Pr8 - MNIST_sol

December 4, 2020

Dieses Notebook ist angelehnt an das Buch *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow* von Aurélien Géron, Notebooks verfügbar auf GitHubPages.

[ ]:
```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

1 Praktikum: Session 8

Video

In diesem Praktikum wollen wir viele der bereits gelernten Konzepte verwenden, um die sog. MNIST-Daten zu klassifizieren. Bei den MNIST-Daten handelt es sich um 70000 handschriftliche Ziffern, die gelabelt sind. Dieses Datenset war lange *der* Benchmark im Bereich Bilderkennung. Der große Erfolg von neuronalen Netzen in den letzten Jahren hat es notwendig gemacht, einen schwierigeren Datensatz zum Benchmarking zu verwenden ("Fashion-MNIST").

1.1 Daten

1.1.1 Kennenlernen der Daten

Laden Sie die MNIST Daten und verschaffen Sie sich einen Eindruck davon.

[ ]:
```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version=1)
mnist.target = mnist.target.astype(np.uint8)
```

[ ]:
```
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
```

```
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = mpl.cm.binary, **options)
    plt.axis("off")
```

[ ]:
```
plt.figure(figsize=(9,9))
example_images = mnist.data[:100]
plot_digits(example_images, images_per_row=10)
plt.show()
```

1.1.2 Aufteilung Trainings, Validierung, Test

Teilen Sie die Daten auf in ein Trainingsset, ein Validierungsset und ein Testset (z.B. 50000, 10000, 10000).

[ ]:
```
from sklearn.model_selection import train_test_split
```

[ ]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(
    mnist.data, mnist.target, test_size=10000, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=10000, random_state=42)
```

1.2 Modelle: Einzelne Classifier

Trainieren Sie einen Random Forest Classifer und einen SVM Classifier. Wie gut performen diese (verwenden Sie die `score` Methode der einzelnen Estimator). Trainieren Sie auch noch einen `ExtraTreeClassifier` und einen `MLPClassifier`.

Hinweis: Einen möglichen Code finden Sie bereits unten. Dieser enthält allerdings nur den `MLPClassifier`.

[ ]:
```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.svm import LinearSVC
from sklearn.neural_network import MLPClassifier
```

[ ]:
```
random_forest_clf = RandomForestClassifier(n_estimators=100, random_state=42,␣
↪n_jobs=-1)
extra_trees_clf = ExtraTreesClassifier(n_estimators=100, random_state=42,␣
↪n_jobs=-1)
svm_clf = LinearSVC(random_state=42)
mlp_clf = MLPClassifier(random_state=42)
```

[ ]:
```
estimators = [random_forest_clf, extra_trees_clf, svm_clf, mlp_clf]
for estimator in estimators:
    print("Training the", estimator)
    estimator.fit(X_train, y_train)
```

```
Training the RandomForestClassifier(n_jobs=-1, random_state=42)
Training the ExtraTreesClassifier(n_jobs=-1, random_state=42)
Training the LinearSVC(random_state=42)

C:\Users\Cris\Anaconda3\lib\site-packages\sklearn\svm\_base.py:977:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)

Training the MLPClassifier(random_state=42)
```

[ ]:
```
[estimator.score(X_val, y_val) for estimator in estimators]
```

[ ]: [0.9692, 0.9715, 0.8695, 0.9601]

1.3 Modelle: Kombination zu einem Ensemble

Versuchen Sie, diese Modelle so zu einem Ensemble zu kombinieren, dass das Ensemble besser ist als jeder einzelne Classifier. Verwenden Sie dafür `VotingClassifier`, siehe Sklearn-Dokumentation, sowohl in der "Hard-Voting" als auch in der "Soft-Voting" Version. Falls das Ensemble *aller* einzelnen Estimators nicht besser wird, versuchen Sie, nur manche der einzelnen Estimators zu verwenden (vielleicht die besten…).

Hinweise: - Eine Liste von benannten Estimators wie diese könnte nützlich sein: named_estimators = [("random_forest_clf", random_forest_clf), ("extra_trees_clf", extra_trees_clf), ("svm_clf", svm_clf), ("mlp_clf", mlp_clf),] - Zur Entfernung einzelner Estimators ist unten ein Beispiel-Code angegeben.

[ ]:
```
from sklearn.ensemble import VotingClassifier
```

[ ]:
```
named_estimators = [
    ("random_forest_clf", random_forest_clf),
    ("extra_trees_clf", extra_trees_clf),
    ("svm_clf", svm_clf),
    ("mlp_clf", mlp_clf),
]
```

[ ]:
```
voting_clf = VotingClassifier(named_estimators, n_jobs=-1)
```

[ ]:
```
voting_clf.fit(X_train, y_train)
```

[ ]:
```
VotingClassifier(estimators=[('random_forest_clf',
                              RandomForestClassifier(n_jobs=-1,
                                                     random_state=42)),
                             ('extra_trees_clf',
                              ExtraTreesClassifier(n_jobs=-1, random_state=42)),
                             ('svm_clf', LinearSVC(random_state=42)),
                             ('mlp_clf', MLPClassifier(random_state=42))],
                 n_jobs=-1)
```

[ ]:
```
voting_clf.score(X_val, y_val)
```

[ ]:
```
0.9714
```

[ ]:
```
[estimator.score(X_val, y_val) for estimator in voting_clf.estimators_]
```

[ ]:
```
[0.9692, 0.9715, 0.8695, 0.9642]
```

Entfernung eines Estimators, z.B. von `svm_clf`:

[ ]:
```
voting_clf.set_params(svm_clf='drop');
```

```
[ ]: voting_clf.estimators
```

```
[ ]: [('random_forest_clf', RandomForestClassifier(n_jobs=-1, random_state=42)),
      ('extra_trees_clf', ExtraTreesClassifier(n_jobs=-1, random_state=42)),
      ('svm_clf', 'drop'),
      ('mlp_clf', MLPClassifier(random_state=42))]
```

`svm_clf` ist nun zwar aus der Liste der Estimators entfernt. Damit würde er bei einem erneuten Training nicht mehr mit trainiert werden. Allerdings ist er aktuell schon trainiert. Um uns ein erneutes Training zu ersparen, können wir ihn manuell aus der Liste der trainierten Estimators entfernen:

```
[ ]: voting_clf.estimators_
```

```
[ ]: [RandomForestClassifier(n_jobs=-1, random_state=42),
      ExtraTreesClassifier(n_jobs=-1, random_state=42),
      LinearSVC(random_state=42),
      MLPClassifier(random_state=42)]
```

```
[ ]: del voting_clf.estimators_[2]
```

```
[ ]: voting_clf.estimators_
```

```
[ ]: [RandomForestClassifier(n_jobs=-1, random_state=42),
      ExtraTreesClassifier(n_jobs=-1, random_state=42),
      MLPClassifier(random_state=42)]
```

Wir werten nun den Voting Classifier noch einmal aus (ohne den SVM Classifier):

```
[ ]: voting_clf.score(X_val, y_val)
```

```
[ ]: 0.9737
```

Nun versuchen wir statt des "Hard-Votings" noch ein "Soft-Voting":

```
[ ]: voting_clf.voting = "soft"
     voting_clf.score(X_val, y_val)
```

```
[ ]: 0.9692
```

"Hard-Voting" ist hier also besser.

1.4 Anwendung auf das Testset

Wenden Sie den besten gefundenen (Voting-) Classifier auf das Testset an und ermitteln Sie die Score. Geben Sie auch die Scores der einzelnen Classifier auf dem Testset an.

```
[ ]: voting_clf.voting = "hard"
     voting_clf.score(X_test, y_test)
```

[ ]: 0.971

[ ]:
```
[estimator.score(X_test, y_test) for estimator in voting_clf.estimators_]
```

[ ]: [0.9645, 0.9691, 0.9601]

2 Performance-Vergleich Random Forest ohne und mit Dimensionsreduktion

In diesem Abschnitt brauchen wir keine Validierungsdaten mehr. Nehmen Sie daher eine neue Aufteilung der gesamten Daten in Trainings- und Testset vor (z.B. 60000 und 10000).

Trainieren Sie einen Random Forest Classifier und bewerten Sie das resultierende Modell. Geben Sie auch die Rechenzeit des Modells an.

Hinweis: Einen möglichen Code für das Timing finden Sie unten.

[ ]:
```
X_train = mnist['data'][:60000]
y_train = mnist['target'][:60000]

X_test = mnist['data'][60000:]
y_test = mnist['target'][60000:]
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
```

[ ]:
```
import time

t0 = time.time()
rnd_clf.fit(X_train, y_train)
t1 = time.time()
print("Training took {:.2f}s".format(t1 - t0))
```

Training took 5.23s

[ ]:
```
from sklearn.metrics import accuracy_score

y_pred = rnd_clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

[ ]: 0.9705

Verwenden Sie nun zuerst PCA, um die Dimension der Daten zu verringern. Es sollen dabei 95% der Varianz beibehalten werden.

[ ]:
```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=0.95)
X_train_reduced = pca.fit_transform(X_train)
```

Trainieren Sie einen neuen Random Forest Classifier auf dem reduzierten Datenset. Bewerten Sie auch dessen Performance und Rechenzeit und vergleichen Sie es mit der obigen Analyse der Originaldaten.

[ ]:
```
rnd_clf2 = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
t0 = time.time()
rnd_clf2.fit(X_train_reduced, y_train)
t1 = time.time()
print("Training took {:.2f}s".format(t1 - t0))
```

```
Training took 7.32s
```

[ ]:
```
X_test_reduced = pca.transform(X_test)

y_pred = rnd_clf2.predict(X_test_reduced)
accuracy_score(y_test, y_pred)
```

[ ]:
```
0.9481
```

[ ]:
```
```