

```
# 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) (1.26.4)  
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.3.0) (1.13.1)  
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.3.0) (1.4.2)  
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.3.0) (3.5.0)  
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) (12.1.6)  
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.13.1)
```

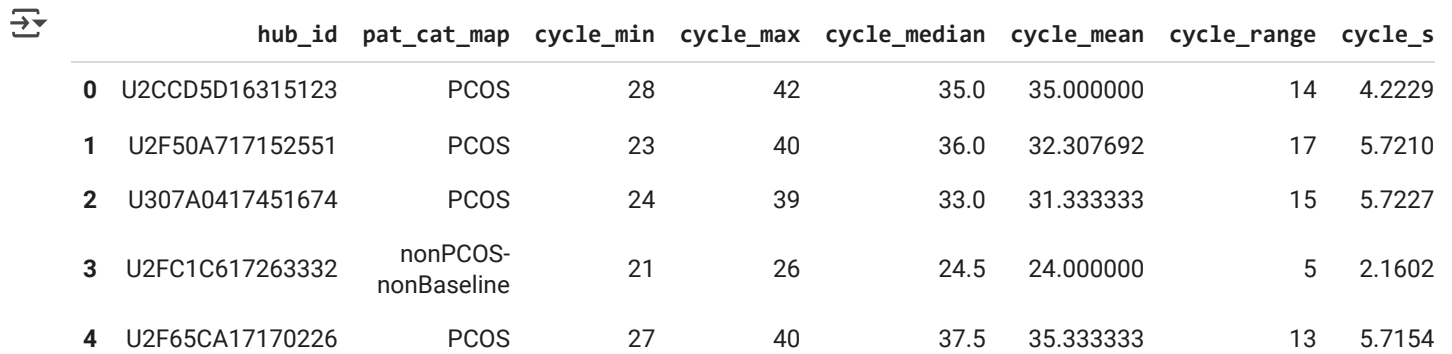
```
#classify with cycle features including alignment  
import pandas as pd  
# import xgboost as xgb  
from sklearn.model_selection import train_test_split  
from sklearn.ensemble import RandomForestClassifier as RFC  
from sklearn.metrics import classification_report  
import xgboost as xgb  
from 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_7-10_dataset_48days.csv')
```

```
df.head()
```



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

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

df.columns

```
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']
```

- ✓ 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'))
```

➡ `[[0.28985507 0.71014493]`  
`[0.07894737 0.92105263]]`

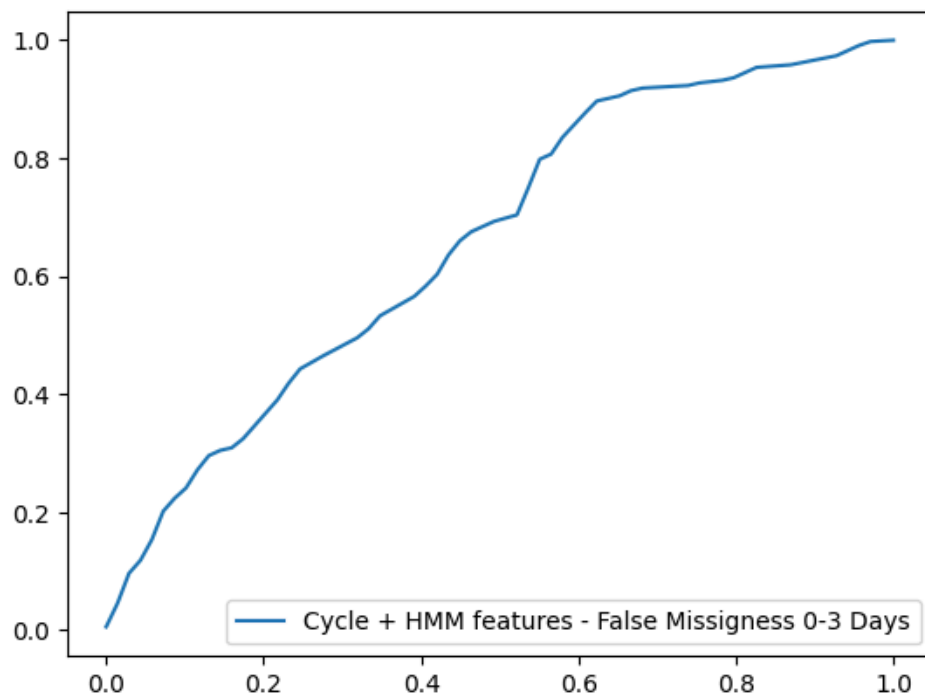
```
print(classification_report(y_pred_all, y_test_all))
```

➡

	precision	recall	f1-score	support
0	0.29	0.53	0.37	38
1	0.92	0.81	0.86	259
accuracy			0.77	297
macro avg	0.61	0.67	0.62	297
weighted avg	0.84	0.77	0.80	297

```
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')
```

➡ <Axes: >



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

➡ 0.7744107744107744

## ✓ 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'))
```

```
➡ [[0.34782609 0.65217391]
   [0.10526316 0.89473684]]
```

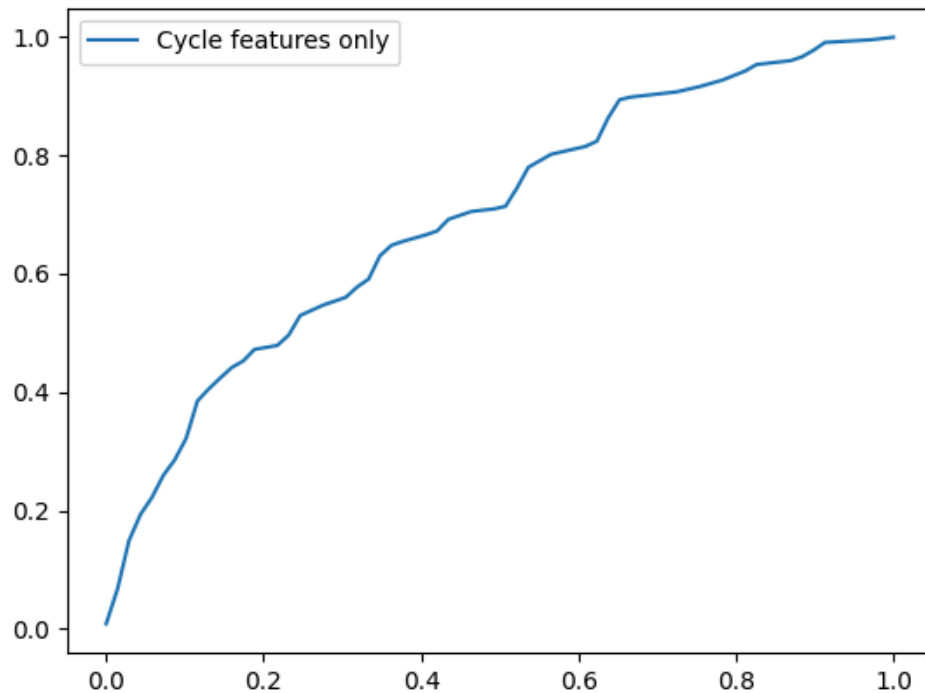
```
print(classification_report(y_pred_cycle, y_test_cycle))
```

```
➡
```

	precision	recall	f1-score	support
0	0.35	0.50	0.41	48
1	0.89	0.82	0.86	249
accuracy			0.77	297
macro avg	0.62	0.66	0.63	297
weighted avg	0.81	0.77	0.78	297

```
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))
```

↗ 0.7676767676767676

## ✓ 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)
```

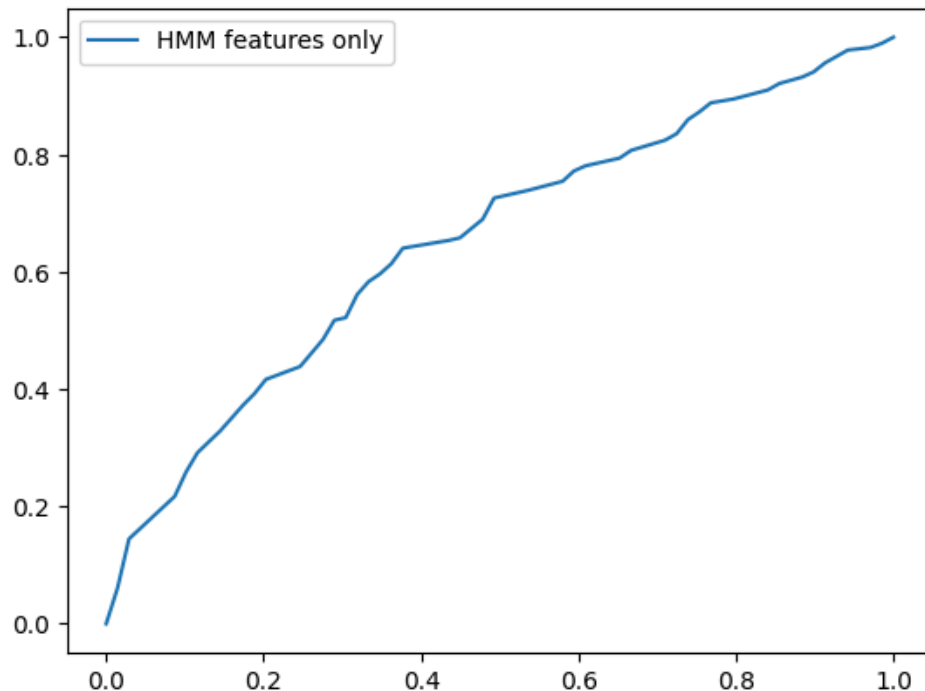
↗ Performance with HMM features only

```
# 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')
```

```

[[0.20289855 0.79710145]
 [0.10526316 0.89473684]]
<Axes: >

```



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

```

precision    recall  f1-score   support

      0       0.35      0.50      0.41         48
      1       0.89      0.82      0.86        249

 accuracy          0.77         297
 macro avg       0.62      0.66      0.63         297
 weighted avg    0.81      0.77      0.78         297

```

```

#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)

```

```
0.7676767676767676
```

```

#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)

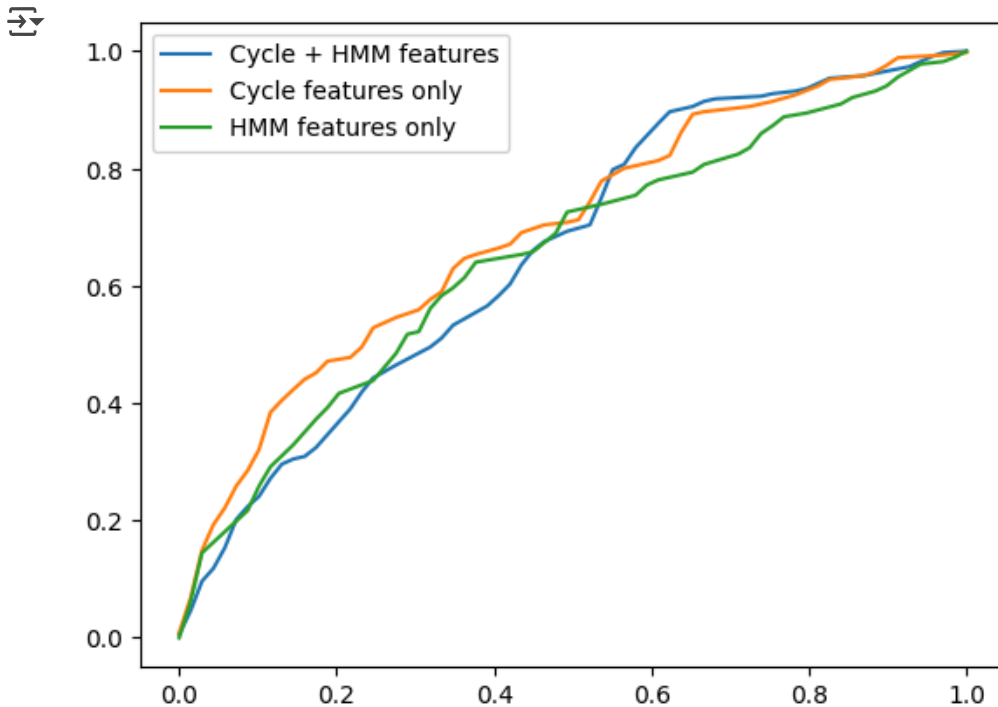
# # Create subplots
# fig, axes = plt.subplots(1, 3, figsize=(15, 5)) # 1 row, 3 columns

# 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']
```

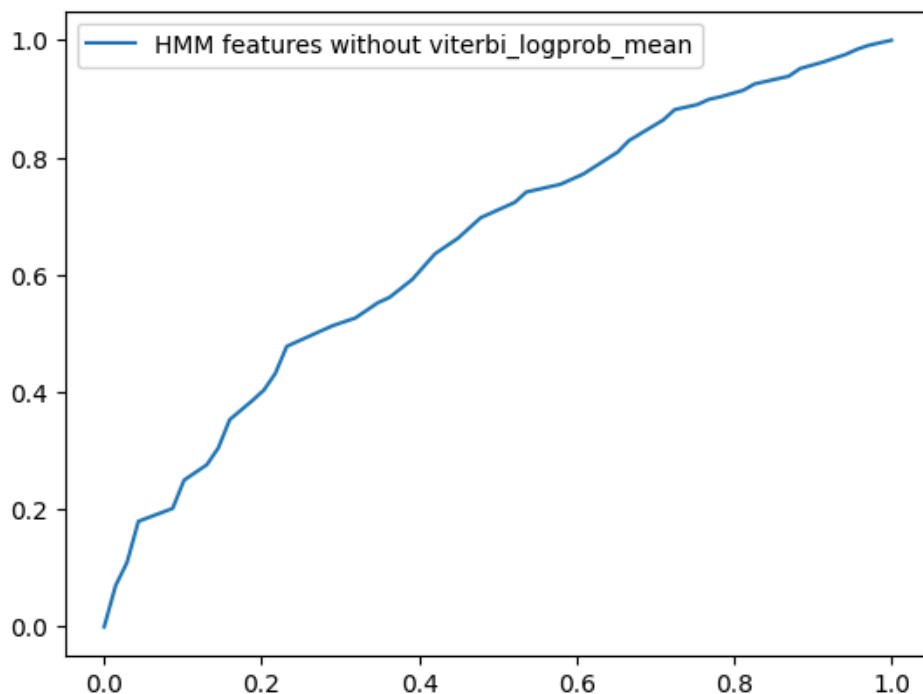
## ✓ without viterbi\_logprob\_mean

```
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_without_viterbi_logprob_mean.png')
```

➡   
[[0.23188406 0.76811594]  
 [0.10526316 0.89473684]]  
<Axes: >





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

```

↔
          precision    recall  f1-score   support

         0         0.23      0.40      0.29         40
         1         0.89      0.79      0.84        257

 accuracy                   0.74         297
 macro avg              0.56      0.60      0.57         297
 weighted avg           0.81      0.74      0.77         297

```

```
#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)
```

```
↔ 0.7407407407407407
```

## ✓ 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_m
shuffle=True, random_state=51)
```

```
↔ Performance with HMM features _without_viterbi_logprob_min
```

```
# Impute missing values using SimpleImputer
```

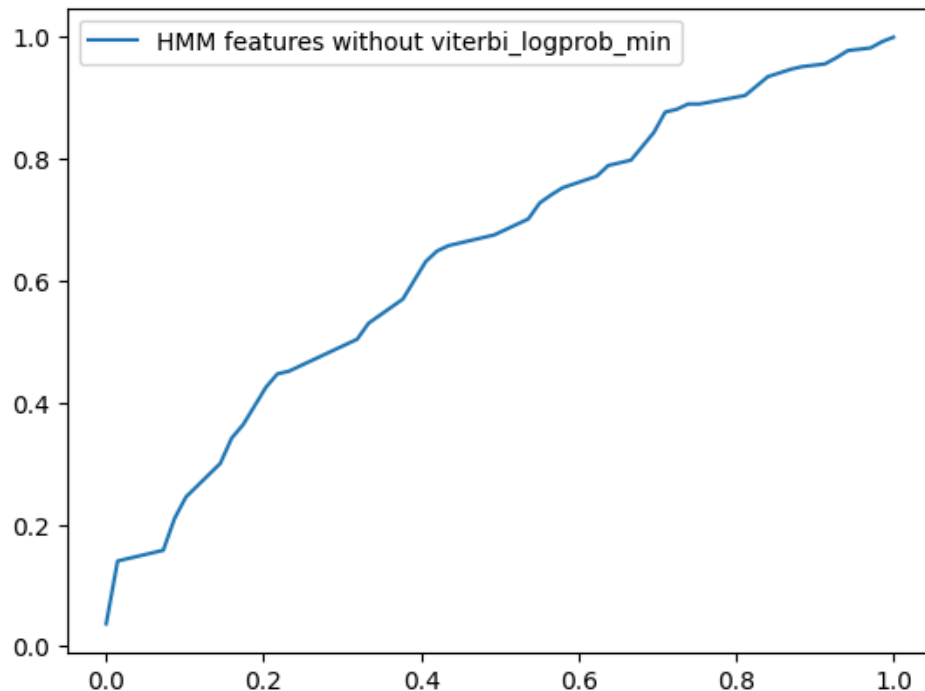
```
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_min = imputer.fit_transform(X_train_without_viterbi_logprob_min)
X_test_without_viterbi_logprob_min = imputer.transform(X_test_without_viterbi_logprob_min)
```

```
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='
fpr_without_viterbi_logprob_min, tpr_without_viterbi_logprob_min, thresholds_without_viterbi_logprob_min =
sns.lineplot(x=fpr_without_viterbi_logprob_min, y=tpr_without_viterbi_logprob_min, label='HMM features wit
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```

→ [[0.26086957 0.73913043]
    [0.10964912 0.89035088]]
<Axes: >

```



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

```

→
              precision    recall  f1-score   support

     0       0.26       0.42       0.32         43
     1       0.89       0.80       0.84        254

 accuracy          0.74         297
 macro avg         0.58         0.61         0.58         297
 weighted avg      0.80         0.74         0.77         297

```

```
#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)

```

```
→ 0.7441077441077442
```

## ✓ 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,
```

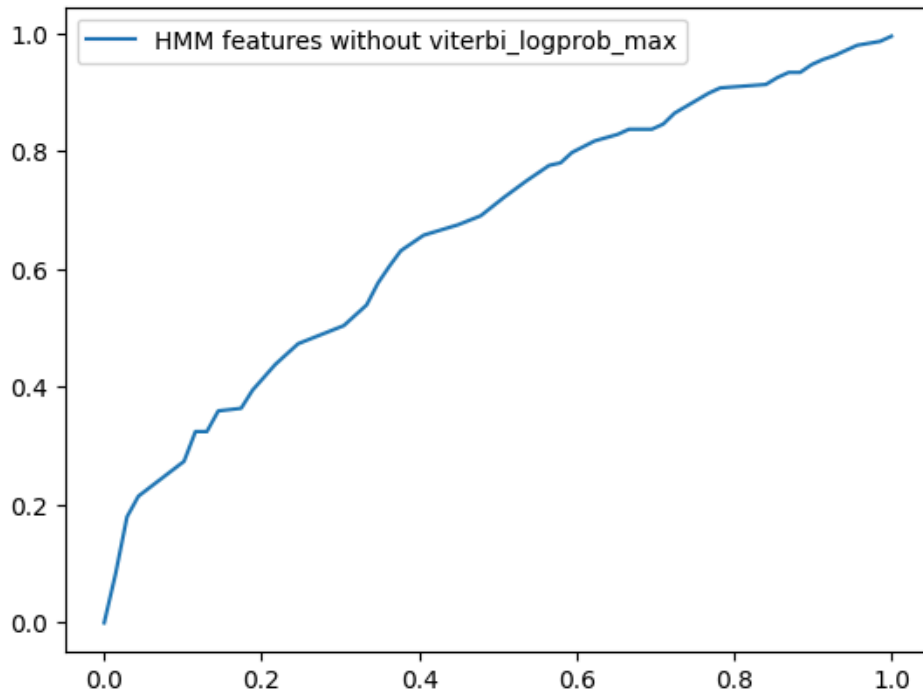
```
shuffle=True, random_state=51)
```

```
➡ 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='t
fpr_without_viterbi_logprob_max, tpr_without_viterbi_logprob_max, thresholds_without_viterbi_logprob_max =
sns.lineplot(x=fpr_without_viterbi_logprob_max, y=tpr_without_viterbi_logprob_max, label='HMM features with
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_wi
```

```
➡ [[0.2173913  0.7826087 ]
    [0.09210526 0.90789474]]
<Axes: >
```



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

```
➡
```

	precision	recall	f1-score	support
0	0.22	0.42	0.29	36
1	0.91	0.79	0.85	261
accuracy			0.75	297
macro avg	0.56	0.60	0.57	297
weighted avg	0.82	0.75	0.78	297

```
#overall accuracy:
```

```
print((y_pred_without_viterbi_logprob_max==y_test_without_viterbi_logprob_max).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)
```

↩ 0.74747474747475

## ✓ 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

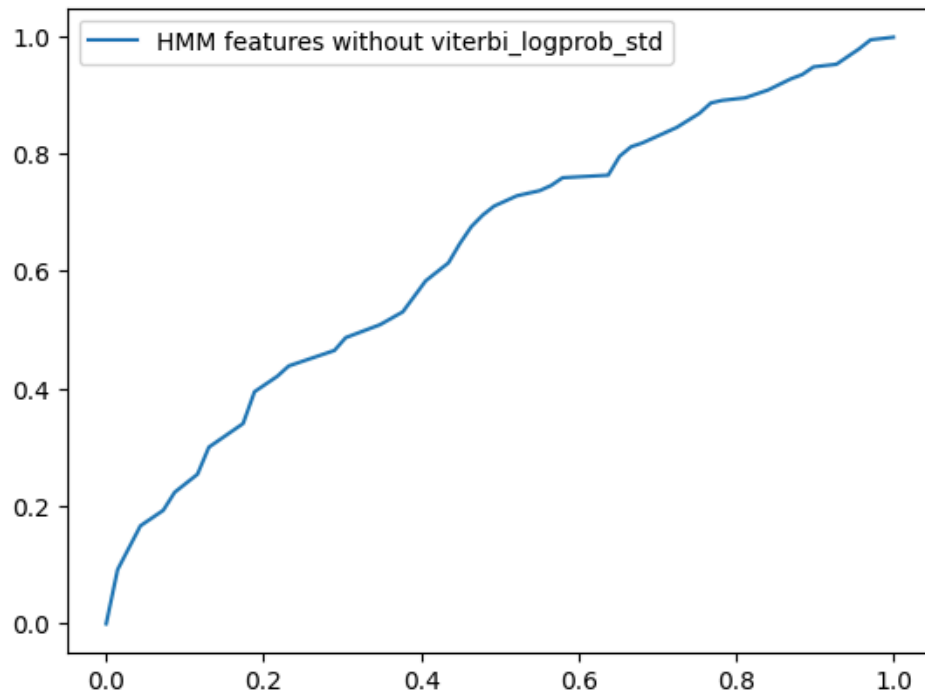
```
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_std = imputer.fit_transform(X_train_without_viterbi_logprob_std)
X_test_without_viterbi_logprob_std = imputer.transform(X_test_without_viterbi_logprob_std)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

```
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
```

```

[[0.2173913  0.7826087 ]
 [0.10526316 0.89473684]]
<Axes: >

```



```

print(classification_report(y_pred_without_viterbi_logprob_std, y_test_without_viterbi_logprob_std))

```

```

precision    recall  f1-score   support

      0       0.22      0.38      0.28         39
      1       0.89      0.79      0.84        258

 accuracy          0.74         297
 macro avg         0.56         297
weighted avg         0.81         297

```

```

#overall accuracy:
print((y_pred_without_viterbi_logprob_std==y_test_without_viterbi_logprob_std).sum()/len(y_pred_without_viterbi_logprob_std))
#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)

```

```

0.7373737373737373

```

## 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 ')

```

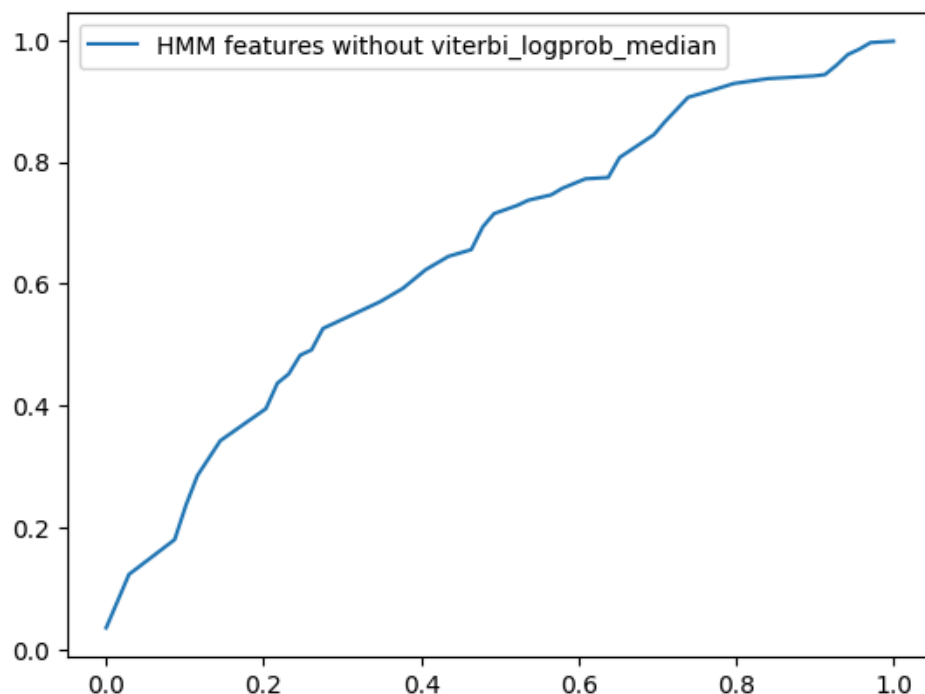
```
X_train_without_viterbi_logprob_median, X_test_without_viterbi_logprob_median, y_train_without_viterbi_logprob_median, y_test_without_viterbi_logprob_median, shuffle=True, random_state=51)
```

➡ 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, 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 viterbi_logprob_median')
plt.savefig('/content/drive/MyDrive/fall_research/feature_distribution_plots/viterbi_adjusted_plots/xgb_w')
```

➡ [[0.23188406 0.76811594]  
[0.08333333 0.91666667]]  
<Axes: >



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

➡

	precision	recall	f1-score	support
0	0.23	0.46	0.31	35
1	0.92	0.80	0.85	262
accuracy			0.76	297
macro avg	0.57	0.63	0.58	297
weighted avg	0.84	0.76	0.79	297

```
#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)

➡ 0.7575757575757576
```

## ✓ 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_logp
shuffle=True, random_state=51)

➡ Performance with HMM features _without_complete_logprob_mean

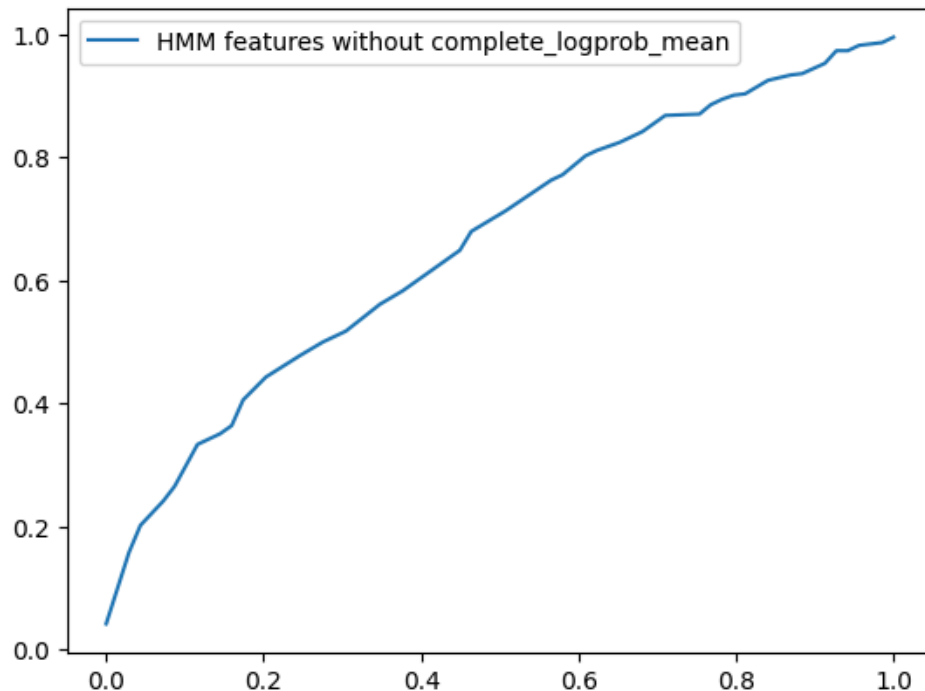
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_complete_logprob_mean = imputer.fit_transform(X_train_without_complete_logprob_mean)
X_test_without_complete_logprob_mean = imputer.transform(X_test_without_complete_logprob_mean)

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, normali
fpr_without_complete_logprob_mean, tpr_without_complete_logprob_mean, thresholds_without_complete_logprob_
sns.lineplot(x=fpr_without_complete_logprob_mean, y=tpr_without_complete_logprob_mean, label='HMM features
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```

[[0.20289855 0.79710145]
 [0.09649123 0.90350877]]
<Axes: >

```



```

print(classification_report(y_pred_without_complete_logprob_mean, y_test_without_complete_logprob_mean))

```

```

precision    recall  f1-score   support

      0       0.20      0.39      0.27        36
      1       0.90      0.79      0.84       261

 accuracy          0.74        297
 macro avg       0.55      0.59      0.55        297
 weighted avg    0.82      0.74      0.77        297

```

```

#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)

```

```

0.7407407407407407

```

## 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 ')

```



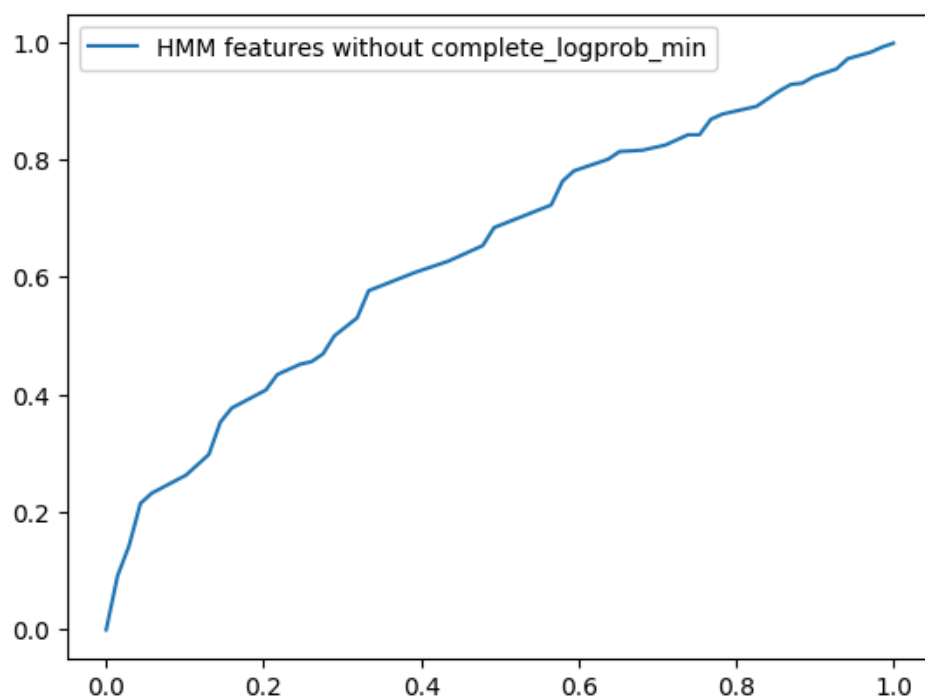
```
X_train_without_complete_logprob_min, X_test_without_complete_logprob_min, y_train_without_complete_logprob_min, y_test_without_complete_logprob_min)
shuffle=True, random_state=51)
```

➡ 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=True))
fpr_without_complete_logprob_min, tpr_without_complete_logprob_min, thresholds_without_complete_logprob_min = roc_curve(y_test_without_complete_logprob_min, y_score_without_complete_logprob_min)
sns.lineplot(x=fpr_without_complete_logprob_min, y=tpr_without_complete_logprob_min, label='HMM features w/o complete logprob min')
plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w/o complete logprob min')
plt.close()
```

➡ [[0.17391304 0.82608696]  
[0.11403509 0.88596491]]  
<Axes: >



```
print(classification_report(y_pred_without_complete_logprob_min, y_test_without_complete_logprob_min))
```

➡

	precision	recall	f1-score	support
0	0.17	0.32	0.22	38
1	0.89	0.78	0.83	259
accuracy			0.72	297
macro avg	0.53	0.55	0.53	297
weighted avg	0.79	0.72	0.75	297

```
#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)

➡ 0.7205387205387206
```

## ✓ 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_logpro
shuffle=True, random_state=51)

➡ Performance with HMM features _without_complete_logprob_max

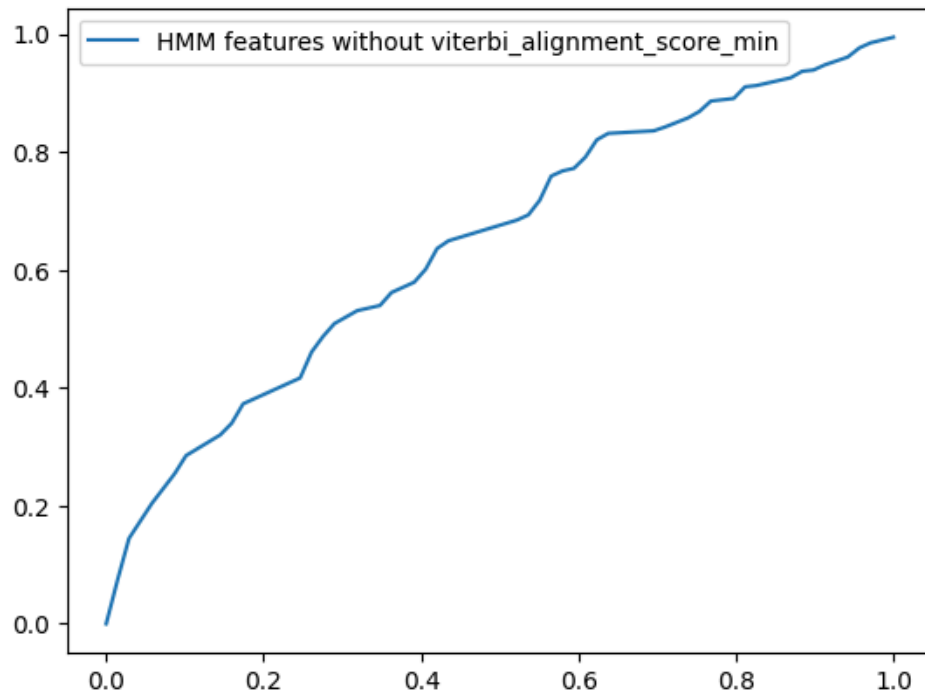
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_complete_logprob_max = imputer.fit_transform(X_train_without_complete_logprob_max)
X_test_without_complete_logprob_max = imputer.transform(X_test_without_complete_logprob_max)

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
fpr_without_complete_logprob_max, tpr_without_complete_logprob_max, thresholds_without_complete_logprob_ma
sns.lineplot(x=fpr_without_complete_logprob_max, y=tpr_without_complete_logprob_max, label='HMM features w
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```

→ [[0.20289855 0.79710145]
    [0.10964912 0.89035088]]
<Axes: >

```



```

print(classification_report(y_pred_without_complete_logprob_max, y_test_without_complete_logprob_max))

```

```

→
              precision    recall  f1-score   support

     0       0.20        0.36        0.26         39
     1       0.89        0.79        0.84        258

 accuracy          0.73         297
 macro avg         0.55         0.57         0.55         297
 weighted avg      0.80         0.73         0.76         297

```

```

#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)

```

```

→ 0.7306397306397306

```

## ✓ 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 ')

```

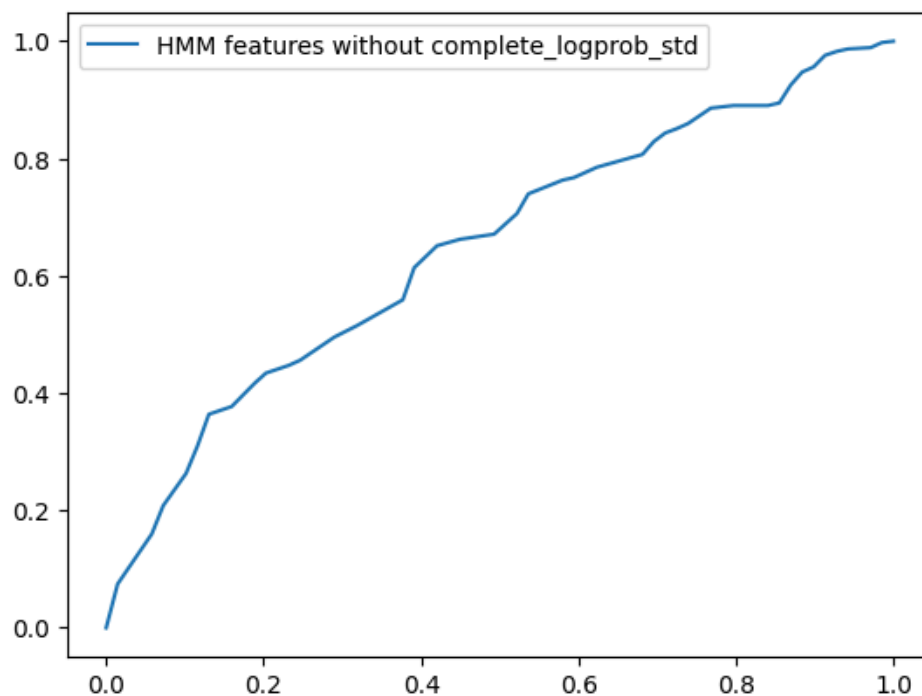
```
X_train_without_complete_logprob_std, X_test_without_complete_logprob_std, y_train_without_complete_logprob_std, y_test_without_complete_logprob_std)
shuffle=True, random_state=51)
```

➡ 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=True))
fpr_without_complete_logprob_std, tpr_without_complete_logprob_std, thresholds_without_complete_logprob_std = roc_curve(y_test_without_complete_logprob_std, y_score_without_complete_logprob_std)
sns.lineplot(x=fpr_without_complete_logprob_std, y=tpr_without_complete_logprob_std, label='HMM features w/o complete logprob std')
plt.savefig('/content/drive/MyDrive/fall_research/feature_distribution_plots/viterbi_adjusted_plots/xgb_w/o_complete_logprob_std.png')
```

➡ [[0.17391304 0.82608696]  
[0.10964912 0.89035088]]  
<Axes: >



```
print(classification_report(y_pred_without_complete_logprob_std, y_test_without_complete_logprob_std))
```

➡

	precision	recall	f1-score	support
0	0.17	0.32	0.23	37
1	0.89	0.78	0.83	260
accuracy			0.72	297
macro avg	0.53	0.55	0.53	297
weighted avg	0.80	0.72	0.76	297

```
#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)

➡ 0.7239057239057239
```

## ✓ 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_log
shuffle=True, random_state=51)

➡ Performance with HMM features _without_viterbi_logprob_median

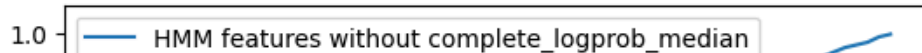
# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_logprob_median = imputer.fit_transform(X_train_without_viterbi_logprob_median)
X_test_without_viterbi_logprob_median = imputer.transform(X_test_without_viterbi_logprob_median)

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
```

```

→ [[0.23188406 0.76811594]
    [0.10087719 0.89912281]]
<Axes: >

```



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

```

→
          precision    recall  f1-score   support

     0           0.23       0.41       0.30         39
     1           0.90       0.79       0.84        258

 accuracy                   0.74         297
 macro avg           0.57       0.60       0.57         297
 weighted avg        0.81       0.74       0.77         297

```

```
#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)

```

```
→ 0.7441077441077442
```

```

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_alignment ')
```

```

X_train_without_viterbi_alignment, X_test_without_viterbi_alignment, y_train_without_viterbi_alignment, y_
shuffle=True, random_state=51)

```

```
→ Performance with HMM features _without_viterbi_alignment
```

```
# Impute missing values using SimpleImputer
```

```

imputer = SimpleImputer(strategy='mean') # Replace 'mean' with other strategies if needed
X_train_without_viterbi_alignment = imputer.fit_transform(X_train_without_viterbi_alignment)
X_test_without_viterbi_alignment = imputer.transform(X_test_without_viterbi_alignment)

```

```

clf = RFC(random_state=101)
clf.fit(X_train_without_viterbi_alignment, y_train_without_viterbi_alignment)
y_pred_without_viterbi_alignment = clf.predict(X_test_without_viterbi_alignment)
y_score_without_viterbi_alignment = clf.predict_proba(X_test_without_viterbi_alignment)
print(confusion_matrix(y_test_without_viterbi_alignment, y_pred_without_viterbi_alignment, normalize='true
fpr_without_viterbi_alignment, tpr_without_viterbi_alignment, thresholds_without_viterbi_alignment = roc_c

```