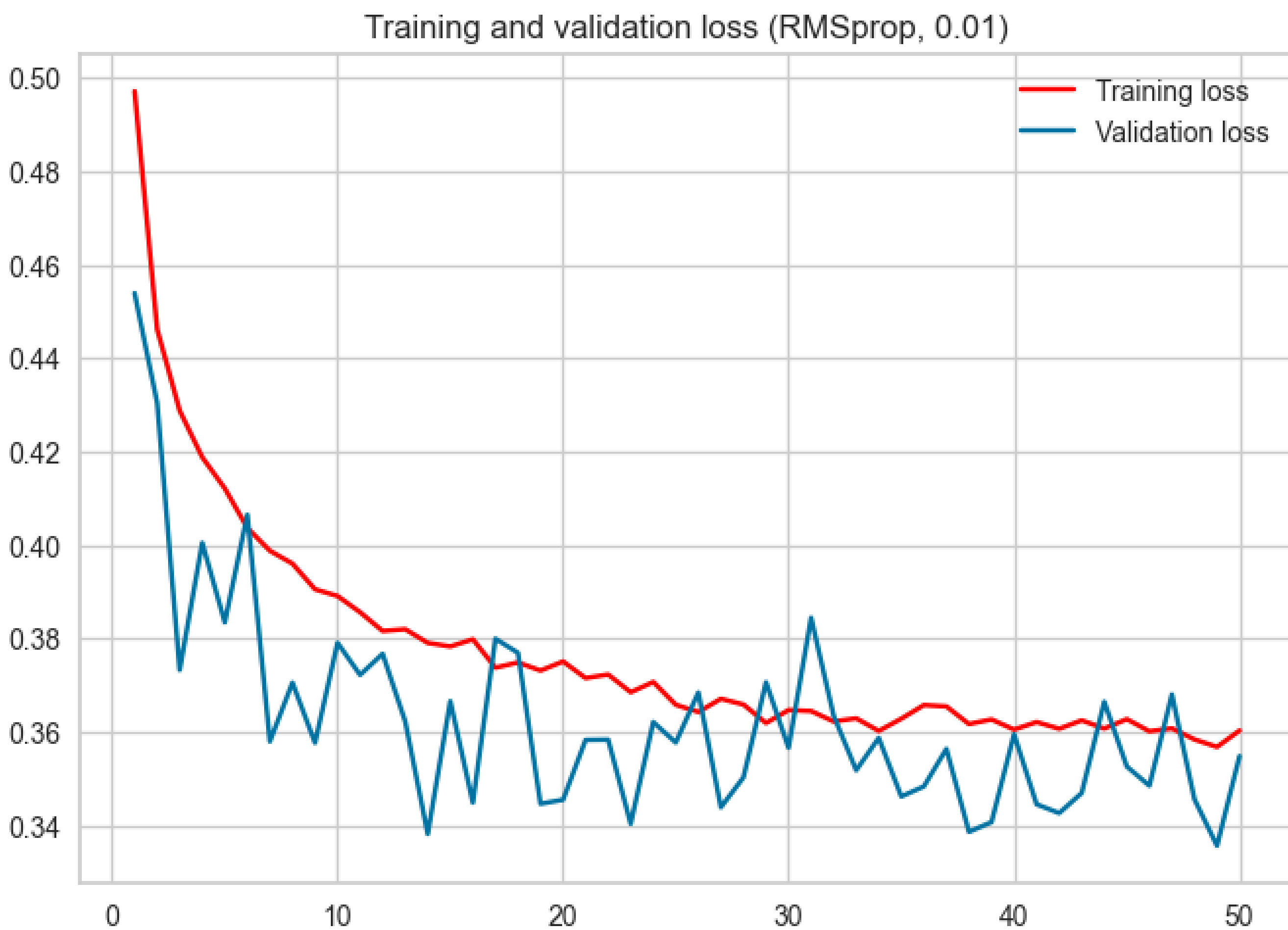

Training and validation loss (SGD, 0.1)
Training loss
Validation loss
0.70
0.65
0.60
0.55
0.50
0.45
0
10
20
30
40
50


Training and validation accuracy (RMSprop, 0.001))
Training acc
Validation acc
0.86
0.84
0.82
0.80
0.78
0.76
0
10
20
30
40
50

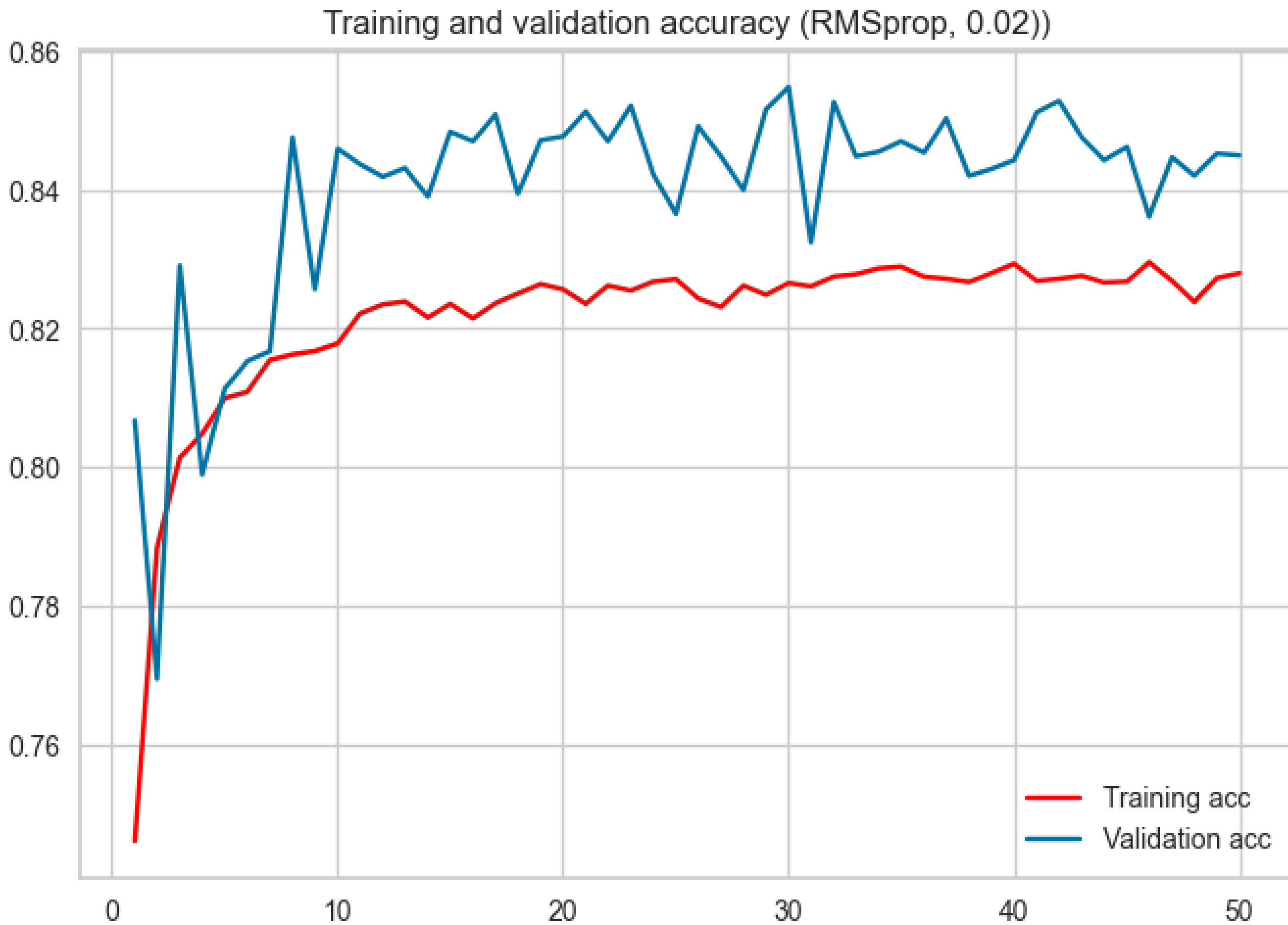

Training and validation loss (RMSprop, 0.001)
Training loss
Validation loss
0.500
0.475
0.450
0.425
0.400
0.375
0.350
0.325
0
10
20
30
40
50

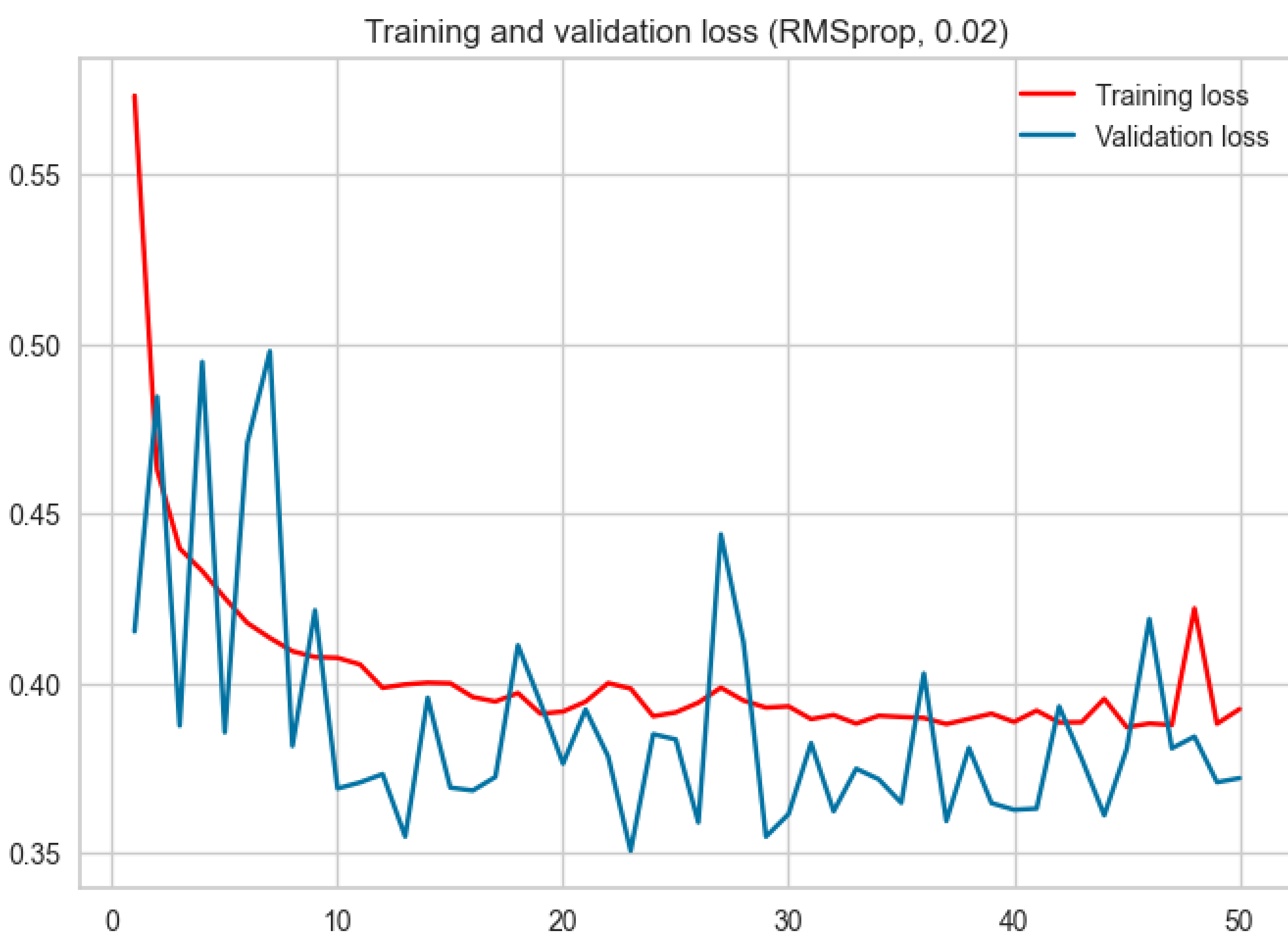


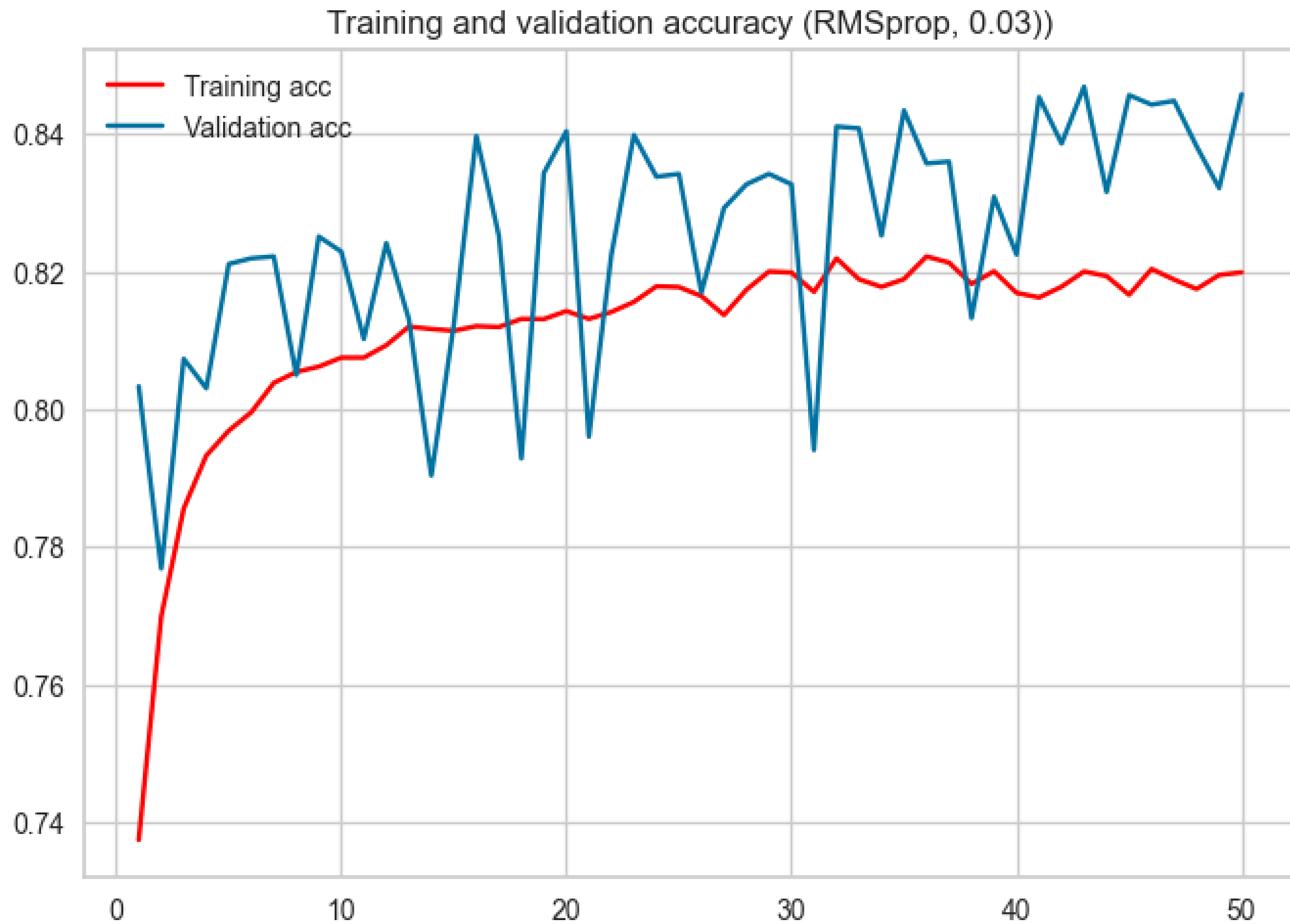
Training and validation accuracy (RMSprop, 0.01))
Training acc
Validation acc
0.84
0.82
0.80
0.78
0
10
20
30
40
50

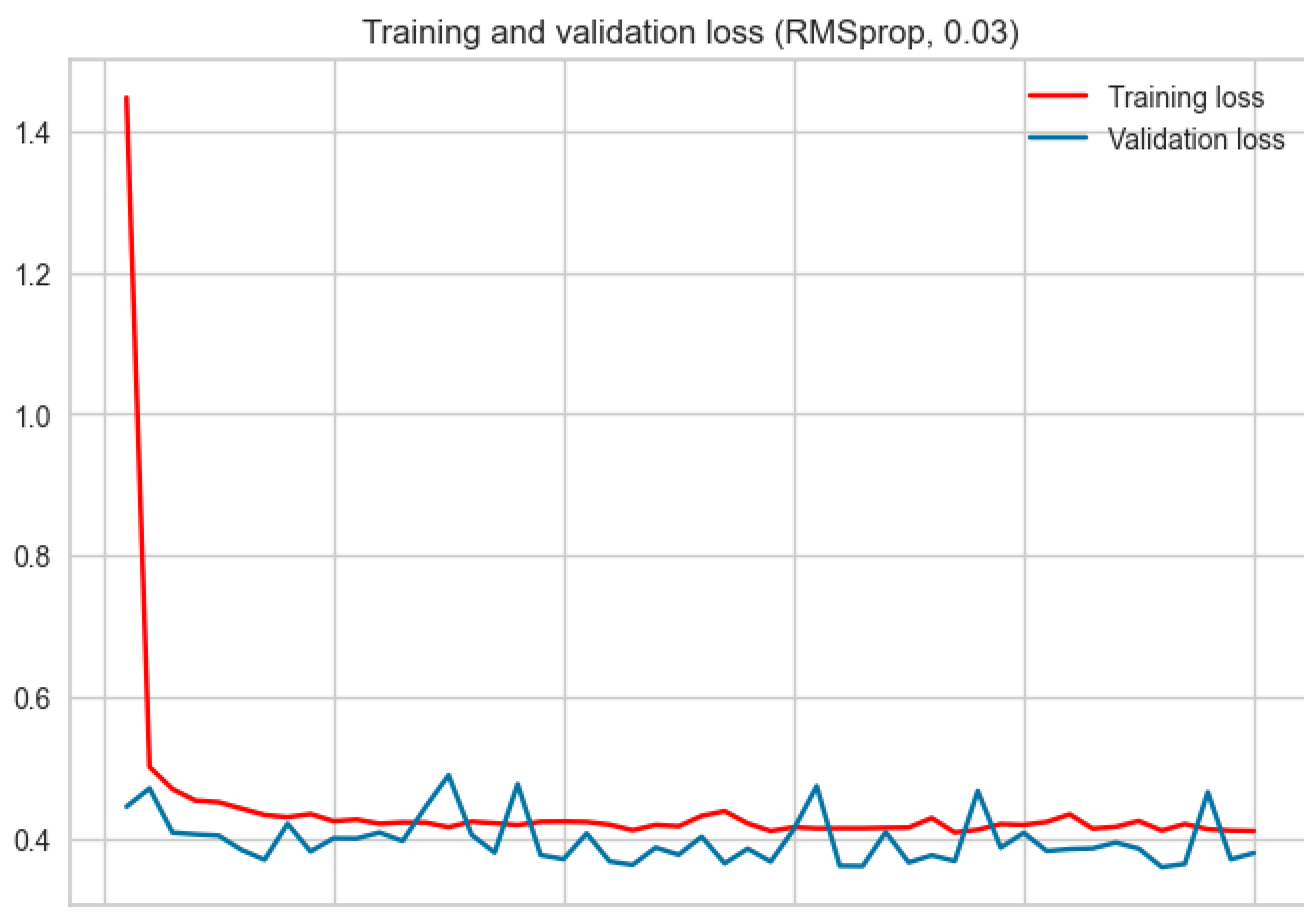


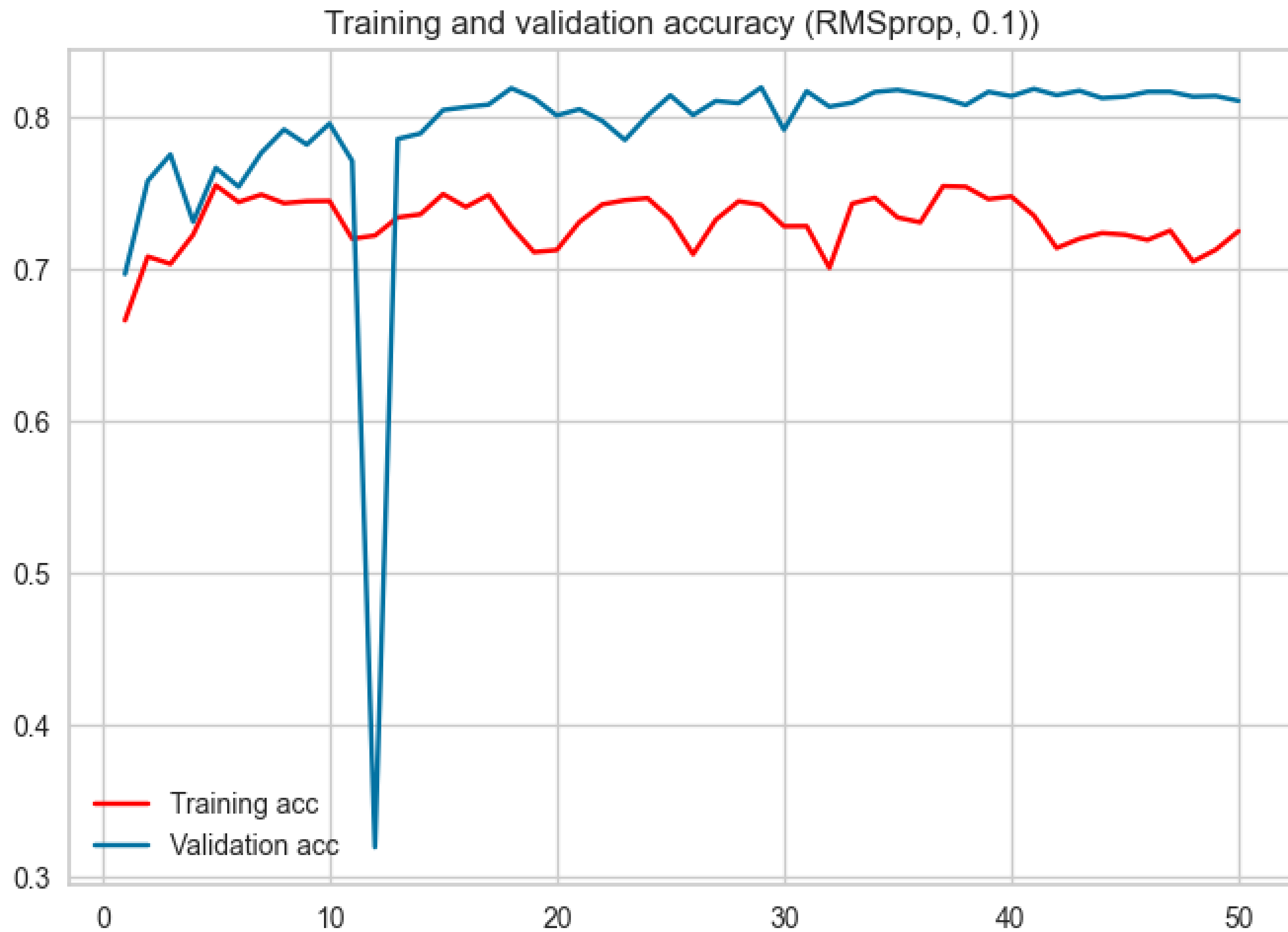
Training and validation loss (RMSprop, 0.01)
Training loss
Validation loss
0.50
0.48
0.46
0.44
0.42
0.40
0.38
0.36
0.34
0
10
20
30
40
50

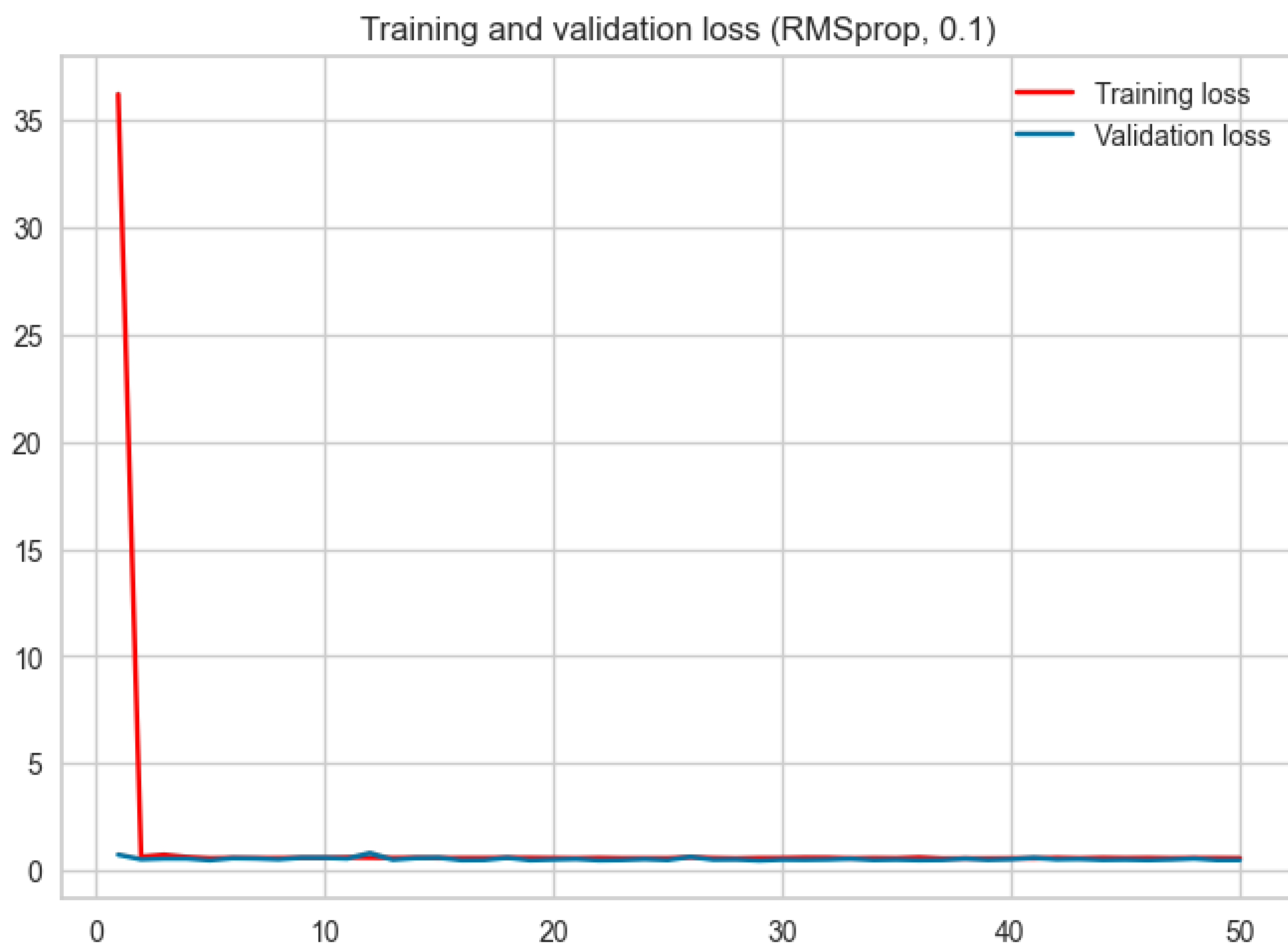

Training and validation accuracy (RMSprop, 0.02))
0.86
0.84
0.82
0.80
0.78
0.76
0
10
20
30
40
50
Training acc
Validation acc


Training and validation loss (RMSprop, 0.02)
0.55
0.50
0.45
0.40
0.35
0
10
20
30
40
50
Training loss
Validation loss


Training and validation accuracy (RMSprop, 0.03))
Training acc
Validation acc
0.84
0.82
0.80
0.78
0.76
0.74
0
10
20
30
40
50


Training and validation loss (RMSprop, 0.03)
Training loss
Validation loss
1.4
1.2
1.0
0.8
0.6
0.4
0
10
20
30
40
50






```
<Figure size 800x550 with 0 Axes>
```

In [48]:

```
#save histories to a file
import pickle
```

```
with open('histories.pickle', 'wb') as handle:
    pickle.dump(histories, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
INFO:tensorflow:Assets written to: ram://87ac715b-ee0b-40e7-8cef-c2bf3a6c33e1/assets
INFO:tensorflow:Assets written to: ram://33f19089-b3e2-423d-a1b2-4a46f3e724a7/assets
INFO:tensorflow:Assets written to: ram://1a1f43a8-ec8c-4b2d-ab88-b3fe2cd6b17b/assets
INFO:tensorflow:Assets written to: ram://cb43283b-5cf5-40c3-ab99-fffad0c5fbe9/assets
INFO:tensorflow:Assets written to: ram://e90ea017-582d-4e6f-be37-0ce9b4838ae5/assets
INFO:tensorflow:Assets written to: ram://c41717f5-2e0a-46d6-864a-3a1b1df590ad/assets
INFO:tensorflow:Assets written to: ram://32b919b3-0931-4295-a5fa-eb06ef95141b/assets
INFO:tensorflow:Assets written to: ram://167f99fa-8f24-4614-a4dd-8017b511433a/assets
INFO:tensorflow:Assets written to: ram://e3fcf1e5-9e82-4667-8da4-e8e8a0940c95/assets
INFO:tensorflow:Assets written to: ram://2d97d46d-9f53-4994-9dc8-36af4760238c/assets
INFO:tensorflow:Assets written to: ram://58413149-6ea2-423a-a369-3fa3390da5fd/assets
INFO:tensorflow:Assets written to: ram://ed57b67f-5eba-4724-bb9a-07491514c5b5/assets
INFO:tensorflow:Assets written to: ram://74bf32fb-5fc2-4320-b5ba-54a290927b7e/assets
INFO:tensorflow:Assets written to: ram://b240e8bf-29fe-4fef-8afb-66d62e62ca2b/assets
INFO:tensorflow:Assets written to: ram://9c5586c4-14db-477f-a17e-f5322839bf7e/assets
```

Decision Trees

In [ ]:

```
min_split = np.array([2, 3, 4, 5, 6, 7])
max_nvl = np.array([3, 4, 5, 6, 7, 9, 11])
alg = ['entropy', 'gini']
values_grid = {'decisiontreeclassifier__min_samples_split': min_split, 'decisiontreeclas

tree = DecisionTreeClassifier()
classifier = make_pipeline(StandardScaler(), SMOTE(random_state=100), tree)

grid = GridSearchCV(classifier, param_grid=values_grid, cv = kf, scoring =  scoring, n_j
```

In [ ]:

```
grid.fit(X, y)
```

In [173…

```
columns = ["mean_test_accuracy", "mean_test_precision", "mean_test_recall", "mean_test_f
params = pd.DataFrame(grid.cv_results_['params'])
scores = pd.DataFrame(grid.cv_results_)[columns]
scores = pd.concat([params, scores], axis=1)

#rename columns
scores.columns = ['criterion', 'max_depth', 'min_samples_split', 'accuracy', 'precision'
scores.sort_values(by=['accuracy'], ascending=False)
```

Out[173]:

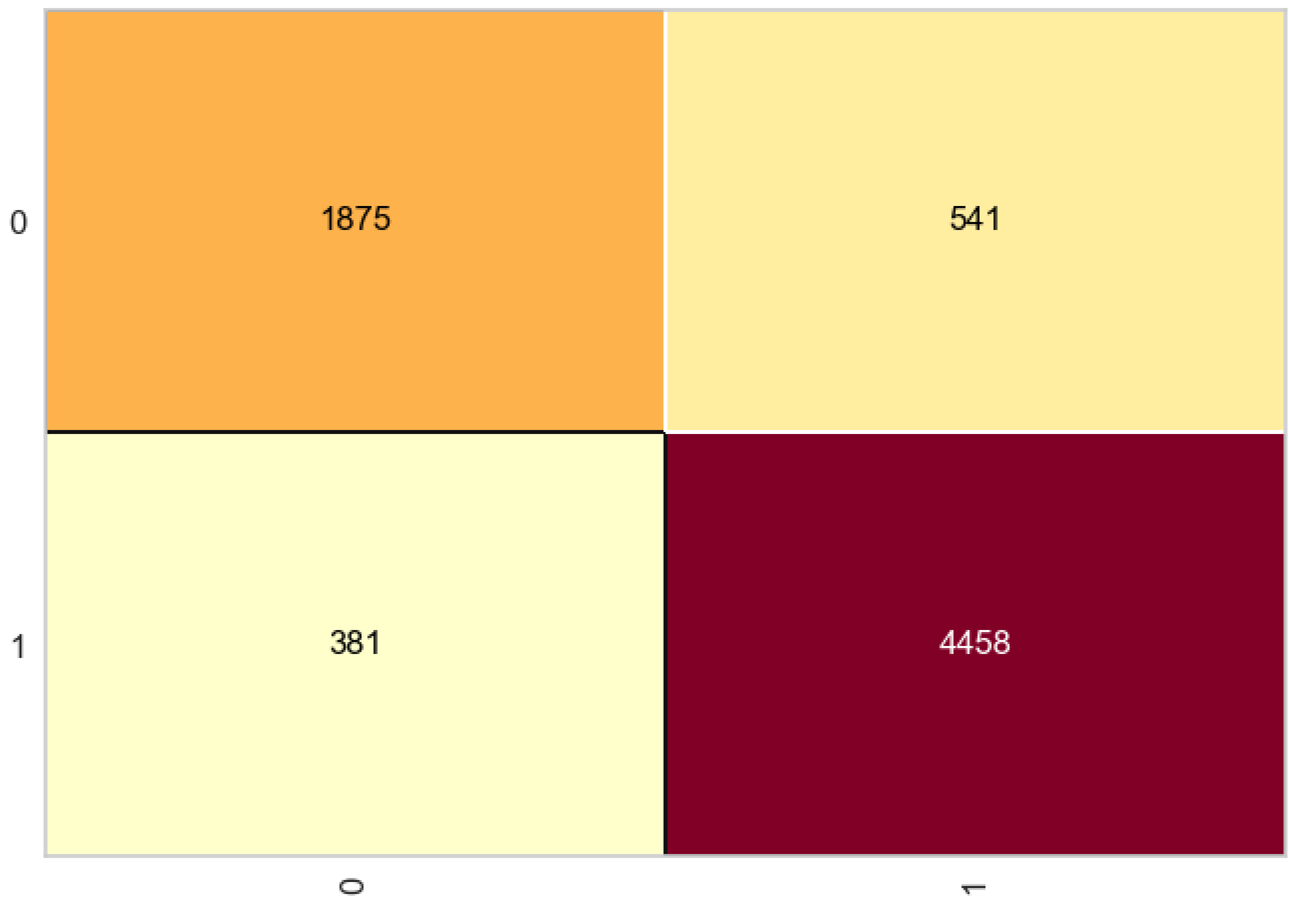
	criterion	max_depth	min_samples_split	accuracy	precision	recall	f1	roc_auc
36	entropy	11	2	0.862853	0.908806	0.885243	0.896604	0.929395
39	entropy	11	5	0.862826	0.908737	0.885284	0.896588	0.929613
38	entropy	11	4	0.862826	0.908664	0.885364	0.896598	0.929581
40	entropy	11	6	0.862660	0.908606	0.885163	0.896471	0.929878
37	entropy	11	3	0.862467	0.908365	0.885125	0.896334	0.928899
...	...	...	...	...	...	...	...	...
5	entropy	3	7	0.786492	0.862204	0.812250	0.836467	0.820231
4	entropy	3	6	0.786492	0.862204	0.812250	0.836467	0.820231
3	entropy	3	5	0.786492	0.862204	0.812250	0.836467	0.820231
2	entropy	3	4	0.786492	0.862204	0.812250	0.836467	0.820231
42	gini	3	2	0.786492	0.862204	0.812250	0.836467	0.821904

84 rows × 8 columns

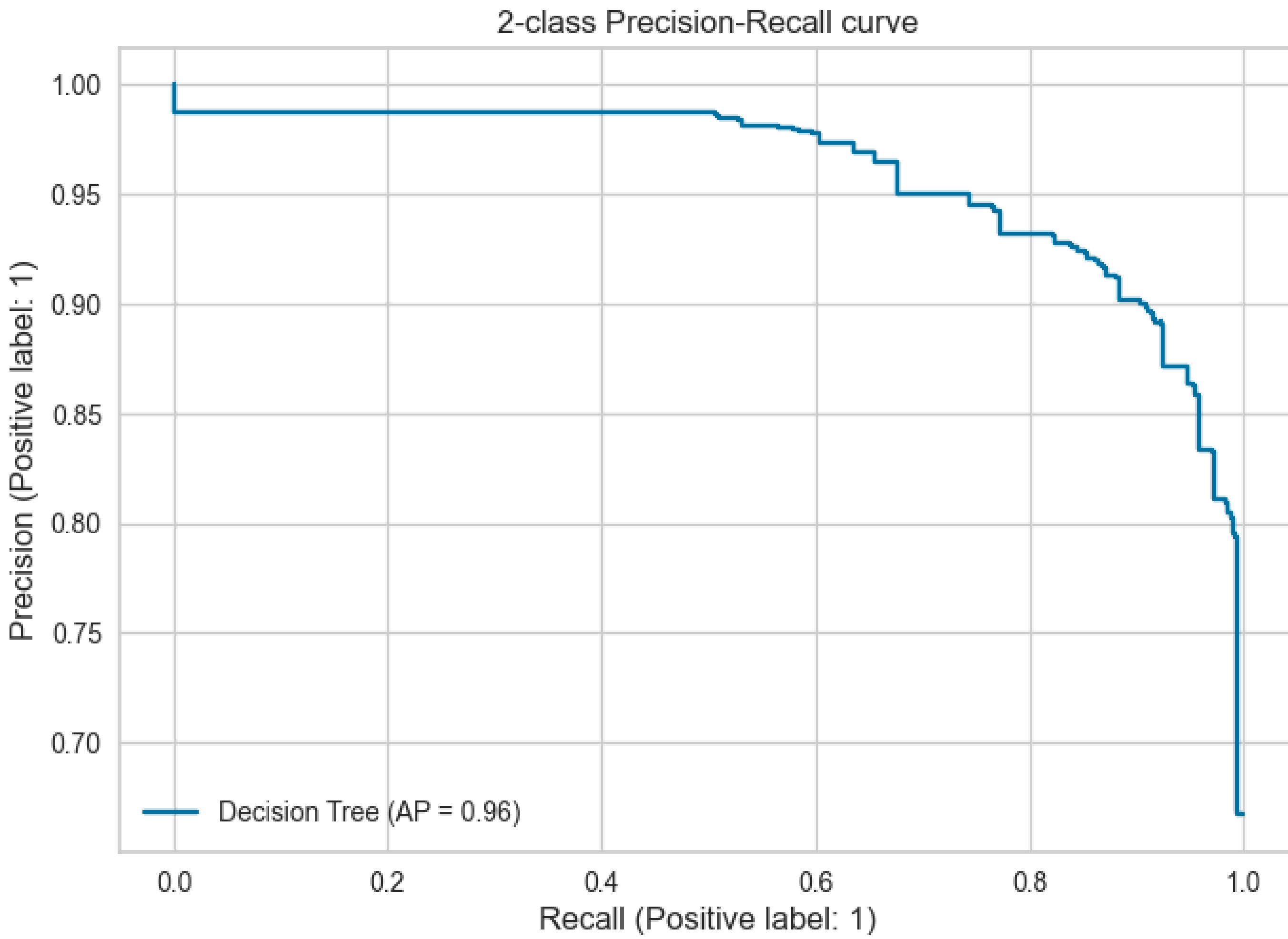
In [21]:
```
tree_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=11, min_samples_
tree_classifier.fit(x_train, y_train)

tree_scores = cross_validate(tree_classifier, X_standard, y, cv=kf, scoring=scoring, n_j
```

In [182…
```
tree_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=11, min_samples_
tree_classifier.fit(x_train, y_train)
y_pred = tree_classifier.predict(x_test)

cm = ConfusionMatrix(tree_classifier)
cm.fit(x_train_balanced, y_train_balanced)
cm.score(x_test, y_test)
```

Out[182]: 0.8729152308752585



In [16]:
```
tree_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=11, min_samples_
tree_classifier.fit(x_train, y_train)

display = PrecisionRecallDisplay.from_estimator(
    tree_classifier, x_test, y_test, name="Decision Tree"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [17]:
```
y_pred = tree_classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.83      0.78      0.80      2416
Not Canceled       0.89      0.92      0.91      4839

    accuracy                           0.87      7255
   macro avg       0.86      0.85      0.85      7255
weighted avg       0.87      0.87      0.87      7255
```

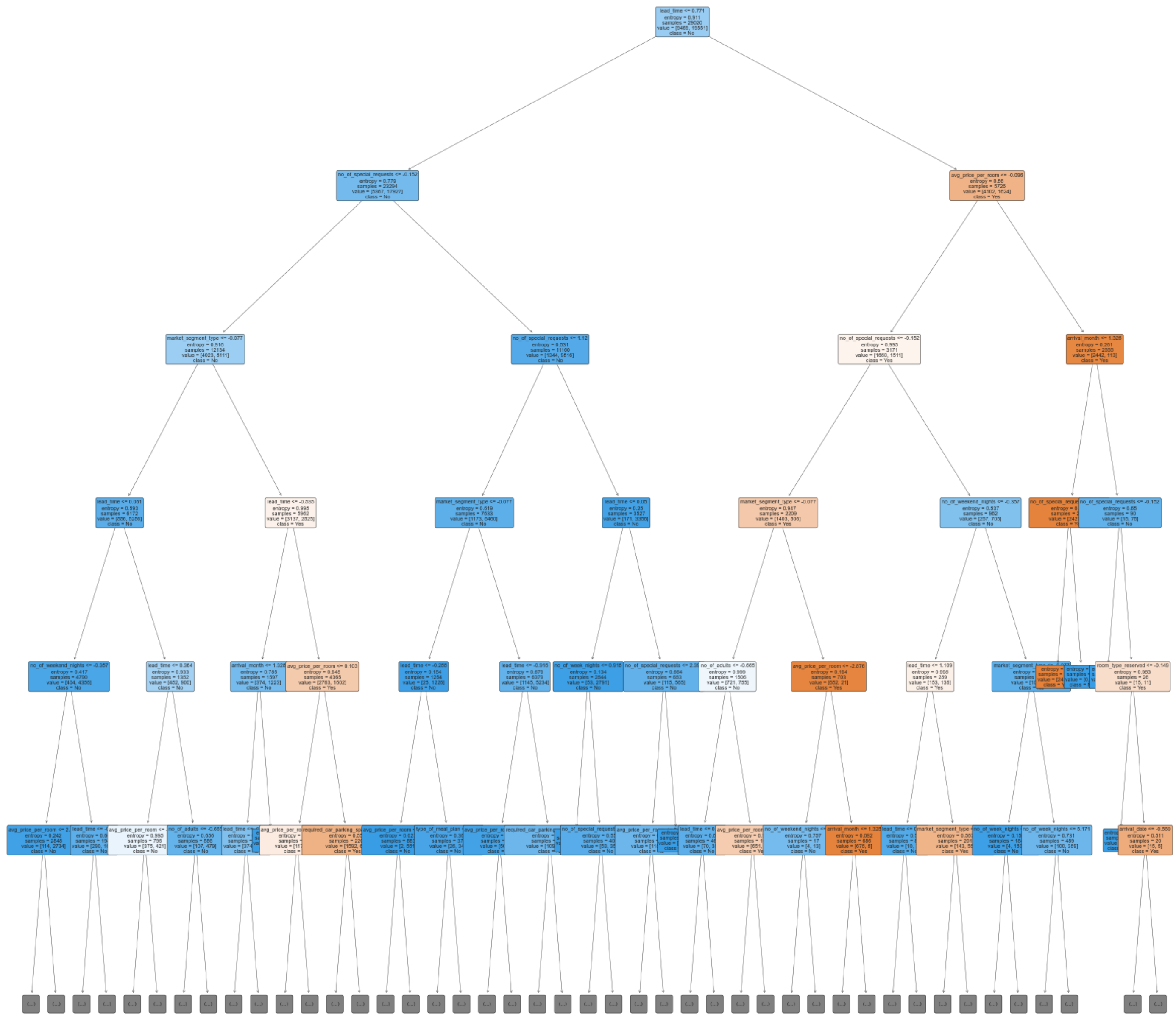
In [189…
```
#We plotted a tree where the depth was 5, to show how the tree is built

plt.figure(figsize=(20,20))
plot_tree(tree_classifier, filled=True, rounded=True, class_names=['Yes', 'No'], feature
```

Out[189]:
```
[Text(0.5762362637362637, 0.9285714285714286, 'lead_time <= 0.771\nentropy = 0.911\nsamp
les = 29020\nvalue = [9469, 19551]\nclass = No'),
 Text(0.3118131868131868, 0.7857142857142857, 'no_of_special_requests <= -0.152\nentropy
= 0.779\nsamples = 23294\nvalue = [5367, 17927]\nclass = No'),
 Text(0.1620879120879121, 0.6428571428571429, 'market_segment_type <= -0.077\nentropy =
0.916\nsamples = 12134\nvalue = [4023, 8111]\nclass = No'),
 Text(0.08791208791208792, 0.5, 'lead_time <= 0.061\nentropy = 0.593\nsamples = 6172\nva
lue = [886, 5286]\nclass = No'),
 Text(0.04395604395604396, 0.35714285714285715, 'no_of_weekend_nights <= -0.357\nentropy
= 0.417\nsamples = 4790\nvalue = [404, 4386]\nclass = No'),
 Text(0.02197802197802198, 0.21428571428571427, 'avg_price_per_room <= 2.795\nentropy =
0.242\nsamples = 2848\nvalue = [114, 2734]\nclass = No'),
 Text(0.01098901098901099, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.03296703296703297, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.06593406593406594, 0.21428571428571427, 'lead_time <= -0.23\nentropy = 0.608\nsa
```

```
mples = 1942\nvalue = [290, 1652]\nclass = No'),
 Text(0.054945054945054944, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.07692307692307693, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.13186813186813187, 0.35714285714285715, 'lead_time <= 0.364\nentropy = 0.933\nsa
mples = 1382\nvalue = [482, 900]\nclass = No'),
 Text(0.10989010989010989, 0.21428571428571427, 'avg_price_per_room <= -0.281\nentropy =
0.998\nsamples = 796\nvalue = [375, 421]\nclass = No'),
 Text(0.0989010989010989, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.12087912087912088, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.15384615384615385, 0.21428571428571427, 'no_of_adults <= -0.665\nentropy = 0.686
\nsamples = 586\nvalue = [107, 479]\nclass = No'),
 Text(0.14285714285714285, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.16483516483516483, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.23626373626373626, 0.5, 'lead_time <= -0.835\nentropy = 0.998\nsamples = 5962\nv
alue = [3137, 2825]\nclass = Yes'),
 Text(0.2087912087912088, 0.35714285714285715, 'arrival_month <= 1.328\nentropy = 0.785
\nsamples = 1597\nvalue = [374, 1223]\nclass = No'),
 Text(0.1978021978021978, 0.21428571428571427, 'lead_time <= -0.951\nentropy = 0.834\nsa
mples = 1411\nvalue = [374, 1037]\nclass = No'),
 Text(0.18681318681318682, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.2087912087912088, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.21978021978021978, 0.21428571428571427, 'entropy = 0.0\nsamples = 186\nvalue =
[0, 186]\nclass = No'),
 Text(0.26373626373626374, 0.35714285714285715, 'avg_price_per_room <= 0.103\nentropy =
0.948\nsamples = 4365\nvalue = [2763, 1602]\nclass = Yes'),
 Text(0.24175824175824176, 0.21428571428571427, 'avg_price_per_room <= -1.209\nentropy =
0.995\nsamples = 2159\nvalue = [1171, 988]\nclass = Yes'),
 Text(0.23076923076923078, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.25274725274725274, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.2857142857142857, 0.21428571428571427, 'required_car_parking_space <= 2.707\nent
ropy = 0.853\nsamples = 2206\nvalue = [1592, 614]\nclass = Yes'),
 Text(0.27472527472527475, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.2967032967032967, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.46153846153846156, 0.6428571428571429, 'no_of_special_requests <= 1.12\nentropy
= 0.531\nsamples = 11160\nvalue = [1344, 9816]\nclass = No'),
 Text(0.3956043956043956, 0.5, 'market_segment_type <= -0.077\nentropy = 0.619\nsamples
= 7633\nvalue = [1173, 6460]\nclass = No'),
 Text(0.3516483516483517, 0.35714285714285715, 'lead_time <= -0.288\nentropy = 0.154\nsa
mples = 1254\nvalue = [28, 1226]\nclass = No'),
 Text(0.32967032967032966, 0.21428571428571427, 'avg_price_per_room <= 1.109\nentropy =
0.023\nsamples = 883\nvalue = [2, 881]\nclass = No'),
 Text(0.31868131868131866, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.34065934065934067, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.37362637362637363, 0.21428571428571427, 'type_of_meal_plan <= 1.416\nentropy =
0.366\nsamples = 371\nvalue = [26, 345]\nclass = No'),
 Text(0.3626373626373626, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.38461538461538464, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.43956043956043955, 0.35714285714285715, 'lead_time <= -0.916\nentropy = 0.679\ns
amples = 6379\nvalue = [1145, 5234]\nclass = No'),
 Text(0.4175824175824176, 0.21428571428571427, 'avg_price_per_room <= 1.545\nentropy =
0.282\nsamples = 1023\nvalue = [50, 973]\nclass = No'),
 Text(0.4065934065934066, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.42857142857142855, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.46153846153846156, 0.21428571428571427, 'required_car_parking_space <= 2.707\nen
tropy = 0.731\nsamples = 5356\nvalue = [1095, 4261]\nclass = No'),
 Text(0.45054945054945056, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4725274725274725, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5274725274725275, 0.5, 'lead_time <= 0.05\nentropy = 0.28\nsamples = 3527\nvalue
= [171, 3356]\nclass = No'),
 Text(0.4945054945054945, 0.35714285714285715, 'no_of_week_nights <= 0.918\nentropy = 0.
134\nsamples = 2844\nvalue = [53, 2791]\nclass = No'),
 Text(0.4835164835164835, 0.21428571428571427, 'entropy = 0.0\nsamples = 2437\nvalue =
[0, 2437]\nclass = No'),
 Text(0.5054945054945055, 0.21428571428571427, 'no_of_special_requests <= 2.392\nentropy
= 0.558\nsamples = 407\nvalue = [53, 354]\nclass = No'),
 Text(0.4945054945054945, 0.07142857142857142, '\n  (...)  \n'),
```

```
 Text(0.5164835164835165, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5604395604395604, 0.35714285714285715, 'no_of_special_requests <= 2.392\nentropy
= 0.664\nsamples = 683\nvalue = [118, 565]\nclass = No'),
 Text(0.5494505494505495, 0.21428571428571427, 'avg_price_per_room <= 2.836\nentropy =
0.731\nsamples = 577\nvalue = [118, 459]\nclass = No'),
 Text(0.5384615384615384, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5604395604395604, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5714285714285714, 0.21428571428571427, 'entropy = 0.0\nsamples = 106\nvalue =
[0, 106]\nclass = No'),
 Text(0.8406593406593407, 0.7857142857142857, 'avg_price_per_room <= -0.096\nentropy =
0.86\nsamples = 5726\nvalue = [4102, 1624]\nclass = Yes'),
 Text(0.7472527472527473, 0.6428571428571429, 'no_of_special_requests <= -0.152\nentropy
= 0.998\nsamples = 3171\nvalue = [1660, 1511]\nclass = Yes'),
 Text(0.6593406593406593, 0.5, 'market_segment_type <= -0.077\nentropy = 0.947\nsamples
= 2209\nvalue = [1403, 806]\nclass = Yes'),
 Text(0.6153846153846154, 0.35714285714285715, 'no_of_adults <= -0.665\nentropy = 0.999
\nsamples = 1506\nvalue = [721, 785]\nclass = No'),
 Text(0.5934065934065934, 0.21428571428571427, 'lead_time <= 0.911\nentropy = 0.61\nsamp
les = 466\nvalue = [70, 396]\nclass = No'),
 Text(0.5824175824175825, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6043956043956044, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6373626373626373, 0.21428571428571427, 'avg_price_per_room <= -0.536\nentropy =
0.954\nsamples = 1040\nvalue = [651, 389]\nclass = Yes'),
 Text(0.6263736263736264, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6483516483516484, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7032967032967034, 0.35714285714285715, 'avg_price_per_room <= -2.876\nentropy =
0.194\nsamples = 703\nvalue = [682, 21]\nclass = Yes'),
 Text(0.6813186813186813, 0.21428571428571427, 'no_of_weekend_nights <= -0.357\nentropy
= 0.787\nsamples = 17\nvalue = [4, 13]\nclass = No'),
 Text(0.6703296703296703, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6923076923076923, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7252747252747253, 0.21428571428571427, 'arrival_month <= 1.328\nentropy = 0.092
\nsamples = 686\nvalue = [678, 8]\nclass = Yes'),
 Text(0.7142857142857143, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7362637362637363, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8351648351648352, 0.5, 'no_of_weekend_nights <= -0.357\nentropy = 0.837\nsamples
= 962\nvalue = [257, 705]\nclass = No'),
 Text(0.7912087912087912, 0.35714285714285715, 'lead_time <= 1.109\nentropy = 0.998\nsam
ples = 289\nvalue = [153, 136]\nclass = Yes'),
 Text(0.7692307692307693, 0.21428571428571427, 'lead_time <= 0.864\nentropy = 0.511\nsam
ples = 88\nvalue = [10, 78]\nclass = No'),
 Text(0.7582417582417582, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7802197802197802, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8131868131868132, 0.21428571428571427, 'market_segment_type <= -0.077\nentropy =
0.867\nsamples = 201\nvalue = [143, 58]\nclass = Yes'),
 Text(0.8021978021978022, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8241758241758241, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8791208791208791, 0.35714285714285715, 'market_segment_type <= -0.077\nentropy =
0.621\nsamples = 673\nvalue = [104, 569]\nclass = No'),
 Text(0.8571428571428571, 0.21428571428571427, 'no_of_week_nights <= 0.21\nentropy = 0.1
51\nsamples = 184\nvalue = [4, 180]\nclass = No'),
 Text(0.8461538461538461, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8681318681318682, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9010989010989011, 0.21428571428571427, 'no_of_week_nights <= 5.171\nentropy = 0.
731\nsamples = 489\nvalue = [100, 389]\nclass = No'),
 Text(0.8901098901098901, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9120879120879121, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9340659340659341, 0.6428571428571429, 'arrival_month <= 1.328\nentropy = 0.261\n
samples = 2555\nvalue = [2442, 113]\nclass = Yes'),
 Text(0.9120879120879121, 0.5, 'no_of_special_requests <= 2.392\nentropy = 0.115\nsample
s = 2465\nvalue = [2427, 38]\nclass = Yes'),
 Text(0.9010989010989011, 0.35714285714285715, 'entropy = 0.0\nsamples = 2427\nvalue =
[2427, 0]\nclass = Yes'),
 Text(0.9230769230769231, 0.35714285714285715, 'entropy = 0.0\nsamples = 38\nvalue = [0,
38]\nclass = No'),
 Text(0.9560439560439561, 0.5, 'no_of_special_requests <= -0.152\nentropy = 0.65\nsample
```

```
s = 90\nvalue = [15, 75]\nclass = No'),
 Text(0.945054945054945, 0.35714285714285715, 'entropy = 0.0\nsamples = 64\nvalue = [0, 64]\nclass = No'),
 Text(0.967032967032967, 0.35714285714285715, 'room_type_reserved <= -0.149\nentropy = 0.983\nsamples = 26\nvalue = [15, 11]\nclass = Yes'),
 Text(0.9560439560439561, 0.21428571428571427, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6]\nclass = No'),
 Text(0.978021978021978, 0.21428571428571427, 'arrival_date <= -0.869\nentropy = 0.811\nsamples = 20\nvalue = [15, 5]\nclass = Yes'),
 Text(0.967032967032967, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.989010989010989, 0.07142857142857142, '\n  (...)  \n')]
```



Random forest

In [214…

```
n_estimators = np.array([100])
alg = ['entropy', 'gini']
min_split = np.array([2, 3, 4, 5, 6, 7])
max_nvl = np.array([3, 4, 5, 6, 7, 9, 11])
values_grid = {'randomforestclassifier__n_estimators': n_estimators, 'randomforestclassi
classifier = make_pipeline(StandardScaler(), SMOTE(random_state=100), RandomForestClassi
```

```
gridRandomForest = GridSearchCV(classifier , param_grid = values_grid, cv = kf, scoring
```

In [215…

```
gridRandomForest.fit(X, y)
```

Out[215]:

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ StandardScaler
  ▸ SMOTE
  ▸ RandomForestClassifier
```

In [218…

```
columns = ["mean_test_accuracy", "mean_test_precision", "mean_test_recall", "mean_test_f
params = pd.DataFrame(gridRandomForest.cv_results_['params'])
scores = pd.DataFrame(gridRandomForest.cv_results_)[columns]
scores = pd.concat([params, scores], axis=1)

#rename columns
scores.columns = ['criterion', 'max_depth', 'min_samples_split', 'n_estimators', 'accura
scores.sort_values(by=['accuracy'], ascending=False)
```

Out[218]:

	criterion	max_depth	min_samples_split	n_estimators	accuracy	precision	recall	f1	roc_auc
79	gini	11	3	100	0.879338	0.906817	0.914511	0.910642	0.940346
83	gini	11	7	100	0.879311	0.906353	0.915045	0.910672	0.939952
78	gini	11	2	100	0.879283	0.907447	0.913656	0.910534	0.940471
80	gini	11	4	100	0.879228	0.906445	0.914798	0.910595	0.940221
81	gini	11	5	100	0.879145	0.906900	0.914102	0.910480	0.939925
...	...	...	...	...	...	...	...	...	...
0	entropy	3	2	100	0.783846	0.878268	0.787810	0.830477	0.858781
5	entropy	3	7	100	0.783790	0.873806	0.792988	0.831422	0.856339
1	entropy	3	3	100	0.783487	0.871850	0.794877	0.831552	0.858246
46	gini	3	6	100	0.783405	0.874912	0.790992	0.830811	0.859535
3	entropy	3	5	100	0.783129	0.869865	0.796698	0.831618	0.856017

84 rows × 9 columns

In [20]:

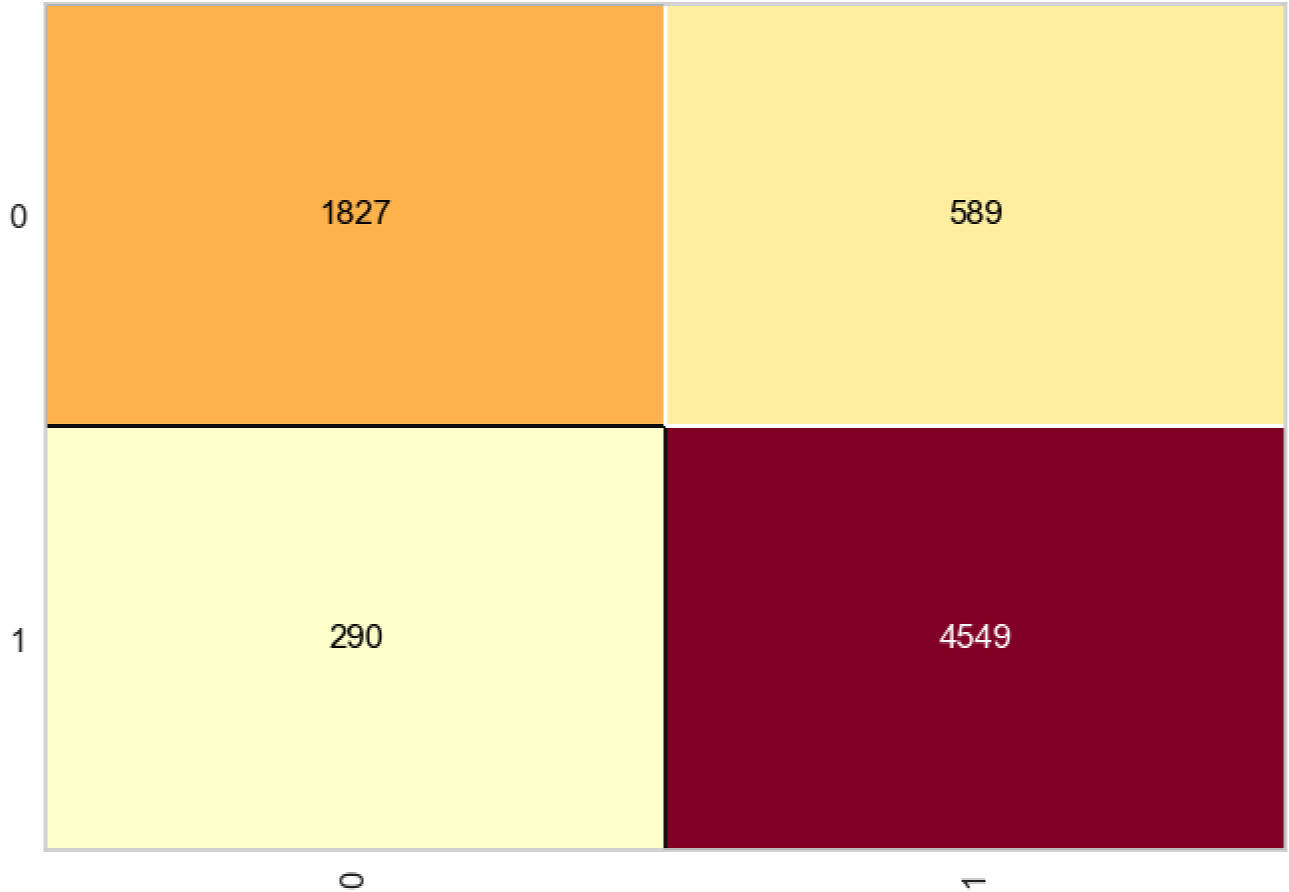
```
forest_classifier = RandomForestClassifier(criterion='gini', max_depth=11, min_samples_s
forest_classifier.fit(x_train_balanced, y_train_balanced)

forest_scores = cross_validate(forest_classifier, X_standard, y, cv=kf, scoring=scoring,
```

In [21]:

```
forest_classifier = RandomForestClassifier(criterion='gini', max_depth=11, min_samples_s
forest_classifier.fit(x_train_balanced, y_train_balanced)
y_pred = forest_classifier.predict(x_test)

cm = ConfusionMatrix(forest_classifier)
cm.fit(x_train_balanced, y_train_balanced)
cm.score(x_test, y_test)
```

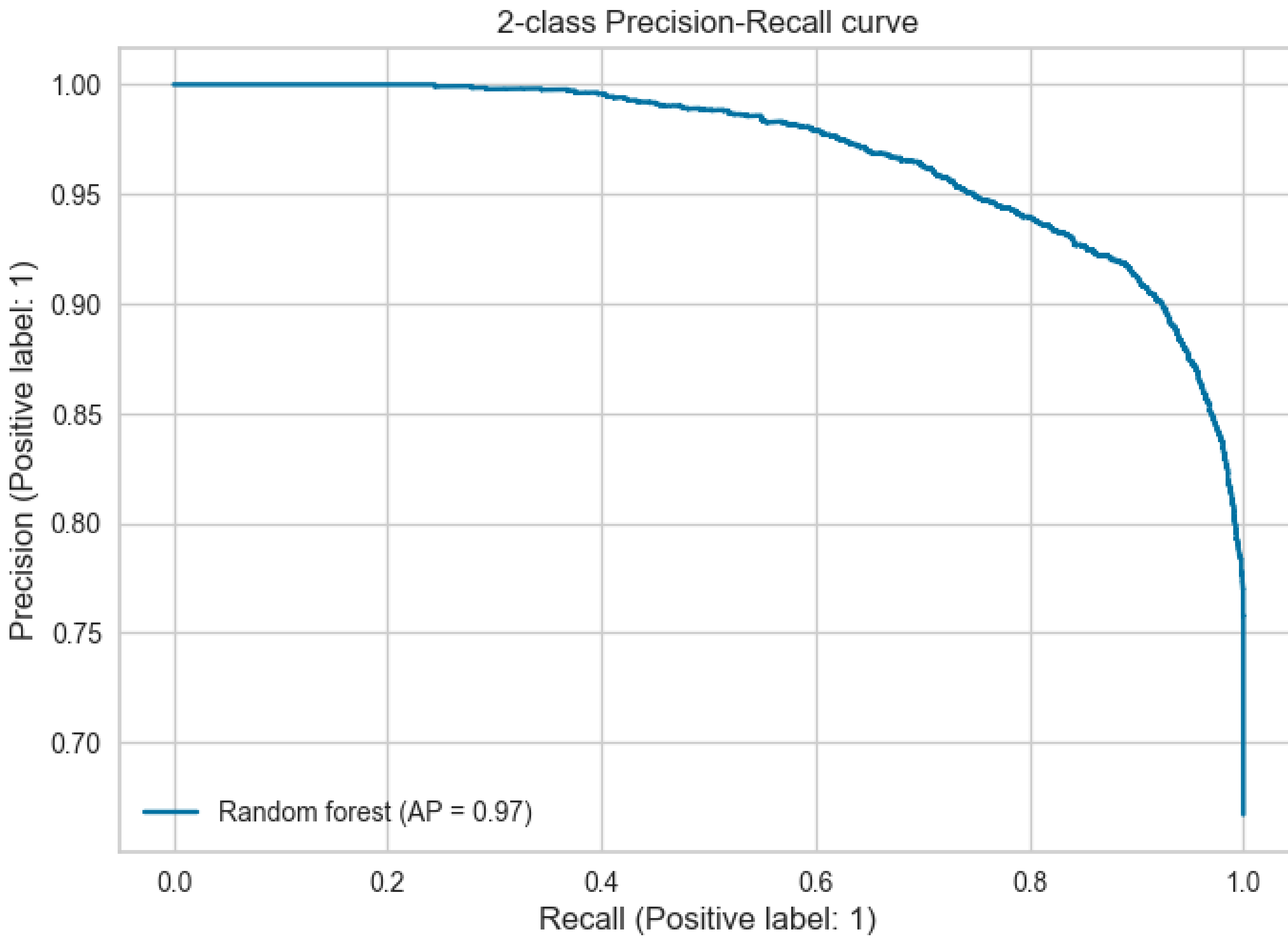
```
0.878842177808408
```

Out[21]:



In [18]:

```
forest_classifier = RandomForestClassifier(criterion='gini', max_depth=11, min_samples_s
forest_classifier.fit(x_train_balanced, y_train_balanced)

display = PrecisionRecallDisplay.from_estimator(
    forest_classifier, x_test, y_test, name="Random forest"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [19]:
```
y_pred = forest_classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.86      0.76      0.81      2416
Not Canceled       0.89      0.94      0.91      4839

    accuracy                           0.88      7255
   macro avg       0.87      0.85      0.86      7255
weighted avg       0.88      0.88      0.88      7255
```

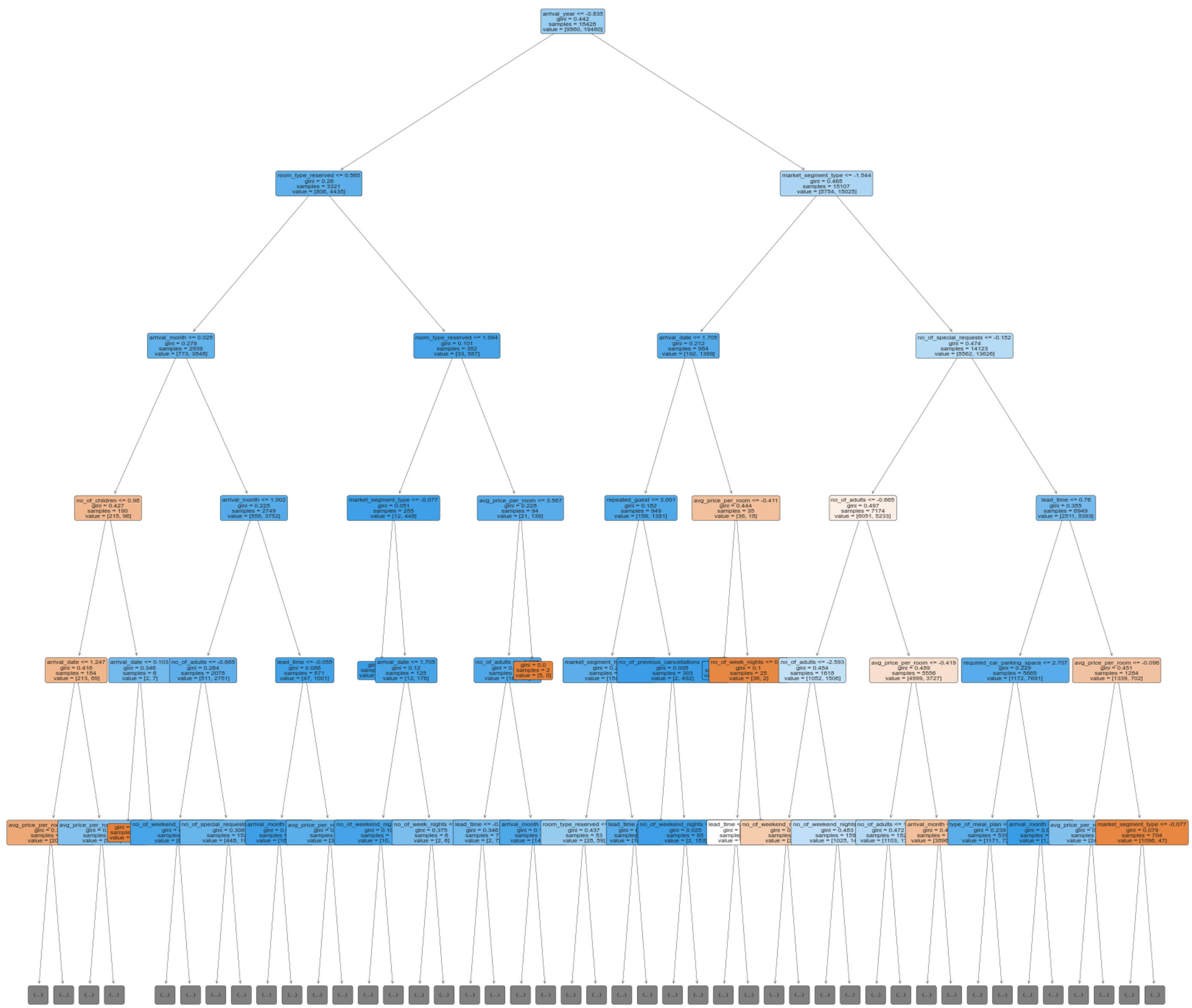
In [22]:
```
#We plotted a tree where the depth was 5, to show how the tree is built
plt.figure(figsize=(20,20))
plot_tree(forest_classifier.estimators_[1], feature_names=X_not_altered.columns, filled=
```

Out[22]:
```
[Text(0.4791666666666667, 0.9285714285714286, 'arrival_year <= -0.835\ngini = 0.442\nsam
ples = 18428\nvalue = [9560, 19460]'),
 Text(0.2569444444444444, 0.7857142857142857, 'room_type_reserved <= 0.565\ngini = 0.26
\nsamples = 3321\nvalue = [806, 4435]'),
 Text(0.1361111111111111, 0.6428571428571429, 'arrival_month <= 0.025\ngini = 0.279\nsam
ples = 2939\nvalue = [773, 3848]'),
 Text(0.07222222222222222, 0.5, 'no_of_children <= 0.98\ngini = 0.427\nsamples = 190\nva
lue = [215, 96]'),
 Text(0.044444444444444446, 0.35714285714285715, 'arrival_date <= 1.247\ngini = 0.416\ns
amples = 184\nvalue = [213, 89]'),
 Text(0.022222222222222223, 0.21428571428571427, 'avg_price_per_room <= -0.155\ngini =
0.366\nsamples = 163\nvalue = [204, 65]'),
 Text(0.011111111111111112, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.03333333333333333, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.06666666666666667, 0.21428571428571427, 'avg_price_per_room <= -1.311\ngini = 0.
397\nsamples = 21\nvalue = [9, 24]'),
```

```
 Text(0.05555555555555555, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.07777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.1, 0.35714285714285715, 'arrival_date <= 0.103\ngini = 0.346\nsamples = 6\nvalue
= [2, 7]'),
 Text(0.08888888888888889, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [2,
0]'),
 Text(0.1111111111111111, 0.21428571428571427, 'gini = 0.0\nsamples = 5\nvalue = [0,
7]'),
 Text(0.2, 0.5, 'arrival_month <= 1.002\ngini = 0.225\nsamples = 2749\nvalue = [558, 375
2]'),
 Text(0.15555555555555556, 0.35714285714285715, 'no_of_adults <= -0.665\ngini = 0.264\ns
amples = 2078\nvalue = [511, 2751]'),
 Text(0.13333333333333333, 0.21428571428571427, 'no_of_weekend_nights <= -0.357\ngini =
0.133\nsamples = 556\nvalue = [63, 819]'),
 Text(0.12222222222222222, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.14444444444444443, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.17777777777777778, 0.21428571428571427, 'no_of_special_requests <= -0.152\ngini
= 0.306\nsamples = 1522\nvalue = [448, 1932]'),
 Text(0.16666666666666666, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.18888888888888888, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.24444444444444444, 0.35714285714285715, 'lead_time <= -0.055\ngini = 0.086\nsamp
les = 671\nvalue = [47, 1001]'),
 Text(0.2222222222222222, 0.21428571428571427, 'arrival_month <= 1.328\ngini = 0.038\nsa
mples = 528\nvalue = [16, 817]'),
 Text(0.2111111111111111, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.23333333333333334, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.26666666666666666, 0.21428571428571427, 'avg_price_per_room <= -0.911\ngini = 0.
247\nsamples = 143\nvalue = [31, 184]'),
 Text(0.25555555555555554, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.2777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.37777777777777777, 0.6428571428571429, 'room_type_reserved <= 1.994\ngini = 0.10
1\nsamples = 382\nvalue = [33, 587]'),
 Text(0.32222222222222224, 0.5, 'market_segment_type <= -0.077\ngini = 0.051\nsamples =
288\nvalue = [12, 448]'),
 Text(0.3111111111111111, 0.35714285714285715, 'gini = 0.0\nsamples = 163\nvalue = [0, 2
72]'),
 Text(0.3333333333333333, 0.35714285714285715, 'arrival_date <= 1.705\ngini = 0.12\nsamp
les = 125\nvalue = [12, 176]'),
 Text(0.3111111111111111, 0.21428571428571427, 'no_of_weekend_nights <= 0.792\ngini = 0.
105\nsamples = 119\nvalue = [10, 170]'),
 Text(0.3, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.32222222222222224, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.35555555555555557, 0.21428571428571427, 'no_of_week_nights <= 0.21\ngini = 0.375
\nsamples = 6\nvalue = [2, 6]'),
 Text(0.34444444444444444, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.36666666666666664, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.43333333333333335, 0.5, 'avg_price_per_room <= 3.567\ngini = 0.228\nsamples = 94
\nvalue = [21, 139]'),
 Text(0.4222222222222222, 0.35714285714285715, 'no_of_adults <= -0.665\ngini = 0.185\nsa
mples = 92\nvalue = [16, 139]'),
 Text(0.4, 0.21428571428571427, 'lead_time <= -0.398\ngini = 0.346\nsamples = 7\nvalue =
[2, 7]'),
 Text(0.3888888888888889, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4111111111111111, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4444444444444444, 0.21428571428571427, 'arrival_month <= 1.002\ngini = 0.173\nsa
mples = 85\nvalue = [14, 132]'),
 Text(0.43333333333333335, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.45555555555555555, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4444444444444444, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue = [5,
0]'),
 Text(0.7013888888888888, 0.7857142857142857, 'market_segment_type <= -1.544\ngini = 0.4
65\nsamples = 15107\nvalue = [8754, 15025]'),
 Text(0.5805555555555556, 0.6428571428571429, 'arrival_date <= 1.705\ngini = 0.212\nsamp
les = 984\nvalue = [192, 1399]'),
 Text(0.5388888888888889, 0.5, 'repeated_guest <= 3.001\ngini = 0.182\nsamples = 949\nva
lue = [156, 1381]'),
```

```
 Text(0.5111111111111111, 0.35714285714285715, 'market_segment_type <= -4.479\ngini = 0.
252\nsamples = 646\nvalue = [154, 889]'),
 Text(0.4888888888888889, 0.21428571428571427, 'room_type_reserved <= 0.565\ngini = 0.43
7\nsamples = 53\nvalue = [28, 59]'),
 Text(0.4777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5333333333333333, 0.21428571428571427, 'lead_time <= 0.632\ngini = 0.229\nsample
s = 593\nvalue = [126, 830]'),
 Text(0.5222222222222223, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5444444444444444, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5666666666666667, 0.35714285714285715, 'no_of_previous_cancellations <= 1.294\ng
ini = 0.008\nsamples = 303\nvalue = [2, 492]'),
 Text(0.5555555555555556, 0.21428571428571427, 'gini = 0.0\nsamples = 208\nvalue = [0, 3
39]'),
 Text(0.5777777777777777, 0.21428571428571427, 'no_of_weekend_nights <= -0.357\ngini =
0.025\nsamples = 95\nvalue = [2, 153]'),
 Text(0.5666666666666667, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5888888888888889, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6222222222222222, 0.5, 'avg_price_per_room <= -0.411\ngini = 0.444\nsamples = 35
\nvalue = [36, 18]'),
 Text(0.6111111111111112, 0.35714285714285715, 'gini = 0.0\nsamples = 10\nvalue = [0, 1
6]'),
 Text(0.6333333333333333, 0.35714285714285715, 'no_of_week_nights <= 0.21\ngini = 0.1\ns
amples = 25\nvalue = [36, 2]'),
 Text(0.6222222222222222, 0.21428571428571427, 'lead_time <= -0.905\ngini = 0.5\nsamples
= 3\nvalue = [2, 2]'),
 Text(0.6111111111111112, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6333333333333333, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6444444444444445, 0.21428571428571427, 'gini = 0.0\nsamples = 22\nvalue = [34,
0]'),
 Text(0.8222222222222222, 0.6428571428571429, 'no_of_special_requests <= -0.152\ngini =
0.474\nsamples = 14123\nvalue = [8562, 13626]'),
 Text(0.7333333333333333, 0.5, 'no_of_adults <= -0.665\ngini = 0.497\nsamples = 7174\nva
lue = [6051, 5233]'),
 Text(0.6888888888888889, 0.35714285714285715, 'no_of_adults <= -2.593\ngini = 0.484\nsa
mples = 1618\nvalue = [1052, 1506]'),
 Text(0.6666666666666666, 0.21428571428571427, 'no_of_weekend_nights <= 0.792\ngini = 0.
467\nsamples = 23\nvalue = [27, 16]'),
 Text(0.6555555555555556, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7111111111111111, 0.21428571428571427, 'no_of_weekend_nights <= 0.792\ngini = 0.
483\nsamples = 1595\nvalue = [1025, 1490]'),
 Text(0.7, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7222222222222222, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7777777777777778, 0.35714285714285715, 'avg_price_per_room <= -0.419\ngini = 0.4
89\nsamples = 5556\nvalue = [4999, 3727]'),
 Text(0.7555555555555555, 0.21428571428571427, 'no_of_adults <= 1.263\ngini = 0.472\nsam
ples = 1829\nvalue = [1103, 1794]'),
 Text(0.7444444444444445, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7666666666666667, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8, 0.21428571428571427, 'arrival_month <= 1.328\ngini = 0.443\nsamples = 3727\nv
alue = [3896, 1933]'),
 Text(0.7888888888888889, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8111111111111111, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9111111111111111, 0.5, 'lead_time <= 0.76\ngini = 0.355\nsamples = 6949\nvalue =
[2511, 8393]'),
 Text(0.8666666666666667, 0.35714285714285715, 'required_car_parking_space <= 2.707\ngin
i = 0.229\nsamples = 5665\nvalue = [1172, 7691]'),
 Text(0.8444444444444444, 0.21428571428571427, 'type_of_meal_plan <= -0.015\ngini = 0.23
9\nsamples = 5397\nvalue = [1171, 7276]'),
 Text(0.8333333333333334, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8555555555555555, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8888888888888888, 0.21428571428571427, 'arrival_month <= 0.025\ngini = 0.005\nsa
mples = 268\nvalue = [1, 415]'),
 Text(0.8777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9, 0.07142857142857142, '\n  (...)  \n'),
```

```
 Text(0.9555555555555556, 0.35714285714285715, 'avg_price_per_room <= -0.096\ngini = 0.451\nsamples = 1284\nvalue = [1339, 702]'),
 Text(0.9333333333333333, 0.21428571428571427, 'avg_price_per_room <= -0.72\ngini = 0.395\nsamples = 580\nvalue = [243, 655]'),
 Text(0.9222222222222223, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9444444444444444, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9777777777777777, 0.21428571428571427, 'market_segment_type <= -0.077\ngini = 0.079\nsamples = 704\nvalue = [1096, 47]'),
 Text(0.9666666666666667, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9888888888888889, 0.07142857142857142, '\n  (...)  \n')]
```



xgBoost

In [223…
```python
n_estimators = np.array([100])
max_nvl = np.array([3, 4, 5, 6, 7, 9, 11])
values_grid = {'xgbclassifier__n_estimators': n_estimators, 'xgbclassifier__max_depth': 
classifier = make_pipeline(StandardScaler(), SMOTE(random_state=100), XGBClassifier())

gridXGBoost = GridSearchCV(classifier , param_grid = values_grid, cv = kf, scoring = sco
```

In [224…
```python
gridXGBoost.fit(X, y)
```

Out[224]:

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ StandardScaler
  ▸ SMOTE
  ▸ XGBClassifier
```

In [225…

```
columns = ["mean_test_accuracy", "mean_test_precision", "mean_test_recall", "mean_test_f
params = pd.DataFrame(gridXGBoost.cv_results_['params'])
scores = pd.DataFrame(gridXGBoost.cv_results_)[columns]
scores = pd.concat([params, scores], axis=1)

#rename columns
scores.columns = ['max_depth', 'n_estimators', 'accuracy', 'precision', 'recall', 'f1',
scores.sort_values(by=['accuracy'], ascending=False)
```

Out[225]:

	max_depth	n_estimators	accuracy	precision	recall	f1	roc_auc
6	11	100	0.897753	0.918842	0.930097	0.924423	0.956994
5	9	100	0.896320	0.918709	0.927915	0.923279	0.956821
4	7	100	0.893535	0.915723	0.926975	0.921305	0.954392
3	6	100	0.887829	0.911667	0.922590	0.917083	0.951227
2	5	100	0.884025	0.910894	0.917267	0.914054	0.947139
1	4	100	0.876003	0.905799	0.910255	0.908012	0.940132
0	3	100	0.860703	0.901044	0.890667	0.895811	0.928753

In [19]:

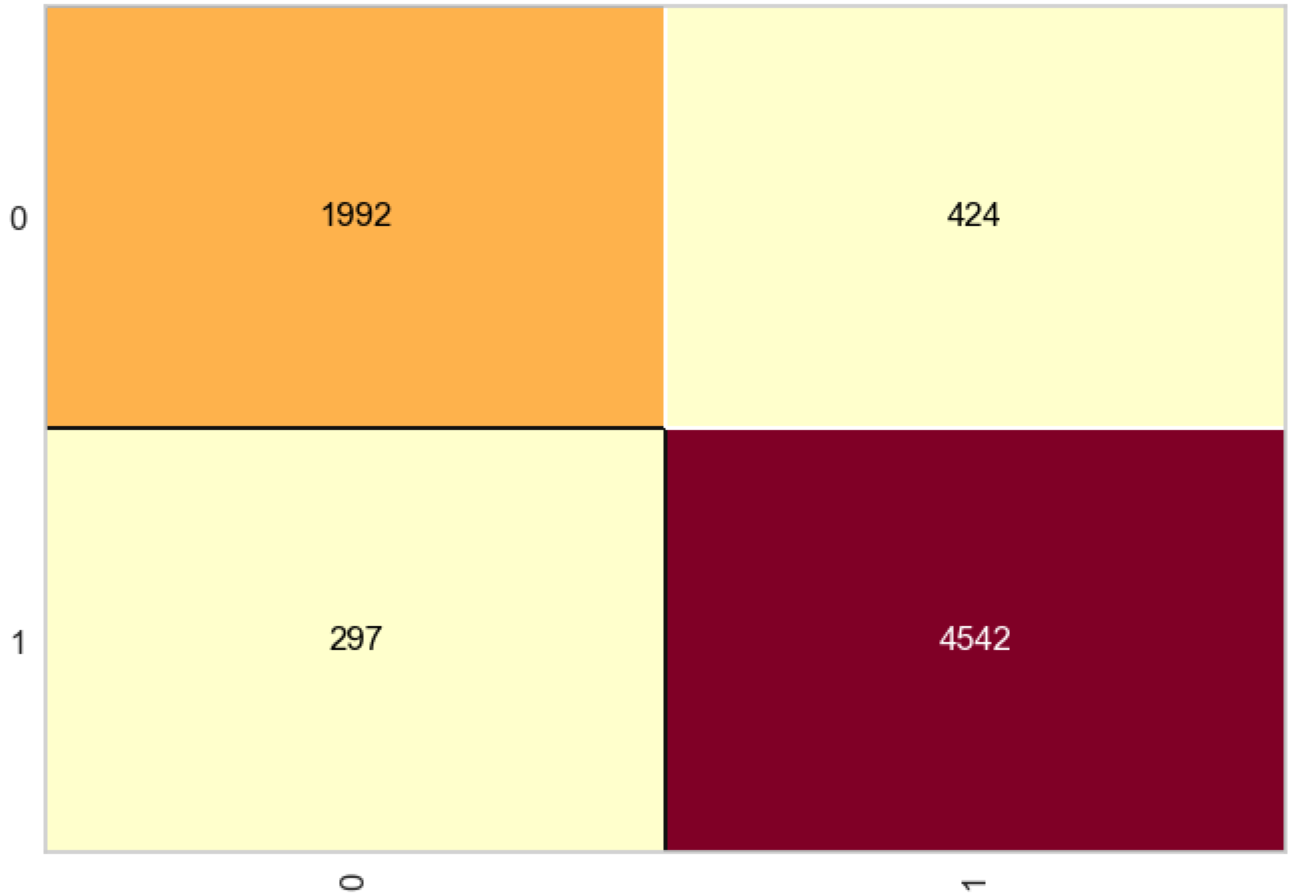
```
xgb_classifier = XGBClassifier(max_depth=11, n_estimators=100)
xgb_classifier.fit(x_train, y_train)

xgb_scores = cross_validate(xgb_classifier, X_standard, y, cv=kf, scoring=scoring, n_job
```

In [17]:

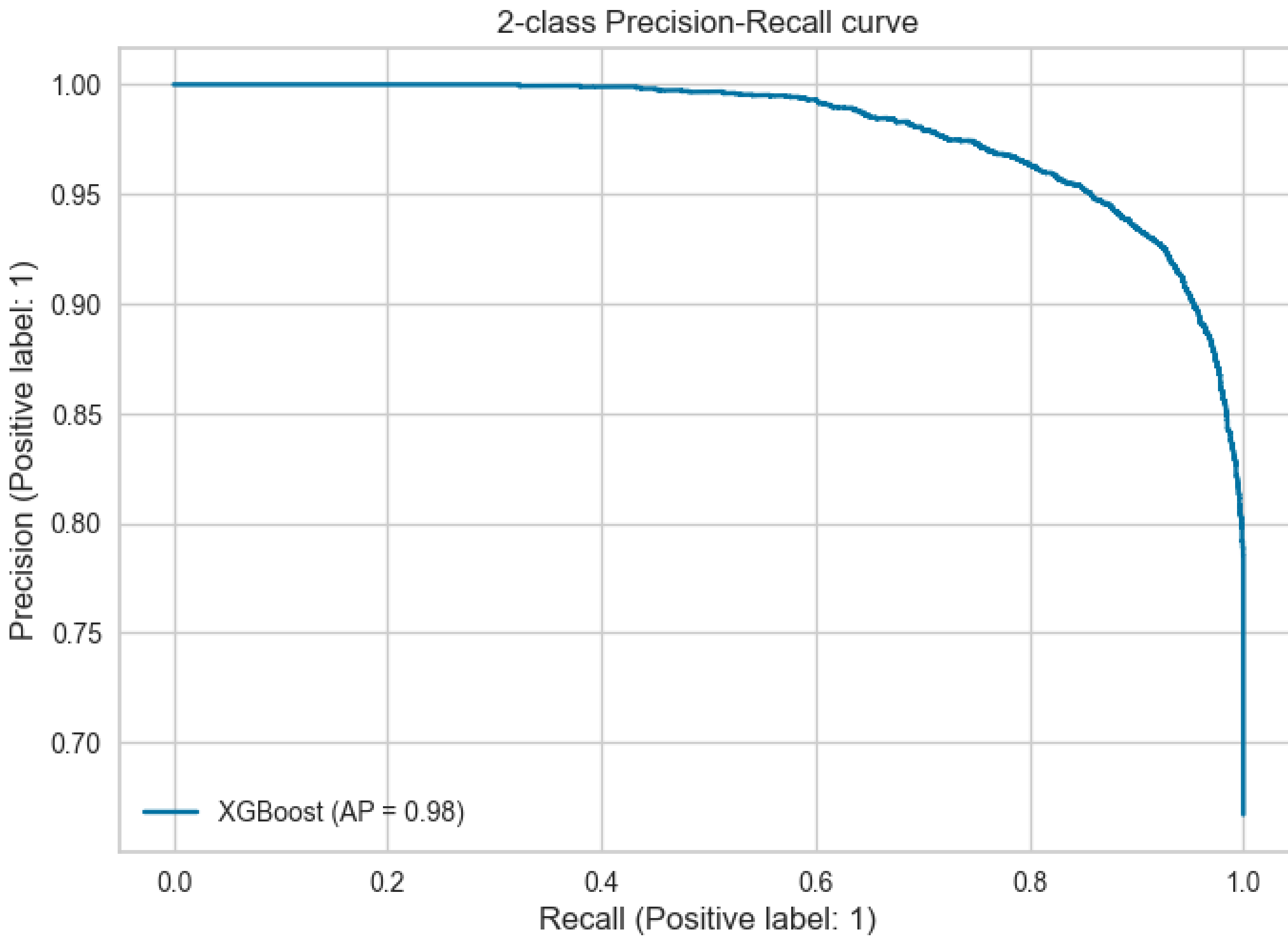
```
xgb_classifier = XGBClassifier(max_depth=11, n_estimators=100)
xgb_classifier.fit(x_train_balanced, y_train_balanced)
y_pred = xgb_classifier.predict(x_test)

cm = ConfusionMatrix(xgb_classifier)
cm.fit(x_train_balanced, y_train_balanced)
cm.score(x_test, y_test)
```

Out[17]:

```
0.9006202618883529
```



In [28]:
```
xgb_classifier = XGBClassifier(max_depth=11, n_estimators=100)
xgb_classifier.fit(x_train, y_train)
display = PrecisionRecallDisplay.from_estimator(
    xgb_classifier, x_test, y_test, name="XGBoost"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [21]:
```
y_pred = xgb_classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.87      0.82      0.85      2416
Not Canceled       0.91      0.94      0.93      4839

    accuracy                           0.90      7255
   macro avg       0.89      0.88      0.89      7255
weighted avg       0.90      0.90      0.90      7255
```

In [18]:
```
#We plotted a tree where the depth was 5, to show how the tree is built

plt.figure(figsize=(20,20))
ax = plt.subplot(111, title='XGBoost')

plot_tree_xgb(xgb_classifier, num_trees=1, ax=ax)
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.855491 to fit

(process:12652): GLib-GIO-WARNING **: 20:19:53.603: Unexpectedly, UWP app `Clipchamp.Cli
pchamp_2.5.15.0_neutral__yxz26nhyzhsrt' (AUMId `Clipchamp.Clipchamp_yxz26nhyzhsrt!App')
supports 41 extensions but has no verbs

(process:12652): GLib-GIO-WARNING **: 20:19:53.763: Unexpectedly, UWP app `Microsoft.Scr
eenSketch_11.2302.20.0_x64__8wekyb3d8bbwe' (AUMId `Microsoft.ScreenSketch_8wekyb3d8bbwe!
App') supports 29 extensions but has no verbs
```

Out[18]:
```
<Axes: title={'center': 'XGBoost'}>
```

XGBoost

SVM

In [17]:
```python
# use SVC with linear kernel
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
values_grid = {'svc__kernel': kernels, 'svc__C': [0.1, 1, 10, 20] }
classifier = make_pipeline(StandardScaler(), SMOTE(random_state=100), SVC())
gridSVM = GridSearchCV(classifier , param_grid = values_grid, cv = kf, scoring = scoring
```

In [24]:
```python
gridSVM.fit(X, y)
```

Out[24]:

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ StandardScaler
  ▸ SMOTE
  ▸ SVC
```

In [26]:
```python
columns = ["mean_test_accuracy", "mean_test_precision", "mean_test_recall", "mean_test_f
params = pd.DataFrame(gridSVM.cv_results_['params'])
scores = pd.DataFrame(gridSVM.cv_results_)[columns]
scores = pd.concat([params, scores], axis=1)

#rename columns
scores.columns = ['C', 'kernel', 'accuracy', 'precision', 'recall', 'f1', 'roc_auc']
scores.sort_values(by=['accuracy'], ascending=False)
```

Out[26]:

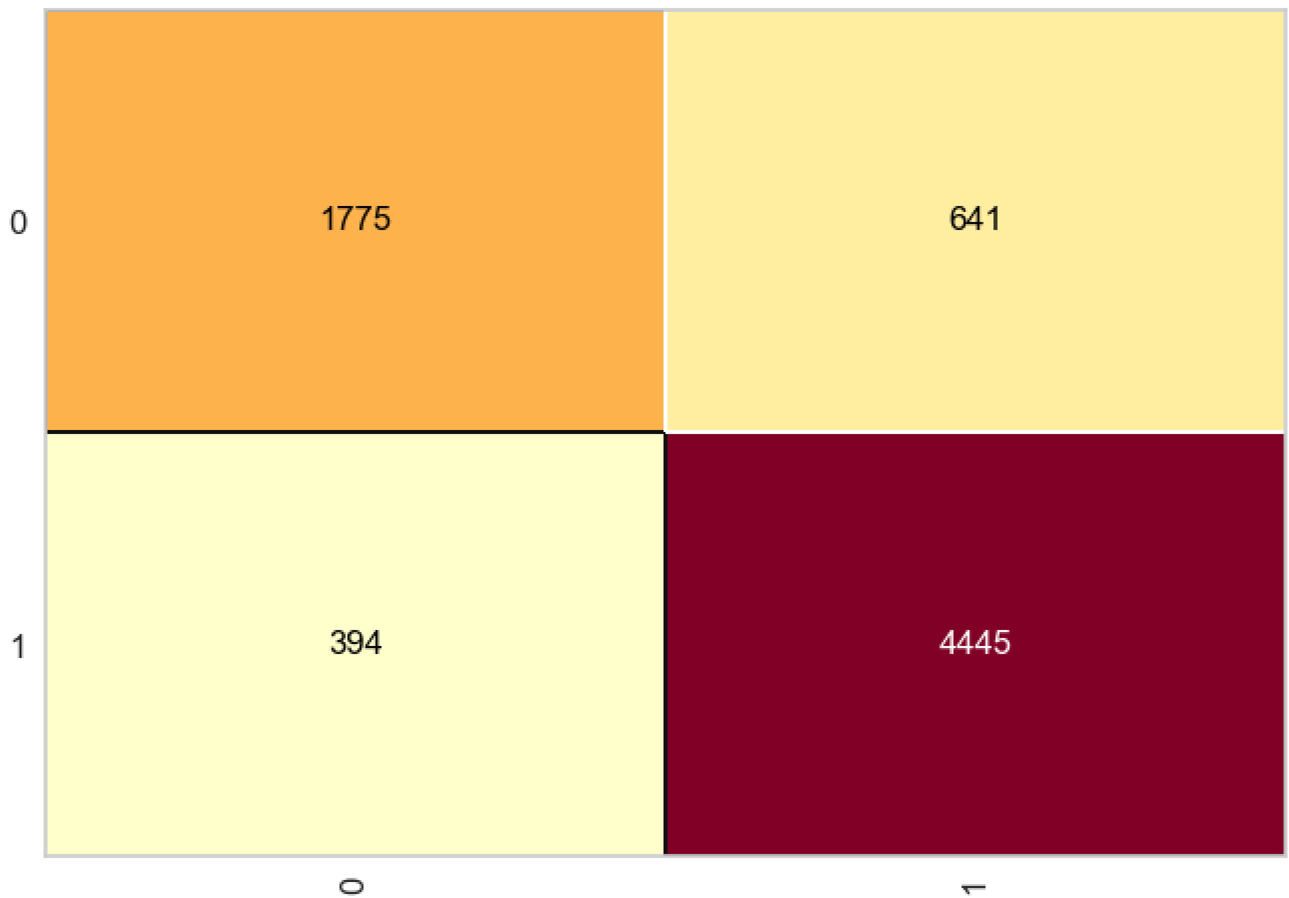
	C	kernel	accuracy	precision	recall	f1	roc_auc
14	20.0	rbf	0.849649	0.914288	0.856709	0.884555	0.916821
10	10.0	rbf	0.846478	0.913634	0.852236	0.881860	0.915063
13	20.0	poly	0.825996	0.906957	0.825952	0.864549	0.898181
6	1.0	rbf	0.825555	0.905868	0.826438	0.864319	0.904624
9	10.0	poly	0.824976	0.906038	0.825284	0.863761	0.896817
5	1.0	poly	0.815134	0.899602	0.816129	0.855814	0.890563
2	0.1	rbf	0.806671	0.894036	0.808274	0.848971	0.888114
1	0.1	poly	0.800882	0.895751	0.796590	0.843233	0.878429
12	20.0	linear	0.777395	0.878354	0.776474	0.824253	0.856372
0	0.1	linear	0.777367	0.878348	0.776432	0.824227	0.856372
4	1.0	linear	0.777312	0.878303	0.776392	0.824183	0.856374
8	10.0	linear	0.777229	0.878286	0.776269	0.824107	0.856371
3	0.1	sigmoid	0.693784	0.825732	0.690257	0.751920	0.741730
15	20.0	sigmoid	0.672171	0.806583	0.674032	0.734348	0.707559
11	10.0	sigmoid	0.672006	0.806656	0.673633	0.734140	0.707558

7	1.0	sigmoid	0.668973	0.805825	0.668895	0.730964	0.712123

In [22]:
```
SVC_classifier = SVC(kernel='rbf', C=20)
SVC_classifier.fit(x_train_balanced, y_train_balanced)

svm_scores = cross_validate(SVC_classifier, X_standard, y, cv=kf, scoring=scoring, n_job
```

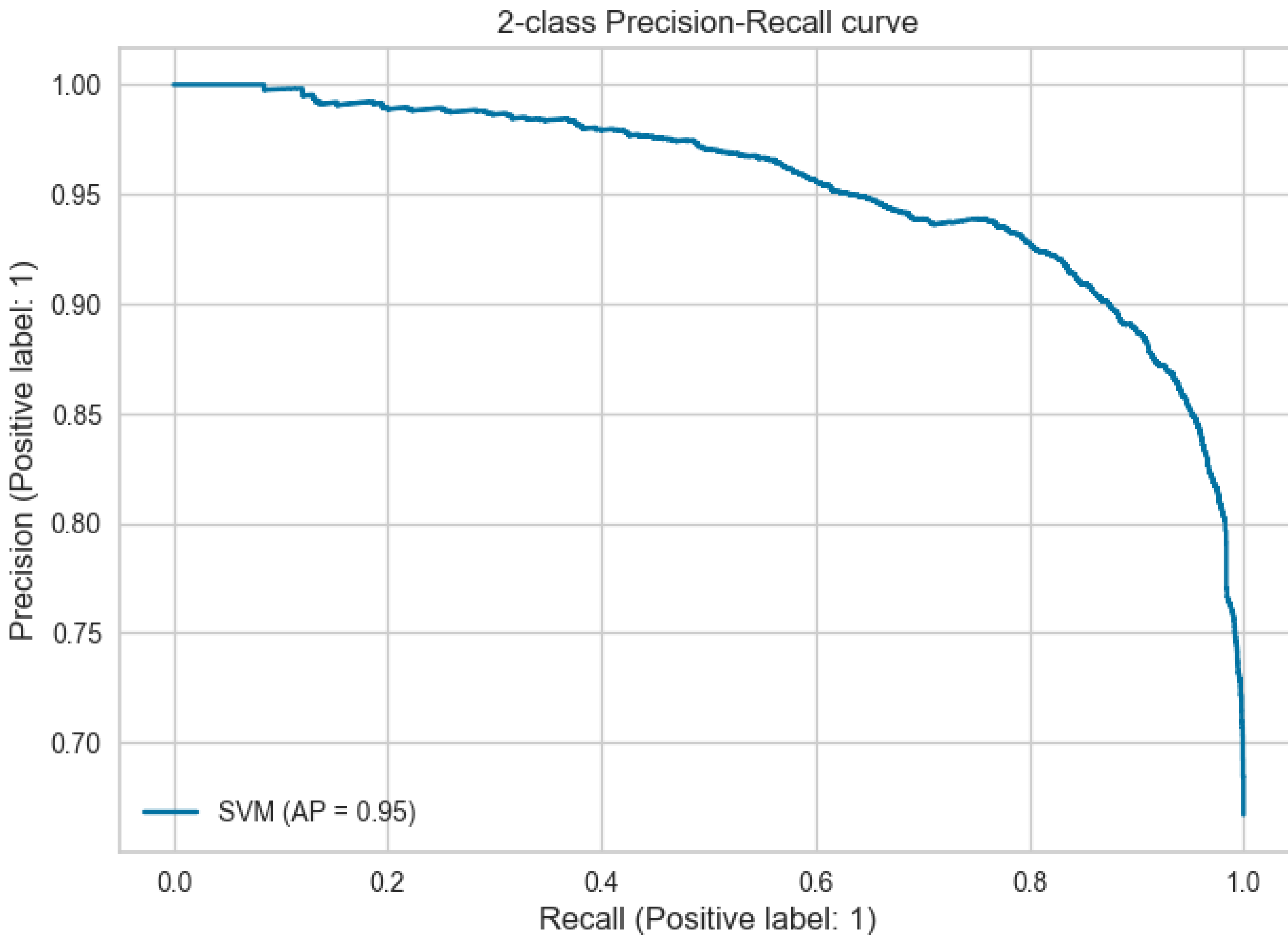
In [27]:
```
SVC_classifier = SVC(kernel='rbf', C=20)
SVC_classifier.fit(x_train_balanced, y_train_balanced)
y_pred = SVC_classifier.predict(x_test)

cm = ConfusionMatrix(SVC_classifier)
cm.fit(x_train_balanced, y_train_balanced)
cm.score(x_test, y_test)
```

Out[27]:
```
0.8573397656788422
```



In [ ]:
```
SVC_classifier = SVC(kernel='rbf', C=20)
SVC_classifier.fit(x_train, y_train)
```

In [29]:
```
display = PrecisionRecallDisplay.from_estimator(
    SVC_classifier, x_test, y_test, name="SVM"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [27]:
```
y_pred = SVC_classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.82      0.73      0.77      2416
Not Canceled       0.87      0.92      0.90      4839

    accuracy                           0.86      7255
   macro avg       0.85      0.83      0.83      7255
weighted avg       0.86      0.86      0.86      7255
```

ExtraTrees

In [15]:
```
n_estimators = np.array([10, 100, 200])
alg = ['entropy', 'gini', 'log_loss']
values_grid = {'extratreesclassifier__n_estimators': n_estimators, 'extratreesclassifier

classifier = make_pipeline(StandardScaler(), SMOTE(random_state=100), ExtraTreesClassifi
gridExTrees = GridSearchCV(classifier, param_grid = values_grid, cv = kf, scoring = scor
```

In [29]:
```
gridExTrees.fit(X, y)
```

Out[29]:

```
▸ GridSearchCV
▸ estimator: Pipeline
    ▸ StandardScaler
    ▸ SMOTE
    ▸ ExtraTreesClassifier
```

In [30]:
```python
columns = ["mean_test_accuracy", "mean_test_precision", "mean_test_recall", "mean_test_f
params = pd.DataFrame(gridExTrees.cv_results_['params'])
scores = pd.DataFrame(gridExTrees.cv_results_)[columns]
scores = pd.concat([params, scores], axis=1)

#rename columns
scores.columns = ['criterion', 'n_estimators', 'accuracy', 'precision', 'recall', 'f1',
scores.sort_values(by=['accuracy'], ascending=False)
# scores
```

Out[30]:

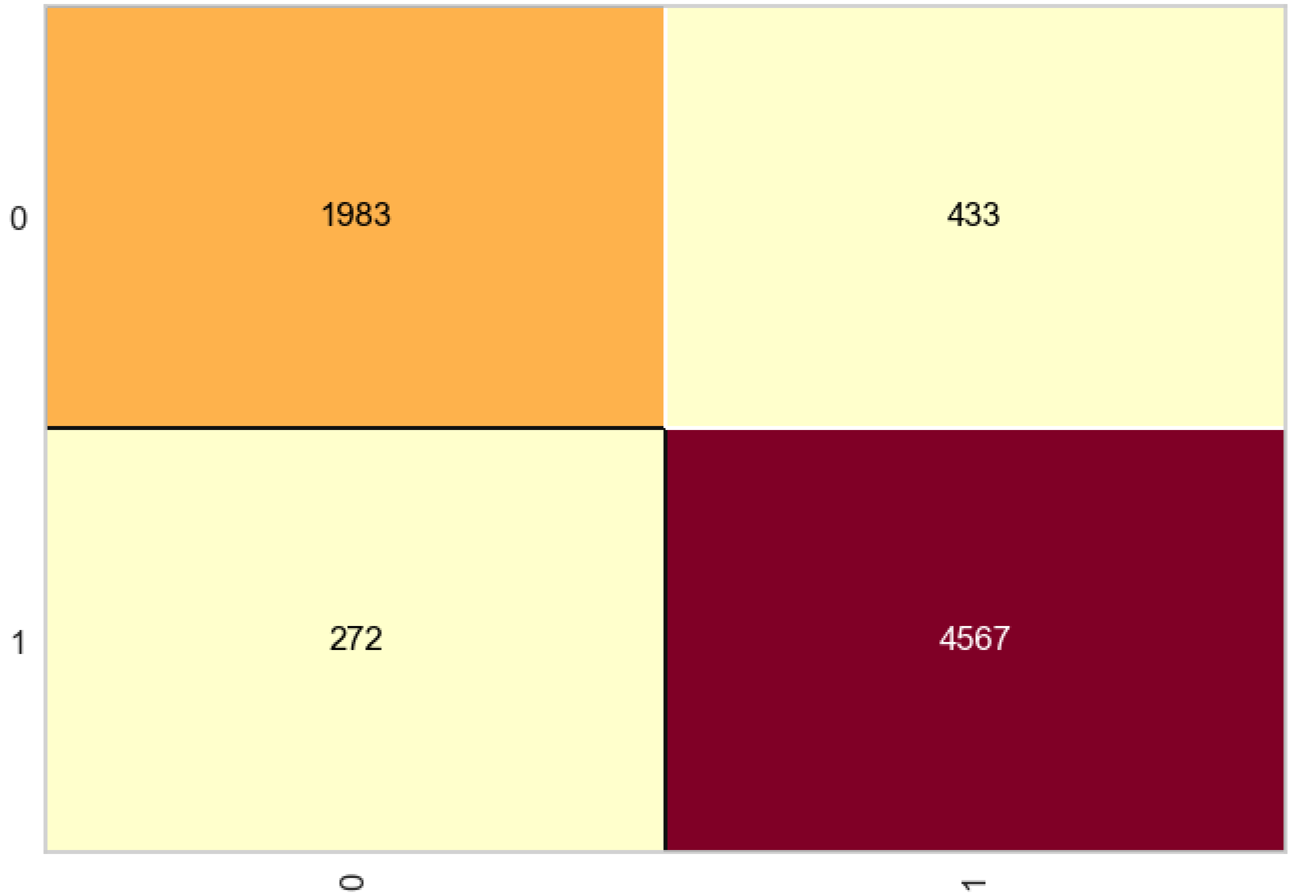
	criterion	n_estimators	accuracy	precision	recall	f1	roc_auc
2	entropy	200	0.895217	0.918807	0.926003	0.922388	0.951749
8	log_loss	200	0.895190	0.918699	0.926083	0.922374	0.951866
7	log_loss	100	0.894666	0.918921	0.924976	0.921934	0.951593
4	gini	100	0.894363	0.918741	0.924689	0.921704	0.951284
5	gini	200	0.894170	0.917930	0.925341	0.921618	0.951707
1	entropy	100	0.893728	0.918263	0.924239	0.921236	0.951491
6	log_loss	10	0.884769	0.921864	0.905374	0.913541	0.940068
0	entropy	10	0.883363	0.921018	0.904060	0.912451	0.940124
3	gini	10	0.883032	0.922089	0.902325	0.912095	0.939588

In [16]:
```python
extraTrees_classifier = ExtraTreesClassifier(criterion='entropy', n_estimators=200, max_
extraTrees_classifier.fit(x_train_balanced, y_train_balanced)
y_pred = extraTrees_classifier.predict(x_test)

extra_scores = cross_validate(extraTrees_classifier, X_standard, y, cv=kf, scoring=scori
```

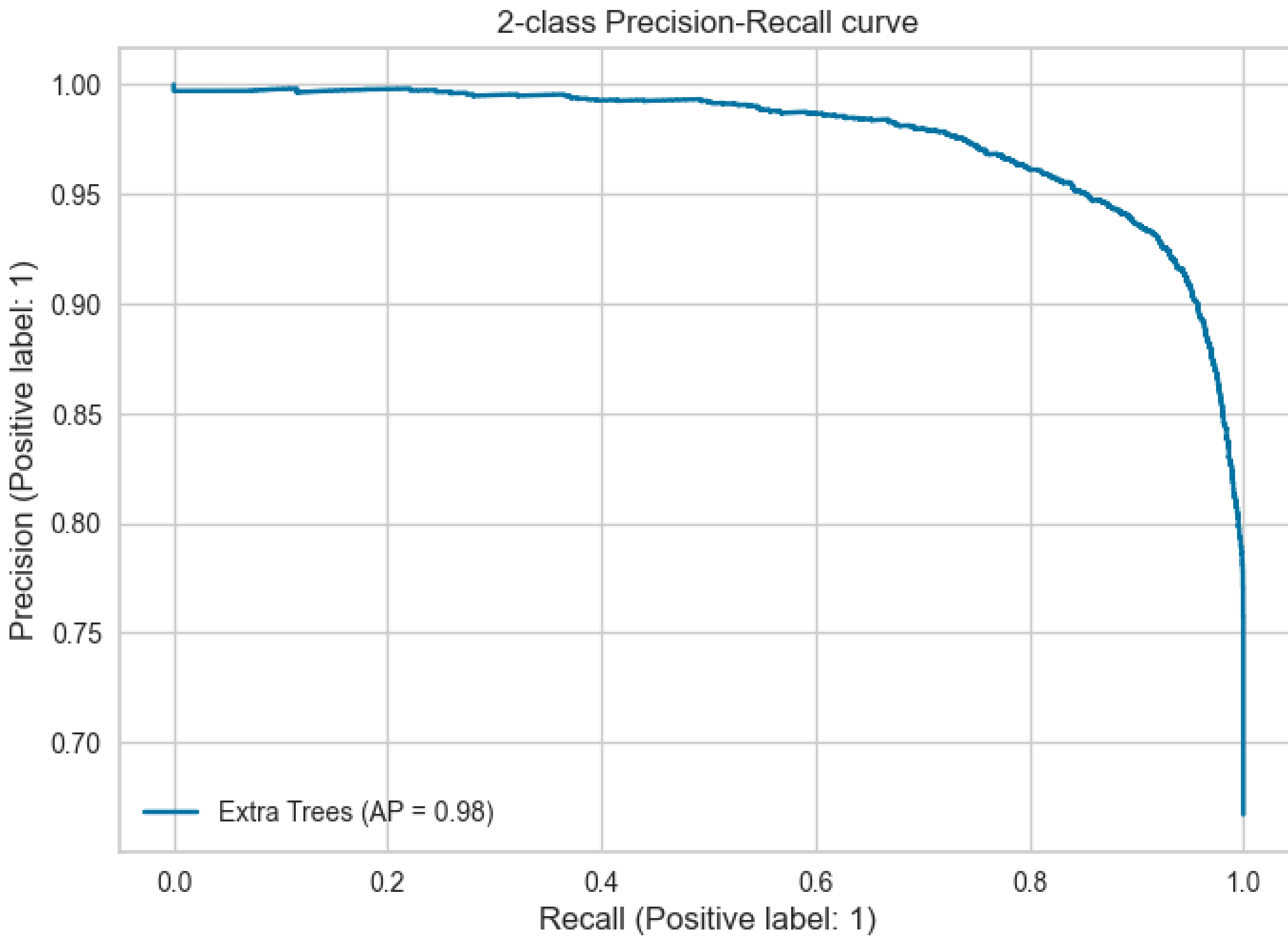
In [177…
```python
extraTrees_classifier = ExtraTreesClassifier(criterion='entropy', n_estimators=200, max_
extraTrees_classifier.fit(x_train_balanced, y_train_balanced)
y_pred = extraTrees_classifier.predict(x_test)

cm = ConfusionMatrix(extraTrees_classifier)
cm.fit(x_train_balanced, y_train_balanced)
cm.score(x_test, y_test)
```

Out[177]:

```
0.9028256374913852
```



In [30]:
```
extraTrees_classifier = ExtraTreesClassifier(criterion='entropy', n_estimators=200, max_
extraTrees_classifier.fit(x_train_balanced, y_train_balanced)
display = PrecisionRecallDisplay.from_estimator(
    extraTrees_classifier, x_test, y_test, name="Extra Trees"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [31]:
```
y_pred = extraTrees_classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.88      0.82      0.85      2416
Not Canceled       0.91      0.95      0.93      4839

    accuracy                           0.90      7255
   macro avg       0.90      0.88      0.89      7255
weighted avg       0.90      0.90      0.90      7255
```

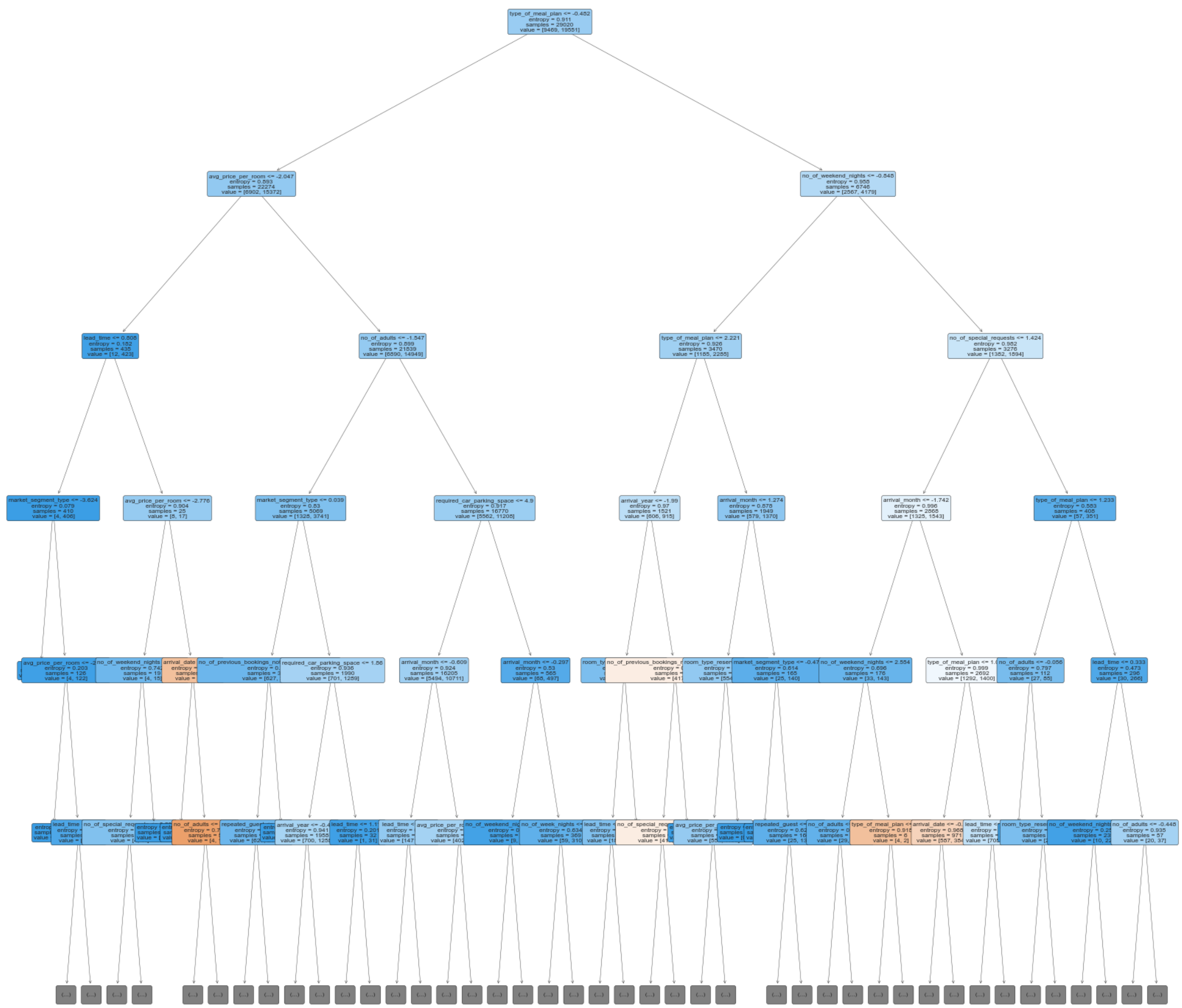
In [178…
```
#We plotted a tree where the depth was 5, to show how the tree is built
plt.figure(figsize=(20,20))
plot_tree(extraTrees_classifier.estimators_[1], feature_names=X_not_altered.columns, fil
```

Out[178]:
```
[Text(0.45694444444444443, 0.9285714285714286, 'type_of_meal_plan <= -0.482\nentropy =
0.911\nsamples = 29020\nvalue = [9469, 19551]'),
 Text(0.19583333333333333, 0.7857142857142857, 'avg_price_per_room <= -2.047\nentropy =
0.893\nsamples = 22274\nvalue = [6902, 15372]'),
 Text(0.07222222222222222, 0.6428571428571429, 'lead_time <= 0.808\nentropy = 0.182\nsam
ples = 435\nvalue = [12, 423]'),
 Text(0.022222222222222223, 0.5, 'market_segment_type <= -3.624\nentropy = 0.079\nsample
s = 410\nvalue = [4, 406]'),
 Text(0.011111111111111112, 0.35714285714285715, 'entropy = 0.0\nsamples = 284\nvalue =
[0, 284]'),
 Text(0.03333333333333333, 0.35714285714285715, 'avg_price_per_room <= -2.814\nentropy =
0.203\nsamples = 126\nvalue = [4, 122]'),
 Text(0.022222222222222223, 0.21428571428571427, 'entropy = 0.0\nsamples = 83\nvalue =
[0, 83]'),
 Text(0.044444444444444446, 0.21428571428571427, 'lead_time <= -0.69\nentropy = 0.446\ns
amples = 43\nvalue = [4, 39]'),
```

```
 Text(0.03333333333333333, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.05555555555555555, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.12222222222222222, 0.5, 'avg_price_per_room <= -2.776\nentropy = 0.904\nsamples
= 25\nvalue = [8, 17]'),
 Text(0.1, 0.35714285714285715, 'no_of_weekend_nights <= 1.073\nentropy = 0.742\nsamples
= 19\nvalue = [4, 15]'),
 Text(0.08888888888888889, 0.21428571428571427, 'no_of_special_requests <= 0.235\nentrop
y = 0.837\nsamples = 15\nvalue = [4, 11]'),
 Text(0.07777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.1, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.1111111111111111, 0.21428571428571427, 'entropy = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.14444444444444443, 0.35714285714285715, 'arrival_date <= 0.741\nentropy = 0.918
\nsamples = 6\nvalue = [4, 2]'),
 Text(0.13333333333333333, 0.21428571428571427, 'entropy = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.15555555555555556, 0.21428571428571427, 'no_of_adults <= -0.964\nentropy = 0.722
\nsamples = 5\nvalue = [4, 1]'),
 Text(0.14444444444444443, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.16666666666666666, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.3194444444444444, 0.6428571428571429, 'no_of_adults <= -1.547\nentropy = 0.899\n
samples = 21839\nvalue = [6890, 14949]'),
 Text(0.2388888888888889, 0.5, 'market_segment_type <= 0.039\nentropy = 0.83\nsamples =
5069\nvalue = [1328, 3741]'),
 Text(0.2111111111111111, 0.35714285714285715, 'no_of_previous_bookings_not_canceled <=
23.697\nentropy = 0.729\nsamples = 3079\nvalue = [627, 2452]'),
 Text(0.2, 0.21428571428571427, 'repeated_guest <= 4.994\nentropy = 0.731\nsamples = 306
1\nvalue = [627, 2434]'),
 Text(0.18888888888888888, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.2111111111111111, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.2222222222222222, 0.21428571428571427, 'entropy = 0.0\nsamples = 18\nvalue = [0,
18]'),
 Text(0.26666666666666666, 0.35714285714285715, 'required_car_parking_space <= 1.86\nent
ropy = 0.936\nsamples = 1990\nvalue = [701, 1289]'),
 Text(0.24444444444444444, 0.21428571428571427, 'arrival_year <= -0.481\nentropy = 0.941
\nsamples = 1958\nvalue = [700, 1258]'),
 Text(0.23333333333333334, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.25555555555555554, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.28888888888888886, 0.21428571428571427, 'lead_time <= 1.117\nentropy = 0.201\nsa
mples = 32\nvalue = [1, 31]'),
 Text(0.2777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.3, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4, 0.5, 'required_car_parking_space <= 4.9\nentropy = 0.917\nsamples = 16770\nva
lue = [5562, 11208]'),
 Text(0.35555555555555557, 0.35714285714285715, 'arrival_month <= -0.609\nentropy = 0.92
4\nsamples = 16205\nvalue = [5494, 10711]'),
 Text(0.3333333333333333, 0.21428571428571427, 'lead_time <= 1.682\nentropy = 0.894\nsam
ples = 4733\nvalue = [1471, 3262]'),
 Text(0.32222222222222224, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.34444444444444444, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.37777777777777777, 0.21428571428571427, 'avg_price_per_room <= -0.886\nentropy =
0.935\nsamples = 11472\nvalue = [4023, 7449]'),
 Text(0.36666666666666664, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.3888888888888889, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4444444444444444, 0.35714285714285715, 'arrival_month <= -0.297\nentropy = 0.53
\nsamples = 565\nvalue = [68, 497]'),
 Text(0.4222222222222222, 0.21428571428571427, 'no_of_weekend_nights <= 1.366\nentropy =
0.269\nsamples = 196\nvalue = [9, 187]'),
 Text(0.4111111111111111, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.43333333333333335, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4666666666666667, 0.21428571428571427, 'no_of_week_nights <= 4.128\nentropy = 0.
634\nsamples = 369\nvalue = [59, 310]'),
 Text(0.45555555555555555, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7180555555555556, 0.7857142857142857, 'no_of_weekend_nights <= -0.848\nentropy =
0.958\nsamples = 6746\nvalue = [2567, 4179]'),
```

```
 Text(0.5888888888888889, 0.6428571428571429, 'type_of_meal_plan <= 2.221\nentropy = 0.9
26\nsamples = 3470\nvalue = [1185, 2285]'),
 Text(0.5444444444444444, 0.5, 'arrival_year <= -1.99\nentropy = 0.97\nsamples = 1521\nv
alue = [606, 915]'),
 Text(0.5222222222222223, 0.35714285714285715, 'room_type_reserved <= 0.848\nentropy =
0.817\nsamples = 745\nvalue = [189, 556]'),
 Text(0.5111111111111111, 0.21428571428571427, 'lead_time <= -0.989\nentropy = 0.832\nsa
mples = 718\nvalue = [189, 529]'),
 Text(0.5, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5222222222222223, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5333333333333333, 0.21428571428571427, 'entropy = 0.0\nsamples = 27\nvalue = [0,
27]'),
 Text(0.5666666666666667, 0.35714285714285715, 'no_of_previous_bookings_not_canceled <=
0.033\nentropy = 0.996\nsamples = 776\nvalue = [417, 359]'),
 Text(0.5555555555555556, 0.21428571428571427, 'no_of_special_requests <= -0.64\nentropy
= 0.996\nsamples = 773\nvalue = [417, 356]'),
 Text(0.5444444444444444, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5666666666666667, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5777777777777777, 0.21428571428571427, 'entropy = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.6333333333333333, 0.5, 'arrival_month <= 1.274\nentropy = 0.878\nsamples = 1949
\nvalue = [579, 1370]'),
 Text(0.6111111111111112, 0.35714285714285715, 'room_type_reserved <= 0.251\nentropy =
0.894\nsamples = 1784\nvalue = [554, 1230]'),
 Text(0.6, 0.21428571428571427, 'avg_price_per_room <= 1.6\nentropy = 0.901\nsamples = 1
750\nvalue = [554, 1196]'),
 Text(0.5888888888888889, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6111111111111112, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6222222222222222, 0.21428571428571427, 'entropy = 0.0\nsamples = 34\nvalue = [0,
34]'),
 Text(0.6555555555555556, 0.35714285714285715, 'market_segment_type <= -0.47\nentropy =
0.614\nsamples = 165\nvalue = [25, 140]'),
 Text(0.6444444444444445, 0.21428571428571427, 'entropy = 0.0\nsamples = 5\nvalue = [0,
5]'),
 Text(0.6666666666666666, 0.21428571428571427, 'repeated_guest <= 5.237\nentropy = 0.625
\nsamples = 160\nvalue = [25, 135]'),
 Text(0.6555555555555556, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.6777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8472222222222222, 0.6428571428571429, 'no_of_special_requests <= 1.424\nentropy
= 0.982\nsamples = 3276\nvalue = [1382, 1894]'),
 Text(0.7777777777777778, 0.5, 'arrival_month <= -1.742\nentropy = 0.996\nsamples = 2868
\nvalue = [1325, 1543]'),
 Text(0.7333333333333333, 0.35714285714285715, 'no_of_weekend_nights <= 2.554\nentropy =
0.696\nsamples = 176\nvalue = [33, 143]'),
 Text(0.7111111111111111, 0.21428571428571427, 'no_of_adults <= -0.914\nentropy = 0.659
\nsamples = 170\nvalue = [29, 141]'),
 Text(0.7, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7222222222222222, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7555555555555555, 0.21428571428571427, 'type_of_meal_plan <= 1.303\nentropy = 0.
918\nsamples = 6\nvalue = [4, 2]'),
 Text(0.7444444444444445, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7666666666666667, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8222222222222222, 0.35714285714285715, 'type_of_meal_plan <= 1.072\nentropy = 0.
999\nsamples = 2692\nvalue = [1292, 1400]'),
 Text(0.8, 0.21428571428571427, 'arrival_date <= -0.29\nentropy = 0.968\nsamples = 971\n
value = [587, 384]'),
 Text(0.7888888888888889, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8111111111111111, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8444444444444444, 0.21428571428571427, 'lead_time <= -0.626\nentropy = 0.976\nsa
mples = 1721\nvalue = [705, 1016]'),
 Text(0.8333333333333334, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8555555555555555, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9166666666666666, 0.5, 'type_of_meal_plan <= 1.233\nentropy = 0.583\nsamples = 4
08\nvalue = [57, 351]'),
 Text(0.8777777777777778, 0.35714285714285715, 'no_of_adults <= -0.056\nentropy = 0.797
\nsamples = 112\nvalue = [27, 85]'),
```

```
 Text(0.8666666666666667, 0.21428571428571427, 'entropy = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.8888888888888888, 0.21428571428571427, 'room_type_reserved <= 2.036\nentropy =
0.811\nsamples = 108\nvalue = [27, 81]'),
 Text(0.8777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9555555555555556, 0.35714285714285715, 'lead_time <= 0.333\nentropy = 0.473\nsam
ples = 296\nvalue = [30, 266]'),
 Text(0.9333333333333333, 0.21428571428571427, 'no_of_weekend_nights <= 3.658\nentropy =
0.251\nsamples = 239\nvalue = [10, 229]'),
 Text(0.9222222222222223, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9444444444444444, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9777777777777777, 0.21428571428571427, 'no_of_adults <= -0.448\nentropy = 0.935
\nsamples = 57\nvalue = [20, 37]'),
 Text(0.9666666666666667, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.9888888888888889, 0.07142857142857142, '\n  (...)  \n')]
```



Final metrics

In [12]:

```python
f1_nn = 2 * 0.914202 *0.857168 / (0.914202 + 0.857168)
print(f1_nn)
```

```
nn_scores = [0.8821262717247009, 0.9142017483711242, 0.857167637348175, f1_nn, 0.9239423
```

```
0.8847668188306226
```

In [25]:
```
all_scores = [
    [nb_scores['test_accuracy'].mean(), nb_scores['test_precision'].mean(), nb_scores['t
    [knn_scores['test_accuracy'].mean(), knn_scores['test_precision'].mean(), knn_scores
    [*nn_scores],
    [tree_scores['test_accuracy'].mean(), tree_scores['test_precision'].mean(), tree_sco
    [forest_scores['test_accuracy'].mean(), forest_scores['test_precision'].mean(), fore
    [xgb_scores['test_accuracy'].mean(), xgb_scores['test_precision'].mean(), xgb_scores
    [svm_scores['test_accuracy'].mean(), svm_scores['test_precision'].mean(), svm_scores
    [extra_scores['test_accuracy'].mean(), extra_scores['test_precision'].mean(), extra_
]

models = [
    'Naive Bayes',
    'KNN',
    "Neural Network",
    'Decision Trees',
    'Random Forest',
    'XGBoost',
    'SVM',
    'Extra Trees'
]
```

In [26]:
```
# create a dataframe with the results
results = pd.DataFrame(all_scores, columns=['Accuracy', 'Precision', 'Recall', 'F1', 'RO
results
```

Out[26]:

	Accuracy	Precision	Recall	F1	ROC AUC
Naive Bayes	0.730393	0.805912	0.788995	0.797339	0.777116
KNN	0.813811	0.863211	0.859238	0.861215	0.789870
Neural Network	0.882126	0.914202	0.857168	0.923942	NaN
Decision Trees	0.872695	0.893378	0.920671	0.906761	0.930599
Random Forest	0.881930	0.890051	0.940586	0.914617	0.939940
XGBoost	0.899021	0.914134	0.937922	0.925870	0.957921
SVM	0.859435	0.878419	0.917992	0.897765	0.914422
Extra Trees	0.900289	0.910624	0.944415	0.927209	0.955094

Stacking the models

In [11]:
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

extraTrees_classifier = ExtraTreesClassifier(criterion='entropy', n_estimators=200, max_
xgb_classifier = XGBClassifier(max_depth=11, n_estimators=100)

# stack xgb and extra trees
stacked_classifier = StackingClassifier(estimators=[('xgb', xgb_classifier), ('extra', e
stacked_classifier.fit(x_train_balanced, y_train_balanced)

stacked_scores = cross_validate(stacked_classifier, X_standard, y, cv=kf, scoring=scorin
```

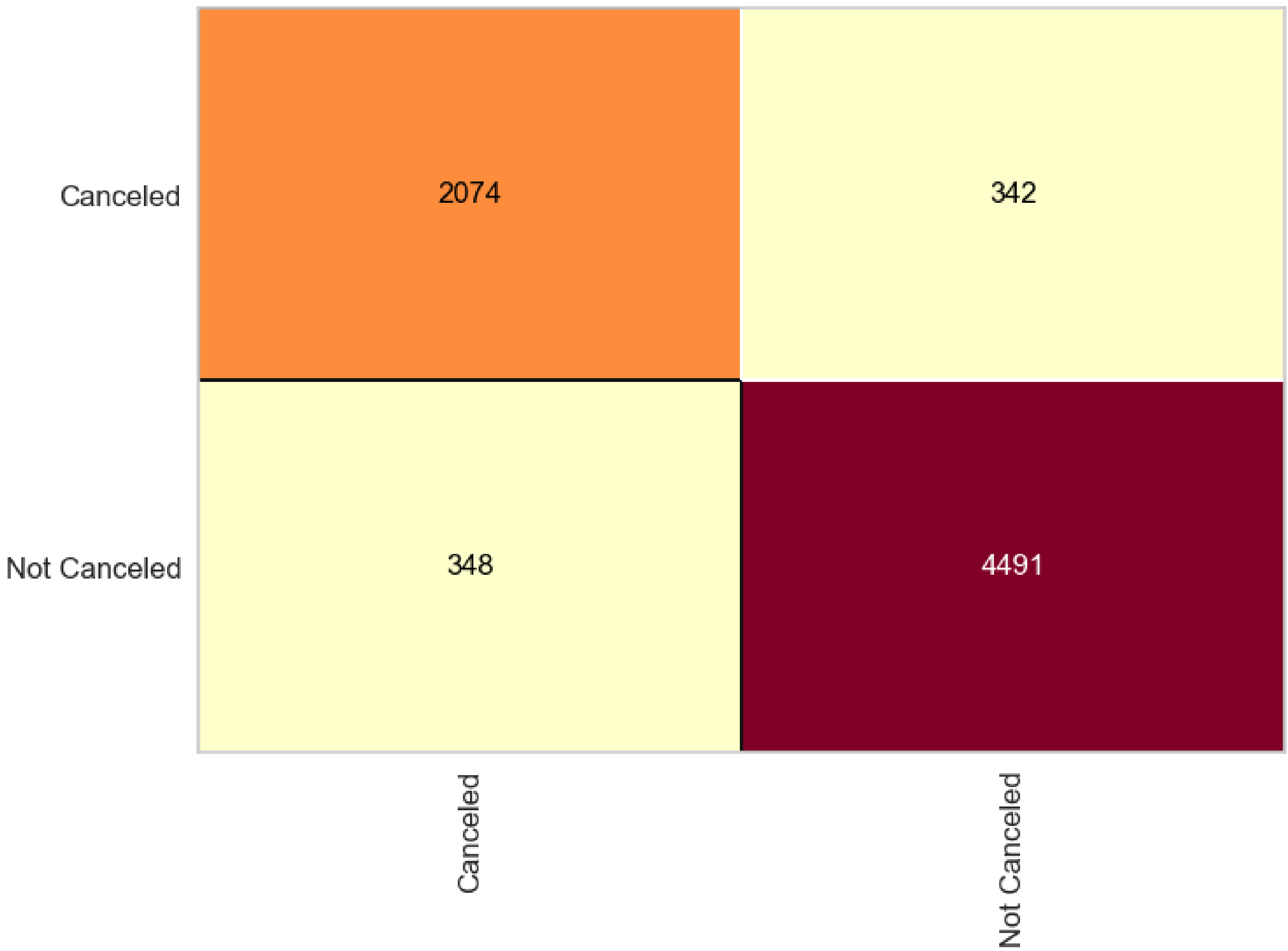
In [12]:
```
stacked_scores
```

Out[12]:
```
{'fit_time': array([177.66511369, 177.40860176, 180.12067938, 177.75984597,
        177.46689415]),
 'score_time': array([1.71090007, 1.69927979, 0.9882555 , 1.66127062, 1.67360401]),
 'test_accuracy': array([0.90820124, 0.90640937, 0.90475534, 0.90558236, 0.90020675]),
 'test_precision': array([0.92024169, 0.91821782, 0.91441712, 0.92017649, 0.91491038]),
 'test_recall': array([0.94420335, 0.9457475 , 0.94561621, 0.94112821, 0.94085477]),
 'test_f1': array([0.93206854, 0.93177936, 0.92975501, 0.93053443, 0.92770122]),
 'test_roc_auc': array([0.96242677, 0.95680088, 0.95956292, 0.95991454, 0.95648087])}
```

In [16]:
```
cm = ConfusionMatrix(stacked_classifier, is_fitted=True, encoder={0: 'Canceled', 1: 'Not
cm.score(x_test, y_test)
```

```
d:\ProgramData\Anaconda3\envs\DataMining\lib\site-packages\yellowbrick\classifier\base.p
y:232: YellowbrickWarning: could not determine class_counts_ from previously fitted clas
sifier
  warnings.warn(
```

Out[16]:
```
0.9048931771192281
```



In [13]:
```
y_pred = stacked_classifier.predict(x_test)
print(classification_report(y_test, y_pred, target_names=['Canceled', 'Not Canceled']))
```

```
              precision    recall  f1-score   support

    Canceled       0.86      0.86      0.86      2416
Not Canceled       0.93      0.93      0.93      4839

    accuracy                           0.90      7255
   macro avg       0.89      0.89      0.89      7255
weighted avg       0.90      0.90      0.90      7255
```

Decision Boundaries

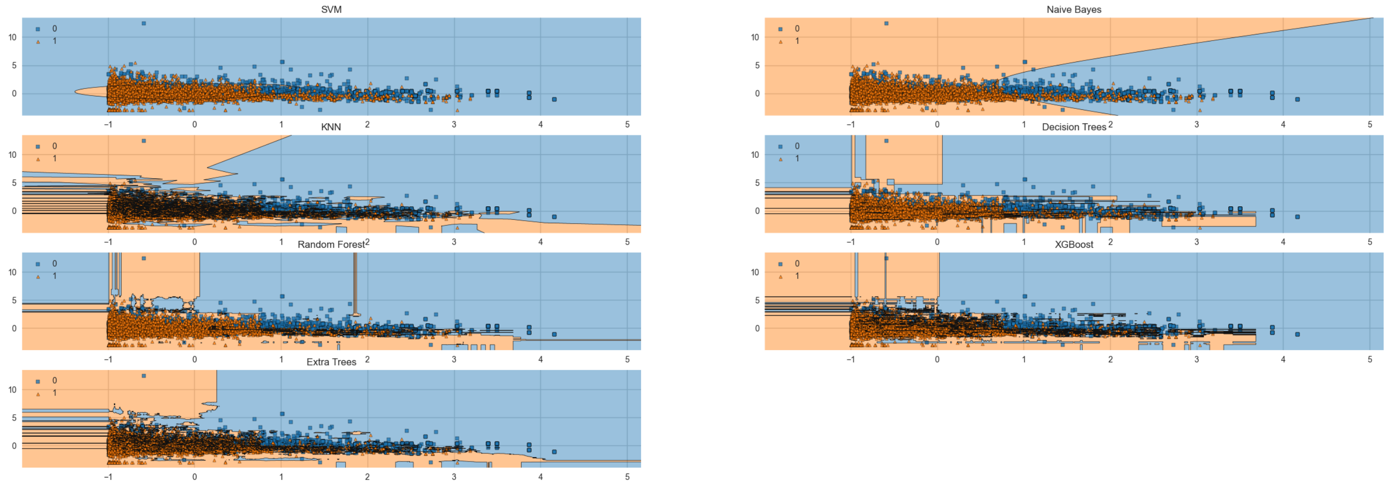
In [10]:

```
# Initializing Classifiers\
classifiers = [
    SVC(kernel='rbf', C=20, max_iter=200),
    GaussianNB(var_smoothing=1e-03),
    KNeighborsClassifier(n_neighbors=1),
    DecisionTreeClassifier(criterion='entropy', max_depth=11, min_samples_split=2),
    RandomForestClassifier(criterion='gini', max_depth=11, min_samples_split=3, n_estima
    XGBClassifier(max_depth=11, n_estimators=100),
    ExtraTreesClassifier(criterion='entropy', n_estimators=200, max_depth=25),
]
```

In [11]:

```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
gs = gridspec.GridSpec(4, 2)
grid_positions = [(i,j) for i in range(4) for j in range(2)]
fig = plt.figure(figsize=(30,10))

labels = [
    'SVM',
    'Naive Bayes',
    'KNN',
    'Decision Trees',
    'Random Forest',
    'XGBoost',
    'Extra Trees',

]
for clf, lab, grd in zip(classifiers,
                         labels,
                         grid_positions[:-1]):
    print(lab)
    clf.fit(x_train[:, [7, 15]], y_train)
    ax = plt.subplot(gs[grd[0], grd[1]])
    print("done")
    fig = plot_decision_regions(X=x_test[:, [7, 15]], y=np.array(y_test), clf=clf, legen
    plt.title(lab)

plt.show()
```

```
SVM
d:\ProgramData\Anaconda3\envs\DataMining\lib\site-packages\sklearn\svm\_base.py:299: Con
vergenceWarning: Solver terminated early (max_iter=100).  Consider pre-processing your d
ata with StandardScaler or MinMaxScaler.
  warnings.warn(
done
Naive Bayes
done
KNN
done
Decision Trees
done
Random Forest
done
XGBoost
done
Extra Trees
done
```



Observations

- We noticed that using StandardScaler on the data decreased the accuracy on Naive Bayes classifier.
- We noticed that removing arrival_year from the data had little impact on the metrics (maximum 0.2% on each metric).
- Some of the columns are unbalanced (such as arrival_year or no_of_children or repeated_guest) and this may affect the accuracy of the models, and removing them from the data decreased the accuracy of the models in some cases.