CS109A Project Milestone 4: Models

Group 29

- Tatyana Zyabkina
- Joanna Guo
- John Liang
- Ray Ortigas

Our models are partial reproductions of the models described in the Ritter et al. 2015 paper, ["Multimodal prediction of conversion to Alzheimer's disease based on incomplete biomarkers" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4877756/?report=reader)](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4877756/?report=reader), with some adjustments based on experimentation.

Using the ADNIMERGE data set along with features from other ADNI datasets, which based on our research appeared likely to be predictive, we used the following process:

- Select analysis cohort, participants who completed a 36-month or later follow-up
- Identify "converters" who transitioned from non-AD to AD diagnosis sometime in the first 36 months of their study participation
- Impute missing values with mean
- Dummy-encode and scale features
- Fit several models and evaluate their predictive powers

Styling

In [1]:

```
#RUN THIS CELL
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/master/content/styles/c
HTML(styles)
```

Out[1]:

Imports

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import scipy
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
import itertools
from sklearn.base import clone
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

In [4]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
```

In [5]:
```
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
```

In [6]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegressionCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

Loading

ADNIMERGE

In [7]:
```
adnimerge_df = pd.read_csv("../data/ADNIMERGE.csv")
```

In [8]:
```
adnimerge_df.shape
```

Out[8]: (13017, 94)

In [9]:
```
adnimerge_df.head()
```

Out[9]:

	RID	PTID	VISCODE	SITE	COLPROT	ORIGPROT	EXAMDATE	DX_bl	AGE	PTGENDER	...	EcogSPDivatt_bl	EcogSPTotal_bl	I
0	2	011_S_0002	bl	11	ADNI1	ADNI1	2005-09-08	CN	74.3	Male	...	NaN	NaN	1
1	3	011_S_0003	bl	11	ADNI1	ADNI1	2005-09-12	AD	81.3	Male	...	NaN	NaN	1
2	3	011_S_0003	m06	11	ADNI1	ADNI1	2006-03-13	AD	81.3	Male	...	NaN	NaN	1
3	3	011_S_0003	m12	11	ADNI1	ADNI1	2006-09-12	AD	81.3	Male	...	NaN	NaN	1
4	3	011_S_0003	m24	11	ADNI1	ADNI1	2007-09-12	AD	81.3	Male	...	NaN	NaN	1

5 rows × 94 columns

In [10]:
```
adnimerge_df.dtypes
```

Out[10]:
```
RID                       int64
PTID                     object
VISCODE                  object
SITE                      int64
COLPROT                  object
ORIGPROT                 object
EXAMDATE                 object
DX_bl                    object
AGE                     float64
PTGENDER                 object
PTEDUCAT                  int64
PTETHCAT                 object
PTRACCAT                 object
PTMARRY                  object
APOE4                   float64
FDG                     float64
PIB                     float64
AV45                    float64
CDRSB                   float64
ADAS11                  float64
ADAS13                  float64
MMSE                    float64
RAVLT_immediate         float64
RAVLT_learning          float64
RAVLT_forgetting        float64
RAVLT_perc_forgetting   float64
FAQ                     float64
MOCA                    float64
EcogPtMem               float64
EcogPtLang              float64
                         ...
Ventricles_bl           float64
Hippocampus_bl          float64
WholeBrain_bl           float64
Entorhinal_bl           float64
Fusiform_bl             float64
MidTemp_bl              float64
ICV_bl                  float64
MOCA_bl                 float64
EcogPtMem_bl            float64
EcogPtLang_bl           float64
EcogPtVisspat_bl        float64
EcogPtPlan_bl           float64
EcogPtOrgan_bl          float64
EcogPtDivatt_bl         float64
EcogPtTotal_bl          float64
EcogSPMem_bl            float64
EcogSPLang_bl           float64
EcogSPVisspat_bl        float64
EcogSPPlan_bl           float64
EcogSPOrgan_bl          float64
EcogSPDivatt_bl         float64
EcogSPTotal_bl          float64
FDG_bl                  float64
PIB_bl                  float64
AV45_bl                 float64
Years_bl                float64
Month_bl                float64
Month                     int64
M                         int64
update_stamp             object
Length: 94, dtype: object
```

In [11]:
```
adnimerge_df.describe()
```

Out[11]:

	RID	SITE	AGE	PTEDUCAT	APOE4	FDG	PIB	AV45	CDRSB	ADA
count	13017.000000	13017.000000	13017.000000	13017.000000	12958.000000	3353.000000	223.000000	2161.000000	9016.000000	8959.00(
mean	2285.200584	73.892064	73.767220	15.994930	0.535654	1.208225	1.783161	1.195504	2.163598	11.398
std	1871.013213	110.533877	6.979685	2.824862	0.655480	0.160972	0.422511	0.227999	2.805879	8.616
min	2.000000	2.000000	54.400000	4.000000	0.000000	0.636804	1.095000	0.814555	0.000000	0.00(
25%	631.000000	21.000000	69.500000	14.000000	0.000000	1.109730	1.361250	1.010140	0.000000	5.33(
50%	1301.000000	41.000000	73.700000	16.000000	0.000000	1.219870	1.850000	1.114670	1.000000	9.00(
75%	4353.000000	116.000000	78.600000	18.000000	1.000000	1.314320	2.127500	1.364980	3.000000	15.00(
max	6094.000000	941.000000	91.400000	20.000000	2.000000	1.753320	2.927500	2.669210	18.000000	70.00(

8 rows × 77 columns

We preface our models with a short EDA on ADNIMERGE, related to Section 2.1.1 of the Ritter paper:

> For this study, patients with a baseline diagnosis of MCI and a follow-up time of at least 36 months were extracted from the ADNI database. Patients who were diagnosed with MCI, NL or MCI to NL at all visits during the 3-year follow-up were included in the MCI-stable group, whereas patients whose diagnosis changed to AD during the 3-year follow-up were regarded as MCI-converters. After this procedure, 237 patients were selected, 151 of which belonged to the MCI-stable group, and 86 to the MCI-converter group (see Table 1).

`DX` gives the diagnosis at any particular visit:

In [12]:
```
adnimerge_df["DX"].unique()
```

Out[12]:
```
array(['CN', 'Dementia', 'MCI', nan], dtype=object)
```

`DX_bl` gives the diagnosis at baseline:

In [13]:
```
adnimerge_df["DX_bl"].unique()
```

Out[13]:
```
array(['CN', 'AD', 'LMCI', 'SMC', 'EMCI', nan], dtype=object)
```

The codes differ slightly from the `DX` codes, but for the purposes of this milestone, we will consider Dementia and AD to be synonymous.

`VISCODE` in `ADNIMERGE` is the cleaned up visit code:

In [14]:
```
adnimerge_df["VISCODE"].unique()
```

Out[14]:
```
array(['bl', 'm06', 'm12', 'm24', 'm18', 'm36', 'm48', 'm60', 'm03',
       'm30', 'm42', 'm72', 'm54', 'm66', 'm78', 'm108', 'm96', 'm84',
       'm90', 'm120', 'm114', 'm102', 'm126'], dtype=object)
```

`bl` indicates baseline, and then there are varying month markers. These have been cleaned and extracted with `Month` :

In [15]:
```
HTML(adnimerge_df["Month"].value_counts().to_frame().reset_index().sort_values("index").to_html(index=Fa
```

Out[15]:

index	Month
0	1784
3	795
6	1613
12	1472
18	1299
24	1301
30	768
36	823
42	332
48	658
54	224
60	398
66	295
72	305
78	238
84	207
90	137
96	142
102	19
108	102
114	15
120	79
126	6
132	5

Features from RNA Microarray EDA

In [16]:
```
adni_features_from_rna_microarray_eda_df = pd.read_csv("../data/features-from-rna-microarray-eda.csv")
```

In [17]:
```
adni_features_from_rna_microarray_eda_df.shape
```

Out[17]:
```
(744, 14)
```

In [18]:
```
adni_features_from_rna_microarray_eda_df.head()
```

Out[18]:

	RID	3308	4203	4381	11484	13711	16478	16679	31790	32056	38066	9989	21323	49186
0	1249	10.648	10.202	11.216	10.619	5.682	7.026	8.080	6.856	4.907	8.934	1.766	1.980	2.092
1	4410	10.480	9.824	10.870	9.791	5.320	5.189	8.083	6.173	4.966	8.578	2.230	2.030	2.296
2	4153	10.801	10.381	11.067	10.474	5.108	7.004	7.939	6.727	4.233	8.880	1.765	2.276	2.066
3	1232	10.822	10.306	10.985	10.177	4.736	7.172	7.634	6.895	4.481	8.623	1.915	2.302	2.079
4	4205	10.670	10.264	10.596	10.292	4.667	7.070	8.038	6.871	4.795	8.667	1.842	2.325	2.381

In [19]:
```
adni_features_from_rna_microarray_eda_df.dtypes
```

Out[19]:
```
RID        int64
3308     float64
4203     float64
4381     float64
11484    float64
13711    float64
16478    float64
16679    float64
31790    float64
32056    float64
38066    float64
9989     float64
21323    float64
49186    float64
dtype: object
```

In [20]:
```
adni_features_from_rna_microarray_eda_df.describe()
```

Out[20]:

	RID	3308	4203	4381	11484	13711	16478	16679	31790	32056	380
count	744.000000	744.000000	744.000000	744.000000	744.000000	744.000000	744.000000	744.000000	744.000000	744.000000	744.0000
mean	2741.111559	10.768437	10.336457	10.974757	10.394950	5.620620	6.511915	8.058327	6.787516	4.676445	8.7546
std	1691.997650	0.145510	0.319776	0.266156	0.275946	0.592729	0.722024	0.508518	0.457811	0.508287	0.3615
min	2.000000	10.268000	8.796000	9.987000	9.317000	3.653000	2.974000	6.393000	4.768000	3.222000	7.5210
25%	1014.500000	10.670000	10.132000	10.802000	10.217750	5.263000	6.110500	7.738750	6.498000	4.328000	8.5157
50%	3204.000000	10.773500	10.355000	11.002500	10.410500	5.628000	6.578000	8.081500	6.786000	4.675000	8.7590
75%	4338.250000	10.869250	10.571000	11.160000	10.589250	6.012000	6.981250	8.397000	7.092250	5.027250	9.0042
max	4707.000000	11.239000	11.400000	11.601000	11.066000	7.634000	10.330000	9.735000	8.488000	6.403000	9.8690

Features from TOMM40 EDA

In [21]:
```
adni_features_from_tomm40_eda_df = pd.read_csv("../data/features-from-tomm40-eda.csv")
```

In [22]:
```
adni_features_from_tomm40_eda_df.shape
```

Out[22]:
```
(757, 3)
```

In [23]:
```
adni_features_from_tomm40_eda_df.head()
```

Out[23]:

	RID	TOMM40_A1	TOMM40_A2
0	295	16.0	21.0
1	413	16.0	34.0
2	559	35.0	35.0
3	619	28.0	28.0
4	685	33.0	34.0

In [24]:
```
adni_features_from_tomm40_eda_df.dtypes
```

Out[24]:
```
RID            int64
TOMM40_A1    float64
TOMM40_A2    float64
dtype: object
```

In [25]:
```
adni_features_from_tomm40_eda_df.describe()
```

Out[25]:

	RID	TOMM40_A1	TOMM40_A2
count	757.000000	746.000000	746.000000
mean	693.376486	21.132708	29.971850
std	415.087697	6.856522	6.457752
min	2.000000	14.000000	15.000000
25%	331.000000	16.000000	28.000000
50%	689.000000	16.000000	33.000000
75%	1051.000000	28.000000	34.000000
max	1435.000000	38.000000	51.000000

Combined ADNI Data

In [26]:
```
def combine_adni_data(
    adnimerge_df,
    adni_features_from_rna_microarray_eda_df,
    adni_features_from_tomm40_eda_df
):
    adni_df = adnimerge_df.copy()
    adni_df = pd.merge(
        adni_df,
        adni_features_from_rna_microarray_eda_df,
        how='outer',
        left_on='RID',
        right_on='RID'
    )
    adni_df = pd.merge(
        adni_df,
        adni_features_from_tomm40_eda_df,
        how='outer',
        left_on='RID',
        right_on='RID'
    )
    return adni_df
```

In [27]:
```
adni_df = combine_adni_data(
    adnimerge_df,
    adni_features_from_rna_microarray_eda_df,
    adni_features_from_tomm40_eda_df
)
```

In [28]:
```
adni_df.shape
```

Out[28]: (13017, 109)

In [29]:
```
adni_df.head()
```

Out[29]:

	RID	PTID	VISCODE	SITE	COLPROT	ORIGPROT	EXAMDATE	DX_bl	AGE	PTGENDER	...	16478	16679	31790	32056	3806(
0	2	011_S_0002	bl	11	ADNI1	ADNI1	2005-09-08	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
1	2	011_S_0002	m06	11	ADNI1	ADNI1	2006-03-06	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
2	2	011_S_0002	m36	11	ADNI1	ADNI1	2008-08-27	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
3	2	011_S_0002	m60	11	ADNIGO	ADNI1	2010-09-22	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
4	2	011_S_0002	m66	11	ADNIGO	ADNI1	2011-03-04	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191

5 rows × 109 columns

In [30]:
```
adni_df.dtypes
```

Out[30]:
```
RID                          int64
PTID                        object
VISCODE                     object
SITE                         int64
COLPROT                     object
ORIGPROT                    object
EXAMDATE                    object
DX_bl                       object
AGE                        float64
PTGENDER                    object
PTEDUCAT                     int64
PTETHCAT                    object
PTRACCAT                    object
PTMARRY                     object
APOE4                      float64
FDG                        float64
PIB                        float64
AV45                       float64
CDRSB                      float64
ADAS11                     float64
ADAS13                     float64
MMSE                       float64
RAVLT_immediate            float64
RAVLT_learning             float64
RAVLT_forgetting           float64
RAVLT_perc_forgetting      float64
FAQ                        float64
MOCA                       float64
EcogPtMem                  float64
EcogPtLang                 float64
                            ...
EcogSPMem_bl               float64
EcogSPLang_bl              float64
EcogSPVisspat_bl           float64
EcogSPPlan_bl              float64
EcogSPOrgan_bl             float64
EcogSPDivatt_bl            float64
EcogSPTotal_bl             float64
FDG_bl                     float64
PIB_bl                     float64
AV45_bl                    float64
Years_bl                   float64
Month_bl                   float64
Month                        int64
M                            int64
update_stamp                object
3308                       float64
4203                       float64
4381                       float64
11484                      float64
13711                      float64
16478                      float64
16679                      float64
31790                      float64
32056                      float64
38066                      float64
9989                       float64
21323                      float64
49186                      float64
TOMM40_A1                  float64
TOMM40_A2                  float64
Length: 109, dtype: object
```

In [31]:
```
adni_df.describe()
```

Out[31]:

	RID	SITE	AGE	PTEDUCAT	APOE4	FDG	PIB	AV45	CDRSB	ADA
count	13017.000000	13017.000000	13017.000000	13017.000000	12958.000000	3353.000000	223.000000	2161.000000	9016.000000	8959.00(
mean	2285.200584	73.892064	73.767220	15.994930	0.535654	1.208225	1.783161	1.195504	2.163598	11.398
std	1871.013213	110.533877	6.979685	2.824862	0.655480	0.160972	0.422511	0.227999	2.805879	8.616
min	2.000000	2.000000	54.400000	4.000000	0.000000	0.636804	1.095000	0.814555	0.000000	0.00(
25%	631.000000	21.000000	69.500000	14.000000	0.000000	1.109730	1.361250	1.010140	0.000000	5.33(
50%	1301.000000	41.000000	73.700000	16.000000	0.000000	1.219870	1.850000	1.114670	1.000000	9.00(
75%	4353.000000	116.000000	78.600000	18.000000	1.000000	1.314320	2.127500	1.364980	3.000000	15.00(
max	6094.000000	941.000000	91.400000	20.000000	2.000000	1.753320	2.927500	2.669210	18.000000	70.00(

8 rows × 92 columns

Data Preparation

Exclude Entries for Respondents with Alzheimer's at Baseline

Since we want to predict conversion to Alzheimer's Disease, we will remove all entries for respondents for which AD was their baseline diagnosis.

In [32]:
```
def select_non_AD_DX_bl(adni_df):
    return adni_df[adni_df["DX_bl"] != "AD"]
```

In [33]:
```
adni_non_AD_DX_bl_df = select_non_AD_DX_bl(adni_df)
adni_non_AD_DX_bl_df.head()
```

Out[33]:

	RID	PTID	VISCODE	SITE	COLPROT	ORIGPROT	EXAMDATE	DX_bl	AGE	PTGENDER	...	16478	16679	31790	32056	3806(
0	2	011_S_0002	bl	11	ADNI1	ADNI1	2005-09-08	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
1	2	011_S_0002	m06	11	ADNI1	ADNI1	2006-03-06	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
2	2	011_S_0002	m36	11	ADNI1	ADNI1	2008-08-27	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
3	2	011_S_0002	m60	11	ADNIGO	ADNI1	2010-09-22	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
4	2	011_S_0002	m66	11	ADNIGO	ADNI1	2011-03-04	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191

5 rows × 109 columns

Select Respondents with Data at 36 Months or Later

The Ritter paper focused on respondents with data points at 36 months or later. This boundary seems to have been chosen to permit experimentation around conversion time and how that affected sensitivity (true positive rates):

> The sensitivity for patients converting after different time frames (i.e., 12–36 months) is shown in Fig. 3D. As ex- pected, the onset of AD could be best predicted for patients converting after 12 months and worst for patients converting after 36 months.

In [34]:
```
def select_has_m36_or_later(adni_df):
    has_m36_or_later_df = \
        adni_non_AD_DX_bl_df \
            .groupby("RID") \
            .agg({"Month": lambda viscodes: (viscodes >= 36).any()}) \
            .rename(columns={"Month": "has_m36_or_later"})
    adni_has_m36_or_later_mask = adni_df.join(has_m36_or_later_df, on="RID")["has_m36_or_later"]
    return adni_df[adni_has_m36_or_later_mask]
```

In [35]:
```
adni_non_AD_DX_bl_has_m36_or_later_df = select_has_m36_or_later(adni_non_AD_DX_bl_df)
adni_non_AD_DX_bl_has_m36_or_later_df.head()
```

Out[35]:

	RID	PTID	VISCODE	SITE	COLPROT	ORIGPROT	EXAMDATE	DX_bl	AGE	PTGENDER	...	16478	16679	31790	32056	38066
0	2	011_S_0002	bl	11	ADNI1	ADNI1	2005-09-08	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
1	2	011_S_0002	m06	11	ADNI1	ADNI1	2006-03-06	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
2	2	011_S_0002	m36	11	ADNI1	ADNI1	2008-08-27	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
3	2	011_S_0002	m60	11	ADNIGO	ADNI1	2010-09-22	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191
4	2	011_S_0002	m66	11	ADNIGO	ADNI1	2011-03-04	CN	74.3	Male	...	5.718	8.606	7.115	5.072	9.191

5 rows × 109 columns

Annotate Respondents Converted from Non-AD to AD through 36 Months

For the purposes of this project, we will only look at conversions that happen anytime through to the 36 month mark of a respondent's participation in the study.

In [36]:
```
def annotate_converted_non_AD_to_AD_through_m36(adni_df):
    adni_through_m36_df = adni_df[adni_df["Month"] <= 36]
    converted_non_AD_to_AD_through_m36_df = \
        adni_through_m36_df \
            .groupby("RID") \
            .agg({"DX": lambda dxs: "Dementia" in dxs.values}) \
            .rename(columns={"DX": "converted_non_AD_to_AD_through_m36"})
    return adni_df.join(converted_non_AD_to_AD_through_m36_df, on="RID")
```

In [37]:
```
adni_annotated_converted_non_AD_to_AD_through_m36_df = \
    annotate_converted_non_AD_to_AD_through_m36(adni_non_AD_DX_bl_has_m36_or_later_df)
adni_annotated_converted_non_AD_to_AD_through_m36_df.shape
```

Out[37]:
```
(9863, 110)
```

Select Predictor Values Recorded at Baseline

To approximate the model in the Ritter paper, we will use some of the candidate predictors the Ritter paper used, and specifically the values for those predictors recorded at baseline.

Note that although some baseline predictors in `ADNIMERGE` are suffixed with `_bl` , others are not. To simplify our usage of `ADNIMERGE` , we'll only use the non- `_bl` -suffixed columns, and then take the rows where `VISCODE` is `bl` to get the baseline-coded values.

In [38]:
```
adni_annotated_converted_non_AD_to_AD_through_m36_bl_df = \
    adni_annotated_converted_non_AD_to_AD_through_m36_df[
        adni_annotated_converted_non_AD_to_AD_through_m36_df.columns[
            ~adni_annotated_converted_non_AD_to_AD_through_m36_df.columns.str.endswith("_bl")
        ]
    ][
        adni_annotated_converted_non_AD_to_AD_through_m36_df["VISCODE"] == "bl"
    ]
```

In [39]:
```
adni_annotated_converted_non_AD_to_AD_through_m36_bl_df.describe()
```

Out[39]:

	RID	SITE	AGE	PTEDUCAT	APOE4	FDG	PIB	AV45	CDRSB	ADAS11	...
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	767.000000	9.000000	542.000000	1009.000000	1008.000000	...
mean	2505.576809	76.570862	73.379485	16.141724	0.473736	1.278575	1.566944	1.173215	0.920218	8.338899	...
std	1874.221531	116.408026	6.905936	2.730765	0.626430	0.129538	0.315631	0.208780	0.980124	4.374082	...
min	2.000000	2.000000	55.000000	6.000000	0.000000	0.782496	1.180000	0.838537	0.000000	0.000000	...
25%	702.000000	22.000000	68.900000	14.000000	0.000000	1.194700	1.360000	1.012010	0.000000	5.000000	...
50%	2146.000000	51.000000	73.400000	16.000000	0.000000	1.280300	1.490000	1.088980	0.500000	7.670000	...
75%	4419.000000	123.000000	78.300000	18.000000	1.000000	1.359995	1.672500	1.315997	1.500000	11.000000	...
max	5290.000000	941.000000	90.100000	20.000000	2.000000	1.707170	2.232500	2.025560	5.500000	27.670000	...

8 rows × 56 columns

In [40]:
```
adni_annotated_converted_non_AD_to_AD_through_m36_bl_na_sum = \
    adni_annotated_converted_non_AD_to_AD_through_m36_bl_df.isna().sum()
adni_annotated_converted_non_AD_to_AD_through_m36_bl_na_pct = \
    adni_annotated_converted_non_AD_to_AD_through_m36_bl_na_sum / \
        adni_annotated_converted_non_AD_to_AD_through_m36_bl_df.shape[0]
adni_annotated_converted_non_AD_to_AD_through_m36_bl_na_pct[
    adni_annotated_converted_non_AD_to_AD_through_m36_bl_na_pct > 0
]
```

Out[40]:
```
FDG                        0.239841
PIB                        0.991080
AV45                       0.462834
ADAS11                     0.000991
ADAS13                     0.002973
RAVLT_immediate            0.001982
RAVLT_learning             0.001982
RAVLT_forgetting           0.001982
RAVLT_perc_forgetting      0.001982
FAQ                        0.003964
MOCA                       0.466799
EcogPtMem                  0.461843
EcogPtLang                 0.461843
EcogPtVisspat              0.463826
EcogPtPlan                 0.461843
EcogPtOrgan                0.468781
EcogPtDivatt               0.462834
EcogPtTotal                0.461843
EcogSPMem                  0.463826
EcogSPLang                 0.463826
EcogSPVisspat              0.471754
EcogSPPlan                 0.467790
EcogSPOrgan                0.490585
EcogSPDivatt               0.477701
EcogSPTotal                0.464817
FLDSTRENG                  0.127849
FSVERSION                  0.007929
Ventricles                 0.037661
Hippocampus                0.128840
WholeBrain                 0.015857
Entorhinal                 0.135778
Fusiform                   0.135778
MidTemp                    0.135778
ICV                        0.007929
3308                       0.398414
4203                       0.398414
4381                       0.398414
11484                      0.398414
13711                      0.398414
16478                      0.398414
16679                      0.398414
31790                      0.398414
32056                      0.398414
38066                      0.398414
9989                       0.398414
21323                      0.398414
49186                      0.398414
TOMM40_A1                  0.575818
TOMM40_A2                  0.575818
dtype: float64
```

Almost all of the features have missing values for some percentage of the respondents.

Utility Functions

Imputation

As discussed in the previous section, and as noted in the Ritter paper, various datasets and predictors are missing data for respondents to varying degrees.

In class we learned about mean imputation, which was also applied in the Ritter paper.

In [41]:
```
def impute_missing_values_with_mean(df, columns, missing_values):
    imputed_df = df.copy()
    for column in columns:
        imputed_df[f"{column}"] = impute_column_missing_values_with_mean(df, column, missing_values)
    return imputed_df

def impute_column_missing_values_with_mean(df, column, missing_values):
    imputer = SimpleImputer(copy=True, missing_values=missing_values, strategy="mean")
    imputed = pd.Series(
        imputer.fit_transform(df[column].astype(float).values.reshape(-1, 1)).reshape(-1),
        index=df[column].index
    )
    return imputed
```

Dummy Encoding

Some features like PTGENDER are categorical, so we need to encode these properly.

In [42]:
```
def dummy_encode_predictors(df, predictors):
    return pd.get_dummies(df, columns=predictors, drop_first=True)
```

Scaling

We will scale our non-categorical data, to let us apply algorithms to the ADNI data which assume or work best with scaled data (e.g. [SVM with RBF kernel (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)).

In [43]:
```
def scale_predictors(df, predictors, scaler):
    df_scaled = df.copy()
    df_subset_scaled = pd.DataFrame(scaler.transform(df[predictors]), columns=predictors)
    for predictor in predictors:
        df_scaled[predictor] = df_subset_scaled[predictor].values
    return df_scaled

def create_scaler(df, predictors):
    scaler = StandardScaler()
    return scaler.fit(df[predictors])
```

Creating Design Matrixes

As with our assignments, we have a utility function to split datasets into training and test datasets, creating response vectors and design matrixes where imputation, dummy encoding, and scaling have been applied.

In [44]:
```
def create_design_mats(
    df,
    test_size,
    response,
    predictors,
    predictors_to_scale,
    predictors_to_dummy_encode
):
    X = df[predictors]
    y = df[response]

    X_train, X_test, y_train, y_test = train_test_split(
        impute_missing_values_with_mean(
            dummy_encode_predictors(X, predictors_to_dummy_encode),
            predictors_to_scale,
            np.nan),
        y,
        test_size=test_size,
        stratify=y
    )

    scaler = create_scaler(X_train, predictors_to_scale)
    X_train = scale_predictors(X_train, predictors_to_scale, scaler)
    X_test = scale_predictors(X_test, predictors_to_scale, scaler)

    return X_train, y_train, X_test, y_test
```

Plotting Confusion Matrix

Since we are framing this as a classification problem, we have a utility method to get our confusion matrix (from scikit-learn examples). The matrix can help us calculate true positive rate (sensitivity/recall) and true negative rate (specificity), among other things.

In [45]:
```
CLASS_NAMES = ["Cognitively Normal (CN)", "Dementia"]
```

In [46]:
```
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto
def plot_confusion_matrix_without_and_with_normalization(cnf_matrix, title):
    np.set_printoptions(precision=2)

    fig, axs = plt.subplots(1, 2, figsize=(16, 10), sharex=False, sharey=False)
    fig.suptitle(title, fontsize=24)

    plot_confusion_matrix(axs[0], cnf_matrix, CLASS_NAMES,
                          title="Confusion matrix, without normalization")
    plot_confusion_matrix(axs[1], cnf_matrix, CLASS_NAMES, normalize=True,
                          title="Confusion matrix, with normalization")

    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(ax, cnf_matrix, class_names,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cnf_matrix = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]

    ax.imshow(cnf_matrix, interpolation='nearest', cmap=cmap)
    ax.set_title(title, fontsize=24)
    tick_marks = np.arange(len(class_names))
    ax.set_xticks(tick_marks)
    ax.set_xticklabels(class_names)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
    ax.set_yticks(tick_marks)
    ax.set_yticklabels(class_names)
    ax.set_yticklabels(ax.get_yticklabels(), rotation=45)

    fmt = '.2f' if normalize else 'd'
    thresh = cnf_matrix.max() / 2.
    for i, j in itertools.product(range(cnf_matrix.shape[0]), range(cnf_matrix.shape[1])):
        ax.text(j, i, format(cnf_matrix[i, j], fmt),
                horizontalalignment="center",
                color="white" if cnf_matrix[i, j] > thresh else "black",
                fontsize=24)
    ax.set_xlabel("Predicted Label", fontsize=14)
    ax.set_ylabel("True Label", fontsize=14)
```

Plotting ROC Curves

In this project, we will attempt to tune our models, as Ritter et al. did, using ROC curves. In the same spirit of going beyond just looking at overall classification accuracy, we will use the ROC curve to optimize for balanced accuracy--mean of TPR and TNR (or 1 - FPR)--and find a suitable classification probability threshold.

In [47]:
```
def augmented_roc_curves_cv(base_model, X, y, n_splits=3):
    cv = StratifiedKFold(n_splits=n_splits)
    X_matrix = X.as_matrix()
    y_matrix = y.as_matrix()
    return [
        augmented_roc_curve_cv(base_model, X_matrix[train], y_matrix[train], X_matrix[val], y_matrix[val
        for train, val in cv.split(X_matrix, y_matrix)
    ]

def augmented_roc_curve_cv(base_model, X_train, y_train, X_val, y_val):
    model = clone(base_model).fit(X_train, y_train)
    probabilities = model.predict_proba(X_val)
    fpr, tpr, threshold = roc_curve(y_val, probabilities[:, 1], drop_intermediate=False)
    tnr = 1 - fpr
    balanced_accuracy = (tpr + tnr) / 2.0
    return fpr, tpr, threshold, balanced_accuracy
```

In [48]:
```
def mean_threshold_best_balanced_accuracy(augmented_roc_curves):
    threshold_best_balanced_accuracies, best_balanced_accuracies = zip(*[
        threshold_best_balanced_accuracy(augmented_roc_curve)
        for augmented_roc_curve in augmented_roc_curves
    ])
    return np.mean(threshold_best_balanced_accuracies), np.mean(best_balanced_accuracies)

def threshold_best_balanced_accuracy(augmented_roc_curve):
    _, _, threshold, balanced_accuracy = augmented_roc_curve
    idx_best_balanced_accuracy = balanced_accuracy.argmax()
    return threshold[idx_best_balanced_accuracy], balanced_accuracy[idx_best_balanced_accuracy]
```

In [49]:
```
# Adapted from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html
def plot_augmented_roc_curves(augmented_roc_curves):
    fig, ax = plt.subplots(figsize=(16, 16))
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")

    for i, (fpr, tpr, threshold, balanced_accuracy) in enumerate(augmented_roc_curves):
        roc_auc = auc(fpr, tpr)
        ax.plot(fpr, tpr, alpha=0.3, label=f"ROC fold {i} (AUC = {roc_auc:.2f})")

        idx_best_balanced_accuracy = balanced_accuracy.argmax()
        fpr_best_balanced_accuracy = fpr[idx_best_balanced_accuracy]
        tpr_best_balanced_accuracy = tpr[idx_best_balanced_accuracy]
        threshold_best_balanced_accuracy = threshold[idx_best_balanced_accuracy]
        ax.scatter(fpr_best_balanced_accuracy,
                   tpr_best_balanced_accuracy,
                   marker="*",
                   label=f"best balanced accuracy fold {i} (threshold = {threshold_best_balanced_accurac

    ax.legend()
    plt.show()
```

Evaluating Models

In order to evaluate and compare various models, we have some utility functions to consolidate the metrics that concern us for a given model.

In [50]:
```
def create_and_evaluate_model(base_model, X_train, y_train, X_test, y_test):
    model = clone(base_model).fit(X_train, y_train)

    y_predicted_train = model.predict(X_train)
    y_predicted_test = model.predict(X_test)

    cnf_matrix_train = confusion_matrix(y_train, y_predicted_train)
    tn_train, fp_train, fn_train, tp_train = cnf_matrix_train.ravel()

    cnf_matrix_test = confusion_matrix(y_test, y_predicted_test)
    tn_test, fp_test, fn_test, tp_test = cnf_matrix_test.ravel()

    return (
        model,
        (
            cnf_matrix_train,
            accuracy_score(y_train, y_predicted_train),
            balanced_accuracy_score(y_train, y_predicted_train),
            tn_train / (tn_train + fp_train),
            tp_train / (tp_train + fn_train)
        ),
        (
            cnf_matrix_test,
            accuracy_score(y_test, y_predicted_test),
            balanced_accuracy_score(y_test, y_predicted_test),
            tn_test / (tn_test + fp_test),
            tp_test / (tp_test + fn_test)
        )
    )
```

In [51]:
```
def create_and_evaluate_probabilistic_model(base_model, X_train, y_train, X_test, y_test, threshold):
    model = clone(base_model).fit(X_train, y_train)

    y_predicted_train = model.predict_proba(X_train)[:, 1] > threshold
    y_predicted_test = model.predict_proba(X_test)[:, 1] > threshold

    cnf_matrix_train = confusion_matrix(y_train, y_predicted_train)
    tn_train, fp_train, fn_train, tp_train = cnf_matrix_train.ravel()

    cnf_matrix_test = confusion_matrix(y_test, y_predicted_test)
    tn_test, fp_test, fn_test, tp_test = cnf_matrix_test.ravel()

    return (
        model,
        (
            cnf_matrix_train,
            accuracy_score(y_train, y_predicted_train),
            balanced_accuracy_score(y_train, y_predicted_train),
            tn_train / (tn_train + fp_train),
            tp_train / (tp_train + fn_train)
        ),
        (
            cnf_matrix_test,
            accuracy_score(y_test, y_predicted_test),
            balanced_accuracy_score(y_test, y_predicted_test),
            tn_test / (tn_test + fp_test),
            tp_test / (tp_test + fn_test)
        )
    )
```

In [52]:
```
def plot_model_results(model_results, base_title):
    confusion_matrix, accuracy, balanced_accuracy, tnr, tpr = model_results
    plot_confusion_matrix_without_and_with_normalization(
        confusion_matrix,
        f"{base_title}"
        f"\n(accuracy: {accuracy:.2}, balanced accuracy: {balanced_accuracy:.2}"
        f", specificity: {tnr:.2}, sensitivity: {tpr:.2})"
    )
```

In [53]:
```
def model_results_to_dict(model_results, name):
    cnf_matrix, accuracy, balanced_accuracy, specificity, sensitivity = model_results
    return {
        "model": name,
        "confusion matrix": cnf_matrix,
        "accuracy": accuracy,
        "balanced accuracy": balanced_accuracy,
        "specificity": specificity,
        "sensitivity": sensitivity
    }
```

Models

Response and Predictors

Similar to the models in the Ritter paper, our models will predict conversions to AD within the first 36 months.

In [54]:
```
ADNI_RESPONSE = "converted_non_AD_to_AD_through_m36"
```

We also consider the predictors from the ADNIMERGE dataset.

In [55]:
```
ADNI_PREDICTORS = [
    "AGE",
    "PTGENDER",
    "PTEDUCAT",
    "PTETHCAT",
    "PTRACCAT",
    "APOE4",
    "FDG",
    "CDRSB",
    "ADAS11",
    "ADAS13",
    "MMSE",
    "RAVLT_immediate",
    "RAVLT_learning",
    "RAVLT_forgetting",
    "RAVLT_perc_forgetting",
    "FAQ",
    "Ventricles",
    "Hippocampus",
    "WholeBrain",
    "Entorhinal",
    "Fusiform",
    "MidTemp",

    "3308",
    "4203",
    "4381",
    "11484",
    "13711",
    "16478",
    "16679",
    "31790",
    "32056",
    "38066",
    "9989",
    "21323",
    "49186",
    "TOMM40_A1",
    "TOMM40_A2"
]

ADNI_PREDICTORS_TO_DUMMY_ENCODE = [
    "PTGENDER",
    "PTEDUCAT",
    "PTETHCAT",
    "PTRACCAT"
]

ADNI_PREDICTORS_TO_SCALE = [
    "AGE",
    "APOE4",
    "FDG",
    "CDRSB",
    "ADAS11",
    "ADAS13",
    "MMSE",
    "RAVLT_immediate",
    "RAVLT_learning",
    "RAVLT_forgetting",
    "RAVLT_perc_forgetting",
    "FAQ",
    "Ventricles",
    "Hippocampus",
    "WholeBrain",
    "Entorhinal",
    "Fusiform",
    "MidTemp",

    "3308",
    "4203",
    "4381",
    "11484",
    "13711",
    "16478",
    "16679",
    "31790",
```

```
    "32056",
    "38066",
    "9989",
    "21323",
    "49186",
    "TOMM40_A1",
    "TOMM40_A2"
]
```

Now let's create the response vectors and design matrixes for training and test data.

In [56]:
```
X_train, y_train, X_test, y_test = create_design_mats(
    adni_annotated_converted_non_AD_to_AD_through_m36_bl_df,
    test_size=.25,
    response="converted_non_AD_to_AD_through_m36",
    predictors=ADNI_PREDICTORS,
    predictors_to_scale=ADNI_PREDICTORS_TO_SCALE,
    predictors_to_dummy_encode=ADNI_PREDICTORS_TO_DUMMY_ENCODE
)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[56]:
```
((756, 55), (756,), (253, 55), (253,))
```

In [57]:
```
X_train.dtypes
```

Out[57]:
```
AGE                         float64
APOE4                       float64
FDG                         float64
CDRSB                       float64
ADAS11                      float64
ADAS13                      float64
MMSE                        float64
RAVLT_immediate             float64
RAVLT_learning              float64
RAVLT_forgetting            float64
RAVLT_perc_forgetting       float64
FAQ                         float64
Ventricles                  float64
Hippocampus                 float64
WholeBrain                  float64
Entorhinal                  float64
Fusiform                    float64
MidTemp                     float64
3308                        float64
4203                        float64
4381                        float64
11484                       float64
13711                       float64
16478                       float64
16679                       float64
31790                       float64
32056                       float64
38066                       float64
9989                        float64
21323                       float64
49186                       float64
TOMM40_A1                   float64
TOMM40_A2                   float64
PTGENDER_Male                 uint8
PTEDUCAT_7                    uint8
PTEDUCAT_8                    uint8
PTEDUCAT_9                    uint8
PTEDUCAT_10                   uint8
PTEDUCAT_11                   uint8
PTEDUCAT_12                   uint8
PTEDUCAT_13                   uint8
PTEDUCAT_14                   uint8
PTEDUCAT_15                   uint8
PTEDUCAT_16                   uint8
PTEDUCAT_17                   uint8
PTEDUCAT_18                   uint8
PTEDUCAT_19                   uint8
PTEDUCAT_20                   uint8
PTETHCAT_Not Hisp/Latino      uint8
PTETHCAT_Unknown              uint8
PTRACCAT_Asian                uint8
PTRACCAT_Black                uint8
PTRACCAT_More than one        uint8
PTRACCAT_Unknown              uint8
PTRACCAT_White                uint8
dtype: object
```

In [58]:
```
X_train.head()
```

Out[58]:

	AGE	APOE4	FDG	CDRSB	ADAS11	ADAS13	MMSE	RAVLT_immediate	RAVLT_learning	RAVLT_forgetting	...
2363	1.743022	-0.724964	-0.022628	-0.932353	-0.707667	-0.911686	-0.770116	-0.235592	-0.694131	0.287764	...
63	0.723033	-0.724964	-0.216673	-0.932353	-0.937790	-0.761790	0.439038	0.554166	0.467958	-0.085735	...
11919	-1.287805	-0.724964	-0.269262	-0.432807	-0.783608	-0.811256	0.439038	0.905170	0.080595	-0.085735	...
11113	0.315037	-0.724964	-0.067535	0.066738	0.367007	0.987492	0.439038	-1.113102	0.080595	1.408259	...
8578	0.213038	-0.724964	-1.460116	1.065829	0.136884	0.837596	-2.583848	-1.464105	-1.081494	0.287764	...

5 rows × 55 columns

In [59]:
```
X_train.describe()
```

Out[59]:

	AGE	APOE4	FDG	CDRSB	ADAS11	ADAS13	MMSE	RAVLT_immediate	RAVLT_le
count	7.560000e+02	7.560000e+02	7.560000e+02	7.560000e+02	7.560000e+02	7.560000e+02	7.560000e+02	7.560000e+02	7.5600(
mean	-3.219059e-16	1.057355e-17	1.335205e-15	4.170679e-17	-7.849394e-17	8.825979e-17	-1.074097e-15	5.404627e-16	-6.3147
std	1.000662e+00	1.000662e+00	1.000662e+00	1.000662e+00	1.000662e+00	1.000662e+00	1.000662e+00	1.000662e+00	1.0006(
min	-2.701219e+00	-7.249643e-01	-3.293422e+00	-9.323526e-01	-1.934222e+00	-1.860526e+00	-3.188426e+00	-2.429366e+00	-2.6309
25%	-6.357397e-01	-7.249643e-01	-5.491539e-01	-9.323526e-01	-7.836077e-01	-8.112559e-01	-7.701164e-01	-7.620979e-01	-6.9413
50%	-2.010298e-02	-7.249643e-01	-2.262801e-02	-4.328072e-01	-9.323901e-02	-1.869405e-01	4.390383e-01	-6.009037e-02	8.0595
75%	6.975328e-01	8.636531e-01	4.831908e-01	5.662836e-01	5.971297e-01	6.877006e-01	1.043616e+00	7.296680e-01	8.5532
max	2.413302e+00	2.452271e+00	3.830271e+00	4.562647e+00	4.433278e+00	3.935939e+00	1.043616e+00	2.747940e+00	2.4047

8 rows × 55 columns

The proportion of non-AD respondents in the training set is:

In [60]:
```
y_train[~y_train].count() / y_train.count()
```

Out[60]: 0.8042328042328042

The proportion of non-AD respondents in the test set is:

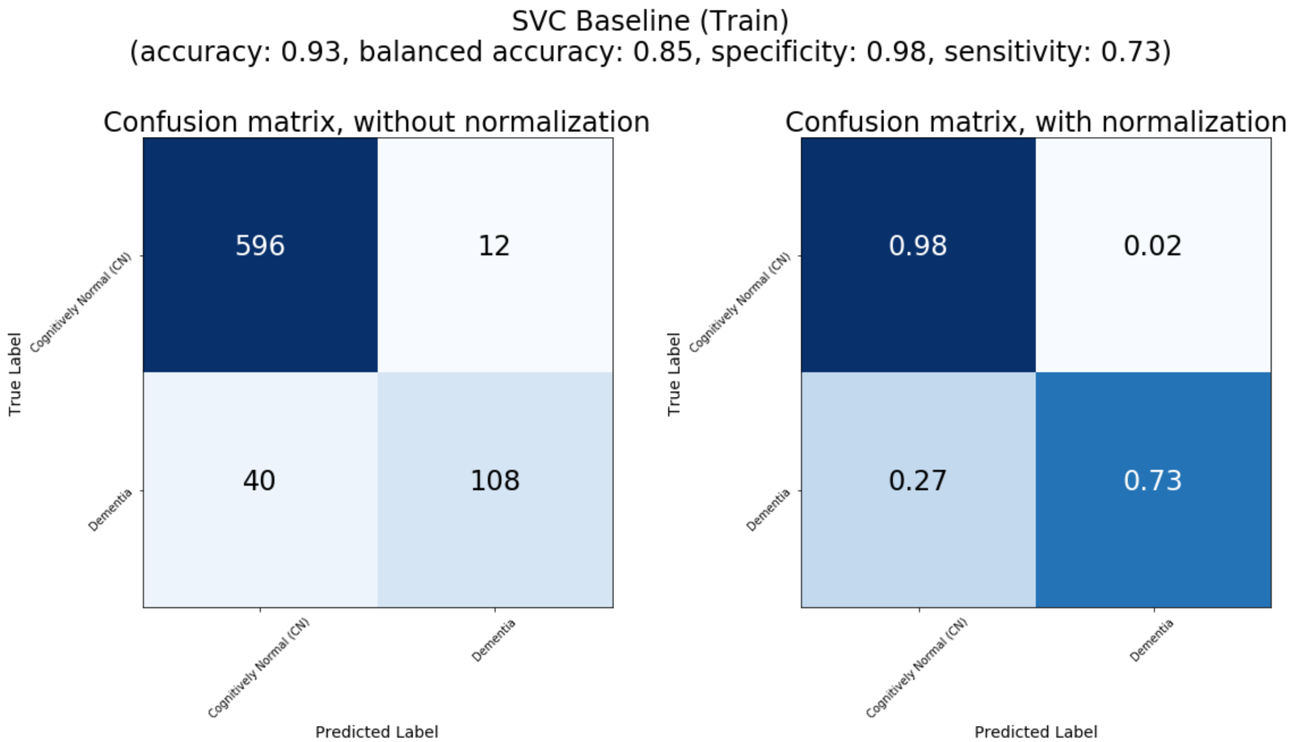
In [61]:
```
y_test[~y_test].count() / y_test.count()
```

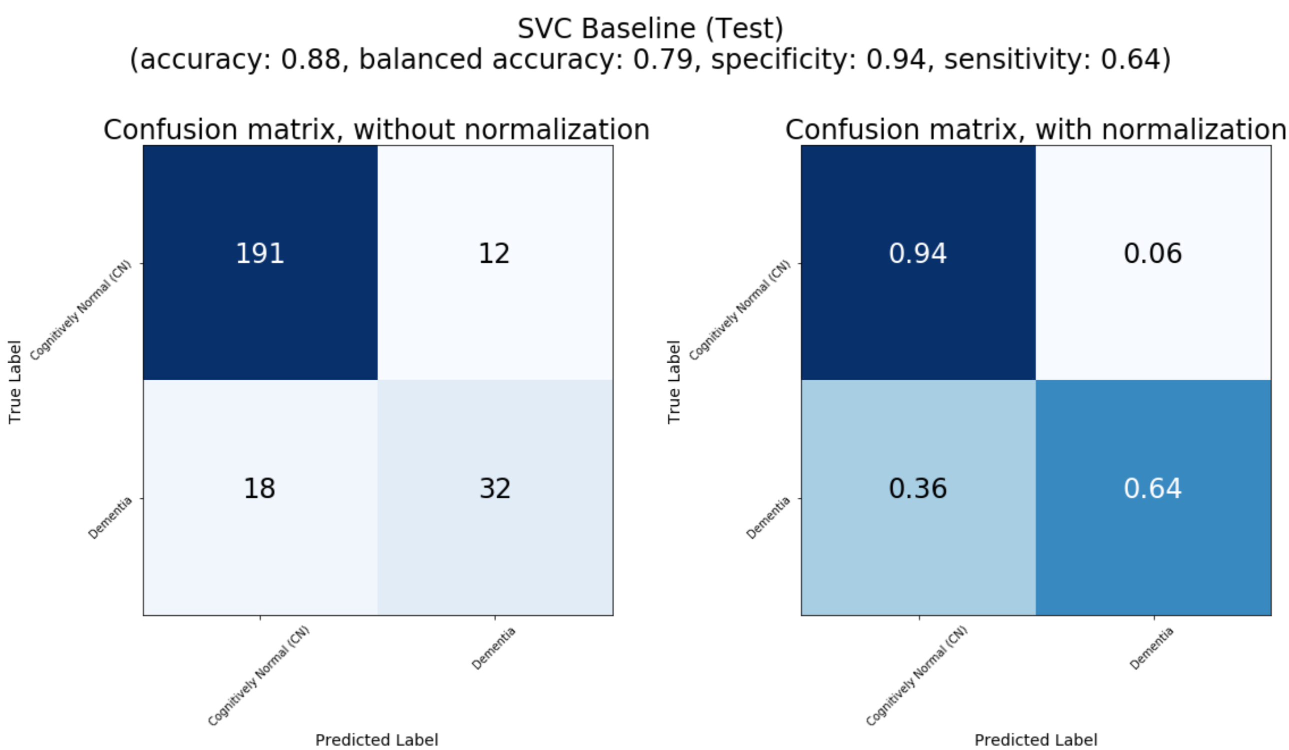
Out[61]: 0.8023715415019763

If you just guessed non-AD for all respondents, you'd be 80% correct.

SVM

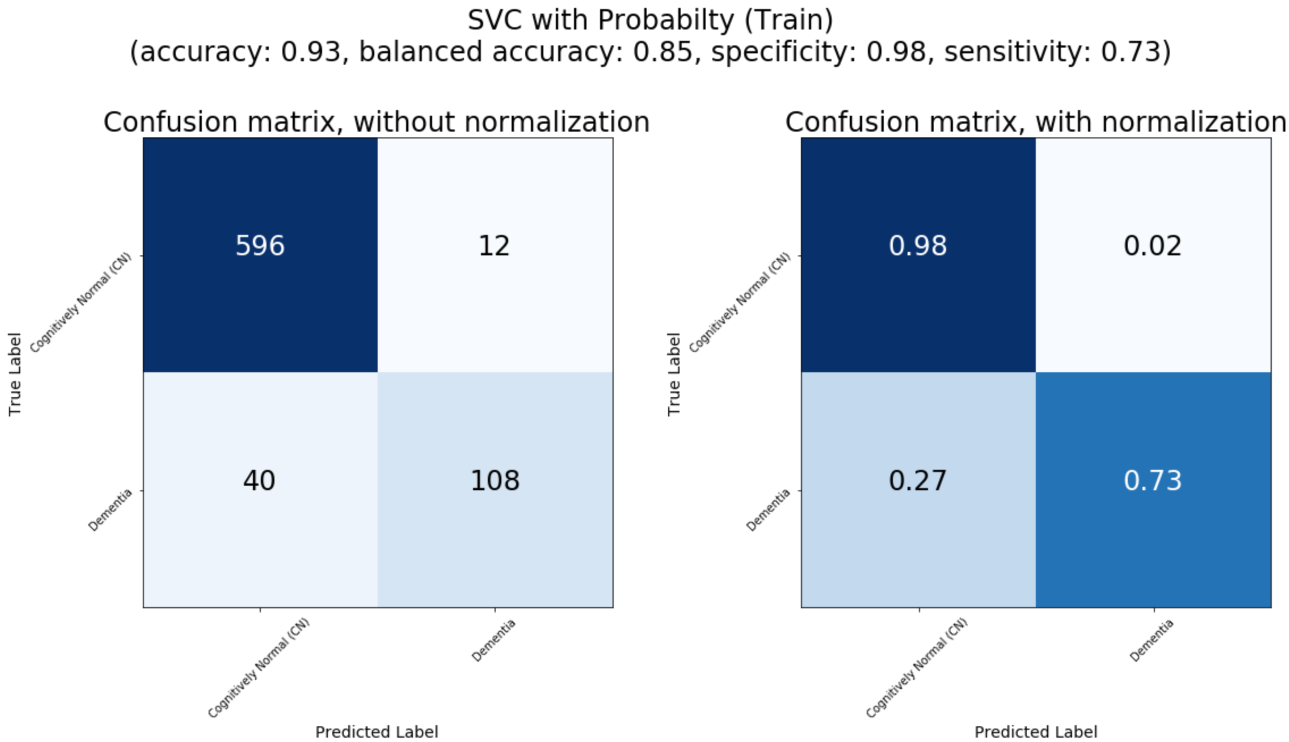
In [62]:
```python
svc_baseline_model, svc_baseline_results_train, svc_baseline_results_test = \
    create_and_evaluate_model(
        GridSearchCV(
            SVC(),
            {
                "C": [100, 20, 10, 2, 1, 0.5, 0.1, 0.05, 0.01]
            },
            cv=3
        ).fit(X_train, y_train).best_estimator_,
        X_train,
        y_train,
        X_test,
        y_test
    )
plot_model_results(svc_baseline_results_train, "SVC Baseline (Train)")
plot_model_results(svc_baseline_results_test, "SVC Baseline (Test)")
```

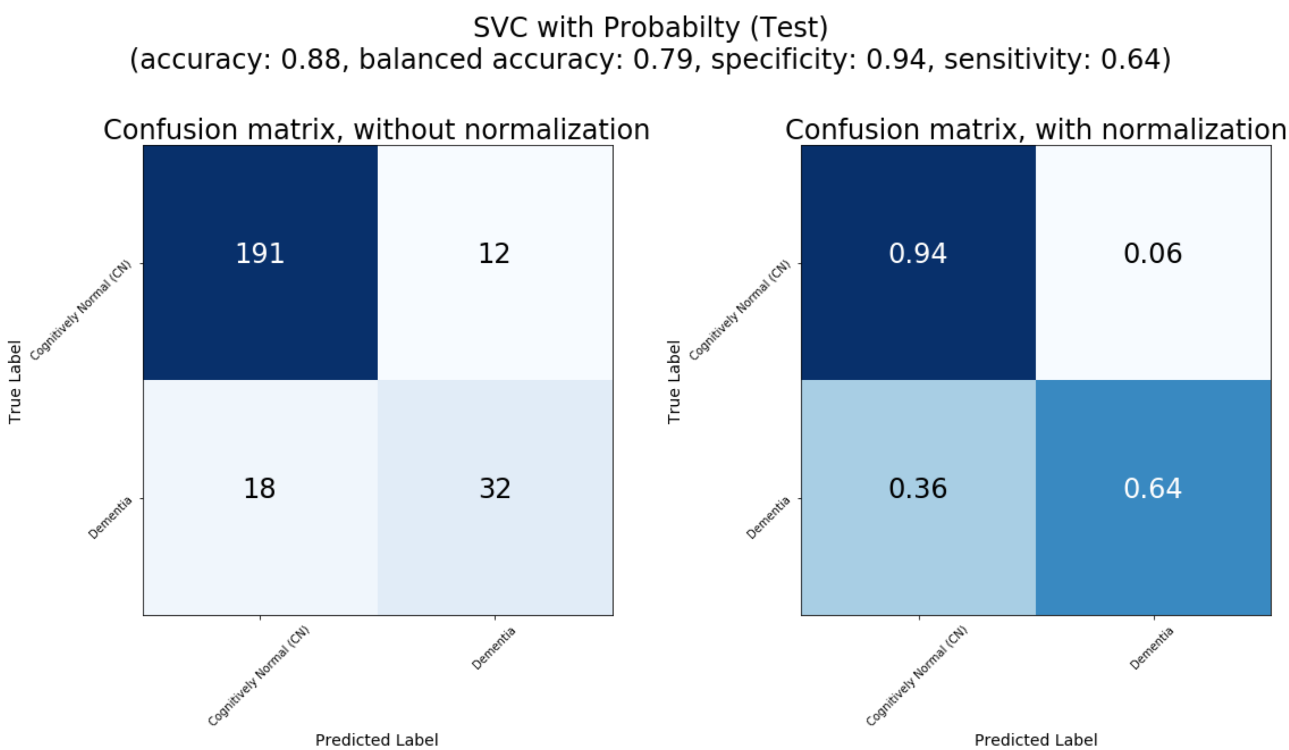




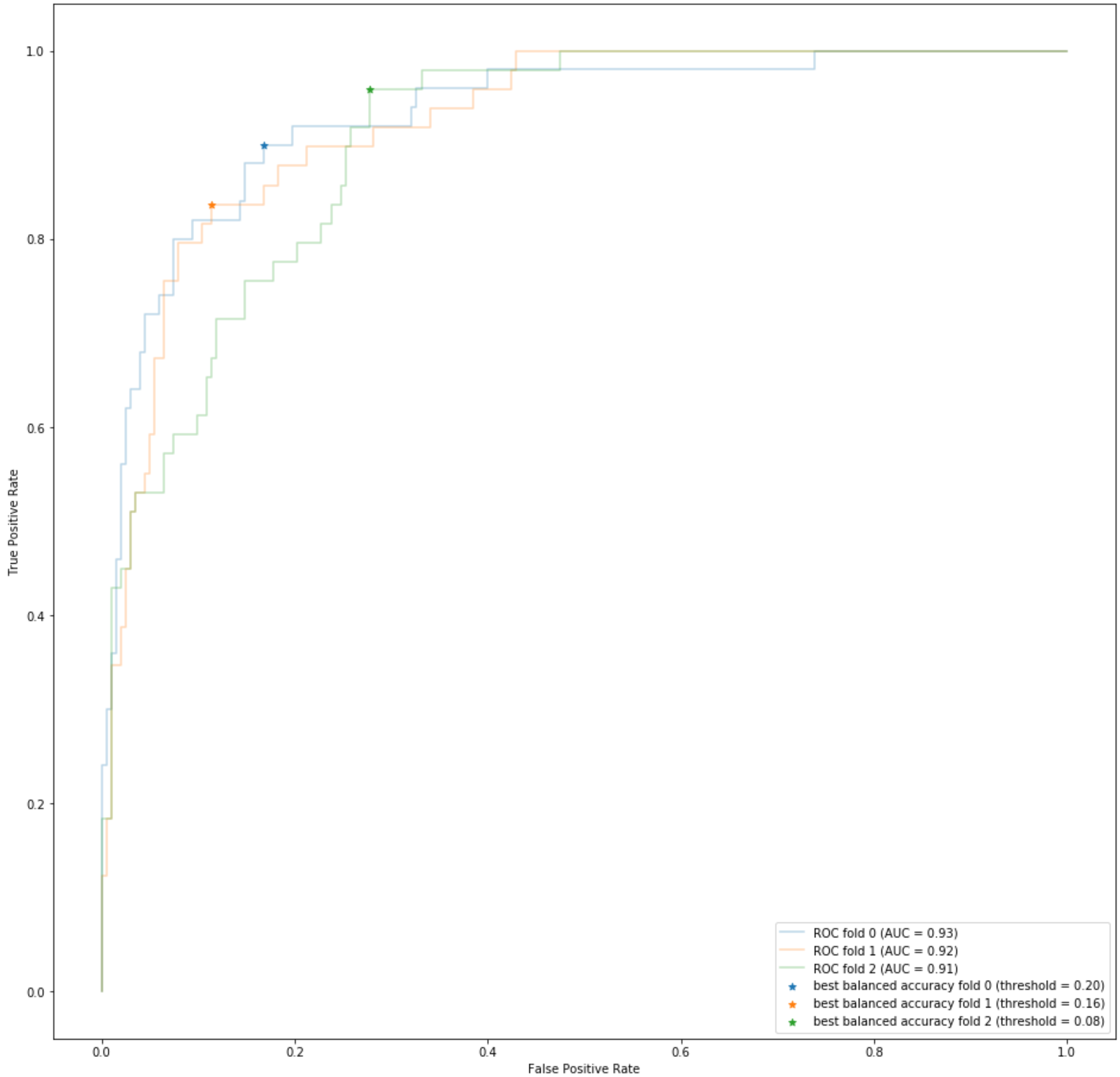
Adjusting SVM with ROC

In [63]:
```
svc_probabilistic_model, svc_probabilistic_results_train, svc_probabilistic_results_test = \
    create_and_evaluate_model(
        GridSearchCV(
            SVC(probability=True),
            {
                "C": [100, 20, 10, 2, 1, 0.5, 0.1, 0.05, 0.01]
            },
            cv=3
        ).fit(X_train, y_train).best_estimator_,
        X_train,
        y_train,
        X_test,
        y_test
    )
plot_model_results(svc_probabilistic_results_train, "SVC with Probabilty (Train)")
plot_model_results(svc_probabilistic_results_test, "SVC with Probabilty (Test)")
```

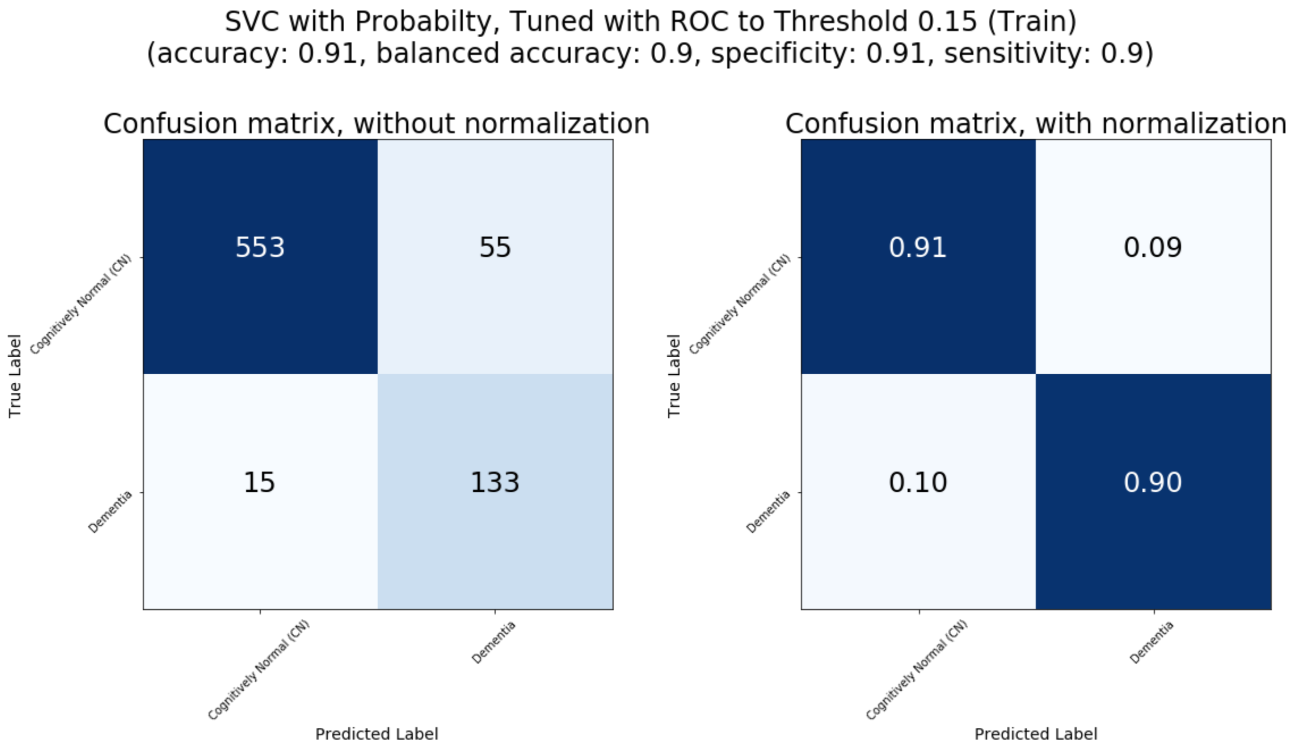


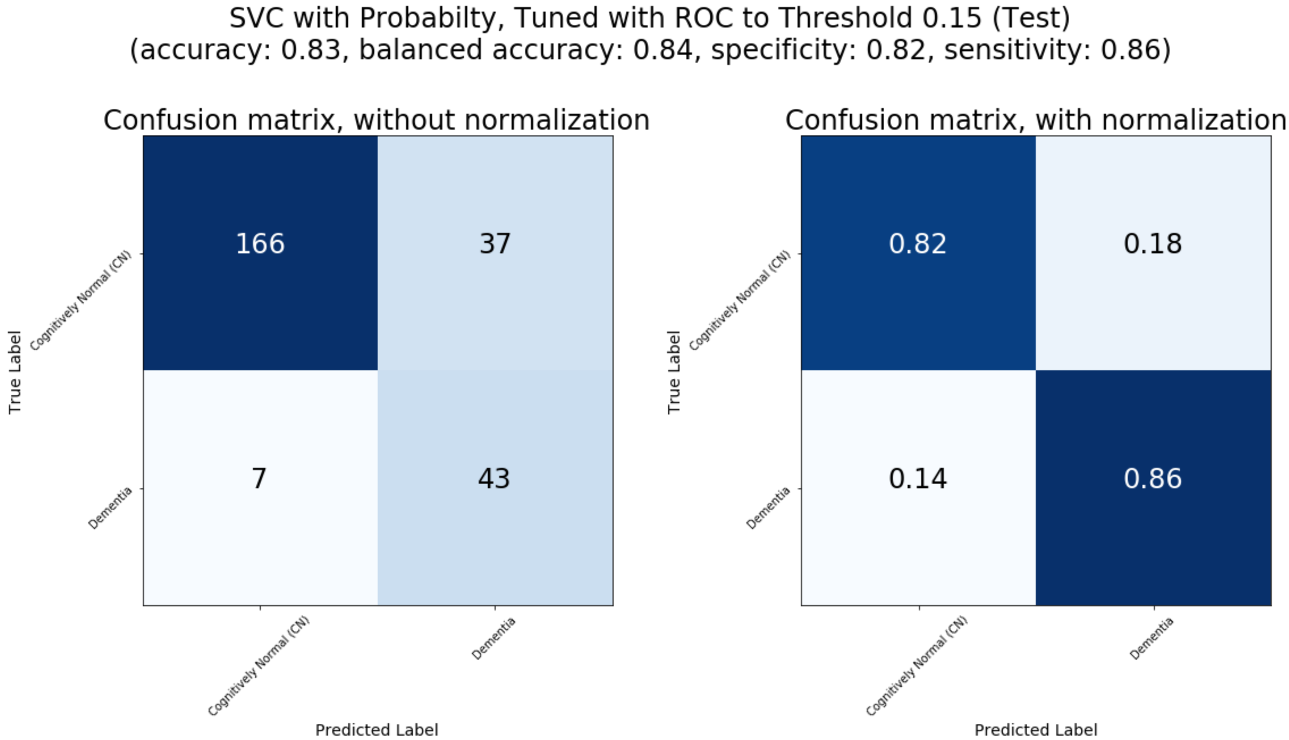


In [64]:
```
svc_probabilistic_augmented_roc_curves = augmented_roc_curves_cv(
    svc_probabilistic_model,
    X_train,
    y_train
)
svc_roc_tuned_threshold, _ = mean_threshold_best_balanced_accuracy(svc_probabilistic_augmented_roc_curve:
plot_augmented_roc_curves(svc_probabilistic_augmented_roc_curves)
```



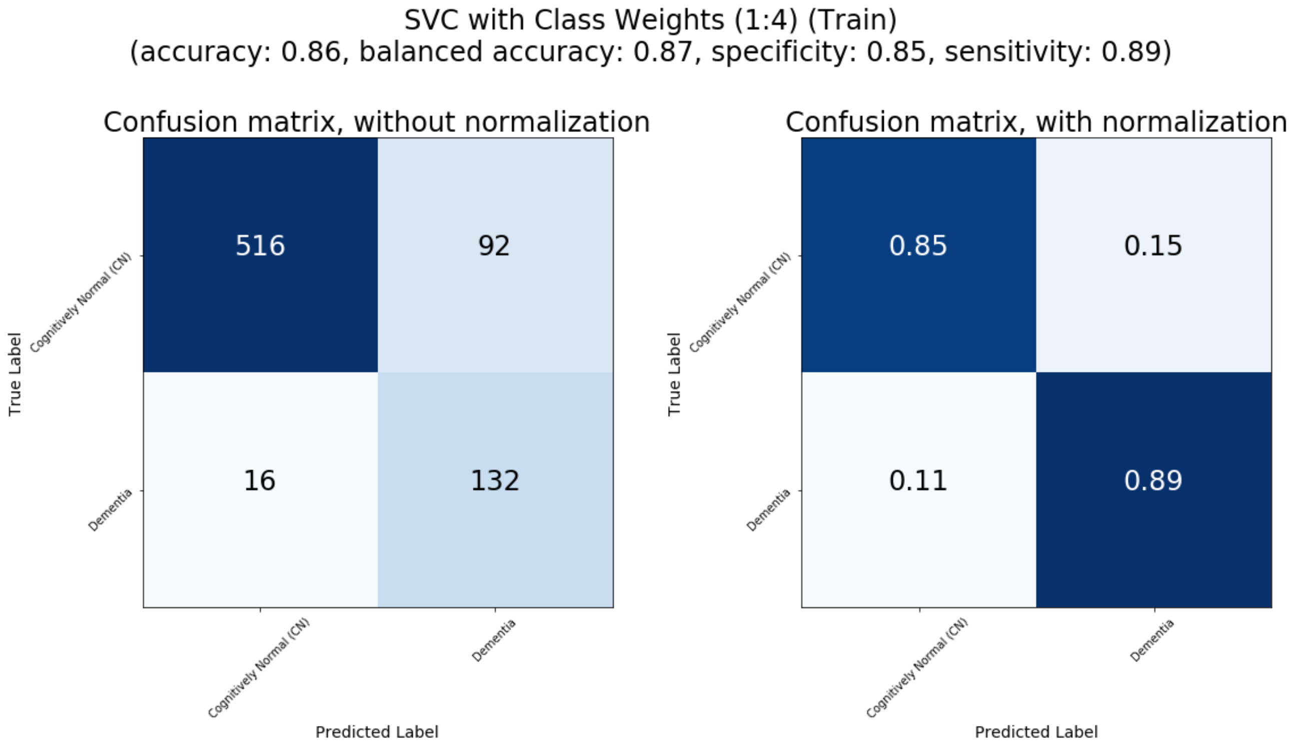
In [65]:
```
svc_roc_tuned_model, svc_roc_tuned_results_train, svc_roc_tuned_results_test = \
    create_and_evaluate_probabilistic_model(
        svc_probabilistic_model,
        X_train,
        y_train,
        X_test,
        y_test,
        svc_roc_tuned_threshold
    )
plot_model_results(
    svc_roc_tuned_results_train,
    f"SVC with Probabilty, Tuned with ROC to Threshold {svc_roc_tuned_threshold:.2} (Train)"
)
plot_model_results(
    svc_roc_tuned_results_test,
    f"SVC with Probabilty, Tuned with ROC to Threshold {svc_roc_tuned_threshold:.2} (Test)"
)
```

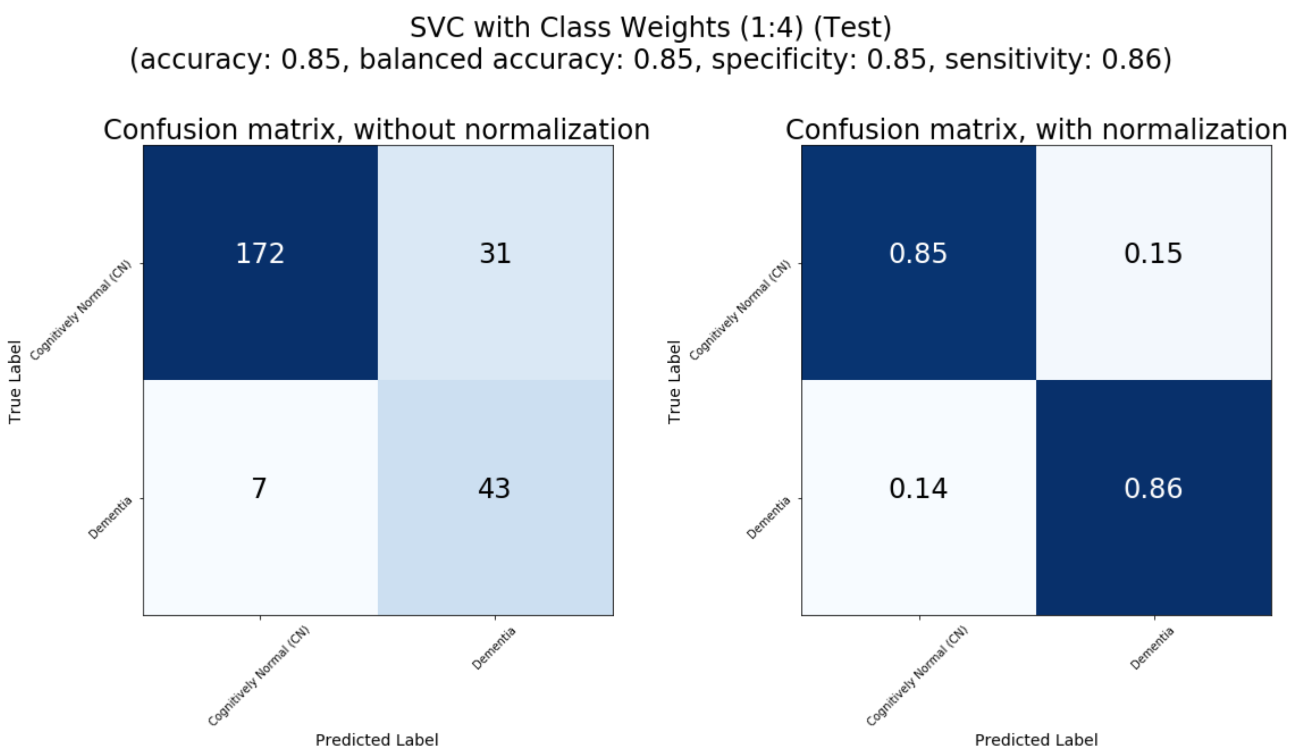




Adjusting SVM with Class Weights

In [66]:
```
svc_weighted_model, svc_weighted_results_train, svc_weighted_results_test = \
    create_and_evaluate_model(
        GridSearchCV(
            SVC(class_weight={0: 1, 1: 4}),
            {
                "C": [100, 20, 10, 2, 1, 0.5, 0.1, 0.05, 0.01]
            },
            cv=3
        ).fit(X_train, y_train).best_estimator_,
        X_train,
        y_train,
        X_test,
        y_test
    )
plot_model_results(svc_weighted_results_train, "SVC with Class Weights (1:4) (Train)")
plot_model_results(svc_weighted_results_test, "SVC with Class Weights (1:4) (Test)")
```





In [ ]:

In [ ]:

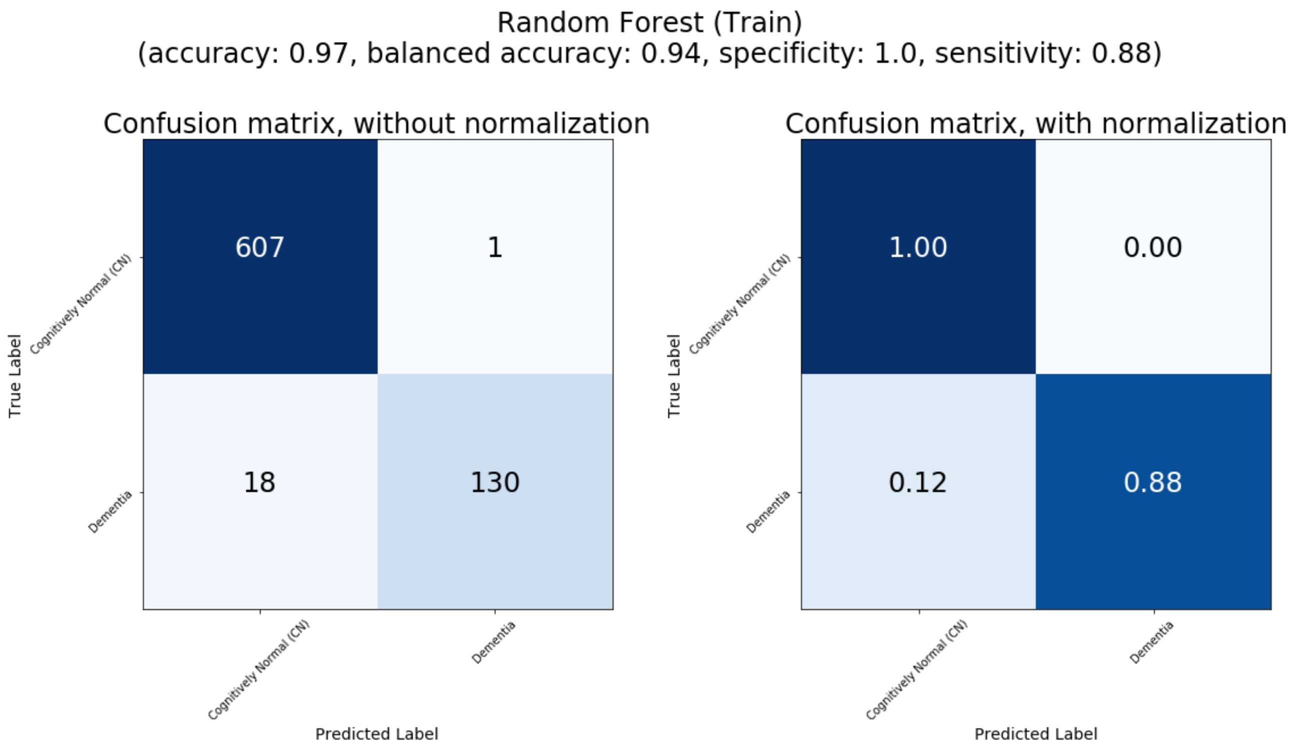
Random Forest

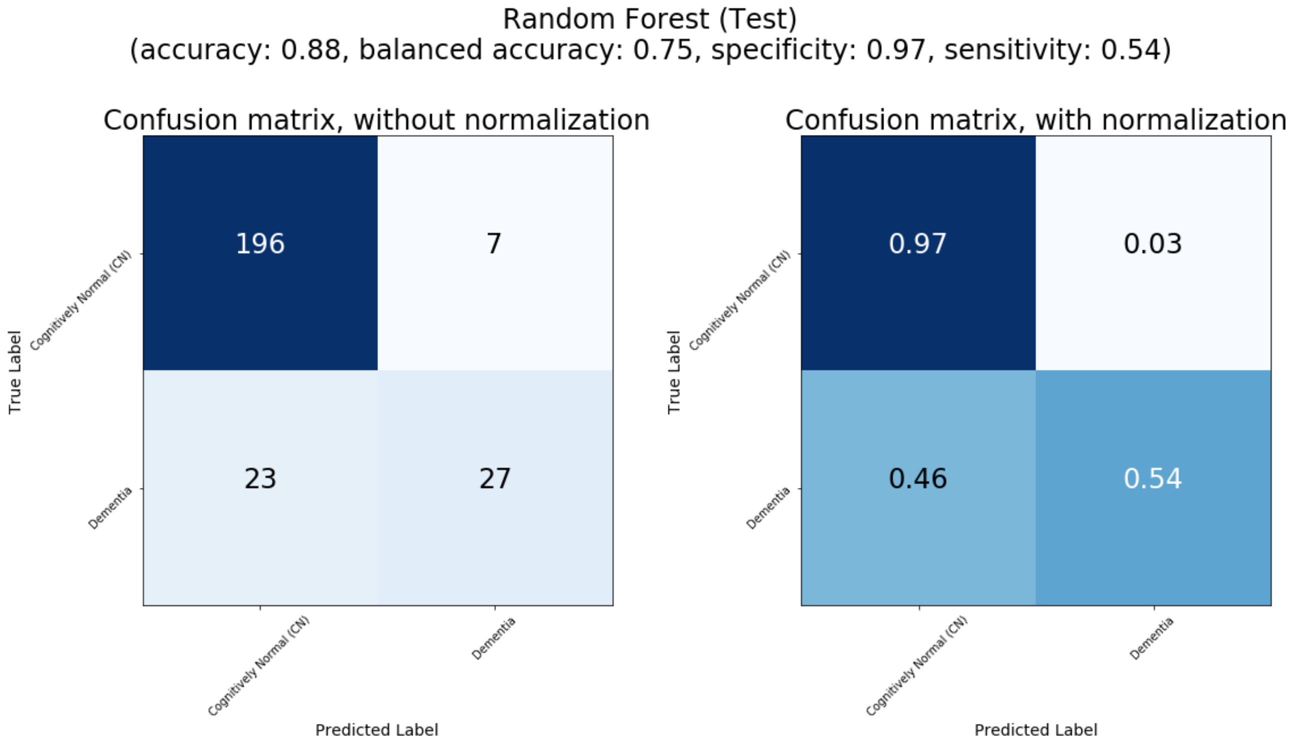
In [67]:
```
def rf_top_feature_counts(rf_model, columns):
    random_forest_top_feature_counts = \
        pd.Series(np.array([tree.tree_.feature[0] for tree in rf_model.estimators_])).value_counts()
    labeled_random_forest_top_feature_counts = \
        label_top_feature_counts(random_forest_top_feature_counts, columns)
    return labeled_random_forest_top_feature_counts

def label_top_feature_counts(top_feature_counts, columns):
    top_feature_count_labels = \
        top_feature_counts.index.to_series().apply(lambda i: columns[i])
    top_feature_counts = pd.concat(
        [top_feature_count_labels.rename("feature"), top_feature_counts.rename("count")],
        axis=1
    )
    return top_feature_counts.set_index("feature")
```

In [68]:

```
rf_model, rf_results_train, rf_results_test = create_and_evaluate_model(
    GridSearchCV(
        RandomForestClassifier(n_estimators=100),
        {
            "max_depth": range(3, 20)
        },
        cv=3
    ).fit(X_train, y_train).best_estimator_,
    X_train,
    y_train,
    X_test,
    y_test
)
plot_model_results(rf_results_train, "Random Forest (Train)")
plot_model_results(rf_results_test, "Random Forest (Test)")
display(HTML("<h4>Top Feature Counts for Random Forest</h4>"))
display(rf_top_feature_counts(rf_model, X_train.columns))
```



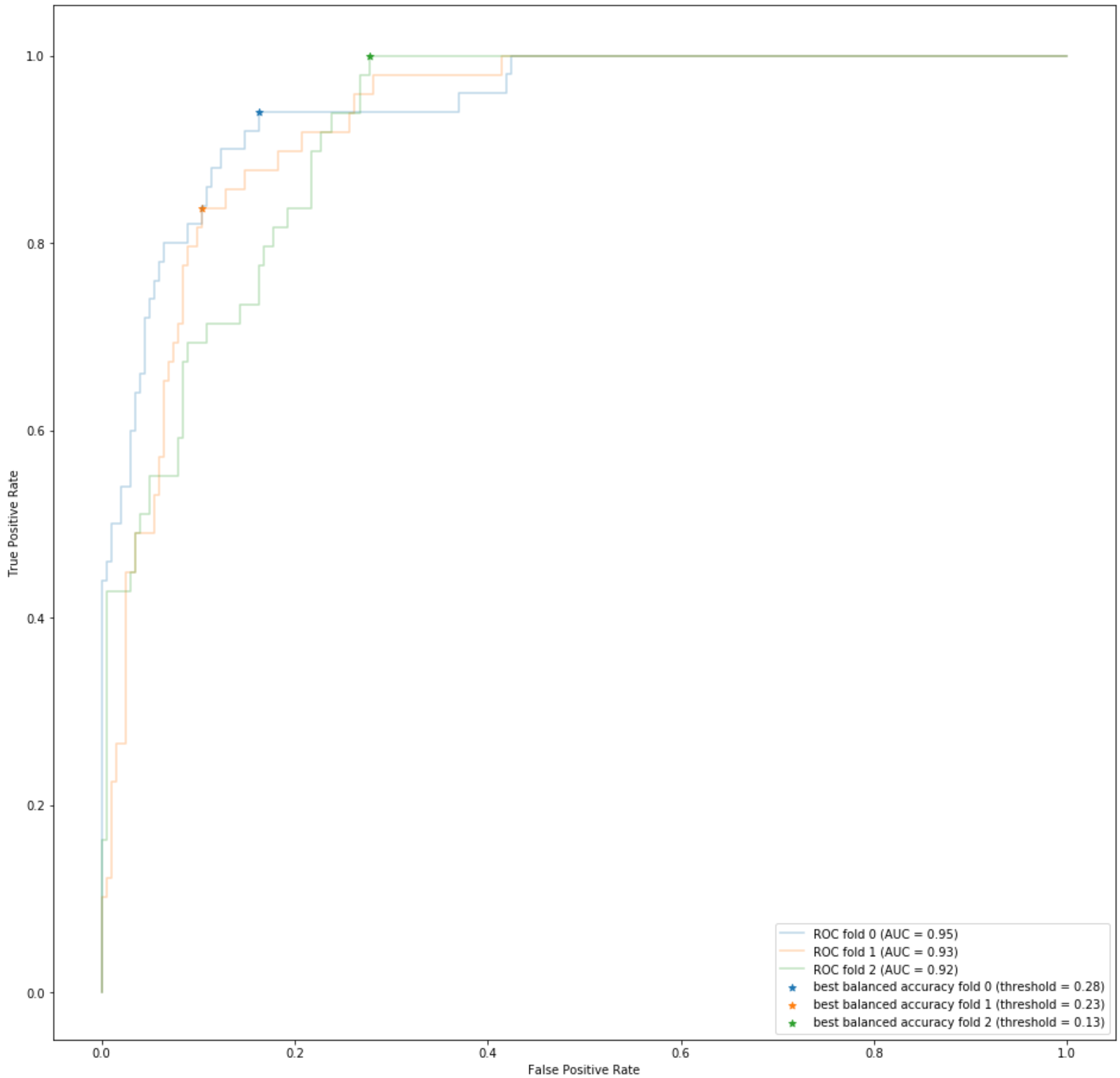


Top Feature Counts for Random Forest

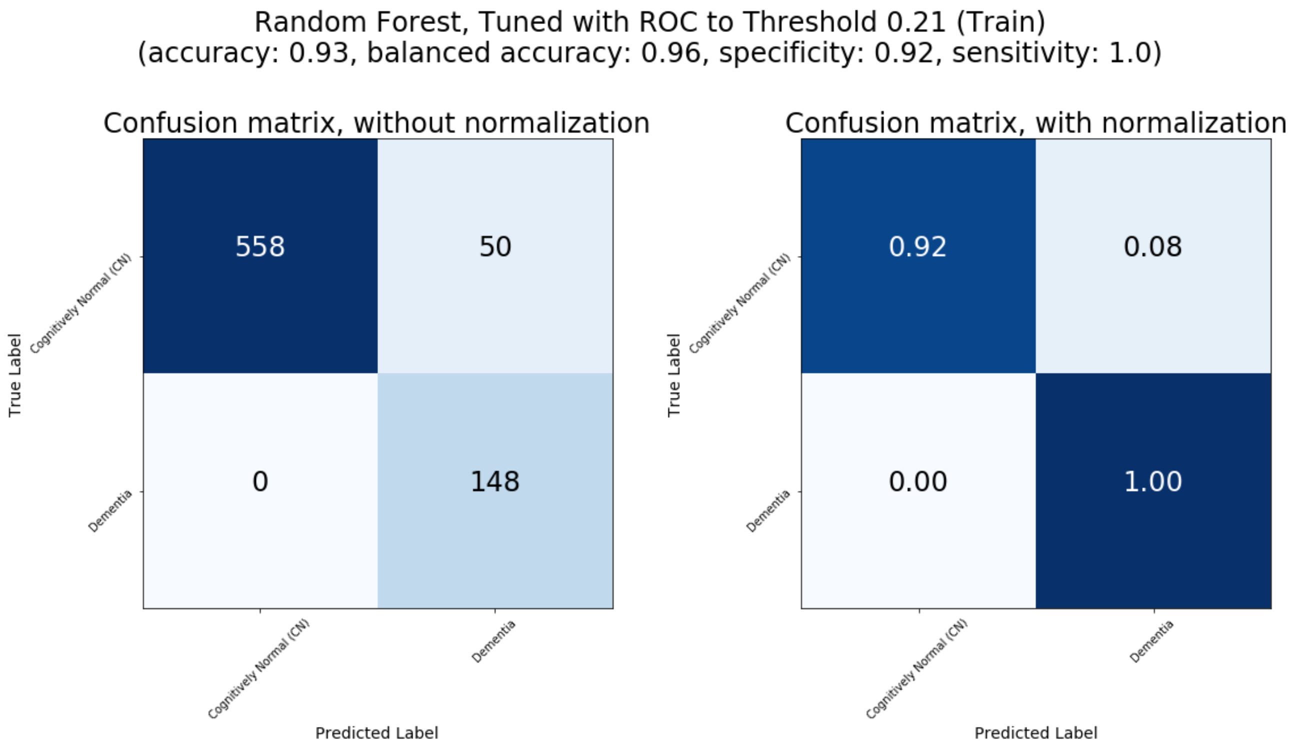
feature	count
FAQ	12
ADAS13	8
RAVLT_learning	8
Hippocampus	8
RAVLT_perc_forgetting	8
ADAS11	7
RAVLT_immediate	7
FDG	5
CDRSB	5
Entorhinal	5
MMSE	4
Ventricles	3
APOE4	3
WholeBrain	3
Fusiform	3
MidTemp	2
13711	2
TOMM40_A2	2
9989	1
RAVLT_forgetting	1
3308	1
16679	1
31790	1

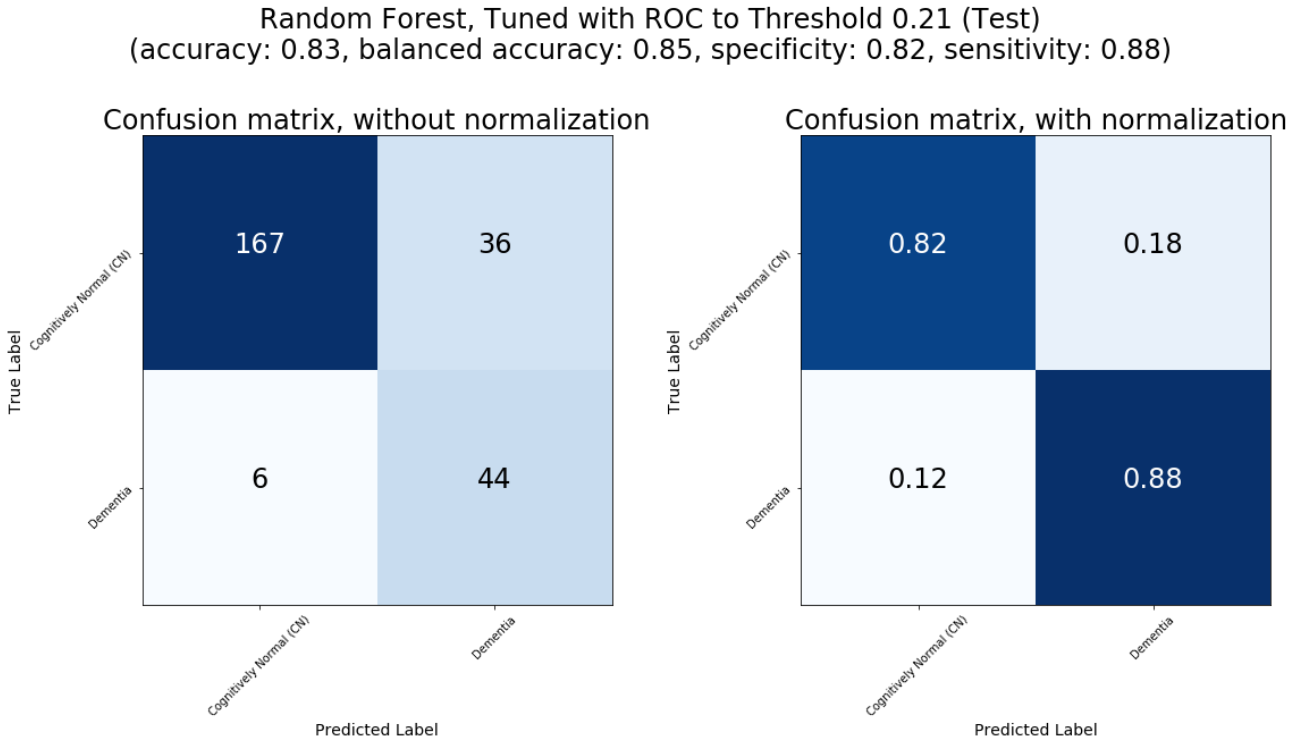
Adjusting Random Forest with ROC

In [69]:
```
rf_augmented_roc_curves = augmented_roc_curves_cv(
    rf_model,
    X_train,
    y_train
)
rf_roc_tuned_threshold, _ = mean_threshold_best_balanced_accuracy(rf_augmented_roc_curves)
plot_augmented_roc_curves(rf_augmented_roc_curves)
```



In [70]:
```
rf_roc_tuned_model, rf_roc_tuned_results_train, rf_roc_tuned_results_test = \
    create_and_evaluate_probabilistic_model(
        rf_model,
        X_train,
        y_train,
        X_test,
        y_test,
        rf_roc_tuned_threshold
    )
plot_model_results(
    rf_roc_tuned_results_train,
    f"Random Forest, Tuned with ROC to Threshold {rf_roc_tuned_threshold:.2} (Train)"
)
plot_model_results(
    rf_roc_tuned_results_test,
    f"Random Forest, Tuned with ROC to Threshold {rf_roc_tuned_threshold:.2} (Test)"
)
display(HTML(f"<h4>Top Feature Counts for Random Forest, Tuned with ROC to Threshold {rf_roc_tuned_thres
display(rf_top_feature_counts(rf_roc_tuned_model, X_train.columns))
```



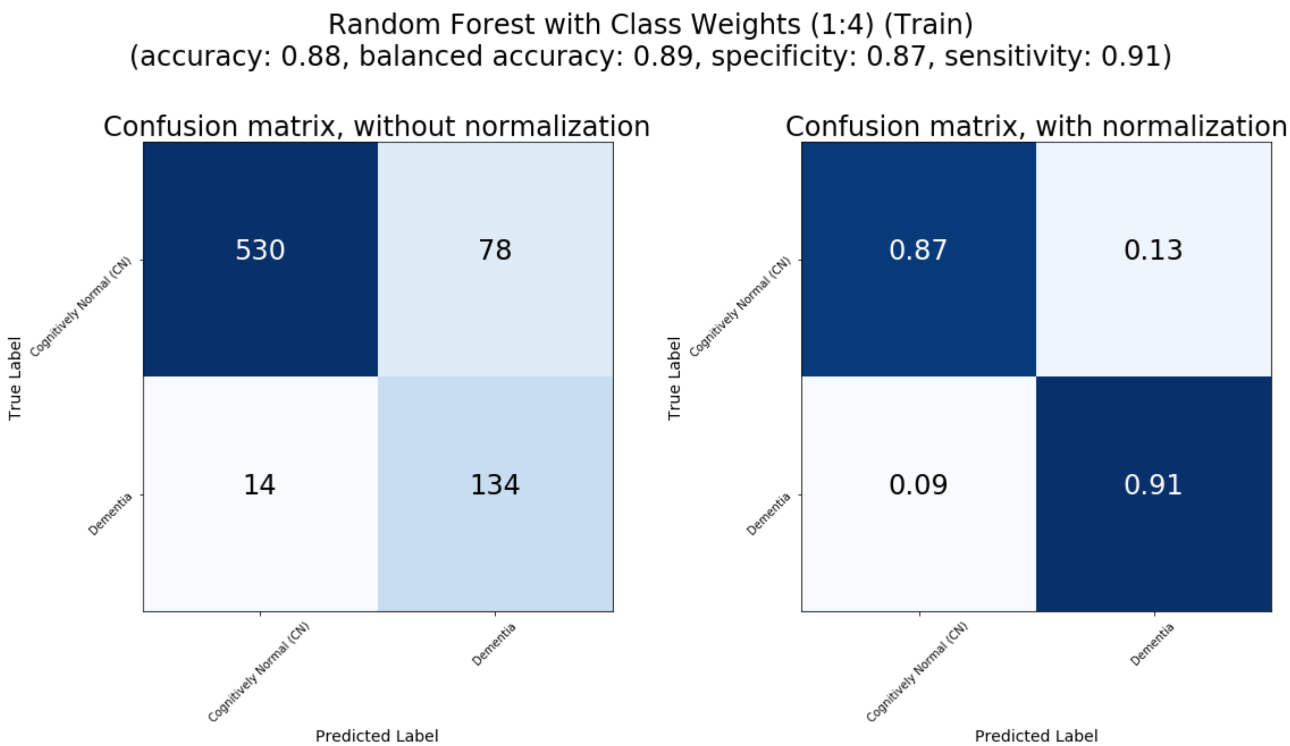


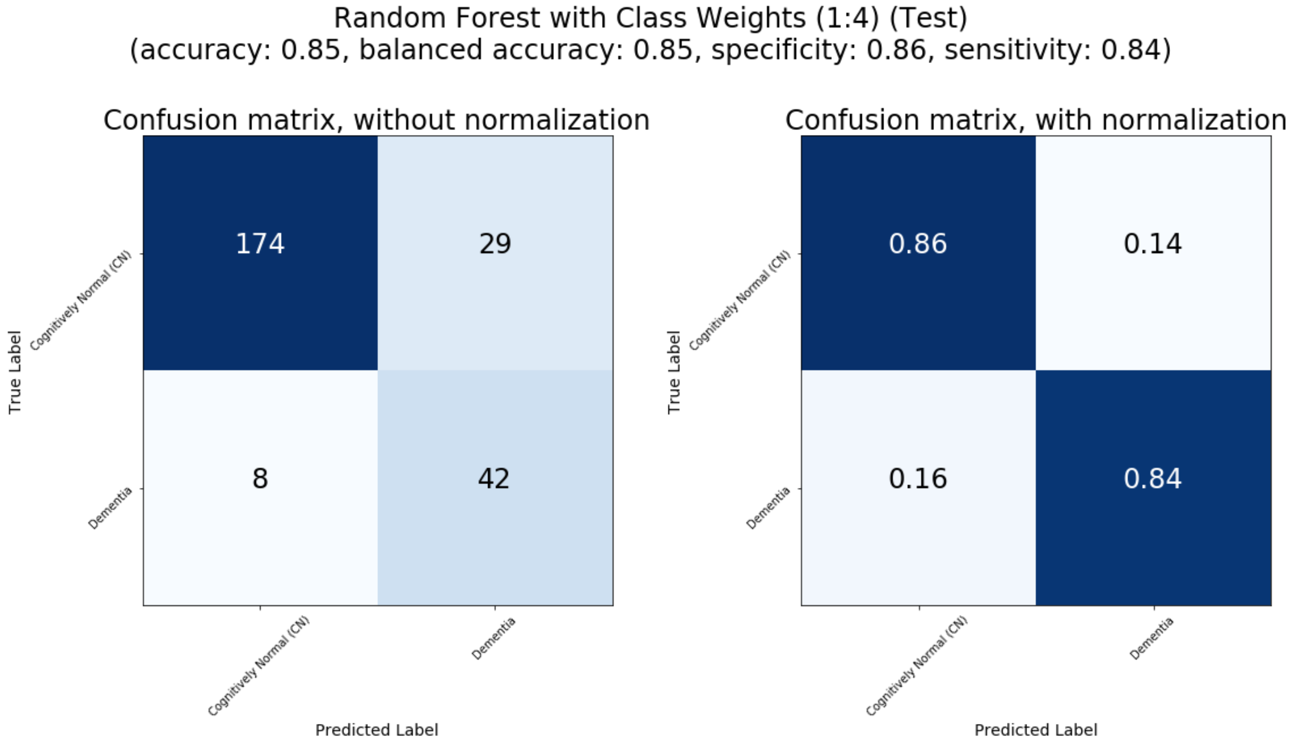
Top Feature Counts for Random Forest, Tuned with ROC to Threshold 0.21

feature	count
FAQ	20
CDRSB	13
ADAS11	9
ADAS13	8
RAVLT_immediate	8
RAVLT_perc_forgetting	5
MMSE	4
APOE4	4
Entorhinal	4
WholeBrain	4
Hippocampus	4
RAVLT_learning	4
FDG	3
AGE	2
Fusiform	2
13711	2
9989	1
Ventricles	1
MidTemp	1
TOMM40_A2	1

Adjusting Random Forest with Class Weights

In [71]:
```python
rf_weighted_model, rf_weighted_results_train, rf_weighted_results_test = \
    create_and_evaluate_model(
        GridSearchCV(
            RandomForestClassifier(n_estimators=100, class_weight={0: 1, 1: 4}),
            {
                "max_depth": range(3, 20)
            },
            scoring="balanced_accuracy",
            cv=3
        ).fit(X_train, y_train).best_estimator_,
        X_train,
        y_train,
        X_test,
        y_test
    )
plot_model_results(rf_weighted_results_train, "Random Forest with Class Weights (1:4) (Train)")
plot_model_results(rf_weighted_results_test, "Random Forest with Class Weights (1:4) (Test)")
display(HTML(f"<h4>Top Feature Counts for Random Forest with Class Weights (1:4)</h4>"))
display(rf_top_feature_counts(rf_weighted_model, X_train.columns))
```





Top Feature Counts for Random Forest with Class Weights (1:4)

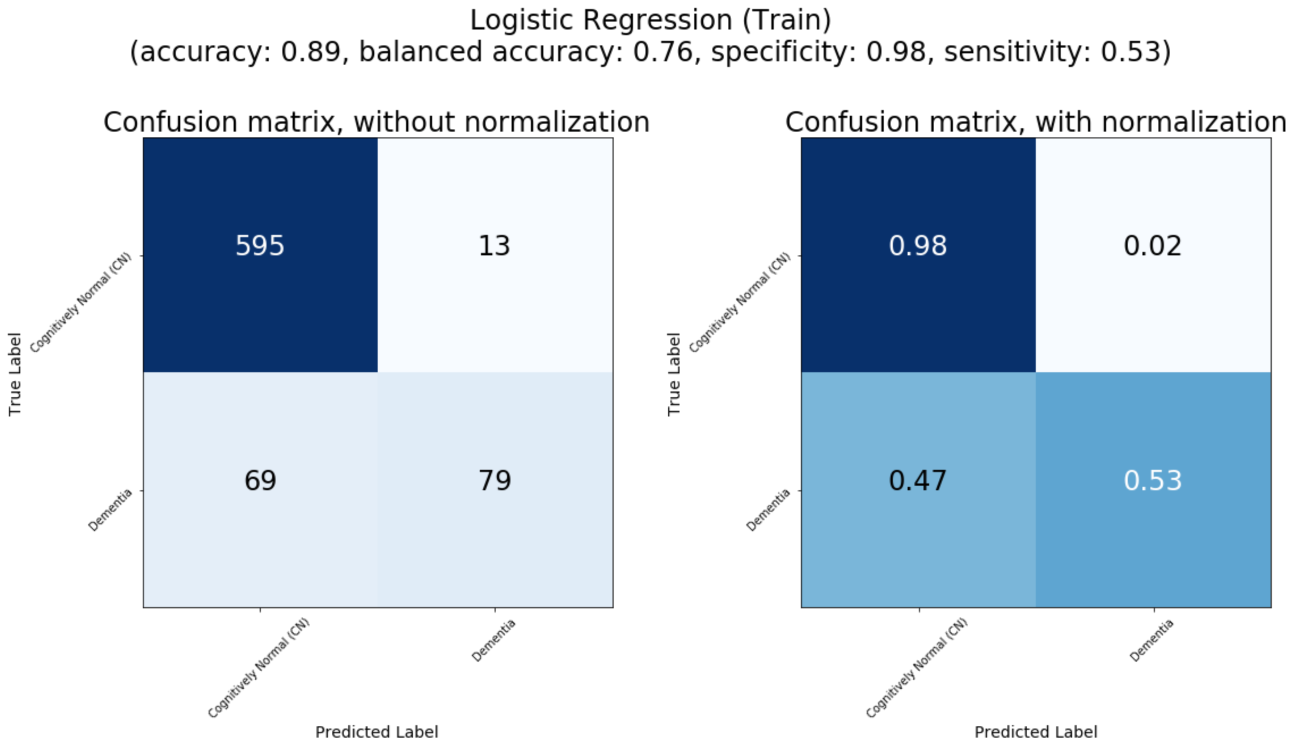
feature	count
RAVLT_immediate	14
FAQ	11
CDRSB	10
ADAS11	9
ADAS13	9
MMSE	6
RAVLT_learning	6
RAVLT_perc_forgetting	5
Hippocampus	5
FDG	4
Fusiform	4
Ventricles	2
RAVLT_forgetting	2
APOE4	2
Entorhinal	2
4203	2
21323	1
MidTemp	1
13711	1
16679	1
31790	1
38066	1
TOMM40_A2	1

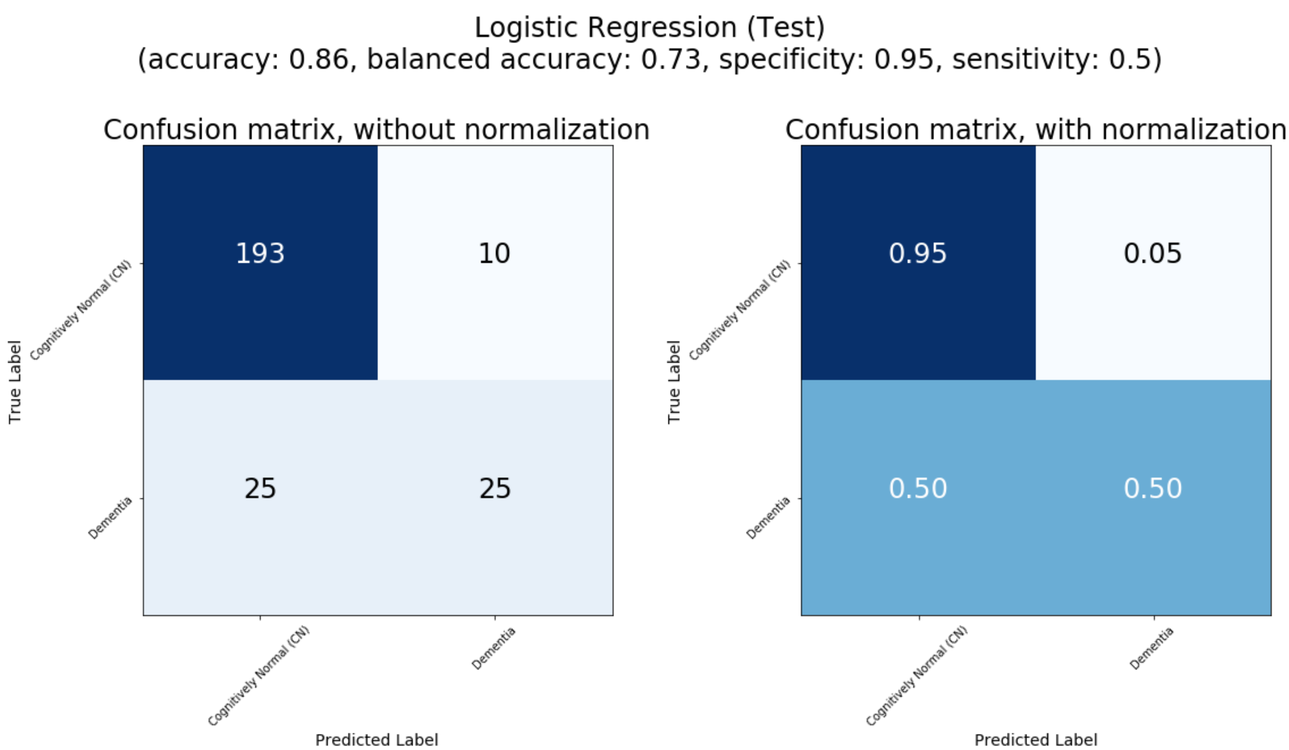
```
In [ ]:
```

```
In [ ]:
```

Logistic Regression

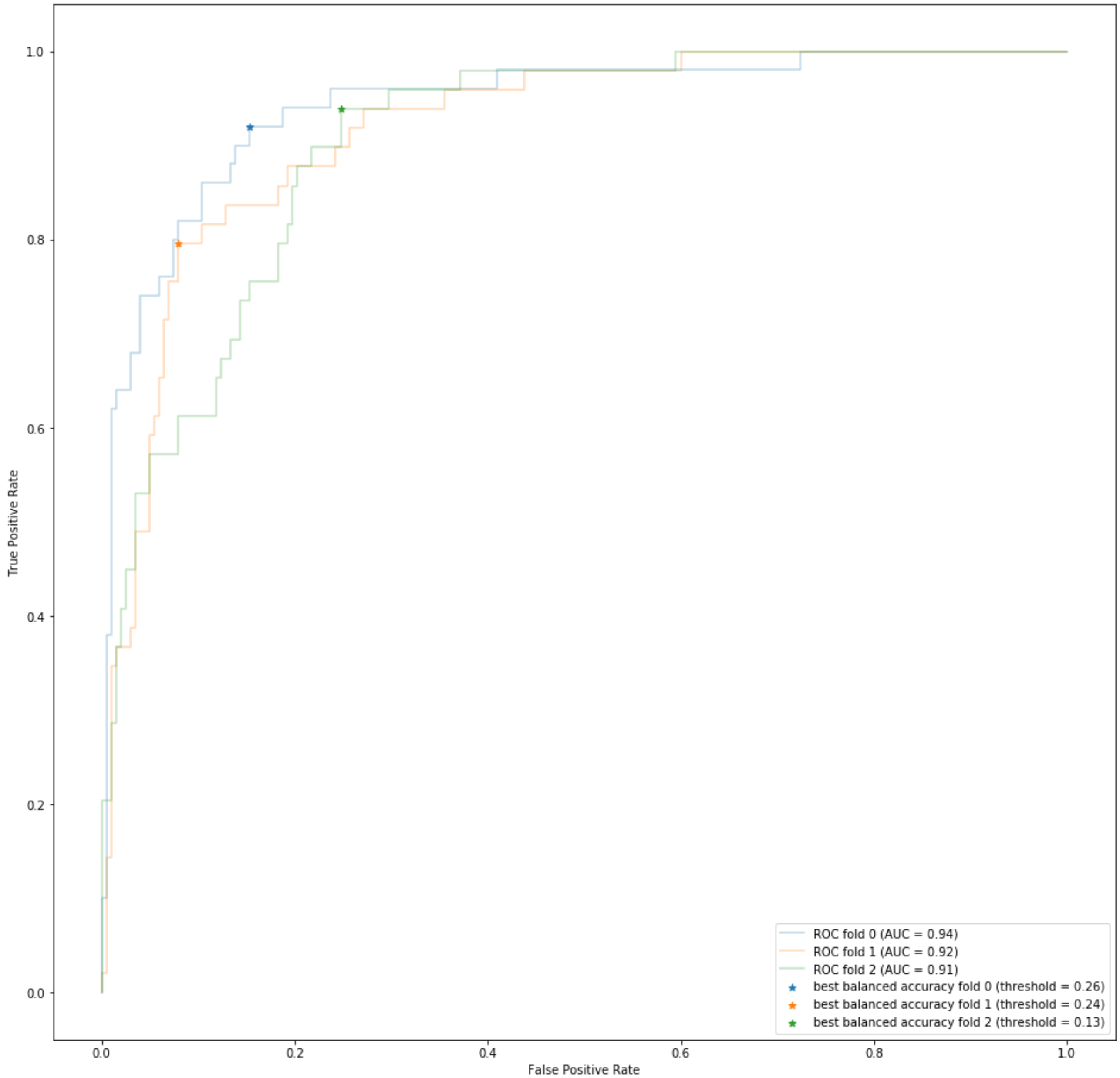
```
In [72]: logreg_model, logreg_results_train, logreg_results_test = create_and_evaluate_model(
             LogisticRegressionCV(max_iter=1000).fit(X_train, y_train),
             X_train,
             y_train,
             X_test,
             y_test
         )
         plot_model_results(logreg_results_train, "Logistic Regression (Train)")
         plot_model_results(logreg_results_test, "Logistic Regression (Test)")
```



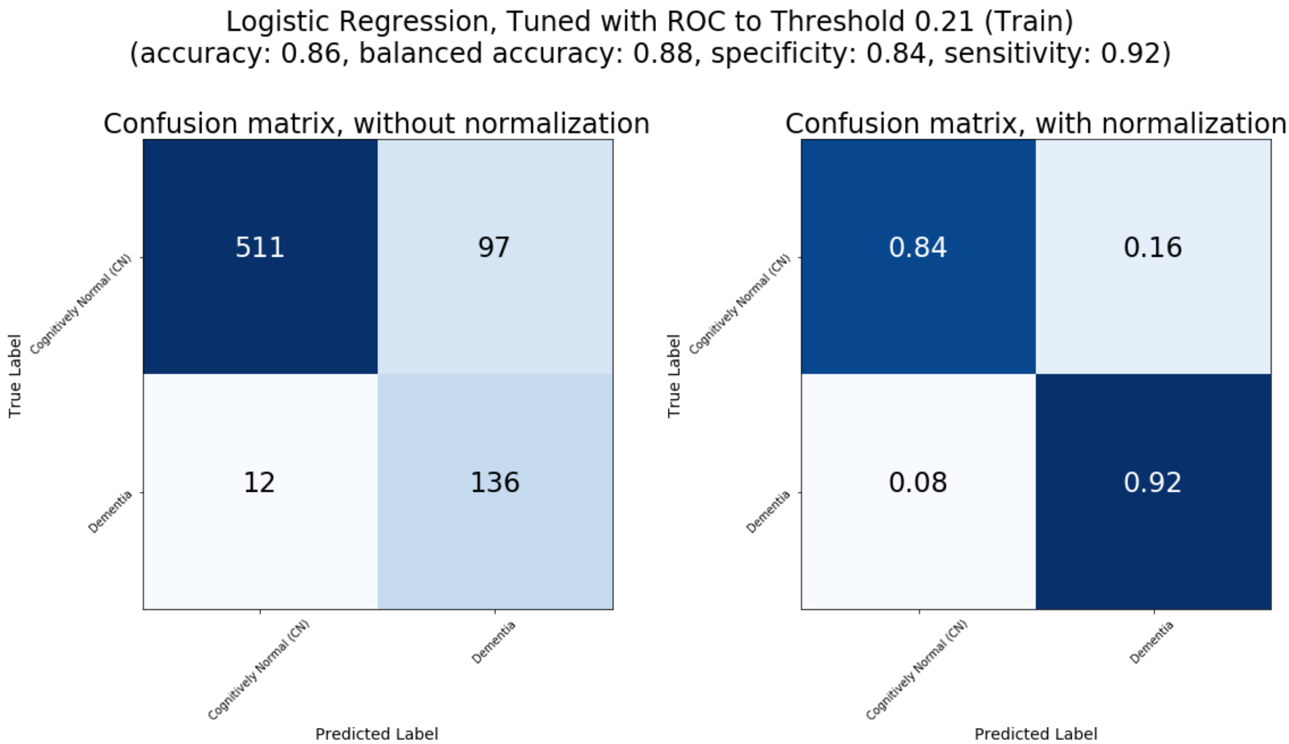


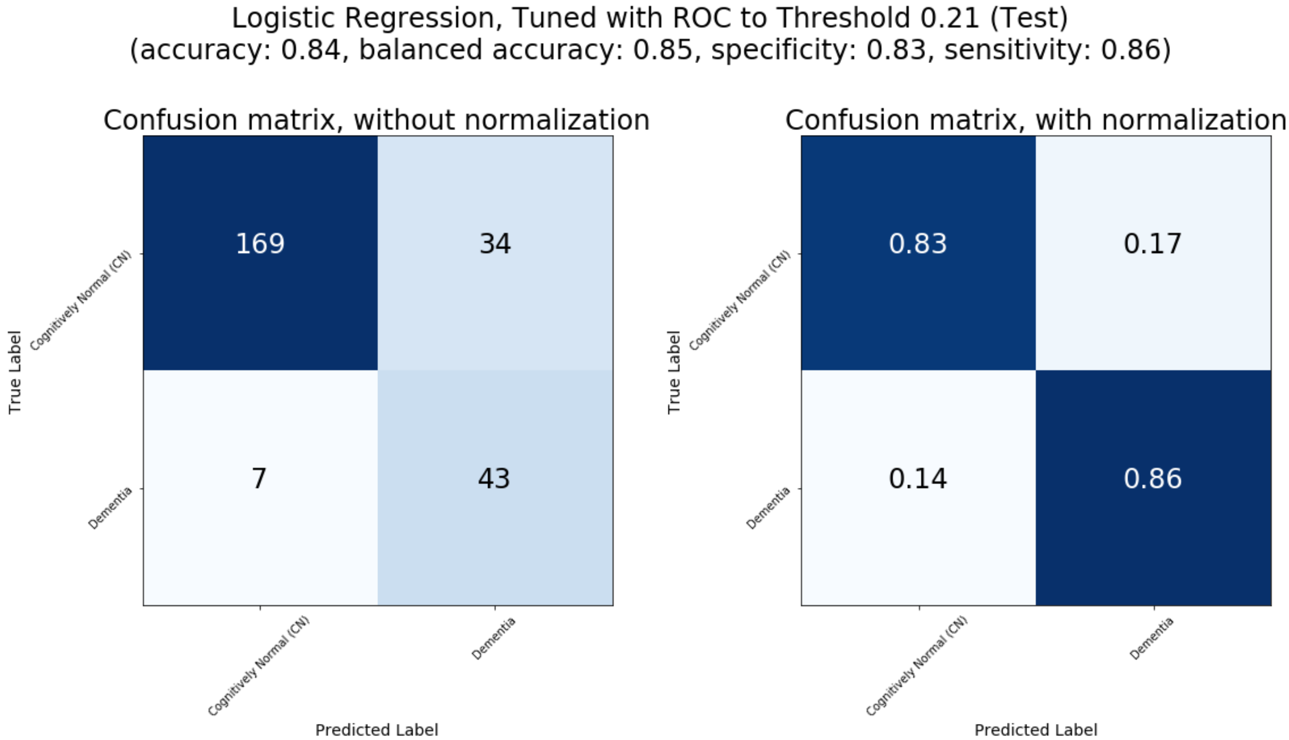
Adjusting Logistic Regression with ROC

In [73]:
```
logreg_augmented_roc_curves = augmented_roc_curves_cv(
    logreg_model,
    X_train,
    y_train
)
logreg_roc_tuned_threshold, _ = mean_threshold_best_balanced_accuracy(logreg_augmented_roc_curves)
plot_augmented_roc_curves(logreg_augmented_roc_curves)
```



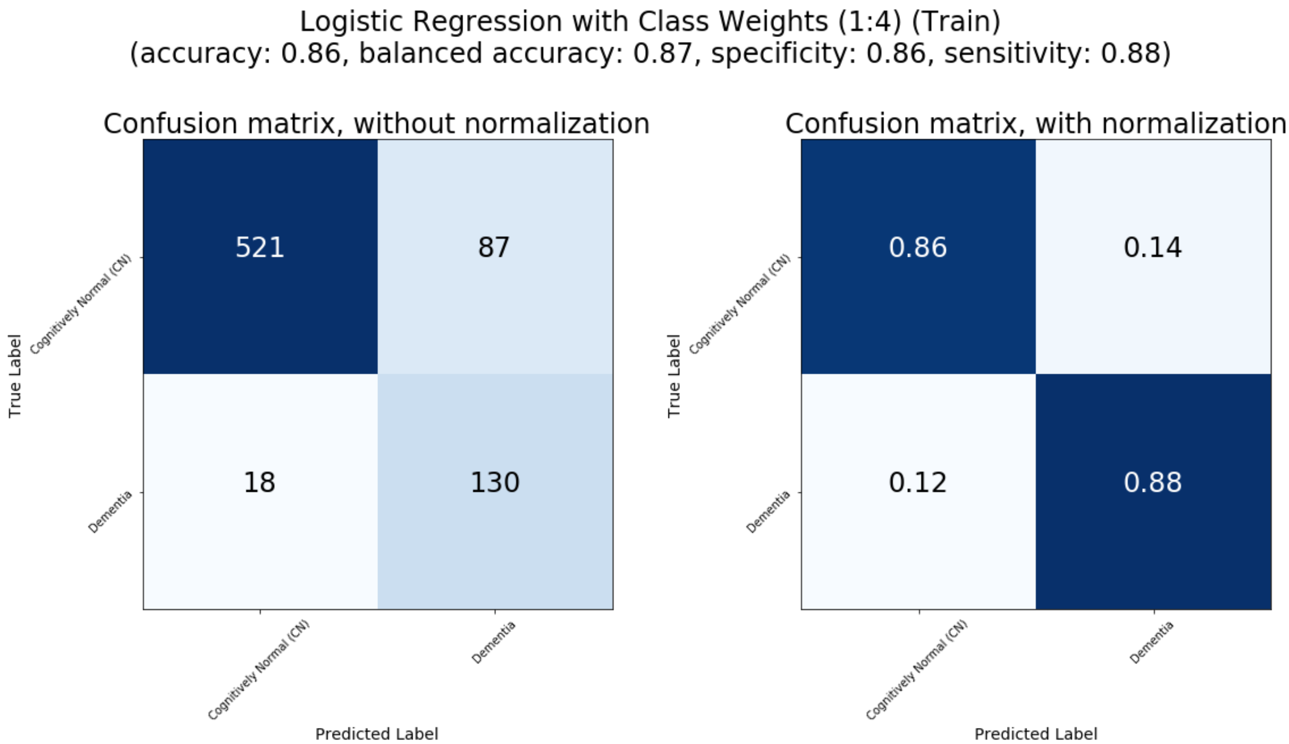
In [74]:
```
logreg_roc_tuned_model, logreg_roc_tuned_results_train, logreg_roc_tuned_results_test = \
    create_and_evaluate_probabilistic_model(
        logreg_model,
        X_train,
        y_train,
        X_test,
        y_test,
        logreg_roc_tuned_threshold
    )
plot_model_results(
    logreg_roc_tuned_results_train,
    f"Logistic Regression, Tuned with ROC to Threshold {logreg_roc_tuned_threshold:.2} (Train)"
)
plot_model_results(
    logreg_roc_tuned_results_test,
    f"Logistic Regression, Tuned with ROC to Threshold {logreg_roc_tuned_threshold:.2} (Test)"
)
```

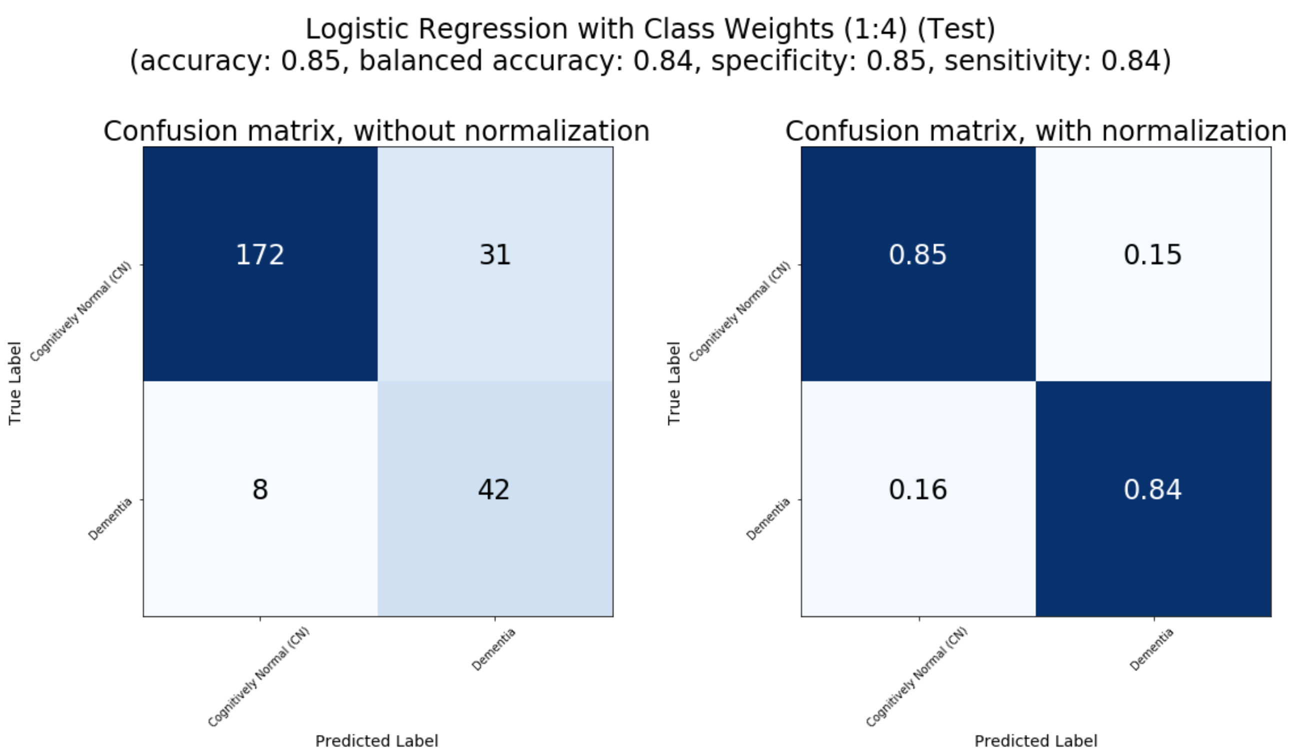




Adjusting Logistic Regression with Class Weights

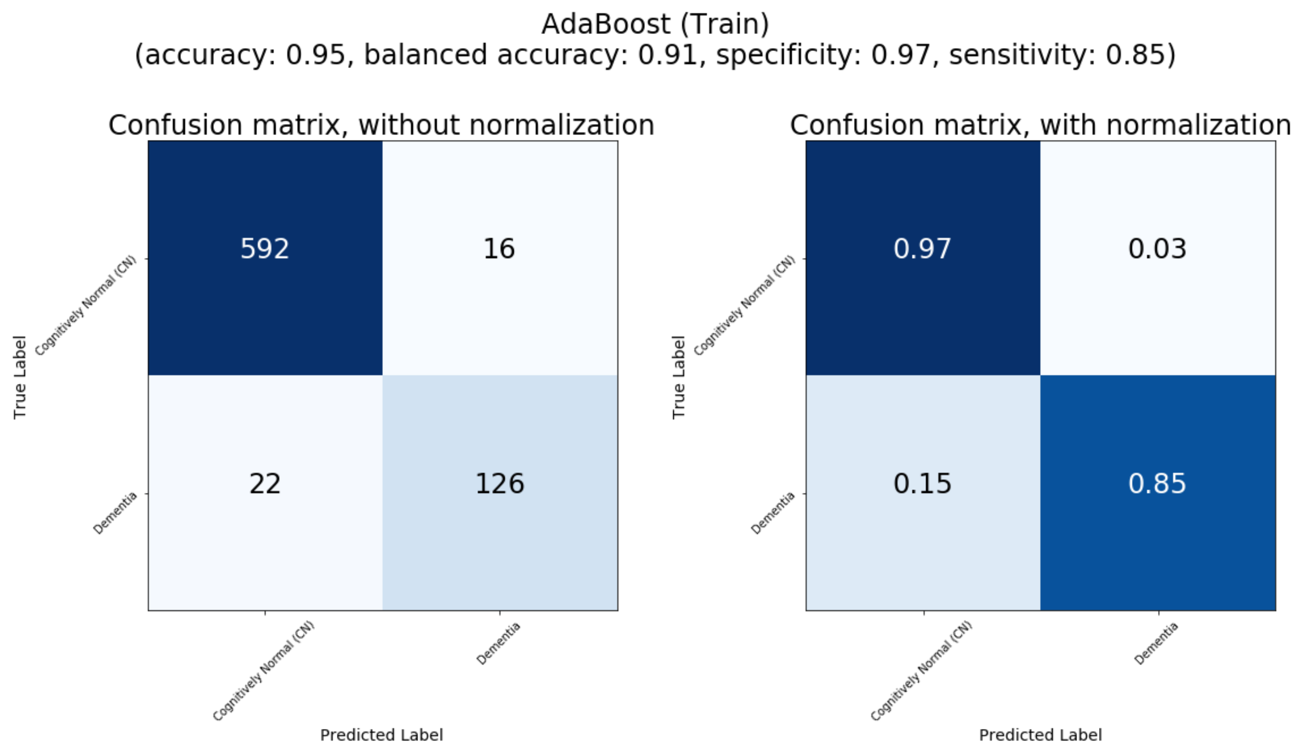
In [75]:
```python
logreg_weighted_model, logreg_weighted_results_train, logreg_weighted_results_test = \
    create_and_evaluate_model(
        LogisticRegressionCV(
            max_iter=1000,
            class_weight={0: 1, 1: 4},
            scoring="balanced_accuracy"
        ).fit(X_train, y_train),
        X_train,
        y_train,
        X_test,
        y_test
    )
plot_model_results(logreg_weighted_results_train, "Logistic Regression with Class Weights (1:4) (Train)"
plot_model_results(logreg_weighted_results_test, "Logistic Regression with Class Weights (1:4) (Test)")
```

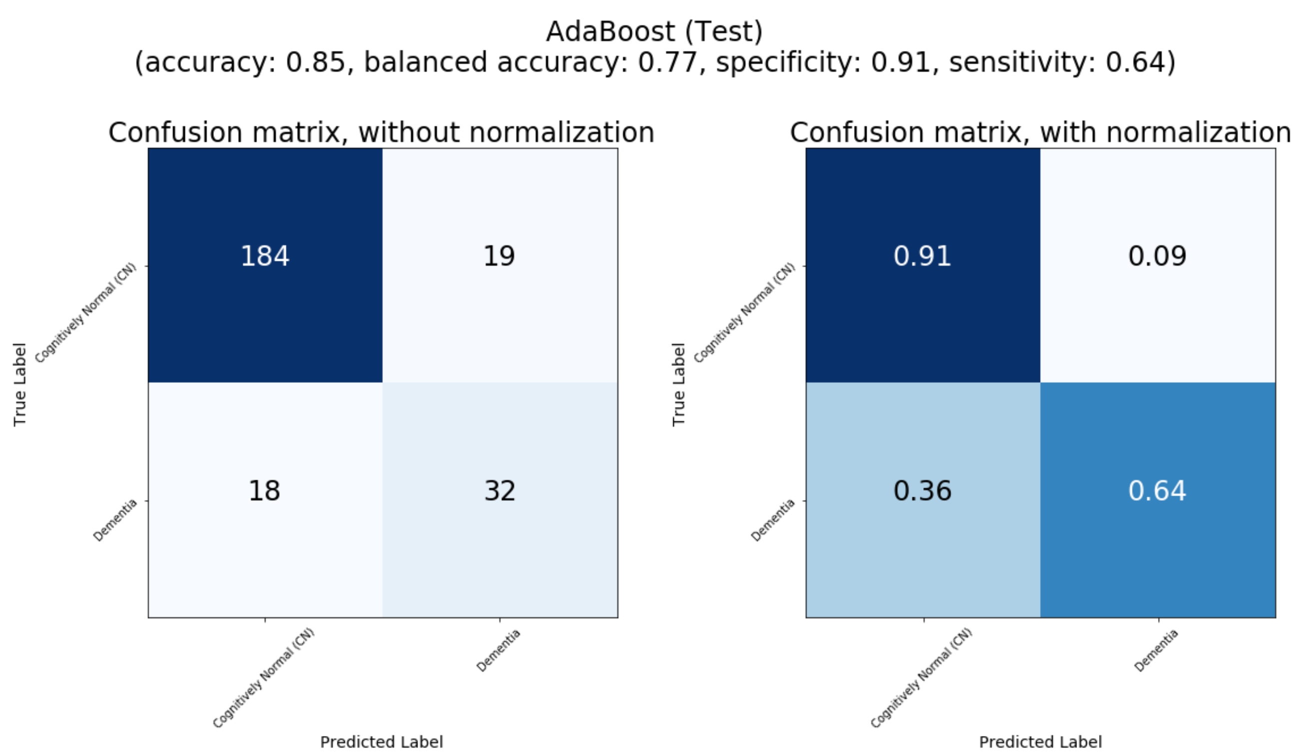




AdaBoost

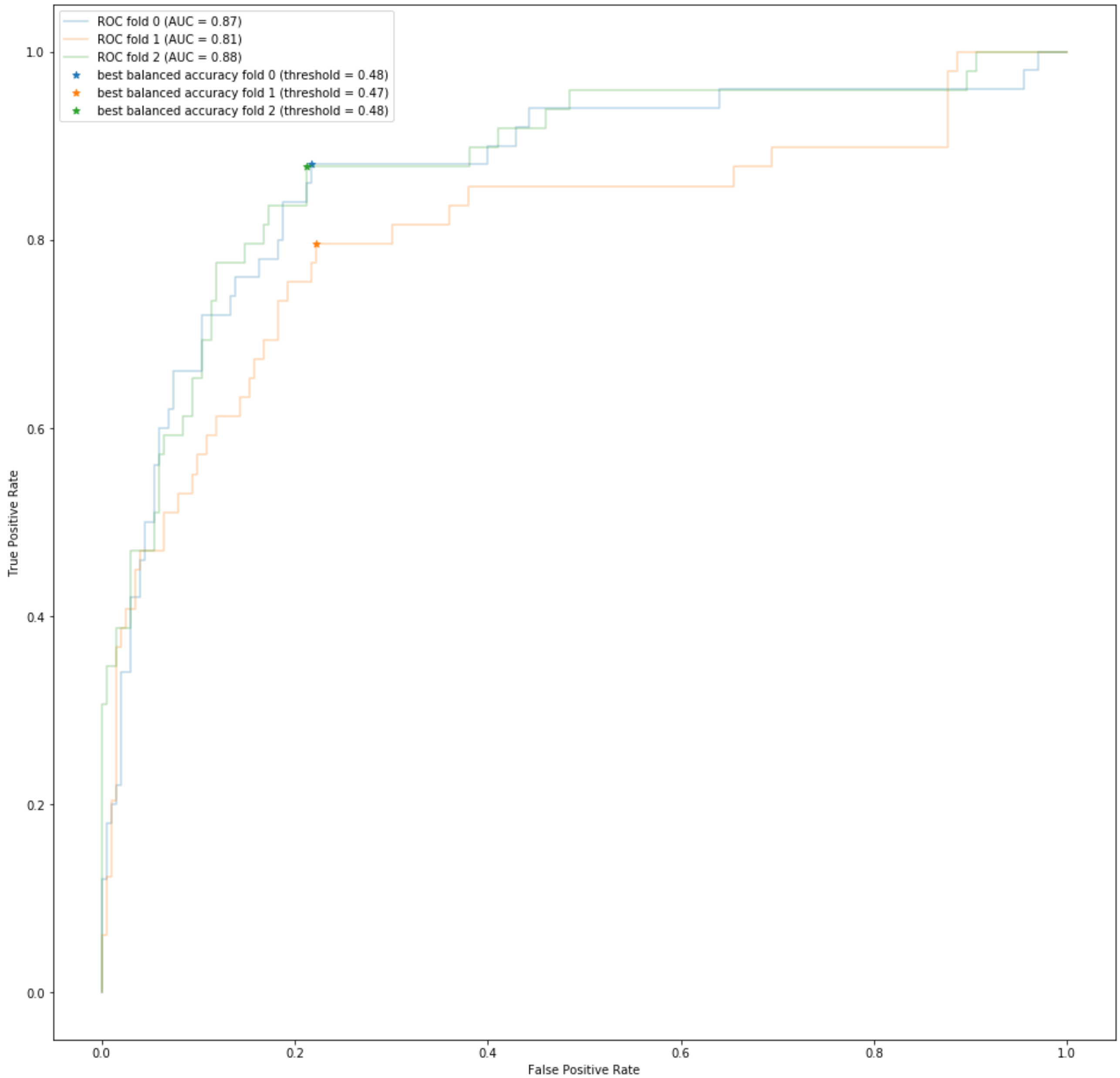
In [76]:
```
adaboost_model, adaboost_results_train, adaboost_results_test = create_and_evaluate_model(
    GridSearchCV(
        AdaBoostClassifier(base_estimator=DecisionTreeClassifier()),
        {
            "base_estimator__max_depth": range(1, 5)
        },
        cv=3
    ).fit(X_train, y_train).best_estimator_,
    X_train,
    y_train,
    X_test,
    y_test
)
plot_model_results(adaboost_results_train, f"AdaBoost (Train)")
plot_model_results(adaboost_results_test, f"AdaBoost (Test)")
```



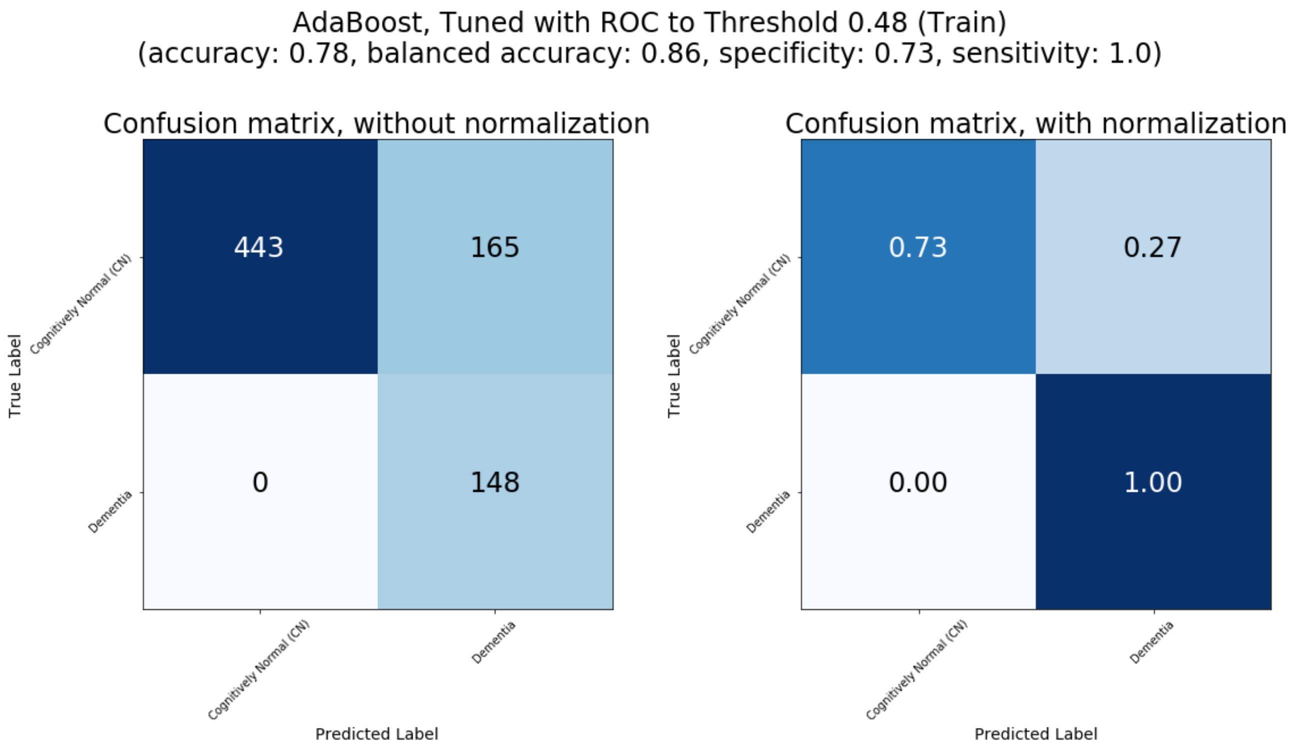


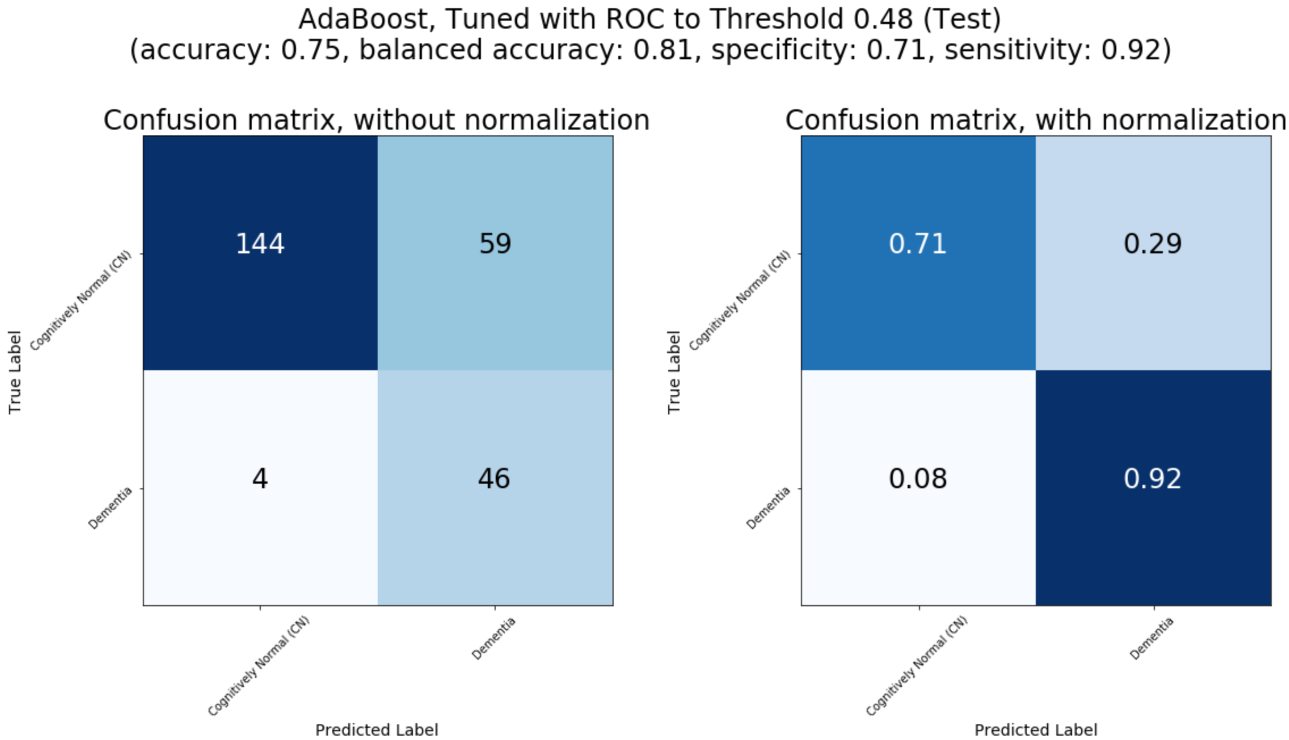
Adjusting AdaBoost with ROC

In [77]:
```
adaboost_augmented_roc_curves = augmented_roc_curves_cv(
    adaboost_model,
    X_train,
    y_train
)
adaboost_roc_tuned_threshold, _ = mean_threshold_best_balanced_accuracy(adaboost_augmented_roc_curves)
plot_augmented_roc_curves(adaboost_augmented_roc_curves)
```



In [78]:
```
adaboost_roc_tuned_model, adaboost_roc_tuned_results_train, adaboost_roc_tuned_results_test = \
    create_and_evaluate_probabilistic_model(
        adaboost_model,
        X_train,
        y_train,
        X_test,
        y_test,
        adaboost_roc_tuned_threshold
    )
plot_model_results(
    adaboost_roc_tuned_results_train,
    f"AdaBoost, Tuned with ROC to Threshold {adaboost_roc_tuned_threshold:.2} (Train)"
)
plot_model_results(
    adaboost_roc_tuned_results_test,
    f"AdaBoost, Tuned with ROC to Threshold {adaboost_roc_tuned_threshold:.2} (Test)"
)
```



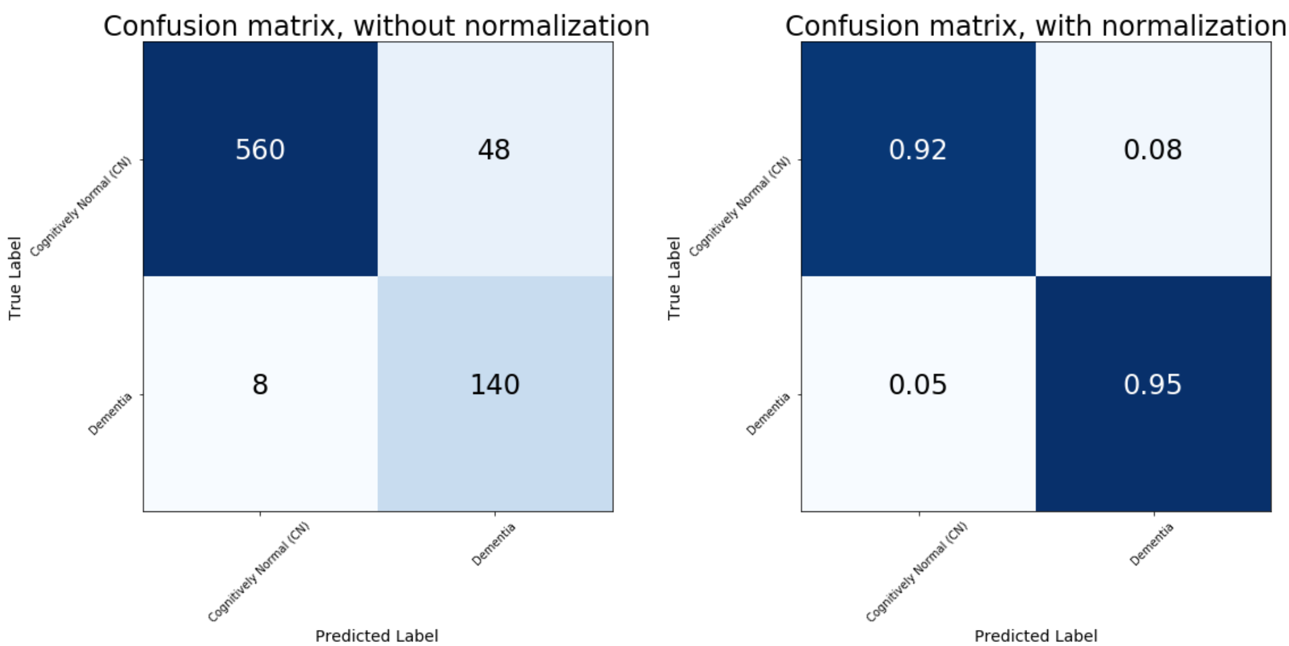


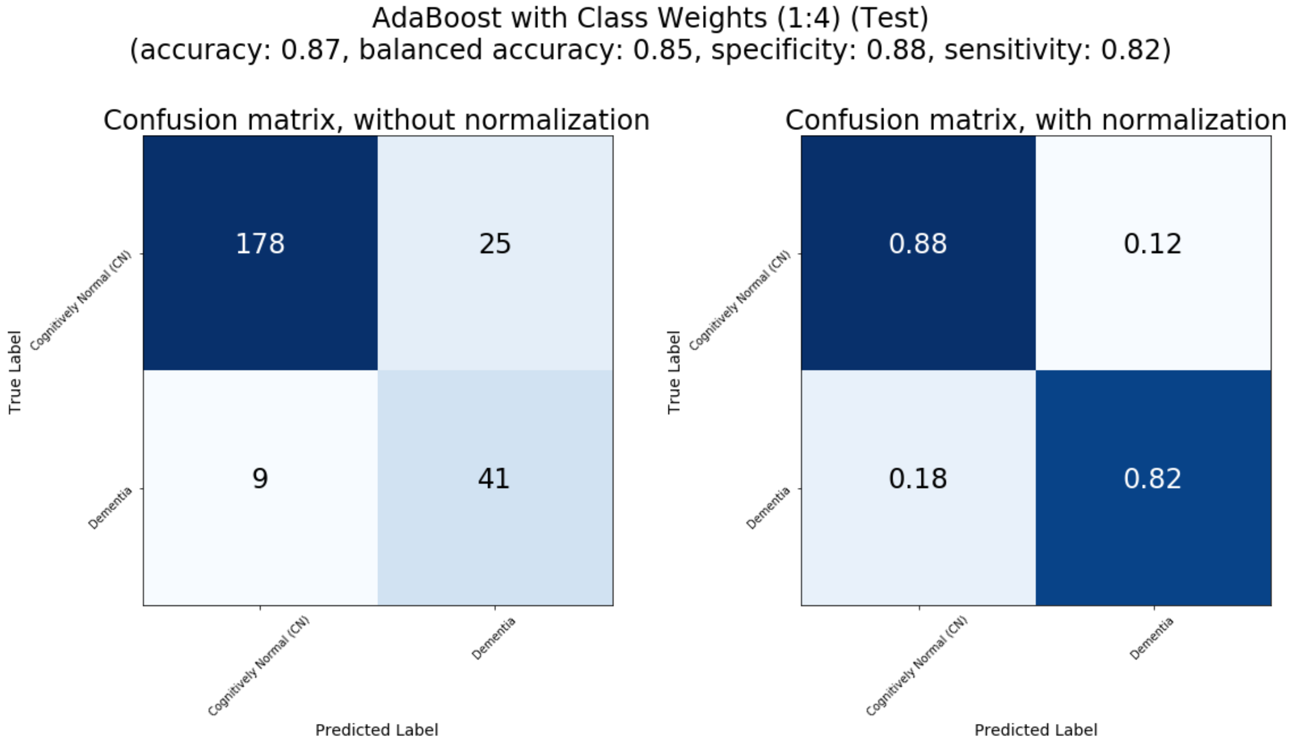
Adjusting AdaBoost with Class Weights

Since AdaBoost already deliberately focuses on misclassified points on each iteration, it effectively does some sort of weighting. Furthermore, in sklearn, `AdaBoostClassifier` per se does not take any class weight parameter; it would have to be applied to the boosted estimators, which are shallow decision trees.

In [79]:
```
adaboost_weighted_model, adaboost_weighted_results_train, adaboost_weighted_results_test = \
    create_and_evaluate_model(
        GridSearchCV(
            AdaBoostClassifier(
                base_estimator=DecisionTreeClassifier(class_weight={0: 1, 1: 4})
            ),
            {
                "base_estimator__max_depth": range(1, 5)
            },
            scoring="balanced_accuracy",
            cv=3
        ).fit(X_train, y_train).best_estimator_,
        X_train,
        y_train,
        X_test,
        y_test
    )
plot_model_results(adaboost_weighted_results_train, "AdaBoost with Class Weights (1:4) (Train)")
plot_model_results(adaboost_weighted_results_test, "AdaBoost with Class Weights (1:4) (Test)")
```

AdaBoost with Class Weights (1:4) (Train)
(accuracy: 0.93, balanced accuracy: 0.93, specificity: 0.92, sensitivity: 0.95)



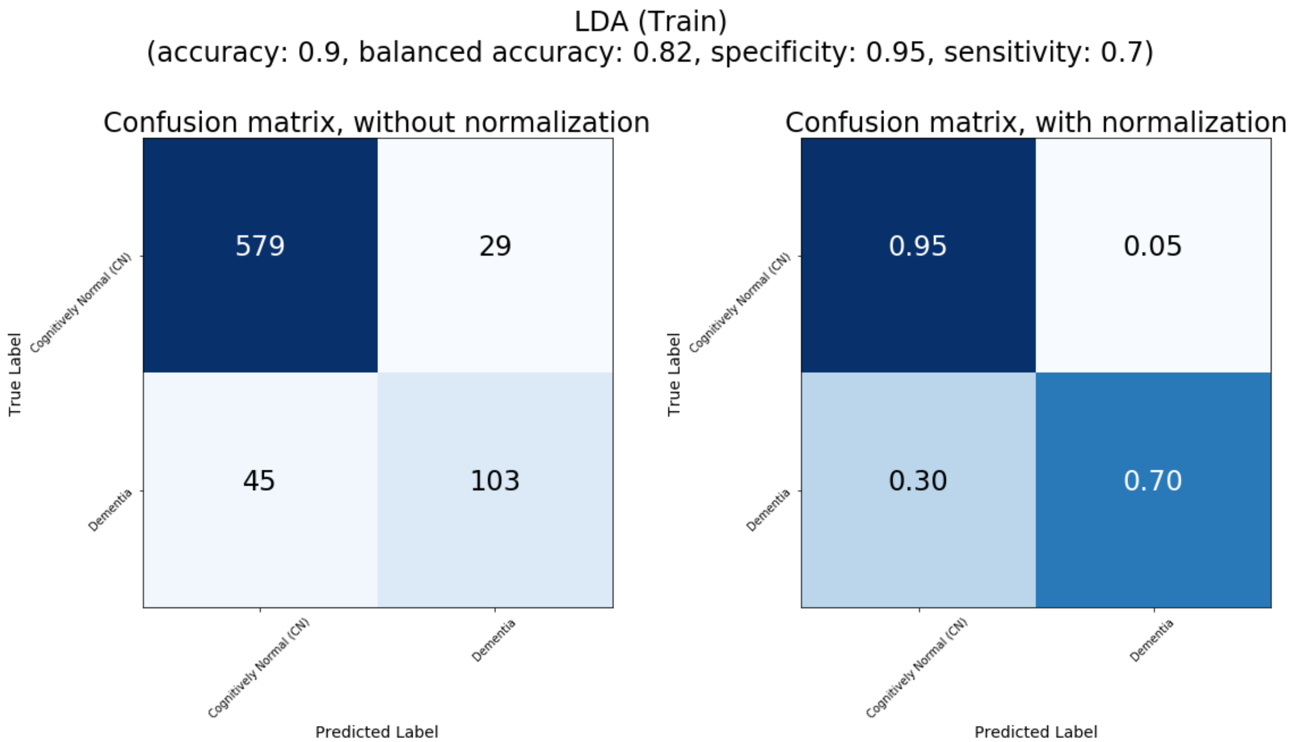


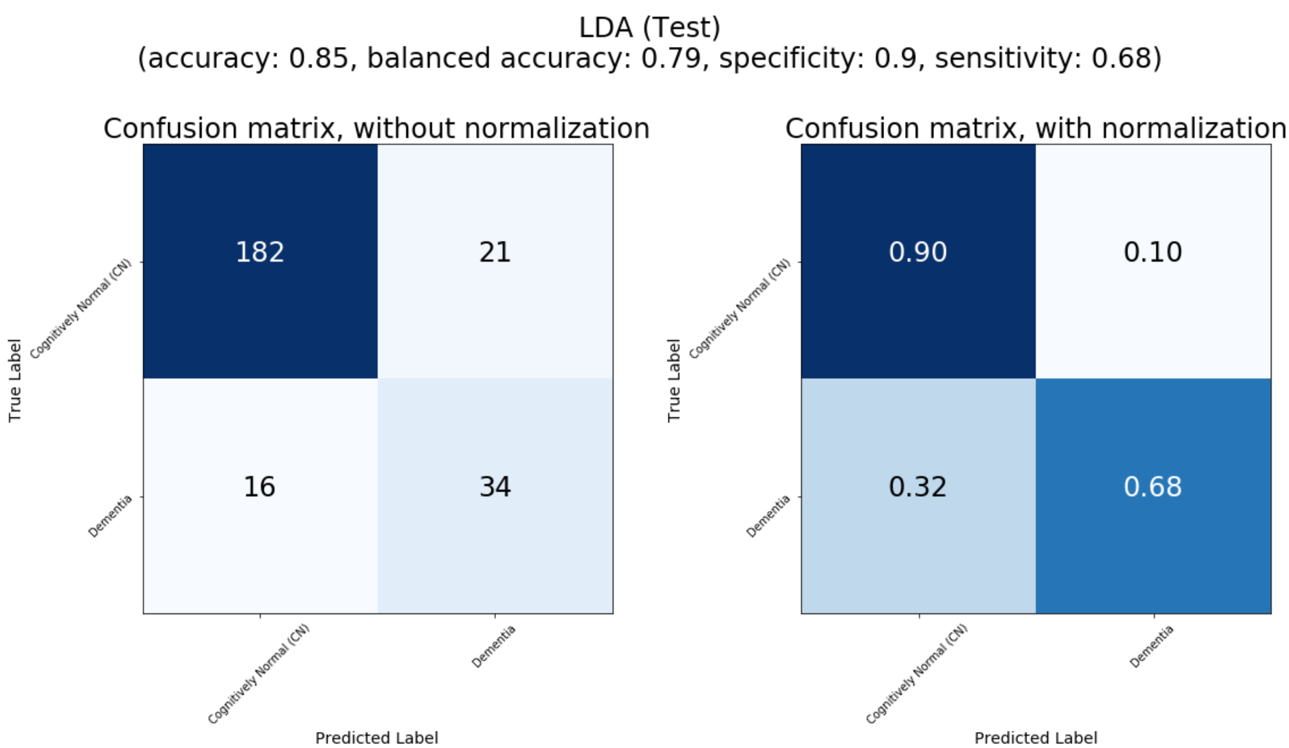

```
In [ ]:
```

Linear Discriminant Analysis

In [80]:
```python
lda_model, lda_results_train, lda_results_test = create_and_evaluate_model(
    LinearDiscriminantAnalysis().fit(X_train, y_train),
    X_train,
    y_train,
    X_test,
    y_test
)
plot_model_results(lda_results_train, "LDA (Train)")
plot_model_results(lda_results_test, "LDA (Test)")
```

```
/usr/local/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWarning: Va
riables are collinear.
  warnings.warn("Variables are collinear.")
/usr/local/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWarning: Va
riables are collinear.
  warnings.warn("Variables are collinear.")
```

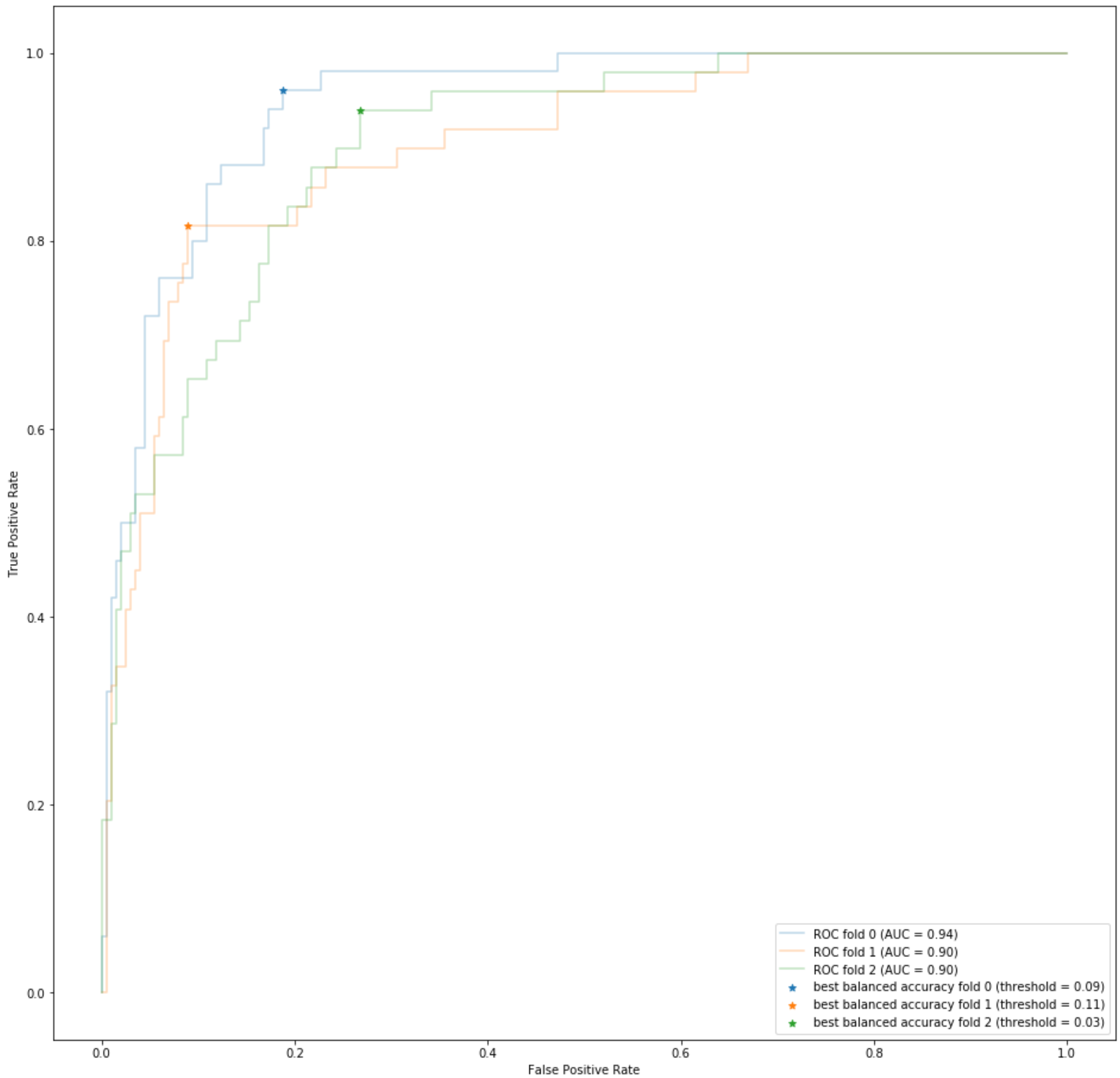




Adjusting LDA with ROC

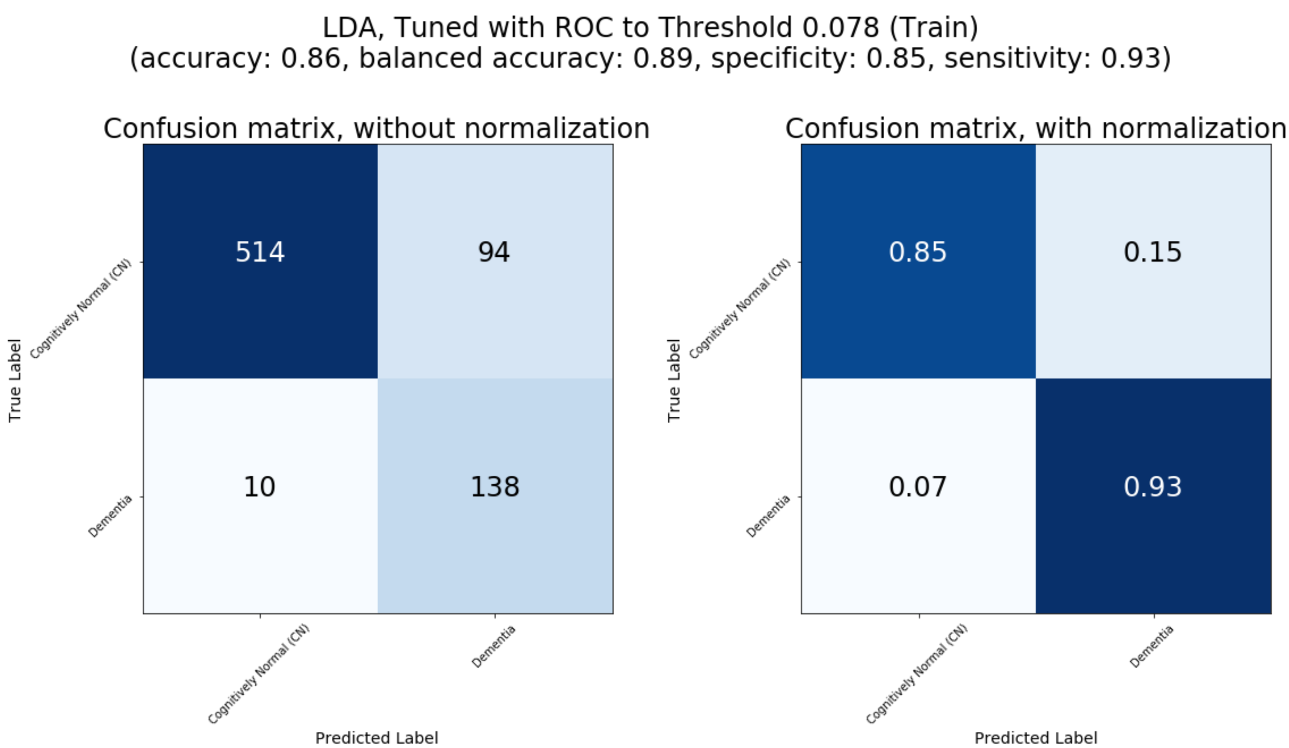
In [81]:
```
lda_augmented_roc_curves = augmented_roc_curves_cv(
    lda_model,
    X_train,
    y_train
)
lda_roc_tuned_threshold, _ = mean_threshold_best_balanced_accuracy(lda_augmented_roc_curves)
plot_augmented_roc_curves(lda_augmented_roc_curves)
```

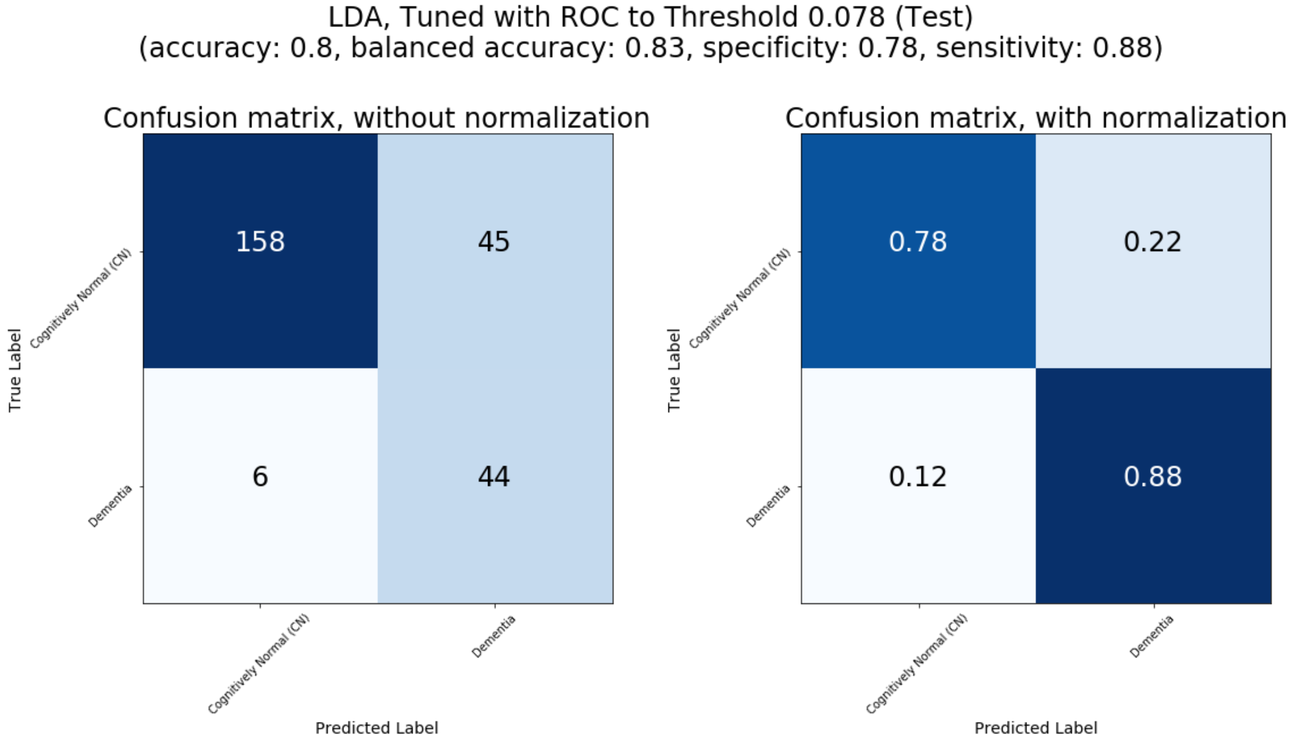
```
/usr/local/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWarning: Va
riables are collinear.
  warnings.warn("Variables are collinear.")
/usr/local/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWarning: Va
riables are collinear.
  warnings.warn("Variables are collinear.")
/usr/local/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWarning: Va
riables are collinear.
  warnings.warn("Variables are collinear.")
```



In [82]:
```
lda_roc_tuned_model, lda_roc_tuned_results_train, lda_roc_tuned_results_test = \
    create_and_evaluate_probabilistic_model(
        lda_model,
        X_train,
        y_train,
        X_test,
        y_test,
        lda_roc_tuned_threshold
    )
plot_model_results(
    lda_roc_tuned_results_train,
    f"LDA, Tuned with ROC to Threshold {lda_roc_tuned_threshold:.2} (Train)"
)
plot_model_results(
    lda_roc_tuned_results_test,
    f"LDA, Tuned with ROC to Threshold {lda_roc_tuned_threshold:.2} (Test)"
)
```

```
/usr/local/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWarning: Va
riables are collinear.
  warnings.warn("Variables are collinear.")
```





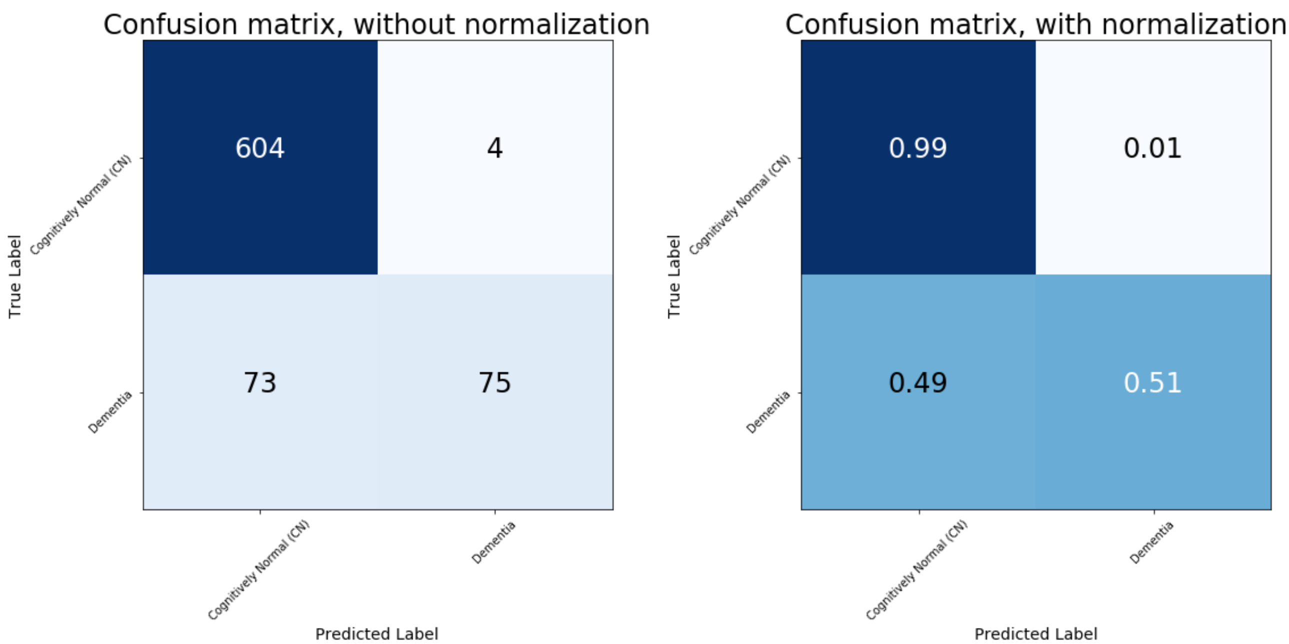
Adjusting LDA with Class Weights

LDA does not take into account class weights.

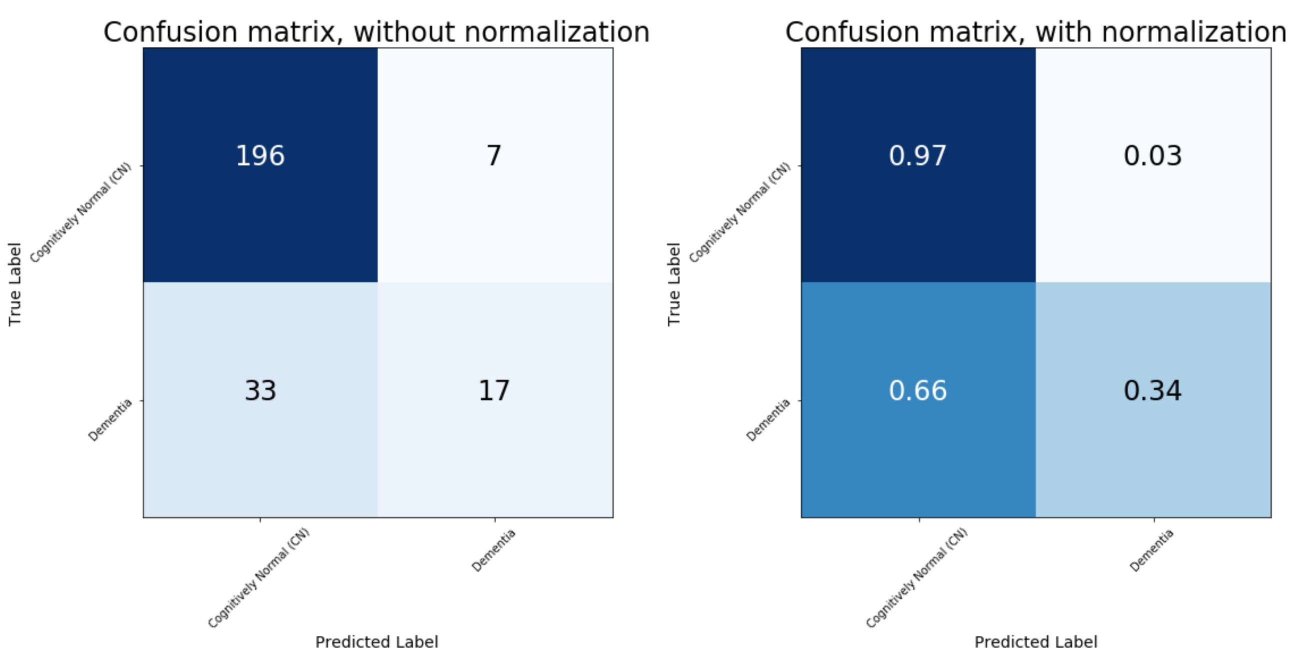
kNN

```
In [83]: knn_model, knn_results_train, knn_results_test = create_and_evaluate_model(
             GridSearchCV(
                 KNeighborsClassifier(),
                 {
                     "n_neighbors": [1, 2, 4, 8, 16, 32, 64]
                 },
                 cv=3
             ).fit(X_train, y_train).best_estimator_,
             X_train,
             y_train,
             X_test,
             y_test
         )
         plot_model_results(knn_results_train, f"kNN (k = {knn_model.get_params()['n_neighbors']}) (Train)")
         plot_model_results(knn_results_test, f"kNN (k = {knn_model.get_params()['n_neighbors']}) (Test)")
```

kNN (k = 4) (Train)
(accuracy: 0.9, balanced accuracy: 0.75, specificity: 0.99, sensitivity: 0.51)

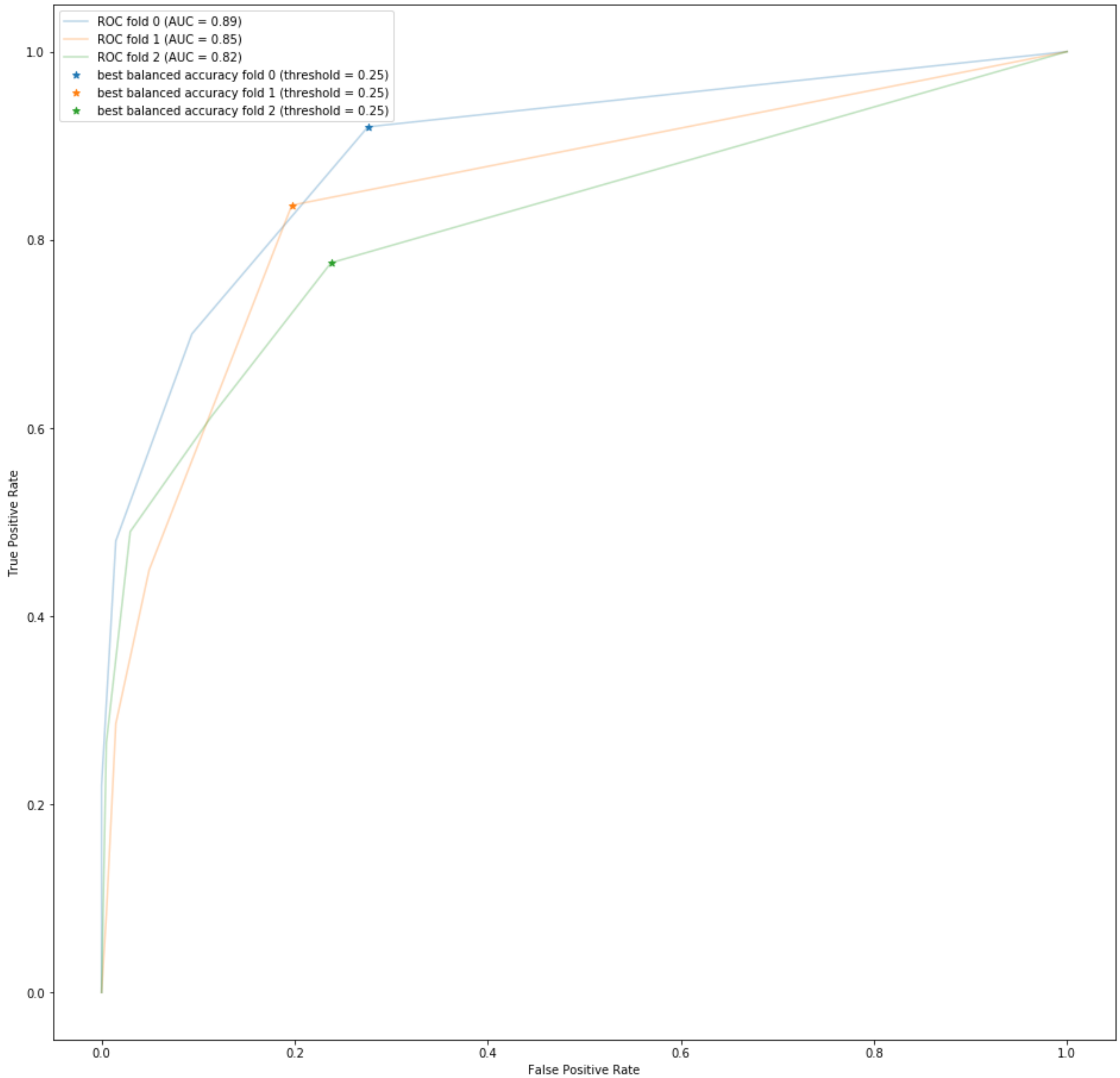


kNN (k = 4) (Test)
(accuracy: 0.84, balanced accuracy: 0.65, specificity: 0.97, sensitivity: 0.34)

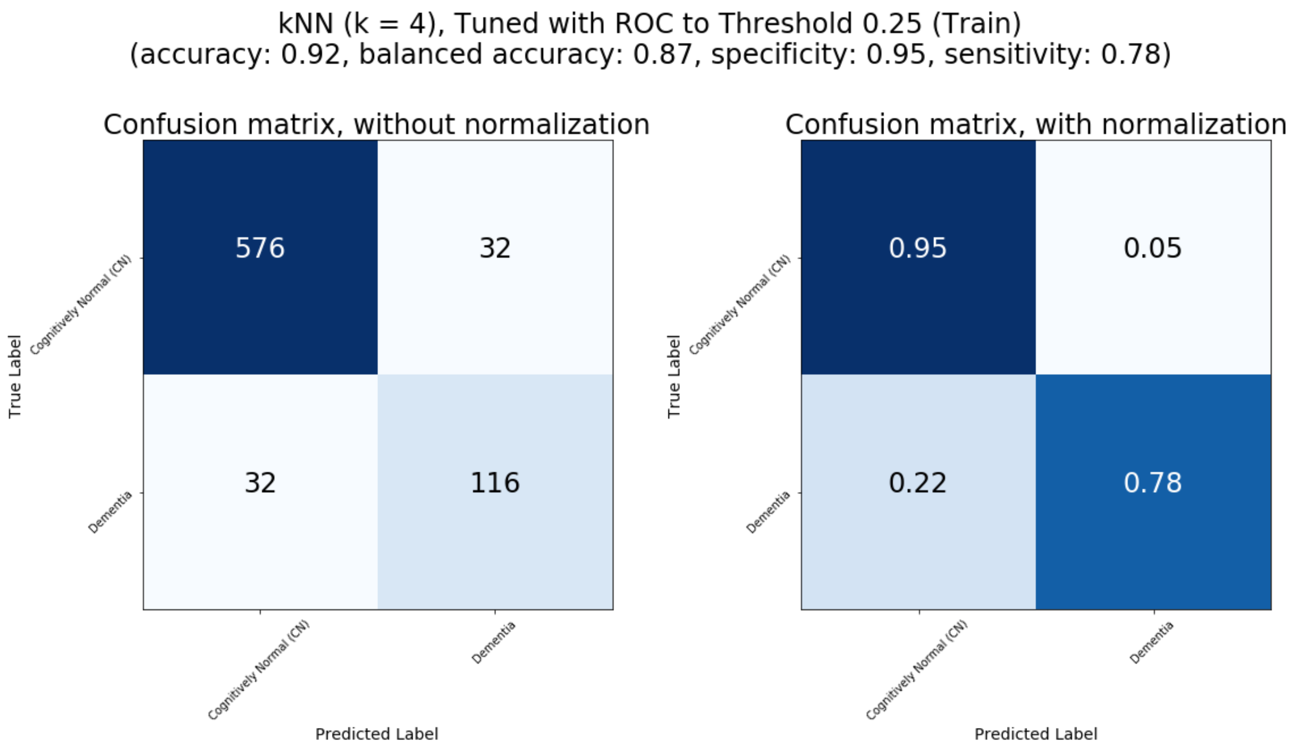


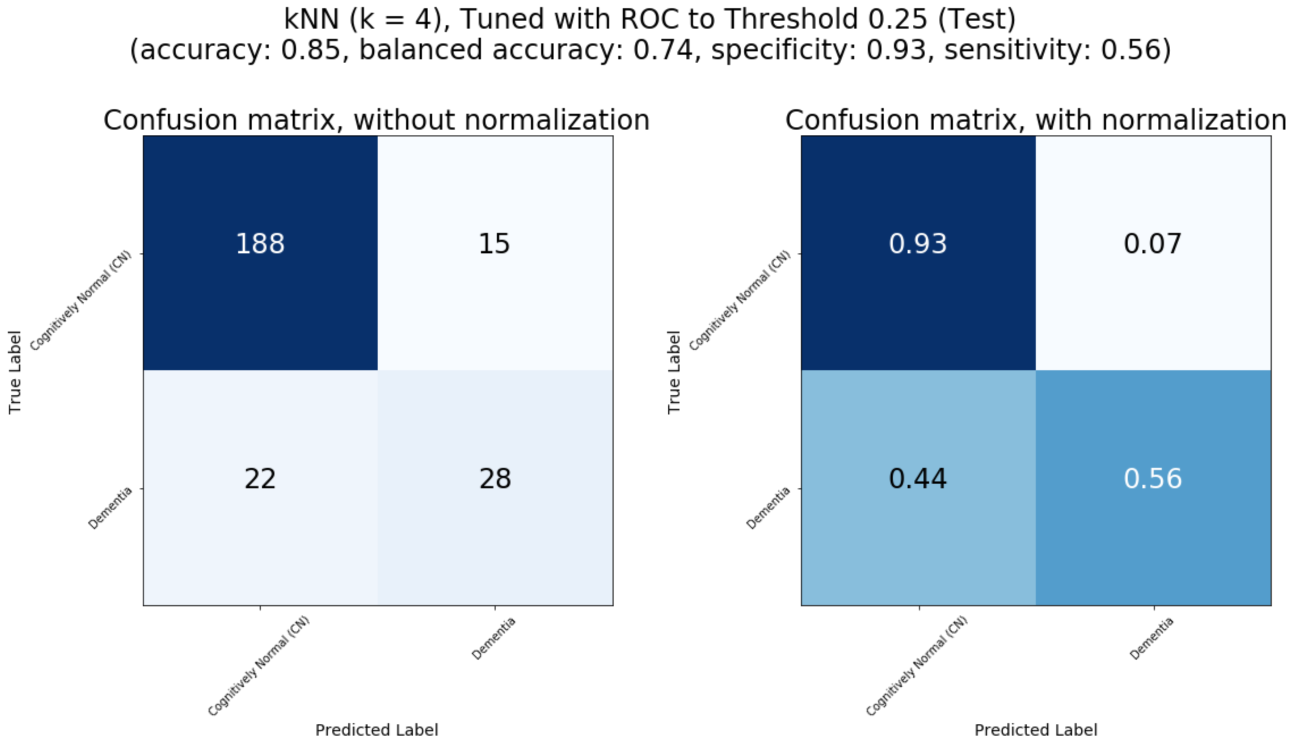
Adjusting kNN with ROC

In [84]:
```
knn_augmented_roc_curves = augmented_roc_curves_cv(
    knn_model,
    X_train,
    y_train
)
knn_roc_tuned_threshold, _ = mean_threshold_best_balanced_accuracy(knn_augmented_roc_curves)
plot_augmented_roc_curves(knn_augmented_roc_curves)
```



In [85]:
```
knn_roc_tuned_model, knn_roc_tuned_results_train, knn_roc_tuned_results_test = \
    create_and_evaluate_probabilistic_model(
        knn_model,
        X_train,
        y_train,
        X_test,
        y_test,
        knn_roc_tuned_threshold
    )
plot_model_results(
    knn_roc_tuned_results_train,
    f"kNN (k = {knn_model.get_params()['n_neighbors']})"
    f", Tuned with ROC to Threshold {knn_roc_tuned_threshold:.2} (Train)"
)
plot_model_results(
    knn_roc_tuned_results_test,
    f"kNN (k = {knn_model.get_params()['n_neighbors']})"
    f", Tuned with ROC to Threshold {knn_roc_tuned_threshold:.2} (Test)"
)
```





kNN with Class Weights

[scikit-learn does not provide this option with `KNeighborsClassifier`. (https://stackoverflow.com/a/49492492)](https://stackoverflow.com/a/49492492)

An alternative may be to [use a `contrib` package like `imbalanced-learn` to deliberately undersample (https://imbalanced-learn.readthedocs.io/en/stable/under_sampling.html)](https://imbalanced-learn.readthedocs.io/en/stable/under_sampling.html) the Cognitively Normal class.

In [ ]:

```
```

Summary

In [86]:
```
train_results = pd.DataFrame(
    list(map(
        model_results_to_dict,
        *zip(*[
            (svc_baseline_results_train, "SVC"),
            (svc_probabilistic_results_train, "SVC with Probability Outputs"),
            (rf_results_train, "Random Forest"),
            (logreg_results_train, "Logistic Regression"),
            (adaboost_results_train, "AdaBoost"),
            (lda_results_train, "LDA"),
            (knn_results_train, "kNN"),

            (svc_roc_tuned_results_train, "SVC Tuned with ROC"),
            (rf_roc_tuned_results_train, "Random Forest Tuned with ROC"),
            (logreg_roc_tuned_results_train, "Logistic Regression Tuned with ROC"),
            (adaboost_roc_tuned_results_train, "AdaBoost Tuned with ROC"),
            (lda_roc_tuned_results_train, "LDA Tuned with ROC"),
            (knn_roc_tuned_results_train, "kNN Tuned with ROC"),

            (svc_weighted_results_train, "SVC with Class Weights"),
            (rf_weighted_results_train, "Random Forest with Class Weights"),
            (logreg_weighted_results_train, "Logistic Regression with Class Weights"),
            (adaboost_weighted_results_train, "AdaBoost with Class Weights")
        ])
    )),
    columns=["model", "accuracy", "specificity", "sensitivity", "balanced accuracy"]
).set_index("model")

with pd.option_context("precision", 3):
    display(HTML("<h3>Model Performance on Training Data</h3>"))
    print(train_results.to_string())
```

Model Performance on Training Data

model	accuracy	specificity	sensitivity	balanced accuracy
SVC	0.931	0.980	0.730	0.855
SVC with Probability Outputs	0.931	0.980	0.730	0.855
Random Forest	0.975	0.998	0.878	0.938
Logistic Regression	0.892	0.979	0.534	0.756
AdaBoost	0.950	0.974	0.851	0.913
LDA	0.902	0.952	0.696	0.824
kNN	0.898	0.993	0.507	0.750
SVC Tuned with ROC	0.907	0.910	0.899	0.904
Random Forest Tuned with ROC	0.934	0.918	1.000	0.959
Logistic Regression Tuned with ROC	0.856	0.840	0.919	0.880
AdaBoost Tuned with ROC	0.782	0.729	1.000	0.864
LDA Tuned with ROC	0.862	0.845	0.932	0.889
kNN Tuned with ROC	0.915	0.947	0.784	0.866
SVC with Class Weights	0.857	0.849	0.892	0.870
Random Forest with Class Weights	0.878	0.872	0.905	0.889
Logistic Regression with Class Weights	0.861	0.857	0.878	0.868
AdaBoost with Class Weights	0.926	0.921	0.946	0.933

In [87]:
```python
test_results = pd.DataFrame(
    list(map(
        model_results_to_dict,
        *zip(*[
            (svc_baseline_results_test, "SVC"),
            (svc_probabilistic_results_test, "SVC with Probability Outputs"),
            (rf_results_test, "Random Forest"),
            (logreg_results_test, "Logistic Regression"),
            (adaboost_results_test, "AdaBoost"),
            (lda_results_test, "LDA"),
            (knn_results_test, "kNN"),

            (svc_roc_tuned_results_test, "SVC Tuned with ROC"),
            (rf_roc_tuned_results_test, "Random Forest Tuned with ROC"),
            (logreg_roc_tuned_results_test, "Logistic Regression Tuned with ROC"),
            (adaboost_roc_tuned_results_test, "AdaBoost Tuned with ROC"),
            (lda_roc_tuned_results_test, "LDA Tuned with ROC"),
            (knn_roc_tuned_results_test, "kNN Tuned with ROC"),

            (svc_weighted_results_test, "SVC with Class Weights"),
            (rf_weighted_results_test, "Random Forest with Class Weights"),
            (logreg_weighted_results_test, "Logistic Regression with Class Weights"),
            (adaboost_weighted_results_test, "AdaBoost with Class Weights")
        ])
    )),
    columns=["model", "accuracy", "specificity", "sensitivity", "balanced accuracy"]
).set_index("model")

with pd.option_context("precision", 3):
    display(HTML("<h3>Model Performance on Test Data</h3>"))
    print(test_results.to_string())
```

Model Performance on Test Data

model	accuracy	specificity	sensitivity	balanced accuracy
SVC	0.881	0.941	0.64	0.790
SVC with Probability Outputs	0.881	0.941	0.64	0.790
Random Forest	0.881	0.966	0.54	0.753
Logistic Regression	0.862	0.951	0.50	0.725
AdaBoost	0.854	0.906	0.64	0.773
LDA	0.854	0.897	0.68	0.788
kNN	0.842	0.966	0.34	0.653
SVC Tuned with ROC	0.826	0.818	0.86	0.839
Random Forest Tuned with ROC	0.834	0.823	0.88	0.851
Logistic Regression Tuned with ROC	0.838	0.833	0.86	0.846
AdaBoost Tuned with ROC	0.751	0.709	0.92	0.815
LDA Tuned with ROC	0.798	0.778	0.88	0.829
kNN Tuned with ROC	0.854	0.926	0.56	0.743
SVC with Class Weights	0.850	0.847	0.86	0.854
Random Forest with Class Weights	0.854	0.857	0.84	0.849
Logistic Regression with Class Weights	0.846	0.847	0.84	0.844
AdaBoost with Class Weights	0.866	0.877	0.82	0.848

In [88]:
```python
# import time
# test_results.to_csv(f"../output/test_results-{time.time()}.csv")
```

In [ ]: