# AMATH 582 Homework Four: Digit Classification

Daniel W. Crews

March 10, 2021

**Abstract**

A classic problem of image processing is to sort a large amount of data into user-defined categories. This report tests sorting algorithms on a dataset consisting of many handwritten digits using what are called "supervised learning" techniques. These algorithms operate by drawing statistically well-fitting boundaries among labelled points in the space of each image's expansion coefficients onto the proper orthogonal modes of the entire dataset (found via SVD), with methods distinguished by drawing boundaries differently. The support vector machine (SVM), which draws the boundary using supporting vectors between labelled domains, is found to work particularly well for the digit classification problem.

## 1 Introduction and Overview

This work analyzes the MNIST handwritten digit dataset, made up of 60000 training images and 10000 test images each 28×28 pixels, by applying linear classifiers to the principal components of the various images. The classification methods tested consist of i) linear discriminant analysis (LDA), ii) decision tree classification, and iii) the support vector machine (SVM) method. Here, principal components are determined by projection onto the orthogonal eigenmodes of the images obtained by singular value decomposition (SVD) of the training data matrix. While the singular value spectrum of the data matrix has a heavy tail, good classification results are found using only 50 out of the total 784 image modes.

The first section of this report briefly summarizes the SVD theory and that of the linear classifiers, before proceeding into the results of analysis in the second section. Here the various classification methods are compared to one another, in particular on difficult-to-distinguish digit pairs. However, here it's found that the SVM method sorts these with good accuracy. While this analysis utilized the entire training and test dataset and hence had a deterministic outcome, in practical application of these methods cross-validation (repeated classification with a randomly sampled training dataset) is needed to demonstrate robustness.

## 2 Theoretical Background

### 2.1 Singular value decomposition

While the SVD was discussed in the previous homework, it is the central character of this work's analysis and deserves review again. A mean-zero matrix $X \in \mathbb{C}^{m\times n}$ has *two* associated covariance matrices [1]

$$\mathcal{C}_m \equiv \frac{1}{n-1} XX^\dagger, \qquad \mathcal{C}_n \equiv \frac{1}{m-1} X^\dagger X \tag{1}$$

Each component of the two matrices $\mathcal{C}_{n,m}$ expresses the covariance of the data $X$ as a linear operator between the spaces of dimension $m$ and $n$. Now the SVD, as a generalized diagonalization procedure, determines matrices of orthonormal unit vectors $U$ and $V$ and a diagonal matrix of singular values $\Sigma$ such that

$$XV = U\Sigma, \quad V^{-1} = V^\dagger, \quad \implies \quad X = U\Sigma V^\dagger. \tag{2}$$

In the last homework, the projected matrix $Z = U^\dagger X$ was used as a representation of the data in the basis where the covariance $\mathcal{C}_m$ was diagonal. In this homework, for completeness, a projected variable $Y = XV$ is used, as this diagonalizes the covariance matrix $\mathcal{C}_n$. Generally speaking a matrix $X$ is not mean-zero along

both its dimensions, so for covariance diagonalization in either particular direction one should subtract the corresponding mean. In this homework the data matrix is arranged as (images × pixels), so for diagonalized covariance in the pixel space $\mathcal{C}_n$ one should compute the SVD of $X$ minus the mean of each pixel across all images, and then project onto the eigenmodes where $\mathcal{C}_n$ is diagonal as $Y = XV$.

### 2.2 Varieties of linear classifier algorithm

The term classifier refers to a general method to separate data into various categories. A human being with a lifetime of learning experience is very good at this task, yet quickly becomes bored when sorting many thousands of things. For this reason we attempt to have machines perform the task, and as they must learn what belongs where we call it machine learning, even if ultimately what is going on is linear algebra.

There are two categories of learning called unsupervised and supervised, in correspondence with the two types of grad school advisor. In a similar way, there are advantages and disadvantages of each approach. This work studies supervised algorithms which are given pre-categorized training sets to fit to. The first, linear discriminant analysis (LDA), establishes geometric boundaries by minimizing a Rayleigh quotient [1]. The second, the decision tree classifier, develops a logical tree structure to sort provided data according to sophisticated criteria such as information entropy. Finally, the support vector machine (SVM) creates boundary sets of general polytopes whose edges are supported by data vectors in parameter space, depending on their distance from other similar-labeled points, and then fits a curve between these boundary sets.

## 3 Algorithm Implementation and Development

The provided training set is made up of 60000 images each 28×28 pixels. This is shaped into a data matrix X which is 60000×784, and the mean value along each pixel is subtracted. The SVD is then computed, and each image projected into the eigenspace by $Y = XV$. The matrix $Y$ consists of the 784 expansion coefficients of each image, according to the general orthogonal coefficient formula

$$c_n = \frac{\langle x|v_n\rangle}{\langle v_n|v_n\rangle} \tag{3}$$

where here each image $\boldsymbol{v}_n$ is already normalized to one and everything is real-valued. The singular value spectrum, shown in Fig. 1, has a heavy tail yet is generally dominated by its first ∼150 modes. That is, reconstructions $\widetilde{X} = \sum_{n=0}^{N} Y_n = \sum_n c_n v_n$ are converged to the naked eye generally at about 150 terms. However, dimensional reduction to about 50 modes was seen to produce very acceptable results in the classifiers and basically discernable digits, albeit with some small oscillations around sharp image features.

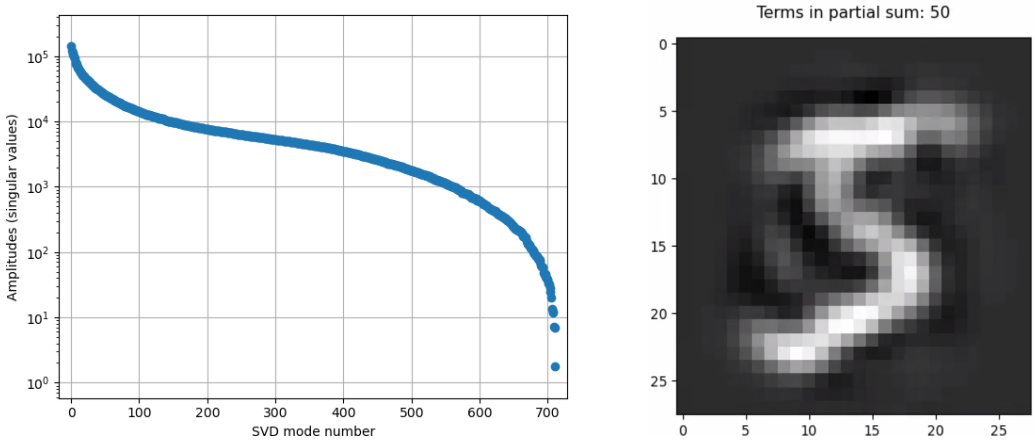


Figure 1: (Left:) Generalized eigenvalues (SVs) of the MNIST training set showing the heavy coefficient tail. A further basically zero spectrum (not shown) exists up to mode 784. (Right:) Reconstruction of the digit "5" truncated at fifty terms in the eigendecomposition. Some oscillations exist, yet the digit is discernable.

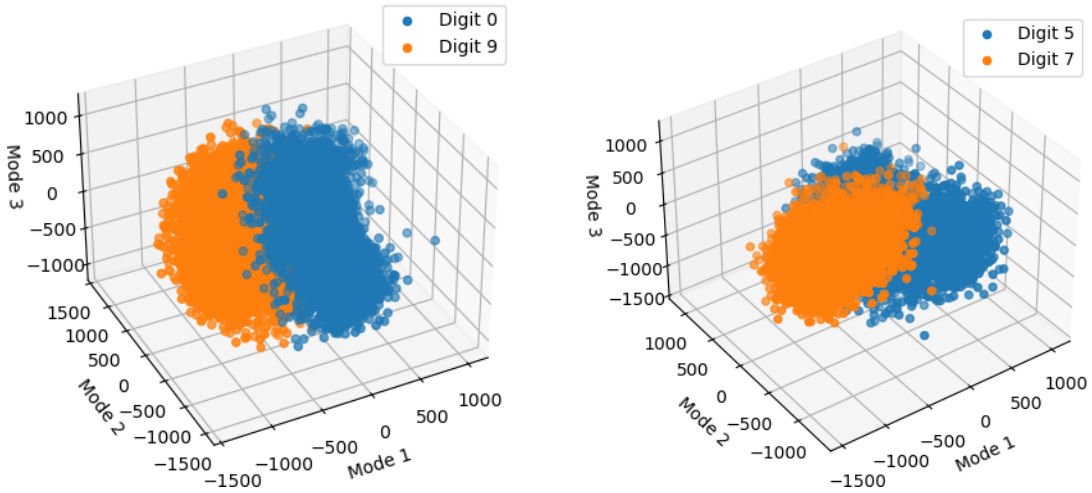


Figure 2: Projections of the eigendecomposition coefficients onto the first three principal modes for training realizations classified as 0 and 9 (left) and 5 and 7 (right). This reveals that, from the perspective of these three modes, digits 0 and 9 are well-separated (particularly in mode 1) while digits 5 and 7 have significant interpenetration in this particular projection. In the higher mode space digits 5 and 7 are better separated.

As seen in Fig. 2, projected onto the first three modes, the expansion coefficients of each image form clusters in the mode space. This geometric property means the classifiers described in Ref. 2.2, which divide the eigenspace into separated groups with boundaries, and predict with these boundaries, will work well for digit classification. Importantly, this geometric separation occurs in the eigenspace, or principal components, of the digit images. Finally, the classifiers were applied in the following manner. First, classification of all pairs of two digits were tested using all three described methods. Second, three-digit classification was tested using LDA only. The following section describes the results obtained with the linear classifiers.

# 4 Computational Results

The first result tested was the LDA on all pairs of digits, shown in Fig. 3. Easily-separated digits are seen to be 0 and 1, and 6 and 7. However, difficult-to-separate digits are 9 and 4, as well as 3 and 7. These easy and difficult to separate pairs are nicely explained by their disparities and similarities in pattern.

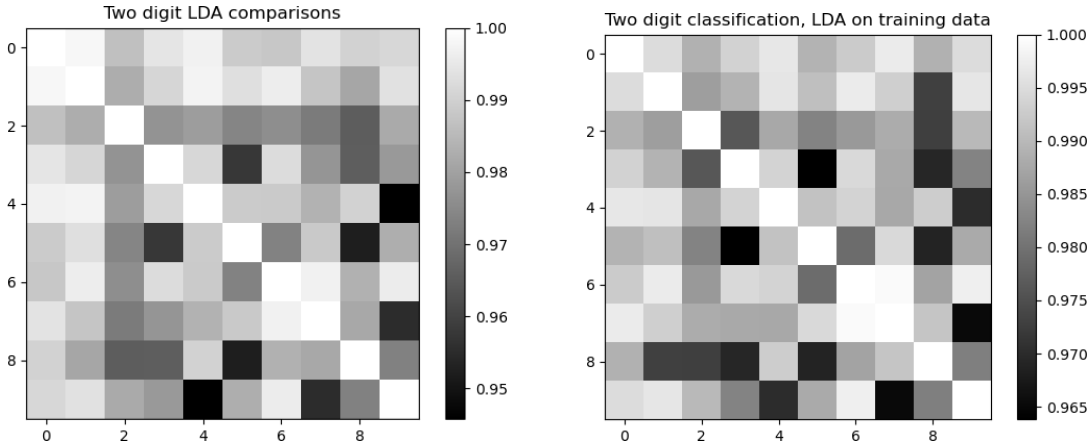


Figure 3: Classification score of LDA method on digit pairs. (Left:) scores of the test data using 50-mode reduced training data. While all scores are $> 94\%$, 2 has a poor trend and certain pairings are not good. (Right:) Classifier score on its own training data without dimension reduction, showing inherent LDA error.

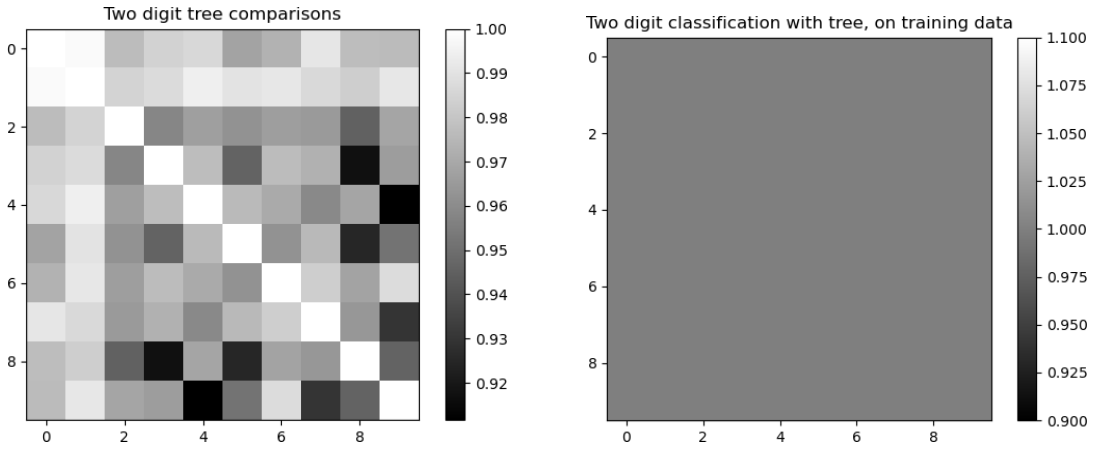


Figure 4: (Left:) Two-digit classification results using the decision tree method. Interestingly, larger errors are observed compared to the LDA, but the digit 2 consistently performs better. Errors on the difficult-to-distinguish pairs of the LDA are exacerbated by the decision tree. (Right:) The tree, at least with default settings, predicts its own training data exactly (so the colorbar was autoset). This indicates overfitting.

Further, predicting the original training data reveals the very same error pattern, though at a lower rate of incidence. This demonstrates an inherent error in the LDA method due to outliers, even though it operates on a minimized Rayleigh quotient. Now Fig. 4 demonstrates results with the decision tree method. The main takeaway from these results is that errors on difficult-to-distinguish data are made even worse by the decision tree compared to the LDA, but it predicts its own training data very well.

Next, the support vector machine was explored with results in Fig. 5. The SVM was observed to be calculated faster than the decision tree *and* to have accurate results to within almost 1%! However, it has a small rate of error in predicting its own training data. As before, the SVM was trained on a 50-mode reduced dimension eigenprojected dataset. Comparison of Figs. 3, 4, and 5 shows that all three methods distinguish 0 and 1 very well, and all have the most difficulty with 4 and 9. However, the SVM classifier predicts with great accuracy and beats the others by several percentage points.

It must be stressed that this analysis is done deterministically on a single set of training and test data. Generally speaking these methods are to be cross-validated by randomly sampling a subset of the training data and doing a statistical analysis of the resulting scores for different training set sizes. The author played around with this a bit while doing the assignment but did not include these results for brevity. The general trend was that scores were lower when using a smaller training set, but the SVM still did the best.

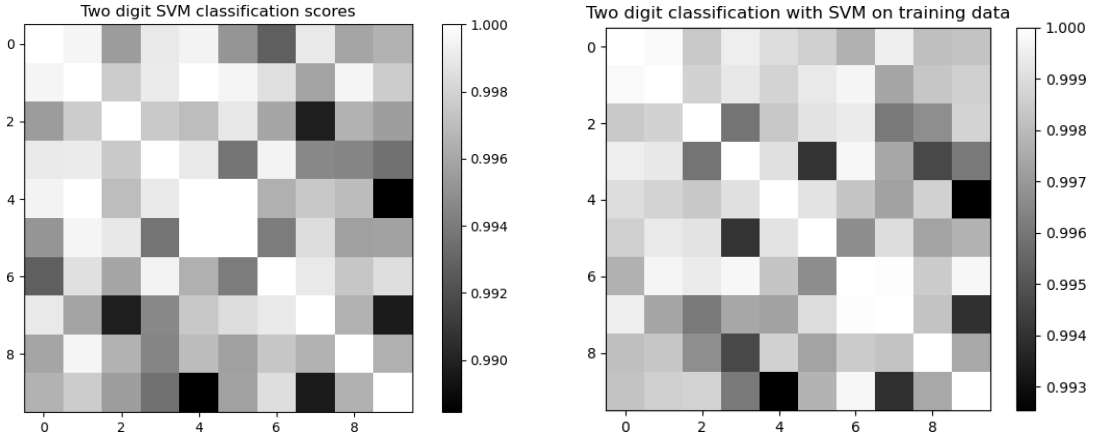


Figure 5: Two-digit classification results with the SVM method. The support vector technique is seen to be the best of the three methods overall, and intriguingly has almost perfect discernment between digits 4 and 5. It stumbles most on 4 and 9, however perhaps even a human would on these digits when handwritten.

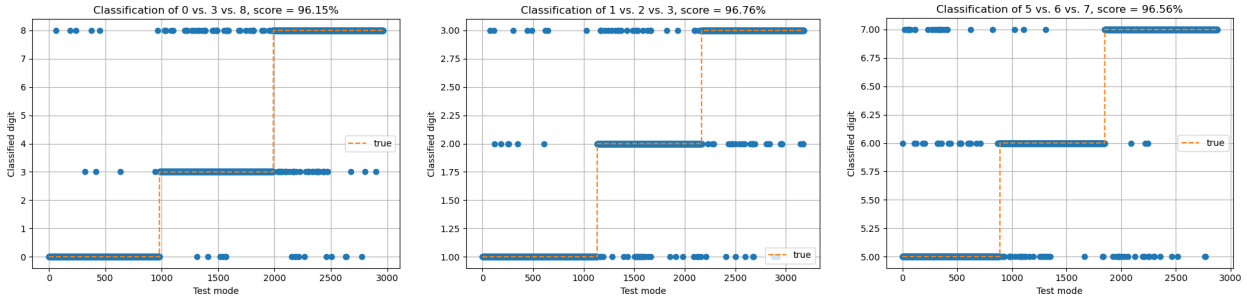


Figure 6: Simultaneous classification of three digits using LDA on the sets of digits 038, 123, and 567. Generally good classification in line with the "worst" results of the two-digit LDA is observed, here $> 96\%$.

### 4.1 Three-digit classification

As an initial exploration of further capabilities, Fig. 6 shows LDA applied to categorize three digits simultaneously. Good results are seen in these cases, with general trends in line with the results of the two-digit LDA in Fig. 3. However, one would expect that simultaneous classification of all nine digits would run into issues due to the general mixing of eigenvalues for the difficult pairs such as 9 and 4, contaminating performance of the entire classification scheme. Three digits was generally seen to work well though.

## 5 Summary and Conclusions

This numerical experiment examined three types of linear classifiers: linear discriminant analysis, decision tree, and support vector machine. The decision tree was observed to overfit the data, however tweaking its settings would probably reduce that effect. An initial dive into this by constraining the number of leaf nodes showed a relaxation of the overfitting, but not really an improvement in its scores.

Principal component analysis was used to formulate a well-separated parameter space in terms of eigendecomposition coefficients and dimensionality reduction was experimented with, using truncated characteristic mode expansions. The support vector machine was seen to be the very best method of those examined here. As usual, Python was used for all implementations in this exercise and may be found at the author's Github page here along with some cool movies of the eigenexpansions converging.

## References

[1] J.N. Kutz. *Data-driven Modeling and Scientific Computation.* Oxford University Press, 1st edition, 2013.

## Appendix A Python Functions

The following list compiles important Python functions used in implementation:

- The all-important masking capability `array[bool condition] = x` is used to identify labelled data,
- The `sklearn` package was used for classifier capabilities, including the subroutines
  - `sklearn.discriminant_analysis` with class `LinearDiscriminantAnalysis`,
  - `sklearn.tree` with class `DecisionTreeClassifier`,
  - `sklearn` with class `svm` (for SVM).
- For each classifier package, one trains using `model.fit(data, labels)` and predicts using `P = model.predict(test)`.

## Appendix B Python Implementation

Main implementation file `hw4.py`:

```
# For database
from keras.datasets import mnist
# General packages
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d import Axes3D

### Packages for classifiers
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm

#########################
######## Main script ##
#########################
### Load training and test data (x), labels (y)
(train_x, train_y), (test_x, test_y) = mnist.load_data()

### Reduction by random sampling
# Training: Reduce down training samples by random permutation
# shuffled_idx = np.random.permutation(np.arange(train_y.shape[0]))[:60000]
# train_x = train_x[shuffled_idx, :, :]
# train_y = train_y[shuffled_idx]

# # Test: Reduce down test samples by random perm.
# shuffled_idx = np.random.permutation(np.arange(test_y.shape[0]))[:10000]
# test_x = test_x[shuffled_idx, :, :]
# test_y = test_y[shuffled_idx]

### Reshape to 2D array "data matrix"
X = train_x.reshape(train_x.shape[0], 28*28)
# Subtract row-wise mean
Xm = X - X.mean(axis=1, keepdims=True)

# Do SVD
u, s, vh = np.linalg.svd(Xm, full_matrices=False)

print(u.shape)
print(s.shape)
print(vh.shape)

### Project
Yc = np.dot(Xm, vh.T) # PC projection / expansion coefficients

### Two and three digit helper functions
def two_dgt_prep(n, m):
    # Training data
    idx_n = np.where(train_y == n)
    idx_m = np.where(train_y == m)
```

```
    # Data for digits n,m
    Ynm = np.concatenate([Yc[idx_n,:][0,:,:], Yc[idx_m,:][0,:,:]])
    ynm = np.concatenate([train_y[idx_n], train_y[idx_m]])

    # Dimension reduction
    cutoff = 50#Yc.shape[1]#300 # modal cutoff (convergence condition)
    Ynm = Ynm[:,:cutoff]

    # Test data
    idx_nt = np.where(test_y == n)
    idx_mt = np.where(test_y == m)

    xt = test_x.reshape(test_x.shape[0], 28*28)

    xt_nm = np.concatenate([xt[idx_nt,:][0,:,:], xt[idx_mt,:][0,:,:]])
    yt_nm = np.concatenate([test_y[idx_nt], test_y[idx_mt]])

    Xt = xt_nm - xt_nm.mean(axis=1, keepdims=True)

    Yct = np.dot(Xt, vh.T) # expansion coefficients
    Yct = Yct[:,:cutoff] # dimension reduction

    return Ynm, ynm, Yct, yt_nm
### Make awesome partial sum movies
# plt.figure()
# plt.imshow(Xm[25,:].reshape(28,28))
# plt.show()

# ims = []
# fig, ax = plt.subplots()
# for j in range(200):
#     x = np.zeros((28*28))
#     for i in range(j):
#         x += Yc[25,i]*vh[i,:]#Y[25,i]*vh[i,:]/(np.dot(vh[i,:], vh[i,:].T))

#     # Plot eigendecomposition
#     im = ax.imshow(x.reshape(28,28), cmap='gray', animated=True)
#     #im.set_title('With ' + str(j) + ' modes')
#     title = ax.text(0.5, 1.05, "Terms in partial sum: {}".format(j), size=plt.rcParams["a
#                     ha='center', transform=ax.transAxes, )

#     ims.append([im, title])
#     #plt.pause(0.1)

# an = animation.ArtistAnimation(fig, ims, interval=50, blit=True, repeat_delay=1000)
# an.save('partial_sums_test.mp4', fps=10)

# quit()

### Spectrum plot
# plt.figure()
# plt.semilogy(s, 'o')
# plt.grid(True)
# plt.xlabel('SVD mode number')
```

```
# plt.ylabel('Amplitudes (singular values)')
# plt.tight_layout()

### Do 3D plot
# fig = plt.figure()
# ax = fig.add_subplot(111, projection='3d')
# for j in [3, 8]:#range(9):
#     idx = np.where(train_y == j)
#     #ax.scatter(Yc[idx, 1], Yc[idx, 2], Yc[idx, 3], 'o', label='Digit ' + str(j))
#     ax.scatter(u[idx, 1], u[idx, 2], u[idx, 3], 'o', label='Digit ' + str(j))
# ax.set_xlabel('Mode 1')
# ax.set_ylabel('Mode 2')
# ax.set_zlabel('Mode 3')
# plt.legend()#loc='best')

# plt.show()

### Classifiers for digit identification
######################################################################
############ Part One: LDA ###########################################
######################################################################
# model = LinearDiscriminantAnalysis()

# # Two digit classification
# score = np.ones((10, 10))
# for i in range(10):
#     for j in range(10):
#         if j != i:
#             n = i
#             m = j

#             Ynm, ynm, Yct, yt_nm = two_dgt_prep(n, m)
#             # Fit model
#             model.fit(Ynm, ynm) # expansion coefficients and labels

#             # On test data
#             #prediction = model.predict(Yct)
#             #right = np.where(prediction == yt_nm)
#             #score[i,j] = yt_nm[right].shape[0]/yt_nm.shape[0]

#             # On training data
#             prediction = model.predict(Ynm)
#             right = np.where(prediction == ynm)
#             score[i,j] = ynm[right].shape[0]/ynm.shape[0]

# #             # Check out
# #             # plt.figure()
# #             # plt.plot(prediction, 'o')
# #             # plt.plot(yt_nm, '--', label='true')
# #             # plt.legend(loc='best')
# #             # plt.xlabel('Test mode')
# #             # plt.ylabel('Classified digit')
# #             # plt.title('Classification of ' + str(n) + ' vs. ' + str(m) + ', score = 
# #             # plt.grid(True)
```

```
# #             # plt.tight_layout()
# #             # #plt.savefig('2digit_lda_' + str(n) + str(m) + '.png')
# #             # plt.show()

# plt.figure()
# plt.imshow(score, cmap='gray')
# plt.colorbar()
# plt.title('Two digit classification, LDA on training data')
# plt.show()

################################################ Three digit LDA classification
# # Obtain labels for digits n,m,l
# n = 5
# m = 6
# l = 7
# idx_n = np.where(train_y == n)
# idx_m = np.where(train_y == m)
# idx_l = np.where(train_y == l)
# # Data for digits n,m
# Ynml = np.concatenate([Yc[idx_n,:][0,:,:], Yc[idx_m,:][0,:,:], Yc[idx_l,:][0,:,:]])
# ynml = np.concatenate([train_y[idx_n], train_y[idx_m], train_y[idx_l]])
# # Fit model
# model.fit(Ynml, ynml) # PCs and labels

# # Try it out
# idx_nt = np.where(test_y == n)
# idx_mt = np.where(test_y == m)
# idx_lt = np.where(test_y == l)

# xt = test_x.reshape(test_x.shape[0], 28*28)

# xt_nml = np.concatenate([xt[idx_nt,:][0,:,:], xt[idx_mt,:][0,:,:], xt[idx_lt,:][0,:,:]])
# yt_nml = np.concatenate([test_y[idx_nt], test_y[idx_mt], test_y[idx_lt]])

# Xt = xt_nml - xt_nml.mean(axis=1, keepdims=True)

# Yct = np.dot(Xt, vh.T)

# prediction = model.predict(Yct)

# right = np.where(prediction == yt_nml)
# score = yt_nml[right].shape[0]/yt_nml.shape[0]

# plt.figure()
# plt.plot(prediction, 'o')
# plt.plot(yt_nml, '--', label='true')
# plt.legend(loc='best')
# plt.xlabel('Test mode')
# plt.ylabel('Classified digit')
# plt.title('Classification of ' + str(n) + ' vs. ' + str(m) + ' vs. ' + str(l) + ', score
# plt.grid(True)
# plt.tight_layout()
# plt.savefig('3digit_lda_' + str(n) + str(m) + str(l) + '.png')
# plt.show()
```

```
##############################################################################
########################## Decision tree analysis ############################
##############################################################################
tree = DecisionTreeClassifier(random_state=0)#, max_leaf_nodes = 4)#random_state=0)

score = np.ones((10,10))
for i in range(10):
    for j in range(10):
        if i != j:
            n = i
            m = j
            Ynm, ynm, Yct, yt_nm = two_dgt_prep(n, m)
            # fit tree
            tree.fit(Ynm, ynm)

            # Predict (test data)
            #prediction = tree.predict(Yct)
            #right = np.where(prediction == yt_nm)
            #score[i,j] = yt_nm[right].shape[0]/yt_nm.shape[0]

            # Predict (training data)
            prediction = tree.predict(Ynm)
            right = np.where(prediction == ynm)
            score[i,j] = ynm[right].shape[0]/ynm.shape[0]

plt.figure()
plt.imshow(score, cmap='gray')
plt.colorbar()
plt.title('Two digit classification with tree, on training data')
plt.show()


# plt.figure()
# plt.plot(prediction, 'o')
# plt.plot(yt_nm, '--', label='true')
# plt.legend(loc='best')
# plt.xlabel('Test mode')
# plt.ylabel('Classified digit')
# plt.title('Tree classification of ' + str(n) + ' vs. ' + str(m) + ', score = %3.2f'%(scor
# plt.grid(True)
# plt.tight_layout()
# plt.savefig('2digit_tree_' + str(n) + str(m) + '.png')
# plt.show()

##############################################################################
######################### Support vector classifier ##########################
##############################################################################
# vector_machine = svm.SVC()

# score = np.ones((10,10))

# for i in range(10):
```

```
#     for j in range(10):
#         if i != j:
#             n = i
#             m = j
#             Ynm, ynm, Yct, yt_nm = two_dgt_prep(n, m)
#             vector_machine.fit(Ynm, ynm)

#             # on test data
#             #prediction = vector_machine.predict(Yct)
#             #right = np.where(prediction == yt_nm)
#             #score[i,j] = yt_nm[right].shape[0]/yt_nm.shape[0]

#             # on training data
#             prediction = vector_machine.predict(Ynm)
#             right = np.where(prediction == ynm)
#             score[i,j] = ynm[right].shape[0]/ynm.shape[0]


# plt.figure()
# plt.imshow(score, cmap='gray')
# plt.colorbar()
# plt.title('Two digit classification with SVM on training data')
# plt.show()
```