AMATH 482 Homework 4

Gg Tran

March 8, 2019

Abstract

We perform rank reduction using Singular Value Decomposition on two data of cropped and uncropped human faces. We reconstruct the images using the rank-reduced data. The cropped data requires much less modes to reconstruct the proper images, as opposes to the uncropped data which requires ten times more modes to achieve the goal.

We perform Naive Bayes, a supervised learning algorithm, on several datasets of music. The goal is to build a model that accurately identify the category of a given song. However, our model only manages to yield modest accuracy rate (~63%).

I. Introduction and Overview

Machine learning seeks to accomplish two goals: Reduce the rank of the data and make predictions. The Singular Value Decomposition (SVD) is an efficient method for dimensionality reduction. Subsequently, machine learning (ML) algorithms try to fit the best model for a given dataset by clustering and classifying different subsets. [1] To perform ML, ones split the data set into training set (used to build the model) and test set (used to check the predictive accuracy of the model). Two broad categories of machine learning are supervised learning and unsupervised learning. In supervised learning, the user provides class labels for the training set. In unsupervised learning, the user provides no labels, and the algorithm needs to find patterns that distinguise between different subsets of the data. One widely used supervised learning algorithm is Naive Bayes.

II. Theoretical Background

SVD

The SVD guarantees that one can always the perform the SVD on any given matrix $\mathbf{A}$. The SVD takes the form of

$$\mathbf{A} = \mathbf{U\Sigma V}^*$$

where $\mathbf{U} \in \mathbb{C}^{m\times m}$ and $\mathbf{V} \in \mathbb{C}^{n\times n}$ are unitary matrices. $\mathbf{V}^*$ is the conjugate transpose of $\mathbf{V}$. $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is a diagonal matrix. [1]

Naive Bayes

Naive Bayes represents each category as a probability distribution. Naive Bayes assumes that each feature (variable) of the measurements is normally distributed. Take a measurement E that is described by a series of feature values $(x_1, x_2, \cdots, x_n)$. Let c represents a class. Given that E is described as the feature values x_j, according to Bayes' Theorem, the probability of E belong to class c is $p(c|E) = \frac{p(E|c)p(c)}{p(E)}$. [2]

Naive Bayes assumes conditional independence of the features x_j: Given that E belong to a class c, $p(E|c) = p(x_1, x_2, \cdots, x_n|c) = \prod_{i=1}^{n} p(x_i|c)$. [2]

Thus, the Naive Bayes classifier function is given by $f(E) = p(c) \prod_{i=1}^{n} p(x_i|c)$. [2]

III. Algorithm Implementation and Development

Yale faces

- Load and read all images into matrices **all_ImagesCropped** and **all_ImagesUncropped**.
- Perform SVD two get the plot of the singular values of the two data. SVD gives us two orthonormal bases **u** and **v** of the original data, and the diagonal matrix **s** which contains the singular values. Plotting the singular values can give help us approximate how many modes are needed to recreate the original data, thus reducing the amount of features of the images.
- Reconstruct the rank-reduced versions of the original: **truncatedCropped** and **truncatedUncropped**

Music classification

- Split the songs of each category (e.g band, genre) into several 5-second songs. We used an external software called *WavePad Sound Editor* to split up the songs into several .wav files, with length of 5 seconds.
- Shuffle all the song files randomly. This randomization is to prevent any chance of the ordering of the song files exerting any meaningful effects on the algorithm's accuracy.
- Chose the number of songs used for each category: **samples = 200**. Total samples across all categories were 600 samples.
- Build the data matrix **data**: > Each column of the data matrix corresponds to a 5-second song. > Perform Fast Fourier Transform on each column vector (song) to obtain the song's frequency (i.e the "fingerprints" of the song).
- Reduce the number of features (variables) of the measurements using SVD.
- Classification: > Split the data **v** into training set (70% of the data) and test set (30%). > Train the data using Naive's Bayes. > Perform the "prediction" on the test set.
- Calculate accuracy rate and class error using k-fold.

IV. Computational Results

Yale faces

With the cropped images: There are about 10 singular values clearly lined up on an axis (Figure 1). Using only 10 modes, we can create a pretty good reconstruction of the original faces (Figure 2). With the uncropped images: The singular values are more "spreaded out". It takes 100 modes to sufficiently recreate the faces.

Music classification

Across all three tests, Naive Bayes's prediction has low to moderate accuracy rates, and moderately high class error (table 1).

	Class Error	Accuracy
Test 1	36.67	63.33
Test 2	54.44	45.56
Test 3	39.44	60.56

Table 1: Distribution of class error and prediction accuracy

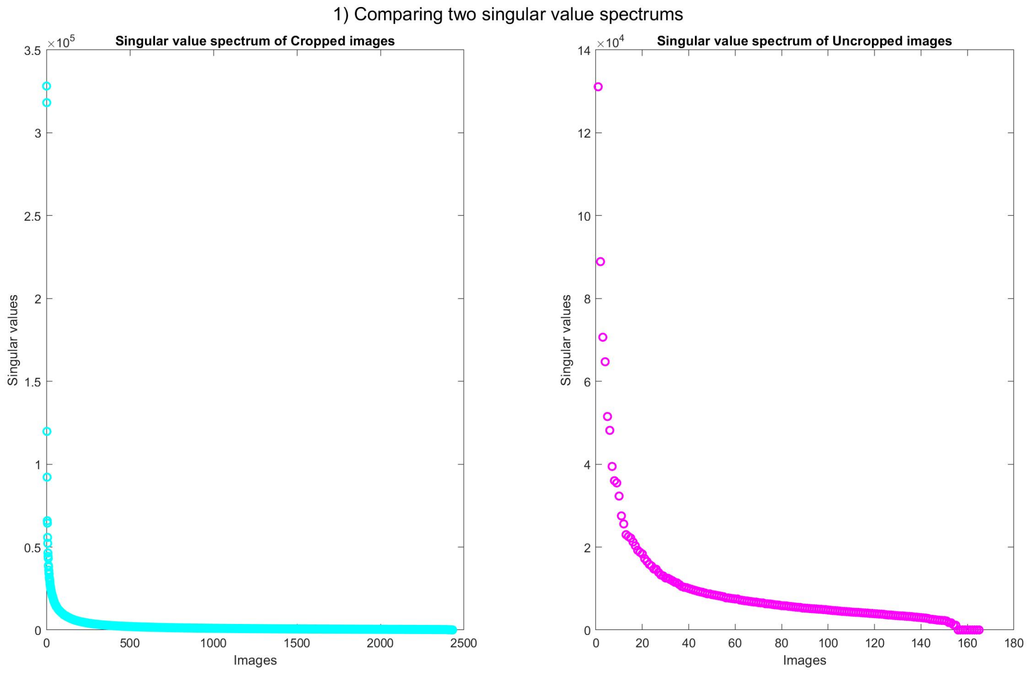

Figure 1: Singular value spectrums of the cropped and uncropped images. For the cropped images, the first ~10 singular values line up nicely on a line.

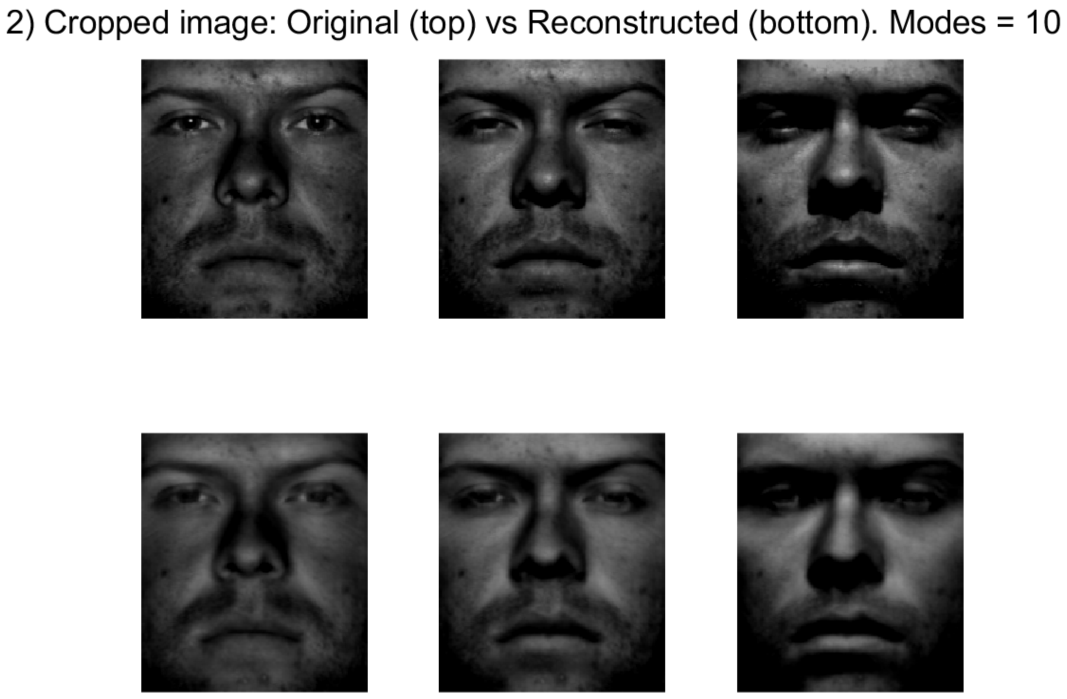
2) Cropped image: Original (top) vs Reconstructed (bottom). Modes = 10

Figure 2: Comparing the original cropped images (top) versus the reconstructed images (bottom). The first 10 modes already give nice image reconstructions, with only slight blurriness.

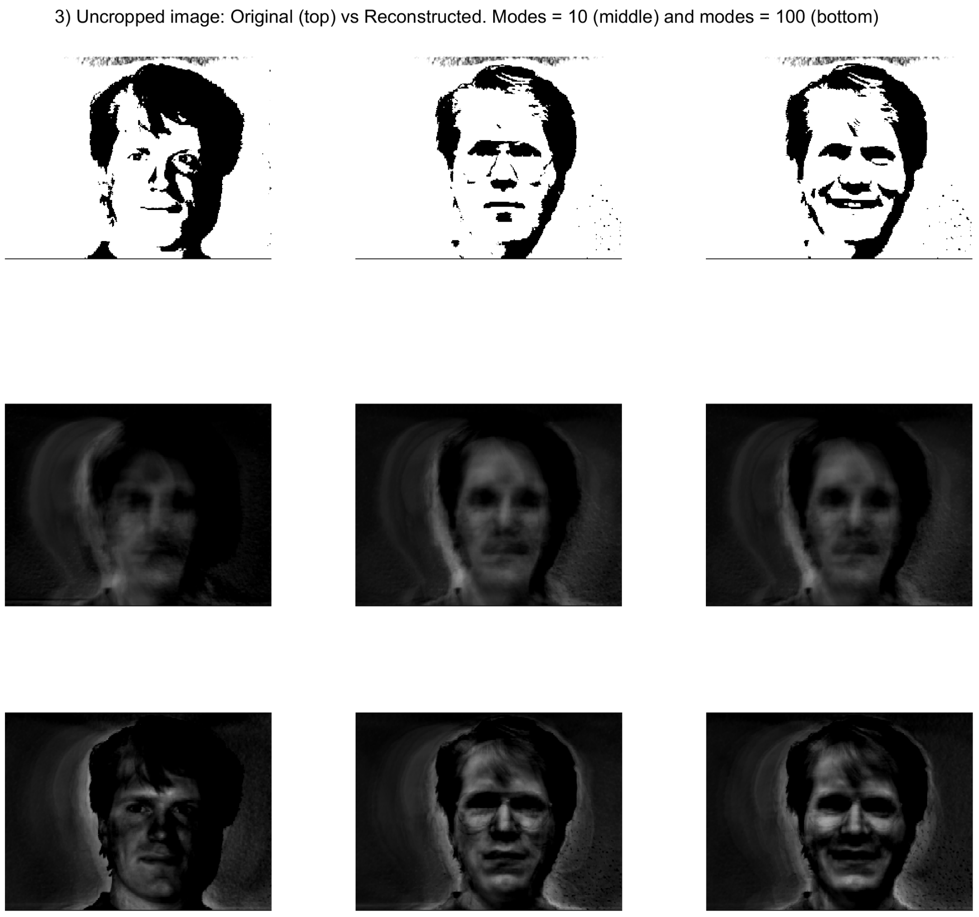
3) Uncropped image: Original (top) vs Reconstructed. Modes = 10 (middle) and modes = 100 (bottom)

Figure 3: Comparing the original uncropped images (top) versus the reconstructed images (middle and bottom). The uncropped images require 100 modes to sufficiently reconstruct the original images. When only 10 modes are used, the results are very blurry.

Summary and Conclusions

For the face datasets, cropped images (zoom in faces with no backgrounds) need much lower modes (10) to sufficiently recreate the original faces. However, uncropped images (include faces and backgrounds) need much higher modes (100) to sufficiently recreate the contour of the faces.

All the songs in the dataset are either non-English (Viet ballad/Viet rap, Korean pop) or nonvocal (instrumental, trap & bass). Across all three tests, Naive Bayes can only achieve 45% - 65% accuracy. The accuracy of Naive Bayes is the lowest in test 2. This is to be expected since it is harder to distinguish songs of the same genre. Interestingly, across all 3 tests, the accuracy rate without randomization tends to be much higher than the one with randomization (65% - 79% accuracy without randomization, a near 20% jump).

Considering that the Naive Bayes has been proven to have high accuracy, we surmise that our low to moderate accuracy is due to our data making. Perhaps cutting up songs into 5-second chunks before processing and sampling them in MATLAB is a terrible idea.

Appendix A MATLAB functions used

[u, s, v] = svd(): Perform Singular Value Decompose on the dataset.

randperm(): Randomly shuffle the data files.

fft(): Fast Fourier Transform to get the song's frequency.

reshape(): Reshape vector into matrix.

fitcnb(): Use Naive Bayes to train the data.

kfoldLoss(model): calculate class error produced by Naive Bayes model.

Appendix B MATLAB codes

Yales faces

```
clear all; close all; clc

homedir = [pwd filesep 'CroppedYale'];
cd(homedir);
imageFolders = dir();
imageFolders = {imageFolders.name};
imageFolders = imageFolders(3:end);  %List of yaleB folders
m = 192;
n = 168;
average = zeros(m*n, 1);
all_Images = [];
%Get into each folder and get the list of the image files in each folder
for folders = 1:length(imageFolders)
    fileLocation = [homedir filesep imageFolders{folders}];
    cd(fileLocation)
    imageFiles = dir();
    imageFiles = {imageFiles.name};
    imageFiles = imageFiles(3:end);
    for nImages = 1:length(imageFiles)
        %each image is a column vector.
        image = double(imread(imageFiles{nImages}));
        image = image(:);
        average = average + image;
        all_Images = [all_Images image];
    end
end
average = average / (nImages * folders);
averageCropped = average;
cd('C:\Users\Gg\Google Drive\AMATH482\HW4')

% save('all_ImagesCropped.mat', 'all_Images')
save('averageCropped.mat', 'averageCropped')

nImages = 64*38;
for j = 1:nImages
    all_Images(:, j) = all_Images(:, j) - average;
end
all_ImagesCropped = all_Images;
save('all_ImagesCropped.mat', 'all_ImagesCropped')


%UNCROPPED
```

```
homedir = [pwd filesep 'yalefaces_uncropped'];
cd(homedir);
imageFiles = dir();
imageFiles = {imageFiles.name};
imageFiles = imageFiles(3:end);
m = 243;
n = 320;
averageUncropped = zeros(m*n,1);
all_ImagesUncropped = [];

 for nImages = 1:length(imageFiles)
     %each image is a column vector.
      image = imread(imageFiles{nImages});
       imshow(image)
      image = double(image(:));
      averageUncropped = averageUncropped + image;
      all_ImagesUncropped = [all_ImagesUncropped image];
 end
averageUncropped = averageUncropped / length(imageFiles);

for j = 1:length(imageFiles)
    all_ImagesUncropped(:, j) = all_ImagesUncropped(:, j) - averageUncropped;
end
cd('C:\Users\Gg\Google Drive\AMATH482\HW4')

 % Perform SVD:
load('all_ImagesCropped.mat'); load('all_ImagesUncropped.mat')
[u1, s1, v1] = svd(all_ImagesCropped, 'econ');
save('svdCropped.mat', 'u1', 's1', 'v1');
[u2, s2, v2] = svd(all_ImagesUncropped, 'econ');
save('svdUncropped.mat', 'u2', 's2', 'v2');
load('svdCropped.mat'); load('svdUncropped.mat')
sCropped = diag(s1);
figure
subplot(1, 2, 1), plot(sCropped, 'ko','Linewidth', [1.5], 'color','cyan')
title('Singular value spectrum of Cropped images'),
ylabel('Singular values'); xlabel('Images')
sUnCropped = diag(s2);
 subplot(1, 2, 2), plot(sUnCropped, 'ko','Linewidth', [1.5], 'color','magenta')
 title('Singular value spectrum of Uncropped images')
 ylabel('Singular values'); xlabel('Images')
 sgtitle('1) Comparing two singular value spectrums')


energyCropped = sum(sCropped(1:1500))/sum(sCropped);
r = 10; % new truncated rank
Truncated data matrix
for j = 1:100   %u(:,1:j)*s(1:j,1:j)*v(:,1:j)'
    truncatedCropped =  u1(:,1:j)*s1(1:j, 1:j)*v1(:,1:j)';
end
%Reconstruct the first 10 images
%Compare the 5 recostructed images with the original images
figure(2),
for k = 1:3
    vect = truncatedCropped(:, k);
    vect = reshape(vect, m, n);
    vect = uint8(double(vect));
    vect2 = all_ImagesCropped(:, k);
    vect2 = uint8(double(reshape(vect2, m, n)));
    subplot(2, 3, k), imshow(vect2)
    subplot(2, 3, 3+k), imshow(vect)
end
sgtitle('2) Cropped image: Original (top) vs Reconstructed (bottom). Modes = 10')


for j = 1:10
    truncatedUnCropped =  u2(:, 1:j)*s2(1:j, 1:j)*v2(:, 1:j)';
end

figure(3),
for k = 1:3
    vect = truncatedUnCropped(:, k);
    vect = reshape(vect, 243, 320);
    vect = uint8(double(vect));
    vect2 = all_ImagesUncropped(:, k);
    vect2 = reshape(vect2, 243, 320);
    subplot(2, 3, k), imshow(vect2)
    subplot(2, 3, 3+k), imshow(vect)
end
sgtitle('3) Uncropped image: Original (top) vs Reconstructed. Modes = 10 (middle) and modes = 100 (bott
for j = 1:100
    truncatedUnCropped =  u2(:, 1:j)*s2(1:j, 1:j)*v2(:, 1:j)';
end
figure(3),
for k = 1:3
    vect = truncatedUnCropped(:, k);
    vect = reshape(vect, 243, 320);
    vect = uint8(double(vect));
    subplot(1, 3, k), imshow(vect)
end
```

Music classification

```
clear all; close all; clc
%%
home = pwd;
dataDir = [home filesep 'artists'];
cd(dataDir);

musicFolders = dir();
musicFolders = {musicFolders.name};
musicFolders = musicFolders(3:end);
 samples = 200;
% %   blackPink = [];
% %  Binz = [];
%   Ghibli = [];
```

```
data = [];
for folders = 1:length(musicFolders)
    fileLocation = [dataDir filesep musicFolders{folders}];
    cd(fileLocation)
    musicFiles = dir();
    musicFiles = {musicFiles.name};
    musicFiles = musicFiles(3:end);
    musicFiles = musicFiles(randperm(length(musicFiles)));
    for nmusics = 1:samples
        %each music is a column vector.
        music = audioread(musicFiles{nmusics});
        music = music(:);
        musicFT = fft(music);
        musicFT = abs(fftshift(musicFT));
%          p9 = audioplayer(music, Fs); playblocking(p9);
%          Binz = [Binz musicFT];
%         blackPink = [blackPink musicFT];
        data = [data musicFT];
    end
end
cd(home)

%%
samples = 200;
cd(pwd);
load('data.mat');
%  [u, s, v] = svd(data - mean(data(:)), 'econ');
% %row V is measurements, col V is variables
%  save('v.mat', 'v')
load('v.mat');
% trainSet = v(1:300, :);
% testSet = v(301:end, :);
 rng(1);
 %create music labeling for the music data
class = {'Vrap', 'Kpop', 'Piano'}';
labeling = repmat(class,1, samples)';
labeling = labeling(:);

 model = fitcnb(v, labeling, 'Holdout', 0.30, ...
    'ClassNames', {'Vrap', 'Kpop', 'Piano'});
% save('model.mat', 'model')
% load('model.mat')
compactModel = model.Trained{1};
testIdx = test(model.Partition); %extract the index of the test data
testData = v(testIdx, :);  %build the test data
testDataTrueLabel = labeling(testIdx); %the true labels for the test data
prediction = predict(compactModel, testData);
%Calculate percent of correctness
correctness = 0;
incorrectIndex = [];
for k = 1:length(prediction)
    if (strcmp(testDataTrueLabel{k}, prediction{k})) == 1
        correctness = correctness + 1;
    else
        incorrectIndex = [incorrectIndex; k];
    end
end
percentCorrect = correctness/length(prediction);
comparison = [testDataTrueLabel(:), prediction(:)];  %True label vs predicted label
%Show side-by-side true label vs inaccurate prediction
check = [];
for k = 1:length(comparison)
    if strcmp(comparison{k, 1}, comparison{k, 2}) == 0
        check = [check; [comparison(k, 1) comparison(k,2)]];
    end
end
classError = kfoldLoss(model);
```

Reference

1. Kutz, Nathan J. “Computational Methods for Data Analysis.” *In Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data.* Oxford University Press, 2013.
2. Zhang, Harry. “The optimality of naive Bayes.” *AA* 1, no. 2 (2004): 3.