pca

November 5, 2019

In [1]:
```
#============================================================
# Read me
#============================================================
# pca.py
# Author: Yesika Contreras
#
# This code calculates the PCA from a collections of dataframes.
# for now we have data from 5 datasets.
#
#
# python scripts generated 11-05-2019
# Modifications on 09-26-2019

import datetime
d = datetime. datetime. today()
print(d.strftime('%m-%d-%Y'))
```

```
11-05-2019
```

0.1 Principal Component Analysis (PCA)

In [2]:
```
#============================================================
# Packages
#============================================================

import pandas as pd
import numpy as np #operations
import os # Accesing to directory
import re # Regular Expressions
from six.moves import reduce # Merge dataframes
from scipy.cluster.hierarchy import dendrogram, linkage #for dendogram
from scipy.cluster.hierarchy import fcluster #cluster labels
from matplotlib import pyplot as plt # for visualizations
from sklearn.decomposition import PCA # PCA
import numpy # adding arrays
import matplotlib
```

```
import seaborn as sns
import matplotlib.style as style
```

In [3]:
```
#============================================================
# Reading Files
#============================================================

def read_files(df_file, skip_rows =0):
    '''
    Function that read the file as dataframe
    Extract the geneSymbol column and the Log Fold Change
    Input: User need to provide the path for file
    Output =  dataframes
    '''
    df_file = pd.read_csv(df_file, delimiter="\t", skiprows= skip_rows,)


    # Removing columns that are not GeneSymbol or logFC
    list_dropping =[]
    for column in df_file.columns:
        if not re.search('^GeneSymbol|^logFC', column):
            #print (column)
            list_dropping.append(column)

    df_file.drop(columns =  list_dropping,  inplace=True)


    # Maintain only one GeneSymbol column
    list_dropping =[]
    for column in df_file.columns:
        if re.search('^GeneSymbol', column):
            #print (column)
            list_dropping.append(column)

    df_file.drop(columns =  list_dropping[1:],  inplace=True)

    old_name = df_file.filter(regex='^GeneSymbol',axis=1).columns[0]
    df_file.rename(columns = { old_name: 'GeneSymbol'},
                   inplace =True)

    return df_file


def set_directory(path_1):
    '''
    set working directory:
    '''
    os.chdir(path_1)
```

```
        path_1 = os.getcwd()
        return path_1

In [7]: def pca_preprocessing(df_file):
        '''
        Operations before running the PCA algorithm
        includes assigning the attribute GeneSymbol as index
        Dropping all the missing values
        transposing the data, we are going to analyze by dataset
        '''
        # Setting index
        df_file= df_file.set_index('GeneSymbol')
        # dropping missing values if exist
        df_file.dropna(how="all", inplace=True)

        #  Transpose data and split data table into data X and class la = bels y
        df_file_transposed = df_file.transpose()
        df_file_transposed.reset_index(inplace =True)

        X = df_file_transposed.iloc[:,1:].values
        y = df_file_transposed.iloc[:,0].values

        # mapping the possible values, as all names are unique using the factorize functio
        df_file_transposed['index_map'] = pd.factorize(df_file_transposed['index'])[0]

        ## Standardizing
        # from sklearn.preprocessing import StandardScaler
        # X_std = StandardScaler().fit_transform(X)

        return (df_file_transposed,X)


    def scatter_plot_PCA(x,y,c,label, title):
        '''
        x= axis x values, series object
        y= axis y values, series object
        c = # c=setting the color based on the index, series object
        label= label for the data points, series object
        '''

        plt.style.use('ggplot')
        palette = plt.get_cmap('viridis_r')
        sns.set_context('talk')

        fig, ax = plt.subplots(figsize=(12,10))
        img = ax.scatter(x =x, y =y, c= c, alpha=0.8, s=100) # s= size of points
    #     plt.colorbar(img, ax=ax)
        plt.title(title)
```

```
            # adding point labels
            for i, txt in enumerate(label):
                ax.annotate(txt, (x[i], y[i]))

            plt.show()
            return img

        sns.reset_defaults()
        sns.reset_orig()
```

```
In [5]: #============================================================
        # User input:
        #============================================================

        file_path = '../output_files/5_datasets_input/df_merged_inner.txt'
        output_path = '../output_files/5_datasets_input'
```

```
In [6]: #============================================================
        # Computation Importing datasets
        #============================================================

        df_pca= read_files(file_path)
```

0.1.1 PCA

Principal Component Analysis (PCA) is a linear transformation technique used to identify patterns in data and it is useful when the variables within a dataset are highly correlated.

PCA allows us to summarize and visualize observations from a multi-dimensional hyperspace by reducing the dimensionality of the data to a lower-dimensional subspace with minimal loss of information. For instance, every attribute in a dataset could be considered as a different dimension, and PCA identify the multiple inter-correlated quantitative variables to transform them and express their content as a set of few new attributes called principal components. This components explaining most of the variance in the original variables, where the information in a given dataset corresponds to the total variation it contains.

Eigenvectors are the principal components and determine the directions of the new feature space, and each of those eigenvectors is associated with an eigenvalue which can be interpreted as the "length" or "magnitude" of the corresponding eigenvector, where the higher the value the more information captured about the distribution of the data(measure the amount of variation retained along the new feature axes).

A Summary of the PCA Approach PCA aims to reduce the dimensions of a -dimensional dataset by projecting it onto a ()-dimensional subspace (where 1 indicates that PCs account for more variance than accounted by one of the original variables in standardized data. This is commonly used as a cutoff point for which PCs are retained or another % decided. 4. Construct the projection matrix from the selected eigenvectors. 5. Transform the original dataset via to obtain a -dimensional feature subspace .

References: ploty kassambara

```
In [8]: df_pca, X = pca_preprocessing(df_pca)
        df_pca
```

Out[8]:

GeneSymbol	index	0610007P14Rik	\
0	logFC_GSM2386506_Kupper.txt	-0.089542	
1	logFC_GSE47045_GEO2R_TRM_v_TN_Carbone.txt	-1.062394	
2	logFC_GSE79858_Slansky_TIL_v_TN.txt	0.973000	
3	logFC_GSE_Anderson_TIL_v_Naive.txt	0.833661	
4	logFC_GSE_Goldrath_IELd35_v_TCMd35.txt	0.005556	

GeneSymbol	0610009B22Rik	0610009L18Rik	0610009O20Rik	0610010B08Rik	\
0	0.082806	0.155382	-0.258520	-0.374875	
1	0.423292	-0.023245	-0.355711	0.367777	
2	1.430000	-0.363000	0.463000	0.944867	
3	0.217723	0.254251	-1.758133	0.664039	
4	-0.100115	-0.010482	0.040761	-0.720720	

GeneSymbol	0610010F05Rik	0610010K14Rik	0610011F06Rik	0610012G03Rik	\
0	0.222593	-0.977021	-0.661878	-0.232291	
1	0.630598	-0.594890	0.209341	-0.198382	
2	1.250000	0.527000	-0.063300	-0.030000	
3	2.133460	1.774259	0.034102	0.767469	
4	-0.574580	-0.395259	-0.179260	-0.174267	

GeneSymbol	...	Zxda	Zxdc	Zyg11a	Zyg11b	Zyx	\
0	...	-0.288031	-0.352311	0.101246	0.044720	0.737536	
1	...	0.048923	0.013014	0.157655	0.296919	0.443751	
2	...	-0.527000	-0.203000	-0.200000	-0.630000	-0.287000	
3	...	1.133303	-0.631485	0.039053	-0.357012	0.976839	
4	...	0.100571	-0.065182	-0.102171	0.453181	-1.773563	

GeneSymbol	Zzef1	Zzz3	a	l7Rn6	index_map
0	-0.305835	-0.522968	0.229560	-0.865903	0
1	-0.426100	-0.766478	0.186984	-0.168868	1
2	-0.787000	-0.210000	0.016700	0.853000	2
3	-0.625164	-0.121589	-0.108131	0.563528	3
4	0.119355	0.035266	0.167257	-0.438411	4

[5 rows x 16063 columns]

0.1.2 Identifying the number of Principal Components

```
In [9]: my_model = PCA(n_components=5)
        X_pca = my_model.fit_transform(X)

In [10]: my_model = PCA().fit(X)

          # single value decomposition (SVD)  to compute eigenvectors
```

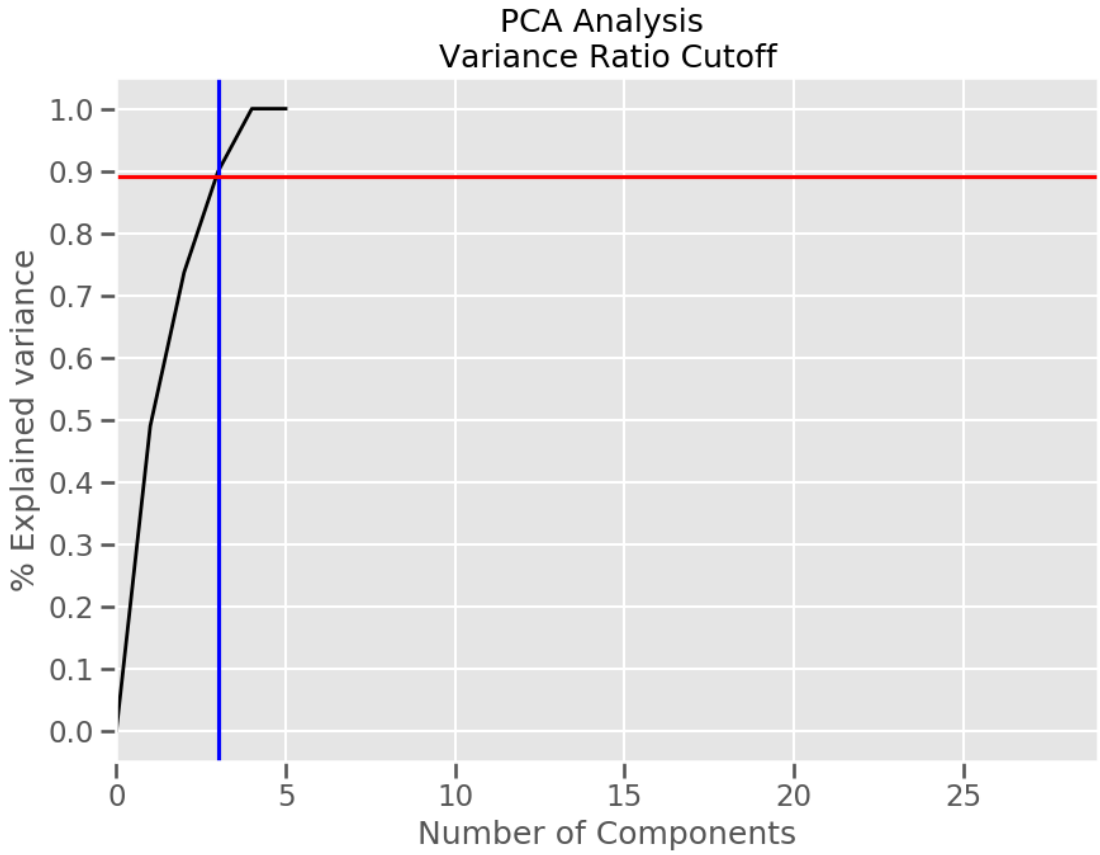
```
        print (my_model.explained_variance_) #returns a vector of the variance explained by e
        print (my_model.explained_variance_ratio_)  # returns variance explained solely by th
        print (my_model.explained_variance_ratio_.cumsum()) #returns the cumulative variance

        plt.style.use('ggplot') #seaborn-whitegrid
        palette = plt.get_cmap('viridis_r')
        sns.set_context('talk')

        plt.figure(figsize=(10,7))
        plt.xlim(0, 29)
        plt.yticks(np.arange(0, 1.1, 0.1))
        var= numpy.append([0],my_model.explained_variance_ratio_.cumsum()) # start from 0
        plt.plot(var, color='k', lw=2)
        plt.xlabel('Number of Components')
        plt.ylabel('% Explained variance')
        plt.title('PCA Analysis \n' +'Variance Ratio Cutoff')
        plt.axvline(3, c='b')
        plt.axhline(0.89, c='r')

        plt.show();
```

```
[2.93959917e+03 1.48470180e+03 9.72492852e+02 6.08358947e+02
 3.02000293e-28]
[4.89512804e-01 2.47237973e-01 1.61943066e-01 1.01306157e-01
 5.02901933e-32]
[0.4895128  0.73675078 0.89869384 1.         1.        ]
```

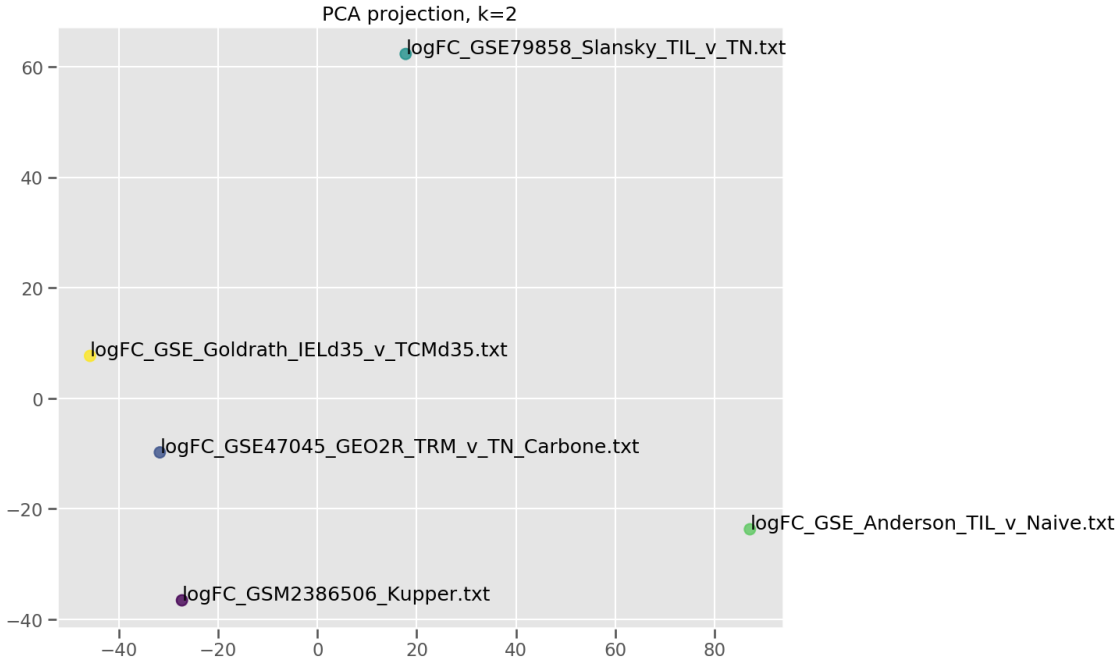



```
In [11]: # my_model_full = sklearnPCA(n_components='mle', svd_solver='full')
         # # Minka's maximum-likelihood estimation-will guess the number of
         # principal components.
         # n_components='mle' is only supported if n_samples >= n_features
         # Y_my_model_full = my_model_full.fit_transform(X)
```

0.1.3 PCA with 2 Components

```
In [12]: np.random.seed(0)
         my_model = PCA(n_components=2,random_state = 17)
         X_pca = my_model.fit_transform(X)

         ## plot
         scatter_plot_PCA(x=X_pca[:, 0],
                          y= X_pca[:, 1],
                          c= df_pca['index_map'],
                          label = df_pca['index'],
                          title= 'PCA projection, k=2')
```



Out[12]: <matplotlib.collections.PathCollection at 0x1a0e83cb38>

0.1.4 t-SNE method

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a is a non-linear dimensionality reduction technique used to represent high-dimensional dataset in a low-dimensional space of two or three dimensions. t-SNE creates a reduced feature space where similar samples are modeled by nearby points and dissimilar samples are modeled by distant points with high probability.

In the distribution, the points with the smallest distance with respect to the current point have a high likelihood, whereas the points far away from the current point have very low likelihoods.

A Summary of the t-SNE pproach

1. Calculate the probability of similarity of points in high-dimensional space and calculating the probability of similarity of points in the corresponding low-dimensional space. The similarity of points is calculated as the conditional probability that a point A would choose point B as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian (normal distribution) centered at A.
2. Minimize the difference between these conditional probabilities (or similarities) in higher-dimensional and lower-dimensional space for a perfect representation of data points in lower-dimensional space. To measure the minimization of the sum of difference of conditional probability t-SNE minimizes the sum of Kullback-Leibler divergence of overall data points using a gradient descent method.

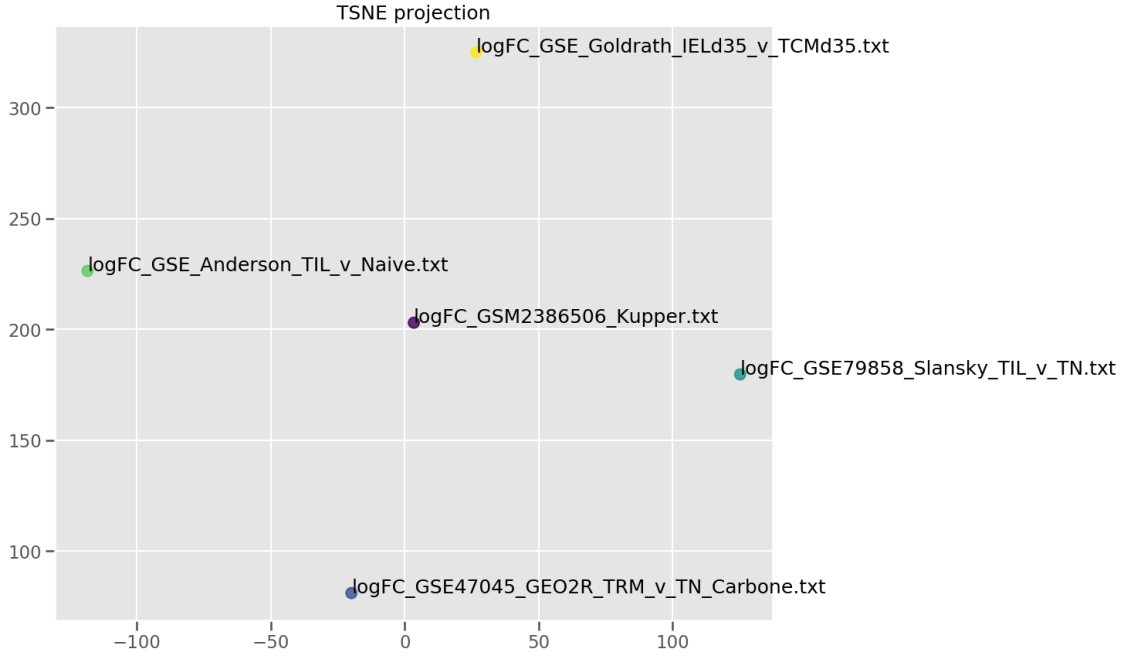
References: Maklin Violante datacamp

In [14]:
```
# Invoke the TSNE method
from sklearn.manifold import TSNE

tsne_model = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=2000,
                  random_state = 17)
X_tsne = tsne_model.fit_transform(X)

scatter_plot_PCA(x=X_tsne[:, 0],
             y= X_tsne[:, 1],
             c= df_pca['index_map'],
             label = df_pca['index'],
             title= 'TSNE projection')
```

```
[t-SNE] Computing 4 nearest neighbors...
[t-SNE] Indexed 5 samples in 0.000s...
[t-SNE] Computed neighbors for 5 samples in 0.001s...
[t-SNE] Computed conditional probabilities for sample 5 / 5
[t-SNE] Mean sigma: 1125899906842624.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration: 34.378033
[t-SNE] Error after 500 iterations: 0.123726
```

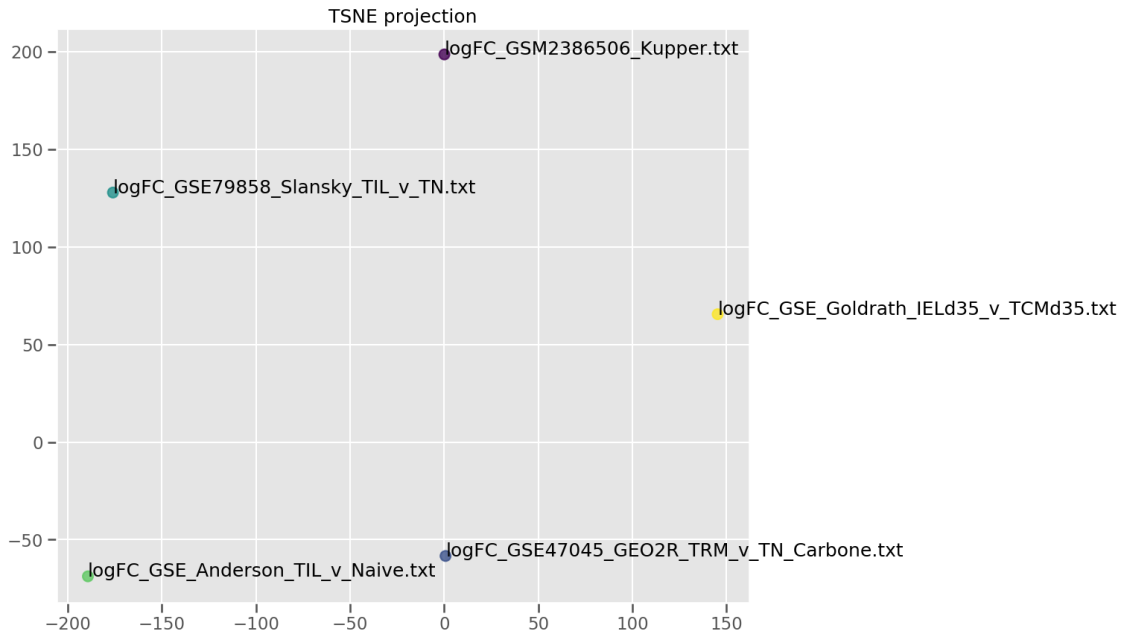


Out[14]: <matplotlib.collections.PathCollection at 0x10d482710>

In [ ]:
```
print(tsne_model)
```

In [15]:
```
tsne_model = TSNE(n_components=2, init='pca',random_state = 17)
X_tsne = tsne_model.fit_transform(X)

scatter_plot_PCA(x=X_tsne[:, 0],
                 y= X_tsne[:, 1],
                 c= df_pca['index_map'],
                 label = df_pca['index'],
                 title= 'TSNE projection')
```



Out[15]: <matplotlib.collections.PathCollection at 0x10d4e0518>

In [16]:
```
print(tsne_model)
```

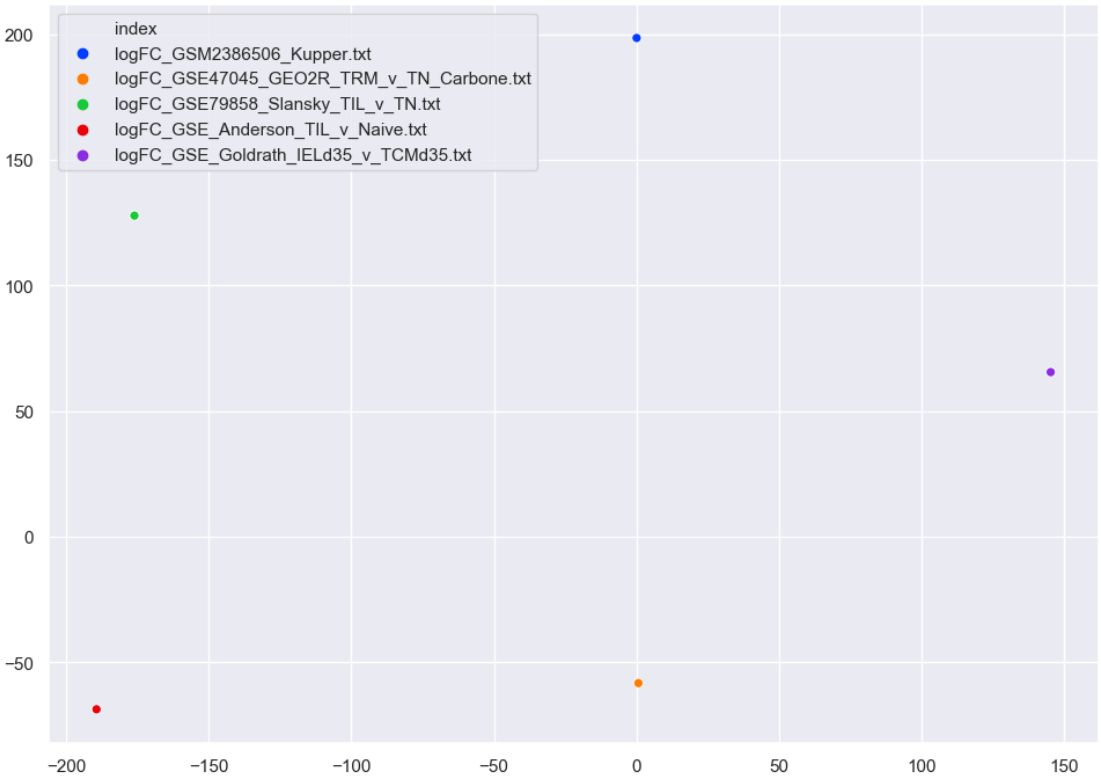
```
TSNE(angle=0.5, early_exaggeration=12.0, init='pca', learning_rate=200.0,
   method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
   n_components=2, n_iter=1000, n_iter_without_progress=300,
   perplexity=30.0, random_state=17, verbose=0)
```

In [20]:
```
## Alternative plot

import seaborn as sns
sns.set(rc={'figure.figsize':(11.7,8.27)})
palette = sns.color_palette("bright", 5)

sns.scatterplot(X_tsne[:,0], X_tsne[:,1], hue=df_pca['index'], legend='full', palette=
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x10d7a4978>


index
logFC_GSM2386506_Kupper.txt
logFC_GSE47045_GEO2R_TRM_v_TN_Carbone.txt
logFC_GSE79858_Slansky_TIL_v_TN.txt
logFC_GSE_Anderson_TIL_v_Naive.txt
logFC_GSE_Goldrath_IELd35_v_TCMd35.txt
200
150
100
50
0
−50
−200
−150
−100
−50
0
50
100
150