IRIS DATASET TESTS

In [1]:

```
import pandas as pd
import numpy as np
from sklearn import datasets
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D
from sklearn.manifold import TSNE
from sklearn.manifold import MDS
from sklearn.decomposition import PCA
from sklearn.neighbors import NearestCentroid
import pickle
import math

# addtional files
import lion_tsne
import input_data
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.tes
ting is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [0]:

```
PALETTE = sns.color_palette('deep', n_colors=3)
CMAP = ListedColormap(PALETTE.as_hex())
RANDOM_STATE = 42
```

In [3]:

```
data_iris = datasets.load_iris()
X_iris = data_iris.data
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
iris = pd.DataFrame(X_iris, columns=features)
iris['species'] = data_iris.target
iris.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [0]:

```
def plot_iris_2d(x, y, title, xlabel="1st eigenvector", ylabel="2nd eigenvector",colors=iris['species']
):
    plt.gcf().set_size_inches(10,10)
    sns.set_style("darkgrid")

    plt.scatter(x, y,
        c=colors,
        cmap=CMAP,
        s=70)

    plt.title(title, fontsize=20, y=1.03)
```

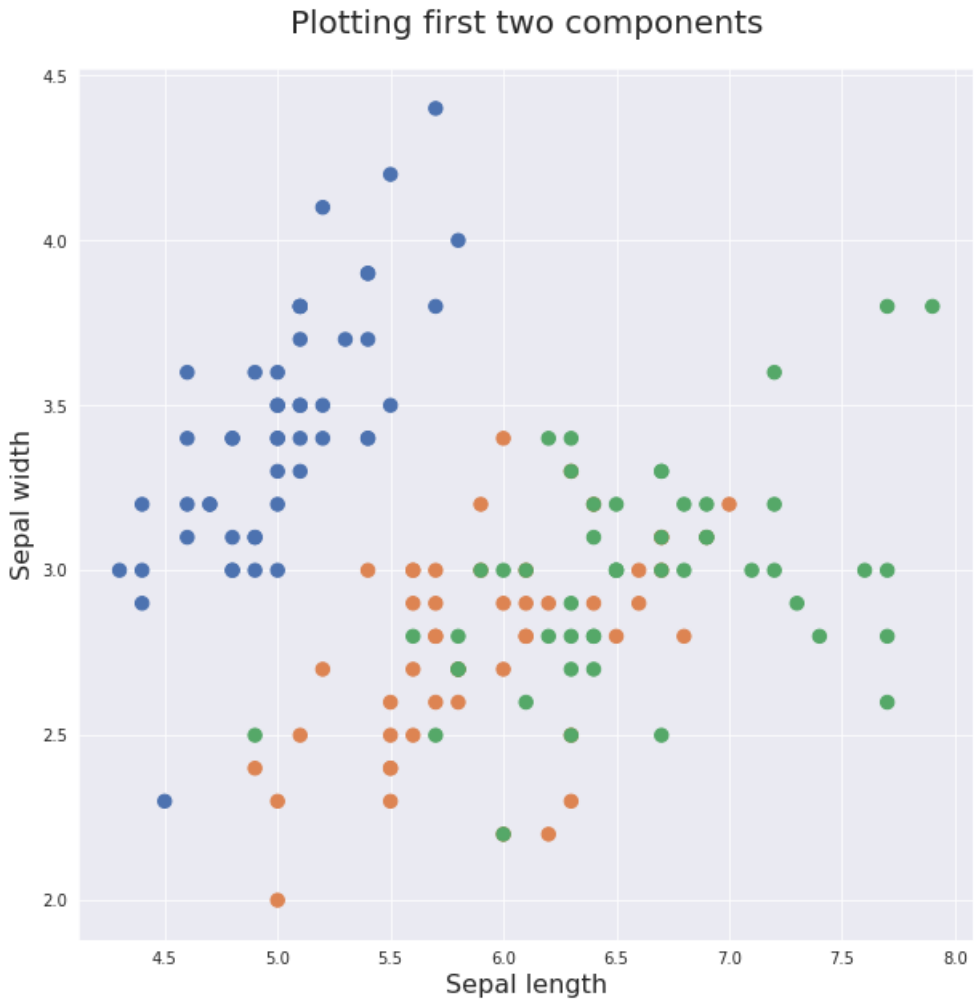
```
    plt.xlabel(xlabel, fontsize=16)
    plt.ylabel(ylabel, fontsize=16)
```

In [0]:

```
def plot_iris_3d(x, y, z, title):
    sns.set_style('whitegrid')

    fig = plt.figure(1, figsize=(8, 6))
    ax = Axes3D(fig, elev=-150, azim=110)

    ax.scatter(x, y, z,
               c=iris['species'],
               cmap=CMAP,
               s=40)

    ax.set_title(title, fontsize=20, y=1.03)

    fsize = 14
    ax.set_xlabel("1st eigenvector", fontsize=fsize)
    ax.set_ylabel("2nd eigenvector", fontsize=fsize)
    ax.set_zlabel("3rd eigenvector", fontsize=fsize)

    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
```

In [166]:

```
plot_iris_2d(
    x = iris['sepal_length'],
    y = iris['sepal_width'],
    title = 'Plotting first two components',
    xlabel = 'Sepal length',
    ylabel = 'Sepal width')
```



Sampling

radnom sampling

In [153]:

```
np.random.seed(0)
ind = np.random.choice(np.arange(data_iris.data.shape[0]), size = 120)
X_iris_random = data_iris.data[ind]
y_iris_random = data_iris.target[ind]

iris_random_df = pd.DataFrame(X_iris_random, columns=features)
iris_random_df['target'] = y_iris_random
iris_random_df['target'].value_counts()
```

Out[153]:

```
2    42
0    40
1    38
Name: target, dtype: int64
```

knn sampling - dummy version

In [0]:

```
def euclidean_distance(point1, point2):
  distance = 0.0
  for i in range(len(point1)-1):
    distance += (point1[i] - point2[i])**2
  return math.sqrt(distance)

def get_neighbors(train, test_row, num_neighbors, class_type):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
  distances.sort(key=lambda tup: tup[1])
  result_df = pd.DataFrame(columns=features)
  for i in range(num_neighbors):
    tmp_df = pd.DataFrame([distances[i][0]], columns=features)
    result_df = result_df.append(tmp_df, ignore_index=True)

  result_df['target'] = class_type
  return result_df
```

In [155]:

```
clf = NearestCentroid()
clf.fit(data_iris.data, data_iris.target)
clf.centroids_
```

Out[155]:

```
array([[5.006, 3.428, 1.462, 0.246],
       [5.936, 2.77 , 4.26 , 1.326],
       [6.588, 2.974, 5.552, 2.026]])
```

In [156]:

```
iris_target_values = np.unique(data_iris.target)
iris_knn_df = pd.DataFrame()

for i in iris_target_values:
  iris_knn_df = iris_knn_df.append(get_neighbors(data_iris.data, clf.centroids_[i], 40, str(i)), ignore
_index=True)
```

```
iris_knn_df['target'].value_counts()
```

Out[156]:

```
1    40
2    40
0    40
Name: target, dtype: int64
```

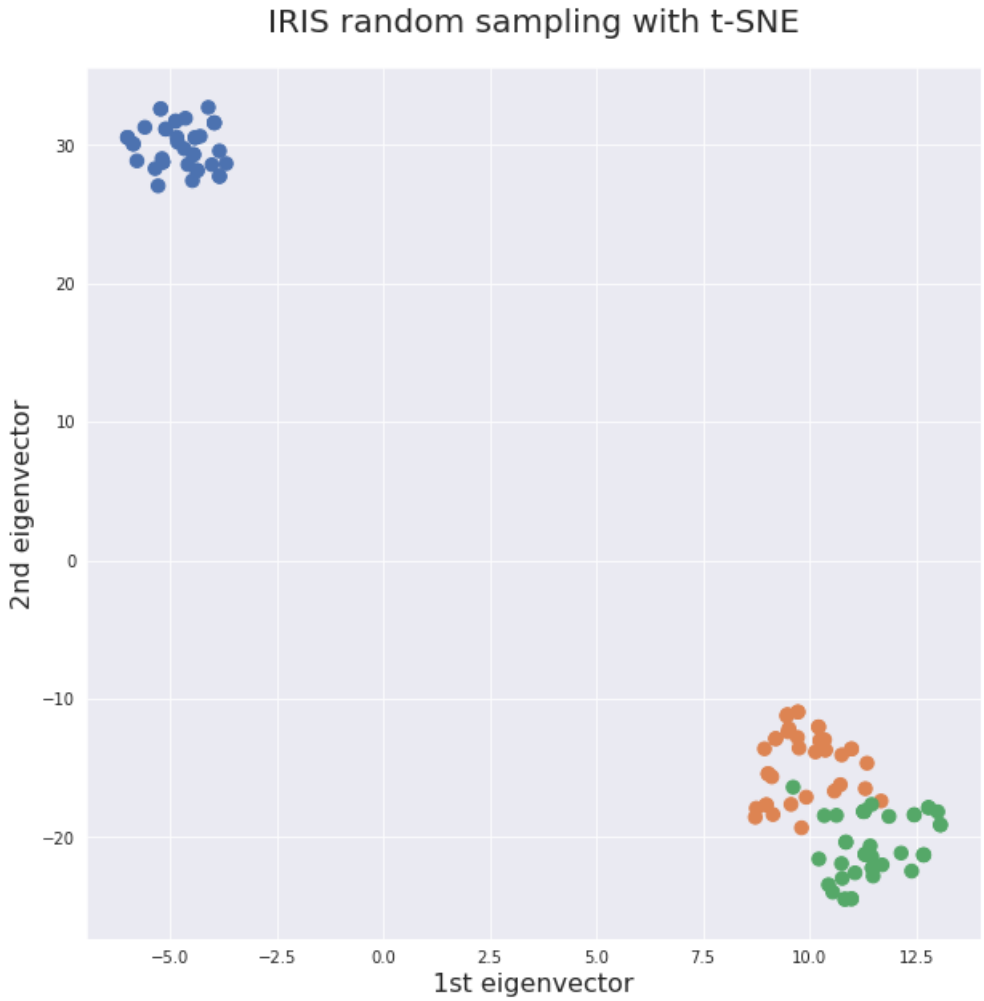
tSNE

In [0]:

```
tsne = TSNE(n_components=2, n_iter=3000, random_state=RANDOM_STATE)
layers = tsne.fit_transform(iris_random_df.loc[:, features].values)
```

In [167]:

```
plot_iris_2d(
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS random sampling with t-SNE',
    colors=iris_random_df.loc[:, ['target']].values)
```



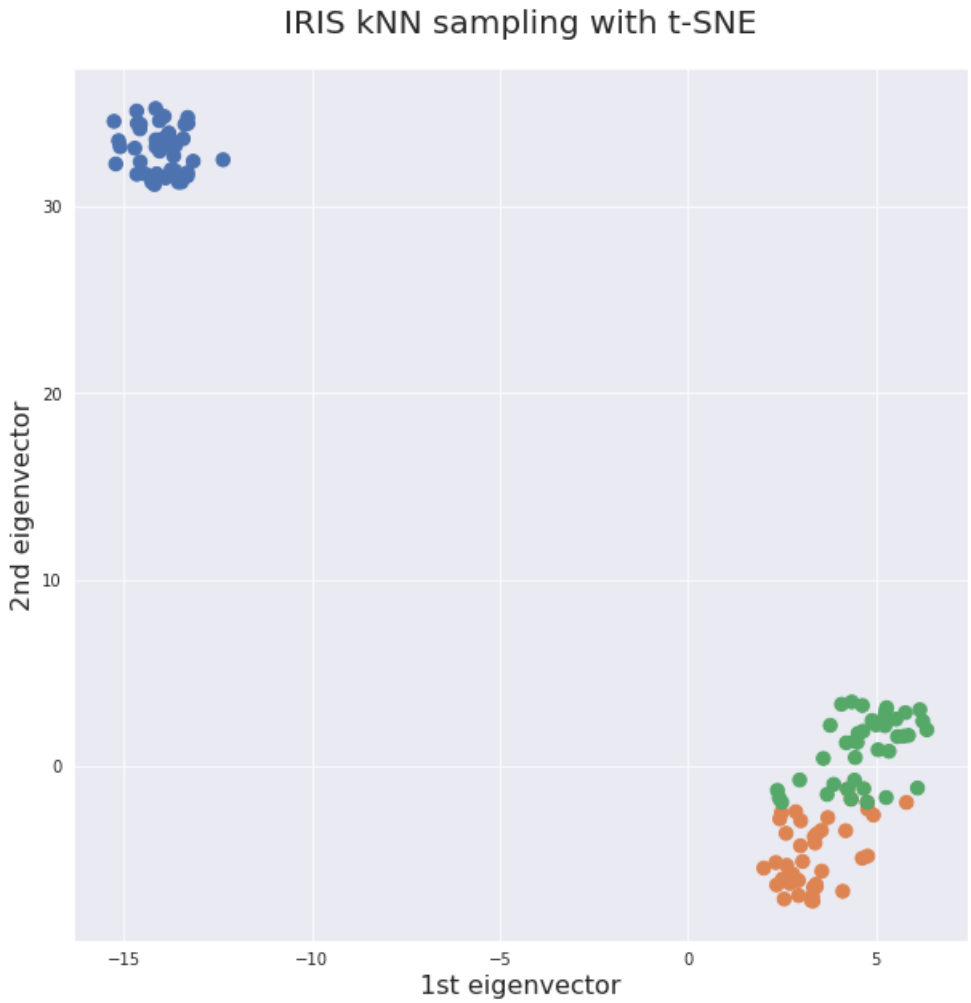
In [0]:

```
tsne = TSNE(n_components=2, n_iter=3000, random_state=RANDOM_STATE)
layers = tsne.fit_transform(iris_knn_df.loc[:, features].values)
```

In [169]:

```
plot_iris_2d(
```

```
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS kNN sampling with t-SNE',
    colors=iris_knn_df.loc[:, ['target']].values)
```



LION tSNE

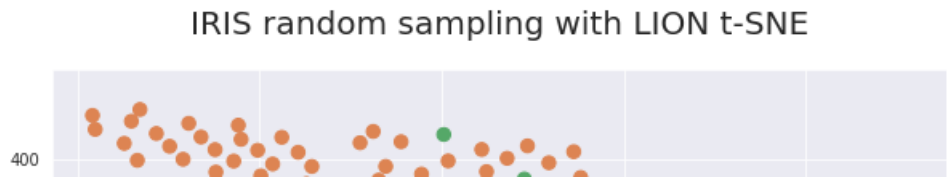
random sampling

In [0]:

```
%%capture
lionTSNE_iris = lion_tsne.LionTSNE(perplexity=30)
layers = lionTSNE_iris.fit(iris_random_df.loc[:, features].values, optimizer_kwargs={'momentum': 0.8, '
n_iter': 3000,
                                                    'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [176]:

```
plot_iris_2d(
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS random sampling with LION t-SNE',
    colors=iris_random_df.loc[:, ['target']].values)
```



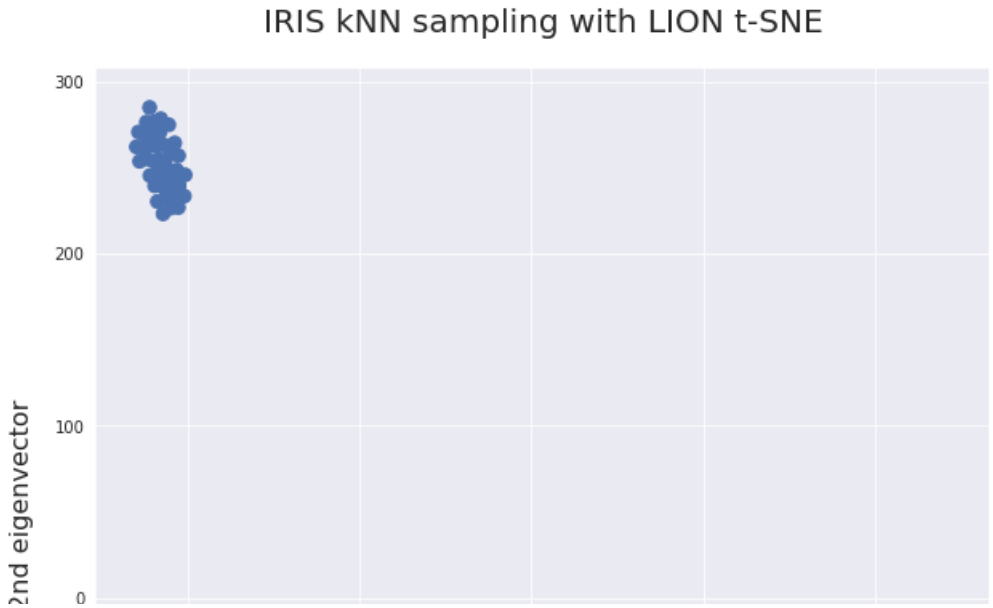
kNN sampling

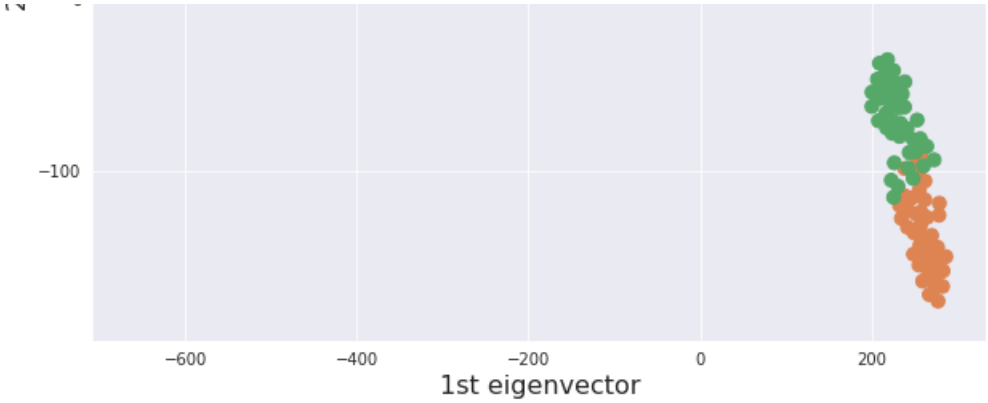
In [0]:

```
%%capture
lionTSNE_iris = lion_tsne.LionTSNE(perplexity=30)
layers = lionTSNE_iris.fit(iris_knn_df.loc[:, features].values, optimizer_kwargs={'momentum': 0.8, 'n_iter': 3000,
                                                                                  'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [178]:

```
plot_iris_2d(
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS kNN sampling with LION t-SNE',
    colors=iris_knn_df.loc[:, ['target']].values)
```





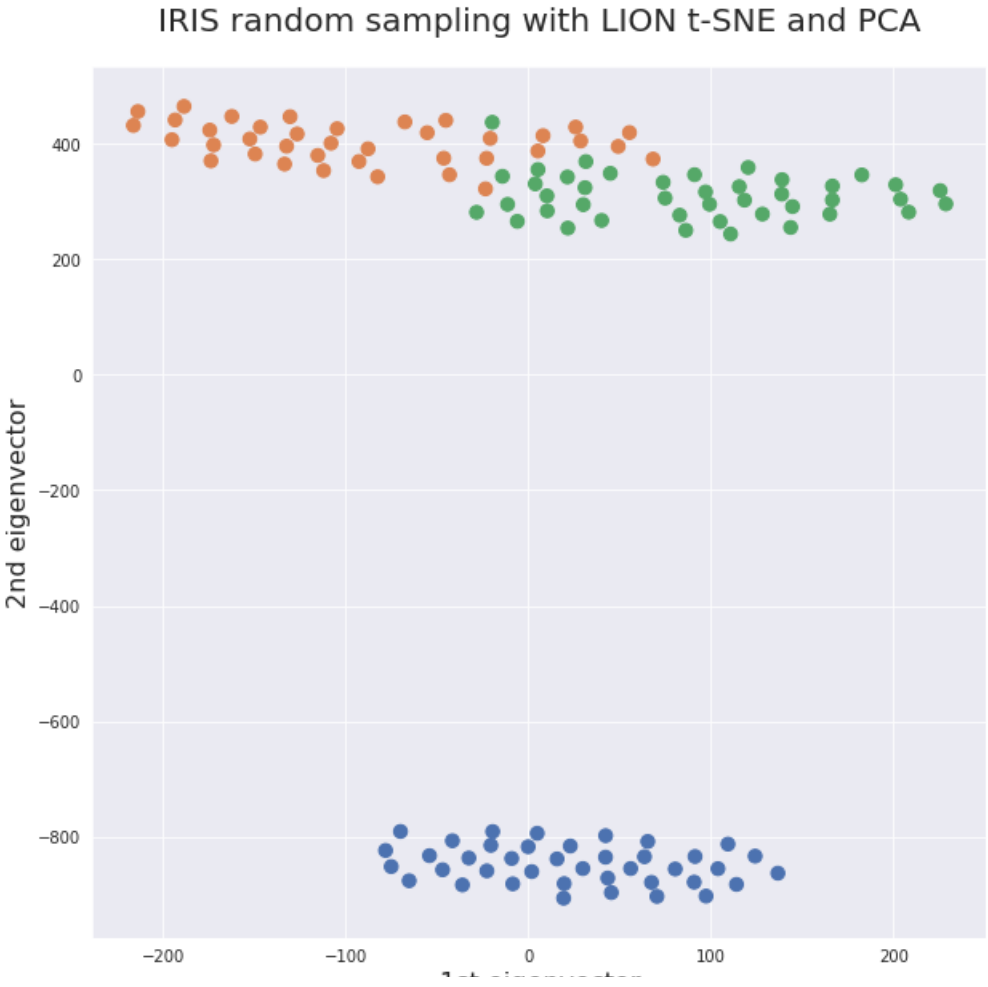
random sampling with PCA

In [0]:

```
%%capture
iris_pca = PCA(n_components=4)
X_iris_pca = iris_pca.fit_transform(iris_random_df.loc[:, features].values)

lionTSNE_iris = lion_tsne.LionTSNE(perplexity=30)
layers = lionTSNE_iris.fit(X_iris_pca, optimizer_kwargs={'momentum': 0.8, 'n_iter': 3000,
                                                         'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [180]:

```
plot_iris_2d(
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS random sampling with LION t-SNE and PCA',
    colors=iris_random_df.loc[:, ['target']].values)
```



1st eigenvector

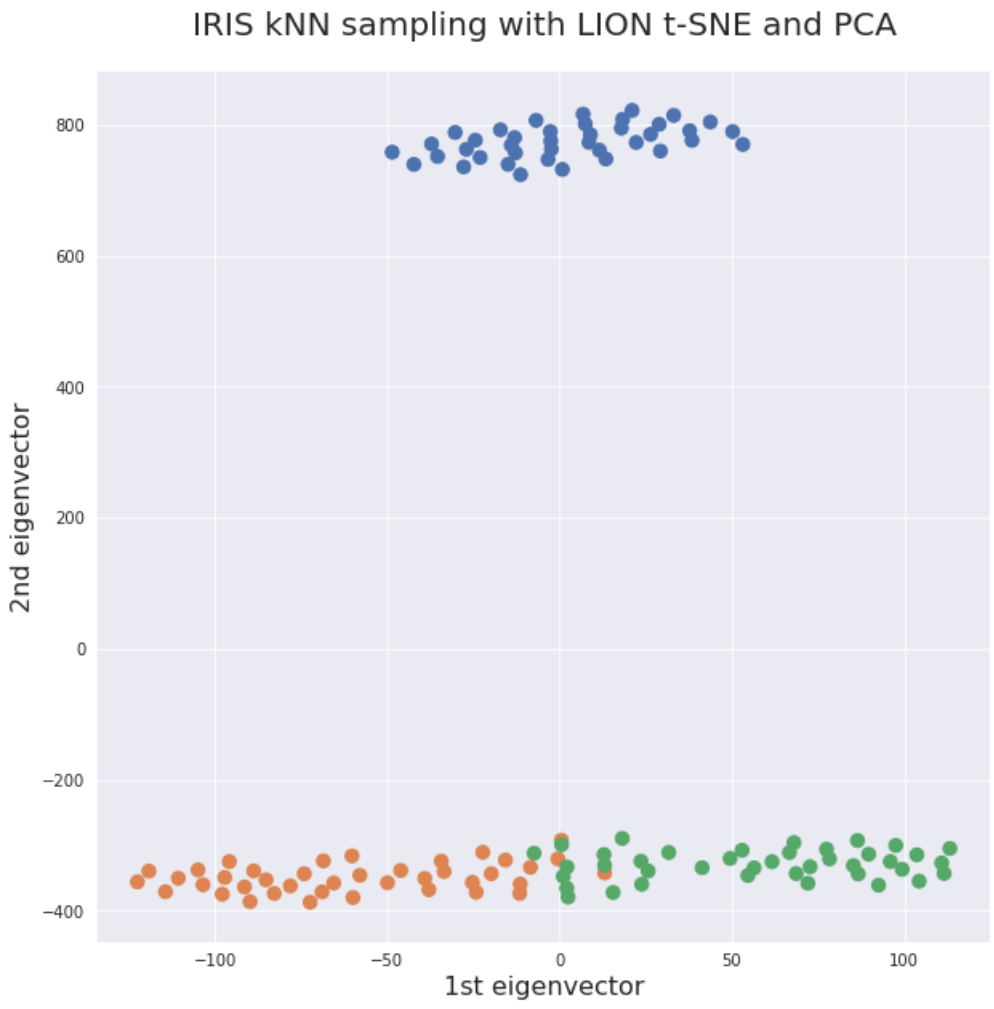
kNN sampling with PCA

In [0]:

```
%%capture
iris_pca = PCA(n_components=4)
X_iris_pca = iris_pca.fit_transform(iris_knn_df.loc[:, features].values)

lionTSNE_iris = lion_tsne.LionTSNE(perplexity=30)
layers = lionTSNE_iris.fit(X_iris_pca, optimizer_kwargs={'momentum': 0.8, 'n_iter': 3000,
                                                         'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [182]:

```
plot_iris_2d(
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS kNN sampling with LION t-SNE and PCA',
    colors=iris_knn_df.loc[:, ['target']].values)
```



random sampling with MDS

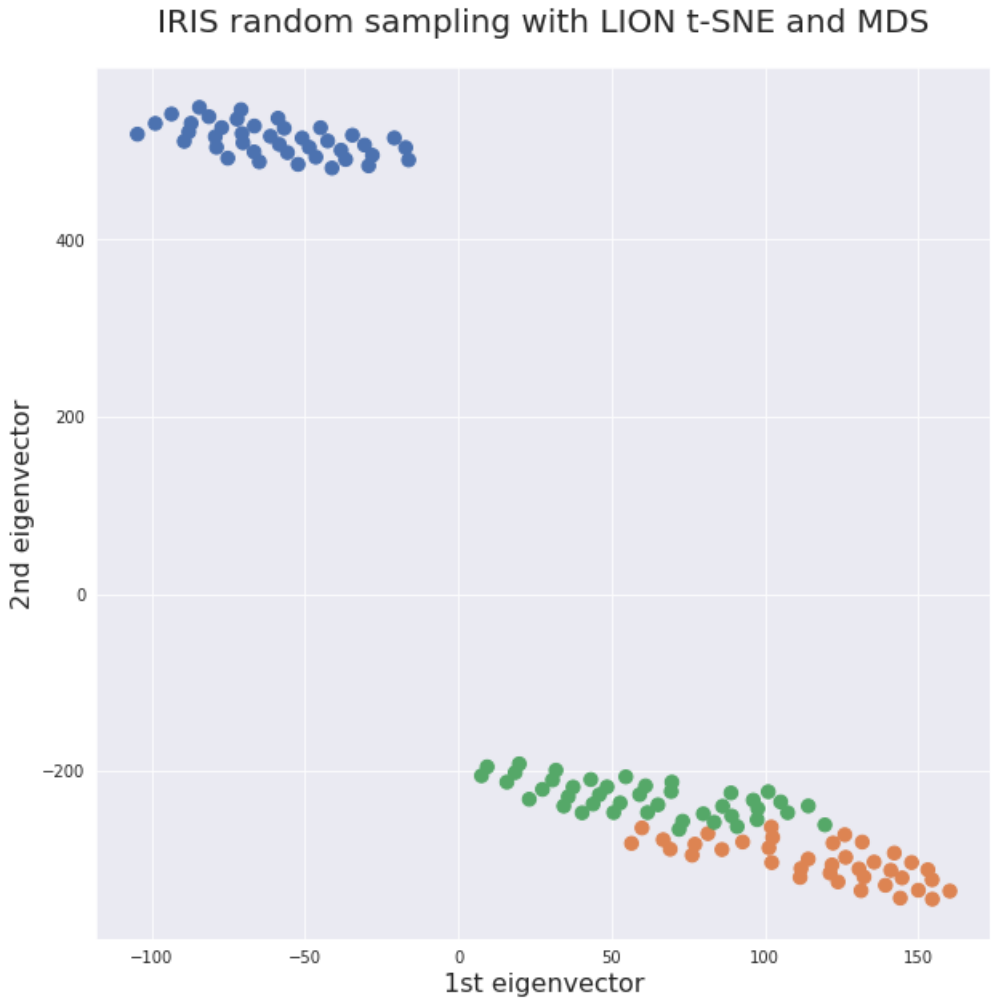
In [0]:

```
%%capture
iris_mds = MDS(n_components=4)
X_iris_mds = iris_mds.fit_transform(iris_random_df.loc[:, features].values)

lionTSNE_iris = lion_tsne.LionTSNE(perplexity=30)
```

```
layers = lionTSNE_iris.fit(X_iris_mds, optimizer_kwargs={'momentum': 0.8, 'n_iter': 3000,
                                                         'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [184]:

```
plot_iris_2d(
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS random sampling with LION t-SNE and MDS',
    colors=iris_random_df.loc[:, ['target']].values)
```



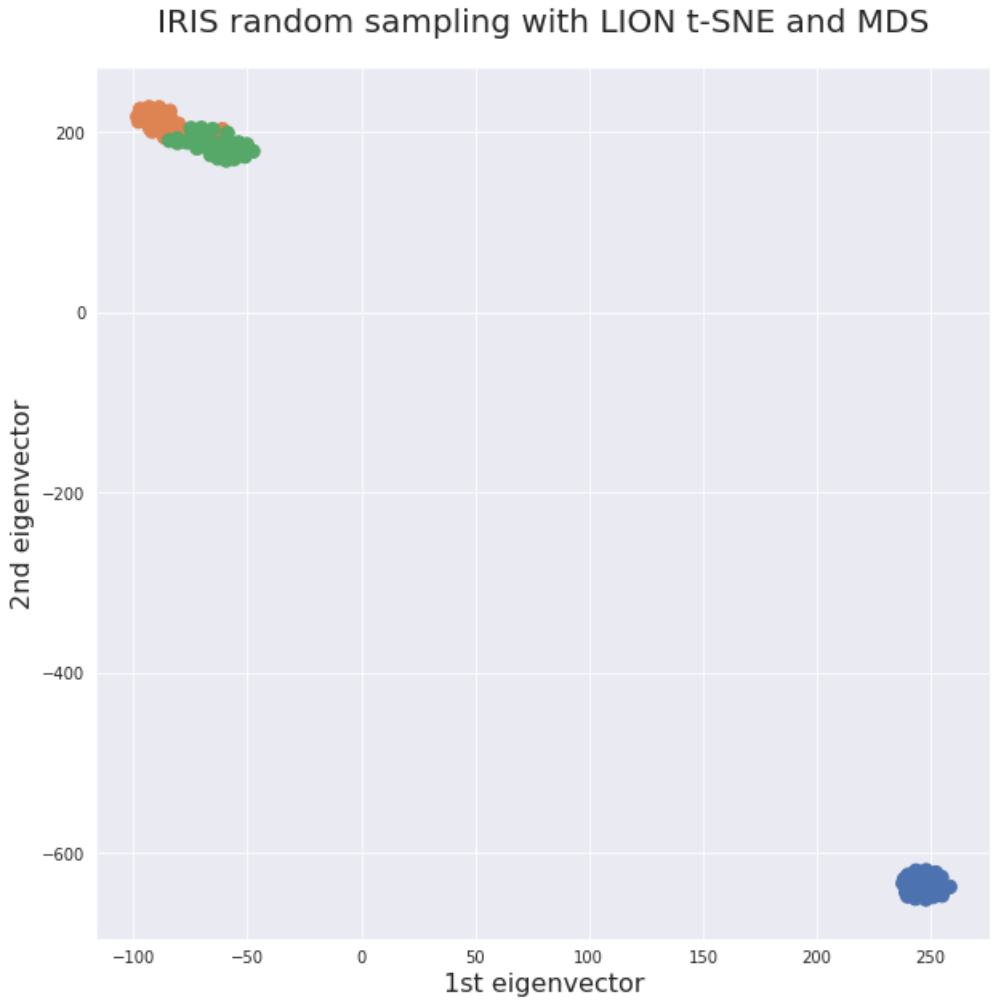
kNN sampling with MDS

In [0]:

```
%%capture
iris_mds = MDS(n_components=4)
X_iris_mds = iris_mds.fit_transform(iris_knn_df.loc[:, features].values)

lionTSNE_iris = lion_tsne.LionTSNE(perplexity=30)
layers = lionTSNE_iris.fit(X_iris_mds, optimizer_kwargs={'momentum': 0.8, 'n_iter': 3000,
                                                         'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [187]:

```
plot_iris_2d(
    x = layers[:, 0],
    y = layers[:, 1],
    title = 'IRIS random sampling with LION t-SNE and MDS',
    colors=iris_knn_df.loc[:, ['target']].values)
```



MNIST DATASET TESTS

In [0]:

```
def plot_mnist_2d(df, title, xlabel="1st eigenvector", ylabel="2nd eigenvector"):
    plt.gcf().set_size_inches(20,15)
    sns.set_style("darkgrid")
    legend_list = list()

    for d in data_digits.target_names:
      plt.scatter(df[df['target'] == str(d)]['1st eigenvector'], df[df['target'] == str(d)]['2nd eigenv
ector'])
      legend_list.append(str(d))

    plt.title(title, fontsize=20, y=1.03)
    plt.legend(legend_list)
    plt.xlabel(xlabel, fontsize=16)
    plt.ylabel(ylabel, fontsize=16)
```

In [52]:

```
data_digits = datasets.load_digits()
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

all_mnist_trained_images = mnist.train.images
all_mnist_labels = mnist.train.labels
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

Random sampling

In [0]:

```
np.random.seed(RANDOM_STATE)
ind = np.random.choice(np.arange(len(mnist.train.images)), size = 2000)

mnist_chosen_indices = ind
X_mnist_rand = mnist.train.images[ind]
y_mnist_raw = mnist.train.labels[ind]
y_mnist_rand = [np.where(r==1)[0][0] for r in y_mnist_raw]
```

In [59]:

```
mnist_random_df = pd.DataFrame(X_mnist_rand)
mnist_random_df['target'] = y_mnist_rand
mnist_random_df['target'].value_counts()
```

Out[59]:

```
7    230
1    228
9    216
3    206
5    193
0    193
8    190
6    188
2    186
4    170
Name: target, dtype: int64
```

kNN sampling

In [0]:

```
def get_mnist_neighbors(train, test_row, num_neighbors, class_type):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
  distances.sort(key=lambda tup: tup[1])
  result_df = pd.DataFrame()
  for i in range(num_neighbors):
    tmp_df = pd.DataFrame([distances[i][0]])
    result_df = result_df.append(tmp_df, ignore_index=True)

  result_df['target'] = class_type
  return result_df
```

In [70]:

```
clf = NearestCentroid()
mnist_train_labels = [np.where(r==1)[0][0] for r in mnist.train.labels[:12000]]
clf.fit(mnist.train.images[:12000], mnist_train_labels)
clf.centroids_
```

Out[70]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [79]:

```
mnist_target_values = np.unique(mnist_train_labels)
mnist_knn_df = pd.DataFrame()
```

```
for i in mnist_target_values:
  mnist_knn_df = mnist_knn_df.append(get_mnist_neighbors(mnist.train.images[:12000], clf.centroids_[i],
200, str(i)), ignore_index=True)

mnist_knn_df['target'].value_counts()
```

Out[79]:

```
2    200
0    200
3    200
6    200
9    200
1    200
8    200
7    200
4    200
5    200
Name: target, dtype: int64
```

tSNE MNIST with random sampling

In [0]:

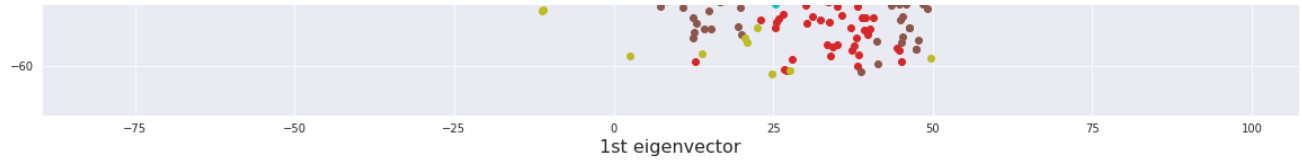
```
tsne = TSNE(n_components=2, n_iter=3000, random_state=RANDOM_STATE)
layers = tsne.fit_transform(mnist_random_df.loc[:, mnist_random_df.columns != 'target'].values)
```

In [0]:

```
mnist_df = pd.DataFrame(layers, columns=['1st eigenvector', '2nd eigenvector'])
mnist_df['target'] = mnist_random_df['target'].values
```

In [88]:

```
plot_mnist_2d(
    mnist_df,
    title = 'MNIST tSNE with random sampling')
```



tSNE MNIST with kNN sampling

In [0]:

```
tsne = TSNE(n_components=2, n_iter=3000, random_state=RANDOM_STATE)
layers = tsne.fit_transform(mnist_knn_df.loc[:, mnist_knn_df.columns != 'target'].values)
```

In [0]:

```
mnist_df = pd.DataFrame(layers, columns=['1st eigenvector', '2nd eigenvector'])
mnist_df['target'] = mnist_knn_df['target'].values
```

In [119]:

```
plot_mnist_2d(
    mnist_df,
    title = 'MNIST tSNE with kNN sampling')
```

LION tSNE MNIST

with random sampling

In [0]:

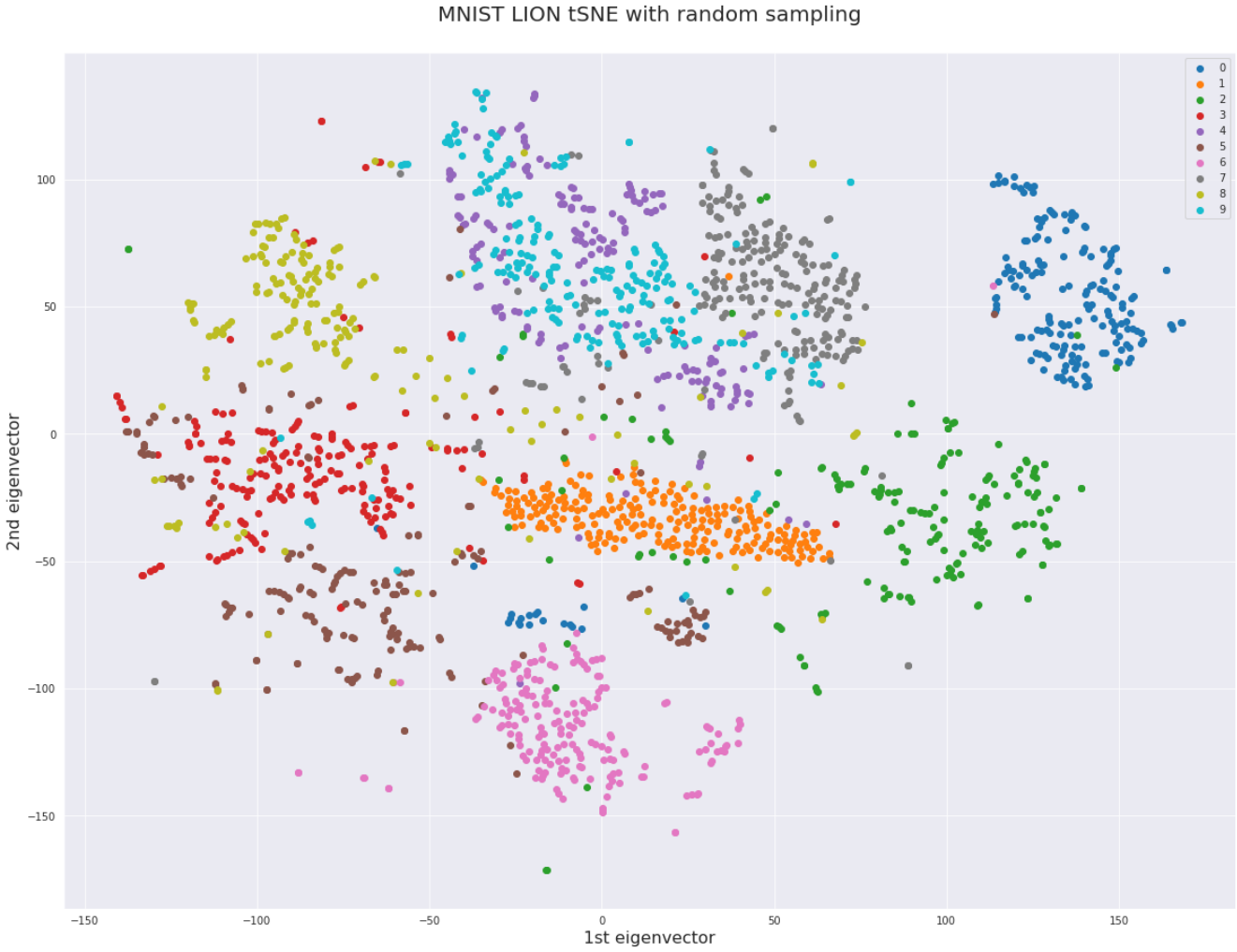
```
%%capture
lionTSNE_mnist = lion_tsne.LionTSNE(perplexity=30)
Y_lionTSNE_mnist = lionTSNE_mnist.fit(mnist_random_df.loc[:, mnist_random_df.columns != 'target'].value
s, optimizer_kwargs={'momentum': 0.8, 'n_iter': 3000,
                                                    'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [0]:

```
lionrn_mnist_df = pd.DataFrame(Y_lionTSNE_mnist, columns=['1st eigenvector', '2nd eigenvector'])
lionrn_mnist_df['target'] = mnist_random_df['target'].values
```

In [127]:

```
plot_mnist_2d(
    lionrn_mnist_df,
    title = 'MNIST LION tSNE with random sampling')
```



with kNN sampling

In [0]:

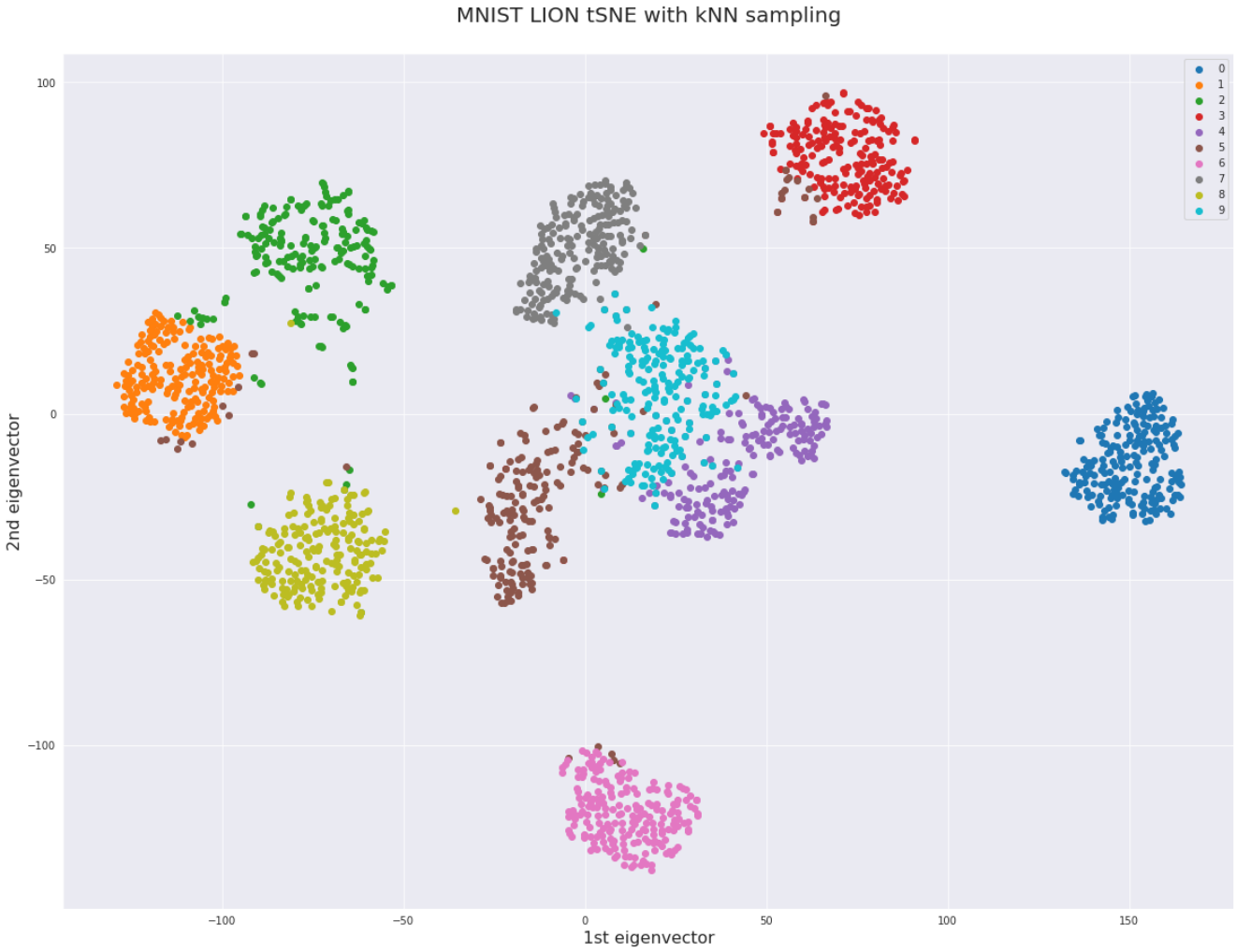
```
%%capture
lionTSNE_mnist = lion_tsne.LionTSNE(perplexity=30)
Y_lionTSNE_knn_mnist = lionTSNE_mnist.fit(mnist_knn_df.loc[:, mnist_knn_df.columns != 'target'].values,
optimizer_kwargs={'momentum': 0.8, 'n_iter': 3000,
                                                    'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [0]:

```
lionknn_mnist_df = pd.DataFrame(Y_lionTSNE_knn_mnist, columns=['1st eigenvector', '2nd eigenvector'])
lionknn_mnist_df['target'] = mnist_knn_df['target'].values
```

In [151]:

```
plot_mnist_2d(
    lionknn_mnist_df,
    title = 'MNIST LION tSNE with kNN sampling')
```



with MDS

In [0]:

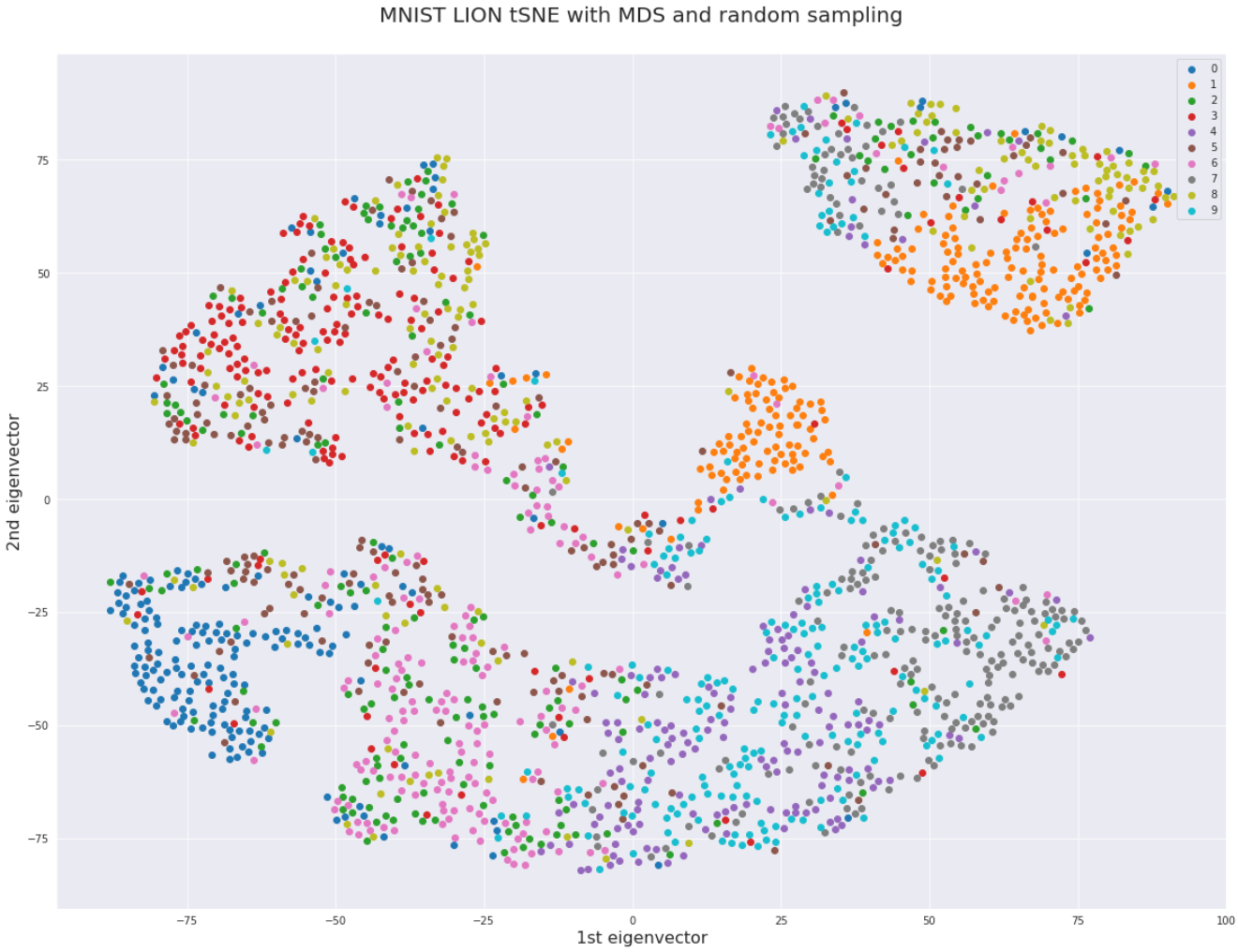
```
%%capture
mnist_mds = MDS(n_components=2)
X_mnist_mds = mnist_mds.fit_transform(mnist_random_df.loc[:, mnist_random_df.columns != 'target'].value
s)

lionTSNE_mnist = lion_tsne.LionTSNE(perplexity=30)
Y_lionTSNE_mds_rn_mnist = lionTSNE_mnist.fit(X_mnist_mds, optimizer_kwargs={'momentum': 0.8, 'n_iter':
3000,
                                                    'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [0]:

```
mnist_lion_mds_df = pd.DataFrame(Y_lionTSNE_mds_rn_mnist, columns=['1st eigenvector', '2nd eigenvector'
])
mnist_lion_mds_df['target'] = mnist_random_df['target'].values
```

In [143]:

```
plot_mnist_2d(
    mnist_lion_mds_df,
    title = 'MNIST LION tSNE with MDS and random sampling')
```



In [0]:

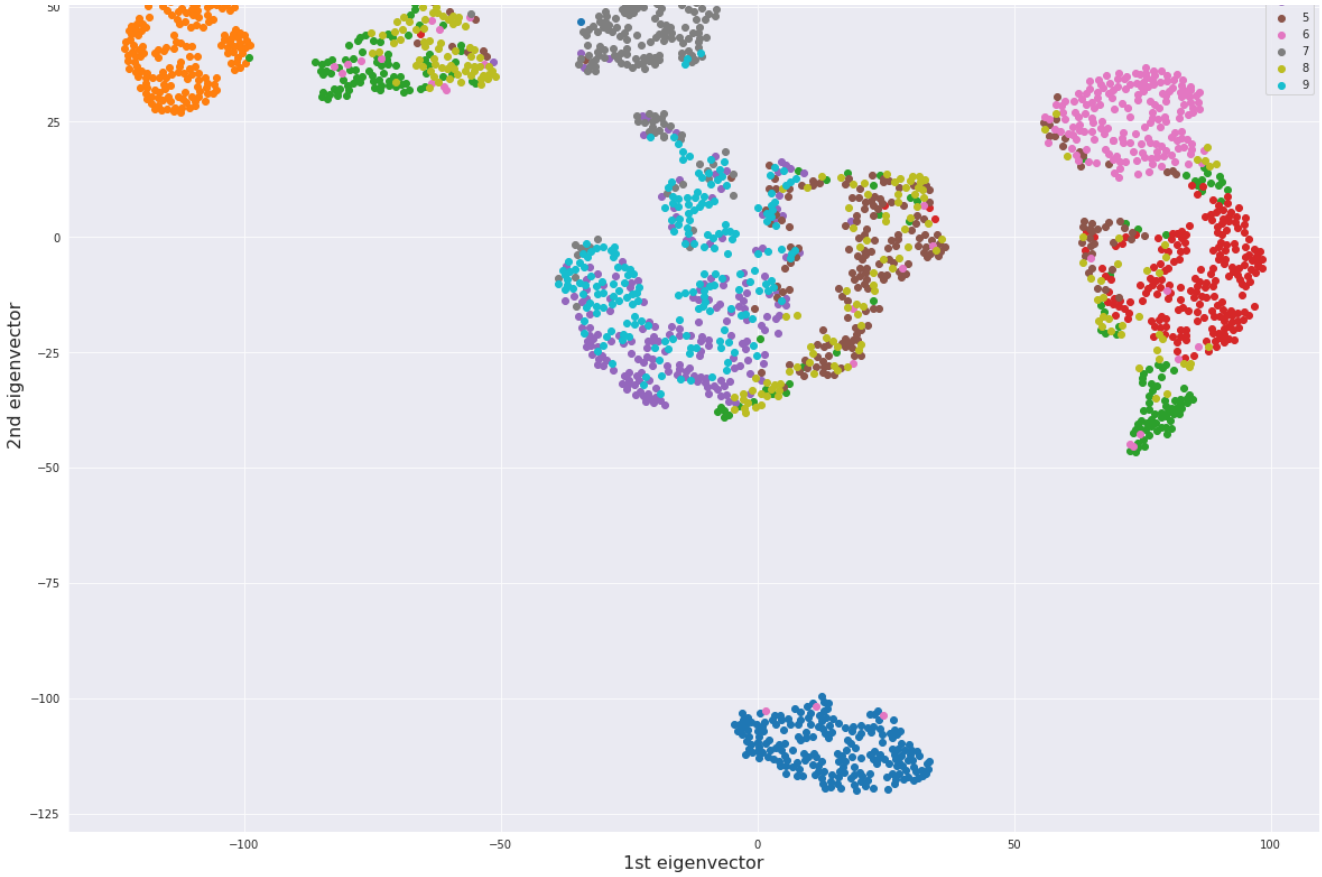
```
%%capture
mnist_mds = MDS(n_components=2)
X_mnist_mds = mnist_mds.fit_transform(mnist_knn_df.loc[:, mnist_knn_df.columns != 'target'].values)

lionTSNE_mnist = lion_tsne.LionTSNE(perplexity=30)
Y_lionTSNE_mds_knn_mnist = lionTSNE_mnist.fit(X_mnist_mds, optimizer_kwargs={'momentum': 0.8, 'n_iter':
3000,
                                                    'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [0]:

```
mnist_lion_knn_mds_df = pd.DataFrame(Y_lionTSNE_mds_knn_mnist, columns=['1st eigenvector', '2nd eigenve
ctor'])
mnist_lion_knn_mds_df['target'] = mnist_knn_df['target'].values
```

In [150]:

```
plot_mnist_2d(
    mnist_lion_knn_mds_df,
    title = 'MNIST LION tSNE with MDS and knn sampling')
```



In [0]:

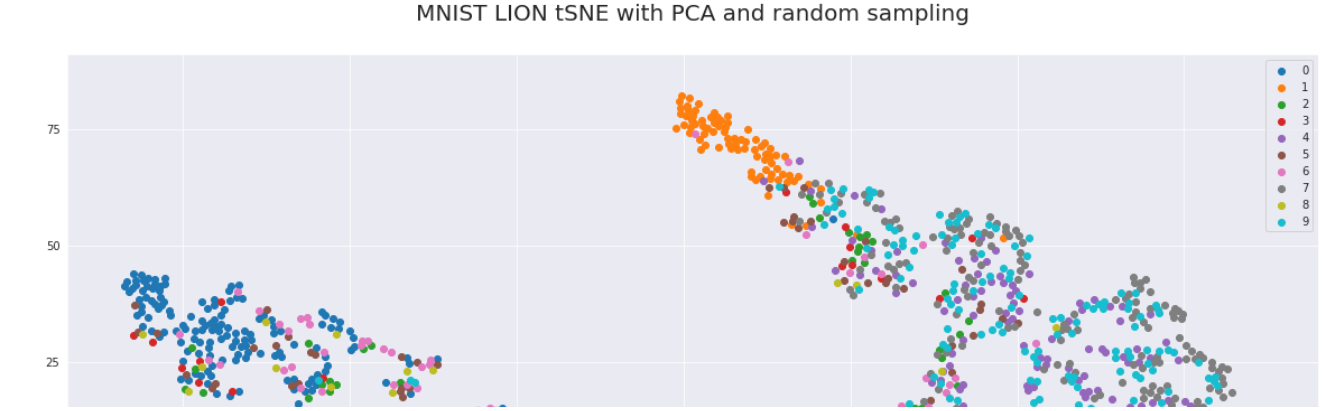
```
%%capture
mnist_pca = PCA(n_components=2)
X_mnist_pca = mnist_pca.fit_transform(mnist_random_df.loc[:, mnist_random_df.columns != 'target'].value
s)

lionTSNE_mnist = lion_tsne.LionTSNE(perplexity=30)
Y_lionTSNE_pca_rn_mnist = lionTSNE_mnist.fit(X_mnist_pca, optimizer_kwargs={'momentum': 0.8, 'n_iter':
3000,
                                                    'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

In [0]:

```
mnist_lion_pca_df = pd.DataFrame(Y_lionTSNE_pca_rn_mnist, columns=['1st eigenvector', '2nd eigenvector'
])
mnist_lion_pca_df['target'] = mnist_random_df['target'].values
```

In [191]:

```
plot_mnist_2d(
    mnist_lion_pca_df,
    title = 'MNIST LION tSNE with PCA and random sampling')
```



In [0]:

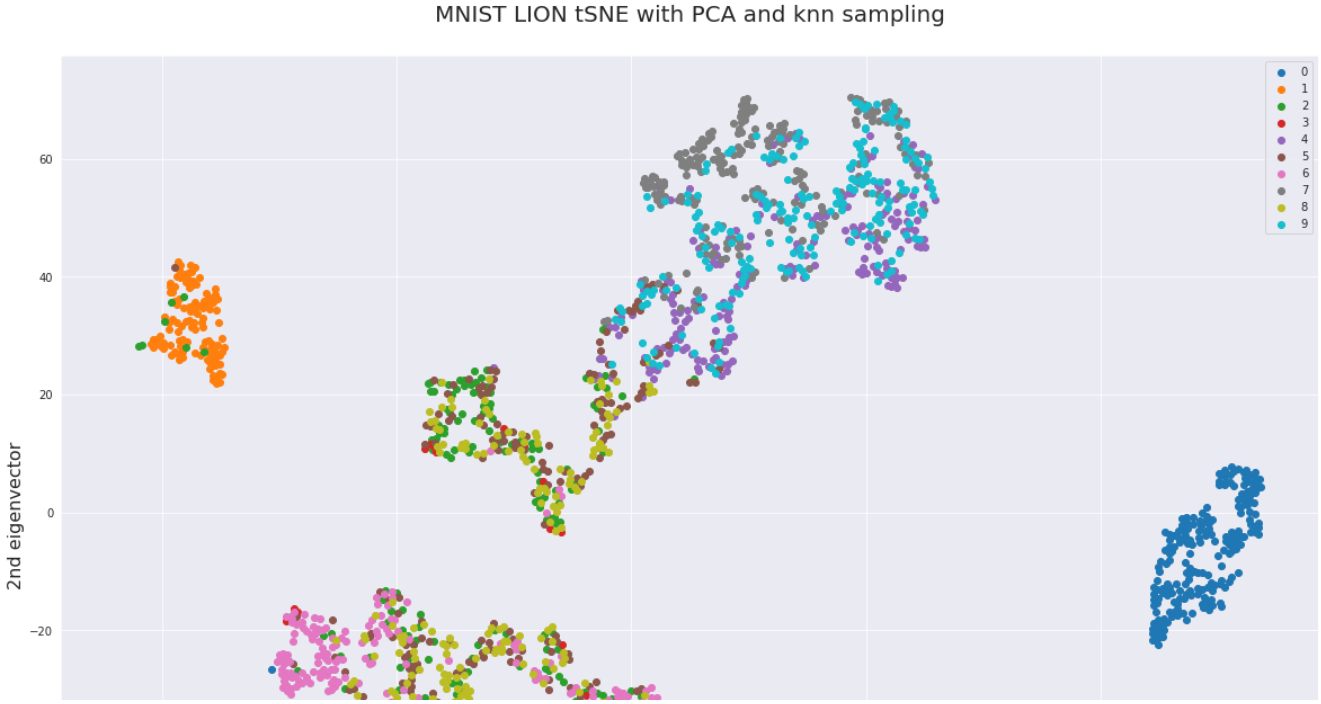
```
%%capture
mnist_pca = PCA(n_components=2)
X_mnist_pca = mnist_pca.fit_transform(mnist_knn_df.loc[:, mnist_knn_df.columns != 'target'].values)

lionTSNE_mnist = lion_tsne.LionTSNE(perplexity=30)
Y_lionTSNE_pca_knn_mnist = lionTSNE_mnist.fit(X_mnist_pca, optimizer_kwargs={'momentum': 0.8, 'n_iter':
3000,
                                                                             'early_exaggeration_iters' : 300}, random_seed=1,
verbose=2)
```

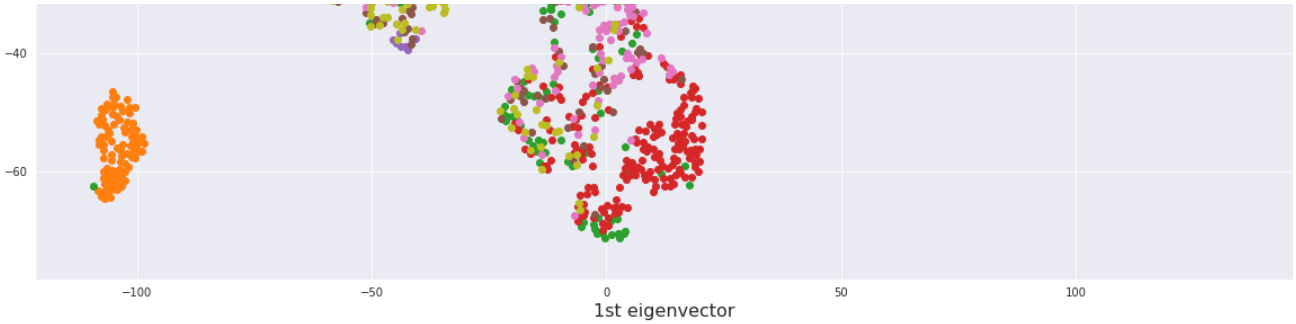
In [0]:

```
mnist_lion_knn_pca_df = pd.DataFrame(Y_lionTSNE_pca_knn_mnist, columns=['1st eigenvector', '2nd eigenve
ctor'])
mnist_lion_knn_pca_df['target'] = mnist_knn_df['target'].values
```

In [195]:

```
plot_mnist_2d(
    mnist_lion_knn_pca_df,
    title = 'MNIST LION tSNE with PCA and knn sampling')
```




−40
−60
−100
−50
0
50
100
1st eigenvector