Advanced Classification using Machine Learning in HealthCare

Abstract

This lab is dedicated to application of an advanced analysis of Big Data to the spread of COVID19 in the world. We will study how to make image recognition of a chest X-ray to provide a diagnosis forecast.

Introduction

X-rays are widely used in medical practice. They can be used to identify various diseases.
However, a diagnosis depends on a doctor's experience, which can lead to improper treatment. Modern methods of artificial intelligence and pattern recognition make it possible to create expert systems that allow you to establish a diagnosis automatically.

This lab will show you how to upload images, transform them, and determine the basic features that underlie diseases classification.

Two different approaches to the classification of images (diseases) will be shown:

1. Different classical methods and their comparison
2. Convolutional Neural Networks.

Materials and methods

In this lab, we will learn the basic methods of images classifications. The laboratory consists of four stages:

- Download and preliminary transformation of images
- Creating image features
- Comparing different classical classification methods
- Building and fitting Convolutional Neural Network

Prerequisites

- Python
- os
- numpy
- glob
- SeaBorn
- Matplotlib
- mahotas
- keras
- scikit-learn

- pandas

Objectives

After completing this lab, you will be able to:

- Download and transform images.
- Create features of images.
- Build different classification models.
- Build CNN models.
- Build a diagnosis based on X-ray photos.

Download and preliminary transformation of images

Required libraries installation

We need to install additional libraries and upgrade existing ones in the lab.

```
# conda install matplotlib >= 3.4 # conda

config --add channels conda-forge pip

install mahotas conda install scikit-

learn

conda install -c conda-forge tensorflow --yes
conda install -c conda-forge keras --yes
```

Required libraries import

Here we are going to use Mahotas for image processing and Keras library for creating our CNN model and its training. We will also use Matplotlib and Seaborn for visualizing our dataset to gain a better understanding of the images we are going to handle. We will also use libraries os and glob to work with files and folders. NumPy will be applied for arrays of images. Scikit-Learn will be used for classical classification models. And we will take Pandas for present comparison of classifiers.

```
import mahotas as mh
import seaborn as sns
```

```
from matplotlib import pyplot as plt
from glob import glob import os
import numpy as np import pandas as
pd

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

#Classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF from
sklearn.tree import DecisionTreeClassifier from
sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import
QuadraticDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import plot_confusion_matrix
```

Data loading

```
import skillsnetwork

await skillsnetwork.prepare("https://cf-courses-
data.s3.us.cloudobject-storage.appdomain.cloud/data-science-in-health-
care-advancedmachine-learning-classification/LargeData/Covid19-
dataset.zip", overwrite=True)
await skillsnetwork.prepare("https://cf-courses-
data.s3.us.cloudobject-storage.appdomain.cloud/data-science-in-health-
care-advancedmachine-learning-classification/NN.zip", overwrite=True)
```

Images have to be of the same size for classification. In order to achieve this, let's create a global variable that will determine the size (height and width) for image resizing. Both are 224 in our case.

```
IMM_SIZE = 224
```

For convenience, we create a function that downloads and displays all the pictures from a specified directory. In order to classify images, all pictures should be placed in subdirectories. The names of these subdirectories are actually class names. In our case, the images are X-rays which should be placed into subfolders with the names of diagnoses. For example, a COVID subfolder is to contain X-rays of people with this disease. First of all, we should create a list of subfolders, which is a list of possible disease classes. In our case, it will be: Normal, COVID, Viral Pneumonia.

Next, we need to create a list of all images. After that, let's download and preprocess all the images:

1. Download by mahotas.imread()
2. It is necessary to resize the images to (IMM_SIZE x IMM_SIZE). If an image is graycolored it is presented as a 2D matrix: (high, width). jpg and png images are 3D (high, width, 3 or 4). To do this, we should use: mahotas.resize_to()
3. If the third parameter of an image shape equals 4, it means alpha channel. We can remove it using slice image[:,:,:3].
4. Then we need to transform all the images to gray 2D format by: mahotas.colors.rgb2grey()

The function returns an array of tuples [image, class name].

```
def get_data(folder):

    class_names = [f for f in os.listdir(folder) if not
f.startswith('.')] # ctreate a list of SubFolders
data = []
    # class_names = ['Covid', 'Normal', 'Viral Pneumonia']
print(class_names)     for t, f in enumerate(class_names):
images = glob(folder + "/" + f + "/*") # create a list of files
        print("Downloading: ", f)
        fig = plt.figure(figsize = (50,50))         for im_n, im in
enumerate(images):             plt.gray() # set grey colormap of
images             image = mh.imread(im)             if
len(image.shape) > 2:                 image = mh.resize_to(image,
[IMM_SIZE, IMM_SIZE, image.shape[2]]) # resize of RGB and png
images             else:                 image =
mh.resize_to(image, [IMM_SIZE, IMM_SIZE]) # resize of grey images
if len(image.shape) > 2:                 image =
mh.colors.rgb2grey(image[:,:,:3], dtype = np.uint8)  # change of
colormap of images alpha chanel delete
plt.subplot(int(len(images)/5)+1,5,im_n+1) # create a table of
images
            plt.imshow(image)
data.append([image, f])
plt.show()     return
np.array(data)
```

For training and testing, all the pictures should be divided into training and test groups and located in separate folders. Let's upload all the images to the corresponding arrays.

```
d = "Covid19-dataset/train"
train = get_data(d)

d = "Covid19-dataset/test"
val = get_data(d)

print("Train shape", train.shape) # Size of the training DataSet
print("Test shape", val.shape) # Size of the test DataSet
print("Image size", train[0][0].shape) # Size of image
```

As you can see, the training DataSet consists of 251 images and the test one consists of 66 images. All the images are in grey 2D (224x224) format.

Data visualization

Let's visualize our data and see what exactly we are working with. We use Seaborn to plot the number of images in all the classes. You can see what the output looks like.

```
l = [] for i in
train:
    l.append(i[1])
sns.set_style('darkgrid')
sns.countplot(l)
```

Let us also visualize the first image from the Viral Pneumonia and Covid classes in the training DataSet:

```
plt.figure(figsize = (5,5))
plt.imshow(train[np.where(train[:,1] == 'Viral Pneumonia')[0][0]][0])
plt.title('Viral Pneumonia')

plt.figure(figsize = (5,5))
plt.imshow(train[np.where(train[:,1] == 'Covid')[0][0]][0])
plt.title('Covid')
```

Image features creation

To classify objects, you need to transform the data set so that the input is a set of features and the output is an object class. An image is a matrix of pixels. Each pixel is a color. Therefore, it is impossible to submit a bare picture into a classical classifier's input. It is necessary to turn each picture into a set of certain features.

Mahotas makes it easy to calculate features of an image. The corresponding functions are found in the mahotas.features submodule. The Haralick set of texture features is well known. Like many image processing algorithms, it is named for its inventor. The features are based on textures, that is, they distinguish between structured and unstructured images, as well as various repetitive structures. With the help of Mahotas, these features are calculated very easily: haralick_features = mh.features.haralick (image) haralick_features_mean = np.mean (haralick_features, axis = O) haralick_features_all = np.ravel (haralick_features) The mh.features.haralick function returns a 4 x 13 array wich should be transformad into 1D by NumPy.ravel(). The first dimension is the four possible directions in which the features are calculated (vertical, horizontal, and two diagonals). If we are not interested in any particular direction,

then we can average the features in all directions (in the code above, this variable is called haralick_features_mean). Alternatively, you can use all the characteristics individually (variable haralick_features_all). The choice depends on the properties of a particular dataset. We decided that the vertical and horizontal features should be stored separately in our case, so we use haralick_features_all.

We should make a function for the DataSet features creation.

```
def create_features(data):     features = []     labels = []
for image, label in data:
features.append(mh.features.haralick(image).ravel())
        labels.append(label)
features = np.array(features)
labels = np.array(labels)
return (features, labels)

features_train, labels_train = create_features(train)
features_test, labels_test = create_features(val)
```

Comparing different classical classification methods

If we want to compare some classificators, we should use a Pipeline. A pipeline helps to chain multiple estimators into a single one. This is useful as there is often a fixed number of steps in data processing, for example, feature selection, normalization and classification. A pipeline serves multiple purposes here:

You only have to call fit() once to evaluate a whole sequence of estimators.

You can grid search over parameters of all estimators in the pipeline at once.

Pipelines help to avoid leaking statistics from your test data into the trained model in crossvalidation by ensuring that the same samples are used to train the transformers and predictors. All estimators in a pipeline, except the last one, should be transformers (i.e. should have a transform method). The last estimator may be of any type (transformer, classifier, etc.).

The sklearn.pipeline module implements utilities to build a composite estimator, as a chain of transforms and estimators.

In order to test how it works, we will use LogisticRegression.

```
clf = Pipeline([('preproc', StandardScaler()), ('classifier',
LogisticRegression())])
```

```
clf.fit(features_train, labels_train)
scores_train = clf.score(features_train, labels_train)
scores_test = clf.score(features_test, labels_test)
print('Training DataSet accuracy: {: .1%}'.format(scores_train), 'Test
DataSet accuracy: {: .1%}'.format(scores_test))
plot_confusion_matrix(clf, features_test, labels_test)
plt.show()
```

As you can see, the results are not bad. Conflusion matrix shows us how many mistaken predictions we got. It allows us to check other classifiers and compare the results. We will test:

- Logistic Regression
- Nearest Neighbors
- Linear SVM
- RBF SVM
- Gaussian Process
- Decision Tree
- Random Forest
- Multi-layer Perceptron classifier
- Ada Boost
- Naive Bayes
- Quadratic Discriminant Analysis

```
names = ["Logistic Regression", "Nearest Neighbors", "Linear SVM",
"RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

classifiers = [
    LogisticRegression(),
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10,
max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]
scores_train = [] scores_test = [] for
name, clf in zip(names, classifiers):
print("Fitting:", name)
    clf  =  Pipeline([('preproc',  StandardScaler()),  ('classifier',
clf)])
    clf.fit(features_train, labels_train)
    score_train = clf.score(features_train, labels_train)
score_test = clf.score(features_test, labels_test)
scores_train.append(score_train)
scores_test.append(score_test)
```

Let's print the results as a table.

```
res = pd.DataFrame(index = names)
res['scores_train'] = scores_train
res['scores_test'] = scores_test
res.columns = ['Test','Train']
res.index.name = "Classifier accuracy"
pd.options.display.float_format = '{:,.2f}'.format
print(res)
```

You can see that the calculations are very fast and that Logistic Regression and Neural Network show the best result for the test DataSet.

Let's compare the results on a plot.

```
x = np.arange(len(names))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, scores_train, width, label='Train')
rects2 = ax.bar(x + width/2, scores_test, width, label='Test')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy of classifiers')
ax.set_xticks(x) plt.xticks(rotation =
90) ax.set_xticklabels(names)
ax.legend() fig.tight_layout()
plt.show()
```

Building and fitting Convolutional Neural Network

Required libraries import

We will use Keras library for creating and training our model.

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout

from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam import tensorflow as tf from
sklearn.metrics import classification_report,confusion_matrix
```

Data preprocessing and data augmentation

What is different about Convolutional Neural Networks is that we can submit images directly to the input. However, these images also require pre-processing.

In particular, it is necessary to normalize the pixels color, i.e. to normalize them from the range [0, 255) to [0, 1).

You also need to change the dimension of the input images because of Keras framework.

Image classes should be of numeric type instead of string.

The code below makes the necessary transformations.

```
x_train = []
y_train = []
x_val = []
y_val = []

for feature, label in train:
x_train.append(feature)
y_train.append(label)

for feature, label in val:
x_val.append(feature)
y_val.append(label)

# Normalize the data
x_train = np.array(x_train) / 255
x_val = np.array(x_val) / 255

# Reshaping input images
x_train = x_train.reshape(-1, IMM_SIZE, IMM_SIZE, 1)
x_val = x_val.reshape(-1, IMM_SIZE, IMM_SIZE, 1)

# Creating a dictionary of clases
lab = {} for i, l in
enumerate(set(y_train)):     lab[l]
= i
```

```
y_train = np.array([lab[l] for l in y_train])
y_val = np.array([lab[l] for l in y_val])

print("Shape of the input DataSet:", x_train.shape)
print("Shape of the output DataSet:", y_train.shape)
print("Dictionary of classes:", lab)
```

Data augmentation on the training data

We should perform data augmentation to fit our model better.

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the
dataset
        samplewise_center=False,  # set each sample mean to 0
featurewise_std_normalization=False,  # divide inputs by std of the
dataset
        samplewise_std_normalization=False,  # divide each input by
its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range
(degrees, 0 to 180)
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally
(fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically
(fraction of total height)
        horizontal_flip = True,  # randomly flip images

vertical_flip=False)  # randomly flip images datagen.fit(x_train)
```

Model defining

Let's define a simple CNN model with 3 Convolutional layers followed by max-pooling layers. A dropout layer is added after the 3rd maxpool operation to avoid overfitting.

```
model = Sequential()
model.add(Conv2D(32,1,padding="same", activation="relu",
input_shape=(IMM_SIZE,IMM_SIZE,1)))
model.add(MaxPool2D())

model.add(Conv2D(32, 1, padding="same", activation="relu"))
model.add(MaxPool2D())

model.add(Conv2D(64, 1, padding="same", activation="relu"))
model.add(MaxPool2D())
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(128,activation="relu")) model.add(Dense(3,
activation="softmax")) model.summary()
```

Let's compile the model now using Adam as our optimizer and SparseCategoricalCrossentropy as the loss function. We are using a lower learning rate of 0.000001 for a smoother curve.

```
opt = Adam(lr=0.000001)
model.compile(optimizer = opt , loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) ,
metrics = ['accuracy'])
```

Now, let's train our model for **2000** epochs. Admittedly, the fitting process is very slow. Therefore, we saved the fitted model to a file. To save time, we will upload the fitted model. If you would like, you can change the parameter **fitting to True** in order to refit the model.

```
fitting = False
fitting_save = False
epochs = 200 import
pickle

if fitting:     history =
model.fit(x_train,y_train,epochs = epochs ,
validation_data = (x_val, y_val), shuffle = True)     if
fitting_save:
    # serialize model to JSON         model_json =
model.to_json()         with open("model.json",
"w") as json_file:
json_file.write(model_json)         # serialize
weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")         with
open('history.pickle', 'wb') as f:
            pickle.dump(history.history, f)
with open('lab.pickle', 'wb') as f:
pickle.dump(lab, f)
# load model
from keras.models import model_from_json
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
# load weights into a new model
model.load_weights("model.h5")
```

```
with open('history.pickle', 'rb') as f:
history = pickle.load(f) print("Loaded
model from disk")
```

Results evaluation

We will plot our training and validation accuracy along with the training and validation loss.

```
acc = history['accuracy'] val_acc
= history['val_accuracy'] loss =
history['loss'] val_loss =
history['val_loss'] epochs_range
= range(epochs)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Let's see what the curve looks like. You can see that the accuracy of the training and validation sets is the same. The loss function of validation and training sets is stable. It means that our CNN is fitted well and can be used for classification.

We can print out the classification report to see the precision and accuracy using model.predict_classes() and classification_report().

Also we can create a confusion matrix. Unfortunately, Keras framework does not have plot_confusion_matrix() function. Therefore, we have to create it using Pandas and Seaborn.heatmap()

```
# Classification report
predictions = model.predict_classes(x_val)
predictions = predictions.reshape(1,-1)[0]
print(classification_report(y_val, predictions, target_names =
lab.keys()))

# Confusion matrix
cm = pd.DataFrame(confusion_matrix(y_val, predictions))
cm.index = ["Predicted " + s for s in lab.keys()]
```

```
cm.columns = ["True  " + s for s in lab.keys()]
print(cm)

sns.heatmap(confusion_matrix(y_val, predictions), annot=True,
xticklabels = list(lab.keys()), yticklabels =
list(lab.keys())) plt.xlabel("True labels")
plt.ylabel("Predicted labels") plt.show()

# Accuracy
z = model.predict_classes(x_train) == y_train
scores_train = sum(z+0)/len(z)
z = model.predict_classes(x_val) == y_val
scores_test = sum(z+0)/len(z)
print('Training DataSet accuracy: {: .1%}'.format(scores_train), 'Test
DataSet accuracy: {: .1%}'.format(scores_test))
```

As you can see, the CNN shows better results than classical models. However, fitting takes much longer.

And now, on your own, try to create a function that will establish a diagnosis based on the CNN.

```
def diagnosis(file):
# Download image
    ##YOUR CODE GOES HERE##

    # Prepare image to classification
    ##YOUR CODE GOES HERE##

    # Show image
    ##YOUR CODE GOES HERE##


    # Load model
    ##YOUR CODE GOES HERE##


    # Normalize the data
    ##YOUR CODE GOES HERE##


    # Reshape input images
##YOUR CODE GOES HERE##


    # Predict the diagnosis
    ##YOUR CODE GOES HERE##
```

```
    # Find the name of the diagnosis
    ##YOUR CODE GOES HERE##


    return diag

print ("Diagnosis is:",
diagnosis("Covid19-dataset/test/Covid/0120.jpg"))
print ("Diagnosis is:",
diagnosis("Covid19-dataset/test/Normal/0105.jpeg"))
print ("Diagnosis is:", diagnosis("Covid19-dataset/test/Viral
Pneumonia/0111.jpeg"))
```

Conclusions

This lab has revealed for us how to create an expert system that allows us to obtain diagnosis based on X-Rays images, using different classificators. This principles can be used for any type of X-Rays, not only for COVID diagnostics.

During this laboratory work, we have explored images downloading and transforming. We have learned how to extract features of images and build/fit/test/compare sets of classificators. Also we got to know how to create and fit Convolutional Neural Networks. We have compared accuracy of different classificators.

Authors

Yaroslav Vyklyuk, prof., PhD., DrSc