assignment

November 20, 2021

0.1 Introduction

In this assignment, three kinds of Linear Support Vector Classifiers will be trained to classify Shirts and Jerseys from the ImageNet database. The first will be trained on full images (i.e. flattened `64px x 64px` RGB images, with `64 * 64 * 128 = 12,288` dimensions each), the second will be trained on a PCA-reduced representation (with 561 dimensions), while the third is trained on an LDA-representation (with just 1 dimensions, as there are only 2 class labels). We first import all available images, resize and cohere to a format that sk-learn respects,

[ ]:
```
shirts = glob("../18/Shirts/*") # 0
jerseys  = glob("../18/Jerseys/*") # 1
```

[ ]:
```
WIDTH = 64
HEIGHT = WIDTH
LABELS = ['jerseys', 'shirts']

# Array to hold training images
imgs = []
# Get target labels for LDA
y = []

def preprocess_clothing(dir, label):

    global imgs, y

    for img_path in dir:
        img = image.load_img(img_path, target_size=(WIDTH, HEIGHT))
        x = image.img_to_array(img)
        # Augment (size, size) to (samples, sizes, sizes)
        x = np.expand_dims(x, axis=0)

        imgs.append(x)
        y.append(label)


preprocess_clothing(shirts, label = 1)
preprocess_clothing(jerseys, label = 0)
```

```
imgs = np.vstack(imgs)
y = np.vstack(y)

# Complete preprocessing by flattening imgs
imgs_flat = imgs.reshape(len(imgs), WIDTH*HEIGHT*3)

# Rescale to help with optimization
scale  = StandardScaler().fit(imgs_flat)

imgs_flat = scale.transform(imgs_flat)

# Split into train/validation set
X_train, X_test, y_train, y_test = train_test_split(imgs_flat, y, test_size = 0.
↪2)

# 2802, 12288
assert X_train.shape[0] + X_test.shape[0] == len(shirts) + len(jerseys),␣
↪"Mangled Sizing"
```

We have an input matrix X with size (2,802, 12,288) - a total of 2802 images. Now, we perform a search over possible parameters for the Linear SVC. We do this to compare training times and accuracy scores between the three sets of models.

[ ]:
```
params = {}
params['C'] = (1e-6, 1e-4, 1e-2, 1, 10)
params['tol'] = [1e-6,1e-4]
params['max_iter'] = [10000]

clf = GridSearchCV(svm.LinearSVC(), params, scoring = 'accuracy')
clf.fit(X_train, y_train.ravel())

plot_paramsearch(clf)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
```

```
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

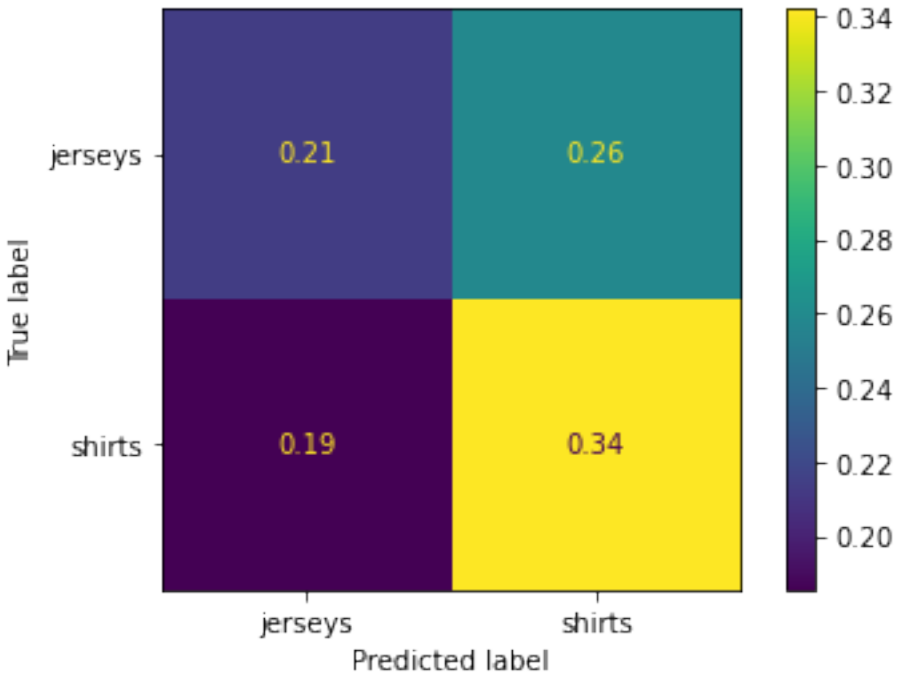
0.2 Basic SVC

We can now initialize the 'basic SVC' model with the best parameters and fit the model to the input data: full and flat images of jerseys and shirts,

[ ]:
```
# Fit SVM model with linear kernel
best_params  = clf.best_estimator_.get_params()

start = time()
svm_base = svm.LinearSVC(**best_params)
svm_base.fit(X_train, y_train.ravel())
stop = time()

training_time = round(stop - start, 2)
```

[ ]:
```
# Score basic linear kernel and show training time
print(
    "Test Accuracy Score:",  accuracy_score(y_test.ravel(), svm_base.
 ↪predict(X_test)), '\n',
    "Train Accuracy Score:", accuracy_score(y_train.ravel(), svm_base.
 ↪predict(X_train)), '\n',
    "Training Time:", f'{training_time}s'
)

# Show confusion matrix for test set
show_cm(y_test, svm_base, X_train)

print(svm_base.support_vectors_.shape[0])
```

```
Test Accuracy Score: 0.5561497326203209
 Train Accuracy Score: 0.9754573850959393
 Training Time: 22.01s
```



After a training time of about 22 seconds, the basic SVC performs extremely well on the training set, with an overall accuracy of 97.5%. However, this strong contrast in the test set accuracy of 55.6% evinces a problem of overfitting. It seems that the classifier is learning "noise" in the images (creases, background elements - though these examples is a lax abstraction over what exactly happens) as discrenible traits of jerseys and shirts. This sense of overfitting is reinforced by the large number of support vectors relative to the total samples (1674 : 2802).

0.3 PCA-Representation SVC

The process of dimensionality reduction is, in a sense, a process of feature summarization. To combat the problem of overfitting that assails the basic SVC above, we can train another SVC on data that is PCA-reduced.

The quality of a PCA in describing the original data (ie total explained variance ratio) increases monotonically with the number of principal components, albeit at a decreasing rate. We want the explained variance ratio to be high, about 95%, which neatly corresponds to the number of samples in the test set,

(Note: this is not a rationale. The author just likes coincidences.)

[ ]:
```
# Transform to PCA representation, with as many components as possible
N_COMPONENTS = X_test.shape[0]

pca = PCA(n_components = N_COMPONENTS)
X_pca = pca.fit_transform(X_train)
```

```
# 12,288 -> 561 features
X_pca.shape, round(sum(pca.explained_variance_ratio_),2)
```

[ ]: ((2241, 561), 0.9453890020231484)

An image's representation has been reduced from 12,288 dimensions to just 561. Our hope is that these new components capture the "essence" of jerseys and shirts. As before, we first scan for the best hyperparameters to the model before fitting,

[ ]:
```
clf_pca = GridSearchCV(svm.LinearSVC(), params, scoring = 'accuracy')
clf_pca.fit(X_pca, y_train.ravel())

plot_paramsearch(clf_pca)
```

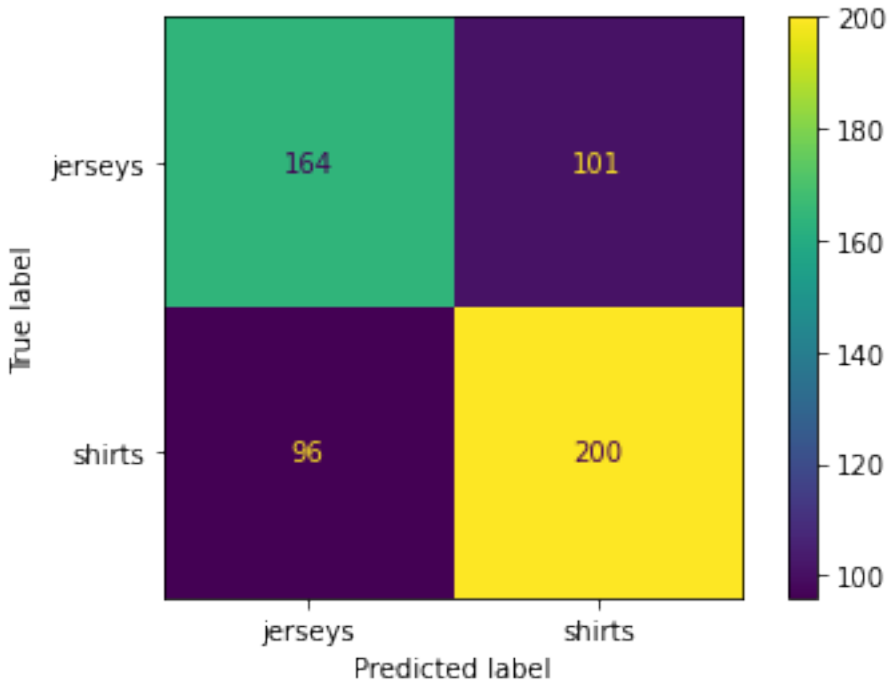
As we increase the C parameter, the model takes longer to train. Simultaneously,

We now fit the model and examine results,

[ ]:
```
best_params  = clf.best_estimator_.get_params()

svm_pca = svm.LinearSVC(
    **best_params)

start = time()
svm_pca.fit(X_pca, y_train.ravel())
stop = time()

pca_time = round(stop - start, 2)
```

[ ]:
```
# Linear transform test data with principal components
X_test_pca = pca.transform(X_test)

# Score PCA-linear SVM and show training time
print(
    "Test Accuracy Score:",  accuracy_score(y_test.ravel(), svm_pca.
 ↪predict(X_test_pca)), '\n',
    "Train Accuracy Score:", accuracy_score(y_train.ravel(), svm_pca.
 ↪predict(X_pca)), '\n',
    "Training Time:", f'{pca_time}s'
)

# Show confusion matrix
show_cm(y_test, svm_pca, X_test_pca)
```

```
Test Accuracy Score: 0.6488413547237076
 Train Accuracy Score: 0.6345381526104418
 Training Time: 0.08s
```



With substantially smaller training data, the PCA-model trains in just 0.08s. The close alignment between test and training accuracy (64.9% and 63.5% respectively) indicates that the model is not overfit and presumably learned good discriminants between jerseys and shirts.

Specifically, it seems to classify Shirts (67.6%) better than Jerseys (61.8%) in the test set. These performance results on the test set outperform the basic SVC, both in terms of mean accuracy (64.9% vs 55.6%) and individual class predictions.

0.4 LDA

The LDA is a *supervised* form of dimensionality reduction by incorporating class labels into the reduction process. By assuming a linear boundary to separate classes modeled by Gaussian densities, the technique is, in a sense, a classifier. At the same time, it can also be viewed as an extreme compressor, reducing the dimensionality to `1 - n_classes`, which here is just 1,

[ ]:
```
# Transform to LDA representation
lda = LinearDiscriminantAnalysis()
X_lda = lda.fit_transform(X_train, y_train.ravel())

X_lda.shape
# print(pd.DataFrame(np.hstack((X_lda[0:5], y_train[0:5]))).to_latex())
```

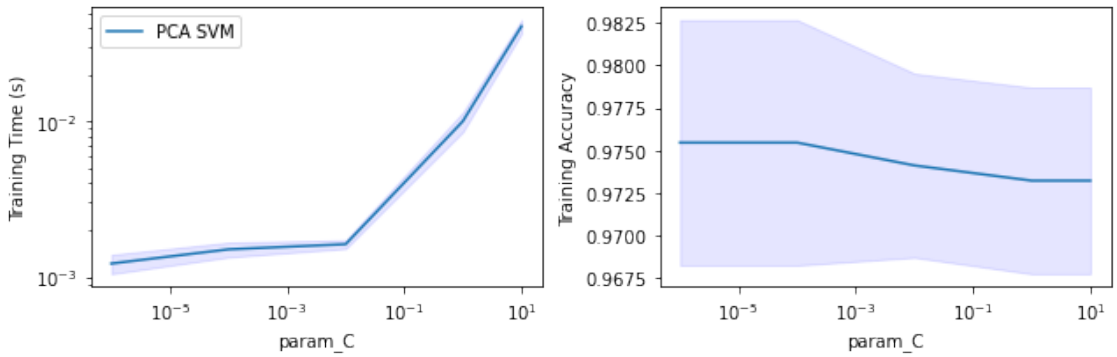
[ ]:
```
(2241, 1)
```

Sampling the first few LDA-representations of training images against their class labels, we can see that positive representations correspond to 1, while negative representations correspond to 0,

Sample	LDA-Representation	Class Label
0	4.405935	1.0
1	4.405935	1.0
2	-4.847767	0.0
3	-4.847767	0.0
4	4.405935	1.0

We will now fit the final SVC using the LDA representation,

[ ]:
```
# Parameter Search for LDA
params = {}
params['C'] = (1e-6, 1e-4, 1e-2, 1, 10)
params['tol'] = [1e-6,1e-4]
params['max_iter'] = [10000]

clf_lda = GridSearchCV(svm.LinearSVC(), params, scoring = 'accuracy')
clf_lda.fit(X_lda, y_train.ravel())

plot_paramsearch(clf_lda)
```



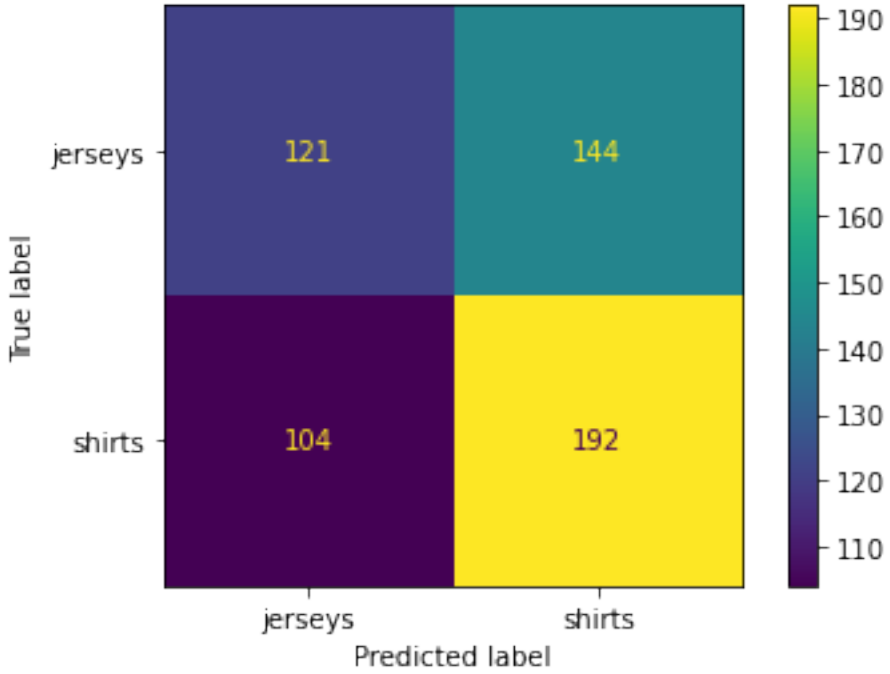
[ ]:
```
# Use PCA rep to fit SVC
best_params  = clf_lda.best_estimator_.get_params()

svm_lda  = svm.LinearSVC(**best_params)
svm_lda.fit(X_lda, y_train.ravel())
```

[ ]:
```
LinearSVC(C=1e-06, max_iter=10000, tol=1e-06)
```

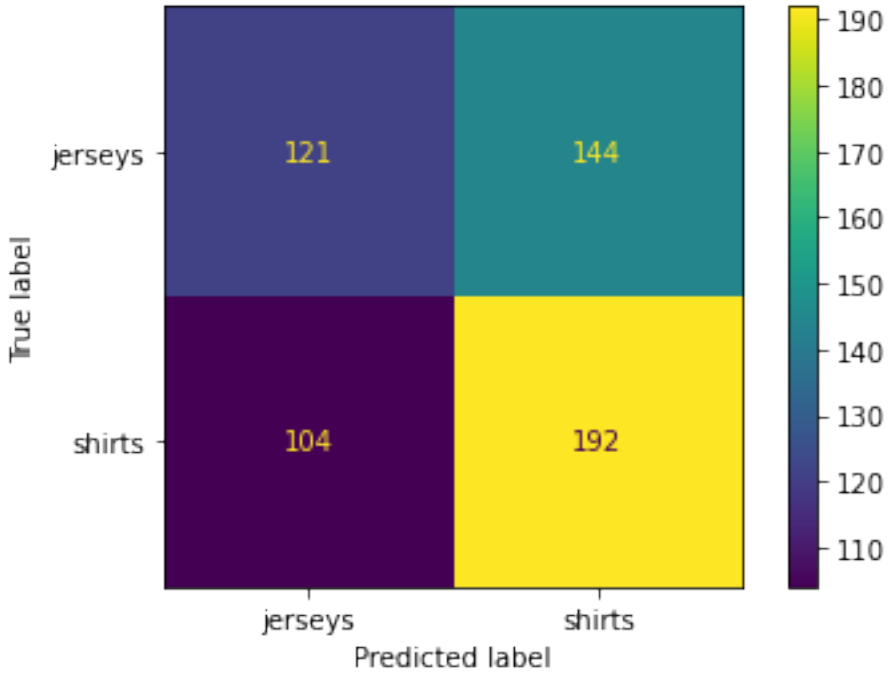
[ ]:
```
# Score LDA representation
X_test_lda = lda.transform(X_test)

print(
    accuracy_score(y_test, svm_lda.predict(X_test_lda))
```

```
)

# Show confusion matrix
show_cm(y_test, svm_lda, X_test_lda)
```

```
0.5579322638146168
```

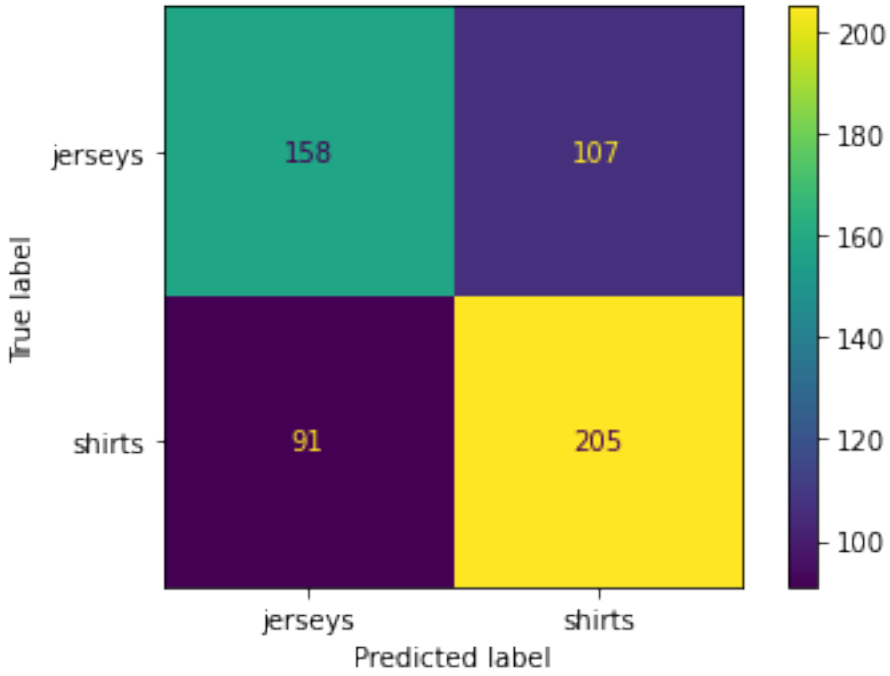


[ ]:



[ ]:
```
show_cm()
```

[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb69d72fbb0>
```



[ ]:
```
## Imports

from glob import glob
import numpy as np
import pandas as pd

from time import time

from tensorflow.keras.preprocessing import image
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import svm


from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
↪accuracy_score

import matplotlib.pyplot as plt
```

[ ]:
```
## Plotting/ Misc Functions (run first)

def plot_paramsearch(clf):
    results = pd.DataFrame(clf.cv_results_)
    param_results = results.groupby('param_C').mean()

    fig = plt.figure(figsize=(9,3), tight_layout = True)

    plt.subplot(121)
    ax = param_results["mean_fit_time"].plot(label = "PCA SVM")

    ax.fill_between(
        param_results.index,
        param_results["mean_fit_time"] + param_results["std_fit_time"],
        param_results["mean_fit_time"] - param_results["std_fit_time"],
        alpha = 0.1,
        color = "blue"
    )

    ax.set_yscale('log')
    ax.set_xscale('log')
```

```
    ax.set_ylabel("Training Time (s)")

    #ax.semilogx(C, training_time, label = "Basic SVM")
    ax.legend()

    plt.subplot(122)

    ax2 = param_results['mean_test_score'].plot(label = "PCA SVM")
    ax2.fill_between(
        param_results.index,
        param_results["mean_test_score"] + param_results["std_test_score"],
        param_results["mean_test_score"] - param_results["std_test_score"],
        alpha = 0.1,
        color = "blue"
    )
    ax2.set_xscale('log')
    ax2.set_ylabel("Training Accuracy")

    plt.show()

def show_cm(y, model, x):
    cm = confusion_matrix(y, model.predict(x))

    ConfusionMatrixDisplay(
        confusion_matrix = cm,
        display_labels=LABELS).plot()
```

[ ]: