Lab8

December 1, 2016

0.1 Laboratorio 8: Aprendizaje no supervisado

Manuel Felipe Pineda ~ 1093223607

- Identifique las caracteristicas y las muestras de su base de datos de interes (proyecto asignatura). Que tipo de variables debe procesar?

  R: La base de datos cuenta con 53 caracteristicas y 486048 muestras. Las muestras corresponden a diferentes usuarios de la plataforma y las caracteristicas proveen informacion de dichos usuarios, por ejemplo su participacion en diferentes contest.

- Cual es la hipotesis u objetivo que pretende comprobar en su aplicacion? relaciones entre caracterısticas? relaciones entre muestras? ambas? Para que?

  R: El objetivo es encontrar relaciones entre muestras y caracteristicas con el fin de detectar cuando un email va a ser abierto por un participante.

- De acuerdo a lo que desea procesar en su base de datos, es necesario aplicar algun tipo de normalizacion? Por que?

  R: Es importante realizar una normalizacion debido a que los datos vienen en diferentes medidas, por ejemplo algunas representan medidas de tiempo, fecuencias o identificadores.

  En este caso se usara una normalizacion robusta para evitar errores que puedan ser introducidos por los outliers.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        from mpl_toolkits.mplot3d import Axes3D
        from sklearn.decomposition import PCA, KernelPCA
```

```
In [2]: X = np.loadtxt('./final/data.txt')
        Y = np.loadtxt('./final/label.txt').astype(int)

        X = np.nan_to_num(X)
```

```
In [3]: print ('%d Samples, %d features' % (X.shape[0], X.shape[1]))
```

```
486048 Samples, 53 features
```

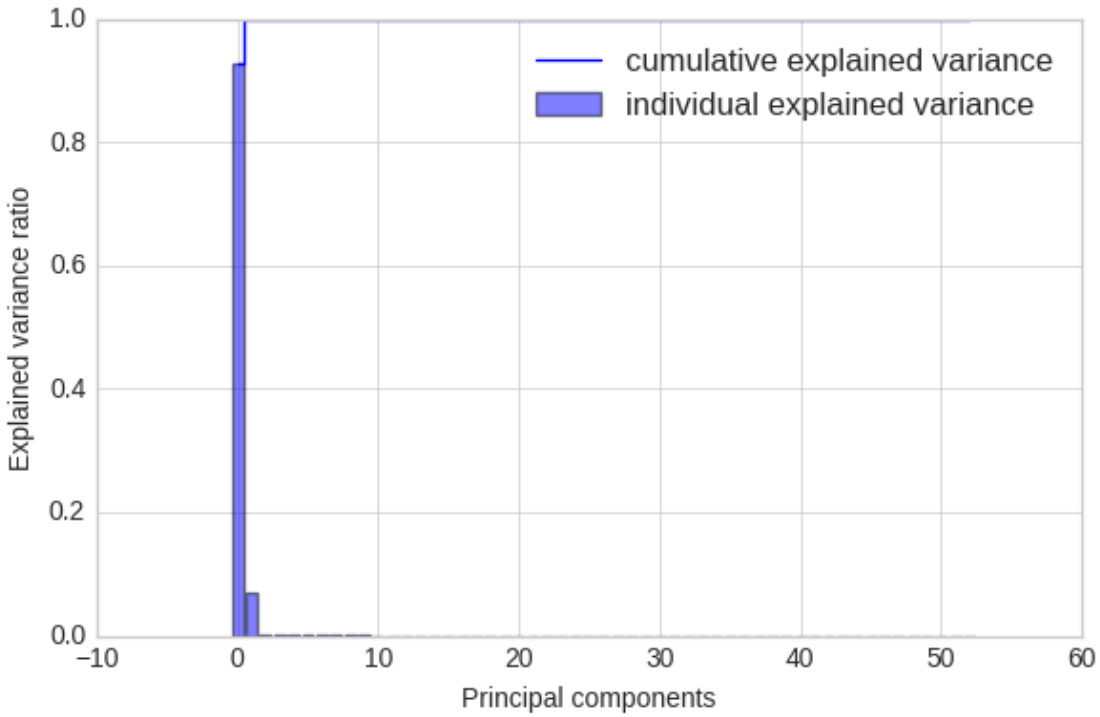
```
In [4]: from sklearn.preprocessing import StandardScaler, RobustScaler

        standard_scaler = StandardScaler()
        Xstd = standard_scaler.fit_transform(X)

        robust_scaler = RobustScaler()
        Xrob = robust_scaler.fit_transform(X)

        Xscaled = Xrob
```

- Aplique el algoritmo de Analisis de Componentes Principales - (PCA) sobre su base de dato. Con cuantas componentes es pertinente representar sus datos para conservar el 95 % de la variabilidad de las muestras de entrada?

```
In [5]: var = 0.95
        pca = PCA(n_components=var)
        pca.fit(Xscaled)
        print ('Required components %d with %.2f of variance' %
               (len(pca.explained_variance_ratio_), var))
        print (pca.explained_variance_ratio_)

        def plot_ranking(pca):
            var_exp = pca.explained_variance_ratio_
            cum_var_exp = np.cumsum(var_exp)

            with plt.style.context('seaborn-whitegrid'):
                plt.figure(figsize=(6, 4))
                plt.bar(range(len(var_exp)), var_exp, alpha=0.5,
                        align='center',
                        label='individual explained variance')

                plt.step(range(len(var_exp)), cum_var_exp, where='mid',
                         label='cumulative explained variance')
                plt.ylabel('Explained variance ratio')
                plt.xlabel('Principal components')
                plt.legend(loc='best')
                plt.tight_layout()
                plt.show()
```

```
Required components 2 with 0.95 of variance
[ 0.9291205  0.0708795]
```

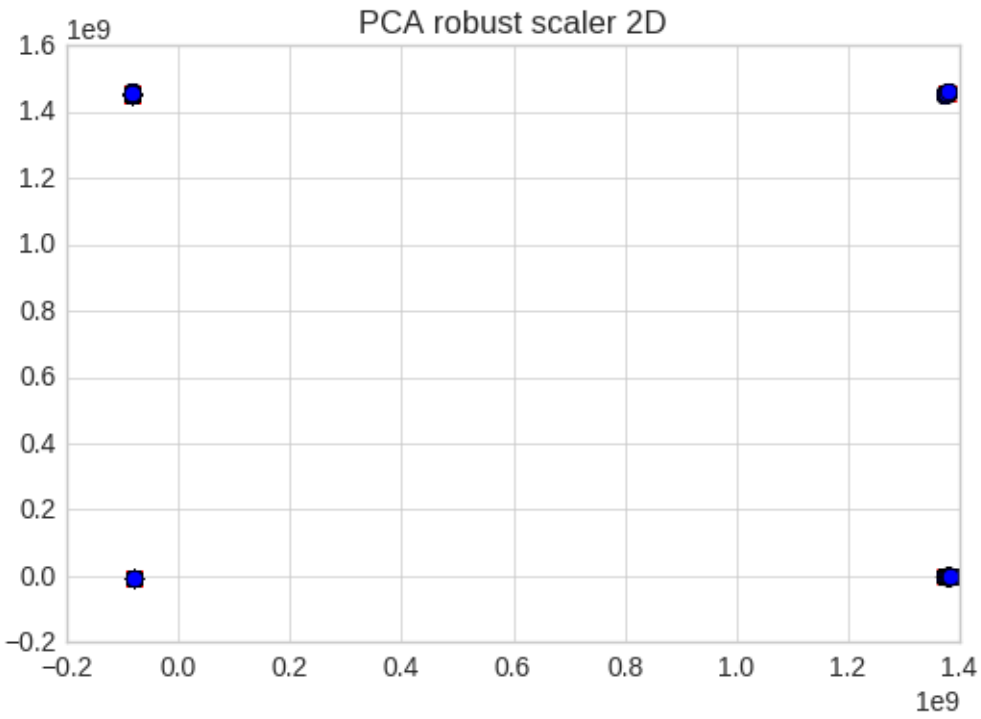
```
In [6]: full_pca = PCA()
        full_pca.fit(Xscaled)
        plot_ranking(full_pca)
```

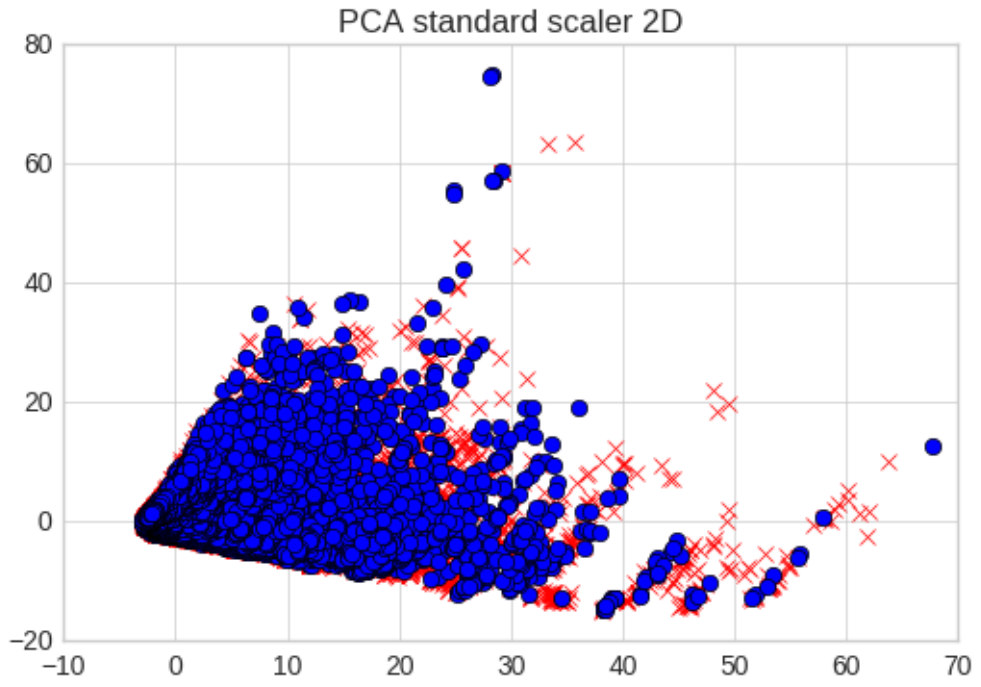


R: Se requieren 2 componentes principales para conservar el 95 % de la variabilidad de las muestras de entrada.

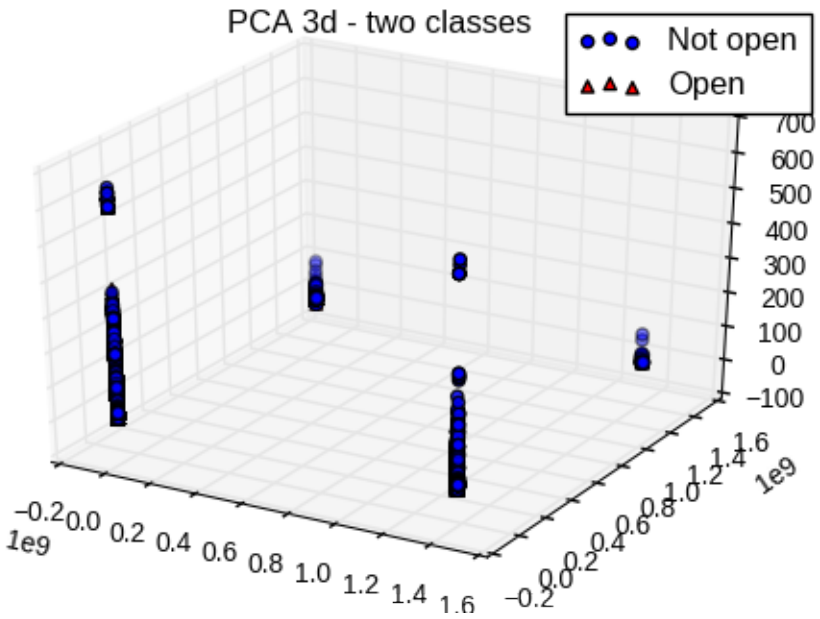
- Grafique el espacio proyectado de PCA y pinte en colores las muestras con base a alguna senal de referencia (etiquetas, senal de salida, etc.). Si su base de datos no posee alguna salida de referencia clara, utilice alguna de las caracteristicas de entrada como referencia para colorear las muestras.

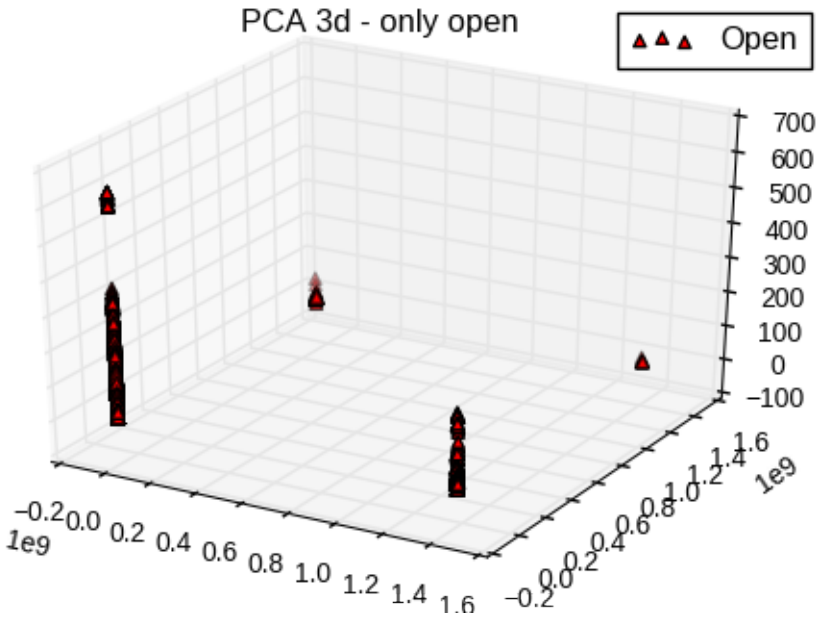
```
In [7]: def plot_pca_per(X, Y, per, pca, name):
            X_pca = pca.fit_transform(X)
            reds = Y == 0
            blues = Y == 1

            with plt.style.context('seaborn-whitegrid'):
                plt.plot(X_pca[reds, 0], X_pca[reds, 1], "rx")
                plt.plot(X_pca[blues, 0], X_pca[blues, 1], "bo")
                plt.title('%s 2D' % name)
                plt.show()


In [8]: pca = PCA(n_components=2)
        plot_pca_per(Xscaled, Y, 2, pca, 'PCA robust scaler')
        plot_pca_per(Xstd, Y, 2, pca, 'PCA standard scaler')
```


PCA robust scaler 2D
1e9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e9


PCA standard scaler 2D
80
60
40
20
0
−20
−10
0
10
20
30
40
50
60
70

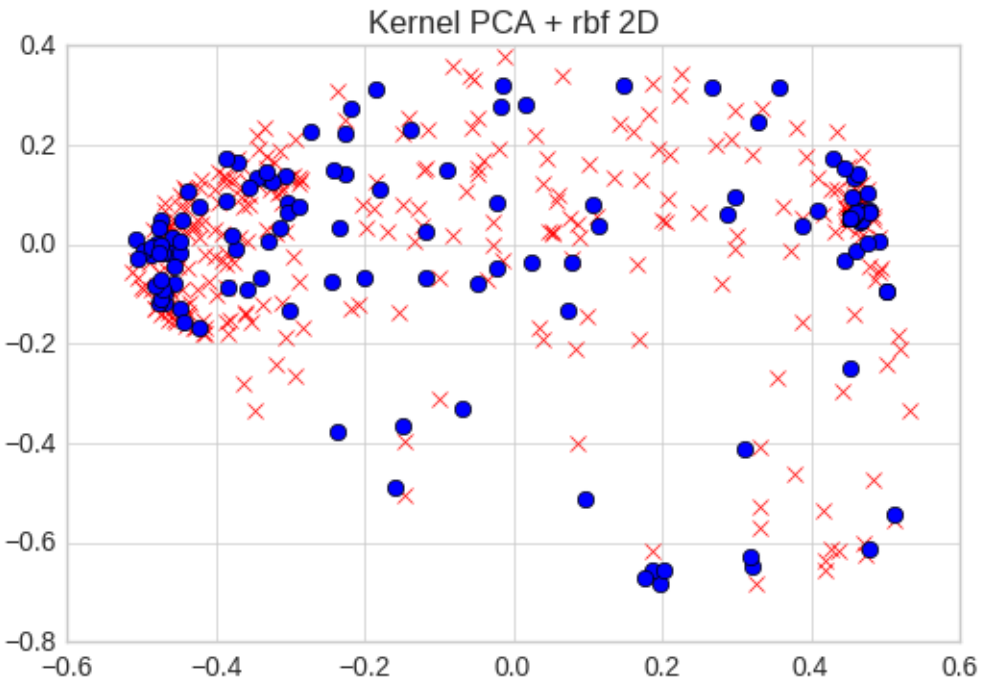
```
In [9]: def plot_pca_3d(X, Y, per, pca, name):
            X_pca = pca.fit_transform(X)
            reds = Y == 0
            blues = Y == 1

            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            ax.scatter(X_pca[blues, 0], X_pca[blues, 1],
                       X_pca[blues, 2], c='b', marker='o',
                       label='Not open')
            ax.scatter(X_pca[reds, 0], X_pca[reds, 1],
                       X_pca[reds, 2], c='r', marker='^',
                       label='Open')
            ax.legend()
            plt.title('PCA 3d - two classes')
            plt.show()
            fig = plt.figure()
            ax_red = fig.add_subplot(111, projection='3d')
            ax_red.scatter(X_pca[reds, 0], X_pca[reds, 1],
                           X_pca[reds, 2], c='r', marker='^',
                           label='Open')
            ax_red.legend()
            plt.title('PCA 3d - only open')
            plt.show()

In [10]: plot_pca_3d(Xscaled, Y, 3, PCA(), 'PCA 3d')
```

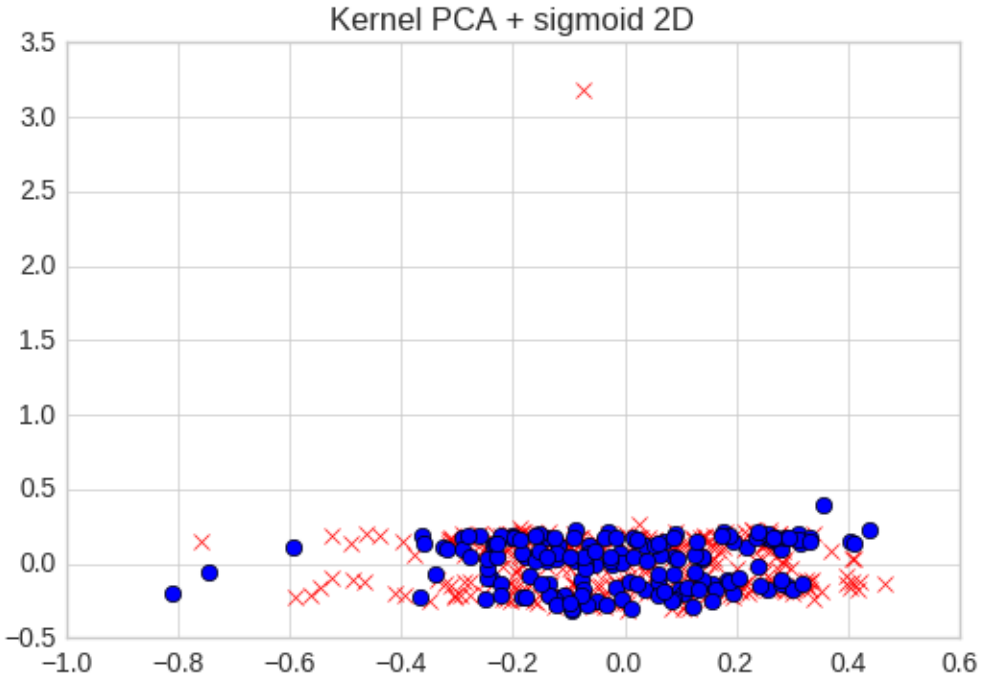


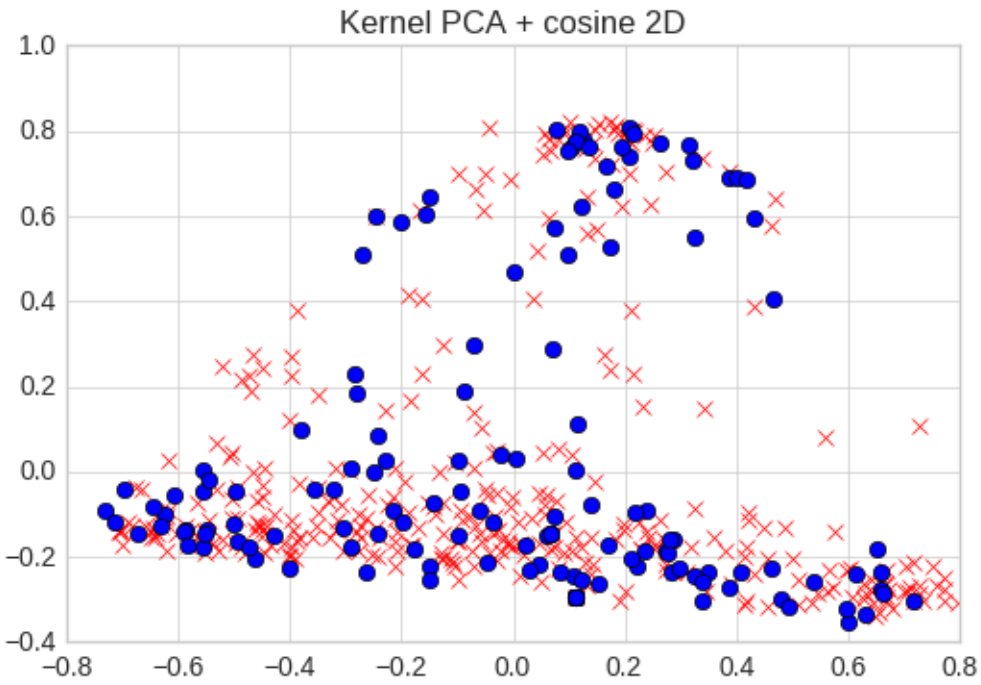


Punto anterior usando Kernel PCA

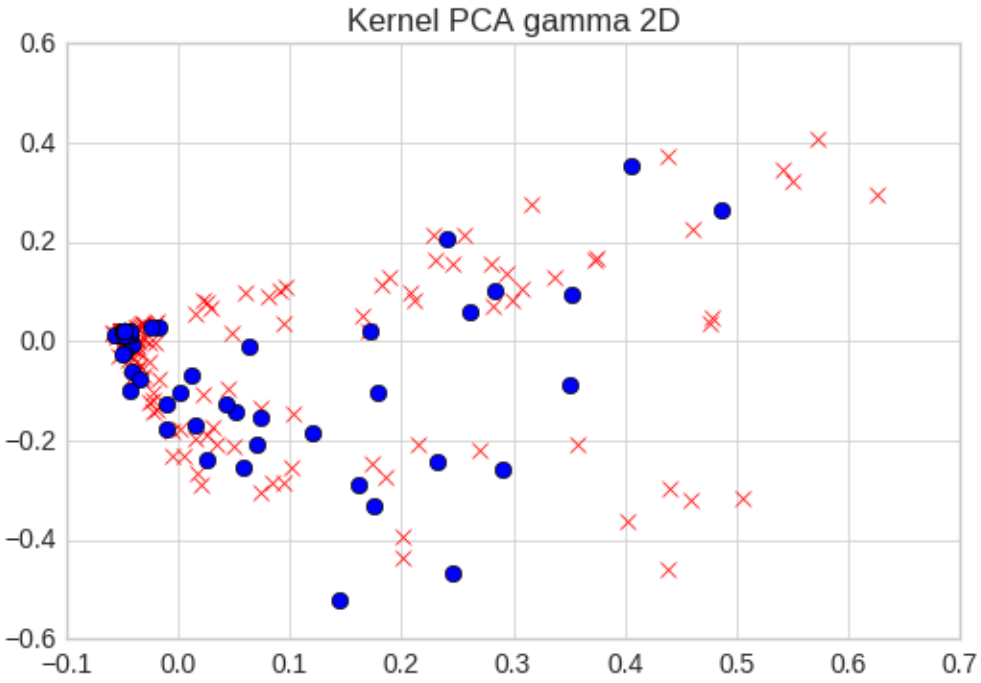
```
In [11]: from sklearn.utils import resample
         xp, yp = resample(Xscaled, Y, n_samples=500)

In [12]: kernels = ['rbf', 'sigmoid', 'cosine']
         for k in kernels:
             kpca = KernelPCA(kernel=k)
             plot_pca_per(xp, yp, 2, kpca, 'Kernel PCA + ' + k)
```


Kernel PCA + rbf 2D
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

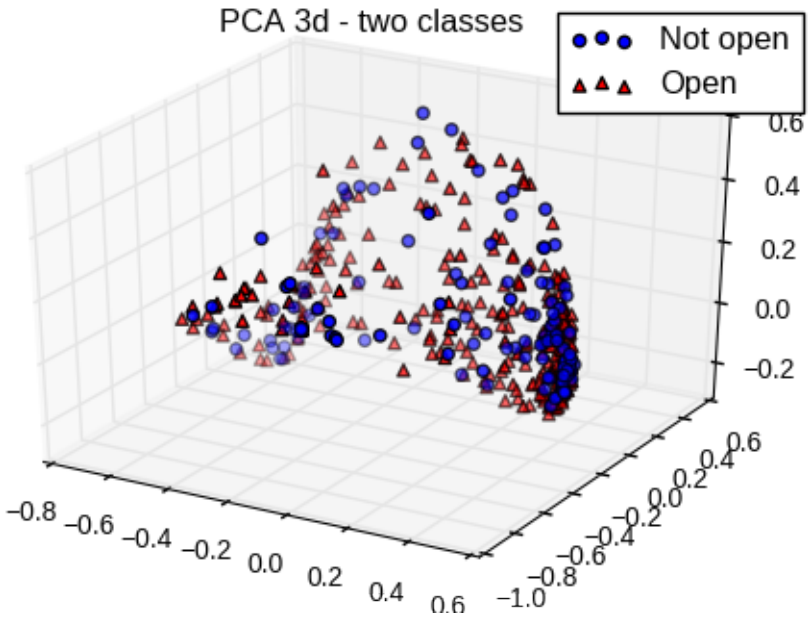

Kernel PCA + sigmoid 2D
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

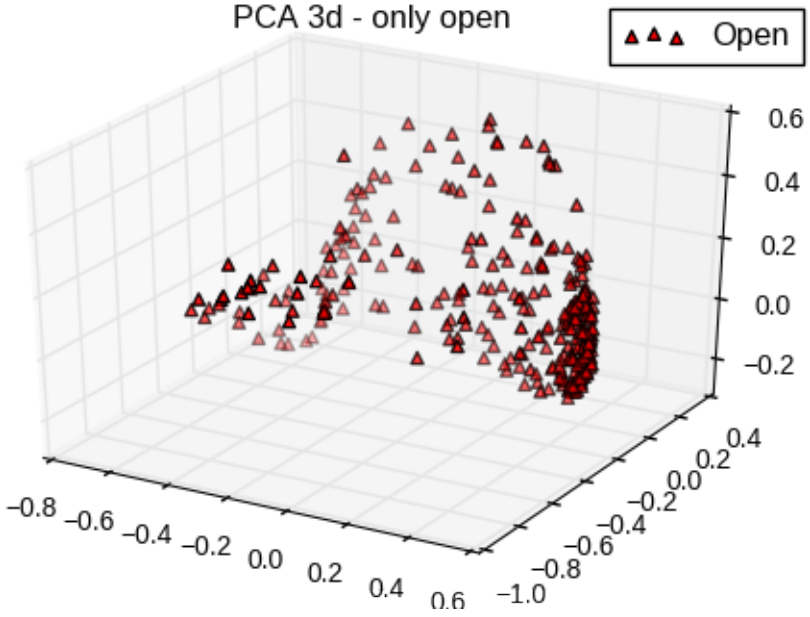


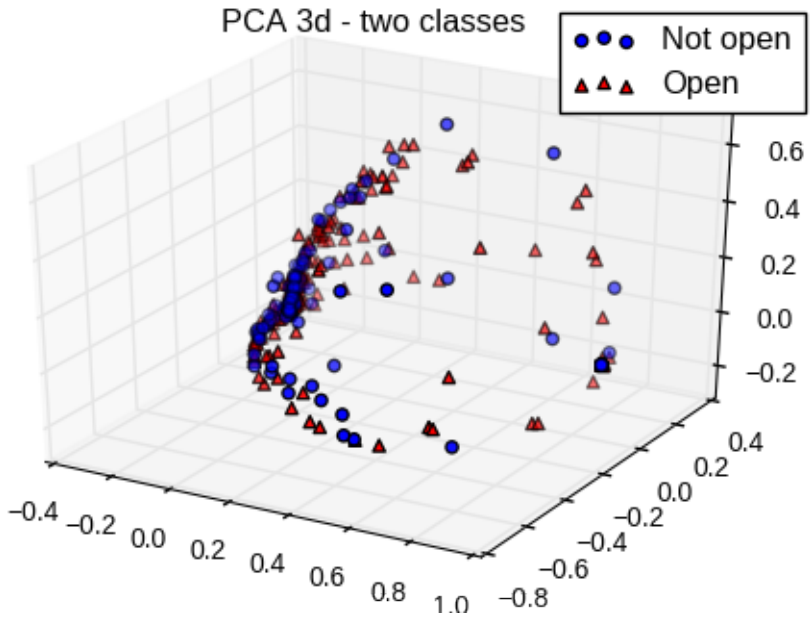
In [13]:
```
kpca = KernelPCA(kernel='rbf', gamma=0.5)
plot_pca_per(xp, yp, 2, kpca, 'Kernel PCA gamma')
```

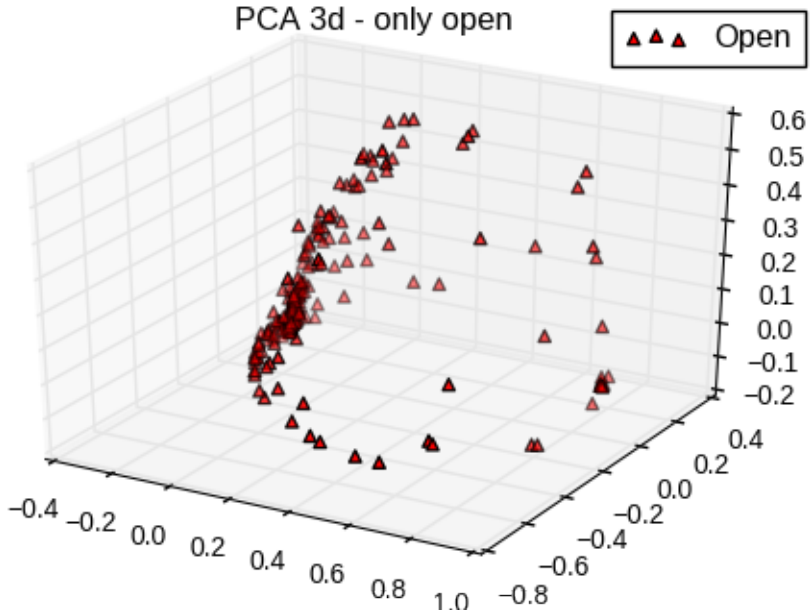


In [14]:
```
kpca = KernelPCA(kernel='rbf', gamma=0.01)
plot_pca_3d(xp, yp, 3, kpca, 'Kernel PCA gamma')
kpca = KernelPCA(kernel='rbf', gamma=0.001)
plot_pca_3d(xp, yp, 3, kpca, 'Kernel PCA gamma')
```





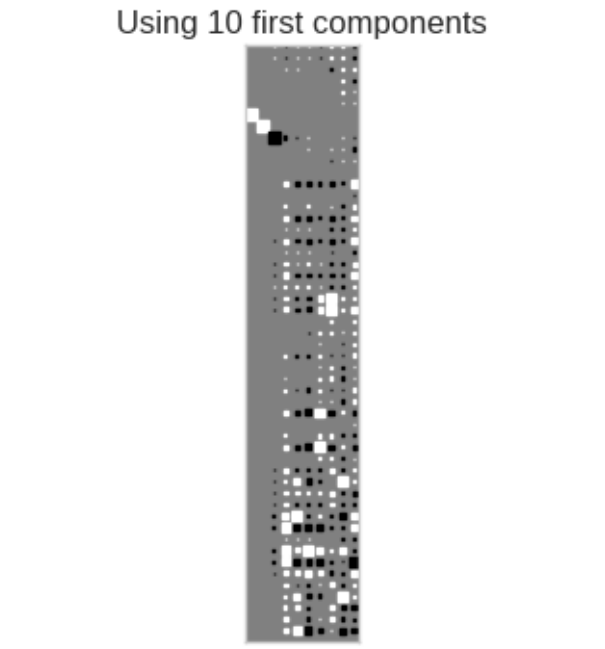


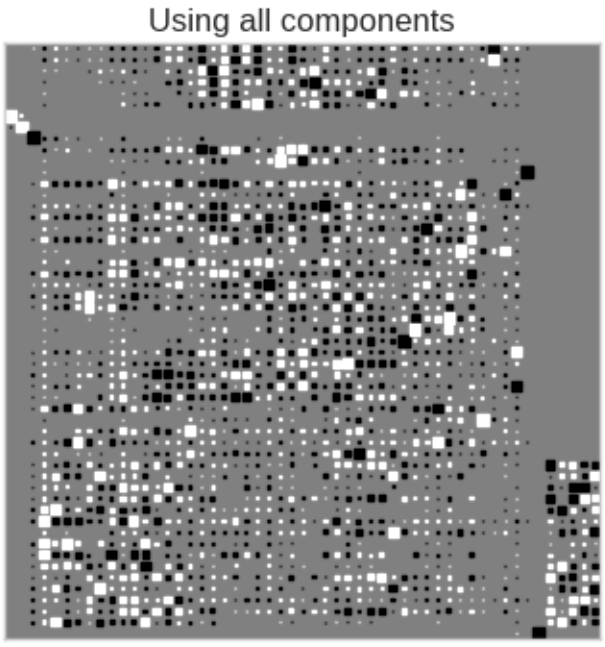


- Grafique la matriz de proyeccion de PCA, como podria cuantificar que tan importantes (ranking) son cada una de las caracteristicas de entrada segun el algoritmo PCA?

```
In [15]: def hinton(matrix, max_weight=None, ax=None, title=None):
             with plt.style.context('seaborn-whitegrid'):
```

```
        ax = ax if ax is not None else plt.gca()

        if not max_weight:
            max_weight = np.abs(matrix).max()

        ax.patch.set_facecolor('gray')
        ax.set_aspect('equal', 'box')
        ax.xaxis.set_major_locator(plt.NullLocator())
        ax.yaxis.set_major_locator(plt.NullLocator())

        for (x, y), w in np.ndenumerate(matrix):
            color = 'white' if w > 0 else 'black'
            size = np.sqrt(np.abs(w) / max_weight)
            rect = plt.Rectangle([x - size / 2, y - size / 2]
                                 , size, size,
                                 facecolor=color, edgecolor=color)
            ax.add_patch(rect)

        ax.autoscale_view()
        ax.invert_yaxis()
        plt.title(title)
        plt.show()

full_pca = PCA(n_components=10)
full_pca.fit(xp)
hinton(full_pca.components_, title='Using 10 first components')
full_pca = PCA()
full_pca.fit(xp)
hinton(full_pca.components_, title='Using all components')
```





R: Podemos recorrer cada una de las componentes principales y ver que caracteristicas aportan mas informacion a dicha componente.

Por ejemplo se puede usar el aporte a la variabilidad para ponderar la importancia de cada caracteristica.

```
In [16]: var = full_pca.explained_variance_
         W = np.abs(full_pca.components_)
         score = var.T.dot(W)
         print ('feature ranking:', np.argsort(score))

feature ranking: [13  0  1 18 33  9 43 21 25 26 16 24  4 28 14 10 19 31 52 47  2 50
 29  5 46 38 22 48 37 23 36 17 39 40 49 44 12 20 15 32 35 34 41 11 51  8 42
 45  7  6]
```

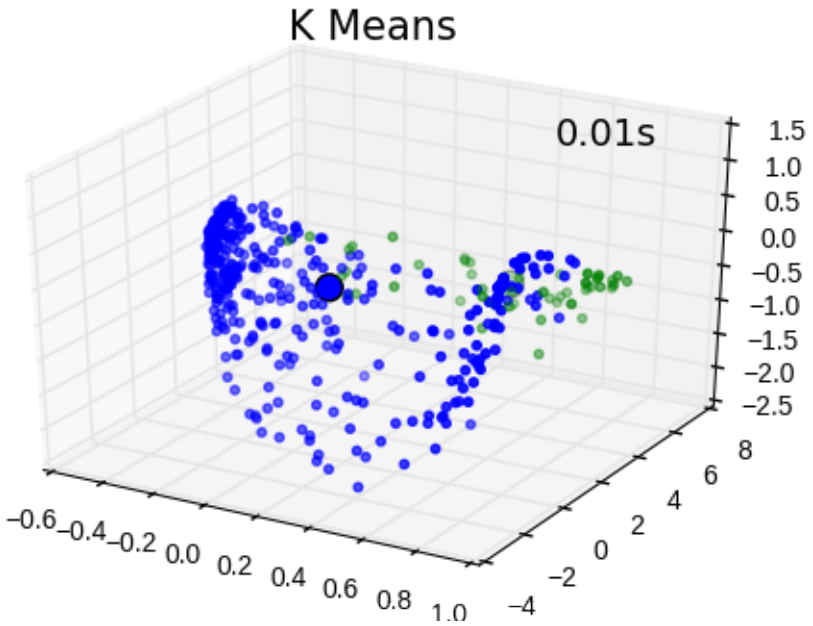
- Utilice el espacio proyectado de PCA como nuevo espacio de representacion, aplique los algoritmos de agrupamiento jerarquico, k-medias, y espectral. Pinte los grupos identificados por cada uno de los algoritmos de agrupamientos sobre el espacio de PCA en las primeras tres componentes, que informacion interesante puede revelar en sus datos?. Para el algoritmo de agrupamiento espectral pruebe con diferentes valores de ancho de banda para el kernel Gaussiano y diferentes tamanos de vecindario.

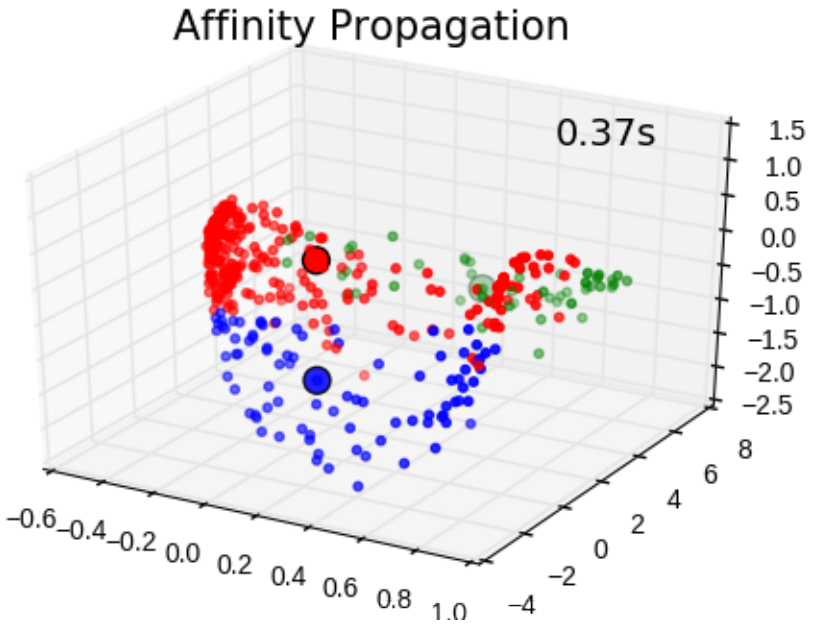
```
In [21]: from sklearn import cluster, datasets
         from sklearn.neighbors import kneighbors_graph
         import time

         kpca = KernelPCA(kernel='rbf', gamma=0.01, n_components=3)
         emails = kpca.fit_transform(xp), yp

         colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
         colors = np.hstack([colors] * 20)

         clustering_names = [
             'K Means', 'Affinity Propagation', 'Mean Shift',
             'Spectral Clustering', 'Spectral 2', 'Spectral 3',
             'Ward', 'Agglomerative Clustering',
             'DBSCAN', 'Birch']

         datasets = [emails]
         for i_dataset, dataset in enumerate(datasets):
             X, y = dataset
             # X = StandardScaler().fit_transform(X)
             X = RobustScaler().fit_transform(X)
             bandwidth = cluster.estimate_bandwidth(X, quantile=0.3)
             connectivity = kneighbors_graph(X, n_neighbors=10,
                                             include_self=False)
             connectivity = 0.5 * (connectivity + connectivity.T)

             # create clustering estimators
             ms = cluster.MeanShift(bandwidth=bandwidth, bin_seeding=True)
             two_means = cluster.MiniBatchKMeans(n_clusters=2)
             ward = cluster.AgglomerativeClustering(n_clusters=2,
                                                    linkage='ward',
```

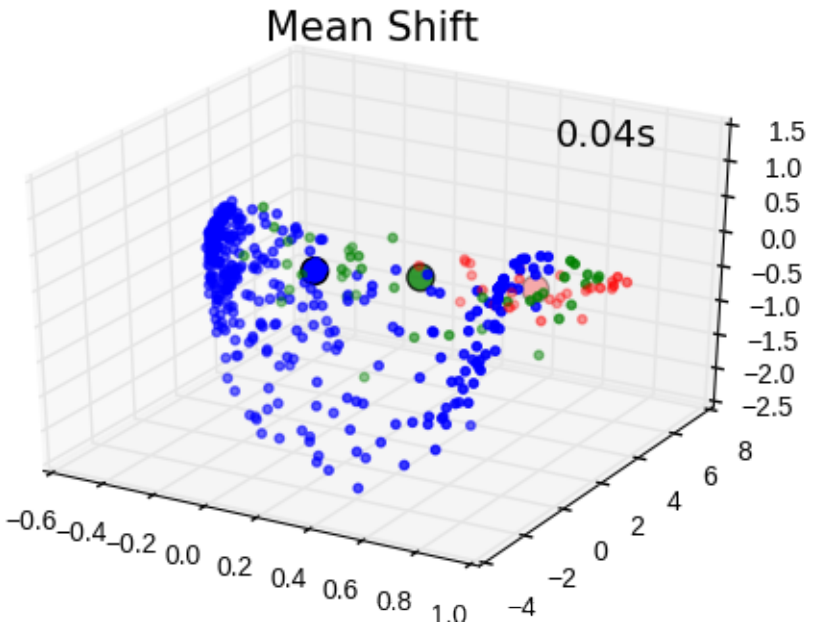
```
                                           connectivity=connectivity)
spectral = cluster.SpectralClustering(n_clusters=2,
                                      eigen_solver='arpack',
                                      gamma=10, n_neighbors=10)

spectral2 = cluster.SpectralClustering(n_clusters=2,
                                      eigen_solver='arpack',
                                      gamma=0.01, n_neighbors=10)

spectral3 = cluster.SpectralClustering(n_clusters=2,
                                      eigen_solver='arpack',
                                      gamma=1, n_neighbors=100)

dbscan = cluster.DBSCAN(eps=.2)
affinity_propagation = cluster.AffinityPropagation(damping=.9,
                                                   preference=-200)

average_linkage = cluster.AgglomerativeClustering(
    linkage="average", affinity="cityblock", n_clusters=2,
    connectivity=connectivity)

birch = cluster.Birch(n_clusters=2)
clustering_algorithms = [
    two_means, affinity_propagation, ms, spectral,
    spectral2, spectral3,
    ward, average_linkage, dbscan, birch]


# Twice as wide as it is tall.
fig = plt.figure(figsize=plt.figaspect(0.5))
for name, algorithm in zip(clustering_names, clustering_algorithms):
    # predict cluster memberships
    t0 = time.time()
    algorithm.fit(X)
    t1 = time.time()
    if hasattr(algorithm, 'labels_'):
        y_pred = algorithm.labels_.astype(np.int)
    else:
        y_pred = algorithm.predict(X)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    if i_dataset == 0:
        plt.title(name, size=15)
    ax.scatter(X[:, 0], X[:, 1], X[:, 2],
               color=colors[y_pred].tolist(), s=10)

    if (hasattr(algorithm, 'cluster_centers_')):
```

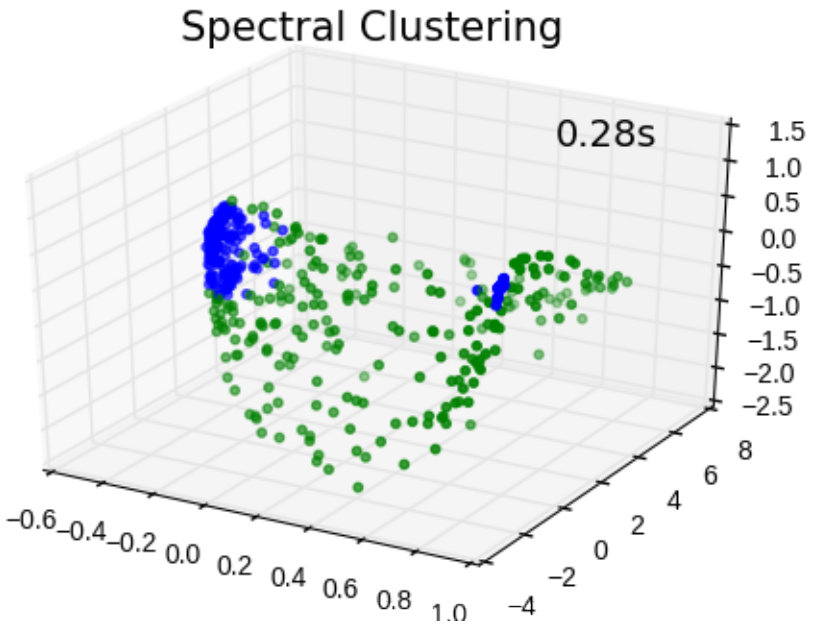
```
                centers = algorithm.cluster_centers_
                center_colors = colors[:len(centers)]
                ax.scatter(centers[:, 0], centers[:, 1],
                           centers[:, 2],
                           s=100, c=center_colors)
            ax.text(0.82, 2, 2 , ('%.2fs' % (t1 - t0)), size=14)
            plt.show()
```

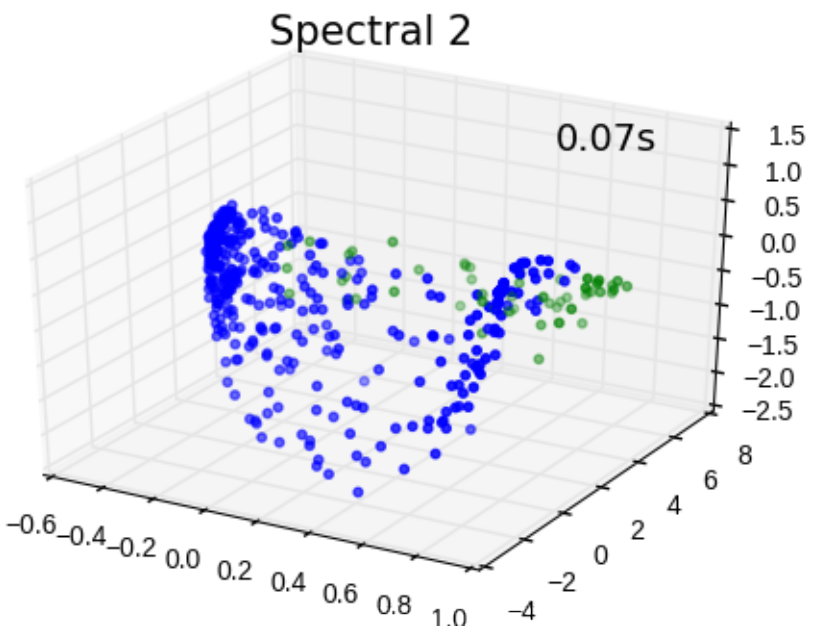
```
<matplotlib.figure.Figure at 0x7f54fa327240>
```

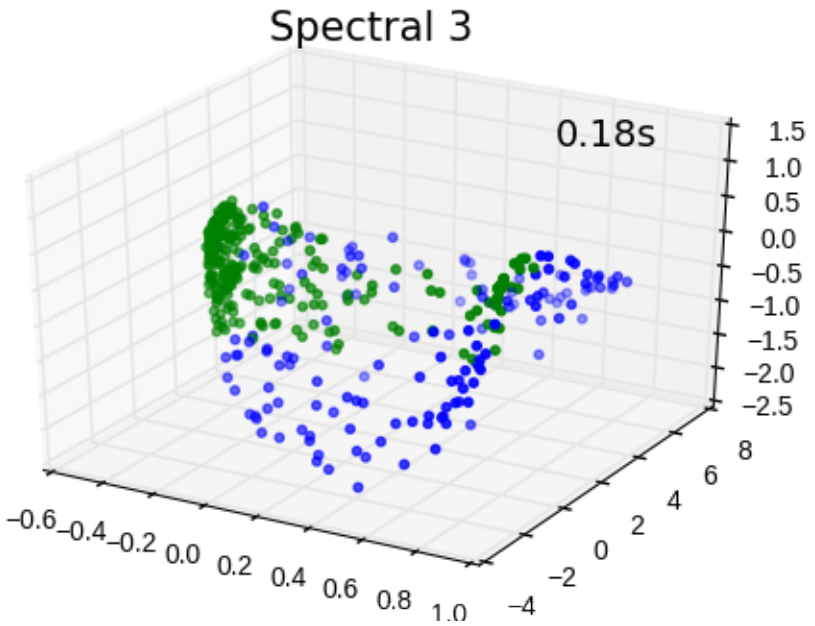


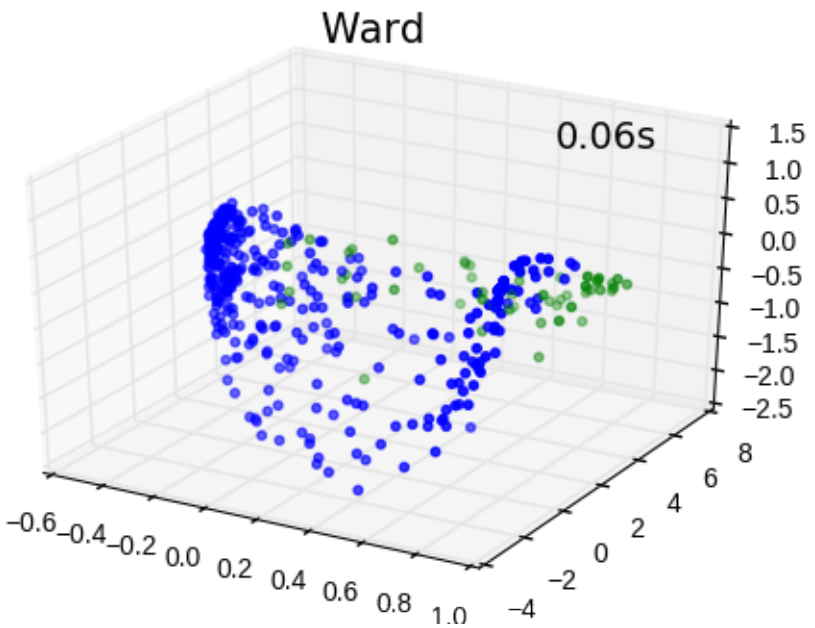


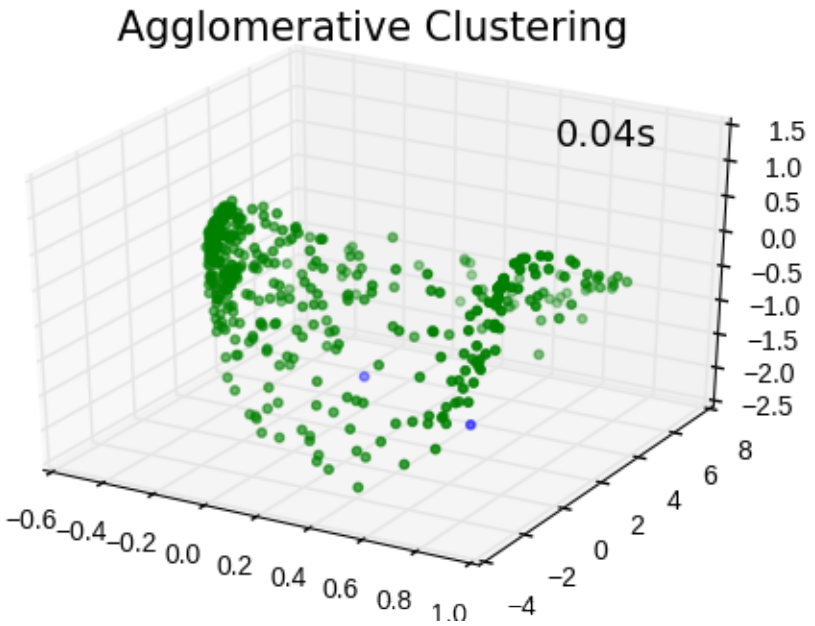

Mean Shift
0.04s
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
8
6
4
2
0
−2
−4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

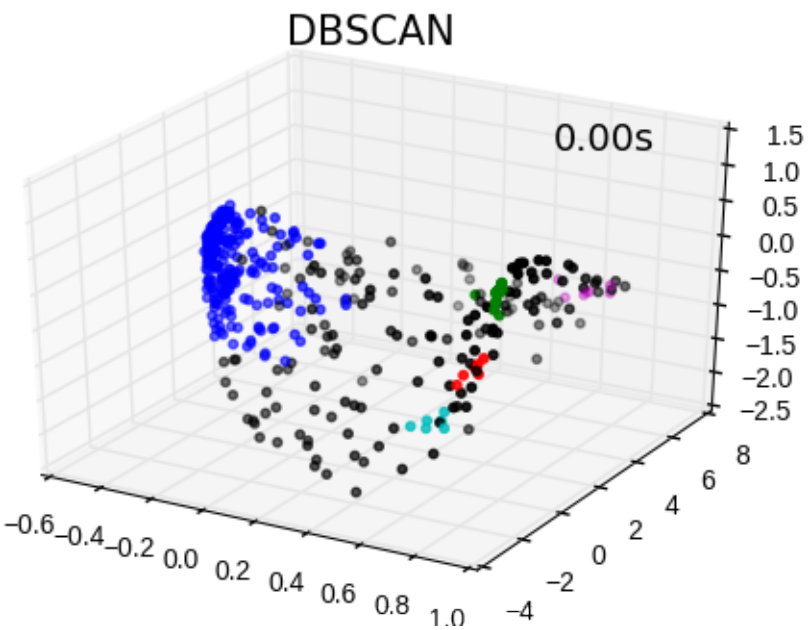

Spectral Clustering
0.28s
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
8
6
4
2
0
−2
−4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

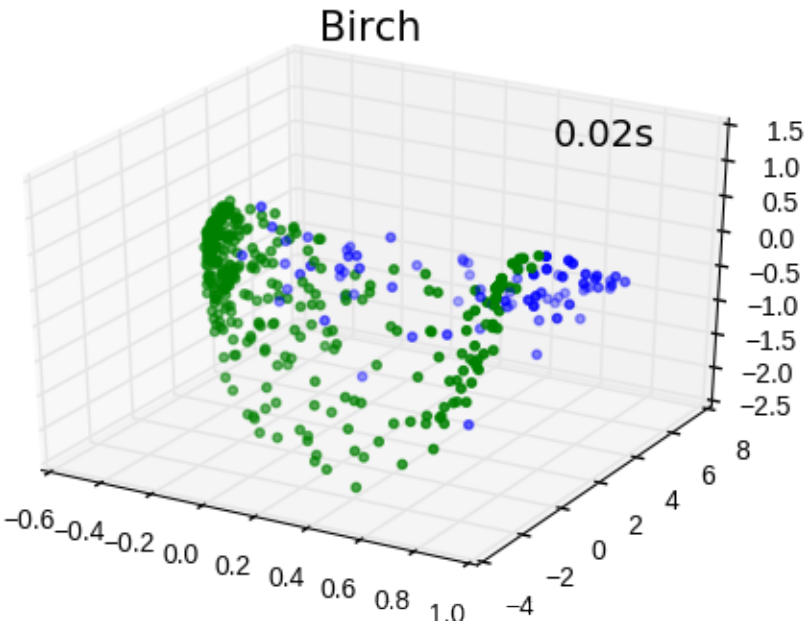

Spectral 2
0.07s
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
8
6
4
2
0
−2
−4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


Spectral 3
0.18s
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
8
6
4
2
0
−2
−4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


Ward
0.06s
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
8
6
4
2
0
−2
−4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


Agglomerative Clustering
0.04s
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
8
6
4
2
0
−2
−4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0





0.1.1 informacion de los datos

Utilizando PCA y una normalizacion robusta se puede observar que los usuarios se clasifican en 4 tipos. Esto no es tan claro cuando utilizamos una normalizacion estandar, debido a que dicha normalizacion puede verse sesgada por datos atipicos.