1. Split your dataset from the PCA pre-class work into 80% training data and 20% testing data.

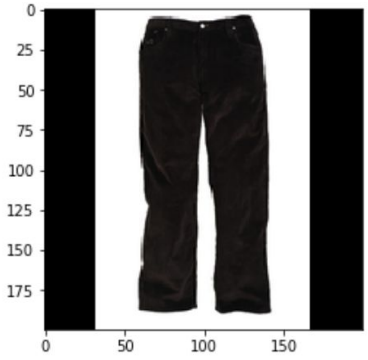
```
# original code by Joel Grus, author of the blog in the pre-class readings
# adapted from https://github.com/joelgrus/shirts/blob/master/

from PIL import Image
import PIL.ImageOps
from collections import defaultdict
import glob
from random import shuffle, seed
import numpy as np
import pylab as pl
import pandas as pd
import re
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
from matplotlib.pyplot import imshow
import matplotlib.pyplot as plt

def img_to_array(filename):
    """
    takes a filename and turns it into a numpy array of RGB pixels
    """
    img = Image.open(filename)
    desired_size = 200
    old_size = img.size
    ratio = float(desired_size)/max(old_size) # preserving image aspect ratio
    new_size = tuple([int(x*ratio) for x in old_size])
    img = img.resize(new_size, Image.ANTIALIAS)
    # creating new 200x200 black image as background to paste resized image
    # images pasted are centered
    new_img = Image.new("RGB", (desired_size, desired_size))
    new_img.paste(img, ((desired_size-new_size[0])//2,
                        (desired_size-new_size[1])//2))
    img = new_img
    img = list(img.getdata())
    img = map(list, img)
    img = np.array(img)
    img_wide = img.reshape(1, -1)
    return img_wide[0]

girls_files = glob.glob('/Women/*.JPEG')
boys_files = glob.glob('/Men/*.JPEG')

process_file = img_to_array

raw_data = [(process_file(filename),'girl',filename) for filename in girls_files] + \
           [(process_file(filename),'boy',filename) for filename in boys_files]

# randomly order the data
seed(0)
shuffle(raw_data)

# pull out the features and the labels
data = np.array([cd for (cd,_y,f) in raw_data])
labels = np.array([_y for (cd,_y,f) in raw_data])
```

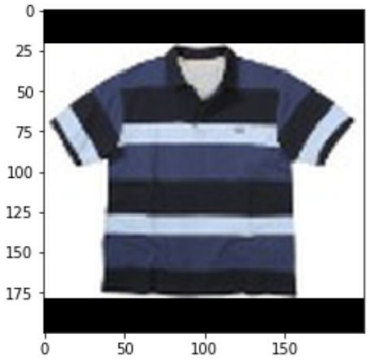
```
X = data
Y = [1 if label == 'boy' else 0 for label in labels]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

print "Relative size of training set: {}".format(X_train / X)
print "Relative size of testing set: {}".format(Y_test / Y)
```

The images are preprocessed to fit into a 200x200 square with black paddings, preserving its original aspect ratio. Resizing without preserving the original ratio might distort the image by stretching or shrinking, resulting in a loss of information. I chose a small size of 200x200 for two reasons: one to reduce the number of arrays processed and therefore improve model run times and two, because all of the photos are larger than this size, so we are not stretching any images beyond its original size (pixelation) during resizing. I also refrained from cropping the image into a specific size as well due to information loss. Therefore, the method of centering and resizing images will best preserve the information of our given dataset. I also manually preprocessed some images by excluding those without a clean foreground and background, here are some examples of the images that are processed into the model:

Old images:


New images:





2. Build a simple linear classifier using the original pixel data. What is your error rate on the training data? What is your error rate on your testing data?

```
from sklearn.svm import LinearSVC

print "Full Data without PCA or LDA"

# Logistic Regression
logreg = LogisticRegression(penalty='l2', class_weight='balanced', solver = 'liblinear')
logreg.fit(X_train, Y_train)
predictions = logreg.predict(X_test)

print ("Accuracy score for Logistic Regression: {}".format(metrics.accuracy_score(Y_test,
predictions)))
print("Classification report for classifier %s:\n%s\n"
      % ('Logistic Regression', metrics.classification_report(Y_test, predictions)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(Y_test, predictions))

# Linear SVC
svc = LinearSVC(random_state=0)
svc.fit(X_train, Y_train)
svcpredictions = svc.predict(X_test)

print ("Accuracy score for Linear SVC: {}".metrics.accuracy_score(Y_test, svcpredictions))
print("Classification report for classifier %s:\n%s\n"
      % ('Linear SVC', metrics.classification_report(Y_test, svcpredictions)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(Y_test, svcpredictions))
```

```
Accuracy score for Logistic Regression: 0.633

Classification report for classifier Logistic Regression:
             precision    recall  f1-score   support

          0       0.60      0.67      0.63       204
          1       0.67      0.60      0.63       226

avg / total       0.64      0.63      0.63       430

Confusion matrix:
[[136  68]
 [ 90 136]]
```

```
Accuracy score for Linear SVC: 0.630

Classification report for classifier Linear SVC:
             precision    recall  f1-score   support

          0       0.60      0.66      0.63       204
          1       0.66      0.60      0.63       226

avg / total       0.63      0.63      0.63       430


Confusion matrix:
[[135  69]
 [ 90 136]]
```

Error rate on the training data: 0 (because they are modelled after the training data and labels; not shown)
Error rate on the test data: 0.367 for Logistic Regression, 0.370 for Linear SVC (obtained using 1 - accuracy score)

Both Logistic Regression and Linear SVC performed similarly on the full test data. The only difference is that Logistic Regression performs 0.01 higher on recall for women clothes and 0.01 higher on precision for men clothes. I thought this difference was quite insignificant, given that both models are fairly accurate, producing scores above 0.6 on accuracy, precision and recall. Logistic regression is only slightly better than Linear SVC. However, empirically I observed that the actual run time of logistic regression is slightly lower than Linear SVC. I also ran Linear SVC for as a benchmark to logistic regression for all the models, but excluded it because its performance were inferior to Logistic Regression for the following questions.

3. Train the same linear model as in question 1, but now on the reduced representation that you created using PCA. What is your error rate on the training data? What is your error rate on your testing data?

```
from sklearn.decomposition import PCA
from skimage import io
from pylab import *
import operator

accuracy_scores = []
n_comps = []
for i in range(1,10):
    n_comp = i * 8
    # Fit different PCA on all images
    pca = PCA(n_components = n_comp)
    x_x = pca.fit_transform(X)

    X_train_PCA_, X_test_PCA_, Y_train_PCA_, Y_test_PCA_ = train_test_split(x_x, Y,
test_size=0.2, random_state=42)

    logreg_logreg = LogisticRegression(penalty='l2', class_weight='balanced', solver =
'saga')
    logreg_logreg.fit(X_train_PCA_, Y_train_PCA_)
    predictions_PCA_ = logreg_logreg.predict(X_test_PCA_)
    accuracy_scores.append(metrics.accuracy_score(Y_test_PCA_, predictions_PCA_))
    n_comps.append(n_comp)

index_with_higest_accuracy = max(enumerate(accuracy_scores),key=lambda x: x[1])[0]
n_components = n_comps[index_with_higest_accuracy]

print "PCA with {} number of components".format(n_components)

pca_final = PCA(n_components = n_components)
new_X = pca_final.fit_transform(X)
X_train_PCA, X_test_PCA, Y_train_PCA, Y_test_PCA = train_test_split(new_X, Y, test_size=0.2,
random_state=42)
logreg_PCA = LogisticRegression(penalty='l2', class_weight='balanced', solver = 'saga')
logreg_PCA.fit(X_train_PCA, Y_train_PCA)
predictions_PCA = logreg_PCA.predict(X_test_PCA)

#obtain the scores to measure
print ("Accuracy score for Logistic Regression:
{}".format(metrics.accuracy_score(Y_test_PCA, predictions_PCA)))
print("Classification report for classifier %s:\n%s\n"
      % ('Logistic Regression', metrics.classification_report(Y_test_PCA, predictions_PCA)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(Y_test_PCA, predictions_PCA))
```

```
PCA with 32 number of components
Accuracy score for Logistic Regression: 0.695348837209
Classification report for classifier Logistic Regression:
             precision    recall  f1-score   support

          0       0.66      0.73      0.69       204
          1       0.73      0.66      0.70       226

avg / total       0.70      0.70      0.70       430


Confusion matrix:
[[149  55]
 [ 76 150]]
```

This method would find the number of principal components that produce the highest accuracy on the logistic regression model, which is 32. The process used here is similar to finding the increase in explained variance with the number of components, and using the number of components that explain the highest amount of variance in the dataset. The final accuracy score is 0.695, which is fairly good and actually higher than the 0.633 that is obtained in question 2 when PCA was not conducted. The PCA also improved average precision, recall and F1 score to 0.70 from 0.64, 0.63 and 0.63 respectively. The reduced representation seems to improve the model, though at the risk of overfitting. To improve the generalizability of our machine learning model, L1 regularization is also implemented to reduce this overfitting.

4. Train the same linear model as in question 1, but now on the reduced representation that you created using LDA. What is your error rate on the training data? What is your error rate on your testing data?

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

print "LDA"

X_train_LDA, X_test_LDA, Y_train_LDA, Y_test_LDA = train_test_split(X, Y, test_size=0.2,
random_state=42)
lda = LDA(n_components=1)
lda.fit_transform(X_train_LDA,Y_train_LDA)
predictions_LDA = lda.predict(X_test_LDA)

print ("Accuracy score for Linear Discriminant Analysis:
{}".format(metrics.accuracy_score(Y_test_LDA, predictions_LDA)))
print("Classification report for classifier %s:\n%s\n"
      % ('LDA', metrics.classification_report(Y_test_LDA, predictions_LDA)))

logreg_LDA = LogisticRegression(penalty='l2', class_weight='balanced', solver = 'saga')
logreg_LDA.fit(X_train_LDA, Y_train_LDA)
log_predictions_LDA = logreg_LDA.predict(X_test_LDA)

print ("Accuracy score for Logistic Regression:
{}".format(metrics.accuracy_score(Y_test_LDA, log_predictions_LDA)))
print("Classification report for classifier %s:\n%s\n"
      % ('Logistic Regression', metrics.classification_report(Y_test_LDA,
log_predictions_LDA)))
```

```
LDA
Accuracy score for Linear Discriminant Analysis: 0.625581395349
Classification report for classifier LDA:
             precision    recall  f1-score   support

          0       0.60      0.63      0.62       204
          1       0.65      0.62      0.63       226

avg / total       0.63      0.63      0.63       430

Accuracy score for Logistic Regression: 0.655813953488
Classification report for classifier Logistic Regression:
             precision    recall  f1-score   support

          0       0.63      0.68      0.65       204
          1       0.69      0.63      0.66       226

avg / total       0.66      0.66      0.66       430
```

Logistic regression performed better than the Linear Discriminant Analysis native classification method in every aspect measured. However, in terms of dimensionality reduction, PCA was better than LDA. The Logistic Regression performed with PCA had an accuracy score of 0.695, which is higher than its LDA counterpart with only 0.656. This suggests that PCA performed better as a dimensionality reduction technique. LDA's weak score here can be attributed to its classification mechanism, because there are only 2 classes so it only had 1 component to work with. LDA would try to maximize the variance between the two classes with this component, which may be problematic, as I will describe in my full analysis later.

5. Write three paragraphs, describing and interpreting your results from questions 1, 2, and 3. Make a recommendation on which classifier you would prefer, and why.

PCA versus LDA

The models results revealed that Logistic Regression with PCA was superior to LDA in classifying images between men and women's clothings. The former had an accuracy score of 0.695 whereas the latter was marginally less accurate with 0.656. Furthermore, PCA analysis found the 32 components that accounted for the largest variance in the data while LDA only had 1 component since its components are based on the number of classes there are (men and women). For this dataset, we conclude that PCA was the better method of dimensionality reduction to the extent that there are only 2 classes and that there were no significant error in the algorithms.

PCA versus no-PCA

This model was also the most accurate (0.695) when compared to native Logistic Regression performed on the full dimensionality of the data (0.633), and Linear SVC (0.630). This is quite counter-intuitive because reducing dimensions tend to result in loss of information, which may reduce the inaccuracy of the model. However, perhaps reducing dimensions helped improve the model because the images provided in the dataset were not the cleanest with a lot of noise present. Noise such as secondary clothes and faces in the foreground and non-uniform backgrounds could be ignored when we focus on finding the components that explains the most variance (i.e. the shirts themselves). In other words, using PCA could improve the signal-to-noise ratio of our dataset for the model to be more accurate.

Notes on Data and Suggested Use for this Model

In reality, the difference between men and women shirts can be quite small. This was reflected on the dataset received in this assignment, where men clothes were generally darker, bluer colors while women clothes were lighter and redder. Ambiguity exists for gender-neutral shirt designs in colors, shape, sizes, etc. This may be traced in the models by the precision and recall scores that resulted in both false negatives and false positives. In this case, both false negatives and positives are equally important and any mistakes will result in wrong classification e.g. mistakenly buying a women shirt online when one intended to buy a men shirt. As a recommendation, this model could be implemented in e-commerce websites to classify clothes.

Practically, this model can be well generalized to the practical problem of classifying men and women clothes. This model performs better than chance (accuracy = 0.5) in classifying men and women clothes and perhaps can match a human that is relatively untrained (accuracy = 0.7) in classifying clothes. Take my own case, for example, I would definitely make errors classifying some shirts in the dataset into one gender or the other. To improve this model, I recommend that we create a third class of gender-neutral shirts to capture the false positives and false negatives from this model, which will call for a human-expert to manually classify these shirts. Having the third class could also mean that LDA might perform better given an increased

component and the curse of dimensionality, where information increases exponentially with components.