pca_gene_data

December 5, 2019

1 Data Cleaning

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [2]: df= pd.read_csv('../input/data_set_ALL_AML_train.csv')
        df.head()
        df1 = [col for col in df.columns if "call" not in col]
        df = df[df1]
        df = df.T
        df2 = df.drop(['Gene Description','Gene Accession Number'],axis=0)
        df2.index = pd.to_numeric(df2.index)
        df2.sort_index(inplace=True)
        df2['cat'] = list(pd.read_csv('../input/actual.csv')[:38]['cancer'])
        dic = {'ALL':0,'AML':1}
        df2.replace(dic,inplace=True)
```

```
In [3]: test = pd.read_csv('../input/data_set_ALL_AML_independent.csv')
        test1 = [col for col in test.columns if "call" not in col]
        test = test[test1]
        test = test.T
        test2 = test.drop(['Gene Description','Gene Accession Number'],axis=0)
        test2.index = pd.to_numeric(test2.index)
        test2.sort_index(inplace=True)
        test2['cat'] = list(pd.read_csv('../input/actual.csv')['cancer'][38:])
```

```
In [4]: df_combined = pd.concat([df2, test2])
        dic = {'ALL':0,'AML':1}
        df_combined.replace(dic,inplace=True)
        df_combined.cat.mean()
```

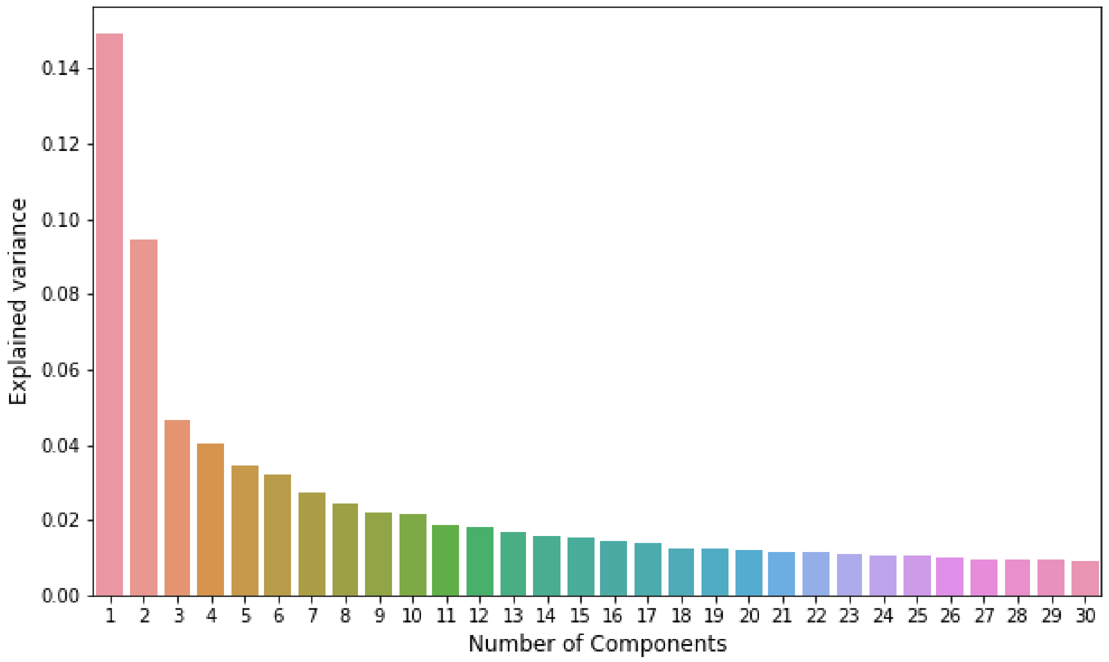
```
Out[4]: 0.3472222222222222
```

```
In [5]: from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA as sklearnPCA
        scalar = StandardScaler()
        scalar.fit(df_combined.drop('cat',axis=1))
        X_std = scalar.transform(df_combined.drop('cat',axis=1))


        sklearn_pca = sklearnPCA(n_components=30)
        sklearn_pca.fit(X_std)
        cat = df_combined.cat
        pc_x = sklearn_pca.transform(X_std)
```

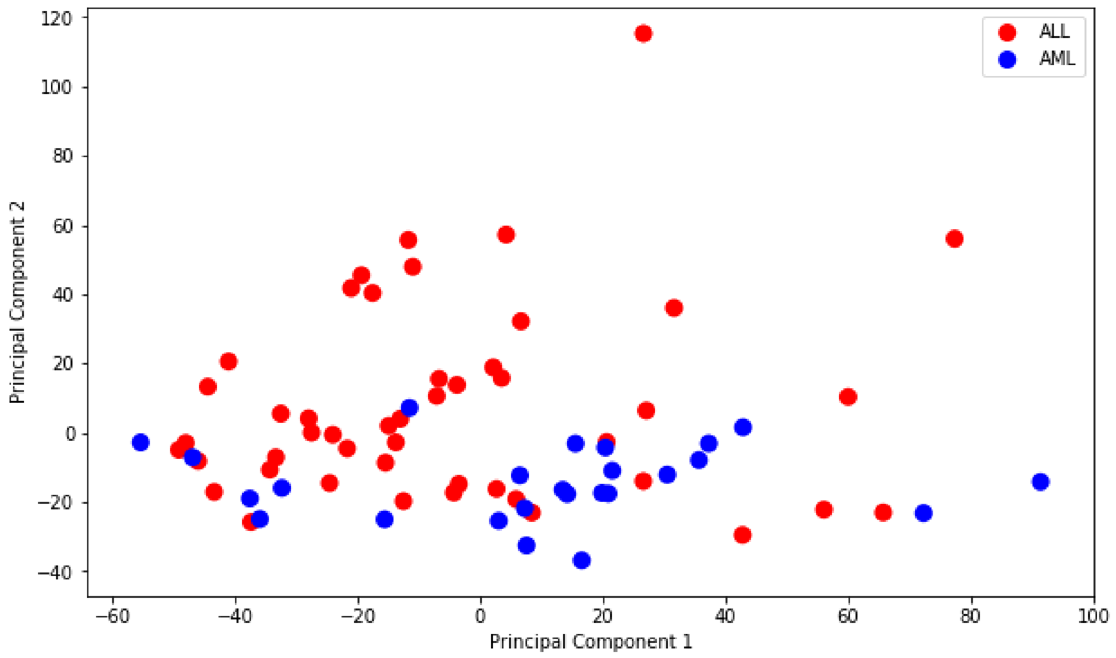
```
In [6]: import seaborn as sns
        plt.rcParams['figure.figsize'] = [10, 6]

        sns.barplot(np.arange(1, len(sklearn_pca.explained_variance_ratio_)+1),\
                    sklearn_pca.explained_variance_ratio_)
        plt.xlabel('Number of Components', size=12)
        plt.ylabel('Explained variance', size=12)
        plt.savefig('variance_bar.png');
```



```
In [7]: plt.rcParams['figure.figsize'] = [10, 6]
        plt.scatter(pc_x[cat==0, 0], pc_x[cat==0, 1],
                    c='red', edgecolor='none', s=100,label='ALL')
        plt.scatter(pc_x[cat==1, 0], pc_x[cat==1, 1],
                    c='blue', edgecolor='none', s= 100,label='AML')
```

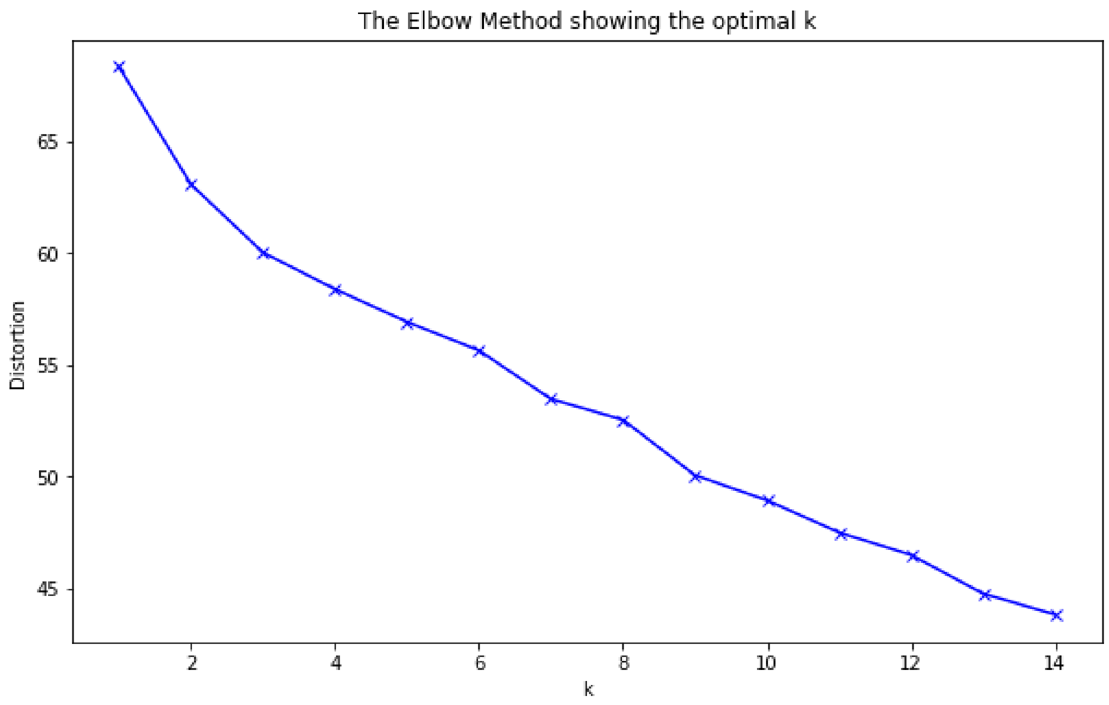
```
plt.xlabel('Principal Component 1', size=10)
plt.ylabel('Principal Component 2', size=10)
plt.legend()
plt.savefig('scatter.png')
```



```
In [8]: # k means determine k
        from scipy.spatial.distance import cdist
        from sklearn.cluster import KMeans

        distortions = []
        K = range(1,15)
        for k in K:
            kmeanModel = KMeans(n_clusters=k).fit(pc_x)
            kmeanModel.fit(pc_x)
            distortions.append(sum(np.min(cdist(pc_x, kmeanModel.cluster_centers_,\
                                                  'euclidean'), axis=1)) / pc_x.shape[0])

        # Plot the elbow
        plt.plot(K, distortions, 'bx-')
        plt.xlabel('k')
        plt.ylabel('Distortion')
        plt.title('The Elbow Method showing the optimal k')
        plt.show()
        plt.savefig('elbow.png')
```

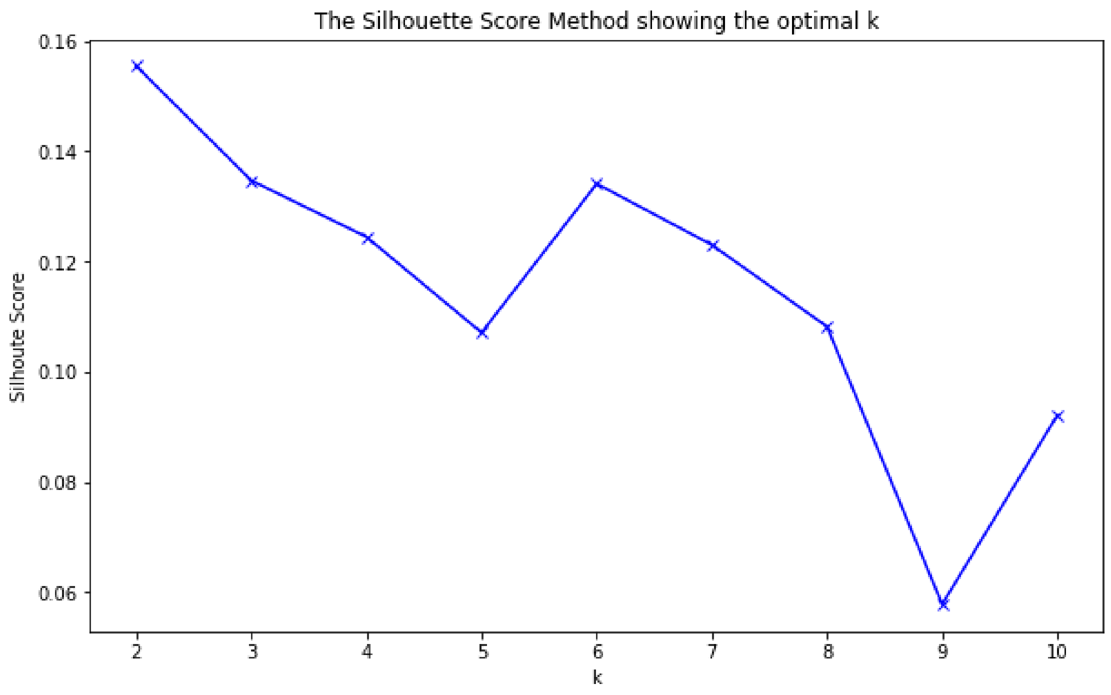


```
<Figure size 720x432 with 0 Axes>
```

```
In [9]: from sklearn.metrics import silhouette_score

        sil = []
        kmax = 10
        # dissimilarity would not be defined for a single cluster,\
        # thus, minimum number of clusters should be 2
        for k in range(2, kmax+1):
          kmeans_1 = KMeans(n_clusters = k).fit(pc_x)
          labels = kmeans_1.labels_
          sil.append(silhouette_score(pc_x, labels, metric = 'euclidean'))

        # Plot the elbow
        plt.plot(range(2, kmax+1), sil, 'bx-')
        plt.xlabel('k')
        plt.ylabel('Silhoute Score')
        plt.title('The Silhouette Score Method showing the optimal k')
        plt.show()
        plt.savefig('silhouette.png')
```



```
<Figure size 720x432 with 0 Axes>
```

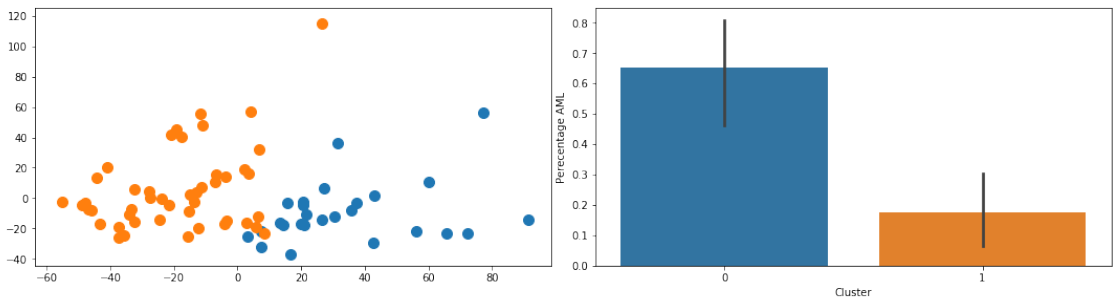
```
In [10]: from sklearn.cluster import KMeans

         kmeans = KMeans(n_clusters=2)
         kmeans.fit(pc_x)
         y_kmeans = kmeans.predict(pc_x)

         fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 4))
         fig.tight_layout()

         for i in np.unique(y_kmeans):
             axes[0].scatter(pc_x[y_kmeans == i, 0], pc_x[y_kmeans == i, 1],\
                             s=100, label=str(i))
         plt.xlabel('Principal Component 1', size=10)
         plt.ylabel('Principal Component 2', size=10)
         #plt.legend(title='Cluster')

         axes[1] = sns.barplot(y_kmeans, cat)
         plt.ylabel('Perecentage AML', size=10)
         plt.xlabel('Cluster', size=10)
         plt.savefig('cluster_bar.png')
         plt.savefig('kmeans_2.png')
```



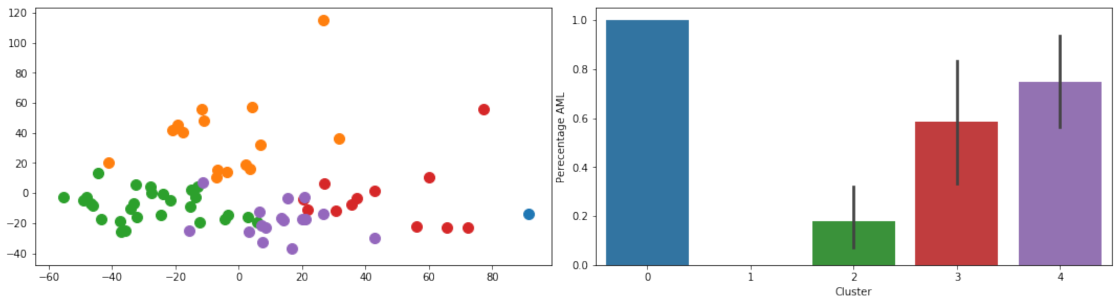
```
In [11]: kmeans = KMeans(n_clusters=5)
         kmeans.fit(pc_x)
         y_kmeans = kmeans.predict(pc_x)

         fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 4))
         fig.tight_layout()

         for i in np.unique(y_kmeans):
             axes[0].scatter(pc_x[y_kmeans == i, 0],\
                             pc_x[y_kmeans == i, 1], s=100, label=str(i))
         plt.xlabel('Principal Component 1', size=10)
         plt.ylabel('Principal Component 2', size=10)
         #plt.legend(title='Cluster')

         axes[1] = sns.barplot(y_kmeans, cat)
         plt.ylabel('Perecentage AML', size=10)
         plt.xlabel('Cluster', size=10)
         plt.savefig('cluster_bar.png')


         plt.savefig('kmeans_5.png')
```



```
In [12]: kmeans = KMeans(n_clusters=7)
         kmeans.fit(pc_x)
```

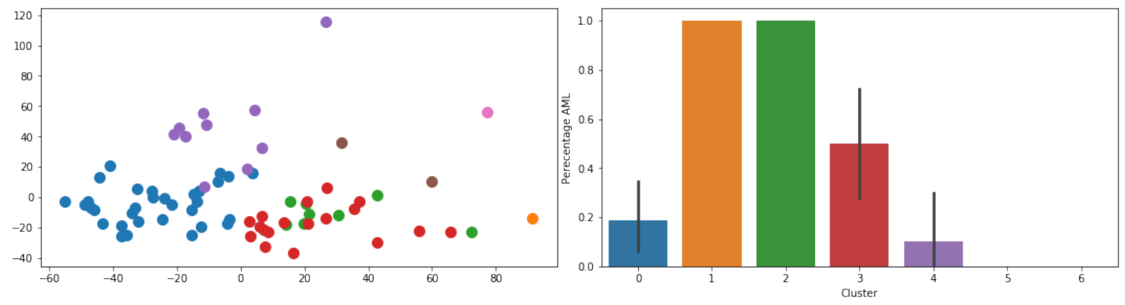
```
y_kmeans = kmeans.predict(pc_x)

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 4))
fig.tight_layout()

for i in np.unique(y_kmeans):
    axes[0].scatter(pc_x[y_kmeans == i, 0],\
                    pc_x[y_kmeans == i, 1], s=100, label=str(i))
plt.xlabel('Principal Component 1', size=10)
plt.ylabel('Principal Component 2', size=10)
#plt.legend(title='Cluster')

axes[1] = sns.barplot(y_kmeans, cat)
plt.ylabel('Perecentage AML', size=10)
plt.xlabel('Cluster', size=10)
plt.savefig('cluster_bar.png')


plt.savefig('kmeans_7.png')
```



```
In [13]: import numpy as np
         import pandas as pd
         import random

         def sample(data):
             sample = [random.choice(data) for _ in range(len(data))]
             return sample

         def bootstrap_t_test(treatment, control, nboot = 1000, direction = "less"):
             ones = np.vstack((np.ones(len(treatment)),treatment))
             treatment = ones.conj().transpose()
             zeros = np.vstack((np.zeros(len(control)), control))
             control = zeros.conj().transpose()
             Z = np.vstack((treatment, control))
             tstat = np.mean(treatment[:,1])-np.mean(control[:,1])
             tboot = np.zeros(nboot)
```

```
            for i in range(nboot):
                sboot = sample(Z)
                sboot = pd.DataFrame(np.array(sboot), columns=['treat', 'vals'])
                tboot[i] = np.mean(sboot['vals'][sboot['treat'] == 1])\
                - np.mean(sboot['vals'][sboot['treat'] == 0]) - tstat
            if direction == "greater":
                pvalue = np.sum(tboot>=tstat-0)/nboot
            elif direction == "less":
                pvalue = np.sum(tboot<=tstat-0)/nboot
            else:
                print('Enter a valid arg for direction')

            print('The p-value is %f' % (pvalue))
```

```
In [14]: kmeans = KMeans(n_clusters=2)
         kmeans.fit(pc_x)
         y_kmeans = kmeans.predict(pc_x)

         bootstrap_t_test(cat[y_kmeans == 0], cat[y_kmeans == 1],\
                          nboot = 1000, direction = "greater")
```

```
The p-value is 0.000000
```