Recognizing Spoken Digits

December 11, 2021

1 Recognizing Spoken Digits: Luis Pereda Amaya

[328]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from sklearn.manifold import TSNE
```

[329]:

```
train_file = open("Train_Arabic_Digit.txt")
test_file = open("Test_Arabic_Digit.txt")
train_lines = train_file.readlines()
test_lines = test_file.readlines()
```

[330]:

```
# Parsing Training Data

count = -1
blocks = []
# Added spaces to the end of my code for this to work
for line in train_lines:
    if(len(line) == 13):
        if(count >= 0):
            blocks.append(block)
        count += 1
        block = []
    else:
        block.append(line.strip())
blocks.append(block[0 : len(block) - 1])

digits = []

for i in range(1, 11):
    digits.append(blocks[((i-1) * 660) : (i * 660)])
```

```

for i in range(10):
    for j in range(len(digits[i])):
        for k in range(len(digits[i][j])):
            digits[i][j][k] = digits[i][j][k].split(" ")
            digits[i][j][k] = [float(num) for num in digits[i][j][k]]


menDigits = []
womenDigits = []
for i in range(10):
    menDigits.append(digits[i][0:330])
    womenDigits.append(digits[i][330:660])

Digits = []
for i in range(10):
    frames = []
    for j in range(len(digits[i])):
        frames = frames + digits[i][j]
    Digits.append(frames)
digits = Digits

MenDigits = []
for i in range(10):
    men_frames = []
    for j in range(len(menDigits[i])):
        men_frames = men_frames + menDigits[i][j]
    MenDigits.append(men_frames)

WomenDigits = []
for i in range(10):
    women_frames = []
    for j in range(len(womenDigits[i])):
        women_frames = women_frames + womenDigits[i][j]
    WomenDigits.append(women_frames)
```

[333]:

```
# Parsing Test Data
count = -1
blocks = []
# Added spaces to the end of my code for this to work
for line in test_lines:
    if(len(line) == 13):
        if(count >= 0):
            blocks.append(block)
        count += 1
        block = []
    else:
        block.append(line.strip())
blocks.append(block[0:len(block) - 1])


test_digits = []
for i in range(1,11):
    test_digits.append(blocks[((i-1) * 220) : (i * 220)])

for i in range(10):
    for j in range(len(test_digits[i])):
        for k in range(len(test_digits[i][j])):
            test_digits[i][j][k] = test_digits[i][j][k].split(" ")
            test_digits[i][j][k] = [float(num) for num in test_digits[i][j][k]]
menTestDigits = []
womenTestDigits = []
for i in range(10):
    menTestDigits.append(test_digits[i][0:110])
    womenTestDigits.append(test_digits[i][110:220])

test_Digits = []
for i in range(10):
    frames = []
    for j in range(len(test_digits[i])):
        frames = frames + test_digits[i][j]
    test_Digits.append(frames)
# Will use test_blocks to iterate through blocks in test
test_blocks = test_digits

test_digits = test_Digits
```

[336]:

```
print(len(womenTestDigits[0]))
```

```
110
```

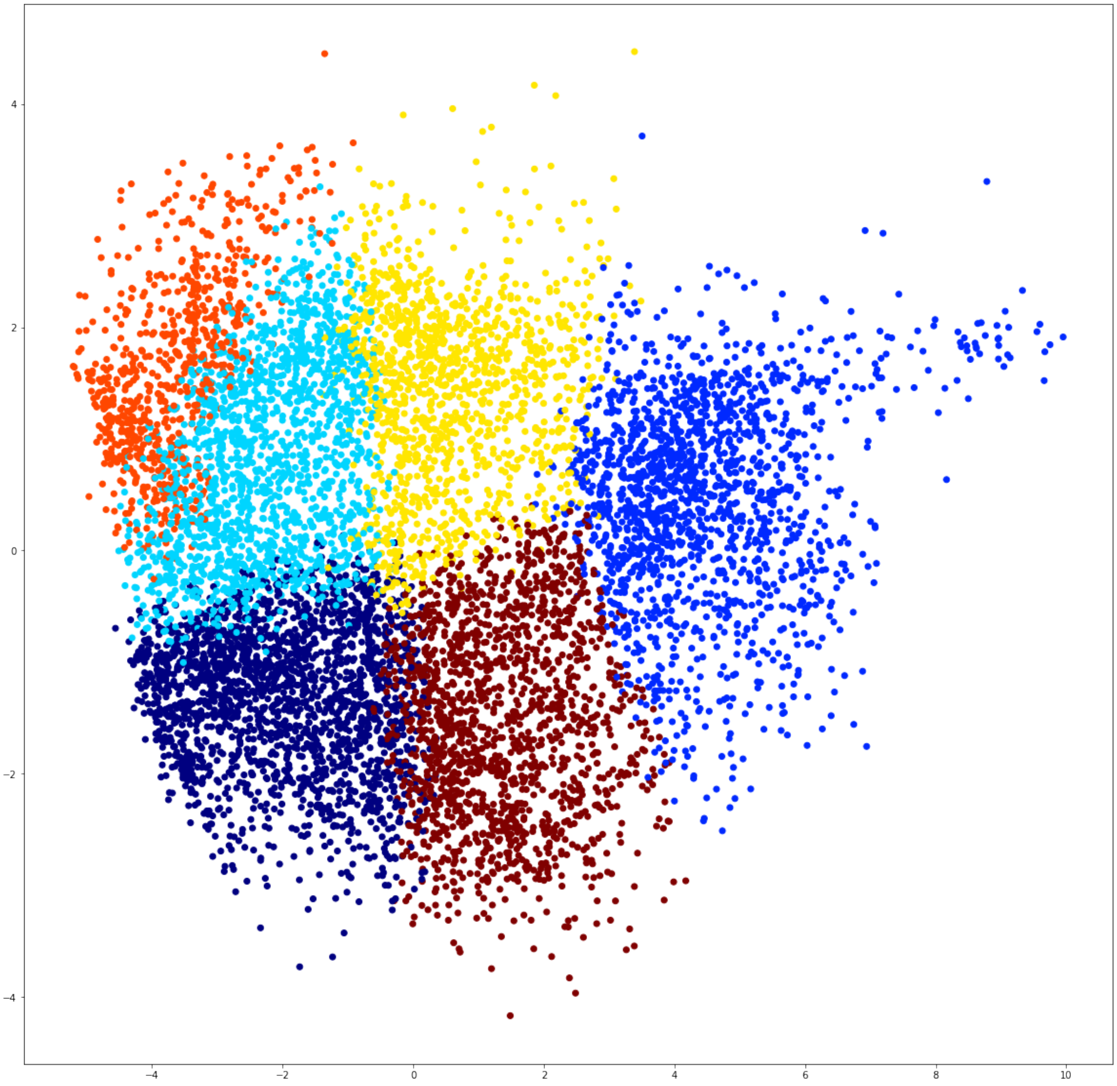
1.1 K-means

[338]:

```
def findKmeans(arr, digit, clusters):
    kmeans = KMeans(n_clusters=clusters).fit(arr[digit])
#     labels = kmeans.predict(arr[digit])
    labels = kmeans.labels_
    return kmeans, labels

def findAndPlotKmeans(arr, digit, clusters):
    plt.figure(figsize=[20,20])

    kmeans = KMeans(n_clusters=clusters).fit(arr[digit])
    cluster_centers = kmeans.cluster_centers_
    labels = kmeans.predict(test_Digits[digit])
    pca = PCA(n_components=2)
    pca_data = pca.fit_transform(test_Digits[digit])

    plt.scatter(pca_data[:, 0], pca_data[:, 1], c = labels, cmap='jet')
    return kmeans
```

[339]:

```
kmeans = findAndPlotKmeans(Digits, 4, 7)
```



1.1.1 GMM from KMeans

[340]:

```
digit_clusters = [6, 5, 7, 6, 7, 6, 5, 5, 9, 4]
digit_components = [6, 5, 7, 6, 7, 6, 5, 5, 9, 4]
kmeans_arr = []
kmeans_labels = []
for i in range(10):
    kmeans_arr.append(findKmeans(Digits, i, digit_clusters[i])[0])
    kmeans_labels.append(findKmeans(Digits, i, digit_clusters[i])[1])
```

[341]:

```
kmeans_arr_men = []
kmeans_arr_women = []
kmeans_labels_men = []
kmeans_labels_women = []

for i in range(10):
    kmeans_arr_men.append(findKmeans(MenDigits, i, digit_clusters[i])[0])
    kmeans_labels_men.append(findKmeans(MenDigits, i, digit_clusters[i])[1])

    kmeans_arr_women.append(findKmeans(WomenDigits, i, digit_clusters[i])[0])
    kmeans_labels_women.append(findKmeans(WomenDigits, i, digit_clusters[i])[1])
```

[356]:

```
kmeans_centers = []
for digit_model in kmeans_arr:
    kmeans_centers.append(digit_model.cluster_centers_)
```

[357]:

```
kmeans_centers_men = []
for digit_model in kmeans_arr_men:
    kmeans_centers_men.append(digit_model.cluster_centers_)
kmeans_centers_women = []
for digit_model in kmeans_arr_women:
    kmeans_centers_women.append(digit_model.cluster_centers_)
```

[359]:

```
df_arr = []
for i in range(10):
    cluster_df = pd.DataFrame()
    cluster_df['cluster'] = kmeans_labels[i]
    cluster_df['data'] = Digits[i]
    df_arr.append(cluster_df)
```

[360]:

```
df_arr_men = []
df_arr_women = []
for i in range(10):
    cluster_df_men = pd.DataFrame()
    cluster_df_men['cluster'] = kmeans_labels_men[i]
    cluster_df_men['data'] = MenDigits[i]
    df_arr_men.append(cluster_df_men)
    cluster_df_women = pd.DataFrame()
    cluster_df_women['cluster'] = kmeans_labels_women[i]
    cluster_df_women['data'] = WomenDigits[i]
    df_arr_women.append(cluster_df_women)
```

[362]:

```
print(len(df_arr[0]))
print(len(df_arr_men[0]))
print(len(df_arr_women[0]))
```

```
23344
11588
11756
       cluster                                               data
0            1  [-0.51891, -3.4561, 1.7697, 0.5174, 0.45923, 0…
1            3  [3.9787, -3.3583, 1.0912, 2.5968, -2.3069, -1…
2            3  [4.7732, -4.2533, 1.6287, 2.8034, -2.7204, -1…
3            3  [4.8029, -4.1085, 1.5626, 2.888, -2.5246, -1.3…
4            3  [4.4216, -3.5715, 1.5231, 2.2161, -2.3795, -1…
…          …                                                 …
23949        2  [5.8878, -5.6965, -4.4088, -2.9723, -0.09737, …
23950        2  [5.6703, -5.2815, -4.0359, -2.5452, -0.28251, …
23951        2  [5.2396, -5.0395, -3.4181, -2.9282, 0.11057, -…
23952        2  [4.6918, -5.0641, -2.7542, -3.3725, -0.20284, …
23953        2  [3.3359, -4.3017, -2.7763, -3.0208, -0.74998, …

[23954 rows x 2 columns]
23344
```

[363]:

```
def findFullCov(data):
    arr = []
    for i in range(10):
        digit_covariances = []
        for j in range(digit_clusters[i]):
            np_arr = np.array(data[i][data[i].cluster == j])
            samples = np.array([x[1] for x in np_arr])
            cov = np.cov(samples.T)
            digit_covariances.append(cov)
        arr.append(digit_covariances)
    return arr
```

[365]:

```
full_covariances = findFullCov(df_arr)
full_covariances_men = findFullCov(df_arr_men)
full_covariances_women = findFullCov(df_arr_women)
```

[203]:

```
print(len(full_covariances[0]))
print(len(full_covariances[0][0]))
print(len(full_covariances[0][0][0]))
```

```
6
13
13
```

[97]:

```
def findDiagCov(fullCov):
    arr = []
    for i in range(10):
        cluster_diag_covs = []
        for cluster_cov in fullCov[i]:
            diag = np.diag(np.diag(cluster_cov))
            cluster_diag_covs.append(diag)
        arr.append(cluster_diag_covs)
    return arr
```

[98]:

```
diag_covariances = findDiagCov(full_covariances)
diag_covariances_men = findDiagCov(full_covariances_men)
diag_covariances_women = findDiagCov(full_covariances_women)
```

[99]:

```
print(len((diag_covariances)))
print(len(diag_covariances[0]))
print(len((diag_covariances[0][0])))
```

```
10
6
13
```

[100]:

```
def findSphericalCov(data):
    arr = []
    for i in range(10):
        demeaned_arr = []
        demeaned_data = np.array([])
        for j in range(digit_clusters[i]):
            np_arr = np.array(data[i][data[i].cluster == j])
            samples = np.array([x[1] for x in np_arr])
            mean = samples.mean(axis = 0)
            demeaned = samples - mean
            demeaned_arr.append(demeaned)
        demeaned_data = np.concatenate(demeaned_arr)
        digit_var = np.var(demeaned_data)
        var_mat = np.identity(13) * digit_var
        arr.append(var_mat)
    return arr
```

[101]:

```
sphere_covariances = findSphericalCov(df_arr)
sphere_covariances_men = findSphericalCov(df_arr_men)
sphere_covariances_women = findSphericalCov(df_arr_women)
```

[102]:

```
print(len(sphere_covariances[0][0]))
```

```
13
```

[103]:

```
def findTiedCov(data):
    arr = []
    for i in range(10):
        demeaned_arr = []
        demeaned_data = np.array([])
        for j in range(digit_clusters[i]):
            np_arr = np.array(data[i][data[i].cluster == j])
            samples = np.array([x[1] for x in np_arr])
            mean = samples.mean(axis = 0)
            demeaned = samples - mean
            demeaned_arr.append(demeaned)
        demeaned_data = np.concatenate(demeaned_arr)
        demeaned_cov = np.cov(demeaned_data.T)
        arr.append(demeaned_cov)
    return arr
```

[104]:

```
tied_covariances = findTiedCov(df_arr)
tied_covariances_men = findTiedCov(df_arr_men)
tied_covariances_women = findTiedCov(df_arr_women)
```

[105]:

```
print(len(tied_covariances[0]))
print(len(tied_covariances[0][0]))
```

```
13
13
```

[106]:

```
def findWeights(data):
    arr = []
    for i in range(10):
        weights = []
        for j in range(digit_clusters[i]):
            np_arr = np.array(data[i][data[i].cluster == j])
            samples = np.array([x[1] for x in np_arr])
            weight = len(samples) / len(data[i])
            weights.append(weight)
        arr.append(weights)
    return arr
```

[107]:

```
digit_weights = findWeights(df_arr)
digit_weights_men = findWeights(df_arr_men)
digit_weights_women = findWeights(df_arr_women)
```

[130]:

```
def setFullCovModel(centers, covs, weights):
    arr = []
    for i in range(10):
        gmm = GaussianMixture(n_components=digit_components[i],␣
 ↪covariance_type='full')
        gmm.means_ = centers[i]
        gmm.covariances_ = covs[i]
        gmm.weights_ = weights[i]
        gmm.precisions_ = np.linalg.inv(covs[i])
        gmm.precisions_cholesky_ = np.linalg.cholesky(gmm.precisions_)
        arr.append(gmm)
    return arr
```

[131]:

```
KMeans_GMM_full_cov = setFullCovModel(kmeans_centers, full_covariances,␣
 ↪digit_weights)
KMeans_GMM_full_cov_men = setFullCovModel(kmeans_centers_men,␣
 ↪full_covariances_men, digit_weights_men)
KMeans_GMM_full_cov_women = setFullCovModel(kmeans_centers_women,␣
 ↪full_covariances_women, digit_weights_women)
```

[110]:

```
def setTiedCovModel(centers, covs, weights):
    arr = []
    for i in range(10):
        gmm = GaussianMixture(n_components=digit_components[i],␣
 ↪covariance_type='tied')
        gmm.means_ = centers[i]
        gmm.covariances_ = covs[i]
        gmm.weights_ = weights[i]
        gmm.precisions_ = np.linalg.inv(covs[i])
        gmm.precisions_cholesky_ = np.linalg.cholesky(gmm.precisions_)
        arr.append(gmm)
    return arr
```

[111]:

```
KMeans_GMM_tied_cov = setTiedCovModel(kmeans_centers, tied_covariances,␣
 ↪digit_weights)
KMeans_GMM_tied_cov_men = setTiedCovModel(kmeans_centers_men,␣
 ↪tied_covariances_men, digit_weights_men)
KMeans_GMM_tied_cov_women = setTiedCovModel(kmeans_centers_women,␣
 ↪tied_covariances_women, digit_weights_women)
```

[112]:

```
def setDiagCovModel(centers, covs, weights):
    arr = []
    for i in range(10):
        gmm = GaussianMixture(n_components=digit_components[i],␣
 ↪covariance_type='diag')
        gmm.means_ = centers[i]
        gmm.covariances_ = [np.diag(cluster) for cluster in covs[i]]
        gmm.weights_ = weights[i]
        diag_precisions_matrix = np.linalg.inv(covs[i])
        gmm.precisions_ = [np.diag(item) for item in diag_precisions_matrix]
        gmm.precisions_cholesky_ = np.array([np.diag(item) for item in np.
 ↪linalg.cholesky(diag_precisions_matrix)])
        arr.append(gmm)
    return arr
```

[113]:
```
KMeans_GMM_diag_cov = setDiagCovModel(kmeans_centers, diag_covariances,␣
 ↪digit_weights)
KMeans_GMM_diag_cov_men = setDiagCovModel(kmeans_centers_men,␣
 ↪diag_covariances_men, digit_weights_men)
KMeans_GMM_diag_cov_women = setDiagCovModel(kmeans_centers_women,␣
 ↪diag_covariances_women, digit_weights_women)
```

[114]:
```
def setSphericalCovModel(centers, covs, weights):
    arr = []
    for i in range(10):
        gmm = GaussianMixture(n_components=digit_components[i],␣
 ↪covariance_type='spherical')
        gmm.means_ = centers[i]
        gmm.covariances_ = covs[i][0][0]
        gmm.weights_ = weights[i]
        sphere_precisions_matrix = np.linalg.inv(covs[i])
        gmm.precisions_ = sphere_precisions_matrix[0][0]
        gmm.precisions_cholesky_ = np.linalg.
 ↪cholesky(sphere_precisions_matrix)[0][0]
        arr.append(gmm)
    return arr
```

[115]:
```
KMeans_GMM_spherical_cov = setSphericalCovModel(kmeans_centers,␣
 ↪sphere_covariances, digit_weights)
KMeans_GMM_spherical_cov_men = setSphericalCovModel(kmeans_centers_men,␣
 ↪sphere_covariances_men, digit_weights_men)
KMeans_GMM_spherical_cov_women = setSphericalCovModel(kmeans_centers_women,␣
 ↪sphere_covariances_women, digit_weights_women)
```

1.1.2 ML Classification

[192]:
```
def findAccuracyNonGendered(covs):
    correct = 0
    total = 0
    for i in range(10):
        for block in test_blocks[i]:
            maxScore = covs[0].score(block)
            maxIndex = 0
            total += 1
            for digit in range(10):
                score = covs[digit].score(block)
                if(score > maxScore):
                    maxScore = score
                    maxIndex = digit
            if(maxIndex == i):
                correct += 1
    print(correct / total)
```

[198]:

```
def findAccuracyGendered(covs_men, covs_women):
    correct = 0
    total = 0
    for i in range(10):
        for block in menTestDigits[i]:
            maxScore = covs_men[0].score(block)
            maxIndex = 0
            total += 1
            for digit in range(10):
                score = covs_men[digit].score(block)
                if(score > maxScore):
                    maxScore = score
                    maxIndex = digit
            if(maxIndex == i):
                correct += 1
    for i in range(10):
        for block in womenTestDigits[i]:
            maxScore = covs_women[0].score(block)
            maxIndex = 0
            total += 1
            for digit in range(10):
                score = covs_women[digit].score(block)
                if(score > maxScore):
                    maxScore = score
                    maxIndex = digit
            if(maxIndex == i):
                correct += 1
    print(correct/total)
```

Non-Gendered

[194]:

```
# Spherical
findAccuracyNonGendered(KMeans_GMM_spherical_cov)
```

```
0.7645454545454545
```

[195]:

```
# Diagonal
findAccuracyNonGendered(KMeans_GMM_diag_cov)
```

```
0.7268181818181818
```

[196]:

```
# Tied
findAccuracyNonGendered(KMeans_GMM_tied_cov)
```

```
0.8313636363636364
```

[197]:

```
# Full
findAccuracyNonGendered(KMeans_GMM_full_cov)
```

```
0.5609090909090909
```

Gendered

[199]:

```
# Spherical
findAccuracyGendered(KMeans_GMM_spherical_cov_men,␣
↪KMeans_GMM_spherical_cov_women)
```

```
0.8013636363636364
```

[200]:

```
# Diagonal
findAccuracyGendered(KMeans_GMM_diag_cov_men, KMeans_GMM_diag_cov_women)
```

```
0.8077272727272727
```

[201]:

```
# Tied
findAccuracyGendered(KMeans_GMM_tied_cov_men, KMeans_GMM_tied_cov_women)
```

```
0.8722727272727273
```

[202]:

```
# Full
findAccuracyGendered(KMeans_GMM_full_cov_men, KMeans_GMM_full_cov_women)
```

```
0.6513636363636364
```

1.2 Expectation-Maximization

[55]:

```
# Seaborn stuff
    # pca_df = pd.DataFrame(data = pca_data, columns = ['1', '2'])
    # pca_df['color'] = pd.DataFrame(labels)
    # ax = sns.scatterplot(x = '1', y = '2', data = pca_df, hue = 'color',␣
↪legend = 'full', palette = 'husl')
```

[56]:

```
def findAndPlotEM(digit, components, covar_type):
#     plt.figure(figsize=[20,20])

    em_model = GaussianMixture(n_components=components, covariance_type=␣
↪covar_type).fit(Digits[digit])
    return em_model

#     pca = PCA(n_components=2)
#     pca_data = pca.fit_transform(test_Digits[digit])
#     plt.scatter(pca_data[:, 0], pca_data[:, 1], c = labels, cmap='jet')
```

[57]:

```
def findAndPlotEMGendered(digit, components, covar_type, genderBoolean):
    # Let true be male and false female (F for female)
    if (genderBoolean):
        em_model = GaussianMixture(n_components=components, covariance_type=␣
 ↪covar_type).fit(MenDigits[digit])
    else:
        em_model = GaussianMixture(n_components=components, covariance_type=␣
 ↪covar_type).fit(WomenDigits[digit])

    return em_model
```

[58]:

```
em_models_full = []
em_male_models_full = []
em_female_models_full = []

em_models_tied = []
em_male_models_tied = []
em_female_models_tied = []

em_models_diag = []
em_male_models_diag = []
em_female_models_diag = []

em_models_sphere = []
em_male_models_sphere = []
em_female_models_sphere = []

# Components defined empirically by looking for well defined clusters near 2n -␣
 ↪1 where n is phonemes
digit_components = [6, 7, 7, 6, 7, 6, 5, 5, 9, 4]
for i in range(10):
    em_models_full.append(findAndPlotEM(i, digit_components[i], 'full'))
    em_male_models_full.append(findAndPlotEMGendered(i, digit_components[i],␣
 ↪'full', True))
    em_female_models_full.append(findAndPlotEMGendered(i, digit_components[i],␣
 ↪'full', False))

    em_models_tied.append(findAndPlotEM(i, digit_components[i], 'tied'))
    em_male_models_tied.append(findAndPlotEMGendered(i, digit_components[i],␣
 ↪'tied', True))
    em_female_models_tied.append(findAndPlotEMGendered(i, digit_components[i],␣
 ↪'tied', False))

    em_models_diag.append(findAndPlotEM(i, digit_components[i], 'diag'))
    em_male_models_diag.append(findAndPlotEMGendered(i, digit_components[i],␣
 ↪'diag', True))
```

```
    em_female_models_diag.append(findAndPlotEMGendered(i, digit_components[i],␣
↪'diag', False))

    em_models_sphere.append(findAndPlotEM(i, digit_components[i], 'spherical'))
    em_male_models_sphere.append(findAndPlotEMGendered(i, digit_components[i],␣
↪'spherical', True))
    em_female_models_sphere.append(findAndPlotEMGendered(i,␣
↪digit_components[i], 'spherical', False))
```

[59]:

```
print(em_models_full[1].weights_)
```

```
[0.14083955 0.14187504 0.13186144 0.14149315 0.13617793 0.17881029
 0.12894259]
```

1.2.1 ML Classification

Can use same functions defined for KMEans

Non-gendered

[205]:

```
# Non-gendered Full
findAccuracyNonGendered(em_models_full)
```

```
0.8859090909090909
```

[206]:

```
# Non-gendered Tied
findAccuracyNonGendered(em_models_tied)
```

```
0.8954545454545455
```

[207]:

```
# Non-gendered Diag
findAccuracyNonGendered(em_models_diag)
```

```
0.8727272727272727
```

[208]:

```
# Non-gendered Spherical
findAccuracyNonGendered(em_models_sphere)
```

```
0.7659090909090909
```

Gendered

[209]:

```
# Gendered full
findAccuracyGendered(em_male_models_full, em_female_models_full)
```

```
0.865909090909091
```

[210]:

```
# Gendered tied
findAccuracyGendered(em_male_models_tied, em_female_models_tied)
```

```
0.8922727272727272
```

[211]:

```
# Gendered diag
findAccuracyGendered(em_male_models_diag, em_female_models_diag)
```

```
0.889090909090909
```

[212]:

```
# Gendered spherical

findAccuracyGendered(em_male_models_sphere, em_female_models_sphere)
```

```
0.8481818181818181
```

[214]:

```
findAccuracyGendered(em_male_models_diag, em_female_models_full)
```

```
0.8722727272727273
```