[CS156] Assignment 4: Machine Learning Fashionista

Akmarzhan Abylay

Preprocessing

For this assignment, I used the jerseys/shirts dataset.

[1]:
```
#importing the libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from glob import glob
from PIL import Image
from skimage.transform import resize
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#paths for images
glob_shirts = glob('shirts/*')
glob_jerseys = glob('jersey/*')
```

[2]:
```
def extract(folder, out):
    """Function for extracting and resizing the images."""
    lst = []
    for path in folder:
        #resizing the image - I used (30, 30) because my computer actually
        crashed when doing (70, 70) for some reason, and it got real slow
        when I did (50, 50), which is why I chose (30, 30)
        conv = Image.open(open(path, 'r+b')).resize((30, 30))

        #some images had a different shape, which is why I had
        #to make the following condition as np.stack didn't want
        #to work otherwise
        if len(np.array(conv).flatten())==2700:
            #flatten the matrix to an array
            lst.append((np.array(conv).flatten(), out))
            Image.open(open(path, 'r+b')).close()
    return lst

#turning into arrays
shirts = np.asarray(extract(glob_shirts, 0))
jerseys = np.asarray(extract(glob_jerseys, 1))
```

For the preprocessing, I manually checked the images to see whether they were misclassified. I found out many images are misclassified, especially within the shirts section, since around 250 "shirts" were, in fact, jerseys). I made sure they were in the right folder.

I also found that many images weren't even shirts or jerseys but other types of clothing. Many pictures included several fragments of shirts/jerseys, many people in one shot, and even dogs wearing mini jerseys. I deleted the unnecessary images since they don't add to the classification (they could have made it worse).

In the end, there were around 1000 pictures left in the shirts section and 1350 in the jerseys section (i.e., initially, there were 1470 shirts and 1331 jerseys). Because the dataset is unbalanced, I decided to up-sample the minority class (i.e., shirts), meaning that I resampled the shirts class with replacements to match the lengths. This wouldn't change our data but would prevent the imbalance as the classification results otherwise might be misleading.

[3]:
```
from sklearn.utils import resample

#sample with replacement
shirts = resample(shirts, replace=True,
                  n_samples=len(jerseys),
                  random_state=0)
#making sure now we have the same number of shirts and jerseys
len(shirts)==len(jerseys)
```

[3]:
```
True
```

[4]:
```
#initializing the X and y using both the jerseys and shirts
X = np.append(jerseys[:, 0], shirts[:, 0])
y = np.append(jerseys[:, 1], shirts[:, 1])

X = np.stack(i for i in X)
y = np.stack(i for i in y)
```

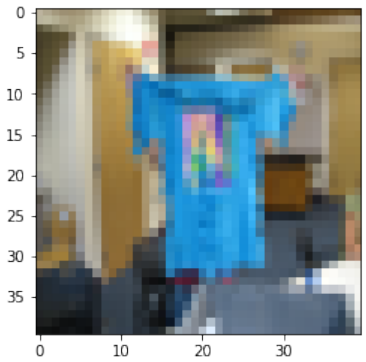
Splitting (Train, Test)

As given in the instructions, I split my dataset into 80% training and 20% testing data. I also scaled it so that each feature contributes equally.

[5]:
```
import random
random.seed(0)

#scaling the inputs to get better results
sc = StandardScaler()
X = sc.fit_transform(X)

#splitting the dataset into 80% training and 20% testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 0)
```

This is how one of the images look. I resized it to (30, 30), which is why it looks blurry.

[6]:
```
def visualize(image):
    """Function for visualizing the images."""
    image = image.astype('uint8')
    np.clip(image, 0, 255)
    image= Image.fromarray(image.reshape(40, 40, 3))
    plt.imshow(image)

original = sc.inverse_transform(X_train[0])
visualize(original)
```



Linear SVC

I classified the images using `LinearSVC()`. I used `GridSearchCV()` to get the best parameters.

[7]:
```
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.001, 0.01, 0.1, 1],
    'tol': [1e-6, 1e-4, 1e-2]}

search = GridSearchCV(LinearSVC(), param_grid, n_jobs=-1)
search.fit(X_train, y_train)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

```
Best parameter (CV score=0.725):
{'C': 0.001, 'tol': 1e-06}
```

[8]:
```
svc = LinearSVC(random_state = 0, C=0.001, tol=1e-06)
svc.fit(X_train, y_train)
```

[8]:
```
LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=0, tol=1e-06,
          verbose=0)
```

[9]:
```
from sklearn.model_selection import cross_val_score

def accuracy(clf, X_train, X_test, data):
    """Function for printing the accuracies."""

    acc_train = cross_val_score(estimator = clf, X = X_train, y = y_train, cv =␣
 ↪5)
    acc_test = cross_val_score(estimator = clf, X = X_test, y = y_test, cv = 5)

    print("[Linear SVC] on {} Data\n\nTrain Set Accuracy: {:.2f} %".format(data,␣
 ↪acc_train.mean()*100))
    print("Test Set Accuracy: {:.2f} %".format(acc_test.mean()*100))

    return acc_train.mean()*100, acc_test.mean()*100
```

[10]:
```
acc_train_svc, acc_test_svc = accuracy(svc, X_train, X_test, "Original")
```

```
[Linear SVC] on Original Data

Train Set Accuracy: 72.52 %
Test Set Accuracy: 65.92 %
```

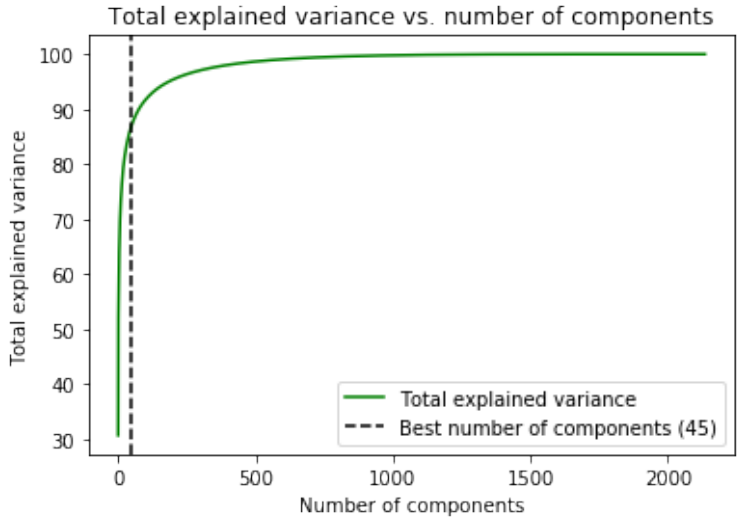
PCA

I found the number of components for PCA, which leads to the best results using `GridSearchCV()` and trained the `SVC()` on the reduced representation using PCA.

[11]:
```
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

pipe = Pipeline(steps=[('pca', PCA()), ('svc', LinearSVC(random_state = 0, C=0.
 ↪001, tol=1e-06))])

param_grid = {'pca__n_components': [i for i in range(50)]}

search = GridSearchCV(pipe, param_grid, n_jobs=-1)
search.fit(X_train, y_train)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

```
Best parameter (CV score=0.684):
{'pca__n_components': 45}
```

[12]:
```
#### starts not from 0
pca1 = PCA()
pca1.fit(X_train)
cumsum = np.cumsum(pca1.explained_variance_ratio_)*100
variances = [i for i in range(len(cumsum))]

plt.plot(variances, cumsum, color = 'green', label='Total explained variance')
plt.title("Total explained variance vs. number of components")
plt.ylabel('Total explained variance')
plt.xlabel('Number of components')

plt.axvline(x = 45, color="black", linestyle='--', label = 'Best number of␣
 ↪components (45)')
plt.legend(loc='best')
plt.show()
```

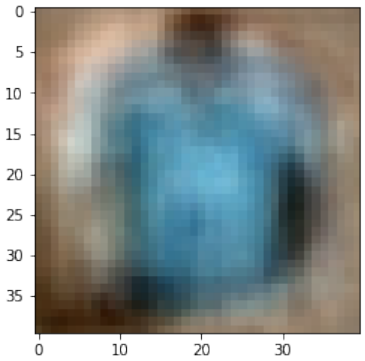


[13]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 45)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
print("The explained variance for 45 components is:",
      sum(pca.explained_variance_ratio_)*100)
```

```
The explained variance for 45 components is: 86.24265055511387
```

This is how the PCA-transformed image looks like. More blurry than the original.

[14]:
```
pca_im = pca.inverse_transform(X_train_pca[0])
pca_im = sc.inverse_transform(pca_im)
visualize(pca_im)
```



[15]:
```
svc_pca = LinearSVC(random_state = 0, C=0.001, tol=1e-06, verbose=0)
svc_pca.fit(X_train_pca, y_train)
```

[16]:
```
acc_train_pca, acc_test_pca = accuracy(svc_pca, X_train_pca, X_test_pca, "PCA")
```

```
[Linear SVC] on PCA Data

Train Set Accuracy: 68.21 %
Test Set Accuracy: 66.65 %
```

LDA

I did the same thing as in the section above, but for LDA. We can only use `n_components=1` as it should be less than the number of classes (i.e., 2).

[17]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
```

[18]:
```
svc_lda = LinearSVC(random_state = 0, C=0.001, tol=1e-06, verbose=0)
svc_lda.fit(X_train_lda, y_train)
```

[19]:
```
acc_train_lda, acc_test_lda = accuracy(svc_lda, X_train_lda, X_test_lda, "LDA")
```

```
[Linear SVC] on LDA Data

Train Set Accuracy: 99.34 %
Test Set Accuracy: 72.85 %
```

Report

It was hard even for me to classify some of the images, so I didn't expect the classifiers to be very successful. Below I made a comparison of all test and train set accuracies.

[20]:
```
accuracies = {'Linear SVC':[acc_train_svc, acc_test_svc],
              'PCA (n=45)':[acc_train_pca, acc_test_pca],
              'LDA':[acc_train_lda, acc_test_lda]}

pd.DataFrame(accuracies, index=['Accuracy (train)', 'Accuracy (test)']).round(2)
```

[20]:

	Linear SVC	PCA (n=45)	LDA
Accuracy (train)	72.52	68.21	99.34
Accuracy (test)	65.92	66.65	72.85

As we can see, SVC with LDA performed exceptionally well for the train set (i.e., 99.3%) but wasn't as successful in distinguishing shirts from jerseys for the test set (i.e., 72.9%). It still performed better than the other two models, but there is high variance, possibly meaning that it is overfitting. This is probable because LDA is optimal whenever all the assumptions it has are valid: data is normally distributed, and each class has an equal variance. It is also susceptible to outliers because it maximizes the separation between **means** of the class projections (as well as attempts to minimize the variance within each class). This is why I tried to remove strange pictures from the dataset and standardize the input. Still, the data wasn't ideal, and I resized the images, which is why we can say LDA performed relatively well.

Compared to LDA, the PCA is unsupervised, which is why it tries to maximize the overall variance of the data along a set of directions. Thus, the PCA might make inefficient choices and pick directions, making it hard to separate classes. This might be the case for our data, too, since Linear SVC with PCA fails to successfully distinguish the jerseys from shirts (train and test set accuracies of around 68.2% and 66.7%, respectively), although the 45 chosen components explain about 86% of the variance. However, given our data, it still isn't that bad, improving the test set accuracy from a Linear SVC with original data by around 0.7%.

Linear SVC did the worst for the test set (i.e., 65.9%), although it did relatively well for the train set (i.e., 72.5%), which might be because it overfitted the data. I would personally recommend LDA for this specific case because it has the best test set accuracy, and it took the labels into account, which is suitable for our aim (i.e., separating and classifying jerseys and shirts). However, we should note that it only works when the data meets the conditions of LDA. Before manually fixing the dataset (i.e., for the original dataset), the LDA performed the worst out of all three with a test set accuracy of around 58% (whereas PCA did the best with 62%), which supports my point about it not being very suitable for data that doesn't satisfy the main conditions and has many outliers. We could have achieved better results with better data, larger size, more pre-processing, or a more suitable classification algorithm.