Lab1

September 2, 2021

1 Introduction to Python and Scikit-Learn

Note: This lab has been generated using material from Chapter 3 in ”Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow. The following link (https://github.com/ageron/handson-ml) contains the extended Jupyter notebook as well as more tasks so you can better familiarize yourself with Python and Scikit-learn.

2 1. MNIST

MNIST dataset, is a set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau. Each image is labeled with the digit it represents. For the first task download and display some of the digits in the dataset.

[94]:
```
import pandas as pd
import os
import numpy as np
# to make this notebook's output stable across runs
np.random.seed(42)
import sklearn # ML in python

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "classification"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
```

```
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

[ ]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.keys()
```

[ ]:
```
mnist.values()
```

[ ]:
```
# set up the feature data and labels
X, y = mnist["data"], mnist["target"]
X.shape
print(X[0]) # X is 2D array
```

[ ]:
```
y.shape
#print(len(y))
print(y[0]) # y is 1D array
```

[ ]:
```
some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap=mpl.cm.binary)
plt.axis("off")
plt.show()
```

3 2. Binary Classifier

2.1 Identify one digit for example, the number 5. This "5-detector" will be an example of a binary classifier, capable of distinguishing between just two classes, 5 and not-5. For this task pick the Stochastic gradient descent classifier from the Scikit-Learn's SGDClassifier class.

2.2 Evaluate the performance of your classifier by
(a) Measuring accuracy using cross-validation.
(b) The use of the confusion matrix.
(c) Understanding the precision/recall trade-off.
(d) The use of the ROC curve.

2.3 Compare the ROC curve generated by the RandomForestClassifier with the ROC curve generated by the SGDClassifier

[ ]:
```
# change y to integer
y = y.astype(np.uint8)
type(y)
```

[ ]:
```
# create a training and testing set
# The MNIST dataset is actually already split into a training set (the first␣
↪60,000 images) and a test set (the last 10,000 images), so:
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
shuffle_index = np.random.permutation(60000)
#print(shuffle_index)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

```
print(X_train)
```

[ ]:
```
# Distinguish between two classes, 5 and not-5. Create the target vectors
y_train_5 = (y_train == 5) # make it as binary
y_test_5 = (y_test == 5)
print(y_train_5)
```

[ ]:
```
#X_train.shape
y_train_5
```

[ ]:
```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(max_iter = 5, tol = -np.infty, random_state = 42) #␣
↪reproducible results so set a random_state
sgd_clf.fit(X_train, y_train_5)
```

[ ]:
```
sgd_clf.predict([some_digit])
```

[ ]:
```
plt.imshow(some_digit_image, cmap="gray")
```

3.1 Measuring Accuracy Using Cross-Vaildation

[ ]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

Let's use the `cross_val_score()` function to evaluate our SGDClassifier model, using K-fold cross-validation with three folds. Remember that K-fold cross-validation means splitting the training set into K folds (in this case, three), then making predictions and evaluating them on each fold using a model trained on the remaining folds

[ ]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

3.2 Confusion Matrix

Each row in a confusion matrix represents an actual class, while each column represents a predicted class.

[ ]:
```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)#␣
↪regression, data, target
print(len(y_train_pred))
```

[ ]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_5, y_train_pred)
```

3.3 Precision/Recall trade-off

3.3.1 Precision = TP/ (TP + FP)

- TP is the number of true positives, and FP is the number of false positives.

3.3.2 Recall (Sensitivity) = TP / (TP + FN)

- FN is, of course, the number of false negatives.

[ ]:
```
# Precision
sklearn.metrics.precision_score(y_train_5, y_train_pred) # equal to 4096 /␣
↪(4096 + 1522)
```

[ ]:
```
# Recall
sklearn.metrics.recall_score(y_train_5, y_train_pred) # equal to 4096 / (4096 +␣
↪1325)
```

3.4 F1 Score

Combining **precision** and **recall** into a single metric called the F1 score

[ ]:
```
sklearn.metrics.f1_score(y_train_5, y_train_pred)
```

4 The ROC Curve

ROC curve plots the true positive rate (another name for recall aka TPR) against the false positive rate (FPR).

[ ]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_train_pred)
```

[ ]:
```
def plot_roc_curve(fpr, tpr, label=None):
  plt.plot(fpr, tpr, linewidth=2, label=label)
  plt.plot([0, 1], [0, 1], 'k--')
  plt.axis([0, 1, 0, 1])
  plt.xlabel('False Positive Rate', fontsize=16)
  plt.ylabel('True Positive Rate', fontsize=16)
plt.figure(figsize=(8, 6))
plot_roc_curve(fpr, tpr)
plt.show()
```

5 Random Forest Classifier

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(n_estimators=10, random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv = 3,␣
↪method="predict_proba")
```

```
[ ]: y_probas_forest[:,1] # take all the row and only the second column
```

```
[ ]: y_probas_forest # The predict_proba() method returns an array containing a row␣
     ↪per instance and a column per class
```

```
[ ]: y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class
      fpr_forest, tpr_forest, thresholds_forest =␣
      ↪roc_curve(y_train_5,y_scores_forest)
```

```
[ ]: # Compare the ROC curve generated by the RandomForestClassifier with the ROC␣
      ↪curve geenrated by the SGDClassifier
     plt.figure(figsize=(8, 6))
     plt.plot(fpr, tpr, "b:", linewidth=2, label="SGD")
     plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
     plt.legend(loc="lower right", fontsize=16)
     plt.show()
```

6 Conclusion

Random Forest model performs better.

```
[ ]: !jupyter nbconvert --to pdf "/content/drive/MyDrive/American_University/
      ↪2021_Fall/DATA-642-001_Advanced Machine Learning/GitHub/Labs/Lab1.ipynb"
```