Prediction with Random Forests

Please open the Read_me before this execution. See to with that all the installations are perfect. Please see to with that you've understood all the concepts in the Chapter-02 thoroughly, so that execution becomes easier.

At the end of this tutorial, you'll be able to build your own prediction model, by using Random Forests. We're going to look at examples of predicting bird species from descriptive attributes and then use a confusion matrix on them.
http://www.vision.caltech.edu/visipedia-data/CUB-200-2011/CUB_200_2011.tgz – To download the data set
After downloading, please extract the data-set to the target destination.

USP: From the above data, we are going to predict bird species using Random Forests.

Step-0:
Download the data-set from the above-mentioned URL. If its in zip format, unzip it otherwise, you'll see error while loading the data set. Here we are not going to look at the pictures because that would need a **convolutional neural network** (**CNN**) and this will be covered in later chapters. CNNs can handle pictures much better than a random forest. Instead, we will be using attributes of the birds such as size, shape and color.

Step-1: Loading the dataset into the algorithm

This is done by using **pandas.read_csv('name')** function offered by pandas library. The reason "sep=';'" is used as the usual reference of separation for csv files is , but in our dataset ; is used.

```
import pandas as pd

# some lines have too many fields (?), so skip bad lines

imgatt = pd.read_csv('E:/Zingpro_Internship/Proiect_2/CUB_200_2011/CUB_200_2011/attributes/image_attribute_labels.txt',

            sep='\s+', header=None, error_bad_lines=False, warn_bad_lines=False,

            usecols=[0,1,2], names=['imgid', 'attid', 'present'])
```

Note: The data is split into several files. We'll discuss those files before jumping into the code
classes.txt - file shows class IDs with the bird species names
images.txt - file shows image IDs and filenames
image_class_labels – file connects the class IDs with the image IDs
Attributes.txt - file gives the name of each attribute, which ultimately is not going to be that important to us.
image_attributes_labels.txt - connects each image with its attributes in a binary value that's either present or absent for
that attribute

Step-2:
Pandas **head()** method is used to return top n (5 by default) rows of a data frame or series.

```
imgatt.head()
```

Image ID number 1 does not have attributes 1, 2, 3, or 4, but it does have attribute 5

Step-3:

Pandas shape method is tell us how many rows and columns we have

```
imgatt.shape
```

It has 3.7 million rows and three columns. This is not the actual formula that you want. You want attributes to be the columns, not rows.

Step-4:

We need to reorganize imgatt to have one row per imgid, and 312 columns (one column per attribute). With 1/0 in each cell representing if that imgid has that attribute or not

```
imgatt2 = imgatt.pivot(index='imgid', columns='attid', values='present')
```

Therefore, we have to use pivot:
1. Pivot on the image ID and make one row for each image ID. There will be only one row for image number one.
2. Turn the attributes into distinct columns, and the values will be ones or twos.

Step-5:

We can now see that each image ID is just one row and each attribute is its own column, and we have the ones and the twos

```
imgatt2.head()
```

Note: Let's feed this data into a random forest. In the previous example, we have 312 columns and 312 attributes, which is ultimately about 12,000 images or 12,000 different examples of birds

Step-6:

```
imgatt2.shape
```

Now, we need to load the answers, such as whether it's a bird and which species it is. Since it is an image class labels file, the separators are spaces.

Step-7: Loading the image into true classes

There is no header row and the two columns are imgid and label. We will be using set_index('imgid') to have the same result produced by, imagatt.head(), where the rows are identified by the image ID

```
imglabels = pd.read_csv("data/CUB_200_2011/image_class_labels.txt",
                        sep=' ', header=None, names=['imgid', 'label'])
imglabels = imglabels.set_index('imgid')
```

Step-8:

```
imglabels.head()
```

Identification of the rows by the image ID using pandas head() method as mentioned above

Step-9:

The imgid column has 1,2,3,4,5, all are labeled as 1. They're all albatrosses at the top of the file. As seen, there are about 12,000 rows, which is perfect.

```
imglabels.shape
```

Step-10:

Now, we'll join the attributes. For that, we will be using join method. In the join, we will use the index on the image ID to join the two data frames. Effectively, what we're going to get is that the label is stuck on as the last column. We will be now shuffling and then be splitting off the attributes. In other words, we want to drop the label from the label. So, here are the attributes, with the first 312 columns and the last column being a label.

```
# now we need to attach the labels to the attribute data set,

# and shuffle; then we'll separate a test set from a training set

df = imgatt2.join(imglabels)

df = df.sample(frac=1)
```

Step-11:

Pandas provide a unique method to retrieve rows from a Data frame. `Dataframe.iloc[]` method is used when the index label of a data frame is something other than numeric series of 0, 1, 2, 3….n or in case the user doesn't know the index label. Rows can be extracted using an imaginary index position which isn't visible in the data frame.

```
df_att = df.iloc[:, :312]

df_label = df.iloc[:, 312:]
```

Step-12:

```
df_att.head()
```

Step-13:

```
df_label.head()
```

Note: In the above 2 steps, we are returning the no. of rows in the data frame.

Step-14: Training the classifier

After shuffling, we have the first row as image 527, the second row as image 1532, and so forth. The attributes in the label data are in agreement. On the first row, it's image 527, which is the number 10. You will not know which bird it is, but it's of the kind, and these are its attributes. But it is finally in the right form. We need to do a training test split. There were 12,000 rows, so let's take the first 8,000 and call them training, and the call rest of them testing (4,000).

```
df_train_att = df_att[:8000]
df_train_label = df_label[:8000]
df_test_att = df_att[8000:]
df_test_label = df_label[8000:]
df_train_label = df_train_label['label']
df_test_label = df_test_label['label']
```

Step-15:

We'll get the answers using RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_features=50, random_state=0, n_estimators=100)
```

Step-16: Fitting the classifier

```
clf.fit(df_train_att, df_train_label)
```

Step-17:

Let's predict a few cases. Let's use attributes from the first five rows of the training set,which will predict species 10, 28, 156, 10, and 43.

```
print(clf.predict(df_train_att.head()))
```

```
clf.score(df_test_att, df_test_label)
```

Step-19: Making a confusion matrix

Let's make a confusion matrix to see which birds the dataset confuses. The confusion_matrix function from scikit-learn will produce the matrix, but it's a pretty big matrix 28, 156, 10, and 43.

```
from sklearn.metrics import confusion_matrix
pred_labels = clf.predict(df_test_att)
cm = confusion_matrix(df_test_label, pred_labels)
```

Step-20: Plotting the data

By now, you must have understood that its not easy to understand the numeric form of data. So let's plot it using matplotlib

```
import matplotlib.pyplot as plt
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    #plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    #for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    #    plt.text(j, i, format(cm[i, j], fmt),
    #             horizontalalignment="center",
    #             color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Step-21:

We will need the actual names of the birds on the matrix so that we know the species that are being confused for each other. So, let's load the classes file:

```
birds = pd.read_csv("data/CUB_200_2011/classes.txt",
                    sep='\s+', header=None, usecols=[1], names=['birdname'])
birds = birds['birdname']
birds
```

Step-22:

Plot the matrix. This is the confusion matrix for this dataset

```
import numpy as np
np.set_printoptions(precision=2)
plt.figure(figsize=(60,60), dpi=300)
plot_confusion_matrix(cm, classes=birds, normalize=True)
plt.show()
```

Step-23:

The output is unreadable because there are 200 rows and columns. But if we open it separately and then start zooming in, on the *y* axis you will see the actual birds, and on the *x* axis, you will see the predicted birds

Step-24: Random Forests vs Decision Trees Comparision

Since the bird's names are sorted, lesser is the square of confusion. Let's compare this with the simple decision tree

```
from sklearn import tree
clftree = tree.DecisionTreeClassifier()
clftree.fit(df_train_att, df_train_label)
clftree.score(df_test_att, df_test_label)
```

Step-25: Support Vector Machine (SVM) vs Decision Trees Comparison

Here, the accuracy is 27%, which is less than the previous 44% accuracy. Therefore, the decision tree is worse. If we use a **Support Vector Machine** (**SVM**), which is the neural network approach, the output is 29%:

```
from sklearn import svm
clfsvm = svm.SVC()
clfsvm.fit(df_train_att, df_train_label)
clfsvm.score(df_test_att, df_test_label)
```

Note: The random forest is still better

Step-26: Cross Validation

Let's perform cross-validation to make sure that we split the training test in different ways.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, df_train_att, df_train_label, cv=5)
# show average score and +/- two standard deviations away (covering 95% of scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
scorestree = cross_val_score(clftree, df_train_att, df_train_label, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scorestree.mean(), scorestree.std() * 2))
```

```
scoressvm = cross_val_score(clfsvm, df_train_att, df_train_label, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scoressvm.mean(), scoressvm.std() * 2))
```

Note: We have compared the accuracy of the 3 classification trainings. The best results are reflected through random forests since we had some options and questions with random forests.

Step-27: Printing features and corresponding accuracy

```
max_features_opts = range(5, 50, 5)
n_estimators_opts = range(10, 200, 20)
rf_params = np.empty((len(max_features_opts)*len(n_estimators_opts),4), float)

i = 0
for max_features in max_features_opts:

  for n_estimators in n_estimators_opts:

    clf = RandomForestClassifier(max_features=max_features, n_estimators=n_estimators)

    scores = cross_val_score(clf, df_train_att, df_train_label, cv=5)

    rf_params[i,0] = max_features

    rf_params[i,1] = n_estimators

    rf_params[i,2] = scores.mean()

    rf_params[i,3] = scores.std() * 2

    i += 1

    print("Max features: %d, num estimators: %d, accuracy: %0.2f (+/- %0.2f)" %          (max_features, n_estimators,
scores.mean(), scores.std() * 2))
```

Step-28: Visualization of the features and corresponding accuracy

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
fig.clf()
ax = fig.gca(projection='3d')
x = rf_params[:,0]
y = rf_params[:,1]
z = rf_params[:,2]
ax.scatter(x, y, z)
ax.set_zlim(0.2, 0.5)
ax.set_xlabel('Max features')
ax.set_ylabel('Num estimators')
ax.set_zlabel('Avg accuracy')
plt.show()
```

Note: This is not a model where you can see the bird images. For that we need to use Convolution Neural Networks which shall be covered in the further projects.