Exam Try 2

March 13, 2018

1 Table of Contents

```
In [1]: import pandas as pd
        import numpy as np
        import scipy
        import sklearn
        from sklearn import manifold
        from sklearn import decomposition

        # Graphics
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import seaborn as sns
        from matplotlib import rc

        rc('text', usetex=True)
        rc('text.latex', preamble=r'\usepackage{cmbright}')
        rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

        # Magic function to make matplotlib inline;
        %matplotlib inline

        # This enables SVG graphics inline.
        # There is a bug, so uncomment if it works.
        %config InlineBackend.figure_formats = {'png', 'retina'}

        # JB's favorite Seaborn settings for notebooks
        rc = {'lines.linewidth': 2,
              'axes.labelsize': 18,
              'axes.titlesize': 18,
              'axes.facecolor': 'DFDFE5'}
        sns.set_context('notebook', rc=rc)
        sns.set_style("dark")

        mpl.rcParams['xtick.labelsize'] = 16
        mpl.rcParams['ytick.labelsize'] = 16
        mpl.rcParams['legend.fontsize'] = 14
```

2 Generate $P(j|i)$ and find the histograms of edges for some node i

I had initially started on problem 1, then found it to be full of over/underflow errors and tricky linear algebra, so now I opted for the bio track.

The function below implements the argument of the exponent, $|y_i - y_j|^2/\sigma_i$. Then I just need to raise that to the power of Euler. Since the 'node' is unspecified, I generated some data, then found the node and plotted its histogram.

```
In [2]: def sne(X):
            """
            # calculate the dotproduct between each sample:
            # calculate |x_j|^2 for each vector
            """
            sum_X = np.sum(np.square(X), 1)
            dotprod = -2 * np.dot(X, X.T)
            # calculate
            # |x_j|^2 - 2*|x_i||x_j|cosTheta = |x_j|^2 - 2*|x_i - x_j|^2
            # this is asymmetric
            Dprime = np.add(dotprod, sum_X)
            # symmetrize by completing the square:
            # |x_j|^2 - 2*|x_i - x_j|^2 + |x_i|
            D = np.add(Dprime.T, sum_X)

            # set D_ii = 0
            D = D.astype(np.float)
            D = np.maximum(D, 0)
            np.fill_diagonal(D, 0)
            return D

In [3]: def shannon(data, sigma=1.0):
            """Given data (squared differences of vectors), return the entropy and p_ij values
            # Compute P-row and corresponding perplexity
            arg = -data/(2*sigma**2)

            if (arg > 0).any():
                raise ValueError('At least one probability is negative')

            if (arg > 710).any():
                raise ValueError('overflow warning, sigma={0:.2g}'.format(sigma))
```

```
        P = np.exp(arg)
        sumP = P.sum(axis=0)

        # H = -Sum_j p_jilogp_ji
        # p_ji = P/sumP
        # log p_ji = log P - log sumP
        # H = Sum_j p_ji/sumP * (D_ji/2*sigma**2 + np.log(sumP))
        # H = Sum_j (p_ji*D_ji/2*sigma**2))/sumP + p_ji/sumP*np.log(sumP)
        # H = beta * Sum_j (p_ji*D_ji)/sumP + Sum_j p_ji/sumP *np.log(sumP)
        # Sum_j p_ji = Sum_j p(j|i) = 1
        # H = beta * meancondD + np.log(sumP)
        H = np.log(sumP) + (2*sigma**2) * np.sum(data * P) / sumP

        if np.abs(H) == np.inf:
            raise ValueError('Entropy is undefined')

        # normalize the p_ij
        P = P/sumP
        return H, P
```

```
In [4]: def pca(X=np.array([]), no_dims=50):
            """PCA on X."""
            n, d = X.shape
            # mean center along columns
            X = X - X.mean(axis=0)
            # use eigh for hermitian/symmetric matrices ;)
            L, W = np.linalg.eigh(np.dot(X.T, X))
            # transform the coordinates
            Y = np.dot(X, W[:, d-no_dims - 1:d - 1])
            return Y, L
```

```
In [5]: dims = 100
        g1 = np.random.normal(10, 2, dims)
        g2 = np.random.normal(9.99, 2, dims)
        g3 = np.random.normal(10, 3, dims)
        g4 = np.random.normal(10, 4, dims)

        mus = np.array([g1, g2, g3, g4])
        mus = np.abs(mus)
```

```
In [6]: # generate data points, complete with random errors:
        measurements = 500
        X = np.zeros((measurements, dims))
        cluster = np.zeros(measurements)

        for i in np.arange(measurements):
            choice = np.random.choice(len(mus), 1)[0]
            sigma = np.abs(np.random.normal(0, choice, dims))
            sample = np.random.normal(mus[choice], sigma)
            sample[sample < 0] = 0
            X[i, :] = sample
            cluster[i] = choice
```

```
In [7]: # normalize cells by counts
        X = X/X.sum(axis=1).reshape(X.shape[0], 1)*10**3
        print(X.max(), X.sum(axis=1).max())
```

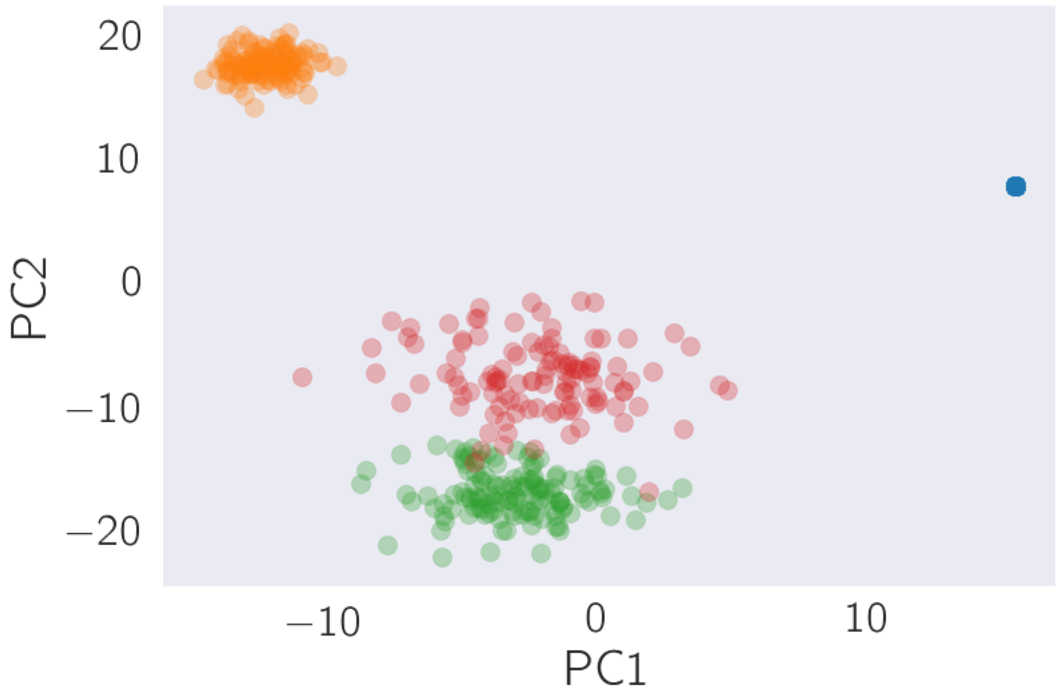
```
36.28033275967146 1000.0000000000005
```

```
In [8]: tX, l = pca(X)
```

```
In [9]: def plot_clusters(coords, jitter=0, **kwargs):
            for i in np.arange(4):
                x = coords[cluster == i, 0]
                y = coords[cluster == i, 1]
                if jitter:
                    x += np.random.normal(0, jitter, len(x))
                    y += np.random.normal(0, jitter, len(x))
                plt.scatter(x, y, label='Cluster {0}'.format(i), **kwargs)

        tX, l = pca(X, no_dims=2)
        plot_clusters(tX, alpha=0.3)
        plt.xlabel('PC1')
        plt.ylabel('PC2')
```

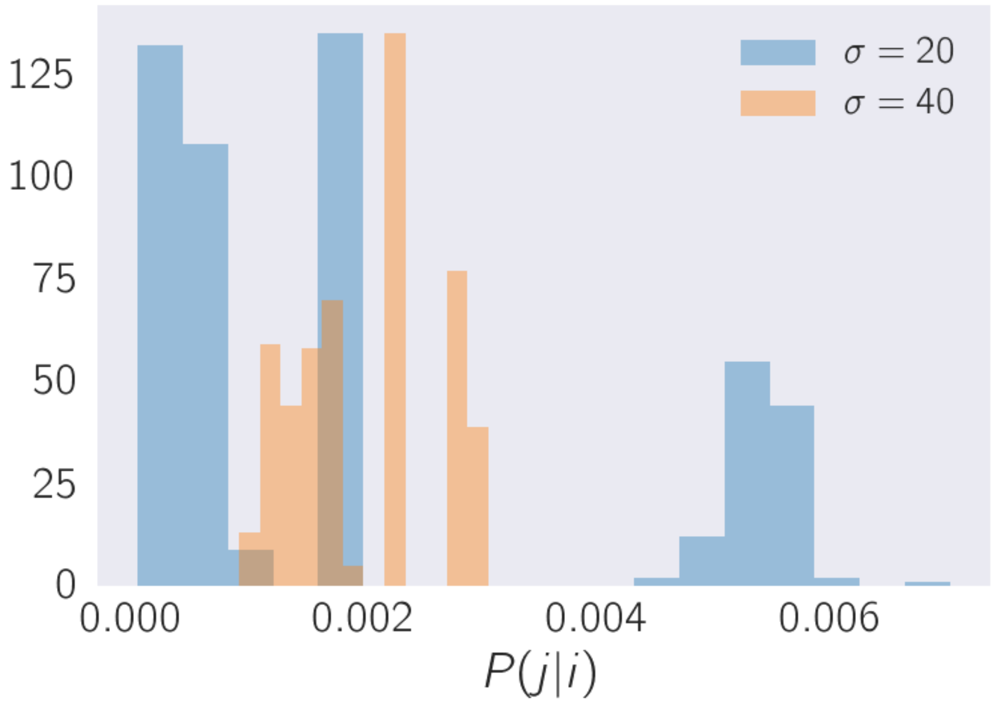
```
Out[9]: Text(0,0.5,'PC2')
```



```
In [10]: H, P20 = shannon(sne(X)[0,:], sigma=20)
         H, P40 = shannon(sne(X)[0,:], sigma=40)

In [11]: sns.distplot(P20, label=r'$\sigma=20$', kde=False)
         sns.distplot(P40, label=r'$\sigma=40$', kde=False)
         plt.legend()
         plt.xlabel(r'$P(j| i)$')

Out[11]: Text(0.5,0,'$P(j| i)$')
```



3 Part 2, data

First, I will load my data. Then I will define a few useful functions. Then I will show all the required plots at once.

```
In [12]: names = pd.read_csv('../input/gene_names_class.csv', header=None)

In [13]: X = pd.read_csv('../input/healthy1.csv', header=None).as_matrix()

In [14]: def do_pca(x):
             normX = (x - x.mean(axis=1)[:, np.newaxis])/x.std(axis=1)[:, np.newaxis]
```

```
            PCA = decomposition.PCA(n_components=10)
            redX = PCA.fit_transform(normX.T)
            return redX

        def do_nmf(k, x):
            k = 5
            nnmfX = x/x.std(axis=1)[:, np.newaxis]
            model = decomposition.NMF(n_components=k)
            W = model.fit_transform(nnmfX)
            H = model.components_
            return W, H

        def do_tsne(x, **kwargs):
            tsne = manifold.TSNE(n_components=2, **kwargs)
            Y = tsne.fit_transform(x)
            return Y

In [15]: def find_marker_genes(W):
            sig_genes = {}
            for i in np.arange(k):
                mu = W[:, i].mean()
                std = W[:, i].std()
                indices = np.where(W[:, i] > mu + std)
                index = np.where(W[:, i] == np.max(W[:, i]))[0][0]
                gene = names[names.index == index].values[0][0]
                sig_genes[i] = index
                print('{0} genes found for cluster {1}'.format(len(indices[0]), i))
                print('The gene with the best weight is {0}'.format(gene))
            return sig_genes

        def plot_and_compare(k, tsne_X, normx, h, sig_genes):
            fig, ax = plt.subplots(ncols=k, nrows=2, figsize=(12, 6))
            fig.suptitle('t-SNE plot colored by gene expression of marker genes', fontsize=16

            for i in np.arange(k):
                ax[0, i].scatter(tsne_X[:, 0], tsne_X[:, 1], c=normx[sig_genes[i]], alpha=0.3
            ax[0, 0].set_ylabel('NMF weight')

            for i in np.arange(k):
                ax[1, i].scatter(tsne_X[:, 0], tsne_X[:, 1], c=h.T[:, i], alpha=0.3, s=15)
            ax[1, 0].set_ylabel('gene marker')


            plt.tight_layout()
            fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

3.1 Plot the data for three different perplexity values and describe.

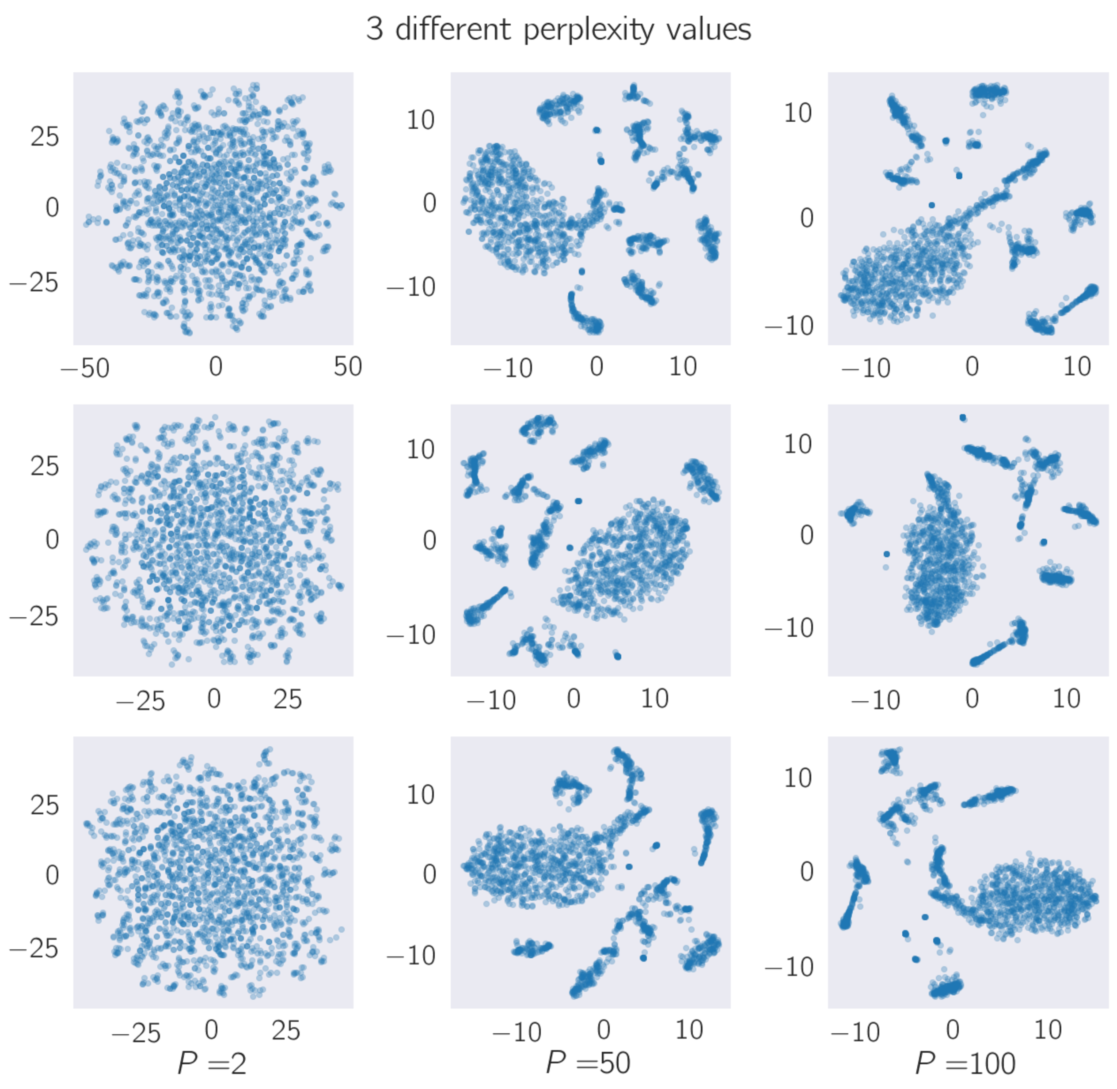
```
In [17]: redX = do_pca(X)

         fig, ax = plt.subplots(nrows=3, ncols=3, figsize=(9, 9))
         perplexities = [2, 50, 100]
         for i in np.arange(3):
             for j in np.arange(3):
                 perp = perplexities[i]
                 Y = do_tsne(redX, perplexity=perp)
                 ax[j, i].scatter(Y[:, 0], Y[:, 1], s=10, alpha=0.3)

         xlab = '$P=$' + '{0}'
         ax[2, 0].set_xlabel(xlab.format(perplexities[0]))
         ax[2, 1].set_xlabel(xlab.format(perplexities[1]))
         ax[2, 2].set_xlabel(xlab.format(perplexities[2]))

         fig.suptitle('3 different perplexity values', fontsize=20)

         plt.tight_layout()
         fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

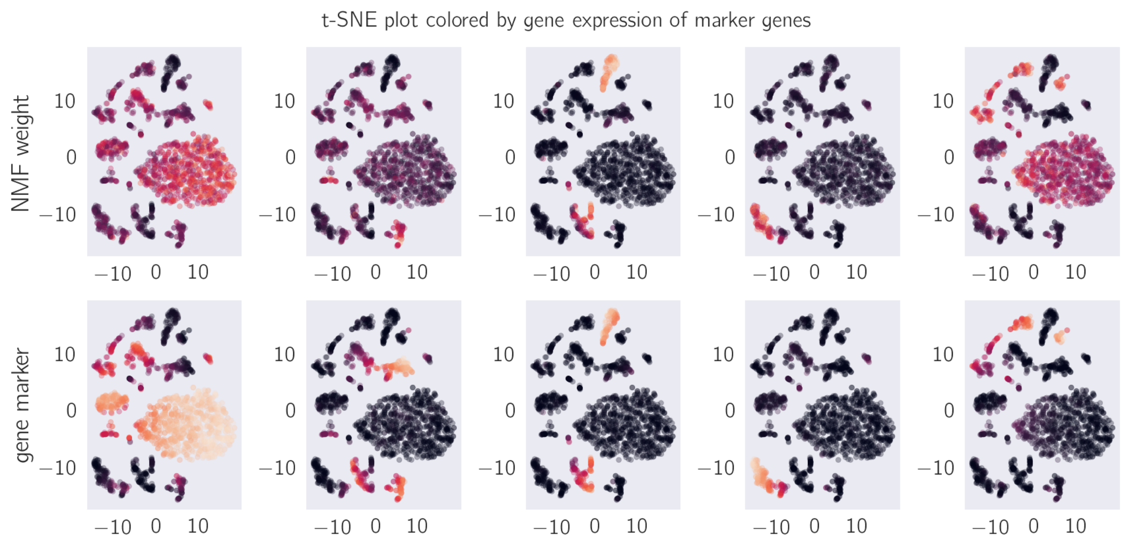


We can see that if the perplexity is too low, everything looks like a ball. For this dataset, the perplexity doesn't seem to matter very much between 50-100, since the diagrams look essentially identical to my eyes. In theory, setting the perplexity too high should make everything appear disjoint.

3.2 Perform NMF with a given K, plot a tSNE and color the data using the weightings. Then explore the weightings from NMF by finding the genes that contribute the most to each weighting.

```
In [18]: k = 5
         Y = do_tsne(redX)
         W, H = do_nmf(5, X)
         sig_genes = find_marker_genes(W)
         plot_and_compare(k, Y, X, H, sig_genes)
```

```
331 genes found for cluster 0
The gene with the best weight is RPS6
867 genes found for cluster 1
The gene with the best weight is PTMA
537 genes found for cluster 2
The gene with the best weight is HBB
565 genes found for cluster 3
The gene with the best weight is FTL
648 genes found for cluster 4
The gene with the best weight is B2M
```



Clearly, NMF and t-SNE both agree for this data. The clusters appear clear, though we ought to remember that distance means `nothing` in t-SNE space. Any separation at all between clusters means they might be totally distinct.

I found the genes that had the maximum weight within each cluster. Some of these genes are famous, and thus "make sense", which is a well known target of c-Myc, an immune protein...

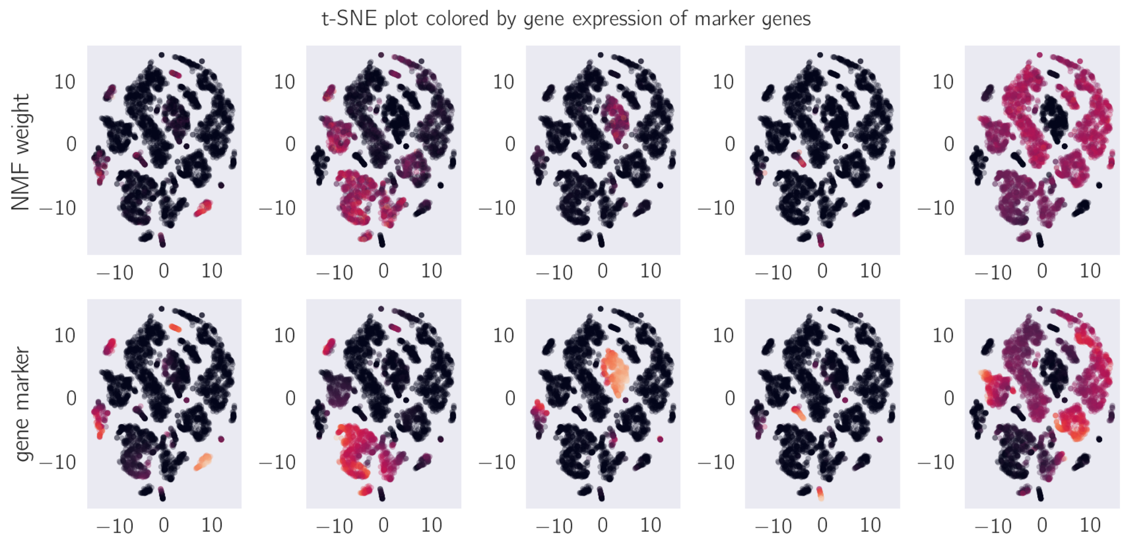
Funnily enough, the largest cluster is characterized by expression of RPS6! Yay! What does this mean? Who knows. Nobody understands the ribosome.

3.3 Take a look at the AML1 data

```
In [19]: unX = pd.read_csv('../input/aml1.csv', header=None).as_matrix()
         # clean up: there's some cells with nothing in here:
         unX = unX[np.where(unX.std(axis=1) != 0)[0], :]
```

```
In [20]: redUnX = do_pca(unX)
         unY = do_tsne(redUnX)
         W, unH = do_nmf(5, unX)
         sig_genes_un = find_marker_genes(W)
         plot_and_compare(k, unY, unX, unH, sig_genes_un)
```

```
848 genes found for cluster 0
The gene with the best weight is NPM1
762 genes found for cluster 1
The gene with the best weight is PYCARD
800 genes found for cluster 2
The gene with the best weight is BCAT2
741 genes found for cluster 3
The gene with the best weight is RPS6KB2
418 genes found for cluster 4
The gene with the best weight is NPRL3
```



Well, these beautiful images make no sense. Some clusters are observable, but the NMF colorings hardly make sense with these clusters. A major problem is that each color spans multiple clusters separated by uninterpretable space-does t-SNE think those are or are not clusters? Again, we get some "famous" genes that would make obvious sense to anyone who has studied them, and some ribosomal subunits that will make 99.9% of people shrug and go, 'yeah, we know that happens sometimes. When we get ribosomal subunits in genetic screens, we just kinda ignore them...'. Hey, maybe cancer cells have upregulated ribosomal subunits because they are actively growing! I'll make that my biological thought for the day.

```
In [ ]:
```