

```
# do the same thing, but use scikitlearn randomforest classifier
```

```
!pip install scikit-learn==1.3.0 --upgrade
```

```
!pip install --upgrade xgboost
```

```
➡ Requirement already satisfied: scikit-learn==1.3.0 in /usr/local/lib/python3.11/dist-packages (1.3.0)  
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.3.0)  
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.3.0)  
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.3.0)  
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.3.0)  
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)  
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.26.4)  
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.11.1)  
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.13.1)
```

```
#classify with cycle features including alignment
```

```
import pandas as pd
```

```
# import xgboost as xgb
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier as RFC
```

```
from sklearn.metrics import classification_report
```

```
import xgboost as xgb
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import roc_curve
```

```
import seaborn as sns
```

```
from matplotlib import pyplot as plt
```

```
import numpy as np
```

```
from IPython import get_ipython
```

```
from IPython.display import display
```

```
from sklearn.impute import SimpleImputer # Import SimpleImputer for imputation
```

```
import shap
```

```
shap.initjs()
```



## ✓ Set up

```
df = pd.read_csv('/content/cycle_and_HMM_features_true_bw-6-3_dataset_48days.csv')
```

```
df.head()
```



	hub_id	pat_cat_map	cycle_min	cycle_max	cycle_median	cycle_mean	cycle_range	cycle_s
0	U2CCD5D16315123	PCOS	27	42	35.0	34.896552	15	4.4024
1	U2E649816722750	PCOS	31	42	34.0	35.461538	11	4.3706
2	U2F191017106760	nonPCOS-nonBaseline	15	32	28.0	27.473684	17	3.8059
3	U2B70EC15755124	PCOS	31	47	38.0	38.000000	16	3.8729
4	U2F65CA17170226	PCOS	35	40	40.0	38.333333	5	2.8867

```
# LOOK AT LAUREN'S GITHUB FOR CODE
```

```
# try w xgboost
# try w subset of features
# explanatory tools to see which variables are important (SHAP values)
```

```
df = df.loc[df['pat_cat_map'].isin(['Baseline','PCOS'])]
```

```
df['label_01'] = df['pat_cat_map'].map({'Baseline':0, 'PCOS':1})
```



```
<ipython-input-988-1fe60784182b>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing)

```
df['label_01'] = df['pat_cat_map'].map({'Baseline':0, 'PCOS':1})
```



```
df = df.replace(-np.inf, np.nan)
```

```
df.columns
```



```
Index(['hub_id', 'pat_cat_map', 'cycle_min', 'cycle_max', 'cycle_median',
       'cycle_mean', 'cycle_range', 'cycle_std', 'num_cycles',
       'viterbi_logprob_mean', 'viterbi_logprob_min', 'viterbi_logprob_max',
       'viterbi_logprob_std', 'viterbi_logprob_median',
       'complete_logprob_mean', 'complete_logprob_min', 'complete_logprob_max',
       'complete_logprob_std', 'complete_logprob_median', 'label_01'],
      dtype='object')
```

```
HMM_features = [ 'viterbi_logprob_mean',
                  'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
                  'viterbi_logprob_median', 'complete_logprob_mean',
                  'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',
                  'complete_logprob_median']
```

```
cycle_features = ['cycle_min', 'cycle_max', 'cycle_median',
                  'cycle_mean', 'cycle_range', 'cycle_std']
```

```
target = 'label_01'
```

## ✓ All features

```
print('Performance with all features')
```

```
X_train_all, X_test_all, y_train_all, y_test_all = train_test_split(df[HMM_features+cycle_features], df[target],
                                                                    shuffle=True, random_state=51)
```

➡ Performance with all features

```
clf = xgb.XGBClassifier(random_state=51)
clf.fit(X_train_all, y_train_all)
y_pred_all = clf.predict(X_test_all)
y_score_all = clf.predict_proba(X_test_all)
print(confusion_matrix(y_test_all, y_pred_all, normalize='true'))
```

➡ 

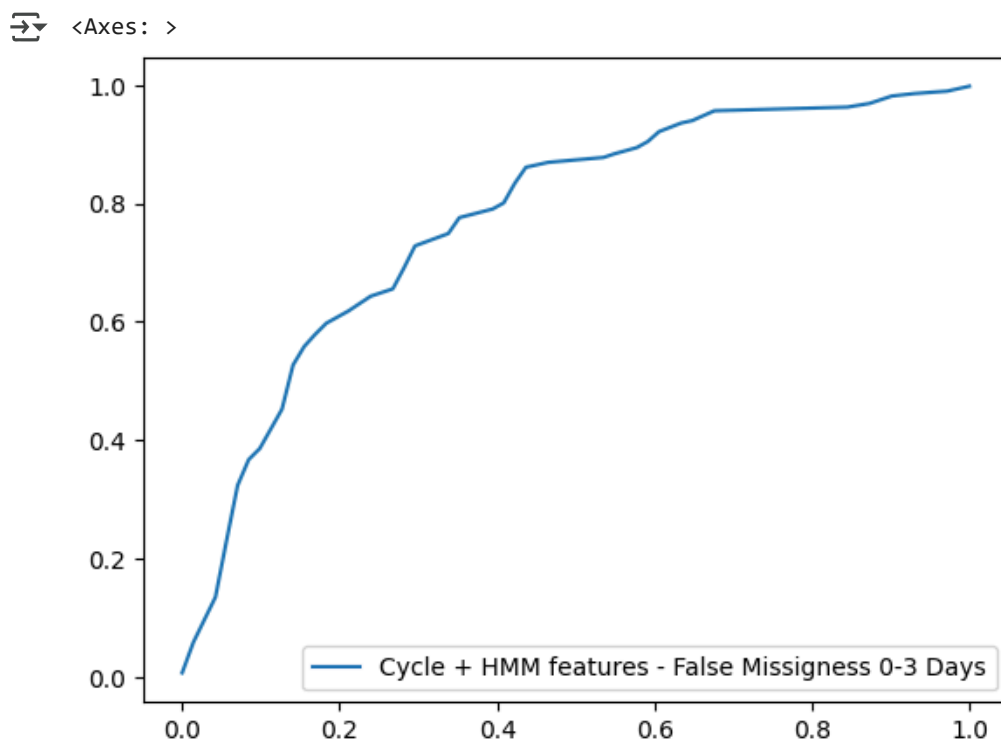
```
[[0.54929577 0.45070423]
 [0.13692946 0.86307054]]
```

```
print(classification_report(y_pred_all, y_test_all))
```

➡

	precision	recall	f1-score	support
0	0.55	0.54	0.55	72
1	0.86	0.87	0.86	240
accuracy			0.79	312
macro avg	0.71	0.70	0.71	312
weighted avg	0.79	0.79	0.79	312

```
fpr_full, tpr_full, thresholds_full = roc_curve(y_test_all, y_score_all[:,1])#, pos_label='PCOS')
sns.lineplot(x=fpr_full, y=tpr_full, label='Cycle + HMM features - False Missigness 0-3 Days', errorbar=None)
plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/xgb_full_features.pdf')
```



```
#overall accuracy:
print((y_pred_all==y_test_all).sum()/len(y_pred_all))
```

>>> 0.7916666666666666

## ✓ Cycle features only

```
#PERFORMANCE WITH CYCLE FEATURES ONLY
print('Performance with cycle features only')
```

```
X_train_cycle, X_test_cycle, y_train_cycle, y_test_cycle = train_test_split(df[cycle_features], df[target],
                                                                              shuffle=True, random_state=51)
```

>>> Performance with cycle features only

```
clf = xgb.XGBClassifier(random_state=51)
clf.fit(X_train_cycle, y_train_cycle)
y_pred_cycle = clf.predict(X_test_cycle)
y_score_cycle = clf.predict_proba(X_test_cycle)
print(confusion_matrix(y_test_cycle, y_pred_cycle, normalize='true'))
```

>>> [[0.38028169 0.61971831]
 [0.14937759 0.85062241]]

```
print(classification_report(y_pred_cycle, y_test_cycle))
```

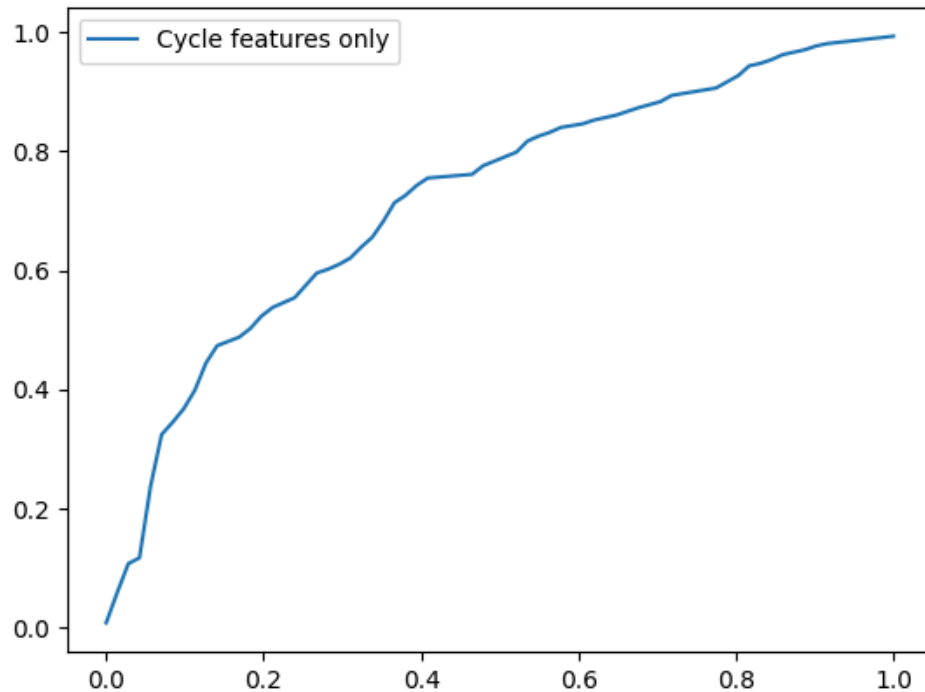
>>>

	precision	recall	f1-score	support
0	0.38	0.43	0.40	63
1	0.85	0.82	0.84	249

accuracy			0.74	312
macro avg	0.62	0.63	0.62	312
weighted avg	0.76	0.74	0.75	312

```
fpr_cycle, tpr_cycle, thresholds_cycle = roc_curve(y_test_cycle, y_score_cycle[:,1])#, pos_label='PCOS')
sns.lineplot(x=fpr_cycle, y=tpr_cycle, label='Cycle features only', errorbar=None)
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/xgb_cycle_features_only.pdf')
```

↔ <Axes: >



```
#overall accuracy:
print((y_pred_cycle==y_test_cycle).sum()/len(y_pred_cycle))
```

↔ 0.7435897435897436

## ✓ HMM Features only

```
#PERFORMANCE WITH HMM FEATURES ONLY
```

```
print('Performance with HMM features only')
```

```
X_train_hmm, X_test_hmm, y_train_hmm, y_test_hmm = train_test_split(df[HMM_features], df[target],
                                                                    shuffle=True, random_state=51)
```

↔ Performance with HMM features only

```
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_hmm = imputer.fit_transform(X_train_hmm)
X_test_hmm = imputer.transform(X_test_hmm)
```

```

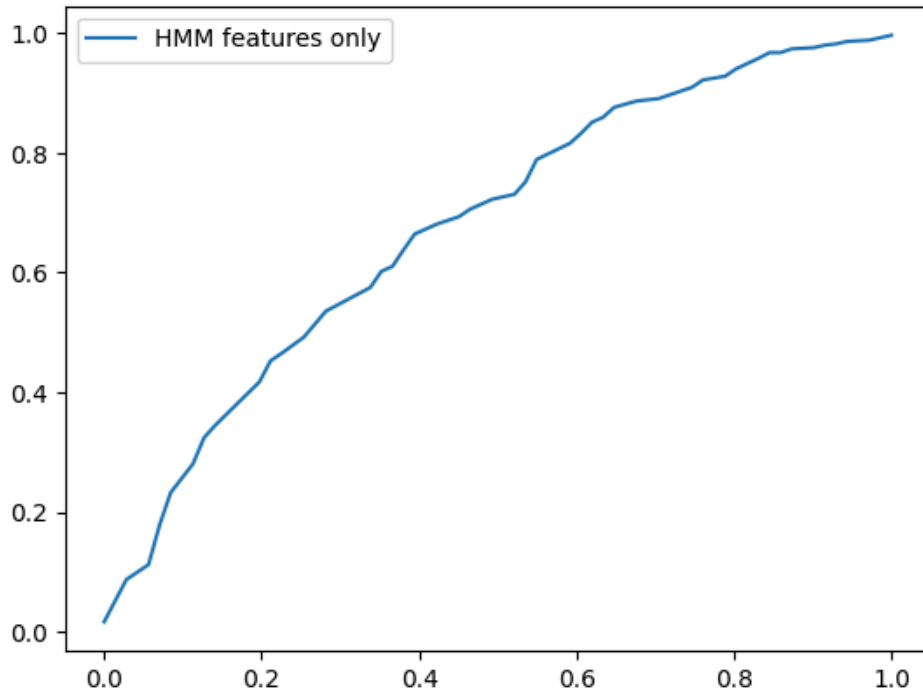
clf = RFC(random_state=101)
clf.fit(X_train_hmm, y_train_hmm)
y_pred_hmm = clf.predict(X_test_hmm)
y_score_hmm = clf.predict_proba(X_test_hmm)
print(confusion_matrix(y_test_hmm, y_pred_hmm, normalize='true'))
fpr_hmm, tpr_hmm, thresholds_hmm = roc_curve(y_test_hmm, y_score_hmm[:,1])#, pos_label='PCOS')
sns.lineplot(x=fpr_hmm, y=tpr_hmm, label='HMM features only', errorbar=None)
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/xgb_hmm_features_only.pdf')

```

```

[[0.23943662 0.76056338]
 [0.07883817 0.92116183]]
<Axes: >

```



```

print(classification_report(y_pred_cycle, y_test_cycle))

```

```

precision    recall  f1-score   support

      0       0.38      0.43      0.40        63
      1       0.85      0.82      0.84       249

 accuracy          0.74        312
  macro avg       0.62      0.63      0.62        312
 weighted avg     0.76      0.74      0.75        312

```

```

#overall accuracy:
print((y_pred_cycle==y_test_cycle).sum()/len(y_pred_cycle))
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

```

```

0.7435897435897436

```

```

#make kdeplots of all features
for feature in HMM_features+cycle_features:
    sns.kdeplot(data=df, x=feature, hue='pat_cat_map', common_norm=False)

```

```
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/xgb_kdeplots_feature_dis
plt.clf()
```

↩ <Figure size 640x480 with 0 Axes>

## ✓ ROC Curves

```
# put 3 ROC curves on one axis (cycle, hmm, all)
```

```
# # Create subplots
```

```
# fig, axes = plt.subplots(1, 3, figsize=(15, 5)) # 1 row, 3 columns
```

```
# Plot Cycle + HMM features
```

```
sns.lineplot(x=fpr_full, y=tpr_full, label='Cycle + HMM features', errorbar=None)
```

```
# axes[0].set_title("Cycle + HMM ROC Curve")
```

```
# Plot Cycle features only
```

```
sns.lineplot(x=fpr_cycle, y=tpr_cycle, label='Cycle features only', errorbar=None)
```

```
# axes[1].set_title("Cycle Only ROC Curve")
```

```
# Plot HMM features only
```

```
sns.lineplot(x=fpr_hmm, y=tpr_hmm, label='HMM features only', errorbar=None)
```

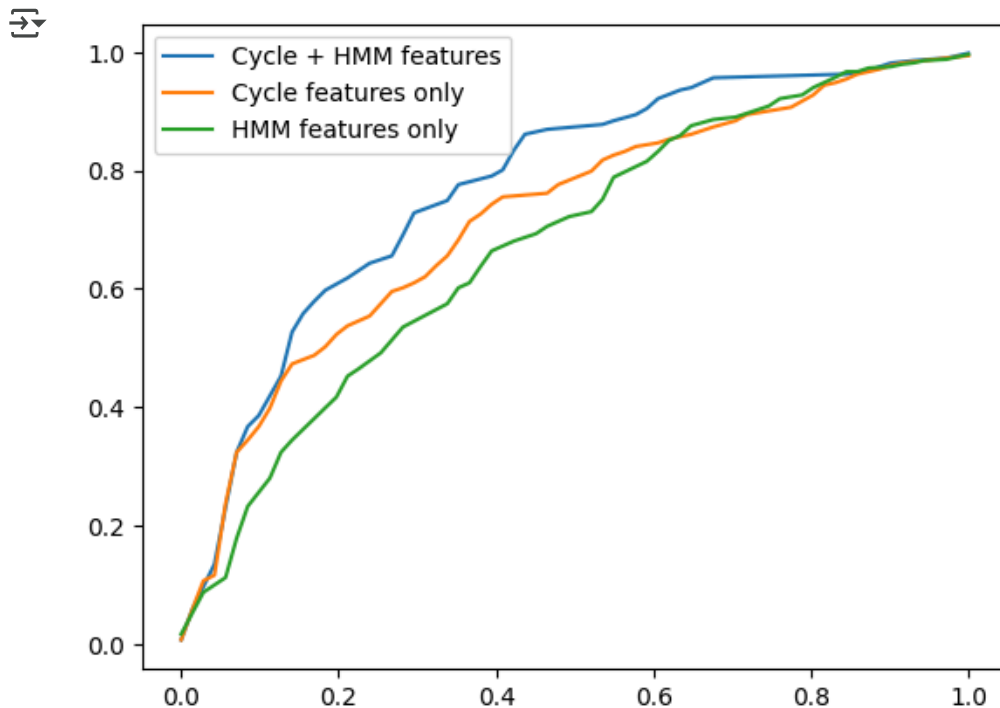
```
# axes[2].set_title("HMM Only ROC Curve")
```

```
# Adjust layout
```

```
# plt.tight_layout()
```

```
plt.show()
```

```
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/xgb_roc_curves.pdf')
```



use HMM features and take one out to see if any features are important  
(leave one out version)

```
HMM_features = ['viterbi_logprob_mean',  
                'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',  
                'viterbi_logprob_median', 'complete_logprob_mean',  
                'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',  
                'complete_logprob_median']
```

✓ without viterbi\_logprob\_mean

```
HMM_features = [  
    'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',  
    'viterbi_logprob_median', 'complete_logprob_mean',  
    'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',  
    'complete_logprob_median']
```

```
print('Performance with HMM features _without_viterbi_logprob_mean ')
```

```
X_train_without_viterbi_logprob_mean, X_test_without_viterbi_logprob_mean, y_train_without_viterbi_logprob_mean, y_test_without_viterbi_logprob_mean = train_test_split(X_train, X_test, y_train, y_test, shuffle=True, random_state=51)
```



Performance with HMM features \_without\_viterbi\_logprob\_mean

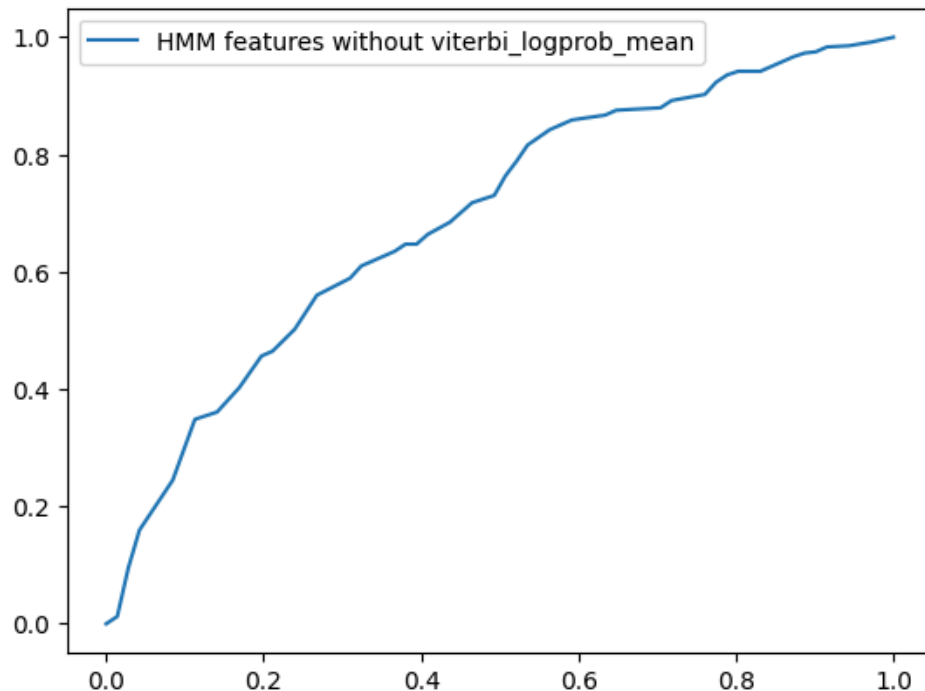
```
# Impute missing values using SimpleImputer  
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed  
X_train_without_viterbi_logprob_mean = imputer.fit_transform(X_train_without_viterbi_logprob_mean)  
X_test_without_viterbi_logprob_mean = imputer.transform(X_test_without_viterbi_logprob_mean)  
  
clf = RFC(random_state=101)  
clf.fit(X_train_without_viterbi_logprob_mean, y_train_without_viterbi_logprob_mean)  
y_pred_without_viterbi_logprob_mean = clf.predict(X_test_without_viterbi_logprob_mean)  
y_score_without_viterbi_logprob_mean = clf.predict_proba(X_test_without_viterbi_logprob_mean)  
print(confusion_matrix(y_test_without_viterbi_logprob_mean, y_pred_without_viterbi_logprob_mean, normalize=True))  
fpr_without_viterbi_logprob_mean, tpr_without_viterbi_logprob_mean, thresholds_without_viterbi_logprob_mean = roc_curve(y_test_without_viterbi_logprob_mean, y_score_without_viterbi_logprob_mean)  
sns.lineplot(x=fpr_without_viterbi_logprob_mean, y=tpr_without_viterbi_logprob_mean, label='HMM features without viterbi_logprob_mean')  
plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_wi')
```



```

→ [[0.22535211 0.77464789]
    [0.07053942 0.92946058]]
<Axes: >

```



```
print(classification_report(y_pred_without_viterbi_logprob_mean, y_test_without_viterbi_logprob_mean))
```

```

→
              precision    recall  f1-score   support

         0       0.23        0.48        0.31         33
         1       0.93        0.80        0.86        279

 accuracy          0.77         312
 macro avg         0.58         0.64         0.58         312
 weighted avg      0.85         0.77         0.80         312

```

```
#overall accuracy:
```

```

print((y_pred_without_viterbi_logprob_mean==y_test_without_viterbi_logprob_mean).sum()/len(y_pred_without_
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

```

```
→ 0.7692307692307693
```

## ✓ without viterbi\_logprob\_min

```

HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_max', 'viterbi_logprob_std',
                'viterbi_logprob_median', 'complete_logprob_mean',
                'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',
                'complete_logprob_median']

```

```
print('Performance with HMM features _without_viterbi_logprob_min ')
```

```
X_train_without_viterbi_logprob_min, X_test_without_viterbi_logprob_min, y_train_without_viterbi_logprob_min, y_test_without_viterbi_logprob_min)
shuffle=True, random_state=51)
```

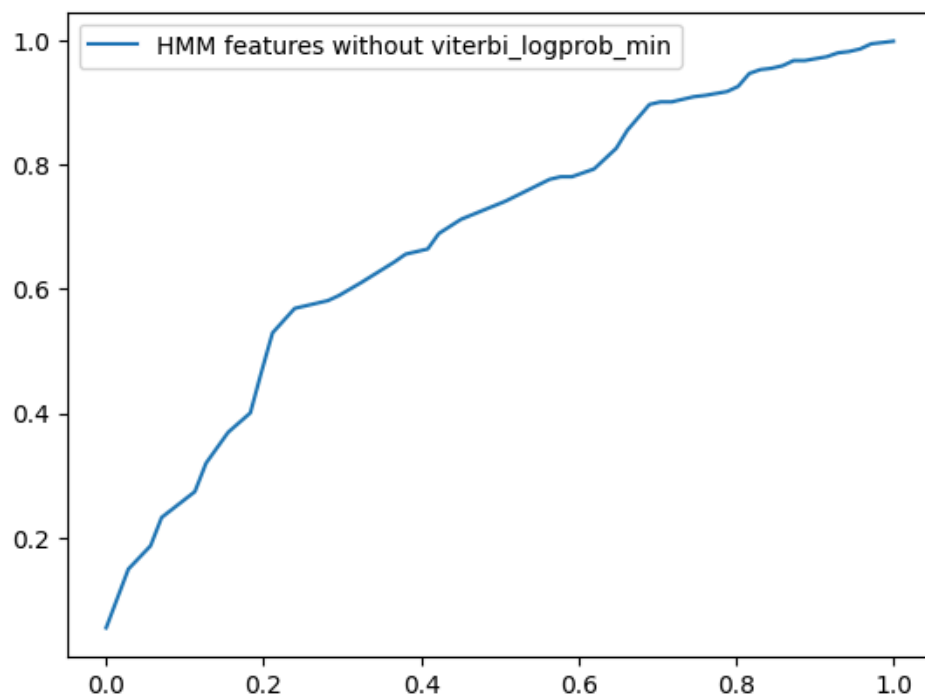
➡ Performance with HMM features \_without\_viterbi\_logprob\_min

```
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_min = imputer.fit_transform(X_train_without_viterbi_logprob_min)
X_test_without_viterbi_logprob_min = imputer.transform(X_test_without_viterbi_logprob_min)

clf = RFC(random_state=101)
clf.fit(X_train_without_viterbi_logprob_min, y_train_without_viterbi_logprob_min)
y_pred_without_viterbi_logprob_min = clf.predict(X_test_without_viterbi_logprob_min)
y_score_without_viterbi_logprob_min = clf.predict_proba(X_test_without_viterbi_logprob_min)
print(confusion_matrix(y_test_without_viterbi_logprob_min, y_pred_without_viterbi_logprob_min, normalize='true'))
fpr_without_viterbi_logprob_min, tpr_without_viterbi_logprob_min, thresholds_without_viterbi_logprob_min = roc_curve(y_test_without_viterbi_logprob_min, y_score_without_viterbi_logprob_min)
sns.lineplot(x=fpr_without_viterbi_logprob_min, y=tpr_without_viterbi_logprob_min, label='HMM features without viterbi_logprob_min')
plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w')
```

➡ 

```
[[0.23943662 0.76056338]
 [0.09128631 0.90871369]]
<Axes: >
```



```
print(classification_report(y_pred_without_viterbi_logprob_min, y_test_without_viterbi_logprob_min))
```

➡

	precision	recall	f1-score	support
0	0.24	0.44	0.31	39
1	0.91	0.80	0.85	273
accuracy			0.76	312
macro avg	0.57	0.62	0.58	312
weighted avg	0.83	0.76	0.78	312

```
#overall accuracy:
print((y_pred_without_viterbi_logprob_min==y_test_without_viterbi_logprob_min).sum())/len(y_pred_without_viterbi_logprob_min)
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

0.7564102564102564
```

## ✓ without viterbi\_logprob\_max

```
HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_std',
                'viterbi_logprob_median', 'complete_logprob_mean',
                'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',
                'complete_logprob_median']

print('Performance with HMM features _without_viterbi_logprob_max ')

X_train_without_viterbi_logprob_max, X_test_without_viterbi_logprob_max, y_train_without_viterbi_logprob_max = train_test_split(X_train, y_train, test_size=0.2,
                                                                    shuffle=True, random_state=51)

Performance with HMM features _without_viterbi_logprob_max

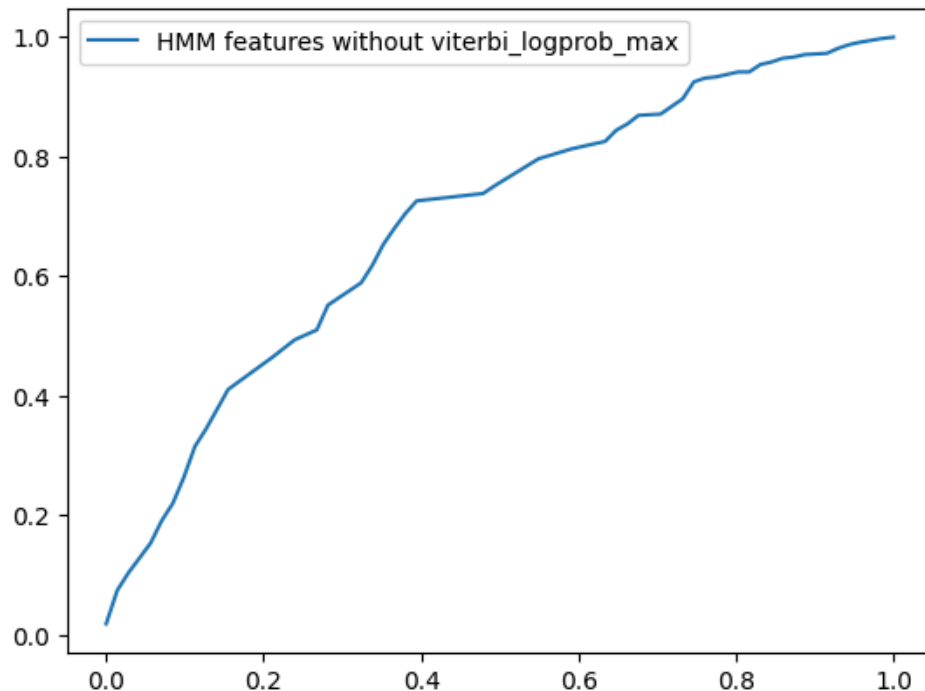
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_max = imputer.fit_transform(X_train_without_viterbi_logprob_max)
X_test_without_viterbi_logprob_max = imputer.transform(X_test_without_viterbi_logprob_max)

clf = RFC(random_state=101)
clf.fit(X_train_without_viterbi_logprob_max, y_train_without_viterbi_logprob_max)
y_pred_without_viterbi_logprob_max = clf.predict(X_test_without_viterbi_logprob_max)
y_score_without_viterbi_logprob_max = clf.predict_proba(X_test_without_viterbi_logprob_max)
print(confusion_matrix(y_test_without_viterbi_logprob_max, y_pred_without_viterbi_logprob_max, normalize='true'))
fpr_without_viterbi_logprob_max, tpr_without_viterbi_logprob_max, thresholds_without_viterbi_logprob_max = roc_curve(y_test_without_viterbi_logprob_max, y_score_without_viterbi_logprob_max)
sns.lineplot(x=fpr_without_viterbi_logprob_max, y=tpr_without_viterbi_logprob_max, label='HMM features without viterbi_logprob_max')
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_without_viterbi_logprob_max.png')
```

```

→ [[0.26760563 0.73239437]
    [0.09543568 0.90456432]]
<Axes: >

```



```
print(classification_report(y_pred_without_viterbi_logprob_max, y_test_without_viterbi_logprob_max))
```

```

→
              precision    recall  f1-score   support

         0       0.27       0.45       0.34         42
         1       0.90       0.81       0.85        270

   accuracy                   0.76         312
  macro avg       0.59       0.63       0.59         312
 weighted avg       0.82       0.76       0.78         312

```

```
#overall accuracy:
```

```

print((y_pred_without_viterbi_logprob_max==y_test_without_viterbi_logprob_max).sum()/len(y_pred_without_viterbi_logprob_max))
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

```

```
→ 0.7596153846153846
```

## ✓ without viterbi\_logprob\_std

```

HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_max',
                'viterbi_logprob_median', 'complete_logprob_mean',
                'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',
                'complete_logprob_median']

```

```
print('Performance with HMM features _without_viterbi_logprob_std ')
```

```
X_train_without_viterbi_logprob_std, X_test_without_viterbi_logprob_std, y_train_without_viterbi_logprob_std,
```

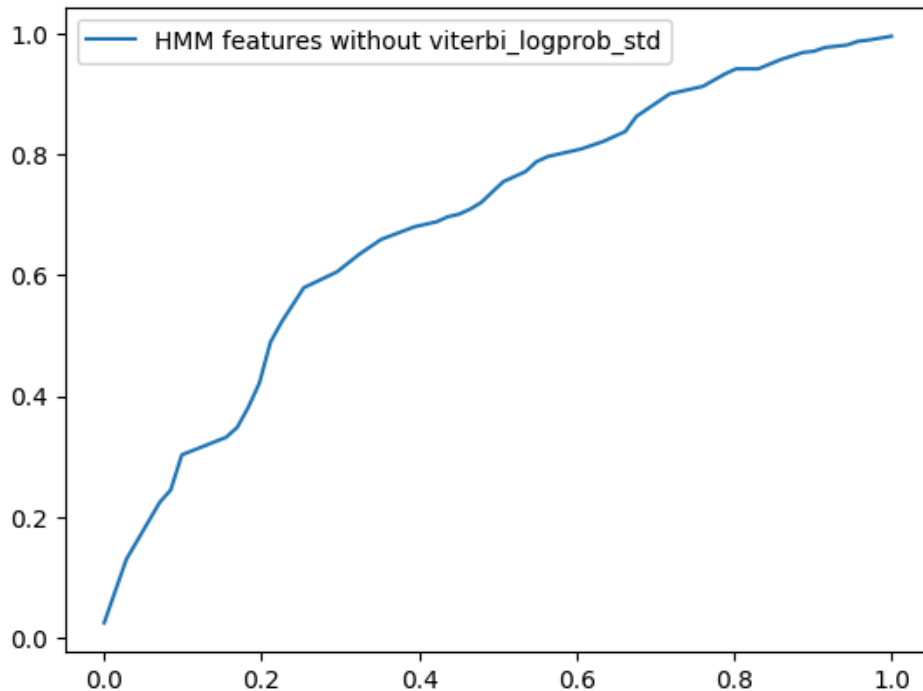
```
shuffle=True, random_state=51)
```

```
➡ Performance with HMM features _without_viterbi_logprob_std
```

```
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_std = imputer.fit_transform(X_train_without_viterbi_logprob_std)
X_test_without_viterbi_logprob_std = imputer.transform(X_test_without_viterbi_logprob_std)

clf = RFC(random_state=101)
clf.fit(X_train_without_viterbi_logprob_std, y_train_without_viterbi_logprob_std)
y_pred_without_viterbi_logprob_std = clf.predict(X_test_without_viterbi_logprob_std)
y_score_without_viterbi_logprob_std = clf.predict_proba(X_test_without_viterbi_logprob_std)
print(confusion_matrix(y_test_without_viterbi_logprob_std, y_pred_without_viterbi_logprob_std, normalize='
fpr_without_viterbi_logprob_std, tpr_without_viterbi_logprob_std, thresholds_without_viterbi_logprob_std =
sns.lineplot(x=fpr_without_viterbi_logprob_std, y=tpr_without_viterbi_logprob_std, label='HMM features wit
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```
➡ [[0.26760563 0.73239437]
    [0.09543568 0.90456432]]
<Axes: >
```



```
print(classification_report(y_pred_without_viterbi_logprob_std, y_test_without_viterbi_logprob_std))
```

```
➡
```

	precision	recall	f1-score	support
0	0.27	0.45	0.34	42
1	0.90	0.81	0.85	270
accuracy			0.76	312
macro avg	0.59	0.63	0.59	312
weighted avg	0.82	0.76	0.78	312

```
#overall accuracy:
```

```
print((y_pred_without_viterbi_logprob_std==y_test_without_viterbi_logprob_std).sum()/len(y_pred_without_v
```

```
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)
```

↔ 0.7596153846153846

## ✓ without viterbi\_logprob\_median

```
HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
                'complete_logprob_mean',
                'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',
                'complete_logprob_median']
```

```
print('Performance with HMM features _without_viterbi_logprob_median ')
```

```
X_train_without_viterbi_logprob_median, X_test_without_viterbi_logprob_median, y_train_without_viterbi_log
                                shuffle=True, random_state=51)
```

↔ Performance with HMM features \_without\_viterbi\_logprob\_median

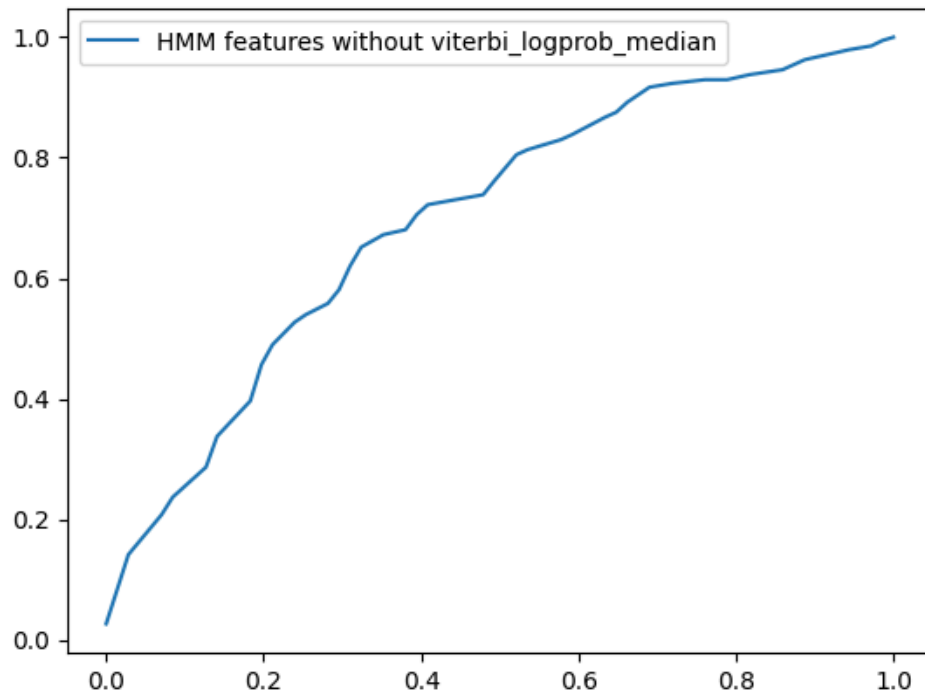
```
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_median = imputer.fit_transform(X_train_without_viterbi_logprob_median)
X_test_without_viterbi_logprob_median = imputer.transform(X_test_without_viterbi_logprob_median)

clf = RFC(random_state=101)
clf.fit(X_train_without_viterbi_logprob_median, y_train_without_viterbi_logprob_median)
y_pred_without_viterbi_logprob_median = clf.predict(X_test_without_viterbi_logprob_median)
y_score_without_viterbi_logprob_median = clf.predict_proba(X_test_without_viterbi_logprob_median)
print(confusion_matrix(y_test_without_viterbi_logprob_median, y_pred_without_viterbi_logprob_median, norma
fpr_without_viterbi_logprob_median, tpr_without_viterbi_logprob_median, thresholds_without_viterbi_logprob
sns.lineplot(x=fpr_without_viterbi_logprob_median, y=tpr_without_viterbi_logprob_median, label='HMM featur
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```

→ [[0.28169014 0.71830986]
   [0.07883817 0.92116183]]
<Axes: >

```



```
print(classification_report(y_pred_without_viterbi_logprob_median, y_test_without_viterbi_logprob_median))
```

```

→
              precision    recall  f1-score   support

     0       0.28         0.51         0.36         39
     1       0.92         0.81         0.86        273

 accuracy          0.78         312
 macro avg         0.60         0.66         0.61         312
 weighted avg      0.84         0.78         0.80         312

```

```

#overall accuracy:
print((y_pred_without_viterbi_logprob_median==y_test_without_viterbi_logprob_median).sum()/len(y_pred_with
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

```

```
→ 0.7756410256410257
```

## ✓ without complete\_logprob\_mean

```

HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
                'viterbi_logprob_median',
                'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std',
                'complete_logprob_median']

```

```
print('Performance with HMM features _without_complete_logprob_mean ')
```

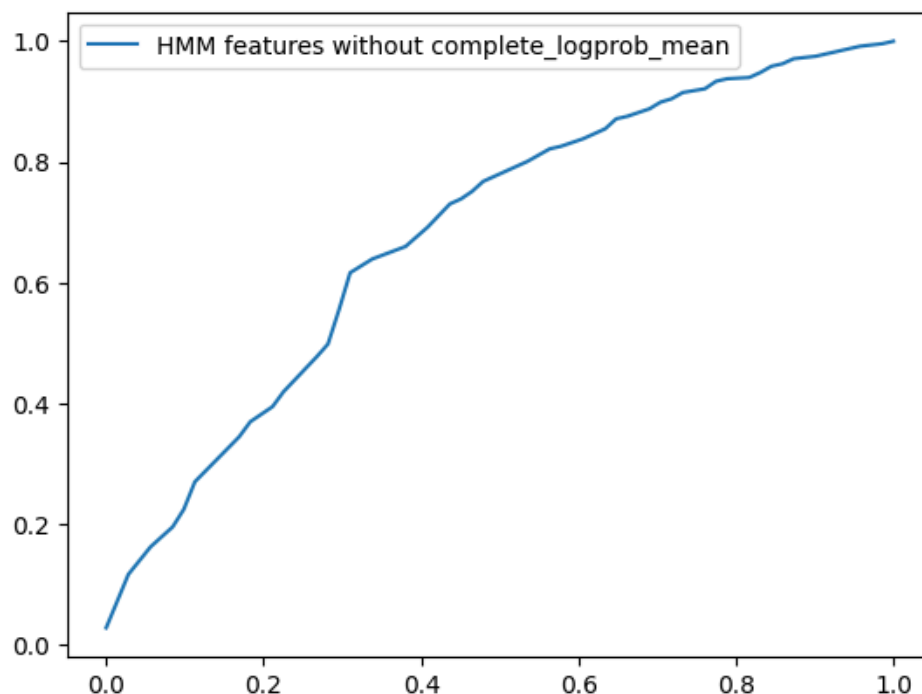
```
X_train_without_complete_logprob_mean, X_test_without_complete_logprob_mean, y_train_without_complete_logprob_mean, y_test_without_complete_logprob_mean)
shuffle=True, random_state=51)
```

➡ Performance with HMM features \_without\_complete\_logprob\_mean

```
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_complete_logprob_mean = imputer.fit_transform(X_train_without_complete_logprob_mean)
X_test_without_complete_logprob_mean = imputer.transform(X_test_without_complete_logprob_mean)

clf = RFC(random_state=101)
clf.fit(X_train_without_complete_logprob_mean, y_train_without_complete_logprob_mean)
y_pred_without_complete_logprob_mean = clf.predict(X_test_without_complete_logprob_mean)
y_score_without_complete_logprob_mean = clf.predict_proba(X_test_without_complete_logprob_mean)
print(confusion_matrix(y_test_without_complete_logprob_mean, y_pred_without_complete_logprob_mean, normalize=True))
fpr_without_complete_logprob_mean, tpr_without_complete_logprob_mean, thresholds_without_complete_logprob_mean = roc_curve(y_test_without_complete_logprob_mean, y_score_without_complete_logprob_mean)
sns.lineplot(x=fpr_without_complete_logprob_mean, y=tpr_without_complete_logprob_mean, label='HMM features without complete_logprob_mean')
plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w')
```

➡ [[0.26760563 0.73239437]  
[0.08298755 0.91701245]]  
<Axes: >



```
print(classification_report(y_pred_without_complete_logprob_mean, y_test_without_complete_logprob_mean))
```

➡

	precision	recall	f1-score	support
0	0.27	0.49	0.35	39
1	0.92	0.81	0.86	273
accuracy			0.77	312
macro avg	0.59	0.65	0.60	312
weighted avg	0.84	0.77	0.80	312



```
#overall accuracy:
print((y_pred_without_complete_logprob_mean==y_test_without_complete_logprob_mean).sum())/len(y_pred_without
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

➡ 0.7692307692307693
```

## ✓ without complete\_logprob\_min

```
HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
                'viterbi_logprob_median', 'complete_logprob_mean',
                'complete_logprob_max', 'complete_logprob_std',
                'complete_logprob_median']

print('Performance with HMM features _without_complete_logprob_min ')

X_train_without_complete_logprob_min, X_test_without_complete_logprob_min, y_train_without_complete_logpro
shuffle=True, random_state=51)

➡ Performance with HMM features _without_complete_logprob_min

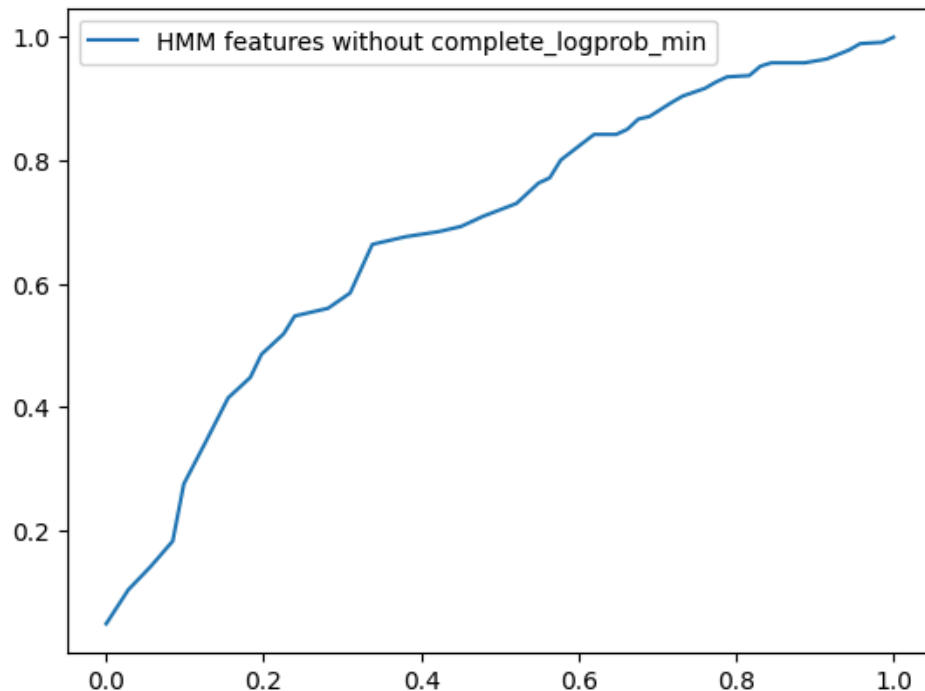
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_complete_logprob_min = imputer.fit_transform(X_train_without_complete_logprob_min)
X_test_without_complete_logprob_min = imputer.transform(X_test_without_complete_logprob_min)

clf = RFC(random_state=101)
clf.fit(X_train_without_complete_logprob_min, y_train_without_complete_logprob_min)
y_pred_without_complete_logprob_min = clf.predict(X_test_without_complete_logprob_min)
y_score_without_complete_logprob_min = clf.predict_proba(X_test_without_complete_logprob_min)
print(confusion_matrix(y_test_without_complete_logprob_min, y_pred_without_complete_logprob_min, normalize
fpr_without_complete_logprob_min, tpr_without_complete_logprob_min, thresholds_without_complete_logprob_mi
sns.lineplot(x=fpr_without_complete_logprob_min, y=tpr_without_complete_logprob_min, label='HMM features w
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```

→ [[0.22535211 0.77464789]
   [0.0746888 0.9253112 ]]
<Axes: >

```



```

print(classification_report(y_pred_without_complete_logprob_min, y_test_without_complete_logprob_min))

```

```

→
              precision    recall  f1-score   support

         0       0.23       0.47       0.30         34
         1       0.93       0.80       0.86        278

   accuracy                   0.77         312
  macro avg       0.58       0.64       0.58         312
 weighted avg       0.85       0.77       0.80         312

```

```

#overall accuracy:
print((y_pred_without_complete_logprob_min==y_test_without_complete_logprob_min).sum()/len(y_pred_without_
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

```

```

→ 0.7660256410256411

```

## ✓ without complete\_logprob\_max

```

HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
                'viterbi_logprob_median', 'complete_logprob_mean',
                'complete_logprob_min', 'complete_logprob_std',
                'complete_logprob_median']

```

```

print('Performance with HMM features _without_complete_logprob_max ')

```

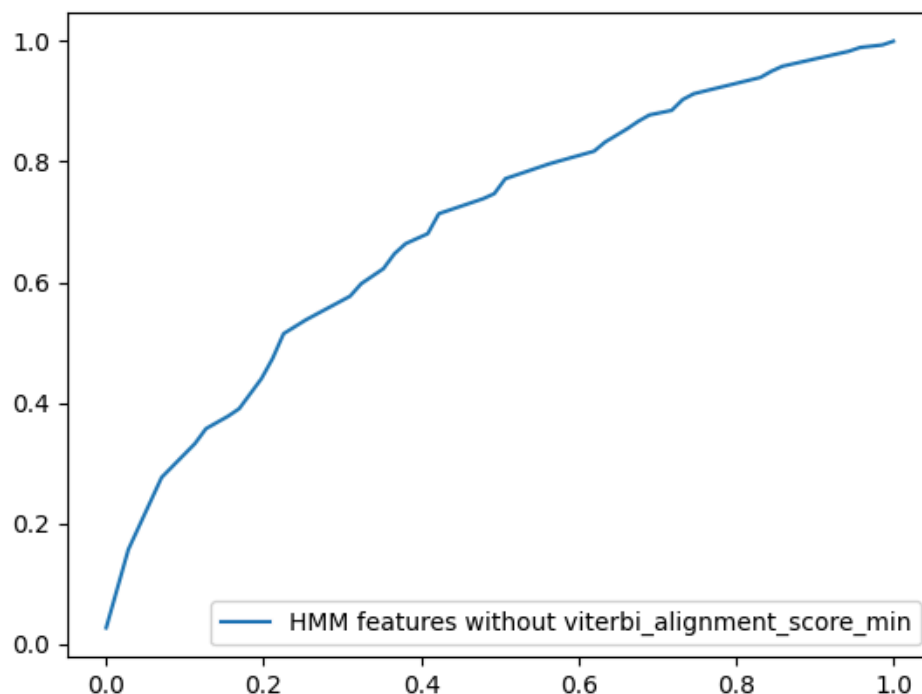
```
X_train_without_complete_logprob_max, X_test_without_complete_logprob_max, y_train_without_complete_logprob_max, y_test_without_complete_logprob_max)
shuffle=True, random_state=51)
```

➡ Performance with HMM features \_without\_complete\_logprob\_max

```
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_complete_logprob_max = imputer.fit_transform(X_train_without_complete_logprob_max)
X_test_without_complete_logprob_max = imputer.transform(X_test_without_complete_logprob_max)

clf = RFC(random_state=101)
clf.fit(X_train_without_complete_logprob_max, y_train_without_complete_logprob_max)
y_pred_without_complete_logprob_max = clf.predict(X_test_without_complete_logprob_max)
y_score_without_complete_logprob_max = clf.predict_proba(X_test_without_complete_logprob_max)
print(confusion_matrix(y_test_without_complete_logprob_max, y_pred_without_complete_logprob_max, normalize=True))
fpr_without_complete_logprob_max, tpr_without_complete_logprob_max, thresholds_without_complete_logprob_max = roc_curve(y_test_without_complete_logprob_max, y_score_without_complete_logprob_max)
sns.lineplot(x=fpr_without_complete_logprob_max, y=tpr_without_complete_logprob_max, label='HMM features w/o viterbi alignment score min')
plt.savefig('/content/drive/MyDrive/fall_research/feature_distribution_plots/viterbi_adjusted_plots/xgb_w/o_viterbi_alignment_score_min.png')
```

➡ [[0.26760563 0.73239437]  
[0.09543568 0.90456432]]  
<Axes: >



```
print(classification_report(y_pred_without_complete_logprob_max, y_test_without_complete_logprob_max))
```

➡

	precision	recall	f1-score	support
0	0.27	0.45	0.34	42
1	0.90	0.81	0.85	270
accuracy			0.76	312
macro avg	0.59	0.63	0.59	312
weighted avg	0.82	0.76	0.78	312

```
#overall accuracy:
print((y_pred_without_complete_logprob_max==y_test_without_complete_logprob_max).sum())/len(y_pred_without_
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

➡ 0.7596153846153846
```

## ✓ without complete\_logprob\_std

```
HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
                'viterbi_logprob_median', 'complete_logprob_mean',
                'complete_logprob_min', 'complete_logprob_max',
                'complete_logprob_median']

print('Performance with HMM features _without_complete_logprob_std ')

X_train_without_complete_logprob_std, X_test_without_complete_logprob_std, y_train_without_complete_logpro
shuffle=True, random_state=51)

➡ Performance with HMM features _without_complete_logprob_std

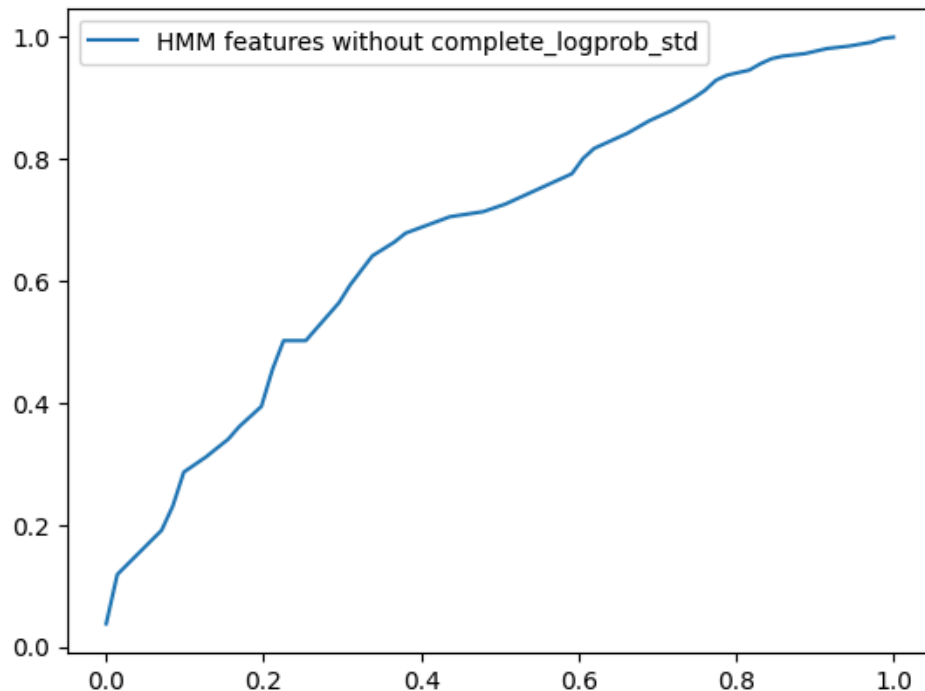
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_complete_logprob_std = imputer.fit_transform(X_train_without_complete_logprob_std)
X_test_without_complete_logprob_std = imputer.transform(X_test_without_complete_logprob_std)

clf = RFC(random_state=101)
clf.fit(X_train_without_complete_logprob_std, y_train_without_complete_logprob_std)
y_pred_without_complete_logprob_std = clf.predict(X_test_without_complete_logprob_std)
y_score_without_complete_logprob_std = clf.predict_proba(X_test_without_complete_logprob_std)
print(confusion_matrix(y_test_without_complete_logprob_std, y_pred_without_complete_logprob_std, normalize
fpr_without_complete_logprob_std, tpr_without_complete_logprob_std, thresholds_without_complete_logprob_std)
sns.lineplot(x=fpr_without_complete_logprob_std, y=tpr_without_complete_logprob_std, label='HMM features w
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```

→ [[0.23943662 0.76056338]
   [0.08298755 0.91701245]]
<Axes: >

```



```

print(classification_report(y_pred_without_complete_logprob_std, y_test_without_complete_logprob_std))

```

```

→
              precision    recall  f1-score   support

         0       0.24       0.46       0.31         37
         1       0.92       0.80       0.86        275

 accuracy              0.76         312
 macro avg              0.58         312
weighted avg              0.84         312

```

```

#overall accuracy:
print((y_pred_without_complete_logprob_std==y_test_without_complete_logprob_std).sum()/len(y_pred_without_
#fpr_algn, tpr_algn, thresholds_algn = roc_curve(y_test, -1*X_test, pos_label='PCOS')
#sns.lineplot(x=fpr_algn, y=tpr_algn, label='HMM features only', errorbar=None)

```

```

→ 0.7628205128205128

```

## ✓ without complete\_logprob\_median

```

HMM_features = ['viterbi_logprob_mean',
                'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
                'viterbi_logprob_median', 'complete_logprob_mean',
                'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std']

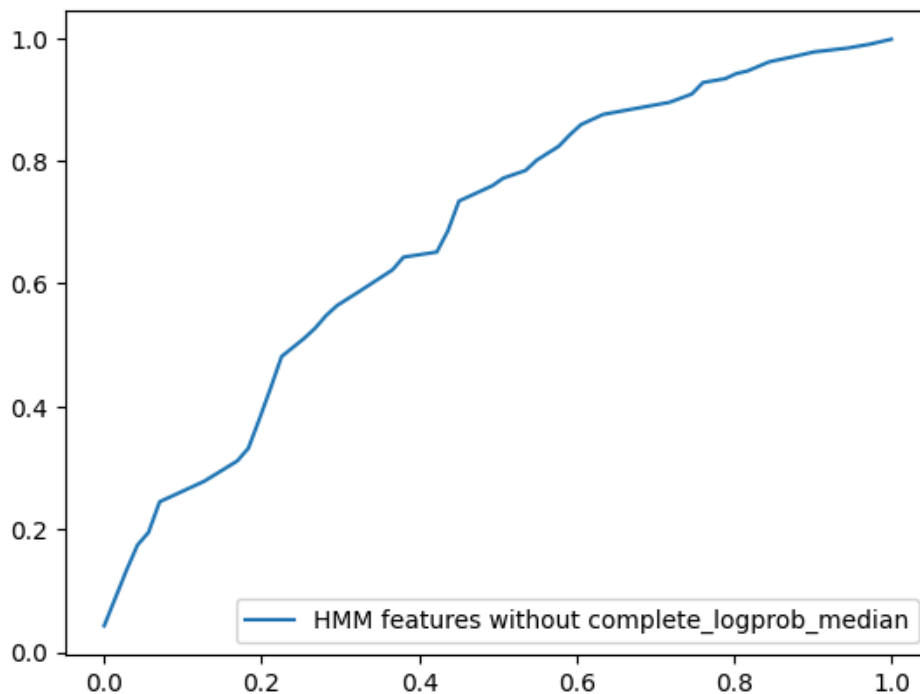
```

```
print('Performance with HMM features _without_viterbi_logprob_median ')
```

```
X_train_without_viterbi_logprob_median, X_test_without_viterbi_logprob_median, y_train_without_viterbi_logprob_median = train_test_split(X_train_without_viterbi_logprob_median, y_train_without_viterbi_logprob_median, test_size=0.2, random_state=101)
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_median = imputer.fit_transform(X_train_without_viterbi_logprob_median)
X_test_without_viterbi_logprob_median = imputer.transform(X_test_without_viterbi_logprob_median)
```

```
clf = RFC(random_state=101)
clf.fit(X_train_without_viterbi_logprob_median, y_train_without_viterbi_logprob_median)
y_pred_without_viterbi_logprob_median = clf.predict(X_test_without_viterbi_logprob_median)
y_score_without_viterbi_logprob_median = clf.predict_proba(X_test_without_viterbi_logprob_median)
print(confusion_matrix(y_test_without_viterbi_logprob_median, y_pred_without_viterbi_logprob_median, normalize=True))
fpr_without_viterbi_logprob_median, tpr_without_viterbi_logprob_median, thresholds_without_viterbi_logprob_median = roc_curve(y_test_without_viterbi_logprob_median, y_score_without_viterbi_logprob_median)
sns.lineplot(x=fpr_without_viterbi_logprob_median, y=tpr_without_viterbi_logprob_median, label='HMM features without complete_logprob_median')
plt.savefig('/content/drive/MyDrive/fall_research/feature_distribution_plots/viterbi_adjusted_plots/xgb_w')
```

```
[[0.25352113 0.74647887]
 [0.09128631 0.90871369]]
<Axes: >
```



```
print(classification_report(y_pred_without_viterbi_logprob_median, y_test_without_viterbi_logprob_median))
```

```
precision    recall  f1-score   support

0           0.25      0.45      0.32         40
1           0.91      0.81      0.85        272

accuracy          0.76         312
macro avg          0.58      0.63      0.59         312
weighted avg       0.82      0.76      0.79         312
```

```
#overall accuracy:
```

```
print((y_pred_without_viterbi_logprob_median==y_test_without_viterbi_logprob_median).sum()/len(y_pred_without_viterbi_logprob_median))
```