IrisClassification

June 16, 2019

1 Iris classification

This notebook targets one of the most common deep learning problems, the classification. The chosen dataset is the well-known and simple Iris dataset.

[1]:
```
from __future__ import absolute_import, division, print_function

import pathlib

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import classification_report
```

1.0.1 Study of the dataset

Here we load and visualize a few examples of this dataset. It contains only 150 examples as a small dataset and each example has 4 features and one class corresponding to the species of the Iris flower.

[2]:
```
dataset_path = "D:\Projetos\DeepLearning_MiniProjects\IrisClassification\iris.
 ↪data"
column_names = ['Sepal length(cm)', 'Sepal width(cm)', 'Petal length(cm)',␣
 ↪'Petal width(cm)', 'Class']
raw_dataset = pd.read_csv(dataset_path, names=column_names)

dataset = raw_dataset.copy()
dataset.tail()
```

[2]:
```
     Sepal length(cm)  Sepal width(cm)  Petal length(cm)  Petal width(cm)  \
145               6.7              3.0               5.2              2.3
146               6.3              2.5               5.0              1.9
```

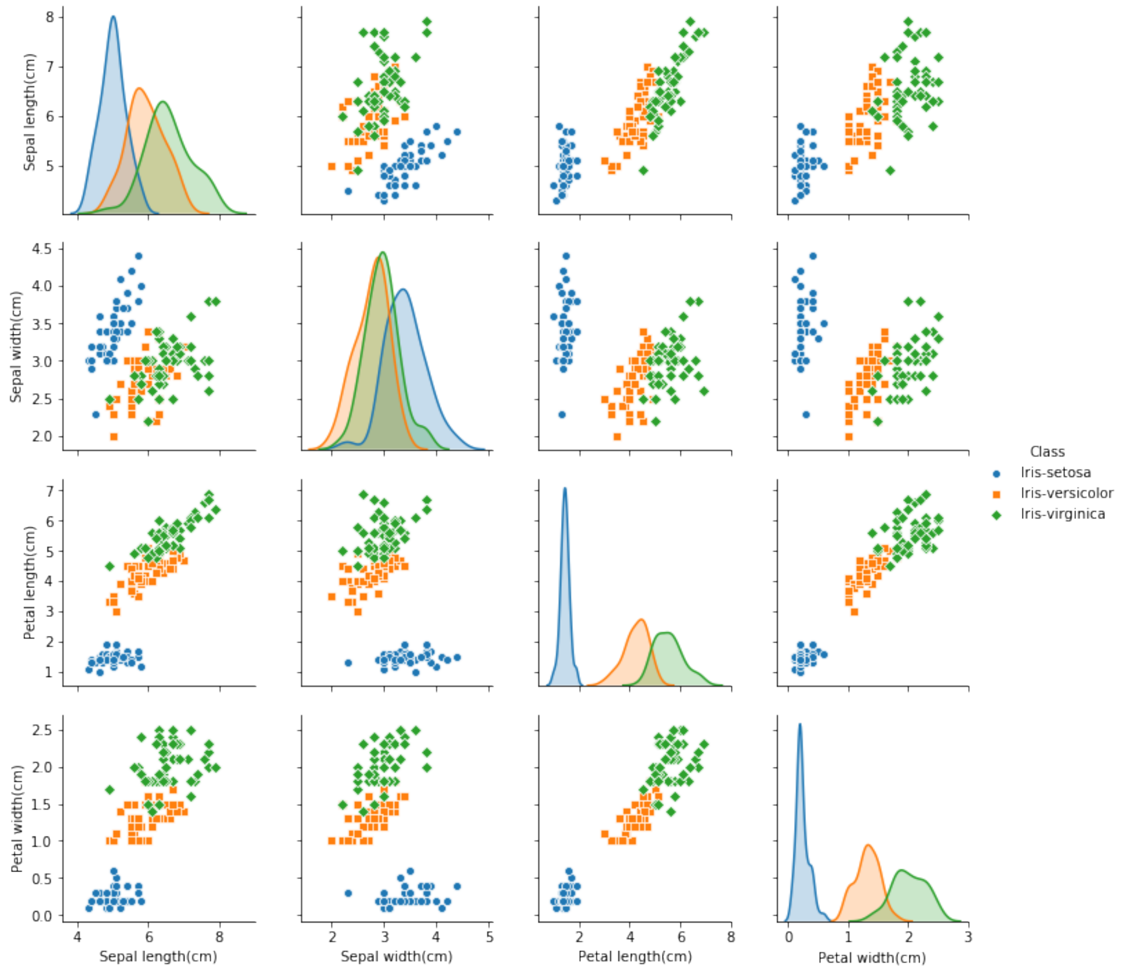
```
147              6.5              3.0              5.2              2.0
148              6.2              3.4              5.4              2.3
149              5.9              3.0              5.1              1.8

               Class
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica
```

The dataset has following distribution of examples for each pair of features. Just by looking at it we can say that Iris-setosa has very distinctive features compared to others. Concerning the other two, they seem to be roughly separable with exception from a region in the hyperspace where both can have similar features.

[3]:
```
sns.pairplot(dataset, hue="Class", vars=['Sepal length(cm)', 'Sepal width(cm)',␣
↪'Petal length(cm)', 'Petal width(cm)'],
             markers=["o", "s", "D"])
```

[3]:
```
<seaborn.axisgrid.PairGrid at 0x155f01d24e0>
```



1.0.2 Data preparation

The target labels are extracted and converted to integers to be used in training.

[4]:
```
def encodeLabelsToInt (dataset):
    class_index = {"Iris-setosa" : 0, "Iris-versicolor" : 1, "Iris-virginica" :␣
 ↪2}
    labels = dataset.pop('Class')
    labels = [class_index[c] for c in labels]
    dataset['Class'] = labels
```

[5]:
```
encodeLabelsToInt(dataset)
labels = dataset.pop('Class')
```

Considering the dataset is quite small, cross-validation is a good option to have a better measure of the model performance, so here the division of the data in folds is done, while also shuffling it to increase generalization of the model. The chosen number of folds is 2, so that we avoid as much as possible overfitting to the training dataset by making it small.

[6]:
```
#train_data, test_data, train_labels, test_labels = train_test_split(dataset,␣
 ↪labels, test_size=0.5, random_state=0)

k = 2
folds = list(StratifiedKFold(n_splits=k, shuffle=True, random_state=0).
 ↪split(dataset, labels))
```

1.0.3 Building the model

Taking the dataset as simple, an also simple model is desirable to avoid overfitting with just 3 dense layers with the number of units on the first and second layers adjusted empiracally. Another measure against the overfitting taken is the use of L2 regularizers which penalize high weights, with the penalty being proportional to their square. The output layer gives the probability of each class given an example, by using the "softmax" activation function.

The chosen loss function is the sparse categorical crossentropy which is suited for classification problems concerning the probability of an example being from a class. The use of the sparse one and not the simple one comes from the fact that the classes are enconded as integers and not with one hot. Lastly, the accuracy is a simple and direct metric for evaluating classification problems.

[7]:
```
def build_model():
    model = keras.Sequential([
        layers.Dense(32, kernel_regularizer=keras.regularizers.l2(0.001),␣
 ↪activation=tf.nn.relu, input_shape=(4,)),
        layers.Dense(64, kernel_regularizer=keras.regularizers.l2(0.001),␣
 ↪activation=tf.nn.relu),
        layers.Dense(3, activation=tf.nn.softmax)
    ])
    optimizer = keras.optimizers.Adam(0.001)

    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['acc'])
    return model
```

[8]:
```
model = build_model()
model.summary()
```

```
WARNING:tensorflow:From C:\Users\Pedro\AppData\Roaming\Python\Python37\site-
packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                160
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 195
=================================================================
Total params: 2,467
Trainable params: 2,467
Non-trainable params: 0
_________________________________________________________________
```

1.0.4 Training the model

Here the test was used as the validation data, because further reduction of the training dataset may impact in the performance of the model, even it is not a good practice in normal cases.

[9]:
```
data = dataset.values
label = labels.values

acc_list = []
history_list = []
models = []
for i, (train_i, test_i) in enumerate(folds):
    print('\nFold ', i)
    models.append(build_model())
    history = models[i].fit(data[train_i], label[train_i],
         epochs=50,
         validation_data= [data[test_i], label[test_i]],
         verbose=0)
    history_list.append(history)
    loss, acc = models[i].evaluate(data[test_i], label[test_i], verbose=1)
    acc_list.append(acc)
```

```
Fold  0
75/75 [==============================] - 0s 39us/sample - loss: 0.2876 - acc:
0.9733

Fold  1
75/75 [==============================] - 0s 54us/sample - loss: 0.2925 - acc:
0.9867
```

1.0.5 Analysis of results

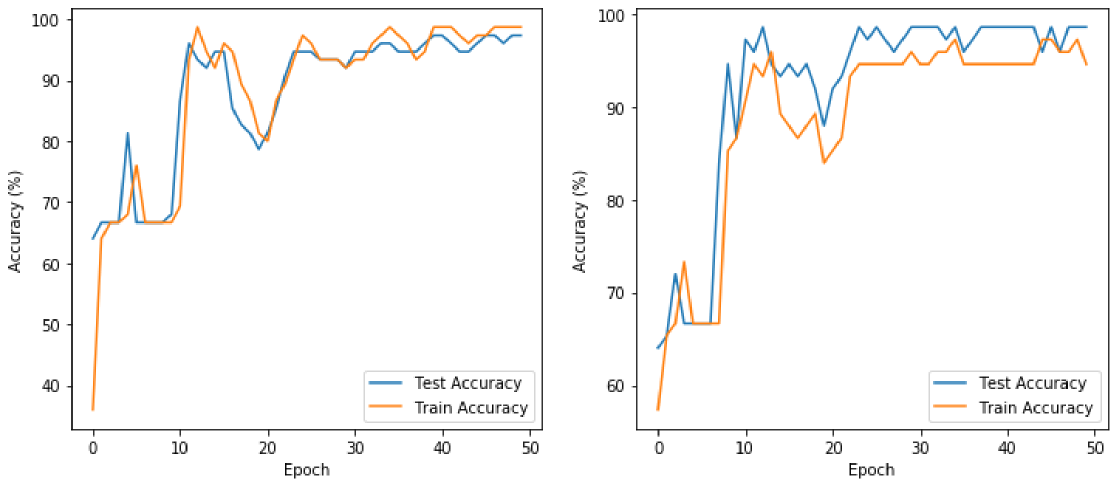
Since the dataset is small, we cannot hope for a smooth accuracy curve throughout training. However we clearly see that the accuracy increases until reaching an almost constant region after a few tens of epochs.

The achieved final accurary is fairly high and might be a result of the simplicity of the data and not from overfitting given all the measures taken against overfitting and the fact that the train and test accuracy are fairly close, even if completely avoiding it is impossible. It is true that using the test set as a validation set might induce some overfitting towards the test set, but considering the simlplicity of the problem and results, it is negligeable.

[10]:
```
def plot_history(history):
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.plot(hist['epoch'], hist['val_acc']*100,
           label='Test Accuracy')
    plt.plot(hist['epoch'], hist['acc']*100,
           label='Train Accuracy')
    plt.legend()

fig = plt.figure()
fig.set_figheight(5)
fig.set_figwidth(12)
for i, h in enumerate(history_list):
    plt.subplot(1, 2, i+1)
    plot_history(h)
```



[11]:
```
def decodeProbToInt(pred):
    return np.array([np.argmax(p) for p in pred])

def decodeIntToClass(labels):
    class_list = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
    return [class_list[l] for l in labels]
```

For the sake of simplicity we take the model trained only with the 1st fold division and analyze other metrics besides the accuracy. These are:

- Precision: ration between the true positives and all examples classified as a given class, it shows how precisily the model can identify one class without missclaassifying examples from other classes into it;
- Recall: ration between the true positives and all examples that are originally labeled as given class, it can identify problems when the model hardly classifies the examples of a given class correctly which may not be identified by the precision;
- F1 score: takes into account both metrics above, so the higher it is the better as it is ideal that both metrics above are as close as possible to one.

[12]:
```
(_, test_i) = folds[0]
pred = models[0].predict(data[test_i])
pred = decodeProbToInt(pred)
```

We can see that the model perfectly classifies the first species which was cleary distinct from the other as seen in the study of the dataset. It is also confirmed that the model when decides all examples which may be from the second or third species as the third species since only examples that can clearly be defined as the are classified as so (shown by precision equal to 1) and the rest of them are classified as the third.

[13]:
```
print(classification_report(label[test_i], pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        25
           1       1.00      0.92      0.96        25
           2       0.93      1.00      0.96        25

    accuracy                           0.97        75
   macro avg       0.98      0.97      0.97        75
weighted avg       0.98      0.97      0.97        75
```