NAME : VAISHNAV NALLADATH **SLOT : C2 + TC2**

REGISTRATION NO. : 20BCE2535 **DATE : APRIL 04, 2023**

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE4020

MACHINE LEARNING

ETH

THEORY

TOPIC:

SEMI – SUPERVISED LEARNING (WEB CONTENT)

Abstract

Semi-supervised learning is a machine learning technique that combines both labeled and unlabeled data to improve the accuracy of models. In the context of web content, semi-supervised learning can be used to classify and categorize large amounts of data, such as text and images, that may not have explicit labels. By leveraging the relationships and patterns between labeled and unlabeled data, semi-supervised learning can effectively identify and classify web content with higher accuracy than traditional supervised learning methods. This approach is particularly useful for applications such as web search, recommendation systems, and content filtering, where there is a vast amount of unstructured data that needs to be organized and analyzed. However, as with any machine learning technique, semi-supervised learning has its limitations and requires careful consideration of the data and model architecture to achieve optimal results. This can be particularly useful for applications such as sentiment analysis, topic modeling, and content recommendation systems. Overall, semi-supervised learning can help to improve the efficiency and accuracy of web content analysis, making it a valuable tool for businesses and researchers alike.

Introduction

Web content analysis has become an essential part of businesses and researchers' strategies to understand their target audience, identify trends, and generate insights. However, analyzing vast amounts of text data can be a daunting task, and manually labeling every data point can be extremely time-consuming and resource intensive. Semi-supervised learning offers an alternative approach to analyzing web content, which can significantly reduce the need for manual labeling and improve the accuracy of analysis. In semi-supervised learning, models are trained using a combination of labeled and unlabeled data, allowing them to learn patterns and relationships in the data and classify new, unlabeled data accurately. This approach can be particularly useful in analyzing web content, where a vast amount of data is available, but only a small portion of it may be labeled. In this paper, we will explore the functionalities, benefits, and challenges of using semi-supervised learning for web content analysis.

Functionalities of Semi – Supervised Learning

Semi-supervised learning is a machine learning technique that falls in between supervised and unsupervised learning. In this case, some of the training data is labeled, while the rest is unlabeled. This allows the model to learn from both labeled and unlabeled data, which can be beneficial when it's labeled data is scarce or expensive to obtain. One potential application of semi-supervised learning is in the field of web content analysis. With the vast amount of information available on the internet, it can be challenging to classify and organize web content. However, with the help of semi-supervised learning, it is possible to classify web content with high accuracy.

In this project we use a combination of supervised and unsupervised learning techniques to classify web pages. Initially, a small set of web pages are physically labeled to provide a starting point for the model. This labeled data is then used to train a supervised learning model. The model is then used to predict the category of other web pages. However, since labeled data is often limited in web content analysis, the model is also trained using unsupervised learning techniques. This involves clustering similar web pages together based on their content. The model then uses these clusters to infer the category of other web pages. The project's functionality is based on a web crawler that scans the internet for web pages and extracts their content. The content is then fed into the model, which predicts the category of the web page. The categories are then displayed on a user interface, which allows users to search for web pages based on their category.

In summary, the Web Content Classification Project is a semi-supervised learning project that aims to classify web pages based on their content. The project uses a combination of supervised and unsupervised learning techniques to classify web pages accurately. The project's functionality is based on a web crawler that extracts web page content and feeds it into the model. The results are then displayed on a user interface, which allows users to search for web pages based on their category.

Semi-supervised learning can be a valuable tool in analyzing and categorizing web content. Some of the key functionalities of semi-supervised learning in the context of web content include:

1. **Data labeling:** In semi-supervised learning, labeled data is used to train a model, which can then be used to categorize larger amounts of unlabeled data. With web content, this can involve manually describing a small set of data, which can then be used to train a model that can categorize much larger amounts of unlabeled data.

2. **Text classification:** One of the main applications of semi-supervised learning in web content analysis is text classification. This involves categorizing text data into different topics or themes. For example, a model might be trained to categorize news articles into different categories such as politics, sports, and entertainment.

3. **Sentiment analysis:** Another application of semi-supervised learning in web content analysis is sentiment analysis. This involves categorizing text data based on the sentiment expressed, such as positive, negative, or neutral. This can be particularly useful for analyzing social media data, online reviews, and customer feedback.

4. **Topic modeling:** Semi-supervised learning can also be used for topic modeling, which involves identifying the underlying themes or topics in a set of text data. This can be useful for understanding the content of large amounts of web content, such as news articles or blog posts.

5. **Content recommendation:** Finally, semi-supervised learning can be used for content recommendation systems, which suggest relevant content to users based on their browsing history or other behavior. This can involve training a model to recommend similar content to a user based on the content they have previously viewed or interacted with.

Overall, semi-supervised learning can be a powerful tool in web content analysis, enabling businesses and researchers to more efficiently and accurately categorize and analyze large amounts of text data.

Observations

The Web Content Classification Project is an interesting and useful application of semi-supervised learning in the field of web content analysis. It demonstrates the potential of semi-supervised learning to improve the accuracy of web content classification by combining both labeled and unlabeled data.

One possible additional functionality for the project could be to allow users to provide feedback on the accuracy of the classification. This feedback can be used to improve the model and make it more accurate over time. For example, users could be given the option to suggest a different category for a misclassified web page, or to flag a web page that is incorrectly classified.

Another possible functionality could be to incorporate user preferences into the classification process. For instance, users could specify their preferred categories for certain topics, and the model could prioritize these preferences when classifying web pages. This would make the classification process more personalized and tailored to individual user needs.

Overall, there are many possibilities for additional functionalities that could be added to the Web Content Classification Project to improve its accuracy and usefulness. By incorporating user feedback and preferences, the project could become even more effective at classifying web content and providing relevant search results.

Code

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline


sns.set()


df = pd.read_csv('oasis_longitudinal.csv')

df.head()


df = df.loc[df['Visit']==1]

df = df.reset_index(drop=True)

df['M/F'] = df['M/F'].replace(['F','M'], [0,1])

df['Group'] = df['Group'].replace(['Converted'], ['Demented'])

df['Group'] = df['Group'].replace(['Demented', 'Nondemented'], [1,0])

df = df.drop(['MRI ID', 'Visit', 'Hand'], axis=1)


def bar_chart(feature):

    Demented = df[df['Group']==1][feature].value_counts()

    Nondemented = df[df['Group']==0][feature].value_counts()

    df_bar = pd.DataFrame([Demented,Nondemented])

    df_bar.index = ['Demented','Nondemented']

    df_bar.plot(kind='bar',stacked=True, figsize=(8,5))
```

```
bar_chart('M/F')
plt.xlabel('Group')
plt.ylabel('Number of patients')
plt.legend()
plt.title('Gender and Demented rate')

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'MMSE',shade= True)
facet.set(xlim=(0, df['MMSE'].max()))
facet.add_legend()
plt.xlim(15.30)

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'ASF',shade= True)
facet.set(xlim=(0, df['ASF'].max()))
facet.add_legend()
plt.xlim(0.5, 2)

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'eTIV',shade= True)
facet.set(xlim=(0, df['eTIV'].max()))
facet.add_legend()
plt.xlim(900, 2100)

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'nWBV',shade= True)
facet.set(xlim=(0, df['nWBV'].max()))
facet.add_legend()
plt.xlim(0.6,0.9)

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, df['Age'].max()))
facet.add_legend()
plt.xlim(50,100)

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'EDUC',shade= True)
facet.set(xlim=(df['EDUC'].min(), df['EDUC'].max()))
facet.add_legend()
plt.ylim(0, 0.16)

pd.isnull(df).sum()

df_dropna = df.dropna(axis=0, how='any')
pd.isnull(df_dropna).sum()

df_dropna['Group'].value_counts()

x = df['EDUC']
y = df['SES']

ses_not_null_index = y[~y.isnull()].index
x = x[ses_not_null_index]
```

```
y = y[ses_not_null_index]

z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x, y, 'go', x, p(x), "r--")
plt.xlabel('Education Level(EDUC)')
plt.ylabel('Social Economic Status(SES)')

plt.show()

df.groupby(['EDUC'])['SES'].median()

df["SES"].fillna(df.groupby("EDUC")["SES"].transform("median"), inplace=True)

pd.isnull(df['SES']).value_counts()

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score

Y = df['Group'].values
X = df[['M/F', 'Age', 'EDUC', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF']]
X_trainval, X_test, Y_trainval, Y_test = train_test_split(X, Y, random_state=0)
```

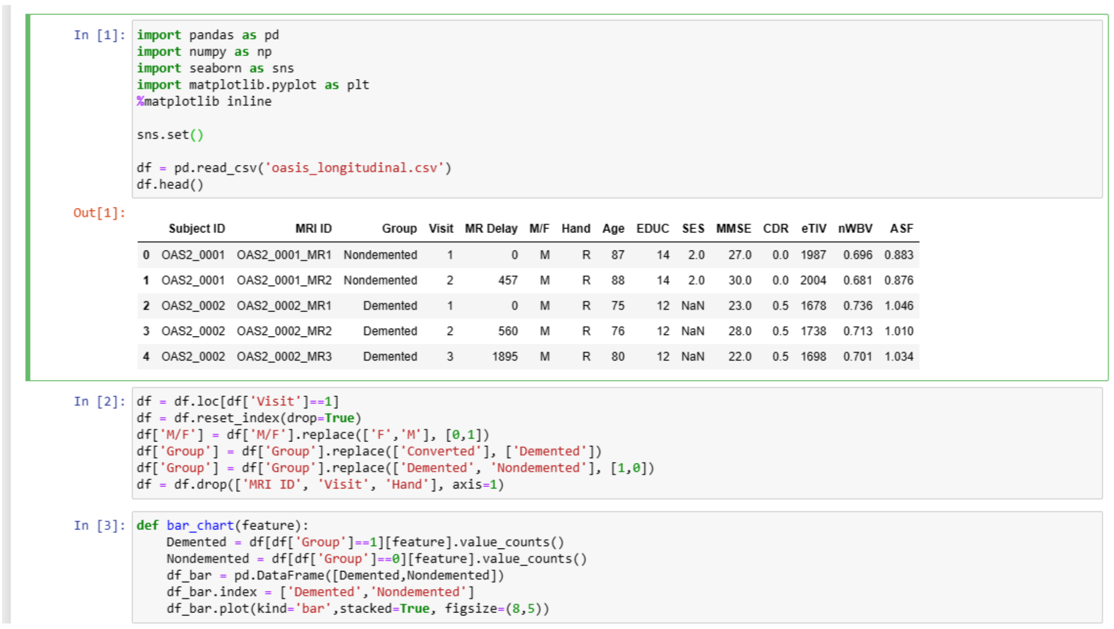
```
scaler = MinMaxScaler().fit(X_trainval)
X_trainval_scaled = scaler.transform(X_trainval)
X_test_scaled = scaler.transform(X_test)

Y = df_dropna['Group'].values
X = df_dropna[['M/F', 'Age', 'EDUC', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF']]
X_trainval_dna, X_test_dna, Y_trainval_dna, Y_test_dna = train_test_split(X, Y, random_state=0)

scaler = MinMaxScaler().fit(X_trainval_dna)
X_trainval_scaled_dna = scaler.transform(X_trainval_dna)
X_test_scaled_dna = scaler.transform(X_test_dna)

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, roc_curve, auc

acc = []

best_score=0
kfolds=5

for c in [0.001, 0.1, 1, 10, 100]:
    logRegModel = LogisticRegression(C=c)
    scores = cross_val_score(logRegModel, X_trainval, Y_trainval, cv=kfolds, scoring='accuracy')
    score = np.mean(scores)
    if score > best_score:
        best_score = score
        best_parameters = c

SelectedLogRegModel = LogisticRegression(C=best_parameters).fit(X_trainval_scaled, Y_trainval)

test_score = SelectedLogRegModel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedLogRegModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for regularization (C) is: ", best_parameters)
print("Test accuracy with best C parameter is", test_score)
print("Test recall with the best C parameter is", test_recall)
print("Test AUC with the best C parameter is", test_auc)
m = 'Logistic Regression (w/ imputation)'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])

best_score=0
kfolds=5

for c in [0.001, 0.1, 1, 10, 100]:
    logRegModel = LogisticRegression(C=c)
    scores = cross_val_score(logRegModel, X_trainval_scaled_dna, Y_trainval_dna, cv=kfolds,
scoring='accuracy')

    score = np.mean(scores)

    if score > best_score:
        best_score = score
        best_parameters = c

SelectedLogRegModel = LogisticRegression(C=best_parameters).fit(X_trainval_scaled_dna,
Y_trainval_dna)

test_score = SelectedLogRegModel.score(X_test_scaled_dna, Y_test_dna)
PredictedOutput = SelectedLogRegModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for regularization (C) is: ", best_parameters)
print("Test accuracy with best C parameter is", test_score)
print("Test recall with the best C parameter is", test_recall)
print("Test AUC with the best C parameter is", test_auc)

m = 'Logistic Regression (w/ dropna)'
acc.append([m, test_score, test_recall, test_recall, fpr, tpr, thresholds])

best_score = 0
```

```
for c_paramter in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
    for gamma_paramter in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
        for k_parameter in ['rbf', 'linear', 'poly', 'sigmoid']:
            svmModel = SVC(kernel=k_parameter, C=c_paramter, gamma=gamma_paramter)
            scores = cross_val_score(svmModel, X_trainval_scaled, Y_trainval, cv=kfolds,
scoring='accuracy')

            score = np.mean(scores)

            if score > best_score:
                best_score = score
                best_parameter_c = c_paramter
                best_parameter_gamma = gamma_paramter
                best_parameter_k = k_parameter



SelectedSVMmodel = SVC(C=best_parameter_c, gamma=best_parameter_gamma,
kernel=best_parameter_k).fit(X_trainval_scaled, Y_trainval)

test_score = SelectedSVMmodel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedSVMmodel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on cross validation set is:", best_score)
print("Best parameter for c is: ", best_parameter_c)
print("Best parameter for gamma is: ", best_parameter_gamma)
print("Best parameter for kernel is: ", best_parameter_k)

print("Test accuracy with the best parameters is", test_score)

print("Test recall with the best parameters is", test_recall)

print("Test recall with the best parameter is", test_auc)


m = 'SVM'

acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])


from sklearn.tree import DecisionTreeClassifier


best_score = 0


for md in range(1, 9):


  treeModel = DecisionTreeClassifier(random_state=0, max_depth=md, criterion='gini')


  scores = cross_val_score(treeModel, X_trainval_scaled, Y_trainval, cv=kfolds,
scoring='accuracy')


  score = np.mean(scores)


  if score > best_score:

    best_score = score

    best_parameter = md


SelectedDTModel = DecisionTreeClassifier(max_depth=best_parameter).fit(X_trainval_scaled,
Y_trainval )
```

```
test_score = SelectedDTModel.score(X_test_scaled, Y_test)

PredictedOutput = SelectedDTModel.predict(X_test_scaled)

test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)

fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)

test_auc = auc(fpr, tpr)

print("Best accuracy on validation set is:", best_score)

print("Best parameter for the maximum depth is: ", best_parameter)

print("Test accuracy with best parameter is ", test_score)

print("Test recall with best parameters is ", test_recall)

print("Test AUC with the best parameter is ", test_auc)


m = 'Decision Tree'

acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])


print("Feature importance: ")

np.array([X.columns.values.tolist(), list(SelectedDTModel.feature_importances_)]).T


from sklearn.tree import export_graphviz

import graphviz

dot_data=export_graphviz(SelectedDTModel,
feature_names=X_trainval.columns.values.tolist(),out_file=None)

graph = graphviz.Source(dot_data)

graph


best_score = 0


for M in range(2, 15, 2):
    for d in range(1, 9):
        for m in range(1, 9):

            forestModel = RandomForestClassifier(n_estimators=M, max_features=d, n_jobs=4,
                                    max_depth=m, random_state=0)

            scores = cross_val_score(forestModel, X_trainval_scaled, Y_trainval, cv=kfolds,
scoring='accuracy')

            score = np.mean(scores)

            if score > best_score:
                best_score = score
                best_M = M
                best_d = d
                best_m = m

SelectedRFModel = RandomForestClassifier(n_estimators=M, max_features=d,
                                    max_depth=m, random_state=0).fit(X_trainval_scaled, Y_trainval )

PredictedOutput = SelectedRFModel.predict(X_test_scaled)
test_score = SelectedRFModel.score(X_test_scaled, Y_test)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameters of M, d, m are: ", best_M, best_d, best_m)
print("Test accuracy with the best parameters is", test_score)
print("Test recall with the best parameters is:", test_recall)
print("Test AUC with the best parameters is:", test_auc)

m = 'Random Forest'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])

print("Feature importance: ")
np.array([X.columns.values.tolist(), list(SelectedRFModel.feature_importances_)]).T

best_score = 0

for M in range(2, 15, 2):
    for lr in [0.0001, 0.001, 0.01, 0.1, 1]:
        # train the model
        boostModel = AdaBoostClassifier(n_estimators=M, learning_rate=lr, random_state=0)

        scores = cross_val_score(boostModel, X_trainval_scaled, Y_trainval, cv=kfolds,
scoring='accuracy')

        score = np.mean(scores)

        if score > best_score:
            best_score = score
            best_M = M
            best_lr = lr
```

```
SelectedBoostModel = AdaBoostClassifier(n_estimators=M, learning_rate=lr,
random_state=0).fit(X_trainval_scaled, Y_trainval )


PredictedOutput = SelectedBoostModel.predict(X_test_scaled)
test_score = SelectedRFModel.score(X_test_scaled, Y_test)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter of M is: ", best_M)
print("best parameter of LR is: ", best_lr)
print("Test accuracy with the best parameter is", test_score)
print("Test recall with the best parameters is:", test_recall)
print("Test AUC with the best parameters is:", test_auc)


m = 'AdaBoost'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])


print("Feature importance: ")
np.array([X.columns.values.tolist(), list(SelectedBoostModel.feature_importances_)]).T


result = pd.DataFrame(acc, columns=['Model', 'Accuracy', 'Recall', 'AUC', 'FPR', 'TPR', 'TH'])
result[['Model', 'Accuracy', 'Recall', 'AUC']]
```

Output



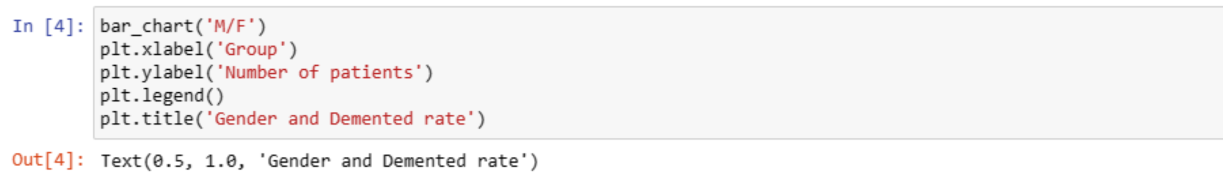
```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline

        sns.set()

        df = pd.read_csv('oasis_longitudinal.csv')
        df.head()
```

Out[1]:

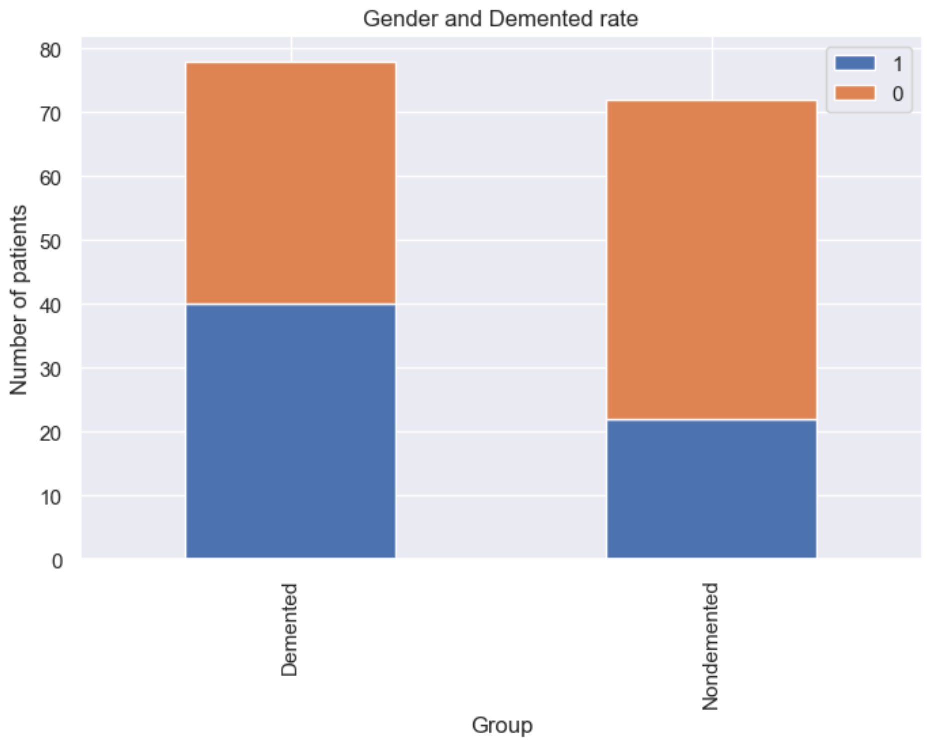
	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	2.0	27.0	0.0	1987	0.696	0.883
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	2.0	30.0	0.0	2004	0.681	0.876
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	NaN	23.0	0.5	1678	0.736	1.046
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	NaN	28.0	0.5	1738	0.713	1.010
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	NaN	22.0	0.5	1698	0.701	1.034

```
In [2]: df = df.loc[df['Visit']==1]
        df = df.reset_index(drop=True)
        df['M/F'] = df['M/F'].replace(['F','M'], [0,1])
        df['Group'] = df['Group'].replace(['Converted'], ['Demented'])
        df['Group'] = df['Group'].replace(['Demented', 'Nondemented'], [1,0])
        df = df.drop(['MRI ID', 'Visit', 'Hand'], axis=1)
```

```
In [3]: def bar_chart(feature):
            Demented = df[df['Group']==1][feature].value_counts()
            Nondemented = df[df['Group']==0][feature].value_counts()
            df_bar = pd.DataFrame([Demented,Nondemented])
            df_bar.index = ['Demented','Nondemented']
            df_bar.plot(kind='bar',stacked=True, figsize=(8,5))
```

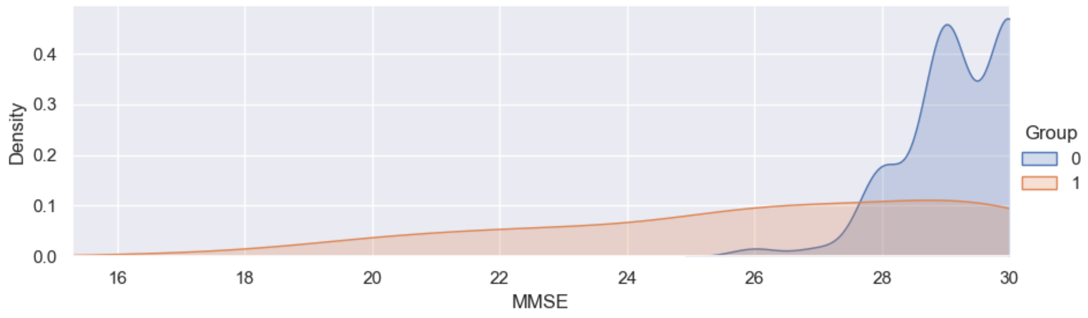


```
In [4]: bar_chart('M/F')
        plt.xlabel('Group')
        plt.ylabel('Number of patients')
        plt.legend()
        plt.title('Gender and Demented rate')

Out[4]: Text(0.5, 1.0, 'Gender and Demented rate')
```



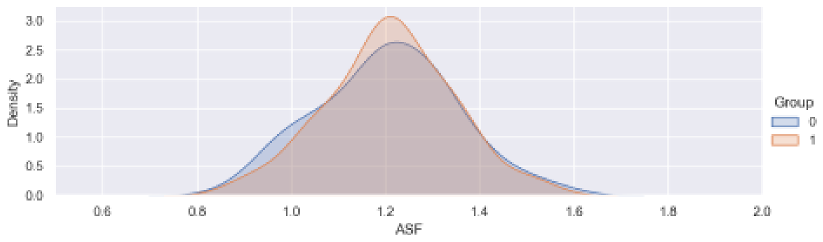
In [5]:
```
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'MMSE',shade= True)
facet.set(xlim=(0, df['MMSE'].max()))
facet.add_legend()
plt.xlim(15.30)
```

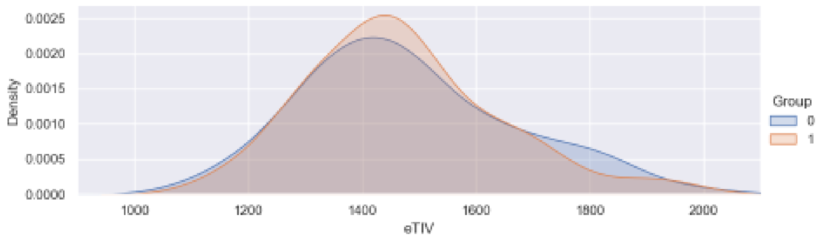
Out[5]:
```
(15.3, 30.0)
```

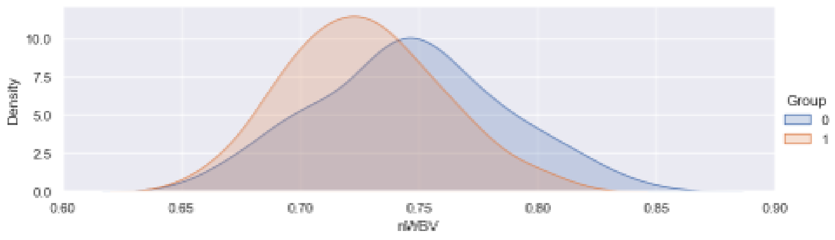


In [6]:
```
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'ASF',shade= True)
facet.set(xlim=(0, df['ASF'].max()))
facet.add_legend()
plt.xlim(0.5, 2)

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'eTIV',shade= True)
facet.set(xlim=(0, df['eTIV'].max()))
facet.add_legend()
plt.xlim(900, 2100)

facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'nWBV',shade= True)
facet.set(xlim=(0, df['nWBV'].max()))
facet.add_legend()
plt.xlim(0.6,0.9)
```

Out[6]:
```
(0.6, 0.9)
```

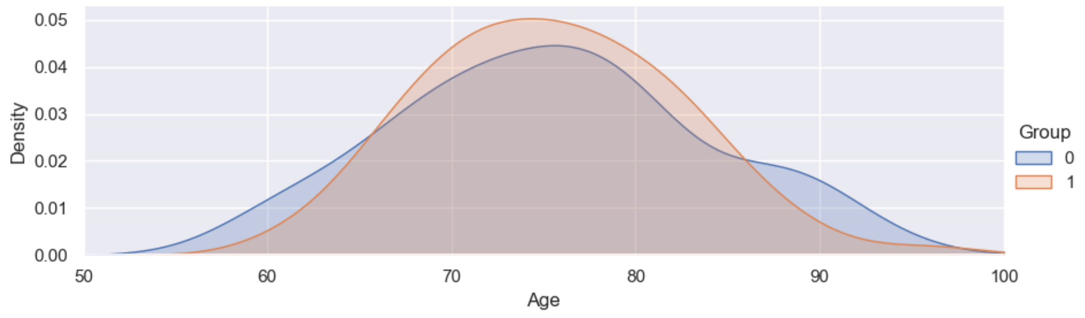






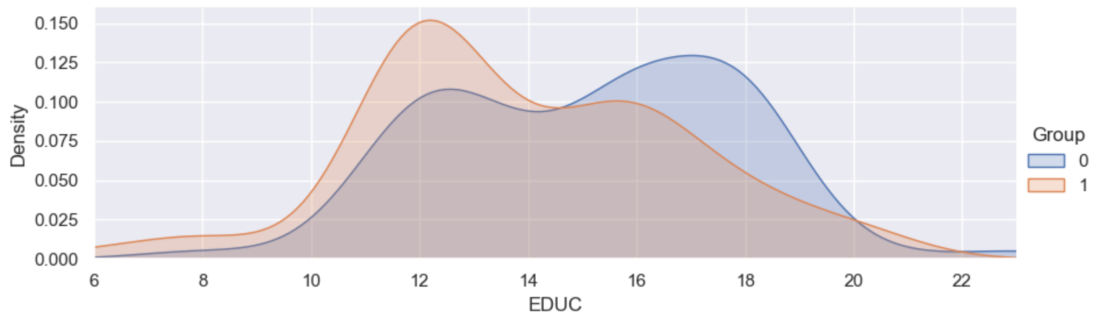
In [7]:
```
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, df['Age'].max()))
facet.add_legend()
plt.xlim(50,100)
```

Out[7]:
```
(50.0, 100.0)
```



In [8]:
```
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'EDUC',shade= True)
facet.set(xlim=(df['EDUC'].min(), df['EDUC'].max()))
facet.add_legend()
plt.ylim(0, 0.16)
```

Out[8]:
```
(0.0, 0.16)
```



In [9]:
```
pd.isnull(df).sum()
```

Out[9]:
```
Subject ID    0
Group         0
MR Delay      0
M/F           0
Age           0
EDUC          0
SES           8
MMSE          0
CDR           0
eTIV          0
nWBV          0
ASF           0
dtype: int64
```

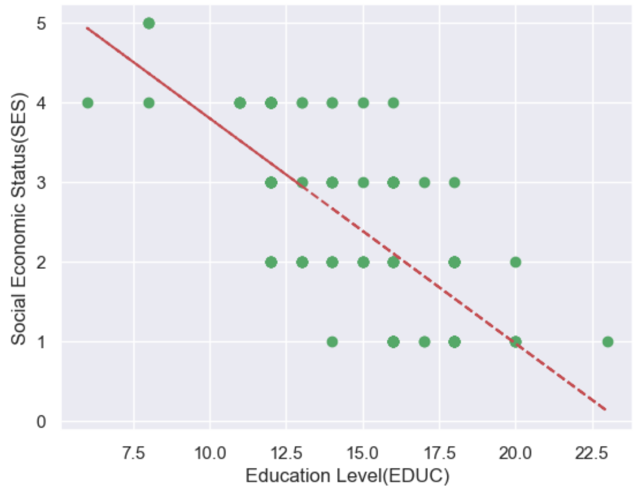
```
In [10]: df_dropna = df.dropna(axis=0, how='any')
         pd.isnull(df_dropna).sum()
```

```
Out[10]: Subject ID    0
         Group         0
         MR Delay      0
         M/F           0
         Age           0
         EDUC          0
         SES           0
         MMSE          0
         CDR           0
         eTIV          0
         nWBV          0
         ASF           0
         dtype: int64
```

```
In [11]: df_dropna['Group'].value_counts()
```

```
Out[11]: 0    72
         1    70
         Name: Group, dtype: int64
```

```
In [12]: x = df['EDUC']
         y = df['SES']

         ses_not_null_index = y[~y.isnull()].index
         x = x[ses_not_null_index]
         y = y[ses_not_null_index]

         z = np.polyfit(x, y, 1)
         p = np.poly1d(z)
         plt.plot(x, y, 'go', x, p(x), "r--")
         plt.xlabel('Education Level(EDUC)')
         plt.ylabel('Social Economic Status(SES)')

         plt.show()
```



In [13]:
```
df.groupby(['EDUC'])['SES'].median()
```
Out[13]:
```
EDUC
6     4.0
8     5.0
11    4.0
12    3.0
13    2.0
14    3.0
15    2.0
16    2.0
17    1.0
18    2.0
20    1.0
23    1.0
Name: SES, dtype: float64
```
In [14]:
```
df["SES"].fillna(df.groupby("EDUC")["SES"].transform("median"), inplace=True)
```
In [15]:
```
pd.isnull(df['SES']).value_counts()
```
Out[15]:
```
False    150
Name: SES, dtype: int64
```
In [16]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
```
In [17]:
```
Y = df['Group'].values
X = df[['M/F', 'Age', 'EDUC', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF']]
X_trainval, X_test, Y_trainval, Y_test = train_test_split(X, Y, random_state=0)

scaler = MinMaxScaler().fit(X_trainval)
X_trainval_scaled = scaler.transform(X_trainval)
X_test_scaled = scaler.transform(X_test)
```
In [18]:
```
Y = df_dropna['Group'].values
X = df_dropna[['M/F', 'Age', 'EDUC', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF']]
X_trainval_dna, X_test_dna, Y_trainval_dna, Y_test_dna = train_test_split(X, Y, random_state=0)

scaler = MinMaxScaler().fit(X_trainval_dna)
X_trainval_scaled_dna = scaler.transform(X_trainval_dna)
X_test_scaled_dna = scaler.transform(X_test_dna)
```
In [19]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, roc_curve, auc
```
In [20]:
```
acc = []
```

In [21]:
```
best_score=0
kfolds=5

for c in [0.001, 0.1, 1, 10, 100]:
    logRegModel = LogisticRegression(C=c)

    scores = cross_val_score(logRegModel, X_trainval, Y_trainval, cv=kfolds, scoring='accuracy')
    score = np.mean(scores)
    if score > best_score:
        best_score = score
        best_parameters = c

SelectedLogRegModel = LogisticRegression(C=best_parameters).fit(X_trainval_scaled, Y_trainval)

test_score = SelectedLogRegModel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedLogRegModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for regularization (C) is: ", best_parameters)
print("Test accuracy with best C parameter is", test_score)
print("Test recall with the best C parameter is", test_recall)
print("Test AUC with the best C parameter is", test_auc)
m = 'Logistic Regression (w/ imputation)'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Best accuracy on validation set is: 0.724901185770751
Best parameter for regularization (C) is:  10
Test accuracy with best C parameter is 0.7631578947368421
Test recall with the best C parameter is 0.7
Test AUC with the best C parameter is 0.7666666666666667
```

In [22]:
```
best_score=0
kfolds=5

for c in [0.001, 0.1, 1, 10, 100]:
    logRegModel = LogisticRegression(C=c)

    scores = cross_val_score(logRegModel, X_trainval_scaled_dna, Y_trainval_dna, cv=kfolds, scoring='accuracy')

    score = np.mean(scores)

    if score > best_score:
        best_score = score
        best_parameters = c

SelectedLogRegModel = LogisticRegression(C=best_parameters).fit(X_trainval_scaled_dna, Y_trainval_dna)

test_score = SelectedLogRegModel.score(X_test_scaled_dna, Y_test_dna)
PredictedOutput = SelectedLogRegModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for regularization (C) is: ", best_parameters)
print("Test accuracy with best C parameter is", test_score)
print("Test recall with the best C parameter is", test_recall)
print("Test AUC with the best C parameter is", test_auc)

m = 'Logistic Regression (w/ dropna)'
acc.append([m, test_score, test_recall, test_recall, fpr, tpr, thresholds])
```

```
Best accuracy on validation set is: 0.725974025974026
Best parameter for regularization (C) is:  10
Test accuracy with best C parameter is 0.8055555555555556
Test recall with the best C parameter is 0.75
Test AUC with the best C parameter is 0.8194444444444443
```

In [23]:
```
best_score = 0

for c_paramter in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
    for gamma_paramter in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
        for k_parameter in ['rbf', 'linear', 'poly', 'sigmoid']:
            svmModel = SVC(kernel=k_parameter, C=c_paramter, gamma=gamma_paramter)
            scores = cross_val_score(svmModel, X_trainval_scaled, Y_trainval, cv=kfolds, scoring='accuracy')

            score = np.mean(scores)

            if score > best_score:
                best_score = score
                best_parameter_c = c_paramter
                best_parameter_gamma = gamma_paramter
                best_parameter_k = k_parameter



SelectedSVMmodel = SVC(C=best_parameter_c, gamma=best_parameter_gamma, kernel=best_parameter_k).fit(X_trainval_scaled, Y_trainval

test_score = SelectedSVMmodel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedSVMmodel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on cross validation set is:", best_score)
print("Best parameter for c is: ", best_parameter_c)
print("Best parameter for gamma is: ", best_parameter_gamma)
print("Best parameter for kernel is: ", best_parameter_k)
print("Test accuracy with the best parameters is", test_score)
print("Test recall with the best parameters is", test_recall)
print("Test recall with the best parameter is", test_auc)

m = 'SVM'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
```

```
Best accuracy on cross validation set is: 0.7687747035573123
Best parameter for c is:  100
Best parameter for gamma is:  0.1
Best parameter for kernel is:  rbf
Test accuracy with the best parameters is 0.8157894736842105
Test recall with the best parameters is 0.7
Test recall with the best parameter is 0.8222222222222222
```

In [24]:
```
from sklearn.tree import DecisionTreeClassifier

best_score = 0

for md in range(1, 9):

    treeModel = DecisionTreeClassifier(random_state=0, max_depth=md, criterion='gini')

    scores = cross_val_score(treeModel, X_trainval_scaled, Y_trainval, cv=kfolds, scoring='accuracy')

    score = np.mean(scores)

    if score > best_score:
        best_score = score
        best_parameter = md

SelectedDTModel = DecisionTreeClassifier(max_depth=best_parameter).fit(X_trainval_scaled, Y_trainval )

test_score = SelectedDTModel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedDTModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for the maximum depth is: ", best_parameter)
print("Test accuracy with best parameter is ", test_score)
print("Test recall with best parameters is ", test_recall)
print("Test AUC with the best parameter is ", test_auc)

m = 'Decision Tree'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
```

```
Best accuracy on validation set is: 0.7770750988142293
Best parameter for the maximum depth is:  1
Test accuracy with best parameter is  0.8157894736842105
Test recall with best parameters is  0.65
Test AUC with the best parameter is  0.825
```

```
In [25]: print("Feature importance: ")
         np.array([X.columns.values.tolist(), list(SelectedDTModel.feature_importances_)]).T
```

```
Feature importance:

Out[25]: array([['M/F', '0.0'],
                ['Age', '0.0'],
                ['EDUC', '0.0'],
                ['SES', '0.0'],
                ['MMSE', '1.0'],
                ['eTIV', '0.0'],
                ['nWBV', '0.0'],
                ['ASF', '0.0']], dtype='<U32')
```

```
In [27]: from sklearn.tree import export_graphviz
         import graphviz
         dot_data=export_graphviz(SelectedDTModel, feature_names=X_trainval.columns.values.tolist(),out_file=None)
         graph = graphviz.Source(dot_data)
         graph
```

```
In [28]: best_score = 0

         for M in range(2, 15, 2):
             for d in range(1, 9):
                 for m in range(1, 9):

                     forestModel = RandomForestClassifier(n_estimators=M, max_features=d, n_jobs=4,
                                                          max_depth=m, random_state=0)

                     scores = cross_val_score(forestModel, X_trainval_scaled, Y_trainval, cv=kfolds, scoring='accuracy')

                     score = np.mean(scores)

                     if score > best_score:
                         best_score = score
                         best_M = M
                         best_d = d
                         best_m = m

         SelectedRFModel = RandomForestClassifier(n_estimators=M, max_features=d,
                                                  max_depth=m, random_state=0).fit(X_trainval_scaled, Y_trainval )

         PredictedOutput = SelectedRFModel.predict(X_test_scaled)
         test_score = SelectedRFModel.score(X_test_scaled, Y_test)
         test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
         fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
         test_auc = auc(fpr, tpr)
         print("Best accuracy on validation set is:", best_score)
         print("Best parameters of M, d, m are: ", best_M, best_d, best_m)
         print("Test accuracy with the best parameters is", test_score)
         print("Test recall with the best parameters is:", test_recall)
         print("Test AUC with the best parameters is:", test_auc)

         m = 'Random Forest'
         acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
```

```
Best accuracy on validation set is: 0.8035573122529645
Best parameters of M, d, m are:  2 5 7
Test accuracy with the best parameters is 0.868421052631579
Test recall with the best parameters is: 0.8
Test AUC with the best parameters is: 0.8722222222222222
```

```
In [29]: print("Feature importance: ")
         np.array([X.columns.values.tolist(), list(SelectedRFModel.feature_importances_)]).T
```

```
Feature importance:

Out[29]: array([['M/F', '0.03503132427481025'],
                ['Age', '0.09551237125526228'],
                ['EDUC', '0.06261556797214127'],
                ['SES', '0.060620327518549066'],
                ['MMSE', '0.4006565962793097'],
                ['eTIV', '0.07005497528287095'],
                ['nWBV', '0.1460571117936201'],
                ['ASF', '0.1294517256234364']], dtype='<U32')
```

In [30]:
```
best_score = 0

for M in range(2, 15, 2):
    for lr in [0.0001, 0.001, 0.01, 0.1, 1]:
        # train the model
        boostModel = AdaBoostClassifier(n_estimators=M, learning_rate=lr, random_state=0)

        scores = cross_val_score(boostModel, X_trainval_scaled, Y_trainval, cv=kfolds, scoring='accuracy')

        score = np.mean(scores)

        if score > best_score:
            best_score = score
            best_M = M
            best_lr = lr

SelectedBoostModel = AdaBoostClassifier(n_estimators=M, learning_rate=lr, random_state=0).fit(X_trainval_scaled, Y_trainval )

PredictedOutput = SelectedBoostModel.predict(X_test_scaled)
test_score = SelectedRFModel.score(X_test_scaled, Y_test)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter of M is: ", best_M)
print("best parameter of LR is: ", best_lr)
print("Test accuracy with the best parameter is", test_score)
print("Test recall with the best parameters is:", test_recall)
print("Test AUC with the best parameters is:", test_auc)

m = 'AdaBoost'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
```

```
Best accuracy on validation set is: 0.7770750988142293
Best parameter of M is:  2
best parameter of LR is:  0.0001
Test accuracy with the best parameter is 0.868421052631579
Test recall with the best parameters is: 0.65
Test AUC with the best parameters is: 0.825
```

In [31]:
```
print("Feature importance: ")
np.array([X.columns.values.tolist(), list(SelectedBoostModel.feature_importances_)]).T
```

```
Feature importance:
```

Out[31]:
```
array([['M/F', '0.07142857142857142'],
       ['Age', '0.14285714285714285'],
       ['EDUC', '0.21428571428571427'],
       ['SES', '0.07142857142857142'],
       ['MMSE', '0.14285714285714285'],
       ['eTIV', '0.21428571428571427'],
       ['nWBV', '0.14285714285714285'],
       ['ASF', '0.0']], dtype='<U32')
```

In [32]:
```
result = pd.DataFrame(acc, columns=['Model', 'Accuracy', 'Recall', 'AUC', 'FPR', 'TPR', 'TH'])
result[['Model', 'Accuracy', 'Recall', 'AUC']]
```

Out[32]:

	Model	Accuracy	Recall	AUC
0	Logistic Regression (w/ imputation)	0.763158	0.70	0.766667
1	Logistic Regression (w/ dropna)	0.805556	0.75	0.750000
2	SVM	0.815789	0.70	0.822222
3	Decision Tree	0.815789	0.65	0.825000
4	Random Forest	0.868421	0.80	0.872222
5	AdaBoost	0.868421	0.65	0.825000

Github Links

Project:

Vaishnav2535/Detecting-Early-Alzheimer-s-using-Semi---Supervised-Learning (github.com)

Code:

https://github.com/Vaishnav2535/Detecting-Early-Alzheimer-s-using-Semi---Supervised-Learning/blob/fa2dab865fd5762280428e1bfaccfb74a5ee23dd/Detecting%20Early%20Alzheimers.ipynb

Detecting Early Alzheimers.ipynb