Experiments

October 30, 2019

Experiment 1 Letter S Sparsity

[54]:
```
letters[18,]
```

[54]:
```
array([19.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

[55]:
```
newLetter=[19.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  1.,  1.,  1.,  0,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]
```

Now we look to perform all filtrations on the new letter, as well as the density tests.

[56]:
```
 #Left to Right Filtration
letter=np.full((10, 10), 100)

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

 # Right to Left Filtration
letter=np.full((10, 10), 100)

    # convert one line letter to 10x10 matrix replacing zeros with 100
for k in range(1,101):
```

```python
    if newLetter[k]==1.0:
        row=int((k-1)/10)
        column=(k-1)%10
        letter[row,column]=10-k%10
dgmNRL = lower_star_img(letter)

 # Angle Filtration
letter=np.full((10, 10), 100)

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

 # Diagonal Filtration
letter=np.full((10, 10), 100)

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

# A test to differentiate some letters

bottom_test=sum(newLetter[51:101])


right_test=sum(np.concatenate((newLetter[6:11],
              newLetter[16:21],
              newLetter[26:31],
              newLetter[36:41],
              newLetter[46:51],
              newLetter[56:61],
              newLetter[66:71],
              newLetter[76:81],
              newLetter[86:91],
              newLetter[96:101]
              )))

botright = sum(np.concatenate((
              newLetter[56:61],
```

```
              newLetter[66:71],
              newLetter[76:81],
              newLetter[86:91],
              newLetter[96:101]
              )))

top_test = sum(newLetter[1:51])

density_test = sum(newLetter[1:101])
```

```
C:\Users\Putts\Anaconda3\lib\site-packages\ripser\ripser.py:342: RuntimeWarning:
invalid value encountered in maximum
  thisD = np.maximum(thisD, tD)
```

Now we use the filtrations to find the bottleneck distance between the new letter and all of the old ones.

[57]:
```
# Change infinities to very large numbers
dgmNLR[np.isinf(dgmNLR)] = 10000
dgmNRL[np.isinf(dgmNRL)] = 10000
dgmNA[np.isinf(dgmNA)] = 10000
dgmND[np.isinf(dgmND)] = 10000

# Find bottleneck distance between new letter and previous letters

# Left to Right
# Calculate bottleneck distances and input into the pairwise matrix
BNDNLR = [None]*26
for i in range(26):
    BNDNLR[i] = pm.bottleneck(dgmLR[i],dgmNLR)
BNDNLR = np.array(BNDNLR)
BNDNLR[BNDNLR>1000]=0

# Right to Left
# Calculate bottleneck distances and input into the pairwise matrix
BNDNRL = [None]*26
for i in range(26):
    BNDNRL[i] = pm.bottleneck(dgmRL[i],dgmNRL)
BNDNRL = np.array(BNDNRL)
BNDNRL[BNDNRL>1000]=0

# Angle
# Calculate bottleneck distances and input into the pairwise matrix
BNDNA = [None]*26
for i in range(26):
    BNDNA[i] = pm.bottleneck(dgmAngle[i],dgmNA)
BNDNA = np.array(BNDNA)
BNDNA[BNDNA>1000]=0
```

```python
# Diagonoal
# Calculate bottleneck distances and input into the pairwise matrix
BNDND = [None]*26
for i in range(26):
    BNDND[i] = pm.bottleneck(dgmDiagonal[i],dgmND)
BNDND = np.array(BNDND)
BNDND[BNDND>1000]=0
```

Convert them into their values using Agglomerative Clustering

[58]:
```python
# Left to Right Value
temp=np.vstack((BNDLR,BNDNLR))
temp2=np.append(BNDNLR,0)
NewBNDLR = np.hstack((temp, np.atleast_2d(temp2).T))
LRClust = AgglomerativeClustering(n_clusters = 5,
                                  affinity = "precomputed",
                                  linkage = "average").fit(NewBNDLR)
LRValue=LRClust.labels_[26]

# Right to Left Value
temp=np.vstack((BNDRL,BNDNRL))
temp2=np.append(BNDNRL,0)
NewBNDRL = np.hstack((temp, np.atleast_2d(temp2).T))
RLClust = AgglomerativeClustering(n_clusters = 5,
                                  affinity = "precomputed",
                                  linkage = "average").fit(NewBNDRL)
RLValue=RLClust.labels_[26]


# Angle Value
temp=np.vstack((BNDAngle,BNDNA))
temp2=np.append(BNDNA,0)
NewBNDA = np.hstack((temp, np.atleast_2d(temp2).T))
AngleClust = AgglomerativeClustering(n_clusters = 5,
                                  affinity = "precomputed",
                                  linkage = "average").fit(NewBNDA)
AValue=AngleClust.labels_[26]


# Diagonal Value
temp=np.vstack((BNDDiagonal,BNDND))
temp2=np.append(BNDND,0)
NewBNDD = np.hstack((temp, np.atleast_2d(temp2).T))
DiagonalClust = AgglomerativeClustering(n_clusters = 5,
                                  affinity = "precomputed",
                                  linkage = "average").fit(NewBNDD)
DValue=DiagonalClust.labels_[26]
```

Combine these values into one vector and run our model on that vector

[59]:
```
New_Letter = np.array((LRValue,
                             RLValue,
                             AValue,
                             DValue,
                             bottom_test,
                             right_test,
                             botright,
                             top_test,
                             density_test))
LogReg.predict(New_Letter.reshape(1,-1))[0]
```

[59]:
```
19.0
```

A slightly sparse S is classified correctly

Experiment 2: P Sparsity

[112]:
```
letters[15,]
```

[112]:
```
array([16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  1.,
        0.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,
        0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

[118]:
```
newLetter=[16.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  1.,  1.,  0.,  0.,  0.,  0.,  0,  0.,  0.,  0.,  1.,  1.,
        0.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,
        0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]
```

Now we look to perform all filtrations on the new letter, as well as the density tests.

[119]:
```
 #Left to Right Filtration
letter=np.full((10, 10), 100)

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

 # Right to Left Filtration
```

```
letter=np.full((10, 10), 100)

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

 # Angle Filtration
letter=np.full((10, 10), 100)

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

 # Diagonal Filtration
letter=np.full((10, 10), 100)

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

# A test to differentiate some letters

bottom_test=sum(newLetter[51:101])


right_test=sum(np.concatenate((newLetter[6:11],
              newLetter[16:21],
              newLetter[26:31],
              newLetter[36:41],
              newLetter[46:51],
              newLetter[56:61],
              newLetter[66:71],
              newLetter[76:81],
              newLetter[86:91],
              newLetter[96:101]
```

```
              )))

botright = sum(np.concatenate((
              newLetter[56:61],
              newLetter[66:71],
              newLetter[76:81],
              newLetter[86:91],
              newLetter[96:101]
              )))

top_test = sum(newLetter[1:51])

density_test = sum(newLetter[1:101])
```

```
C:\Users\Putts\Anaconda3\lib\site-packages\ripser\ripser.py:342: RuntimeWarning:
invalid value encountered in maximum
  thisD = np.maximum(thisD, tD)
```

Now we use the filtrations to find the bottleneck distance between the new letter and all of the old ones.

[120]:
```
# Change infinities to very large numbers
dgmNLR[np.isinf(dgmNLR)] = 10000
dgmNRL[np.isinf(dgmNRL)] = 10000
dgmNA[np.isinf(dgmNA)] = 10000
dgmND[np.isinf(dgmND)] = 10000

# Find bottleneck distance between new letter and previous letters

# Left to Right
# Calculate bottleneck distances and input into the pairwise matrix
BNDNLR = [None]*26
for i in range(26):
    BNDNLR[i] = pm.bottleneck(dgmLR[i],dgmNLR)
BNDNLR = np.array(BNDNLR)
BNDNLR[BNDNLR>1000]=0

# Right to Left
# Calculate bottleneck distances and input into the pairwise matrix
BNDNRL = [None]*26
for i in range(26):
    BNDNRL[i] = pm.bottleneck(dgmRL[i],dgmNRL)
BNDNRL = np.array(BNDNRL)
BNDNRL[BNDNRL>1000]=0

# Angle
# Calculate bottleneck distances and input into the pairwise matrix
BNDNA = [None]*26
```

```
for i in range(26):
    BNDNA[i] = pm.bottleneck(dgmAngle[i],dgmNA)
BNDNA = np.array(BNDNA)
BNDNA[BNDNA>1000]=0

# Diagonoal
# Calculate bottleneck distances and input into the pairwise matrix
BNDND = [None]*26
for i in range(26):
    BNDND[i] = pm.bottleneck(dgmDiagonal[i],dgmND)
BNDND = np.array(BNDND)
BNDND[BNDND>1000]=0
```

Convert them into their values using Agglomerative Clustering

[121]:
```
# Left to Right Value
temp=np.vstack((BNDLR,BNDNLR))
temp2=np.append(BNDNLR,0)
NewBNDLR = np.hstack((temp, np.atleast_2d(temp2).T))
LRClust = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(NewBNDLR)
LRValue=LRClust.labels_[26]

# Right to Left Value
temp=np.vstack((BNDRL,BNDNRL))
temp2=np.append(BNDNRL,0)
NewBNDRL = np.hstack((temp, np.atleast_2d(temp2).T))
RLClust = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(NewBNDRL)
RLValue=RLClust.labels_[26]


# Angle Value
temp=np.vstack((BNDAngle,BNDNA))
temp2=np.append(BNDNA,0)
NewBNDA = np.hstack((temp, np.atleast_2d(temp2).T))
AngleClust = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(NewBNDA)
AValue=AngleClust.labels_[26]


# Diagonal Value
temp=np.vstack((BNDDiagonal,BNDND))
temp2=np.append(BNDND,0)
NewBNDD = np.hstack((temp, np.atleast_2d(temp2).T))
```

```
DiagonalClust = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(NewBNDD)
DValue=DiagonalClust.labels_[26]
```

Combine these values into one vector and run our model on that vector

[122]:
```
New_Letter = np.array((LRValue,
                             RLValue,
                             AValue,
                             DValue,
                             bottom_test,
                             right_test,
                             botright,
                             top_test,
                             density_test))
LogReg.predict(New_Letter.reshape(1,-1))[0]
```

[122]:
```
16.0
```

The P is classified correctly.

Experiment 3: Sparse Q

[124]:
```
letters[16,]
```

[124]:
```
array([17.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,
        0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,
        1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  1.,  0.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,
        1.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

[135]:
```
newLetter=[17.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  1.,  0,  1.,  1.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,
        0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,
        1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  1.,  0.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,
        1.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]
```

Now we look to perform all filtrations on the new letter, as well as the density tests.

[136]:
```
 #Left to Right Filtration
letter=np.full((10, 10), 100)

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

```
        column=(k-1)%10
        letter[row,column]=k%10
dgmNLR = lower_star_img(letter)

 # Right to Left Filtration
letter=np.full((10, 10), 100)

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

 # Angle Filtration
letter=np.full((10, 10), 100)

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

 # Diagonal Filtration
letter=np.full((10, 10), 100)

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

# A test to differentiate some letters

bottom_test=sum(newLetter[51:101])


right_test=sum(np.concatenate((newLetter[6:11],
              newLetter[16:21],
              newLetter[26:31],
              newLetter[36:41],
              newLetter[46:51],
```

```
                newLetter[56:61],
                newLetter[66:71],
                newLetter[76:81],
                newLetter[86:91],
                newLetter[96:101]
                )))

botright = sum(np.concatenate((
                newLetter[56:61],
                newLetter[66:71],
                newLetter[76:81],
                newLetter[86:91],
                newLetter[96:101]
                )))

top_test = sum(newLetter[1:51])

density_test = sum(newLetter[1:101])
```

```
C:\Users\Putts\Anaconda3\lib\site-packages\ripser\ripser.py:342: RuntimeWarning:
invalid value encountered in maximum
  thisD = np.maximum(thisD, tD)
```

Now we use the filtrations to find the bottleneck distance between the new letter and all of the old ones.

[137]:
```
# Change infinities to very large numbers
dgmNLR[np.isinf(dgmNLR)] = 10000
dgmNRL[np.isinf(dgmNRL)] = 10000
dgmNA[np.isinf(dgmNA)] = 10000
dgmND[np.isinf(dgmND)] = 10000

# Find bottleneck distance between new letter and previous letters

# Left to Right
# Calculate bottleneck distances and input into the pairwise matrix
BNDNLR = [None]*26
for i in range(26):
    BNDNLR[i] = pm.bottleneck(dgmLR[i],dgmNLR)
BNDNLR = np.array(BNDNLR)
BNDNLR[BNDNLR>1000]=0

# Right to Left
# Calculate bottleneck distances and input into the pairwise matrix
BNDNRL = [None]*26
for i in range(26):
    BNDNRL[i] = pm.bottleneck(dgmRL[i],dgmNRL)
BNDNRL = np.array(BNDNRL)
```

```
BNDNRL[BNDNRL>1000]=0

# Angle
# Calculate bottleneck distances and input into the pairwise matrix
BNDNA = [None]*26
for i in range(26):
    BNDNA[i] = pm.bottleneck(dgmAngle[i],dgmNA)
BNDNA = np.array(BNDNA)
BNDNA[BNDNA>1000]=0

# Diagonoal
# Calculate bottleneck distances and input into the pairwise matrix
BNDND = [None]*26
for i in range(26):
    BNDND[i] = pm.bottleneck(dgmDiagonal[i],dgmND)
BNDND = np.array(BNDND)
BNDND[BNDND>1000]=0
```

Convert them into their values using Agglomerative Clustering

[138]:
```
# Left to Right Value
temp=np.vstack((BNDLR,BNDNLR))
temp2=np.append(BNDNLR,0)
NewBNDLR = np.hstack((temp, np.atleast_2d(temp2).T))
LRClust = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(NewBNDLR)
LRValue=LRClust.labels_[26]

# Right to Left Value
temp=np.vstack((BNDRL,BNDNRL))
temp2=np.append(BNDNRL,0)
NewBNDRL = np.hstack((temp, np.atleast_2d(temp2).T))
RLClust = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(NewBNDRL)
RLValue=RLClust.labels_[26]


# Angle Value
temp=np.vstack((BNDAngle,BNDNA))
temp2=np.append(BNDNA,0)
NewBNDA = np.hstack((temp, np.atleast_2d(temp2).T))
AngleClust = AgglomerativeClustering(n_clusters = 5,
                                     affinity = "precomputed",
                                     linkage = "average").fit(NewBNDA)
AValue=AngleClust.labels_[26]
```

```python
# Diagonal Value
temp=np.vstack((BNDDiagonal,BNDND))
temp2=np.append(BNDND,0)
NewBNDD = np.hstack((temp, np.atleast_2d(temp2).T))
DiagonalClust = AgglomerativeClustering(n_clusters = 5,
                                        affinity = "precomputed",
                                        linkage = "average").fit(NewBNDD)
DValue=DiagonalClust.labels_[26]
```

Combine these values into one vector and run our model on that vector

[139]:
```python
New_Letter = np.array((LRValue,
                             RLValue,
                             AValue,
                             DValue,
                             bottom_test,
                             right_test,
                             botright,
                             top_test,
                             density_test))
LogReg.predict(New_Letter.reshape(1,-1))[0]
```

[139]: 17.0

The Sparse Q is also classified correctly.

[ ]:
```python

```