CS156 Machine Learning Fashionista

March 13, 2020

1 Machine Learning Fashionista

[1]:
```python
#Import relevant libraries
from PIL import Image
import numpy as np
from sklearn.model_selection import train_test_split
from random import shuffle, seed
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn import svm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from glob import glob
```

1.1 Data Loading and Processing

[2]:
```python
#load data using paths to images
men = glob('/Users/bellabuchanan/Downloads/men/*')
women = glob('/Users/bellabuchanan/Downloads/women/*')
```

[3]:
```python
#Use less of the data to save time on runtime
#Was taking too long with full dataset
#Not because of memory and opening images, that was quick
#It was on the training that it took too long
men = men[:250]
women = women[:250]
```

[4]:
```python
#Code inspired by: https://github.com/joelgrus/shirts/blob/master/visuals.py
def img_to_array(filename):
    """
    takes a filename and turns it into a numpy array of RGB pixels
    """
    img = Image.open(filename)
    img = img.resize((138,138))
    img = list(img.getdata())
    img = list(map(list, img))
    img = np.array(list(img))
    s = img.shape[0] * img.shape[1]
    img_wide = img.reshape(1, s)
    return img_wide[0]
```

[5]:
```
#have image data and labels all together
data = [(img_to_array(filename),'men',filename) for filename in men] + \
           [(img_to_array(filename),'women',filename) for filename in women]
```

[6]:
```
#Shuffle the data (image data with labels)
seed(0)
shuffle(data)

#X and Y features
X = np.array([cd for (cd,_y,f) in data])
labels = np.array([_y for (cd,_y,f) in data])
```

[7]:
```
#split dataset into 80% training and 20% testing
X_train,X_test,y_train,y_test = train_test_split(X, labels, test_size=0.2,␣
 ↪random_state=0)

#check sizes
print(len(X_train))
print(len(X_test))
```

```
400
100
```

[8]:
```
#make the labels into zeros and ones
y_train_label = np.array([1 if label == 'men' else 0 for label in y_train])
y_test_label = np.array([1 if label == 'men' else 0 for label in y_test])
```

1.2 Support Vector Classifier

[9]:
```
#Create and fit model
clf = svm.SVC(kernel = 'linear')
clf.fit(X_train, y_train_label)
```

[9]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

[10]:
```
print("The training accuracy is:", clf.score(X_train, y_train_label))
```

```
The training accuracy is: 1.0
```

[11]:
```
print("The testing accuracy is:", clf.score(X_test, y_test_label))
```

```
The testing accuracy is: 0.67
```

1.3 PCA

[12]:
```
#PCA
N_COMPONENTS = 5
pca = PCA(n_components=N_COMPONENTS, random_state=0)
X_train_pca = pca.fit(X_train).transform(X_train)
X_test_pca = pca.transform(X_test)
```

[13]:
```
# Percentage of variance explained for each components
print('Percentage of variance explained (1st 5 components): %s'
      % str(pca.explained_variance_ratio_))
```

```
explained variance ratio (first 5 components): [0.36142451 0.09083787 0.04379208
0.03174099 0.02902699]
```

[14]:
```
#Fit SVC for pca data
clf.fit(X_train_pca, y_train_label)
```

[14]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

[15]:
```
print("The training accuracy is:", clf.score(X_train_pca, y_train_label))
print("The testing accuracy is:", clf.score(X_test_pca, y_test_label))
```

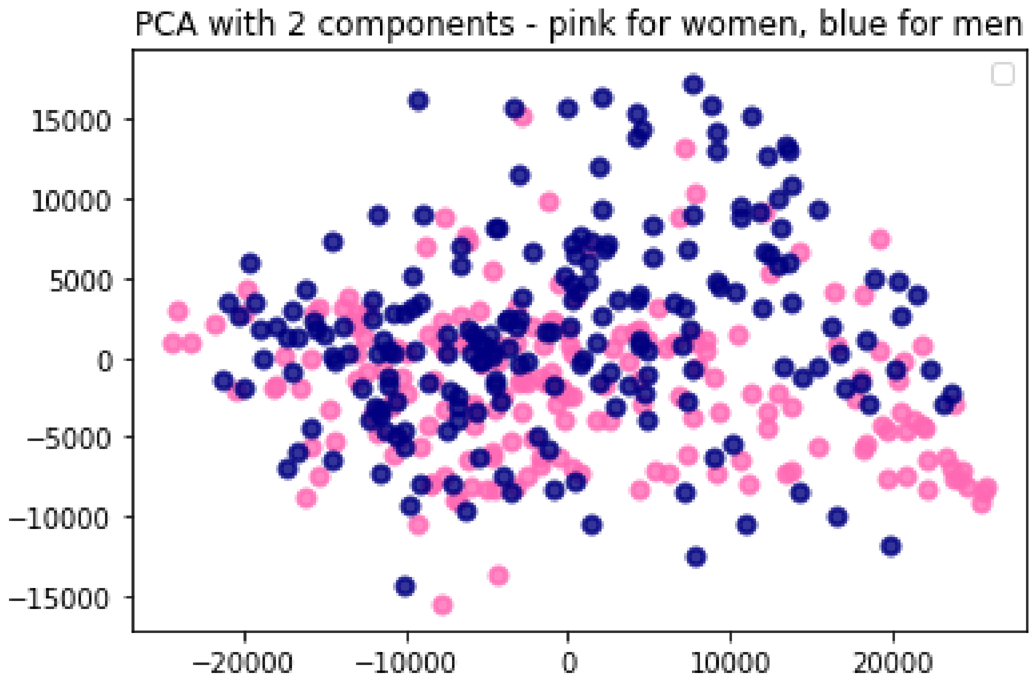
```
The training accuracy is: 0.6075
The testing accuracy is: 0.66
```

[16]:
```
#Plot first 2 components of PCA
plt.figure()
colors = ['hotpink', 'navy']
target_names = ['men', 'women']
lw = 2

for color, i, target_name in zip(colors, [0, 1], target_names):
    plt.scatter(X_train_pca[y_train_label == i, 0], X_train_pca[y_train_label␣
 ↪== i, 1], color=color, alpha=.8, lw=lw)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA with 2 components - pink for women, blue for men')
```

```
No handles with labels found to put in legend.
```

[16]: Text(0.5, 1.0, 'PCA with 2 components - pink for women, blue for men')



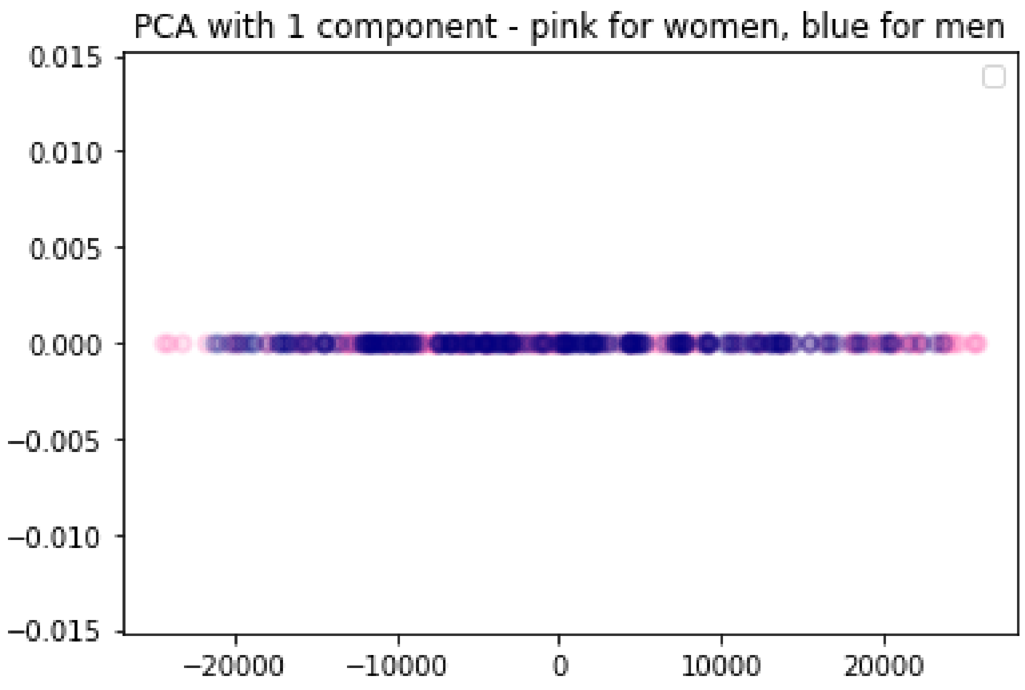
[17]:
```
#Plot first component of PCA so it is comparable to LDA
plt.figure()

target_names = ['men', 'women']
lw = 2

for color, i, target_name in zip(colors, [0, 1], target_names):
    plt.scatter(X_train_pca[y_train_label == i,0], np.
 ↪zeros(len(X_train_pca[y_train_label == i, 0])), color=color, alpha=0.2,␣
 ↪lw=lw)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA with 1 component - pink for women, blue for men')
```

```
No handles with labels found to put in legend.
```

[17]: Text(0.5, 1.0, 'PCA with 1 component - pink for women, blue for men')



1.4 LDA

[18]:
```
lda = LDA()
## train the LDA model with the training data
lda = lda.fit(X_train, y_train_label)

## Apply transform to the training and test set
X_train_LDA = lda.transform(X_train)
X_test_LDA = lda.transform(X_test)
```

```
/Users/bellabuchanan/anaconda3/lib/python3.7/site-
packages/sklearn/discriminant_analysis.py:388: UserWarning: Variables are
collinear.
  warnings.warn("Variables are collinear.")
```

Here, we compute the accuracy scores using the LDA classifier because LDA is in itself a classification method:

[19]:
```
print("The training accuracy is:", lda.score(X_train, y_train_label))
print("The testing accuracy is:", lda.score(X_test, y_test_label))
```

```
The training accuracy is: 0.8625
The testing accuracy is: 0.66
```

However, we also use the SVC we created earlier, so that we can compare the LDA transformed data's performance with the classifier to the PCA transformed data, and the original data.

[20]:
```
#Fit SVC for pca data
clf.fit(X_train_LDA, y_train_label)
```

[20]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

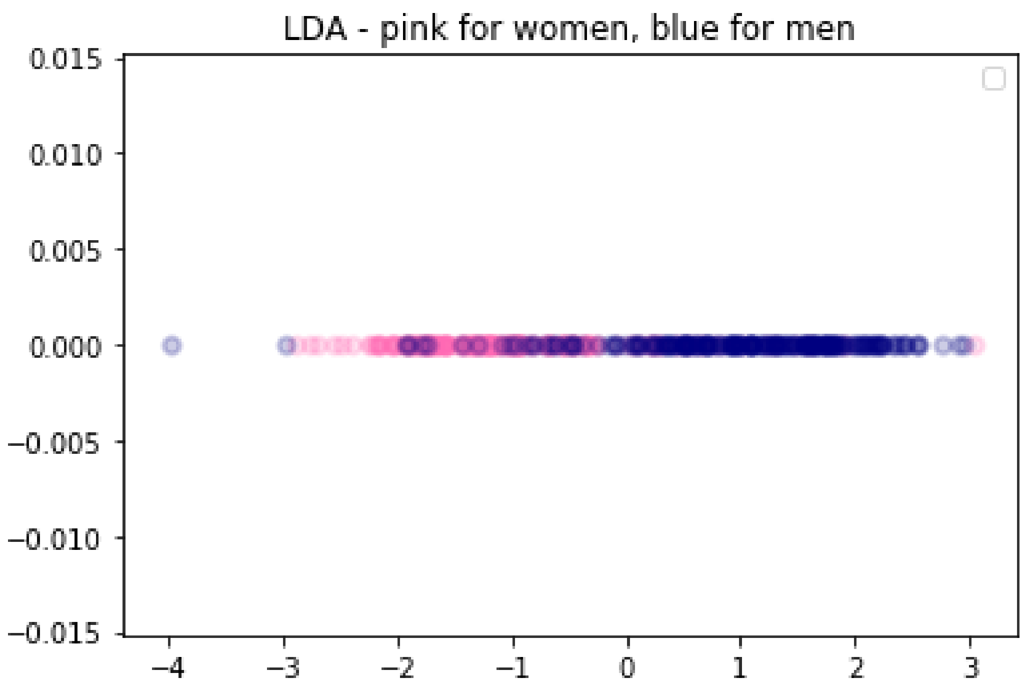
[21]:
```
print("The training accuracy is:", clf.score(X_train_LDA, y_train_label))
print("The testing accuracy is:", clf.score(X_test_LDA, y_test_label))
```

```
The training accuracy is: 0.8625
The testing accuracy is: 0.63
```

[22]:
```
#Plotting of LDA data
for color, i, target_name in zip(colors, [0, 1], target_names):
    plt.scatter(X_train_LDA[y_train_label == i], np.
 ↪zeros(len(X_train_LDA[y_train_label == i, 0])), color=color, alpha=.2, lw=lw)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA - pink for women, blue for men')
```

```
No handles with labels found to put in legend.
```

[22]:
```
Text(0.5, 1.0, 'LDA - pink for women, blue for men')
```



1.5 Description

PCA and LDA are both techniques used in dimensionality reduction. However there are differences between them. PCA is unsupervised and does not use class labels and tries to maximise the variance between the points whereas LDA uses class labels and aims to maximise the separability between the classes.

This comes through when we look at the 1D plots for LDA and PCA. The PCA plot has more overlap between pink and blue points. The 1st principle component explains 36.1% of the variance in the data which explains why there is a lot of overlap (this is not a large amount of the data). The LDA plot has less overlap of points and we see clearly separability between blue and pink, although there is still overlap.

This is because LDA is a classifier in itself, and we can see that the LDA decision boundary gets a high accuracy rate (86.25%) on the training data compared to PCA.

_	SVC	SVC Using PCA	SVC using LDA	LDA
Training Accuracy	1	0.6075	0.8625	0.8625
Testning Accuracy	0.67	0.66	0.66	0.63

1.5.1 Which is best?

However, the highest training accuracy rate is 100%. This is because the support vector classifier is not a dimensionality reduction technique but tries to find a hyperplane that can separate the data, making the 'gutters' of that hyperplane as wide as possible. In such high dimensions (as with our images), finding a hyperplane is possible and can completely separate the data. However, we see that this hyperplane is not very informative as it performs much worse on unseen data (67% accuracy). This could be because there are many components to male and female clothing but perhaps not all of them are useful in making the distinction between them. Because of the drop in accuracy on unseen data, I would not recommend using the SVC alone for this classification problem.

Now, we are left with SVC using PCA and SVC using LDA. Given the clearer separability of the data, and the higher training accuracy (although the drop from test to training is higher), I think that LDA is a better method for classification in this case. I think that PCA could be better if there was one component that explained a higher percentage of the data, but given that the highest is 36%, I think it is better to increase class separability and try and come up with a decision boundary as in LDA.

1.6 HCs

#dataviz: I visualise the LDA and PCA data in the same dimension along a line of zeros so that we can compare how separable the data looks. I explain a greater overlap with PCA and discuss this.

#responsibility: There has been so much panic in the community because of COVID-19. It took a huge amount of responsibility to get this done while dealing with flights in and out of Argentina being cancelled and worried friends and parents constantly calling

#organization: I organized my code by using subheadings to make it more readable. Before, I did not use 3 distinct sections and my code was confusing and did not run properly from start to finish. Reorganising it made it more readable for me, and hopefully for the grader