# CS156 Assignment 4

**Armin Hamp**

In [1]:

```
#load libraries
import numpy as np
import random
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import svm
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.decomposition import PCA
from PIL import Image
from glob import glob
```

# Pre-processing data

In [2]:

```
def img_to_array(filename):
    """
    takes a filename and turns it into a numpy array of RGB pixels
    from: https://github.com/joelgrus/shirts/blob/master/visuals.py
    """
    img = Image.open(filename)
    img = img.resize(new_size)
    img=img.rotate(270, expand=True)
    img1 = list(img.getdata())
    img.close()
    img1 = map(list, img1)
    img1 = np.array(list(img1))
    s = img1.shape[0] * img1.shape[1]
    #print(img.shape[0], img.shape[1])
    img_wide = img1.reshape(1, s)
    return img_wide[0]
```

In [4]:

```
#load pic addresses
men_dir=glob('/Users/Armin/Documents/Minerva Year 3/Semester 2/CS156/Notebooks/male-clothing/*')
women_dir=glob('/Users/Armin/Documents/Minerva Year 3/Semester 2/CS156/Notebooks/female-clothing/*')

men_dir, women_dir=men_dir[:300],women_dir[:300]
#resize images to 128x128
new_size=(128,128)
men=[img_to_array(i) for i in men_dir]
women=[img_to_array(i) for i in women_dir]
```

In [5]:
```
#split data into train and test sets
y=np.concatenate((np.ones(len(men)),np.zeros(len(women)))) #1 for men 0 for women
X=np.concatenate((men,women))

#split data randomly by 80/20 train/test
random.seed(123)
ids=list(range(0,len(X)))
test_ids=random.sample(ids,int(len(X)*0.2))
train_ids=[ids[i] for i in range(len(ids)) if i not in test_ids]
X_train=[X[i] for i in train_ids]
X_test=[X[i] for i in test_ids]
y_train=[y[i] for i in train_ids]
y_test=[y[i] for i in test_ids]

#shuffle training data
indices=np.arange(0,len(X_train))
np.random.shuffle(indices)
X_train=[X_train[i] for i in indices]
y_train=[y_train[i] for i in indices]
```

## SVC on original data

In [6]:
```
clf=svm.SVC(kernel="linear") #set up classifier
clf.fit(X_train,y_train) #fit the classifier
print("Accuracy of model for training data:",clf.score(X_train,y_train))
print("Accuracy of model for test data:",clf.score(X_test,y_test))
```

```
Accuracy of model for training data: 1.0
Accuracy of model for test data: 0.6083333333333333
```

## SVC on PCA data

In [7]:
```
# apply PCA transformation to data
N_COMPONENTS = 10
pca = PCA(n_components=N_COMPONENTS, svd_solver="randomized") #create model
pca_X_train= pca.fit_transform(X_train) # fit model and apply transformation
pca_X_test=pca.fit_transform(X_test)
```

In [8]:
```
#create SVC for PCA data
clf=svm.SVC(kernel="linear") #set up classifier
clf.fit(pca_X_train,y_train)
print("Accuracy of model for training data:",clf.score(pca_X_train,y_train))
print("Accuracy of model for test data:",clf.score(pca_X_test,y_test))
```

```
Accuracy of model for training data: 0.6729166666666667
Accuracy of model for test data: 0.55
```

## SVC on LDA data

In [9]:

```
#apply LDA transormation
lda = LDA()
lda.fit(X_train,y_train) #fit model
lda_X_train=lda.transform(X_train)
lda_X_test=lda.transform(X_test)

#create SVC for LDA data
clf=svm.SVC(kernel="linear") #set up classifier
clf.fit(lda_X_train,y_train)
print("Accuracy of LDA transform on training data:",lda.score(X_train,y_train))
print("Accuracy of LDA transform on test data:",lda.score(X_test,y_test))
print("Accuracy of SVC for training data:",clf.score(lda_X_train,y_train))
print("Accuracy of SVC for test data:",clf.score(lda_X_test,y_test))
```

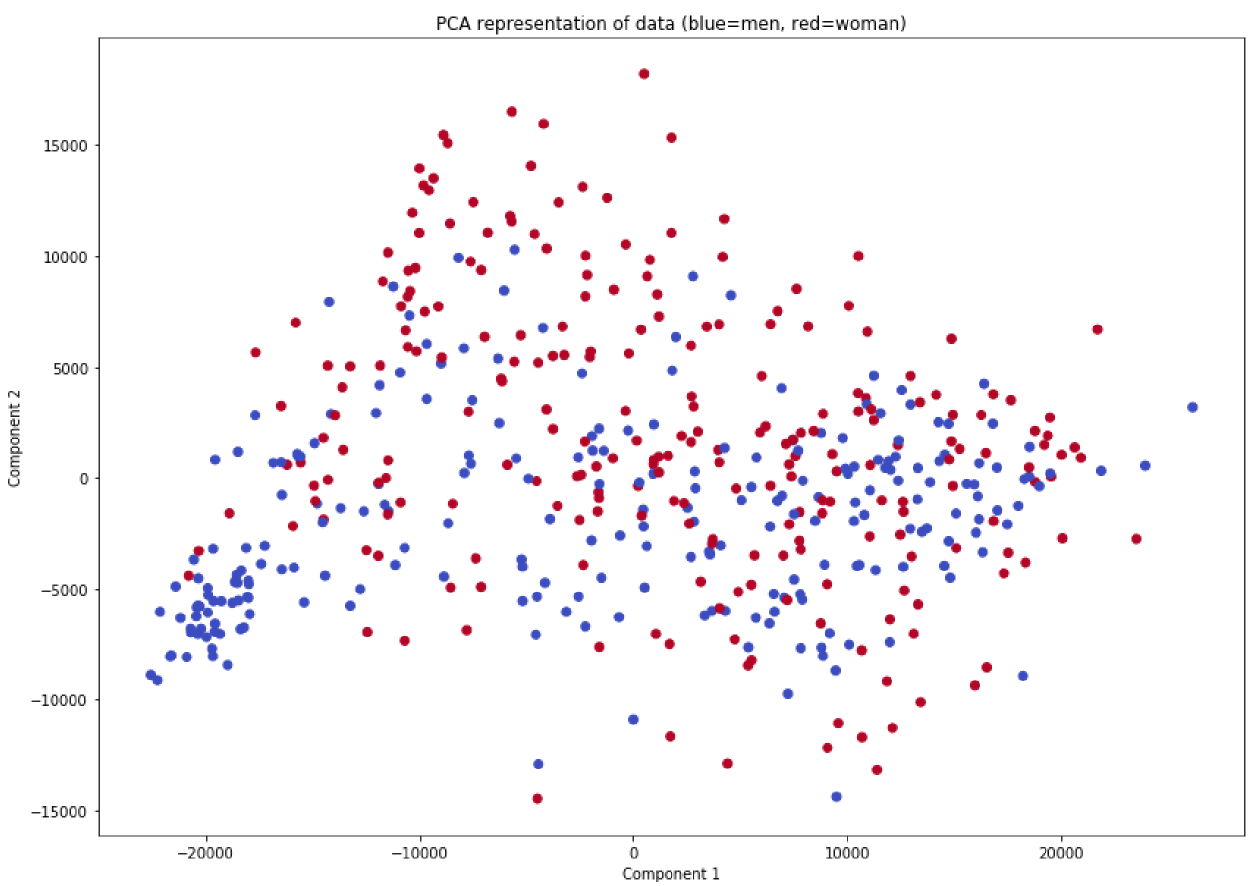
```
C:\Users\Armin\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:388: UserWarning: Variabl
es are collinear.
  warnings.warn("Variables are collinear.")

Accuracy of LDA transform on training data: 0.9020833333333333
Accuracy of LDA transform on test data: 0.6083333333333333
Accuracy of SVC for training data: 0.89375
Accuracy of SVC for test data: 0.6083333333333333
```

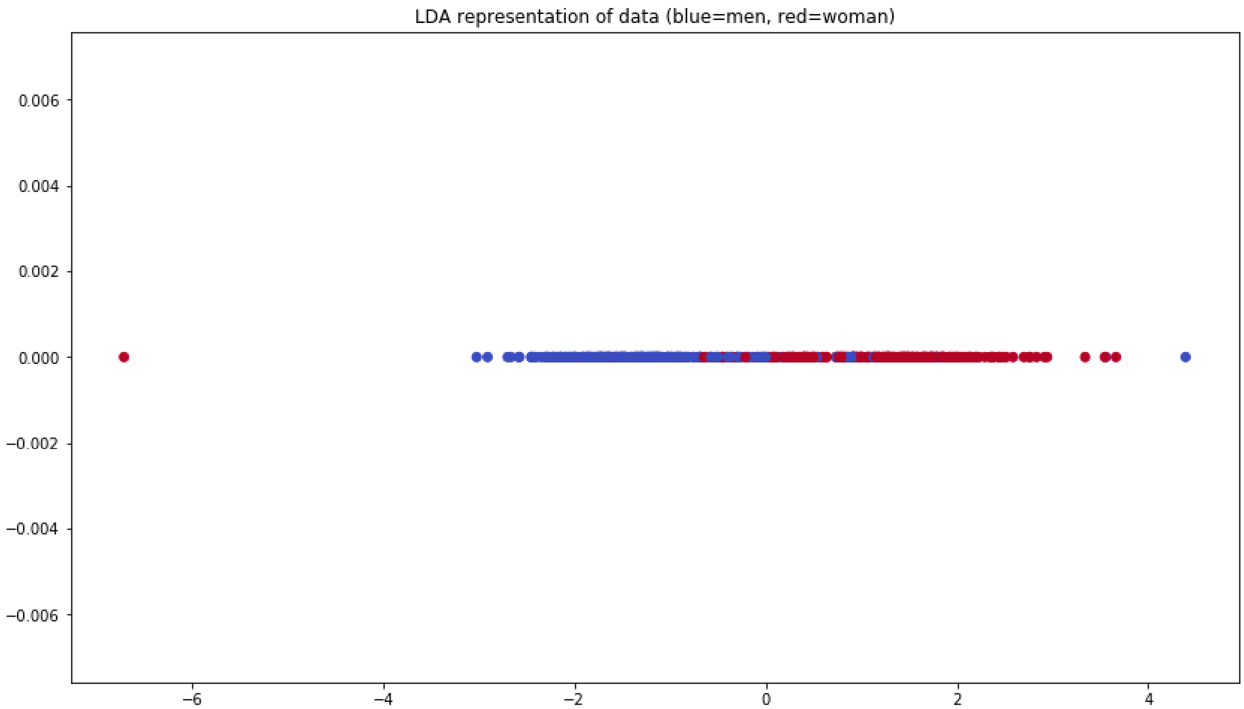
## Visualization of PCA vs LDA data

In [10]:

```
#plot
plt.figure(figsize=(14,10))
plt.title("PCA representation of data (blue=men, red=woman)")
plt.scatter(pca_X_train[:,0],pca_X_train[:,1],c=y_train,cmap=plt.cm.coolwarm,label="men" )
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```



In [11]:
```
#plot
plt.figure(figsize=(14,8))
plt.title("LDA representation of data (blue=men, red=woman)")
plt.scatter(lda_X_train[:,0],np.zeros(len(lda_X_train)),c=y_train,cmap=plt.cm.coolwarm,label="men" )
plt.show()
```




# Discussion

## Original Data

I have reduced my images to a size of $128 \times 128$ pixels. Given that these are RGB images, an image in the data can be represented by $128 \times 128 \times 3 = 49152$ datapoints. I fitted a Support Vector Classifier with a linear kernel on the training data and found that it achieved $100\%$ accuracy on training data, but only $61\%$ on the test data. The problem with this raw data is that the SVC has to find an optimal decision boundary amongst $49152$ dimensional points, out of which dimensions probably most do not help to explain the variance in our data. Yet, this model performs just as well as the best dimensionality reduction technique.

## PCA

Principal Component Analysis is an unsupervised dimensionality reduction technique that eliminates dimensions with low-explanatory power, and using linear projections extracts the most important features of our data. Fitting an SVC on the PCA transformed data results in $67\%$ accuracy on training data, and $55\%$ on the test data. I re-ran the model with a number of different $n_{components}$ values, which had little effect on these scores. PCA is flexible and can reduce the dimensionality of data to a wide range of target dimensions. As opposed to LDA (read below).

## LDA

LDA is a supervised dimensionality reduction technique, that uses a formula based on group variance to create a projection that makes classes the most separable (thus it can be considered a classifier by itself). This results in better training and test accuracies than PCA with $90\%$ accuracy on training and $61\%$ on test data. However, LDA's reduction means that it's classification is just a linear decision boundary (as with two classes, the projection will be 1D). This seems to not be a problem here, but can result in a loss of valuable information that other machine learning techniques would utilize.

Given, no significant improvements in accuracy, I do not think it is worth using any of the dimensionality reduction techniques. Perhaps, with more data the differences of performance would be salient enough to make any of these methods a clear winner.