100%Accuracy

October 27, 2019

0.1 Importing all the libraries

[3]:
```
import numpy as np
from numpy import genfromtxt
import matplotlib.pyplot as plt
import scipy
from scipy import ndimage
import PIL
from persim import plot_diagrams
from ripser import ripser, lower_star_img
import csv
```

0.2 Looping through all the letters in every direction

[4]:
```
# Left-to-right scanning through loops
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmLR = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)/10)
            column=(k-1)%10
            letter[row,column]=k%10
    dgmLR[i] = lower_star_img(letter)
```

[5]:
```
# Print A-Z diagrams
print(dgmLR[0:25])
```

```
[array([[ 2., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[ 3.,
inf]]), array([[ 3., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[
```

```
3., inf]]), array([[ 4.,  5.],
       [ 4., inf]]), array([[ 4.,  6.],
       [ 4., inf]]), array([[ 3., inf]]), array([[ 3., inf]]), array([[ 2.,
inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 3., inf]]), array([[
6.,  7.],
       [ 2., inf]]), array([[ 3., inf]]), array([[ 3.,  8.],
       [ 3., inf]]), array([[ 2., inf]]), array([[ 3., inf]]), array([[ 2.,
inf]]), array([[ 2., inf]]), array([[ 3.,  4.],
       [ 3., inf]]), array([[ 3., inf]])]
```

0.3 Batch process scanning-right-to-left of all letters

[6]:
```
# Right-to-left scanning through loops
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmRL = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)/10)
            column=(k-1)%10
            letter[row,column]=10-k%10
    dgmRL[i] = lower_star_img(letter)
```

[7]:
```
# Print A-Z diagrams
print(dgmRL[0:25])
```

```
[array([[ 2., inf]]), array([[ 3.,  4.],
       [ 2., inf]]), array([[ 2.,  7.],
       [ 2., inf]]), array([[ 1., inf]]), array([[ 2.,  7.],
       [ 2.,  7.],
       [ 2., inf]]), array([[ 3.,  7.],
       [ 3., inf]]), array([[ 3.,  7.],
       [ 3., inf]]), array([[ 3., inf]]), array([[ 4.,  5.],
       [ 4., inf]]), array([[ 4., inf]]), array([[ 3.,  6.],
       [ 3., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[ 3.,
inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[
3.,  5.],
       [ 3., inf]]), array([[ 2.,  7.],
       [ 2., inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 2.,
inf]]), array([[ 1., inf]]), array([[ 3.,  5.],
       [ 3., inf]]), array([[ 3., inf]])]
```

0.4 Batch process scanning-from-up-to-down of all letters

[8]:
```
# Up-to-down scanning through loops
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmUD = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)%10)
            column=(k-1)/10
            letter[row,column]=k%10
    dgmUD[i] = lower_star_img(letter)
```

[9]:
```
# Print A-Z diagrams
print(dgmUD[0:25])
```

```
[array([[ 2., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[ 3.,
inf]]), array([[ 3., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[
3., inf]]), array([[ 4.,  5.],
       [ 4., inf]]), array([[ 4.,  6.],
       [ 4., inf]]), array([[ 3., inf]]), array([[ 3., inf]]), array([[ 2.,
inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 3., inf]]), array([[
6.,  7.],
       [ 2., inf]]), array([[ 3., inf]]), array([[ 3.,  8.],
       [ 3., inf]]), array([[ 2., inf]]), array([[ 3., inf]]), array([[ 2.,
inf]]), array([[ 2., inf]]), array([[ 3.,  4.],
       [ 3., inf]]), array([[ 3., inf]])]
```

0.5 Batch process scanning-from-down-to-up of all letters

[10]:
```
# Up-to-down scanning through loops
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmDU = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
```

```
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)%10)
            column=(k-1)/10
            letter[row,column]=10-k%10
    dgmDU[i] = lower_star_img(letter)
```

[11]:
```
# Print A-Z diagrams
print(dgmDU[0:25])
```

```
[array([[ 2., inf]]), array([[ 3.,  4.],
       [ 2., inf]]), array([[ 2.,  7.],
       [ 2., inf]]), array([[ 1., inf]]), array([[ 2.,  7.],
       [ 2.,  7.],
       [ 2., inf]]), array([[ 3.,  7.],
       [ 3., inf]]), array([[ 3.,  7.],
       [ 3., inf]]), array([[ 3., inf]]), array([[ 4.,  5.],
       [ 4., inf]]), array([[ 4., inf]]), array([[ 3.,  6.],
       [ 3., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[ 3.,
inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[
3.,  5.],
       [ 3., inf]]), array([[ 2.,  7.],
       [ 2., inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 2.,
inf]]), array([[ 1., inf]]), array([[ 3.,  5.],
       [ 3., inf]]), array([[ 3., inf]])]
```

0.6 Batch process angle scanning-from-upper-left of all letters

[12]:
```
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmAngle = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)/10)
            column=(k-1)%10
            letter[row,column]=max(k%10,int(k-1)%10)
    dgmAngle[i] = lower_star_img(letter)
```

[13]:
```
# Print A-Z diagrams
print(dgmAngle[0:25])
```

```
[array([[ 2., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[ 3.,
inf]]), array([[ 3., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[
3., inf]]), array([[ 4.,  5.],
       [ 4., inf]]), array([[ 4.,  6.],
       [ 4., inf]]), array([[ 3., inf]]), array([[ 3., inf]]), array([[ 2.,
inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 3., inf]]), array([[
6.,  7.],
       [ 2., inf]]), array([[ 3., inf]]), array([[ 3.,  8.],
       [ 3., inf]]), array([[ 2., inf]]), array([[ 3., inf]]), array([[ 2.,
inf]]), array([[ 2., inf]]), array([[ 3.,  4.],
       [ 3., inf]]), array([[ 3., inf]])]
```

0.7 Batch process probing scanning-from-lower-left of all letters

[14]:
```python
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmDiagonal = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)/10)
            column=(k-1)%10
            letter[row,column]=(column+row)*k%10
    dgmDiagonal[i] = lower_star_img(letter)
```

[15]:
```python
# Print A-Z diagrams
print(dgmDiagonal[0:25])
```

```
[array([[ 0.,  3.],
       [ 0.,  4.],
       [ 1.,  5.],
       [ 0.,  5.],
       [ 0., inf]]), array([[ 1.,  2.],
       [ 0.,  4.],
       [ 0.,  4.],
       [ 2.,  4.],
       [ 2.,  5.],
       [ 0.,  6.],
       [ 0.,  6.],
       [ 1.,  7.],
       [ 0., inf]]), array([[ 0.,  4.],
       [ 1.,  5.],
```

```
[ 0.,  5.],
[ 4.,  9.],
[ 0., inf]]), array([[ 1.,  2.],
[ 0.,  4.],
[ 5.,  6.],
[ 2.,  6.],
[ 1.,  7.],
[ 0.,  8.],
[ 0., inf]]), array([[ 0.,  4.],
[ 0.,  4.],
[ 5.,  6.],
[ 1.,  7.],
[ 0.,  8.],
[ 0.,  8.],
[ 2.,  8.],
[ 4.,  9.],
[ 0., inf]]), array([[ 0.,  4.],
[ 5.,  6.],
[ 1.,  7.],
[ 0.,  8.],
[ 2.,  8.],
[ 0., inf]]), array([[ 1.,  4.],
[ 0.,  4.],
[ 0.,  5.],
[ 2.,  5.],
[ 5.,  6.],
[ 0., inf]]), array([[ 1.,  7.],
[ 2.,  8.],
[ 1.,  8.],
[ 0., inf]]), array([[ 0.,  5.],
[ 0.,  5.],
[ 0.,  5.],
[ 0., inf]]), array([[ 2.,  5.],
[ 0.,  6.],
[ 2.,  8.],
[ 0., inf]]), array([[ 1.,  2.],
[ 1.,  2.],
[ 2.,  5.],
[ 0.,  7.],
[ 2.,  8.],
[ 0., inf]]), array([[ 0.,  4.],
[ 1.,  7.],
[ 2.,  8.],
[ 0., inf]]), array([[ 0.,  2.],
[ 2.,  4.],
[ 0.,  5.],
[ 2.,  6.],
[ 2.,  8.],
```

```
[ 0.,  8.],
[ 0., inf]]), array([[ 1.,  4.],
[ 2.,  4.],
[ 0.,  5.],
[ 0., inf]]), array([[ 0.,  4.],
[ 0.,  6.],
[ 5.,  6.],
[ 2.,  6.],
[ 1.,  8.],
[ 0.,  8.],
[ 0., inf]]), array([[ 1.,  5.],
[ 5.,  6.],
[ 2.,  6.],
[ 0.,  7.],
[ 0., inf]]), array([[ 0.,  4.],
[ 0.,  6.],
[ 5.,  6.],
[ 0.,  8.],
[ 0.,  8.],
[ 2.,  8.],
[ 0., inf]]), array([[ 1.,  2.],
[ 1.,  2.],
[ 0.,  2.],
[ 2.,  4.],
[ 2.,  6.],
[ 0.,  7.],
[ 0., inf]]), array([[ 0.,  4.],
[ 2.,  5.],
[ 5.,  6.],
[ 0.,  7.],
[ 0.,  8.],
[ 2.,  8.],
[ 4.,  9.],
[ 0., inf]]), array([[ 0.,  5.],
[ 0.,  5.],
[ 0.,  5.],
[ 4.,  9.],
[ 4.,  9.],
[ 0., inf]]), array([[ 0.,  4.],
[ 1.,  7.],
[ 0.,  8.],
[ 0.,  8.],
[ 2.,  8.],
[ 0., inf]]), array([[ 0.,  2.],
[ 2.,  3.],
[ 2.,  8.],
[ 0., inf]]), array([[ 1.,  2.],
[ 0.,  2.],
```

```
       [ 0.,  3.],
       [ 0.,  5.],
       [ 0.,  6.],
       [ 5.,  6.],
       [ 4.,  8.],
       [ 0.,  9.],
       [ 0., inf]]), array([[ 1.,  2.],
       [ 0.,  2.],
       [ 2.,  4.],
       [ 0.,  4.],
       [ 0., inf]]), array([[ 0.,  5.],
       [ 0.,  8.],
       [ 2.,  8.],
       [ 0., inf]])]
```

0.8 Batch process diagonal scanning-from-upper-left of all letters

[16]:
```
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmDiagonalULC = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)/10)
            column=(k-1)%10
            letter[row,column]=k%10 + int((k-1)/10)
    dgmDiagonalULC[i] = lower_star_img(letter)
```

[17]:
```
# Print A-Z diagrams
print(dgmDiagonalULC[0:25])
```

```
[array([[ 5., inf]]), array([[ 3., inf]]), array([[ 5., inf]]), array([[ 4.,
inf]]), array([[ 4., inf]]), array([[ 4., inf]]), array([[ 4., inf]]), array([[
8.,  9.],
       [ 4., inf]]), array([[ 5., inf]]), array([[11., 12.],
       [ 5., inf]]), array([[ 4., inf]]), array([[ 4., inf]]), array([[ 8.,
9.],
       [ 3., inf]]), array([[ 8., 11.],
       [ 3., inf]]), array([[ 4., inf]]), array([[ 4., inf]]), array([[12.,
14.],
       [ 4., inf]]), array([[ 4., inf]]), array([[11., 15.],
       [ 4., inf]]), array([[ 3., inf]]), array([[ 9., 15.],
```

```
       [ 4., inf]]), array([[ 9., 11.],
       [ 3., inf]]), array([[ 7.,  9.],
       [10., 13.],
       [ 3., inf]]), array([[ 8.,  9.],
       [ 4., inf]]), array([[ 8., 10.],
       [ 4., inf]])]
```

0.9 Batch process probing scanning-from-lower-left of all letters

[18]:
```
letters = genfromtxt('letters.csv', delimiter=',') # Upload the file

dgmAngleLF = [None]*26 #Initialize an empty list
for i in range(26):
    letter_one_line=letters[i,:]

    # initialize matrix of size 10x10 with all values 100
    letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
    for k in range(1,101):
        if letter_one_line[k]==1.0:
            row=int((k-1)/10)
            column=(k-1)%10
            letter[row,column]=max(9-(k-1)%10,9-int((k-1)/10))
    dgmAngleLF[i] = lower_star_img(letter)
```

[19]:
```
# Print A-Z diagrams
print(dgmAngleLF[0:25])
```

```
[array([[ 2., inf]]), array([[ 2., inf]]), array([[ 2., inf]]), array([[ 2.,
inf]]), array([[ 5.,  7.],
       [ 2., inf]]), array([[ 5., inf]]), array([[ 3., inf]]), array([[ 3.,
inf]]), array([[ 4., inf]]), array([[ 4., inf]]), array([[ 3., inf]]), array([[
3., inf]]), array([[ 5.,  7.],
       [ 2., inf]]), array([[ 3., inf]]), array([[ 7.,  8.],
       [ 2., inf]]), array([[ 4., inf]]), array([[ 7.,  8.],
       [ 2., inf]]), array([[ 3., inf]]), array([[ 2., inf]]), array([[ 5.,
inf]]), array([[ 2., inf]]), array([[ 4., inf]]), array([[ 5.,  6.],
       [ 2., inf]]), array([[ 3., inf]]), array([[ 4., inf]])]
```

1 Bottle Neck Distance Clustering

[20]:
```
# Import the persim package and run the test in every direction left-to-right
import persim as pm

# Set an empty pairwise distance matrix for future bottleneck distance input
BNDLR = np.zeros((26,26))
```

```
# Change infinities to very large numbers
for i in range(26):
    dgmLR[i][np.isinf(dgmLR[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
    for j in range(26):
        BNDLR[i,j] = pm.bottleneck(dgmLR[i], dgmLR[j])

# The very large values should be set to 0 by bottleneck definity (since the␣
 ↪very large distances would be inifinity)
BNDLR[BNDLR>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
 ↪simplicity we use sklearn's Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering

# For now we set 3 clusters, but we can change this and look for better results␣
 ↪in our model. A lower number here will prevent overfitting.
clusteringLR = AgglomerativeClustering(n_clusters = 5,
                                       affinity = "precomputed",
                                       linkage = "average").fit(BNDLR)

# This will output a vector of length 26 representing a number for each letter␣
 ↪for this scan.
LR_test = clusteringLR.labels_
LR_test
```

[20]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[21]:
```
# Import the persim package and run the test in every direction right-to-left
import persim as pm

# Set an empty pairwise distance matrix for future bottleneck distance input
BNDRL = np.zeros((26,26))


# Change infinities to very large numbers
for i in range(26):
    dgmRL[i][np.isinf(dgmRL[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
    for j in range(26):
        BNDRL[i,j] = pm.bottleneck(dgmRL[i], dgmRL[j])
```

```
# The very large values should be set to 0 by bottleneck definity (since the␣
↪very large distances would be inifinity)
BNDRL[BNDRL>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
↪simplicity we use sklearn's Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering

# For now we set 3 clusters, but we can change this and look for better results␣
↪in our model. A lower number here will prevent overfitting.
clusteringRL = AgglomerativeClustering(n_clusters = 5,
                                       affinity = "precomputed",
                                       linkage = "average").fit(BNDRL)

# This will output a vector of length 26 representing a number for each letter␣
↪for this scan.
RL_test = clusteringRL.labels_
RL_test
```

[21]:
```
array([0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4, 0, 0, 0, 1, 2, 0, 0, 0,
       0, 1, 4, 2])
```

[22]:
```
# Import the persim package and run the test in every direction down-to-up
import persim as pm

# Set an empty pairwise distance matrix for future bottleneck distance input
BNDDU = np.zeros((26,26))


# Change infinities to very large numbers
for i in range(26):
    dgmDU[i][np.isinf(dgmDU[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
    for j in range(26):
        BNDDU[i,j] = pm.bottleneck(dgmDU[i], dgmDU[j])

# The very large values should be set to 0 by bottleneck definity (since the␣
↪very large distances would be inifinity)
BNDDU[BNDDU>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
↪simplicity we use sklearn's Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering
```

```
# For now we set 3 clusters, but we can change this and look for better results␣
↪in our model. A lower number here will prevent overfitting.
clusteringDU = AgglomerativeClustering(n_clusters = 5,
                                       affinity = "precomputed",
                                       linkage = "average").fit(BNDDU)

# This will output a vector of length 26 representing a number for each letter␣
↪for this scan.
DU_test = clusteringDU.labels_
DU_test
```

[22]:
```
array([0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4, 0, 0, 0, 1, 2, 0, 0, 0,
       0, 1, 4, 2])
```

[23]:
```
# Import the persim package and run the test in every direction up-to-down
import persim as pm

# Set an empty pairwise distance matrix for future bottleneck distance input
BNDUD = np.zeros((26,26))


# Change infinities to very large numbers
for i in range(26):
    dgmUD[i][np.isinf(dgmUD[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
    for j in range(26):
        BNDUD[i,j] = pm.bottleneck(dgmUD[i], dgmUD[j])

# The very large values should be set to 0 by bottleneck definity (since the␣
↪very large distances would be inifinity)
BNDUD[BNDUD>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
↪simplicity we use sklearn's Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering

# For now we set 3 clusters, but we can change this and look for better results␣
↪in our model. A lower number here will prevent overfitting.
clusteringUD = AgglomerativeClustering(n_clusters = 5,
                                       affinity = "precomputed",
                                       linkage = "average").fit(BNDUD)

# This will output a vector of length 26 representing a number for each letter␣
↪for this scan.
UD_test = clusteringUD.labels_
```

```
UD_test
```

[23]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[24]:
```
# Import the persim package and run the test in every direction angle␣
 ↪from-upper-left
import persim as pm

# Set an empty pairwise distance matrix for future bottleneck distance input
BNDAngle = np.zeros((26,26))


# Change infinities to very large numbers
for i in range(26):
    dgmAngle[i][np.isinf(dgmAngle[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
    for j in range(26):
        BNDAngle[i,j] = pm.bottleneck(dgmAngle[i], dgmAngle[j])

# The very large values should be set to 0 by bottleneck definity (since the␣
 ↪very large distances would be inifinity)
BNDAngle[BNDAngle>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
 ↪simplicity we use sklearn's Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering

# For now we set 3 clusters, but we can change this and look for better results␣
 ↪in our model. A lower number here will prevent overfitting.
clusteringAngle = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(BNDAngle)

# This will output a vector of length 26 representing a number for each letter␣
 ↪for this scan.
A_test = clusteringAngle.labels_
A_test
```

[24]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[25]:
```
# Import the persim package and run the test in every diagonally␣
 ↪from-lower-left
import persim as pm
```

```
# Set an empty pairwise distance matrix for future bottleneck distance input
BNDDiagonal = np.zeros((26,26))


# Change infinities to very large numbers
for i in range(26):
    dgmDiagonal[i][np.isinf(dgmDiagonal[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
    for j in range(26):
        BNDDiagonal[i,j] = pm.bottleneck(dgmDiagonal[i], dgmDiagonal[j])

# The very large values should be set to 0 by bottleneck definity (since the␣
 ↪very large distances would be inifinity)
BNDDiagonal[BNDDiagonal>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
 ↪simplicity we use sklearn's Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering

# For now we set 3 clusters, but we can change this and look for better results␣
 ↪in our model. A lower number here will prevent overfitting.
clusteringDiagonal = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(BNDDiagonal)

# This will output a vector of length 26 representing a number for each letter␣
 ↪for this scan.
D_test = clusteringDiagonal.labels_
D_test
```

[25]:
```
array([2, 0, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 1, 0, 1, 4, 1, 3,
       0, 2, 0, 0])
```

[26]:
```
# Import the persim package and run the test in every direction left-to-right
import persim as pm

# Set an empty pairwise distance matrix for future bottleneck distance input
BNDULC = np.zeros((26,26))


# Change infinities to very large numbers
for i in range(26):
    dgmDiagonalULC[i][np.isinf(dgmDiagonalULC[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
```

```
    for j in range(26):
        BNDULC[i,j] = pm.bottleneck(dgmDiagonalULC[i], dgmDiagonalULC[j])

# The very large values should be set to 0 by bottleneck definity (since the␣
 ↪very large distances would be inifinity)
BNDULC[BNDULC>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
 ↪simplicity we use sklearn's Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering

# For now we set 3 clusters, but we can change this and look for better results␣
 ↪in our model. A lower number here will prevent overfitting.
clusteringULC = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(BNDULC)

# This will output a vector of length 26 representing a number for each letter␣
 ↪for this scan.
ULC_test = clusteringULC.labels_
ULC_test
```

[26]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 3, 1,
       4, 0, 0, 0])
```

[27]:
```
# Import the persim package and run the test in every direction left-to-right
import persim as pm

# Set an empty pairwise distance matrix for future bottleneck distance input
BNDALF = np.zeros((26,26))



# Change infinities to very large numbers
for i in range(26):
    dgmAngleLF[i][np.isinf(dgmAngleLF[i])] = 10000

# Calculate bottleneck distances and input into the pairwise matrix
for i in range(26):
    for j in range(26):
        BNDALF[i,j] = pm.bottleneck(dgmAngleLF[i], dgmAngleLF[j])

# The very large values should be set to 0 by bottleneck definity (since the␣
 ↪very large distances would be inifinity)
BNDALF[BNDALF>1000]=0

# Now we look to perform clustering on this pairwise distance matrix. For␣
 ↪simplicity we use sklearn's Agglomerative Clustering
```

```
from sklearn.cluster import AgglomerativeClustering

# For now we set 3 clusters, but we can change this and look for better results␣
 ↪in our model. A lower number here will prevent overfitting.
clusteringALF = AgglomerativeClustering(n_clusters = 5,
                                        affinity = "precomputed",
                                        linkage = "average").fit(BNDULC)

# This will output a vector of length 26 representing a number for each letter␣
 ↪for this scan.
ALF_test = clusteringALF.labels_
ALF_test
```

[27]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 3, 1,
       4, 0, 0, 0])
```

[28]:
```
test_array = np.concatenate((LR_test, RL_test, DU_test, UD_test, A_test,␣
 ↪D_test, ULC_test, ALF_test))
test_array
```

[28]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4, 0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4, 0, 0, 0, 1,
       2, 0, 0, 0, 0, 1, 4, 2, 0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4,
       0, 0, 0, 1, 2, 0, 0, 0, 0, 1, 4, 2, 2, 1, 2, 1, 1, 1, 2, 1, 0, 0,
       1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2, 2, 1, 1, 4, 2, 1, 2, 1, 1, 1,
       2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2, 2, 1, 1, 4, 2, 0,
       0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 1, 0, 1, 4, 1, 3, 0, 2,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0,
       3, 1, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 2, 0, 3, 1, 4, 0, 0, 0])
```

[29]:
```
np.shape(test_array)
```

[29]:
```
(208,)
```

[30]:
```
test_array = np.reshape(test_array,(8,26)).T
```

[31]:
```
test_array
```

[31]:
```
array([[2, 0, 0, 2, 2, 2, 0, 0],
       [1, 0, 0, 1, 1, 0, 0, 0],
       [2, 2, 2, 2, 2, 0, 0, 0],
       [1, 0, 0, 1, 1, 0, 0, 0],
       [1, 3, 3, 1, 1, 1, 0, 0],
       [1, 2, 2, 1, 1, 0, 0, 0],
       [2, 2, 2, 2, 2, 2, 0, 0],
       [1, 4, 4, 1, 1, 0, 0, 0],
       [0, 1, 1, 0, 0, 2, 0, 0],
       [0, 1, 1, 0, 0, 0, 0, 0],
       [1, 2, 2, 1, 1, 0, 0, 0],
       [1, 4, 4, 1, 1, 0, 0, 0],
```

```
        [2, 0, 0, 2, 2, 0, 0, 0],
        [2, 4, 4, 2, 2, 2, 1, 1],
        [2, 0, 0, 2, 2, 0, 0, 0],
        [1, 0, 0, 1, 1, 0, 0, 0],
        [2, 0, 0, 2, 2, 1, 0, 0],
        [1, 1, 1, 1, 1, 0, 0, 0],
        [3, 2, 2, 3, 3, 1, 2, 2],
        [2, 0, 0, 2, 2, 4, 0, 0],
        [1, 0, 0, 1, 1, 1, 3, 3],
        [2, 0, 0, 2, 2, 3, 1, 1],
        [2, 0, 0, 2, 2, 0, 4, 4],
        [1, 1, 1, 1, 1, 2, 0, 0],
        [1, 4, 4, 1, 1, 0, 0, 0],
        [4, 2, 2, 4, 4, 0, 0, 0]])
```

1.0.1 Same Distances appear

From the test_array we can see that there are multiple like terms.

[32]:
```
test_array[14:17]
```

[32]:
```
array([[2, 0, 0, 2, 2, 0, 0, 0],
       [1, 0, 0, 1, 1, 0, 0, 0],
       [2, 0, 0, 2, 2, 1, 0, 0]])
```

[33]:
```
np.shape(test_array)
```

[33]:
```
(26, 8)
```

[34]:
```
test_array[:,0]
```

[34]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[35]:
```
LR_test
```

[35]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[36]:
```
test_array[:,1]
```

[36]:
```
array([0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4, 0, 0, 0, 1, 2, 0, 0, 0,
       0, 1, 4, 2])
```

[37]:
```
RL_test
```

[37]:
```
array([0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4, 0, 0, 0, 1, 2, 0, 0, 0,
       0, 1, 4, 2])
```

[38]:
```
test_array[:,2]
```

[38]:
```
array([0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4, 0, 0, 0, 1, 2, 0, 0, 0,
       0, 1, 4, 2])
```

[41]:
```
UD_test
```

[41]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[42]:
```
test_array[:,3]
```

[42]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[43]:
```
DU_test
```

[43]:
```
array([0, 0, 2, 0, 3, 2, 2, 4, 1, 1, 2, 4, 0, 4, 0, 0, 0, 1, 2, 0, 0, 0,
       0, 1, 4, 2])
```

[44]:
```
test_array[:,4]
```

[44]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[50]:
```
D_test
```

[50]:
```
array([2, 0, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 1, 0, 1, 4, 1, 3,
       0, 2, 0, 0])
```

[51]:
```
test_array[:,5]
```

[51]:
```
array([2, 0, 0, 0, 1, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 1, 0, 1, 4, 1, 3,
       0, 2, 0, 0])
```

[52]:
```
ULC_test
```

[52]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 3, 1,
       4, 0, 0, 0])
```

[53]:
```
A_test
```

[53]:
```
array([2, 1, 2, 1, 1, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 3, 2, 1, 2,
       2, 1, 1, 4])
```

[54]:
```
ALF_test
```

[54]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 3, 1,
       4, 0, 0, 0])
```

2 Observing linear dipendencies

LR, DU, Angle

RL, UD all have similar characteristics

Diagonal scanning provided unique characteristics

[55]:
```
# A test to differentiate some letters
bottom_test = [None]*26
```

```
for i in range(26):
    bottom_test[i]=sum(letters[i][51:101])
```

[56]:
```
right_test = [None]*26
for i in range(26):
    right_test[i]=sum(np.concatenate((letters[i][6:11],
              letters[i][16:21],
              letters[i][26:31],
              letters[i][36:41],
              letters[i][46:51],
              letters[i][56:61],
              letters[i][66:71],
              letters[i][76:81],
              letters[i][86:91],
              letters[i][96:101]
              )))
```

[57]:
```
botright = [None]*26
for i in range(26):
    botright[i] = sum(np.concatenate((
              letters[i][56:61],
              letters[i][66:71],
              letters[i][76:81],
              letters[i][86:91],
              letters[i][96:101]
              )))
```

[58]:
```
top_test = [None]*26
for i in range(26):
    top_test[i] = sum(letters[i][1:51])
```

[59]:
```
density_test = [None]*26
for i in range(26):
    density_test[i] = sum(letters[i][1:101])
```

[60]:
```
# Let us combine our vectors into a workable array
test_array = np.concatenate((LR_test,
                             RL_test,
                             A_test,
                             D_test,
                             bottom_test,
                             right_test,
                             botright,
                             top_test,
                             density_test))
test_array = np.reshape(test_array,(9,26)).T
```

[61]:
```
# Grab our labels
training_labels = [None]*26
for i in range(26):
```

```
    training_labels[i] = letters[i][0]
```

[62]:
```
# Fit our model and see if it has 100% accuracy on training data
from sklearn.linear_model import LogisticRegression
LogReg=LogisticRegression()
LogReg.fit(test_array, training_labels)
y_pred = LogReg.predict(test_array)
y_pred
# It does! 100% accuracy. Next step is to create function that will take a new␣
↪letter input and output the prediction using this model.
```

```
/Users/enzo/anaconda2/lib/python2.7/site-
packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/enzo/anaconda2/lib/python2.7/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default
multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to
silence this warning.
  "this warning.", FutureWarning)
```

[62]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.])
```

[ ]:
```

```