Exercises3

November 13, 2020

1 Exercise set 3

1.1 Answered to all questions (1-5)

1.2 Question 1

$$S_W = C_0 + C_1 = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 3 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 6 & -1 \\ -1 & 2 \end{pmatrix}$$

$$S_W^{-1} = \frac{1}{6*2-1} \begin{pmatrix} 6 & -1 \\ -1 & 2 \end{pmatrix} = \begin{pmatrix} \frac{2}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{6}{11} \end{pmatrix}$$

Since the scale of projection vector $\mathbf{w}$ dosen't matter, we can write

$$\mathbf{w} = S_W^{-1}(\boldsymbol{\mu_1} - \boldsymbol{\mu_0}) = \begin{pmatrix} 6 & -1 \\ -1 & 2 \end{pmatrix} (\begin{pmatrix} 1 \\ 2 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \end{pmatrix}) = \begin{pmatrix} 1 \\ 6 \end{pmatrix}$$

1.3 Question 2

When classification problem is defined as a likelihood ratio test, I got following tresholds for x (Derivation of tresholds is omitted, since it is long and messy): Classify projected sample x in class 1 if

$$c - \sqrt{\frac{2log(\sigma_1\sigma_2^{-1}) - \sigma_2^{-2}\mu_2^2 - \sigma_1^{-2}\mu_1^2 + \frac{(\sigma_2^{-2}\mu_2 - \sigma_1^{-2}\mu_1)^2}{\sigma_2^{-2} - \sigma_1^{-2}}}{\sigma_2^{-2} - \sigma_1^{-2}}} < x < c + \sqrt{\frac{2log(\sigma_1\sigma_2^{-1}) - \sigma_2^{-2}\mu_2^2 - \sigma_1^{-2}\mu_1^2 + \frac{(\sigma_2^{-2}\mu_2 - \sigma_1^{-2}\mu_1)^2}{\sigma_2^{-2} - \sigma_1^{-2}}}{\sigma_2^{-2} - \sigma_1^{-2}}}$$

where

$$c = \frac{\sigma_2^{-2}\mu_2 - \sigma_1^{-2}\mu_1}{\sigma_2^{-2} - \sigma_1^{-2}}$$

$$\mu_1 = \mathbf{w^T}\boldsymbol{\mu_0}$$

$$\sigma_1^2 = \mathbf{w^T C_0 w}$$

$$\mu_2 = \mathbf{w^T}\boldsymbol{\mu_1}$$

$$\sigma_2^2 = \mathbf{w^T C_1 w}$$

In the context of Question 1, numeric values of these parameters are:

$$\mu_1 = \mathbf{w^T}\boldsymbol{\mu_0} = \mathbf{7}$$

$$\sigma_1^2 = \mathbf{w^T C_0 w = 39}$$

$$\mu_2 = \mathbf{w^T \boldsymbol{\mu}_1 = 13}$$

$$\sigma_2^2 = \mathbf{w^T C_1 w = 27}$$

By applying these to the formula given above, obtained tresholds are:

$$9.30937 < x < 43.6906$$

By using the $\mathbf{w}$ derived in Q1, projected x is:

$$x = \begin{pmatrix} 1 & 6 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 13$$

Therefore, the point $\mathbf{x}$ is classified to class 1.

1.4 Question 3

[3]:
```
# Imports
import time
import numpy as np
from skimage.io import imread_collection
from skimage.transform import rescale, resize, downscale_local_mean
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import cv2
from sklearn.metrics import accuracy_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler

root = '/home/tuomas/Python/DATA.ML.200/Ex3/'
images = []
labels = []
for i in range(0,9):
    fn = '0000{}/*.jpg'.format(i)
    print(fn)
    imgs = imread_collection(root + fn)
    images.append(np.array(imgs, dtype='object'))
    labels.append( np.ones(len(imgs)) * i )

#images = np.array(images, dtype='object')
images = np.concatenate(images)
labels = np.concatenate(labels).astype('uint8')
```

```
00000/*.jpg
00001/*.jpg
00002/*.jpg
00003/*.jpg
00004/*.jpg
00005/*.jpg
00006/*.jpg
00007/*.jpg
00008/*.jpg
```

[4]:
```
# Preprocess the images
imgs_processed = []
scaler = MinMaxScaler()
for img in images:
    # Resize & vectorize the image
    img = cv2.resize(img, (32,32)).ravel()
    # Scale sample to (0,1)
    # Since MinMaxScaler scales data featurewise, I transposed the row vector␣
↪into column vector
    # so the samplewise scaling is performed
    img_T = img[...,None]
    scaler.fit(img_T)
    img_T = scaler.transform(img_T)

    imgs_processed.append(img_T.ravel())

imgs_processed = np.array(imgs_processed).astype('float32')
```

[5]:
```
# Create training and testing sets
trainX, testX, trainY, testY = train_test_split(imgs_processed,
                                                labels,
                                                test_size=0.3)
```

[4]:
```
# Train, test & evaluate given models
models = [KNeighborsClassifier(n_neighbors=3),
          LinearDiscriminantAnalysis(solver='svd'),
          LogisticRegression(max_iter=10000),
          SVC(kernel='linear'),
          SVC(kernel='rbf'),
          RandomForestClassifier(n_estimators=20)
          ]

accuracy = []
tr_time = []
tst_time = []
for model in models:
    # Training
```

```
    print('Training {} ...'.format(model.__class__.__name__))
    start = time.time()
    model.fit(trainX, trainY)
    tr_time.append( time.time() - start )
    # Testing
    print('Testing {} ...'.format(model.__class__.__name__))
    start = time.time()
    predY = model.predict(testX)
    tst_time.append( time.time() - start )
    # Evaluating
    print('Evaluating {} ...'.format(model.__class__.__name__))
    accuracy.append( accuracy_score(testY, predY) )
```

```
Training KNeighborsClassifier …
Testing KNeighborsClassifier …
Evaluating KNeighborsClassifier …
Training LinearDiscriminantAnalysis …
Testing LinearDiscriminantAnalysis …
Evaluating LinearDiscriminantAnalysis …
Training LogisticRegression …
Testing LogisticRegression …
Evaluating LogisticRegression …
Training SVC …
Testing SVC …
Evaluating SVC …
Training SVC …
Testing SVC …
Evaluating SVC …
Training RandomForestClassifier …
Testing RandomForestClassifier …
Evaluating RandomForestClassifier …
```

[5]:
```
# Print the statistics
model_names = ['3-NN','LDA','LogReg','SVM linear','SVM rbf','Random forest']
print('Training set size (batch) = {}'.format(trainX.shape[0]))
print('Test set size = {}\n'.format(testX.shape[0]))
for i in range(6):
    print('Results for {}'.format(model_names[i]))
    print('----------------------')
    print('Accuracy is {} %'.format(round(accuracy[i]*100, 2)))
    print('Training time / batch {} s'.format(round(tr_time[i], 2)))
    print('Test time / sample {} ms\n'.format(round(tst_time[i]*1000/testX.
 ↪shape[0], 3)))
```

```
Training set size (batch) = 6237
Test set size = 2673

Results for 3-NN
```

```
----------------------
Accuracy is 88.78 %
Training time / batch 1.65 s
Test time / sample 18.116 ms

Results for LDA
----------------------
Accuracy is 80.96 %
Training time / batch 11.56 s
Test time / sample 0.006 ms

Results for LogReg
----------------------
Accuracy is 94.28 %
Training time / batch 32.31 s
Test time / sample 0.006 ms

Results for SVM linear
----------------------
Accuracy is 95.06 %
Training time / batch 32.63 s
Test time / sample 5.539 ms

Results for SVM rbf
----------------------
Accuracy is 91.99 %
Training time / batch 70.33 s
Test time / sample 11.673 ms

Results for Random forest
----------------------
Accuracy is 91.81 %
Training time / batch 3.52 s
Test time / sample 0.007 ms
```

1.5 Question 4

By previous question, it seems that SVM linear had the best performance

[6]:
```python
# Concatenate train and test set from previous exercise
from sklearn.model_selection import cross_val_score
trainX2 = np.concatenate((trainX, testX))
trainY2 = np.concatenate((trainY, testY))
model = SVC(kernel='linear')
cv_scores = cross_val_score(model, trainX2, trainY2, cv=5, verbose=2)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV]  …
[CV] … , total=  51.5s
[CV]  …

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   51.5s remaining:    0.0s

[CV] … , total=  51.0s
[CV]  …
[CV] … , total=  51.1s
[CV]  …
[CV] … , total=  51.1s
[CV]  …
[CV] … , total=  51.0s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  4.3min finished
```

[8]:

```
print('Mean of accuracies = {}'.format(np.mean(cv_scores)))
print('Standard deviation of accuracies = {}'.format(np.std(cv_scores)))
```

```
Mean of accuracies = 0.9584736251402919
Standard deviation of accuracies = 0.0015470312853075455
```

1.6 Question 5

[11]:

```
# Resize all images to 32 x 32
images_resized = []
for img in images:
    images_resized.append(cv2.resize(img, (32,32)))

images_resized = np.array(images_resized)
```

[12]:

```
# Create training and testing sets.
from tensorflow.keras.utils import to_categorical
trainX, testX, trainY, testY = train_test_split(images_resized,
                                                labels,
                                                test_size=0.15)


# One-hot encoding
trainY = to_categorical(trainY, num_classes=9)
testY_cat = to_categorical(testY, num_classes=9)
```

[13]:

```
# Build the model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import␣
 ↪Conv2D,BatchNormalization,Dropout,Flatten,Dense

layers=[Conv2D(filters=32, kernel_size = 3, activation='relu', input_shape =␣
 ↪(32,32,3)),
        BatchNormalization(),
```

```
        Conv2D(filters=32, kernel_size = 3, activation='relu'),
        BatchNormalization(),
        Conv2D(32, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu'),
        BatchNormalization(),
        Dropout(0.4),
        Conv2D(64, kernel_size = 3, activation='relu'),
        BatchNormalization(),
        Conv2D(64, kernel_size = 3, activation='relu'),
        BatchNormalization(),
        Conv2D(64, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu'),
        BatchNormalization(),
        Dropout(0.4),
        Conv2D(128, kernel_size = 4, activation='relu'),
        BatchNormalization(),
        Flatten(),
        Dropout(0.4),
        Dense(9, activation='softmax')]

cnn_model = Sequential(layers)
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        25632
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 12, 12, 64)        256
_________________________________________________________________
```

```
conv2d_4 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 64)          102464
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 64)          256
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 128)         131200
_________________________________________________________________
batch_normalization_6 (Batch (None, 2, 2, 128)         512
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 9)                 4617
=================================================================
Total params: 331,145
Trainable params: 330,313
Non-trainable params: 832
_________________________________________________________________
```

[14]:

```
# Train the model
cnn_model.fit(trainX, trainY, epochs=10, batch_size=64, verbose=1)
```

```
Epoch 1/10
119/119 [==============================] - 6s 48ms/step - loss: 1.4053 -
accuracy: 0.5504
Epoch 2/10
119/119 [==============================] - 1s 10ms/step - loss: 0.3055 -
accuracy: 0.8946
Epoch 3/10
119/119 [==============================] - 1s 10ms/step - loss: 0.1569 -
accuracy: 0.9477
Epoch 4/10
119/119 [==============================] - 1s 10ms/step - loss: 0.0982 -
accuracy: 0.9663
Epoch 5/10
119/119 [==============================] - 1s 10ms/step - loss: 0.0723 -
accuracy: 0.9781
Epoch 6/10
119/119 [==============================] - 1s 10ms/step - loss: 0.0595 -
accuracy: 0.9795
Epoch 7/10
119/119 [==============================] - 1s 10ms/step - loss: 0.0536 -
```

```
accuracy: 0.9814
Epoch 8/10
119/119 [==============================] - 1s 10ms/step - loss: 0.0438 -
accuracy: 0.9852
Epoch 9/10
119/119 [==============================] - 1s 10ms/step - loss: 0.0302 -
accuracy: 0.9901
Epoch 10/10
119/119 [==============================] - 1s 10ms/step - loss: 0.0314 -
accuracy: 0.9880
```

[14]:
```
<tensorflow.python.keras.callbacks.History at 0x7fea1484cf10>
```

[15]:
```
# Evaluate the model
test_loss, test_acc = cnn_model.evaluate(testX, testY_cat, verbose=2)
print("Accuracy of CNN = {}".format(test_acc))
```

```
42/42 - 2s - loss: 0.0337 - accuracy: 0.9933
Accuracy of CNN = 0.9932684898376465
```