CS156 Machine Learning Fashionista 2.0

April 16, 2020

1 Machine Learning Fashionista 2.0

I this assignment we want to compare SVMs and deep neural networks

[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 Support Vector Classifier

1.1.1 Data Loading and Processing

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

jerseys = glob('/Users/bellabuchanan/Downloads/jerseys/*')
jerseys.pop(500) #this was an int and not a string for some reason so removed␣
 ↪it
shirts = glob('/Users/bellabuchanan/Downloads/shirts/*')
shirts.pop(1469) #this was an int and not a string for some reason so removed␣
 ↪it
```

[2]:
```
'/Users/bellabuchanan/Downloads/shirts/n04197391_60.JPEG'
```

[3]:
```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((50,50)) #made images smaller to reduce training time

    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]
```

[4]:
```
data = [(img_to_array(filename),'shirts',filename) for filename in shirts] + \
            [(img_to_array(filename),'jerseys',filename) for filename in␣
 ↪jerseys]
```

[5]:
```
#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])
```

[6]:
```
#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))
```

```
2240
560
```

[7]:
```
#make the labels into zeros and ones
y_train_label = np.array([1 if label == 'jerseys' else 0 for label in y_train])
y_test_label = np.array([1 if label == 'jerseys' else 0 for label in y_test])
```

1.1.2 Linear Kernel

[8]:
```
#Create and fit model
clf = svm.SVC(kernel = 'linear')
clf.fit(X_train, y_train_label)
```

[8]:
```
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)
```

[9]:
```
print("The training accuracy is:", clf.score(X_train, y_train_label))
```

```
The training accuracy is: 0.9767857142857143
```

[10]:
```
print("The testing accuracy is:", clf.score(X_test, y_test_label))
```

```
The testing accuracy is: 0.5625
```

1.1.3 RBF Kernel

[11]:
```
rbf_svc = svm.SVC(kernel='rbf', gamma = 'auto')
rbf_svc.fit(X_train, y_train_label)
```

[11]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

[12]:
```
print("The training accuracy is:", rbf_svc.score(X_train, y_train_label))
```

```
The training accuracy is: 0.9767857142857143
```

[13]:
```
print("The testing accuracy is:", rbf_svc.score(X_test, y_test_label))
```

```
The testing accuracy is: 0.525
```

1.1.4 2nd Degree Polynomial Kernel

[14]:
```
poly_svc = svm.SVC(kernel='poly',degree=2,gamma = 'auto')
poly_svc.fit(X_train, y_train_label)
```

[14]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=2, gamma='auto', kernel='poly',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

[15]:
```
print("The training accuracy is:", poly_svc.score(X_train, y_train_label))
```

```
The training accuracy is: 0.9767857142857143
```

[16]:
```
print("The testing accuracy is:", poly_svc.score(X_test, y_test_label))
```

```
The testing accuracy is: 0.6125
```

We see that the best model for the SVM is the 2nd degree polynomial. We have the best testing accuracy of the 3 options. It makes sense to use this kernel becuase it is assumed that the data is not well suited to a first order linear separability kernel because jerseys and shirts have similar characteristics. A drop from 97% accuracy in training to 61% accuracy is the test set is quite high and suggests that the model is not as high performing on unseen data.

1.2 Deep Neural Networks

We now want to apply transfer learning to this problem. Transfer learning means we will reuse an existing model (VGG 16) to solve a new problem (classify jerseys and shirts)

[45]:
```
#Step 1, import libraries
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
from keras import backend as k
import keras
```

We use the VGG16 model's architecture which has been trained on thousands of images online becsause we think it will be useful in creating separability in our data of jerseys and jackets. We remove the top layer, because we are also going to use a different neural network to do the last bit of classifying after the VGG16 neural network has transformed our data into a very useful format.

[18]:
```
# load the original model
orig_model = VGG16(weights = "imagenet", include_top=False, input_shape = (224,␣
 ↪224, 3))
```

[19]:
```
#Examise layers of model
orig_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
```

```
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

[28]:
```
'''
Function to process an image using the neural network vgg16 model
excluding the top layer
'''
def process(datum):
    image = load_img(datum, target_size=(224, 224))

    #covert to np array
    image = img_to_array(image)
    #reshape data
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    #preprocess for model
    image = preprocess_input(image)
    #extract features
    features = orig_model.predict(image) #extract the features using the vgg16␣
 ↪model
    return features
```

[29]:
```
#extract jersey features
j = []
for i in jerseys:
    j.append(process(i))
```

[30]:
```
#extract jersey features
s = []
for i in shirts:
    s.append(process(i))
```

[31]:
```
#reshape data and put into x ans ys
j = np.asarray(j)
s = np.asarray(s)
j = j.reshape(len(j), 7, 7, 512) #look at shape of last layer of orig model
s = s.reshape(len(s), 7, 7, 512)


#Jerseys are zeroes and shirts are ones
jersey_labels = np.zeros(len(s))
shirt_labels = np.ones(len(j))

X = np.append(j,s, axis = 0)
y = np.append(jersey_labels,shirt_labels, axis = 0)
```

[40]:
```
#split dataset into 80% training and 20% testing
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)
```

[56]:
```
#Create a simple neural network to classify data
#Output is a sigmoid layer with 1 unit because we are classifying with 1s and␣
 ↪zeros
#and we will get a probability of being a jersey
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Flatten(input_shape=(7,7,512)))
model.add(Dense(units=64, activation='relu', input_dim=7*7*512))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

[57]:
```
model.summary()
```

```
Model: "sequential_3"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                1605696
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,607,809
Trainable params: 1,607,809
Non-trainable params: 0
_________________________________________________________________
```

1.2.1 Loss function

I chose a binary cross entropy loss function because this is a loss function that we use when each image has a binary variable as its label. Each photo belongs to only one class.

[60]:
```
#compile the model
model.compile(loss = "binary_crossentropy", optimizer = 'adam',␣
 ↪metrics=["accuracy"])
```

[61]:
```
#Train Model
#I use 10 epochs because I did not see improvement afterwards and it took a␣
 ↪while to run
history=model.fit(X_train, y_train,epochs=10)
```

```
Epoch 1/10
2240/2240 [==============================] - 2s 768us/step - loss: 1.4062 -
accuracy: 0.6353
Epoch 2/10
2240/2240 [==============================] - 2s 707us/step - loss: 0.5625 -
accuracy: 0.8165
Epoch 3/10
2240/2240 [==============================] - 2s 694us/step - loss: 0.2882 -
accuracy: 0.91430s - loss: 0.2780 - accuracy:
Epoch 4/10
2240/2240 [==============================] - 2s 888us/step - loss: 0.3395 -
accuracy: 0.9254
Epoch 5/10
2240/2240 [==============================] - 2s 757us/step - loss: 0.2220 -
accuracy: 0.9482
Epoch 6/10
2240/2240 [==============================] - 2s 725us/step - loss: 0.2702 -
accuracy: 0.94730s - loss: 0.1749 - accu
Epoch 7/10
```

```
2240/2240 [==============================] - 2s 780us/step - loss: 0.2078 -
accuracy: 0.9567
Epoch 8/10
2240/2240 [==============================] - 2s 851us/step - loss: 0.1863 -
accuracy: 0.9598
Epoch 9/10
2240/2240 [==============================] - 2s 777us/step - loss: 0.2428 -
accuracy: 0.9438
Epoch 10/10
2240/2240 [==============================] - 2s 793us/step - loss: 0.1897 -
accuracy: 0.9563
```

[51]:
```
#these are all probabilities
y_pred = model.predict(X_test)
```

[66]:
```
scores = model.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
accuracy: 73.04%
```

1.3 Comparison of SVMs and Deep Neural Networks

We see that the neural network outperforms the SVM in terms of accuracy. In the NN, the training accuracy was 95.63% after 10 epochs (and stayed fairly similar after that). The testing accuracy was 73.04%. This decrease from training to test set accuracy is lower than that of the SVM discussed earlier (22.59% vs 36.42%). This means that the NN is better performing on unseen data! This makes sense since we applied transfer learning to a problem that had been really well solved using vgg16. The SVM accuracy could perhaps be improved with a different kernel but we tried 3 different ones so we gave it a fair chance.

I think what we should also note the differences in the mechanisms of SVMs and NNs. SVMs are algorithms which find hyperplanes to separate data. To find the hyperplane, we try to maximise the margins between the hyperplane and the data. They use kernels to make linearly separable data separable. NNs in contrast, have various layers including convolutional layers convolve learned features with input data. In this case, we used transfer learning, and used a model with layers that had done a good job at extracting features from imagenet images.