```
# 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-l
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.11/dist-packages (from scikit-le
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from scikit-l
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from s
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 xgboo
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-12-9_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	34.434783	15	4.4089
1	U303F6B17404145	PCOS	19	33	26.5	26.250000	14	7.8049
2	U2B70EC15755124	PCOS	28	43	38.0	37.785714	15	3.9258
3	U2F65CA17170226	PCOS	27	40	40.0	36.400000	13	5.6833
4	U2F823A17212446	PCOS	27	36	34.0	32.750000	9	4.0311

```
# 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-1380-1fe60784182b>: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
  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[tar
                                                     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'))
```

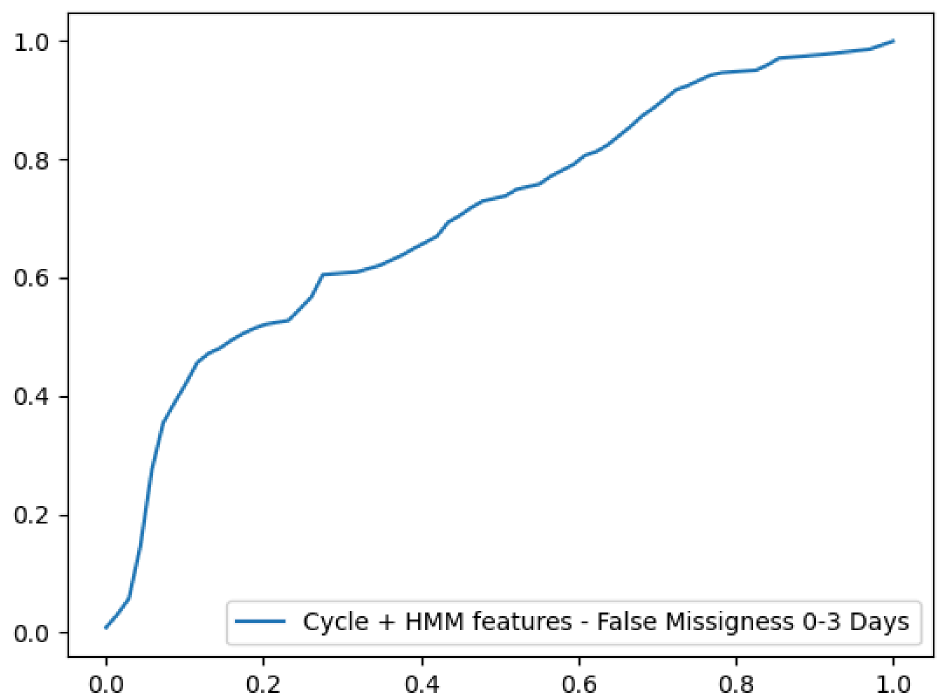
```
[[0.33333333 0.66666667]
 [0.15044248 0.84955752]]
```

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

```
              precision    recall  f1-score   support

           0       0.33      0.40      0.37        57
           1       0.85      0.81      0.83       238

    accuracy                           0.73       295
   macro avg       0.59      0.61      0.60       295
weighted avg       0.75      0.73      0.74       295
```

```
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=No
#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))
```

```
0.7288135593220338
```

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'))
```

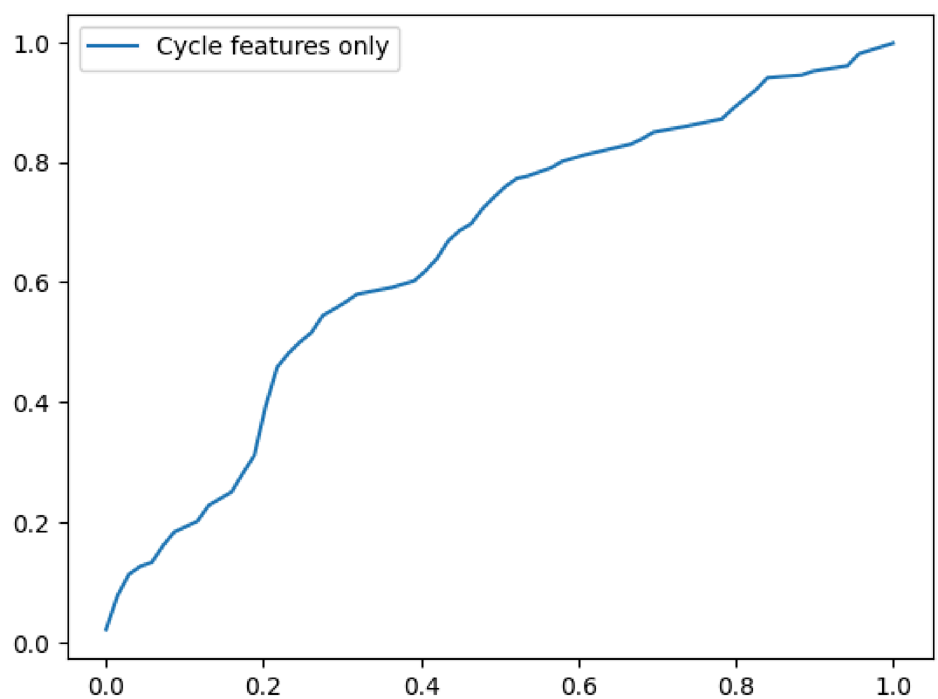
```
[[0.17391304 0.82608696]
 [0.09292035 0.90707965]]
```

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

```
              precision    recall  f1-score   support

           0       0.17      0.36      0.24        33
           1       0.91      0.78      0.84       262

    accuracy                           0.74       295
   macro avg       0.54      0.57      0.54       295
weighted avg       0.83      0.74      0.77       295
```

```
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))
```

```
0.735593220338983
```

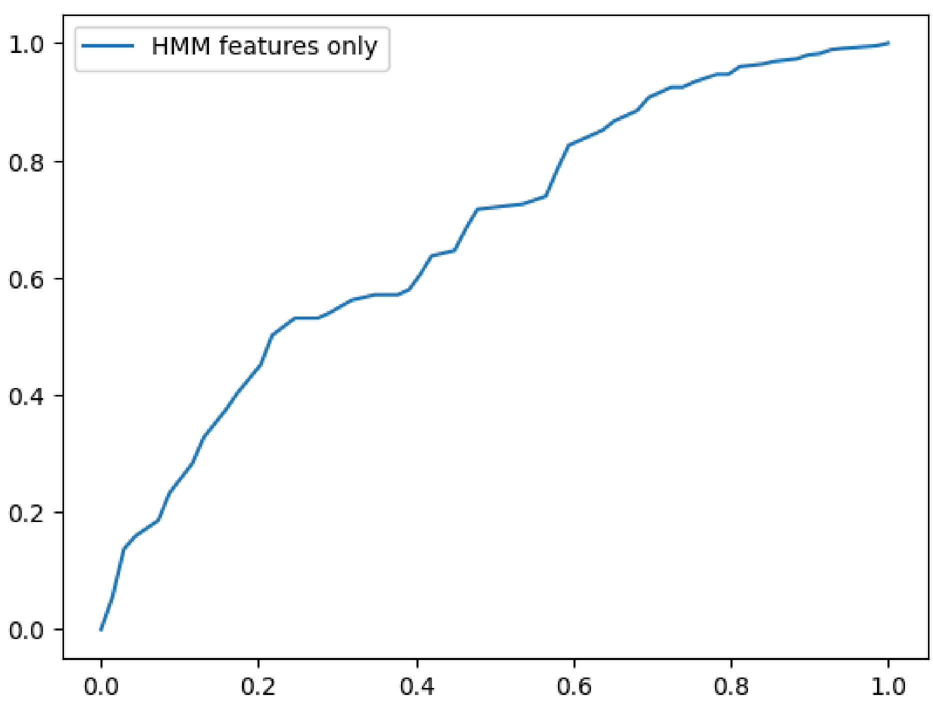
HMM Features only

```
#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')
```

```
[[0.30434783 0.69565217]
 [0.09734513 0.90265487]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.17      0.36      0.24        33
           1       0.91      0.78      0.84       262

    accuracy                           0.74       295
   macro avg       0.54      0.57      0.54       295
weighted avg       0.83      0.74      0.77       295
```

```
#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)
```

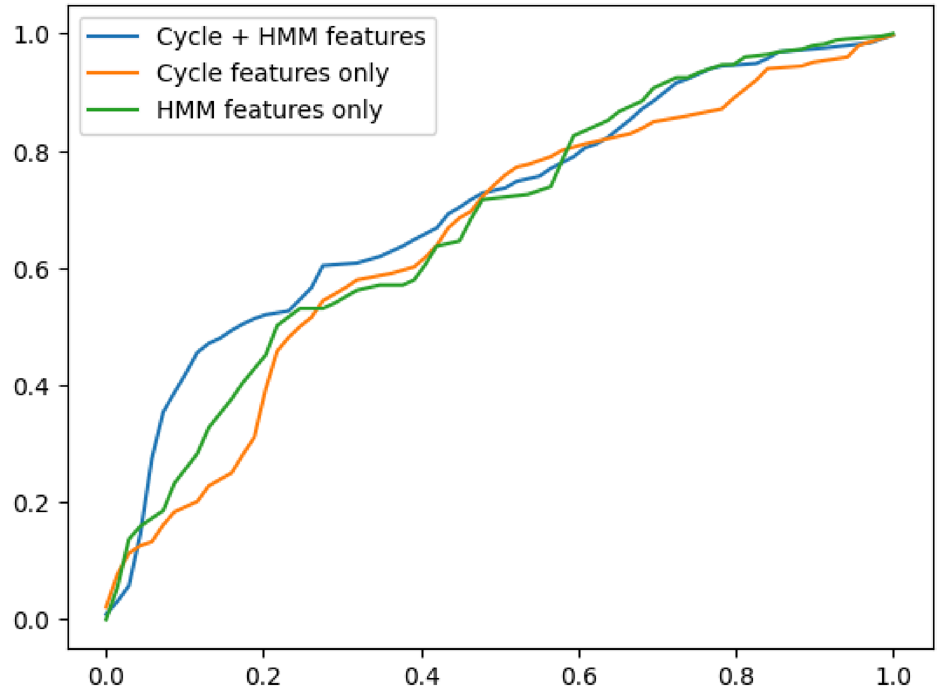
```
0.735593220338983
```

```
#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']
```

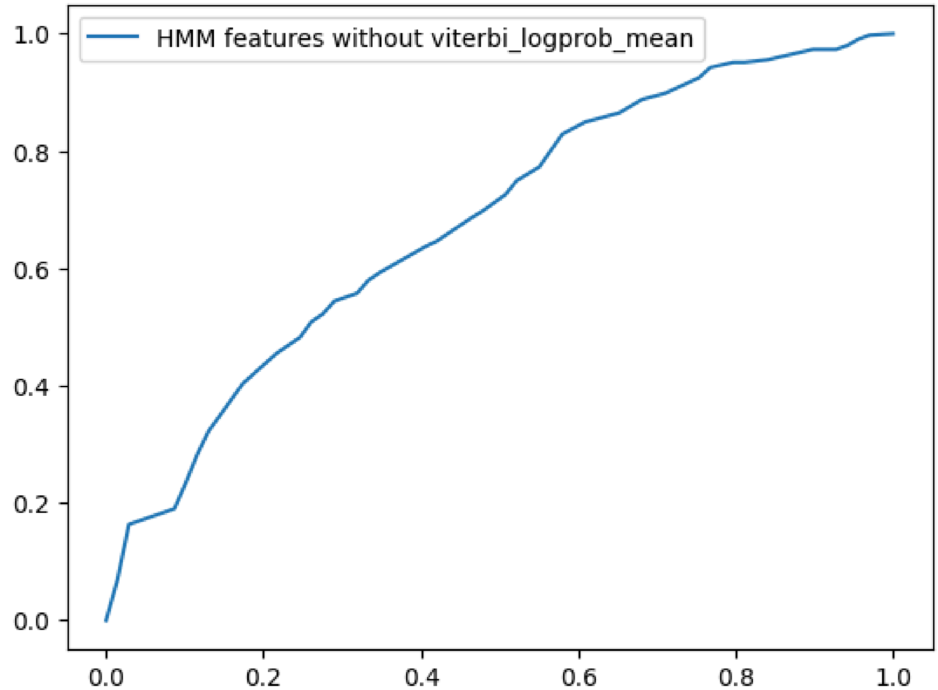
```
print('Performance with HMM features _without_viterbi_logprob_mean ')

X_train_without_viterbi_logprob_mean, X_test_without_viterbi_logprob_mean, y_train_without_viterbi_logprob
                                                    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
fpr_without_viterbi_logprob_mean, tpr_without_viterbi_logprob_mean, thresholds_without_viterbi_logprob_mea
sns.lineplot(x=fpr_without_viterbi_logprob_mean, y=tpr_without_viterbi_logprob_mean, label='HMM features w
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```
[[0.31884058 0.68115942]
 [0.11504425 0.88495575]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.32      0.46      0.38        48
           1       0.88      0.81      0.85       247

    accuracy                           0.75       295
   macro avg       0.60      0.63      0.61       295
weighted avg       0.79      0.75      0.77       295
```

```
#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)
```

```
0.752542372881356
```

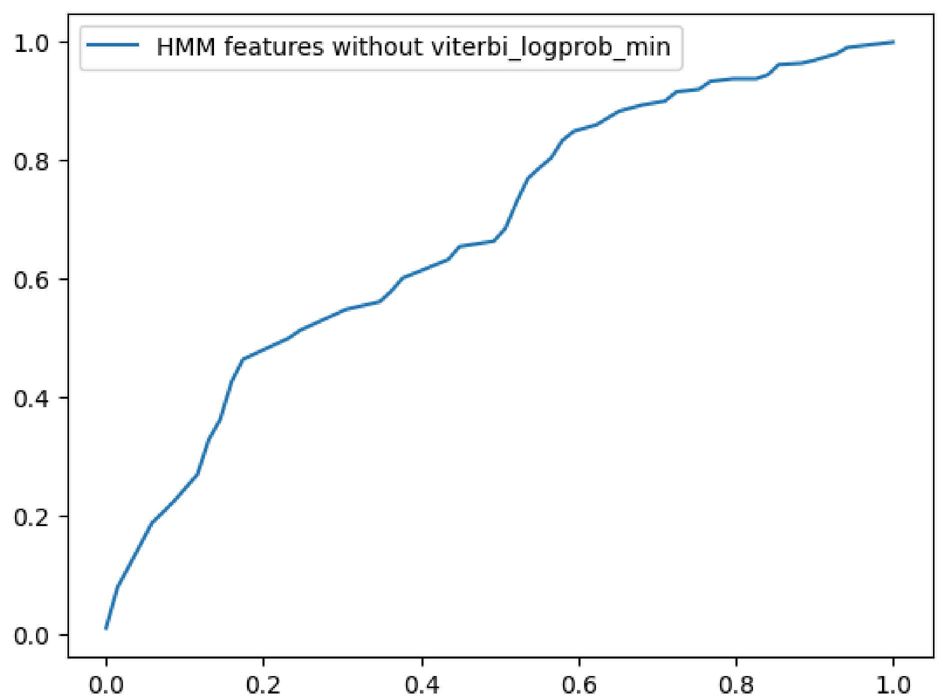
without viterbi_logprob_min

```
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
```

```
[[0.31884058 0.68115942]
 [0.10619469 0.89380531]]
<Axes: >
```




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

```
              precision    recall  f1-score   support
```

```
              0       0.32      0.48      0.38        46
              1       0.89      0.81      0.85       249

       accuracy                           0.76       295
      macro avg       0.61      0.64      0.62       295
   weighted avg       0.80      0.76      0.78       295
```

```
#overall accuracy:
print((y_pred_without_viterbi_logprob_min==y_test_without_viterbi_logprob_min).sum()/len(y_pred_without_vi
#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)
```

```
0.7593220338983051
```

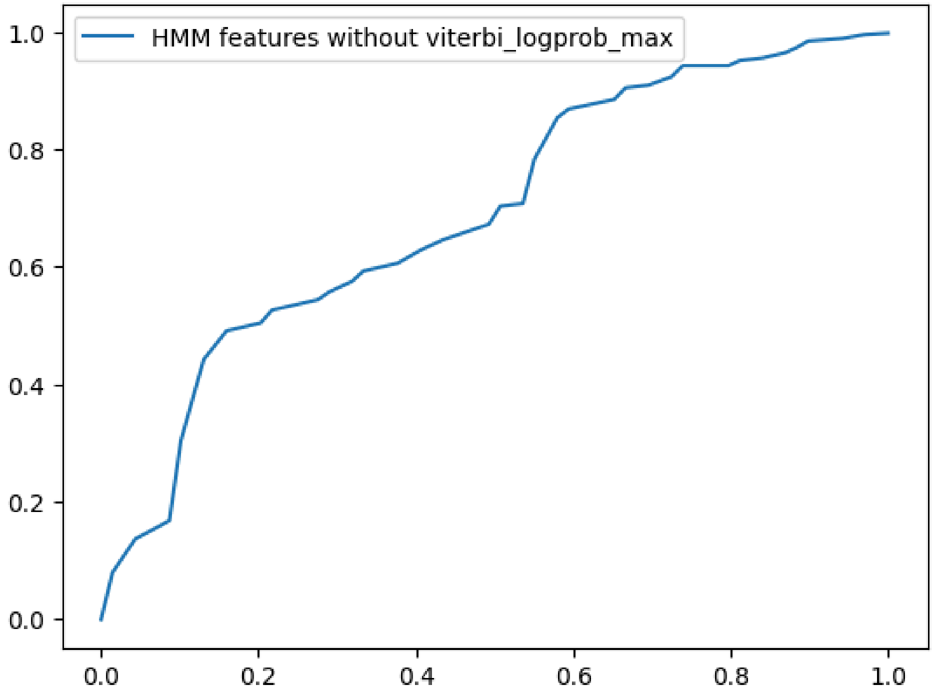
without viterbi_logprob_max

```
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_ma
                                                    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
```

```
[[0.30434783 0.69565217]
 [0.09292035 0.90707965]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.30      0.50      0.38        42
           1       0.91      0.81      0.86       253

    accuracy                           0.77       295
   macro avg       0.61      0.66      0.62       295
weighted avg       0.82      0.77      0.79       295
```

```
#overall accuracy:
print((y_pred_without_viterbi_logprob_max==y_test_without_viterbi_logprob_max).sum()/len(y_pred_without_vi
#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)
```

```
0.7661016949152543
```

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']
```

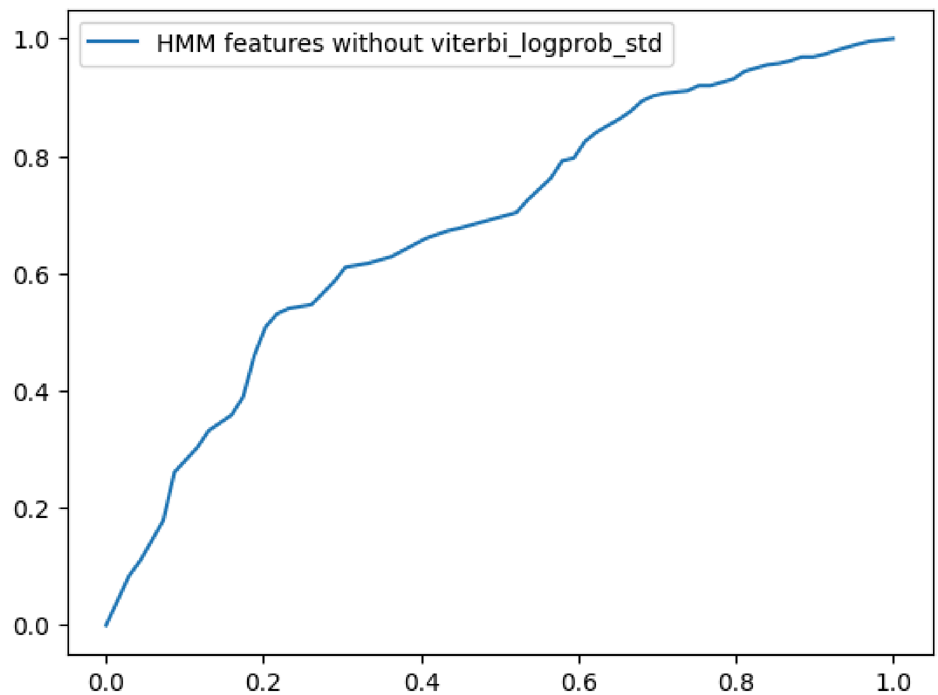
```
print('Performance with HMM features _without_viterbi_logprob_std ')
```

```
X_train_without_viterbi_logprob_std, X_test_without_viterbi_logprob_std, y_train_without_viterbi_logprob_s
                                                    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
```

```
[[0.30434783 0.69565217]
 [0.09734513 0.90265487]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.30      0.49      0.38        43
           1       0.90      0.81      0.85       252

    accuracy                           0.76       295
   macro avg       0.60      0.65      0.61       295
weighted avg       0.82      0.76      0.78       295
```

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

```
0.7627118644067796
```

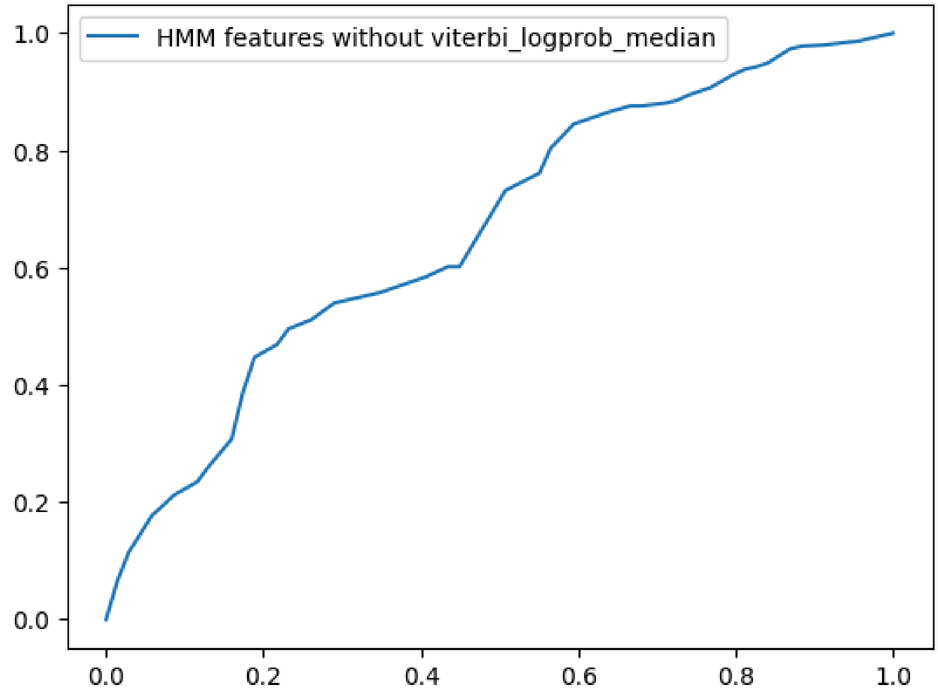
without viterbi_logprob_median

```
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
```

```
[[0.28985507 0.71014493]
 [0.11946903 0.88053097]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.29      0.43      0.34        47
           1       0.88      0.80      0.84       248

    accuracy                           0.74       295
   macro avg       0.59      0.61      0.59       295
weighted avg       0.79      0.74      0.76       295
```

```
#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)
```

```
0.7423728813559322
```

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']
```

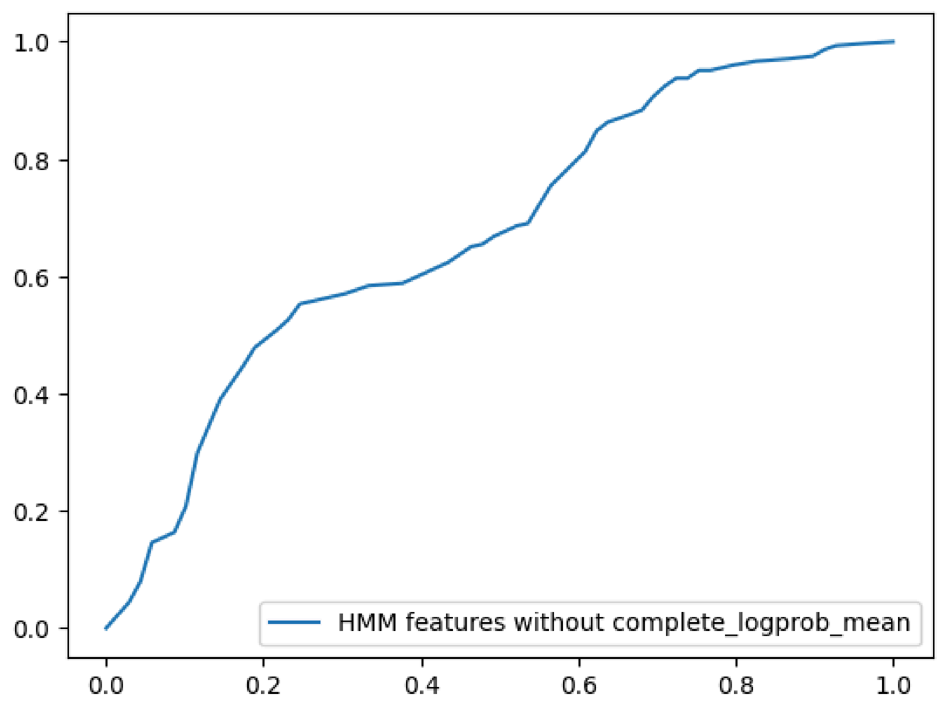
```
print('Performance with HMM features _without_complete_logprob_mean ')
```

```
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
```

```
[[0.31884058 0.68115942]
 [0.11946903 0.88053097]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.32      0.45      0.37        49
           1       0.88      0.81      0.84       246

    accuracy                           0.75       295
   macro avg       0.60      0.63      0.61       295
weighted avg       0.79      0.75      0.77       295
```

```
#overall accuracy:
print((y_pred_without_complete_logprob_mean==y_test_without_complete_logprob_mean).sum()/len(y_pred_withou
#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)
```

```
0.7491525423728813
```

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']
```

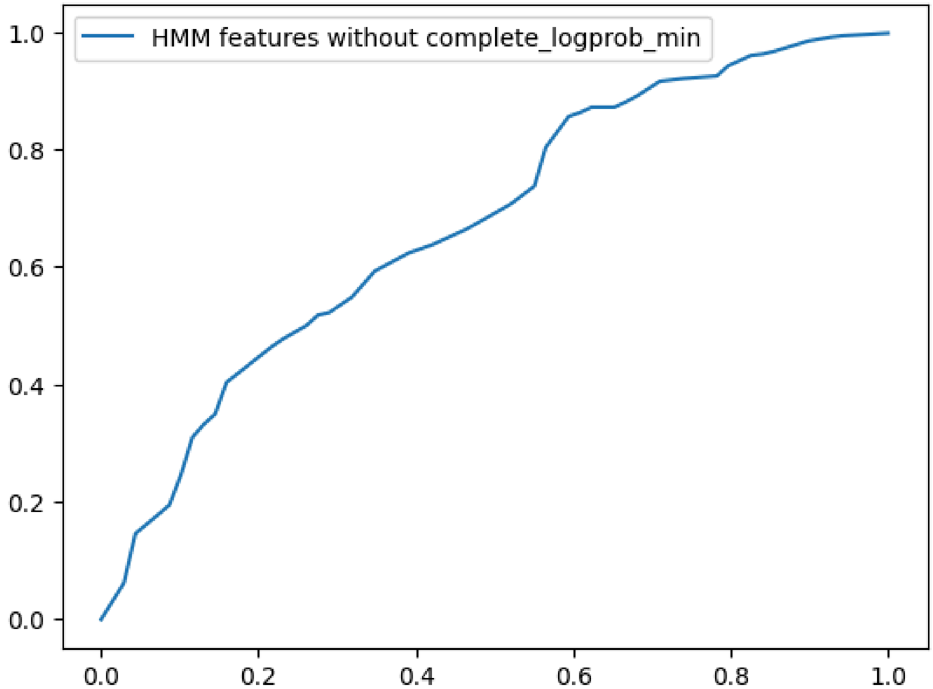
```
print('Performance with HMM features _without_complete_logprob_min ')

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
```

```
[[0.31884058 0.68115942]
 [0.11061947 0.88938053]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.32      0.47      0.38        47
           1       0.89      0.81      0.85       248

    accuracy                           0.76       295
   macro avg       0.60      0.64      0.61       295
weighted avg       0.80      0.76      0.77       295
```

```
#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)
```

```
0.7559322033898305
```

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']
```

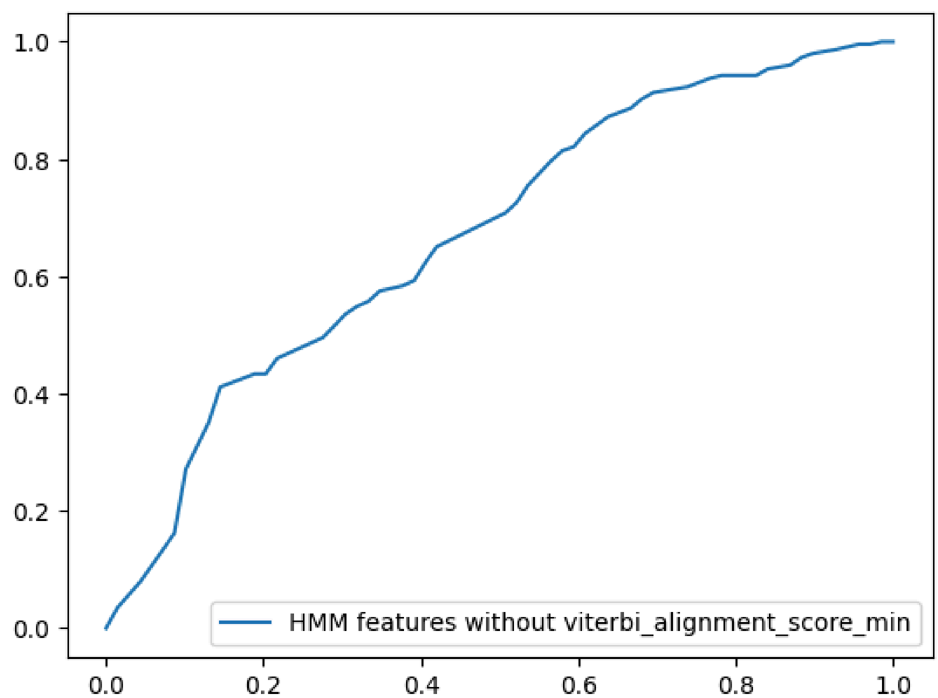
```
print('Performance with HMM features _without_complete_logprob_max ')
```

```
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
```

```
[[0.31884058 0.68115942]
 [0.09292035 0.90707965]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.32      0.51      0.39        43
           1       0.91      0.81      0.86       252

    accuracy                           0.77       295
   macro avg       0.61      0.66      0.63       295
weighted avg       0.82      0.77      0.79       295
```

```
#overall accuracy:
```

```
print((y_pred_without_complete_logprob_max==y_test_without_complete_logprob_max).sum()/len(y_pred_without_c
#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)
```

```
0.7694915254237288
```

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']
```

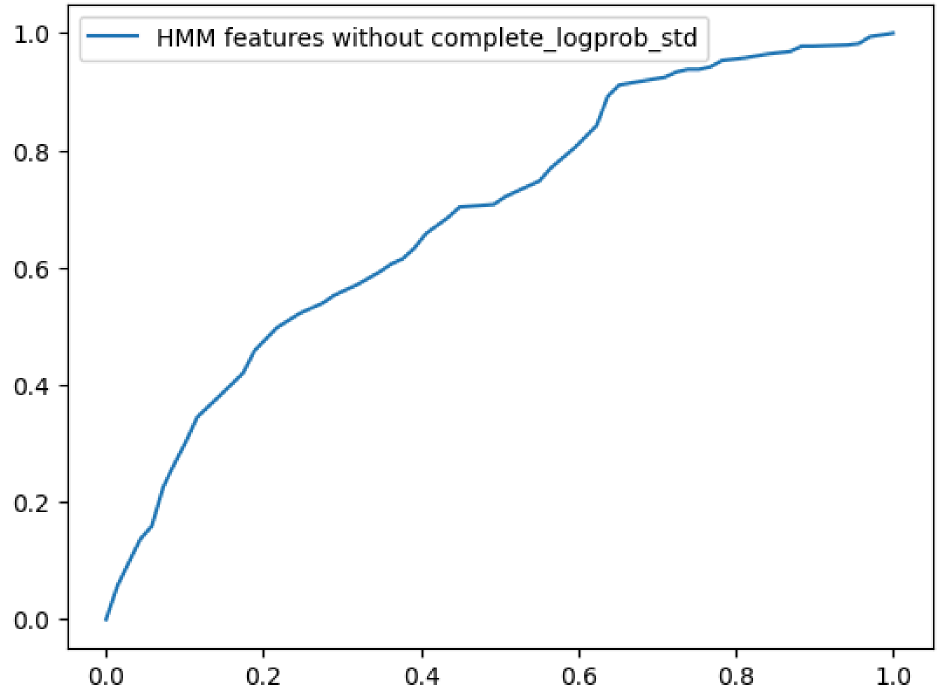
```
print('Performance with HMM features _without_complete_logprob_std ')

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_st
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
```

```
[[0.31884058 0.68115942]
 [0.07964602 0.92035398]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.32      0.55      0.40        40
           1       0.92      0.82      0.86       255

    accuracy                           0.78       295
   macro avg       0.62      0.68      0.63       295
weighted avg       0.84      0.78      0.80       295
```

```
#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)
```

```
0.7796610169491526
```

without complete_logprob_median

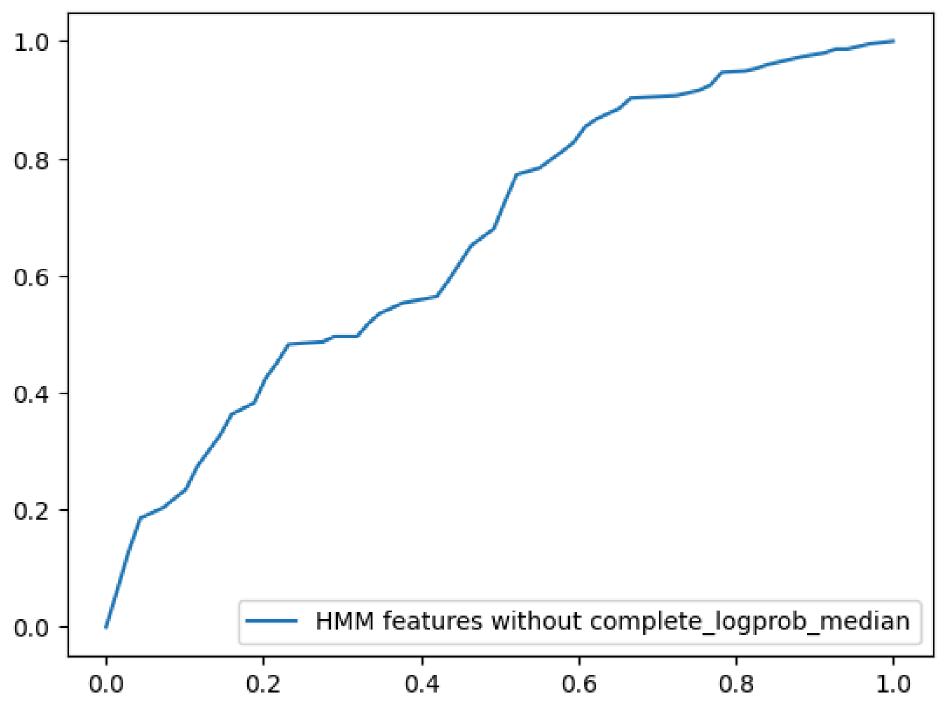
```
HMM_features = ['viterbi_logprob_mean',
       'viterbi_logprob_min', 'viterbi_logprob_max', 'viterbi_logprob_std',
       'viterbi_logprob_median', 'complete_logprob_mean',
       'complete_logprob_min', 'complete_logprob_max', 'complete_logprob_std']
```

```
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
```

```
[[0.27536232 0.72463768]
 [0.09292035 0.90707965]]
<Axes: >
```




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

```
              precision    recall  f1-score   support

           0       0.28      0.47      0.35        40
           1       0.91      0.80      0.85       255

    accuracy                           0.76       295
   macro avg       0.59      0.64      0.60       295
weighted avg       0.82      0.76      0.78       295
```

```
#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)
```

```
0.7593220338983051
```

```
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
sns.lineplot(x=fpr_without_viterbi_alignment, y=tpr_without_viterbi_alignment, label='HMM features without
#plt.savefig('/content/drive/MyDrive/fall_research/feature distribution plots/viterbi adjusted plots/xgb_w
```

```
[[0.27536232 0.72463768]
 [0.09292035 0.90707965]]
<Axes: >
```