Project 2 - Dimensionality Reduction

About the Dataset

In [2]:

```
class_names = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', '$', '#']
```

Let's visualize the dataset:

In [3]:

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')

# Loading Training Data
X_train = np.load('data_train.npy').T
X_test = np.load('data_test.npy').T
t_train = np.load('labels_train_corrected.npy')

print(X_train.shape, t_train.shape)
```

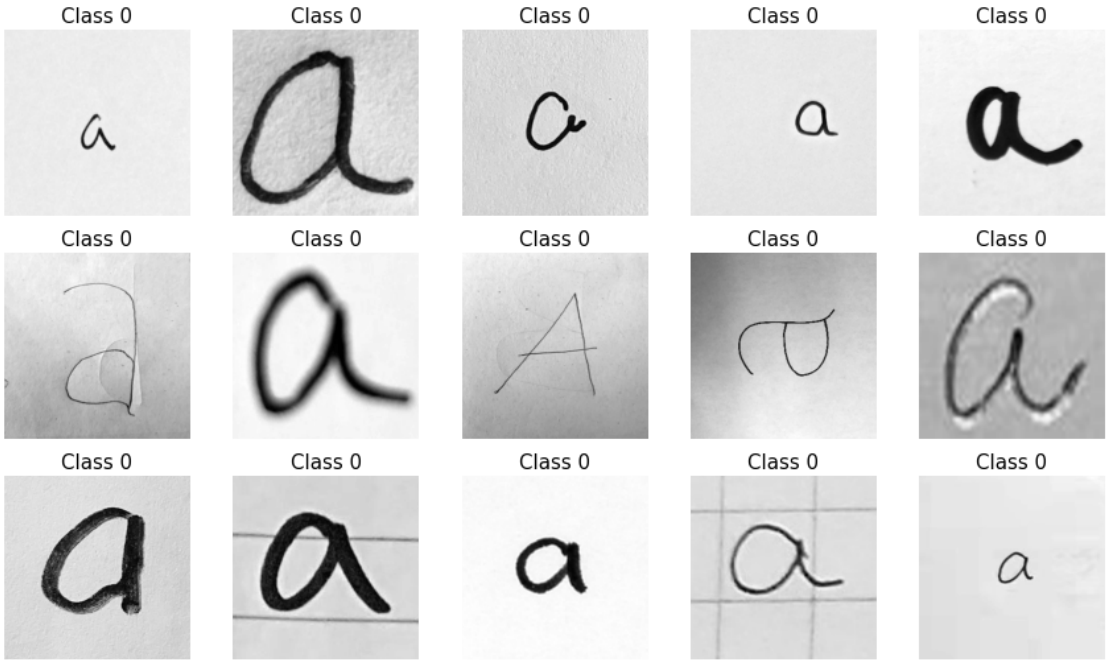
```
(6720, 90000) (6720,)
```

In [4]:

```
# Counting number samples per class
vals, counts = np.unique(t_train, return_counts=True)

plt.bar(vals, counts)
plt.xticks(range(10),range(10))
plt.xlabel('Classes',size=20)
plt.ylabel('# Samples per Class', size=20)
plt.title('Training Data (Total = '+str(X_train.shape[1])+' samples)',size=15);
```

In [5]:

```
# Displaying some random examples per class

for i in range(0,10):
    rnd_sample = npr.permutation(np.where(t_train==i)[0])
    fig=plt.figure(figsize=(15,15))
    for j in range(25):
        fig.add_subplot(5,5,j+1)
        plt.imshow(X_train[rnd_sample[j],:].reshape((300,300)),cmap='gray')
        plt.axis('off');plt.title('Class '+str(int(t_train[rnd_sample[j]])),size=15)
    plt.show()
    print('\n\n')
```



1. Implement Recursive Feature Elimination (RFE) to select the subset of features. Experiment with at least 2 different estimators.
    - Identify which pixels are selected and display mask examples from the training dataset.

In [6]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
import joblib as jb
```

In [7]:

```
import cv2 # use downsampling to reduce the computational expenses
def Downsampling(X, N, n):
    X_res = np.zeros((len(X), n*n))
    for i in range(len(X)):
        im = X[i,:].reshape(N,N)
        res = cv2.resize(im, dsize=(n,n), interpolation=cv2.INTER_CUBIC)
        res1 = res.reshape(1,n*n)
        X_res[i] = res1
    return X_res
X_train_res = Downsampling(X_train, 300, 50) # 50 is an acceptable downsampled size since 3
X_test_res = Downsampling(X_test, 300, 50)
X_train_res.shape, X_test_res.shape
```

Out[7]:

```
((6720, 2500), (2880, 2500))
```

In [8]:

```
# due to the fact that pixels range from 0 to 255, minmaxscaler is a good option.
scaler = MinMaxScaler()
X_train_res = scaler.fit_transform(X_train_res)
X_test_res = scaler.transform(X_test_res)
jb.dump(X_train_res, 'X_train_res.pkl')
jb.dump(X_test_res, 'X_test_res.pkl')
```

Out[8]:

```
['X_test_res.pkl']
```

In [45]:

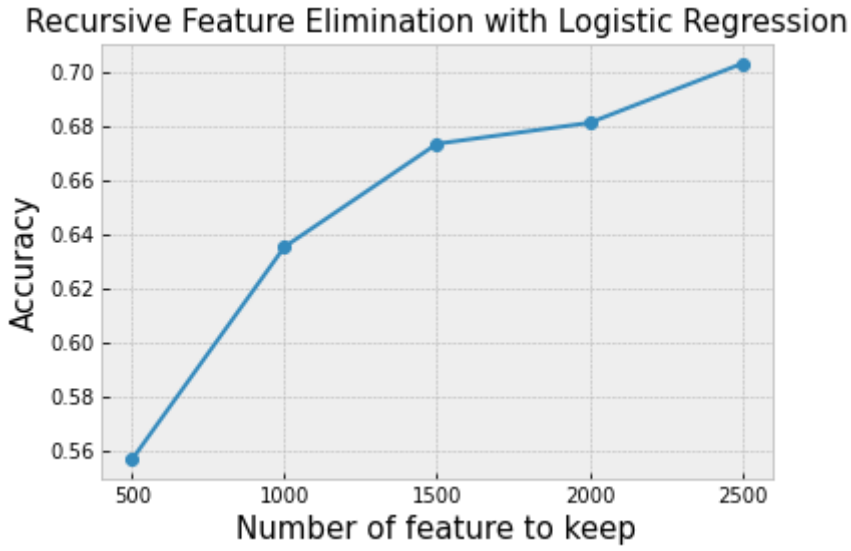
```
lr2 = LogisticRegression(solver='liblinear',max_iter=2000, tol=0.01, random_state=0)
scores1=[]
#features_to_keep1={}

for i in range(5):
    rfe1 = RFE(lr2,
           step=20, n_features_to_select=2500-i*500)
    rfe1.fit(X_train_res, t_train)
    scores1 += [rfe1.score(X_train_res, t_train)]
    #features_to_keep1[5-i] = list(np.where([rfe1.support_])[1])
```

In [46]:

```
svc = SVC(kernel='linear')
scores2=[]
#features_to_keep2={}

for i in range(5):
    rfe2 = RFE(svc,
          step=20, n_features_to_select=2500-i*500)
    rfe2.fit(X_train_res, t_train)
    scores2 += [rfe2.score(X_train_res, t_train)]
    #features_to_keep2[5-i] = list(np.where([rfe2.support_])[1])
```

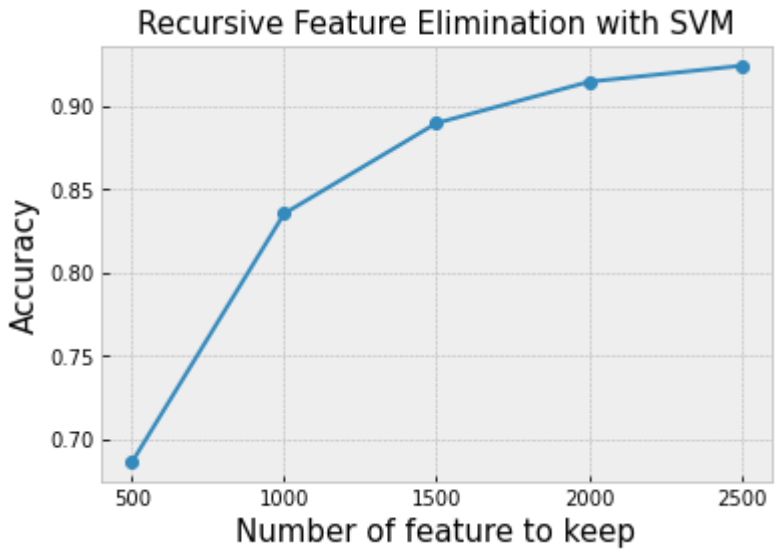
In [47]:

```
plt.plot(range(1,len(scores1)+1)[::-1],scores1,'-o')
plt.xticks(range(1,len(scores1)+1)[::-1], [2500,2000,1500,1000,500])
plt.xlabel('Number of feature to keep',size=15)
plt.ylabel('Accuracy', size=15)
plt.title('Recursive Feature Elimination with Logistic Regression', size=15);
```



In [19]:

```
plt.plot(range(1,len(scores2)+1)[::-1],scores2,'-o')
plt.xticks(range(1,len(scores2)+1)[::-1], [2500,2000,1500,1000,500])
plt.xlabel('Number of feature to keep',size=15)
plt.ylabel('Accuracy', size=15)
plt.title('Recursive Feature Elimination with SVM', size=15);
```

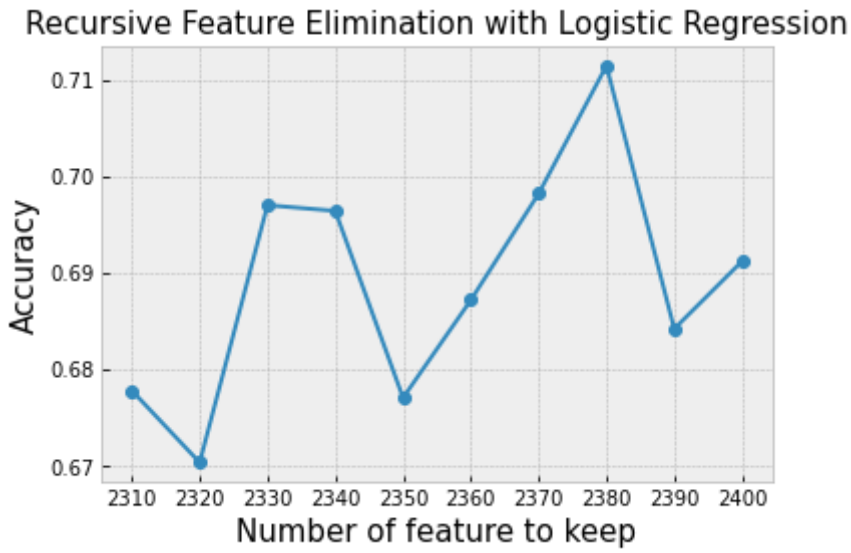


In [17]:

```
scores1=[]
features_to_keep1={}

for i in range(10):
    rfe1 = RFE(lr2,
           step=20, n_features_to_select=2400-i*10)
    rfe1.fit(X_train_res, t_train)
    scores1 += [rfe1.score(X_train_res, t_train)]
    features_to_keep1[10-i] = list(np.where([rfe1.support_])[1])
```

In [21]:

```
plt.plot(range(1,len(scores1)+1)[::-1],scores1,'-o')
plt.xticks(range(1,len(scores1)+1)[::-1], [2400,2390,2380,2370,2360,2350,2340,2330,2320,231
plt.xlabel('Number of feature to keep',size=15)
plt.ylabel('Accuracy', size=15)
plt.title('Recursive Feature Elimination with Logistic Regression', size=15);
```



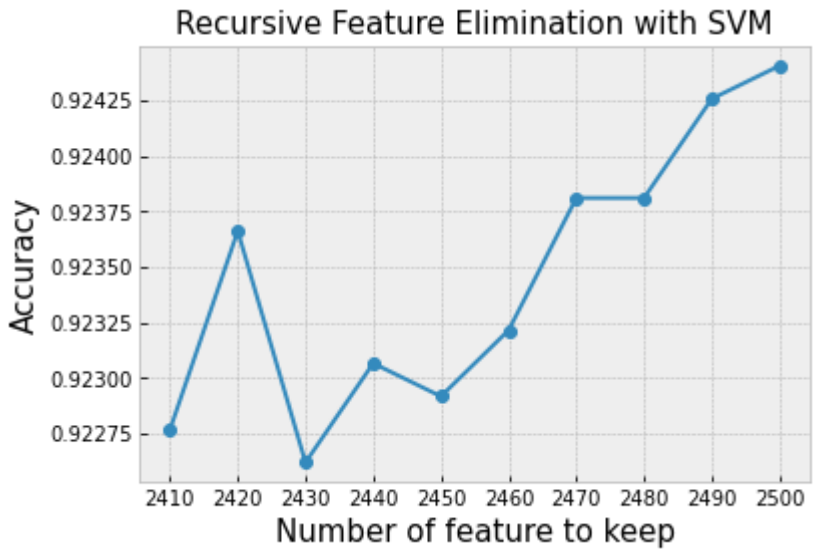
The result shows that big accuracy drop star from about 2320 for RFE with logistic regression as estimator, before that, accuracy are a little jaggy but in average are 0.69.

In [13]:

```
scores2=[]
features_to_keep2={}

for i in range(10):
    rfe2 = RFE(svc,
          step=20, n_features_to_select=2500-i*10)
    rfe2.fit(X_train_res, t_train)
    scores2 += [rfe2.score(X_train_res, t_train)]
    features_to_keep2[10-i] = list(np.where([rfe2.support_])[1])
```

In [14]:

```
plt.plot(range(1,len(scores2)+1)[::-1],scores2,'-o')
plt.xticks(range(1,len(scores2)+1)[::-1], [2500,2490,2480,2470,2460,2450,2440,2430,2420,241
plt.xlabel('Number of feature to keep',size=15)
plt.ylabel('Accuracy', size=15)
plt.title('Recursive Feature Elimination with SVM', size=15);
```



The result show that big accuracy drop start from about 2480 for RFE with SVM as estimator.

In [52]:

```
print('For RFE with logisticregression as estimator between 2310~2400:\n')
for i in range(10):
      print(features_to_keep1[i+1], '\n')
print('\n\n For RFE with svm as estimator between 2410~2500:\n')
for i in range(10):
      print(features_to_keep2[i+1], '\n')
```

```
For RFE with logisticregression as estimator between 2310~2400:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 35, 36, 37, 38, 41, 43, 44, 45, 4
6, 47, 48, 50, 51, 52, 53, 54, 55, 57, 58, 59, 60, 61, 62, 63, 64, 67, 6
8, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 8
7, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 105,
106, 107, 108, 109, 110, 111, 113, 114, 115, 117, 118, 119, 121, 122, 12
3, 125, 126, 128, 129, 131, 132, 133, 134, 135, 136, 138, 140, 141, 142,
143, 144, 145, 146, 147, 150, 151, 152, 153, 154, 155, 156, 157, 158, 15
9, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 173, 174,
175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 18
9, 190, 191, 192, 193, 194, 195, 196, 197, 199, 200, 201, 202, 203, 204,
205, 206, 207, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 22
0, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
238, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 25
4, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268,
269, 270, 271, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 28
4, 285, 286, 287, 288, 289, 290, 292, 293, 294, 295, 296, 297, 298, 299,
```

In [53]:

```
rfe1 = RFE(lr2, step=20, n_features_to_select=2330)
rfe2 = RFE(svc, step=20, n_features_to_select=2490)
rfe1.fit(X_train_res, t_train)
jb.dump(rfe1, 'rfe1.pkl')
rfe2.fit(X_train_res, t_train)
jb.dump(rfe2, 'rfe2.pkl')
```

Out[53]:

```
['rfe2.pkl']
```

In [54]:

```
print('These pixels are selected for each estimator:')
print('For RFE with logistic regression:\n', np.where([rfe1.support_])[1])
print('\n\n For RFE with random forest classifier:\n', np.where([rfe2.support_])[1])
```

```
These pixels are selected for each estimator:
For RFE with logistic regression:
 [   0    1    2 ... 2495 2498 2499]


 For RFE with random forest classifier:
 [   0    1    2 ... 2497 2498 2499]
```

In [ ]:

In [ ]:

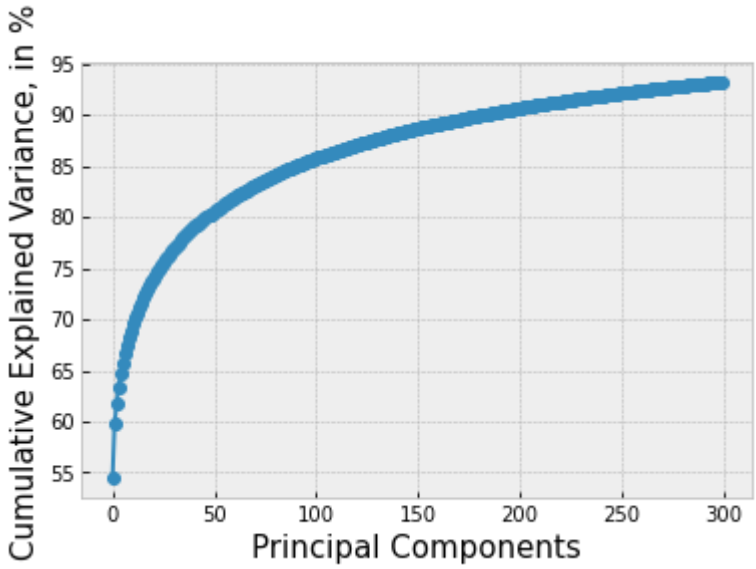
2. Implement Principal Component Analysis (PCA) to select the number of components that explain at least 90% of the explained variance. Train a classifier on the original dataset and the reduced dataset.
   - Was training faster using the reduced dataset?
   - Compare performances.
   - Visualize the top 10 eigenvectors. Discuss what they represent.
   - Visualize examples of image reconstruction from PCA projections.

In [8]:

```
from sklearn.decomposition import PCA
import time
```

In [9]:

```
pca = PCA(n_components=300)
pca.fit(X_train_res)

plt.plot(100*np.cumsum(pca.explained_variance_ratio_), '-o')
plt.xlabel('Principal Components',size=15)
plt.ylabel('Cumulative Explained Variance, in %', size=15);
```

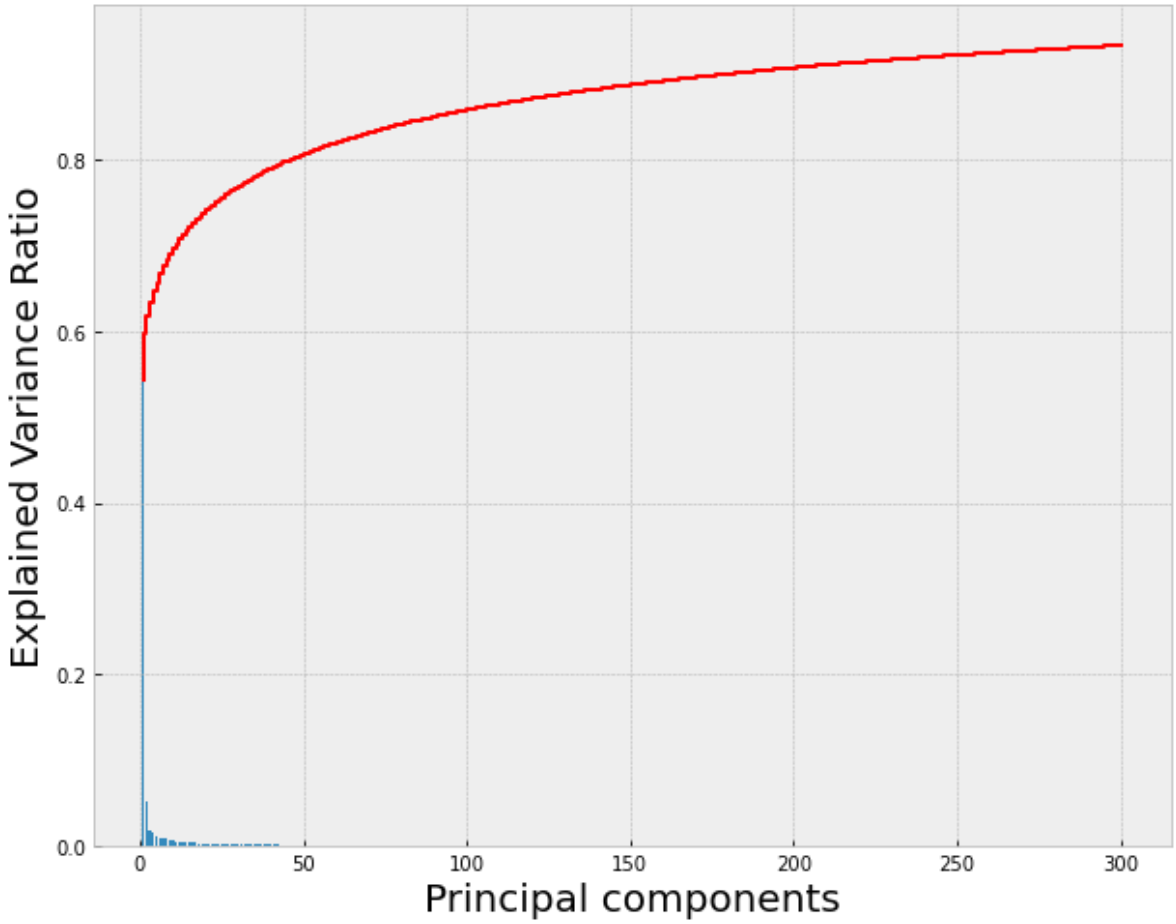


In [10]:

```
plt.figure(figsize=(10,8))
plt.step(range(1,301), np.cumsum(pca.explained_variance_ratio_), c='r')
plt.bar(range(1,301), pca.explained_variance_ratio_)
plt.xlabel('Principal components', fontsize=20)
plt.ylabel('Explained Variance Ratio', fontsize=20)
```

Out[10]:

```
Text(0, 0.5, 'Explained Variance Ratio')
```



In [11]:

```
np.where(np.cumsum(pca.explained_variance_ratio_)>=0.9)
```

Out[11]:

```
(array([182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
        195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
        208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
        221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
        234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
        247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,
        260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,
        273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
        286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
        299]),)
```

In [12]:

```
np.cumsum(pca.explained_variance_ratio_)[183]
```

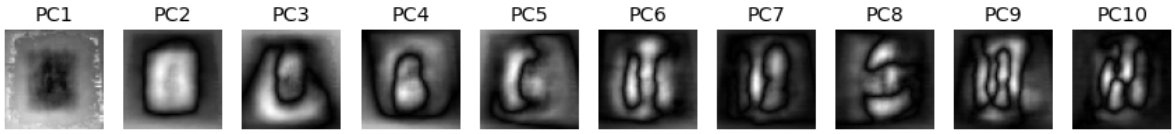
Out[12]:

```
0.9005327903512633
```

Based on the result, we can see that the number of components that explain at least 90% of the explained variance is 183.

In [13]:

```
plt.figure(figsize=(15,12))
for i in range(10):
    plt.subplot(1,10,i+1)
    plt.imshow(abs(pca.components_[i,:].reshape(50,50)), cmap='gray')
    plt.title('PC'+str(i+1))
    plt.axis('off')
```



The top 10 eigenvectors explain the most variance that we can use to reconstruct the image.

- Eigenvectors like PC1 and PC2 show the region that most characters exist is in the middle square area.
- Eigenvectors PC3 and PC4 show that in the middle square area, some characters tend to have a shape like ellipe in the middle.
- Eigenvectors PC5, PC6, PC7 and PC8 show that some characters are located slightly closer to up, down, left and right.

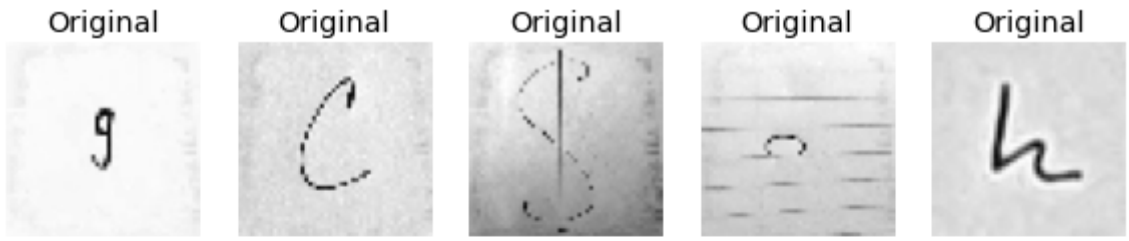
- Eigenvectors PC9 and PC10 show there exist some situation that character edges are complex but still mostly in the middle of the image.

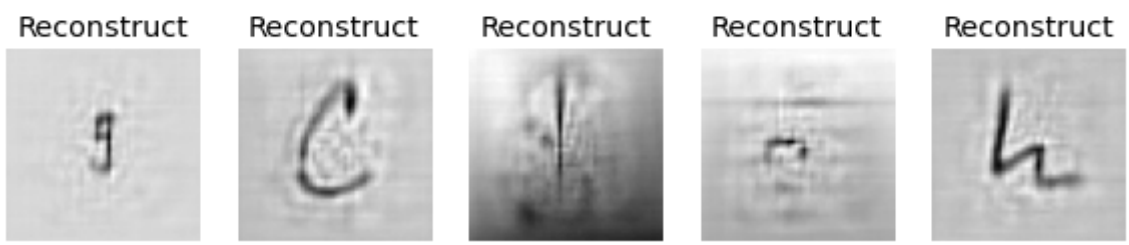
In [14]:

```
Pca = PCA(n_components=183)
y_Pca = Pca.fit_transform(X_train_res)
jb.dump(Pca, 'Pca.pkl') # store trained pca model for test file
X_reconstruct_pca = Pca.inverse_transform(y_Pca)
X_reconstrut = scaler.inverse_transform(X_reconstruct_pca)
```

In [15]:

```
N = 5
idx = np.random.choice(range(X_train_res.shape[0]), replace=False, size=N)
fig = plt.figure(figsize=(10,8))
n=1
for i in range(N):
    fig.add_subplot(2,N,n)
    plt.imshow(X_train_res[idx[i],:].reshape(50,50), cmap='gray')
    plt.title('Original')
    plt.axis('off')

    fig.add_subplot(2,N,N+n)
    plt.imshow(X_reconstrut[idx[i],:].reshape(50,50), cmap='gray')
    plt.title('Reconstruct')
    plt.axis('off')
    n+=1
```





We can see the reconstruct image is a little blur but can still visually identify the characters.

In [40]:

```
# original data
t0= time.time()
log_reg_1 = Pipeline([#('Scaler', MinMaxScaler()),
                      ('log_reg', LogisticRegression(max_iter=1000))])
log_reg_1.fit(X_train_res, t_train)
t1 = time.time()
jb.dump(log_reg_1, 'log_reg_1.pkl') # store first trained classification model for original
print("For original data, time elapsed: ", t1 - t0, 'seconds')


t0= time.time()
log_reg_2 = Pipeline([#('Scaler', MinMaxScaler()),
                       ('pca', PCA(n_components=183)),
                      ('log_reg', LogisticRegression(max_iter=1000))])
log_reg_2.fit(X_train_res, t_train)
t1 = time.time()
jb.dump(log_reg_2, 'log_reg_2.pkl') # store second trained classification model for reduced
print("For reduced data, time elapsed: ", t1 - t0, 'seconds')
```

```
For original data, time elapsed:  81.75102472305298 seconds
For reduced data, time elapsed:  4.838149547576904 seconds
```

We can see that the training time for reduced data is much more faster than original data.

---

3. Use Fisher's Linear Discriminant Analysis (LDA) and t-SNE to reduce the dataset to 2-dimensions and visualize it.
   - Visualize the dataset, be sure to color-code each point to its corresponding target label.
   - How many features would you select? Why?
   - Visualize and compare the 2-dimensional projections with PCA. Discuss your observations.

In [21]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.manifold import TSNE
```

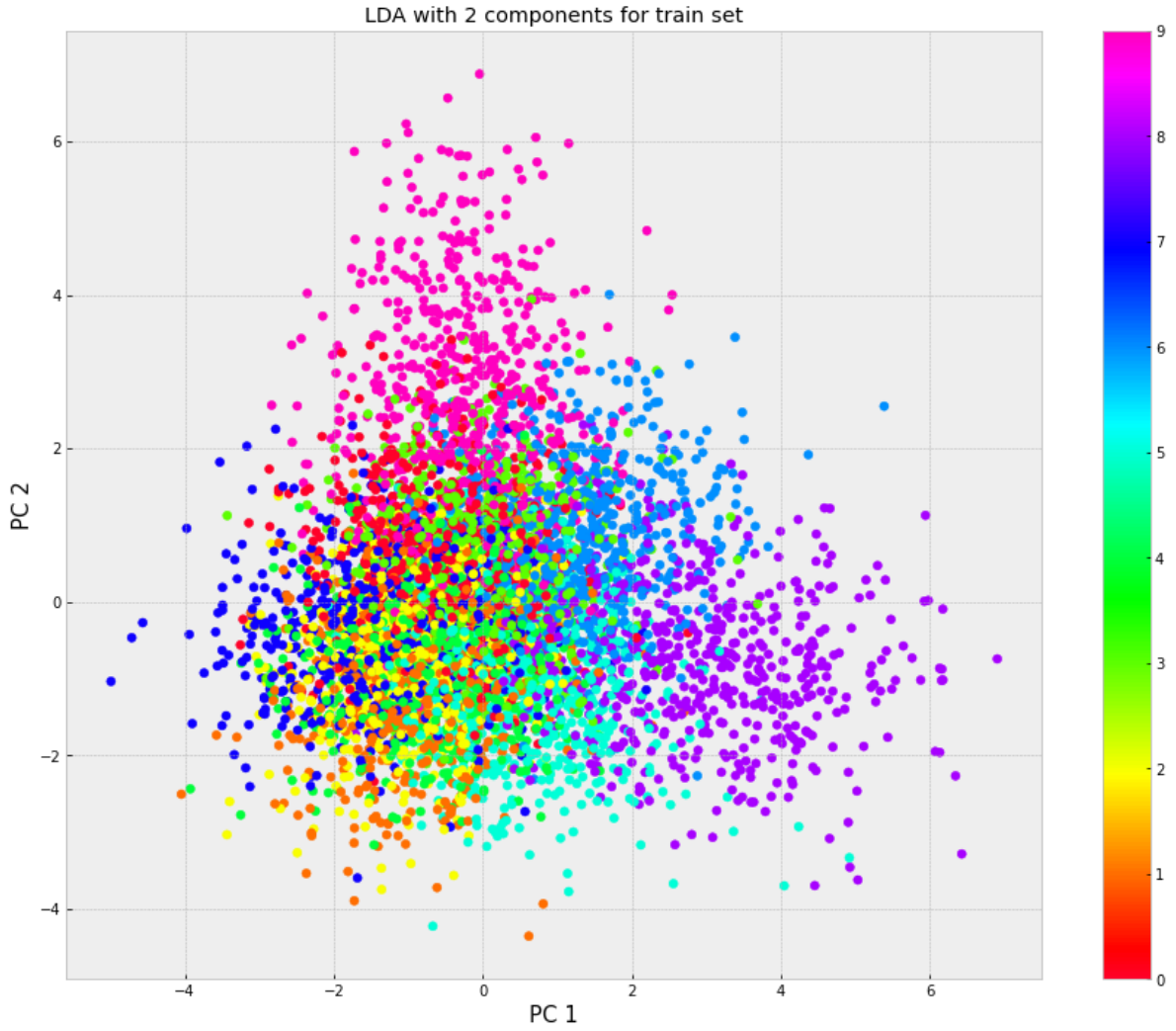
In [22]:

```
lda = LDA(n_components=2)
y_lda = lda.fit_transform(X_train_res, t_train)
jb.dump(lda, 'lda.pkl')

plt.figure(figsize=(15,12))
plt.scatter(y_lda[:,0], y_lda[:,1], c=t_train, cmap=plt.cm.gist_rainbow)
plt.xlabel('PC 1', fontsize=15)
plt.ylabel('PC 2', fontsize=15)
plt.title('LDA with 2 components for train set')
plt.colorbar()
```

```
/scratch/local/51597211/ipykernel_28370/882323541.py:10: MatplotlibDeprecati
onWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated
since 3.5 and will be removed two minor releases later; please call grid(Fal
se) first.
  plt.colorbar()
```

Out[22]:

```
<matplotlib.colorbar.Colorbar at 0x2b32bed29150>
```



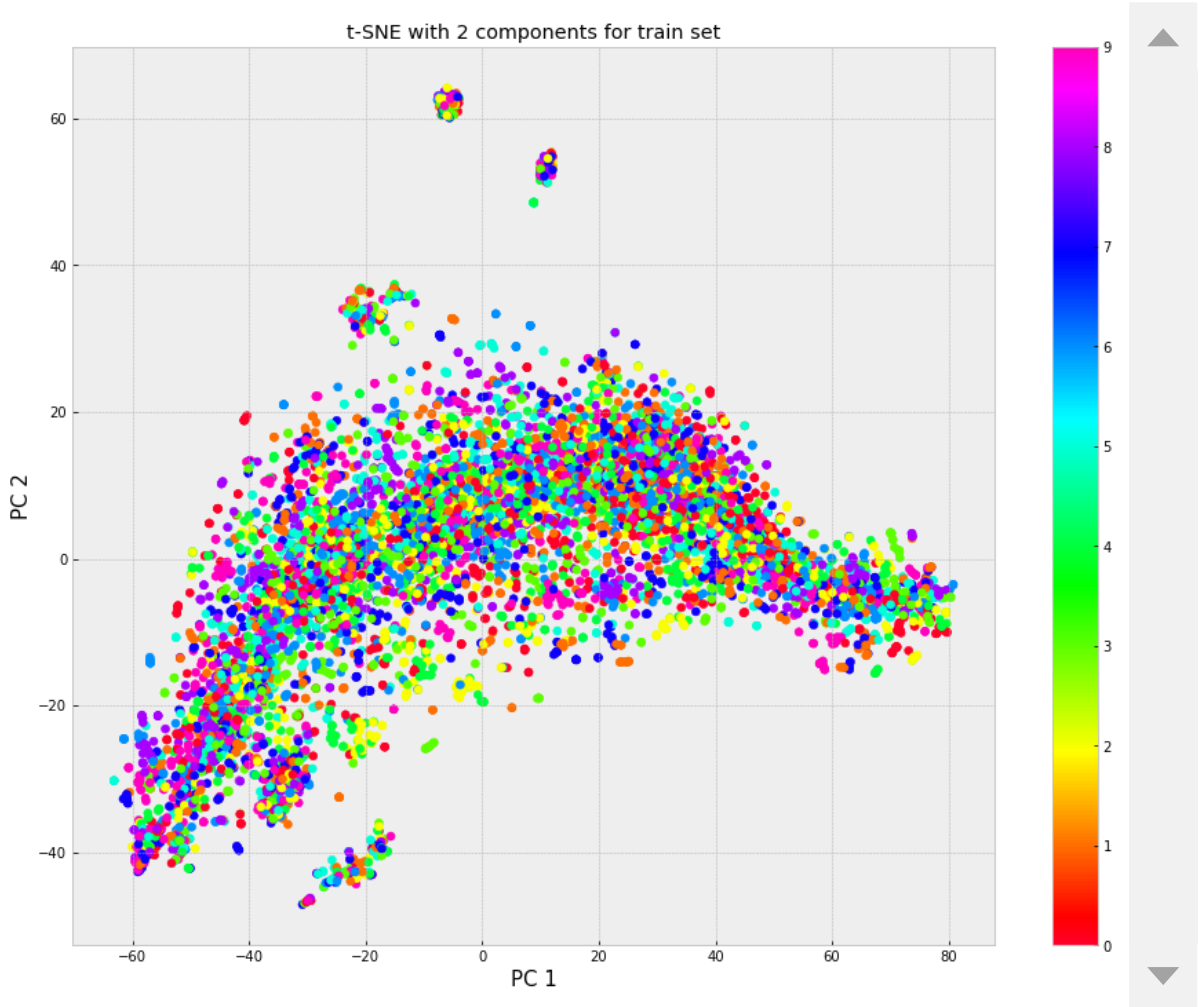
In [23]:

```
tsne = TSNE(n_components=2)
y_tsne = tsne.fit_transform(X_train_res)
jb.dump(tsne, 'tsne.pkl')

plt.figure(figsize=(15,12))
plt.scatter(y_tsne[:,0], y_tsne[:,1], c=t_train, cmap=plt.cm.gist_rainbow)
plt.xlabel('PC 1', fontsize=15)
plt.ylabel('PC 2', fontsize=15)
plt.title('t-SNE with 2 components for train set')
plt.colorbar()
```

```
/apps/python/3.10/lib/python3.10/site-packages/sklearn/manifold/_t_sne.py:80
0: FutureWarning: The default initialization in TSNE will change from 'rando
m' to 'pca' in 1.2.
  warnings.warn(
/apps/python/3.10/lib/python3.10/site-packages/sklearn/manifold/_t_sne.py:81
0: FutureWarning: The default learning rate in TSNE will change from 200.0 t
o 'auto' in 1.2.
  warnings.warn(
/scratch/local/51597211/ipykernel_28370/3558562330.py:10: MatplotlibDeprecat
ionWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated
since 3.5 and will be removed two minor releases later; please call grid(Fal
se) first.
  plt.colorbar()
```

Out[23]:

```
<matplotlib.colorbar.Colorbar at 0x2b32bf20be80>
```



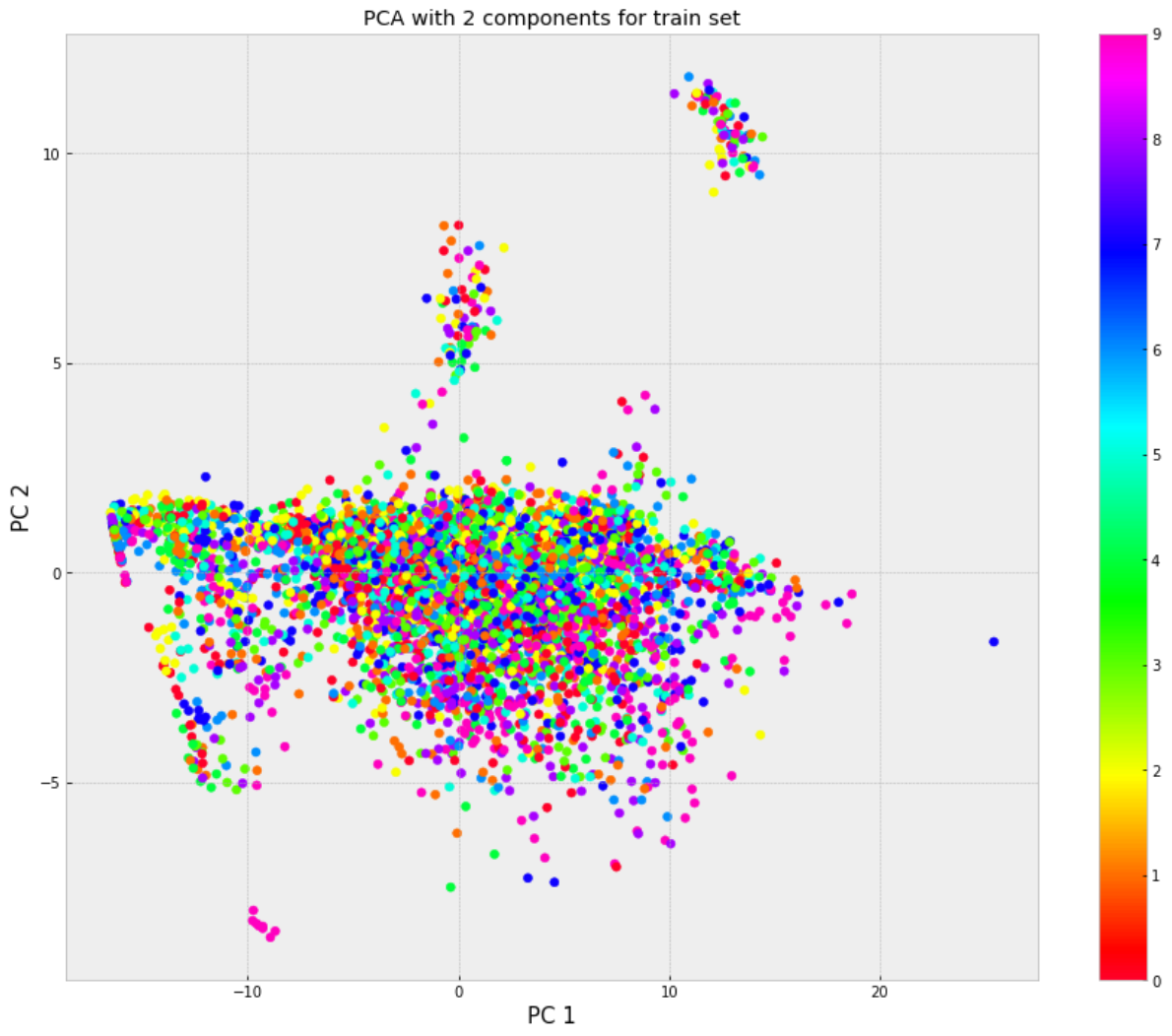
In [24]:

```
pca2 = PCA(n_components=2)
y_pca2 = pca2.fit_transform(X_train_res)
jb.dump(pca2, 'pca2.pkl')

plt.figure(figsize=(15,12))
plt.scatter(y_pca2[:,0], y_pca2[:,1], c=t_train, cmap=plt.cm.gist_rainbow)
plt.xlabel('PC 1', fontsize=15)
plt.ylabel('PC 2', fontsize=15)
plt.title('PCA with 2 components for train set')
plt.colorbar()
```

```
/scratch/local/51597211/ipykernel_28370/4058070226.py:10: MatplotlibDeprecat
ionWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated
since 3.5 and will be removed two minor releases later; please call grid(Fal
se) first.
  plt.colorbar()
```

Out[24]:

```
<matplotlib.colorbar.Colorbar at 0x2b32bf295840>
```



PCA and t-SNE are unsupervised and PCA is not able to visualize non-linear data well. Usually, t-SNE is able to visualize non-linear data into clear clusters and preserve local and global relationship, but in the result I present, overlapping in PCA and t-SNE are severe. Overlapping in t-SNE means that there is high possibility that the performance of classifier will be really bad when selecting 2 components. As

for LDA, LDA is supervised and thus is able to visually separate the data better than PCA, but there still exists some overlap and thus we can conclude that the performance when only select 2 components will not be good and LDA is a better visualization in this problem.

In [ ]:

```
```

4. Implement at least 3 manifold learning algorithms for reducing the dimensionality of the feature space. Utilize the new lower-dimensional feature space to build a classifier.

   - Which manifold learning algorithm would you select?
   - Visualize and interpret what the first 2 dimensions in the manifold learning algorithm you train.

In [15]:

```
from sklearn.manifold import MDS, Isomap
from sklearn.manifold import LocallyLinearEmbedding as LLE
from sklearn.base import BaseEstimator, TransformerMixin
import warnings
# import warning ignore
from scipy.sparse import (spdiags, SparseEfficiencyWarning, csc_matrix,
    csr_matrix, isspmatrix, dok_matrix, lil_matrix, bsr_matrix)
warnings.simplefilter("ignore", UserWarning)
warnings.simplefilter("ignore", SparseEfficiencyWarning)
```

In [16]:

```
# create a mds function to allow transform in pipeline
class mds(BaseEstimator, TransformerMixin):
    def __init__ (self, n):
        #super(mds, self).__init__(n)
        self.n = n
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        y_mds = MDS(n_components = self.n).fit_transform(X)
        return y_mds
```

In [17]:

```
pipeline4_1 = Pipeline([('mds', mds(n=0)),
                    ('log_reg', LogisticRegression())])
param4_1 = {'mds__n':range(2,10,1)}
gridsearch4_1 = GridSearchCV(pipeline4_1,
                             param_grid=param4_1,
                            cv=3,
                            scoring='accuracy')
gridsearch4_1.fit(X_train_res, t_train)
print('Best parameters: ',gridsearch4_1.best_params_)
model_mds = gridsearch4_1.best_estimator_
jb.dump(model_mds, 'model_mds.pkl')
print('Best Score:', gridsearch4_1.best_score_)
```

```
Best parameters:  {'mds__n': 6}
Best Score: 0.10744047619047618
```

In [26]:

```
pipeline4_2 = Pipeline([('isomap', Isomap()),
                    ('log_reg', LogisticRegression())])
param4_2 = {'isomap__n_components':range(5,20,1)}
gridsearch4_2 = GridSearchCV(pipeline4_2,
                             param_grid=param4_2,
                            cv=5,
                            scoring='accuracy')
gridsearch4_2.fit(X_train_res, t_train)
print('Best parameters: ',gridsearch4_2.best_params_)
model_isomap = gridsearch4_2.best_estimator_
jb.dump(model_isomap, 'model_isomap.pkl')
print('Best Score:', gridsearch4_2.best_score_)
```

```
Best parameters:  {'isomap__n_components': 17}
Best Score: 0.17232142857142857
```

In [27]:

```
pipeline4_3 = Pipeline([('lle', LLE()),
                    ('log_reg', LogisticRegression())])
param4_3 = {'lle__n_components':range(2,10,1)}
gridsearch4_3 = GridSearchCV(pipeline4_3,
                             param_grid=param4_3,
                            cv=5,
                            scoring='accuracy')
gridsearch4_3.fit(X_train_res, t_train)
print(gridsearch4_3.best_params_)
model_lle = gridsearch4_3.best_estimator_
jb.dump(model_lle, 'model_lle.pkl')
print('Best Score:', gridsearch4_3.best_score_)
```

```
{'lle__n_components': 8}
Best Score: 0.11086309523809526
```

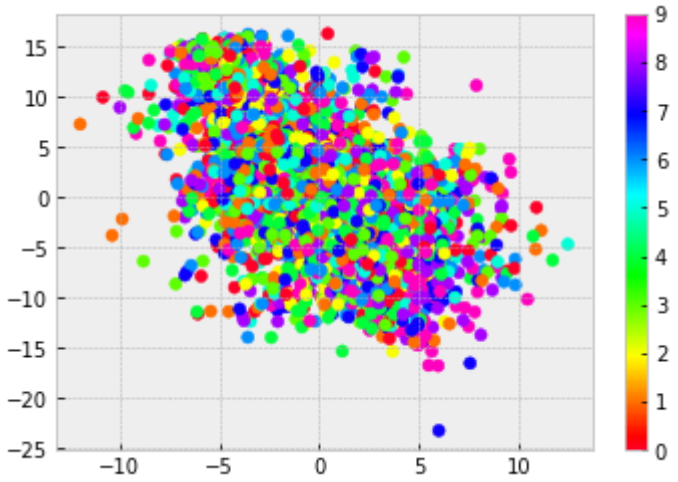
In [26]:

```
# visualize the first 2 dimensions
y_mds = model_mds.named_steps['mds'].transform(X_train_res)
plt.scatter(y_mds[:,0], y_mds[:,1], c=t_train, cmap=plt.cm.gist_rainbow);
plt.colorbar()
```

```
/scratch/local/51666692/ipykernel_23376/3255715277.py:3: MatplotlibDeprecati
onWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated
since 3.5 and will be removed two minor releases later; please call grid(Fal
se) first.
  plt.colorbar()
```

Out[26]:

```
<matplotlib.colorbar.Colorbar at 0x2b7fe8a0c3d0>
```



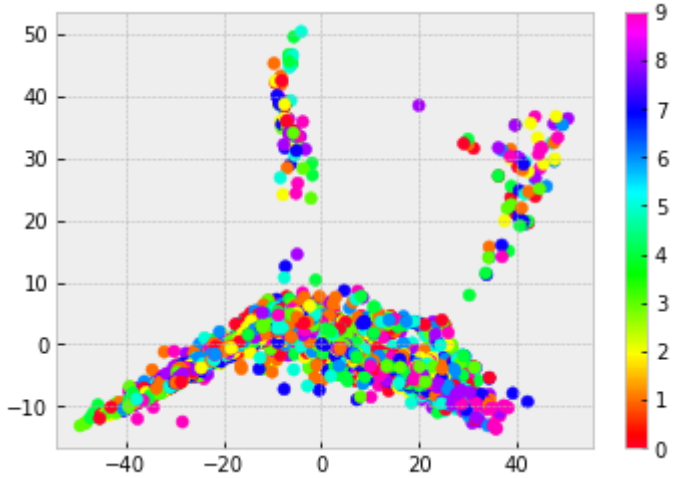
In [29]:

```
# visualize the first 2 dimensions
y_isomap = model_isomap.named_steps['isomap'].embedding_
plt.scatter(y_isomap[:,0], y_isomap[:,1], c=t_train, cmap=plt.cm.gist_rainbow);
plt.colorbar()
```

```
/scratch/local/51666692/ipykernel_23376/3841140412.py:4: MatplotlibDeprecati
onWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated
since 3.5 and will be removed two minor releases later; please call grid(Fal
se) first.
  plt.colorbar()
```

Out[29]:

```
<matplotlib.colorbar.Colorbar at 0x2b7fe8af8f40>
```



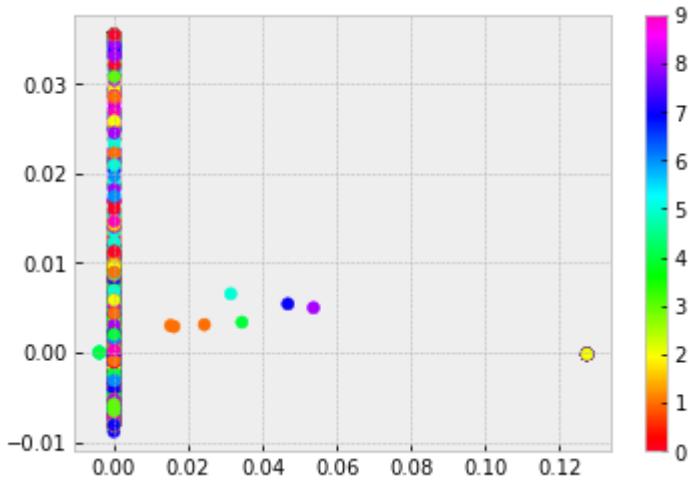
In [30]:

```
# visualize the first 2 dimensions
y_lle = model_lle.named_steps['lle'].embedding_
plt.scatter(y_lle[:,0], y_lle[:,1], c=t_train, cmap=plt.cm.gist_rainbow);
plt.colorbar()
```

```
/scratch/local/51666692/ipykernel_23376/2692775084.py:4: MatplotlibDeprecati
onWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated
since 3.5 and will be removed two minor releases later; please call grid(Fal
se) first.
  plt.colorbar()
```

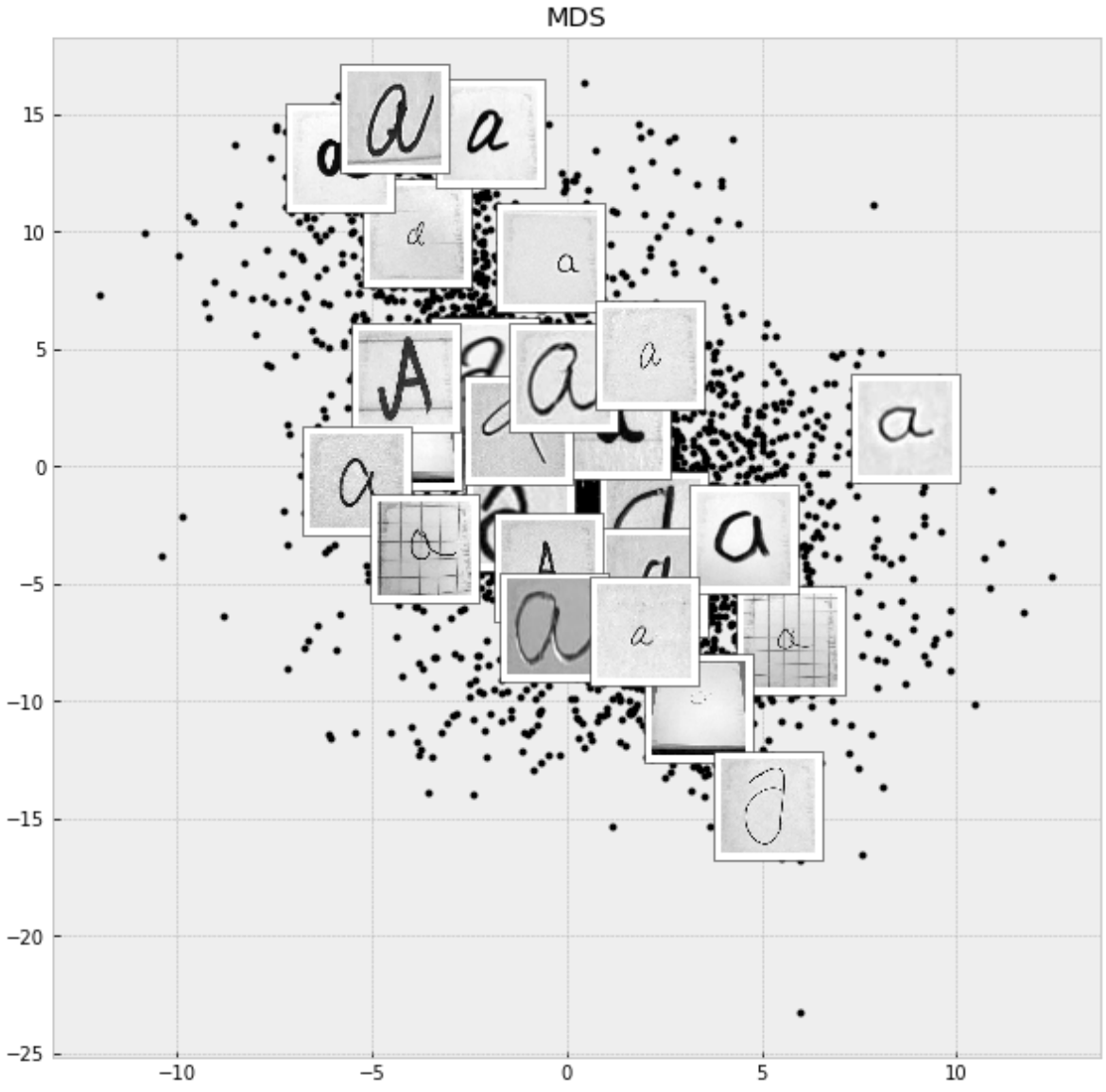
Out[30]:

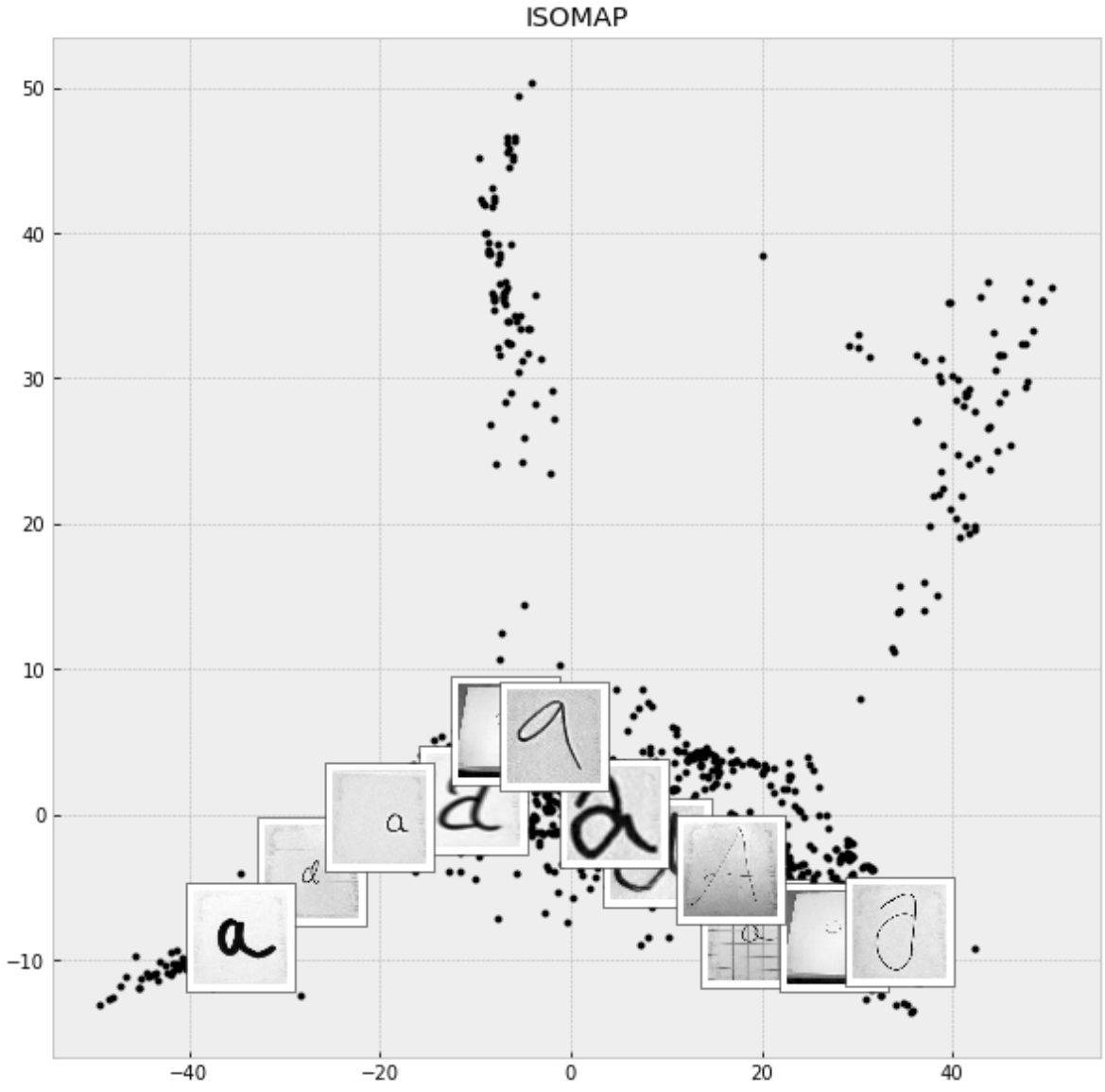
```
<matplotlib.colorbar.Colorbar at 0x2b7fe8902ef0>
```

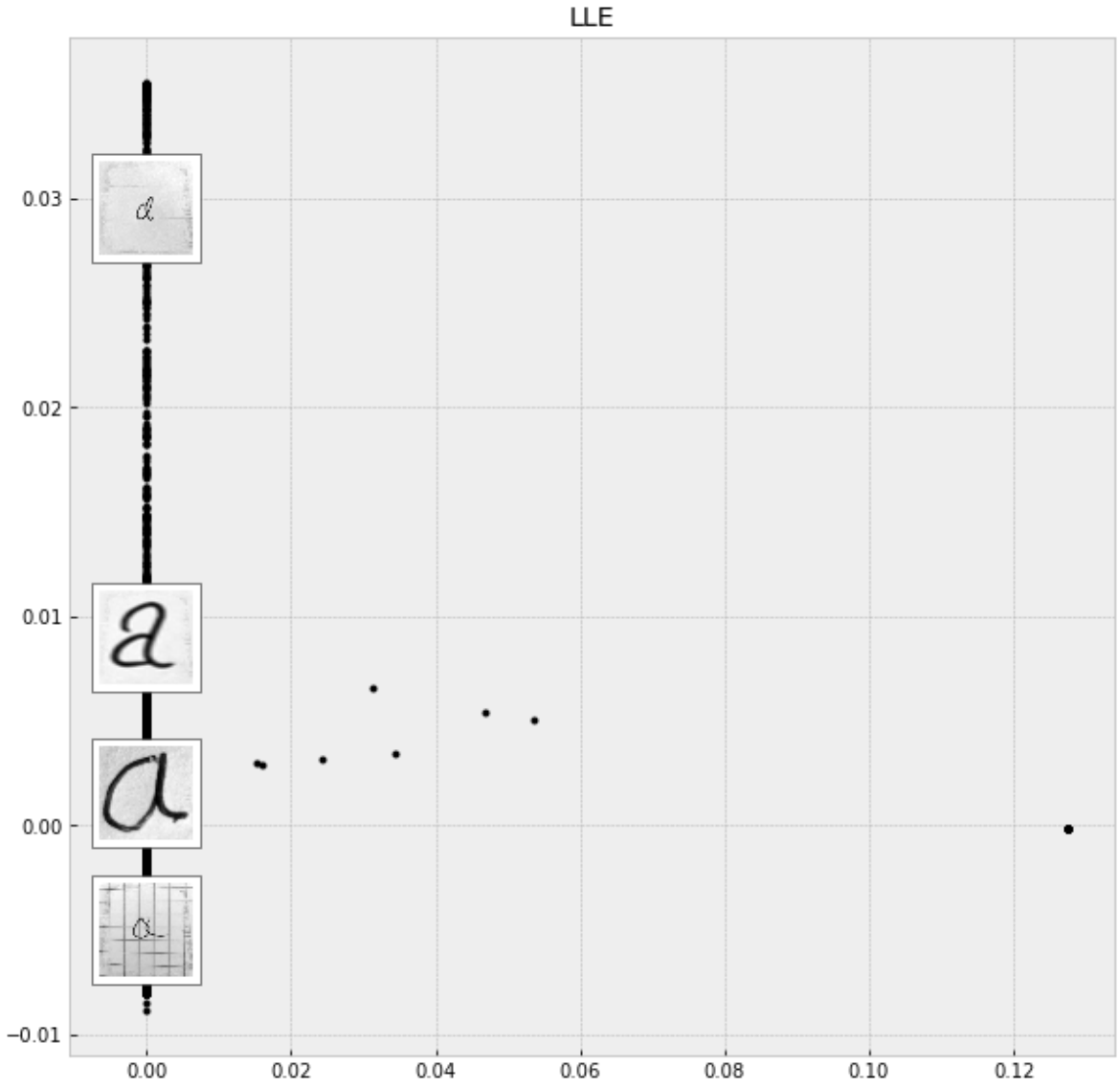


In [32]:

```
from matplotlib import offsetbox
def plot_components(data, proj, images=None, ax=None,
                    thumb_frac=0.05, cmap='gray'):
    ax = ax or plt.gca()

    ax.plot(proj[:, 0], proj[:, 1], '.k')

    if images is not None:
        min_dist_2 = (thumb_frac * max(proj.max(0) - proj.min(0))) ** 2
        shown_images = np.array([2 * proj.max(0)])
        for i in range(data.shape[0]):
            dist = np.sum((proj[i] - shown_images) ** 2, 1)
            if np.min(dist) < min_dist_2:
                # don't show points that are too close
                continue
            shown_images = np.vstack([shown_images, proj[i]])
            imagebox = offsetbox.AnnotationBbox(
                offsetbox.OffsetImage(images[i], cmap=cmap),
                                      proj[i])
            ax.add_artist(imagebox)


digit = 0
data = X_train_res[t_train == digit,:][::10]

# MDS
fig, ax = plt.subplots(figsize=(10, 10))
plot_components(data, y_mds[:,:2], images=data.reshape((-1, 50, 50)),
                ax=ax, thumb_frac=0.05, cmap='gray')
plt.title('MDS')

# ISOMAP
fig, ax = plt.subplots(figsize=(10, 10))
plot_components(data, y_isomap[:,:2], images=data.reshape((-1, 50, 50)),
                ax=ax, thumb_frac=0.05, cmap='gray')
plt.title('ISOMAP');

# LLE
fig, ax = plt.subplots(figsize=(10, 10))
plot_components(data, y_lle[:,:2], images=data.reshape((-1, 50, 50)),
                ax=ax, thumb_frac=0.05, cmap='gray')
plt.title('LLE');
```


MDS
15
10
5
0
−5
−10
−15
−20
−25
−10
−5
0
5
10


ISOMAP
50
40
30
20
10
0
−10
−40
−20
0
20
40



We can see that LLE with number of components 8 only preserves local data structure thus has a bad representation with low accuracy and not able to visually identify the feature it preserve. For MDS, its number of components is 6. We can visually identify that digit in first two dimensions tend to have bigger size in the left and smaller size in the right with thicker edges presented in the diagonal area. However, the use of euclidean distance makes it difficult to correctly unfold the manifold thus lead to a bad accuracy. As for Isomap, its number of components is 17. it makes use of geodesic distance and thus be able to properly unfold the manifold with better representation and higher accuracy. As a result, Isomap is a better manifold learning algorithm in this problem.

In [ ]:

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.
