UE 3 – Durchführung

Merkmalsextraktion, binäre Klassifikation

Überblick

Das Ziel dieser Übung ist die Integration der in der Vorlesung kennengelernten Merkmalsextraktionsverfahren in eine Klassifikationspipeline. Dabei werden wir uns zunächst einmal der Unterscheidung von jeweils **zwei** Klassen widmen, also binären Klassifikationsproblemen. Wir werden den GTSRB Datensatz verwenden (***R**ecognition* statt ***D**etection*) um die einzelnen Verkehrsschildklassen zu unterscheiden. PCA und Lernkurve werden benutzt um das Problem zu veranschaulichen und genauer zu untersuchen.

Verglichen mit den Übungen 1 und 2 ist diese Übung anders gestaltet. Es gibt diesmal keine Jupyter-Notebook-Vorlage. Ihre Aufgabe besteht darin, selbstständig ein Jupyter Notebook anzulegen, die nötigen Pakete zu importieren und die Aufgaben basierend auf dem Wissen aus den vergangenen Übungen und den Hinweisen in diesem Übungsblatt zu lösen. Als Grundlage dient die abgebildete Pipeline.

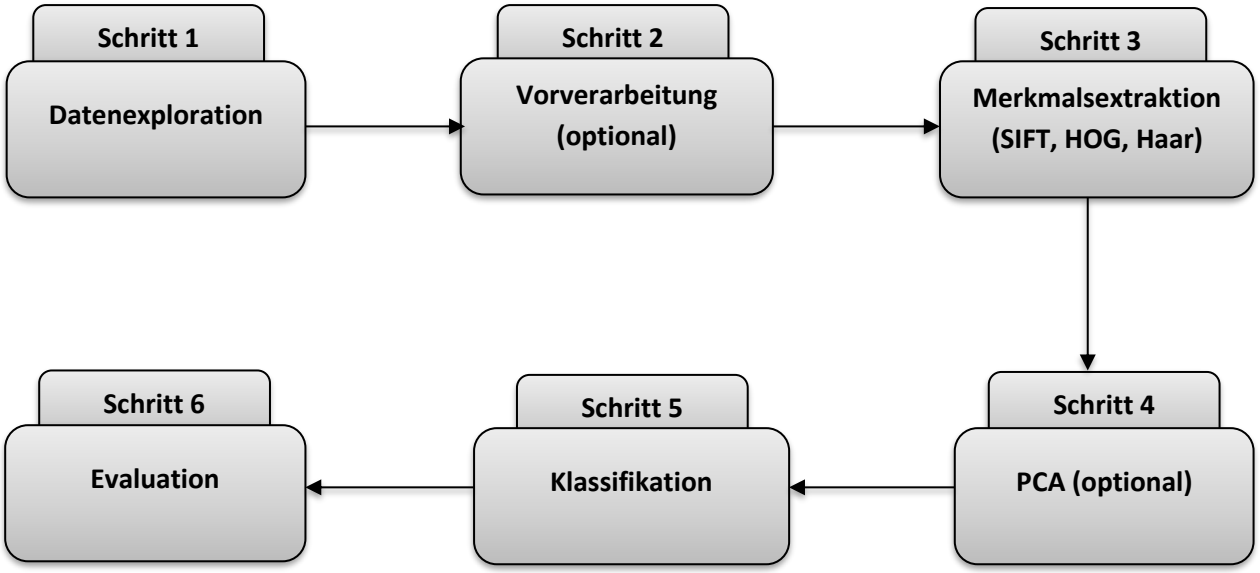

Abbildung 1: Schematische Darstellung der Klassifikationspipeline

Aufgabe 1 – Datenexploration

Die Daten (*official training data* und *official test dataset*) findet ihr unter folgendem Link. Hier sind sowohl die Datensätze mit den Bildern abgelegt als auch die Datensätze mit vorverarbeiteten Features (HOG, Haar-like, Hue Histograms). Ihr könnt sowohl die bereits berechneten Features-Datensätze nutzen als auch eure eigenen Features-Datensätze erstellen.

Eine detaillierte Beschreibung der Datensätze ist unter folgendem Link zu finden. Die README-Dateien in den jeweiligen Ordnern enthalten ebenfalls nützliche Informationen. Für unser binäres Klassifikationsproblem werden nur die Datensätze aus dem *official training data*-Block benötigt.

Daten in Trainings- und Validierungsdaten unterteilen:

1. Festlegen, welche Klassen eingelesen werden (sich zwei aussuchen). Dies könnt ihr im Laufe der Übung variieren, um schwierigere und leichtere Klassifikationsprobleme zu untersuchen, je nachdem ob die Schilder sehr ähnlich sind oder sich sehr unterscheiden.

2. Daten der entsprechenden Klassen einlesen. Nutzt dafür euer Vorwissen aus vergangenen Übungen. Da ihr nur die Datensätze aus dem *official training data*-Block verwendet, sollten die eingelesenen Daten zusätzlich in Trainings- und Validierungsdaten unterteilt werden. Ein mögliches Verhältnis wäre z.B. 75% Trainingsdaten und 25% Validierungsdaten. Dafür könnt ihr beispielsweise die train_test_split-Funktion aus der scikit-learn-Bibliothek nutzen. Bevor die Funktion genutzt werden kann, solltet ihr das scikit-learn-Paket in eurer Entwicklungsumgebung installieren (detaillierte Beschreibung zum Installieren von Paketen könnt ihr in der Anleitung zur Einrichtung der Entwicklungsumgebung nachschlagen).

Aufgabe 2 – Merkmalsextraktion, PCA, Binäre Klassifikation

Hinweise zur Merkmalsextraktion:

Zum Einstieg könnt ihr die bereits berechneten Features (beispielsweise HOG) verwenden. Anschließend könnt ihr andere Merkmalsextraktionsverfahren (siehe Abbildung 2) ausprobieren.

```
cv2.HOGDescriptor()
cv2.xfeatures2d.SIFT_create()
cv2.xfeatures2d.SURF_create()
cv2.xfeatures2d.BriefDescriptorExtractor_create()
cv2.PCACompute()

# mit dem Befehl help() koennt ihr eine detaillierte Beschreibung
# des jeweiligen Deskrtiptors / Detektors ansehen:
help(cv2.HOGDescriptor())
```

Abbildung 2: Hilfreiche cv2-Module

Hinweise zur PCA:

Führt eine PCA auf den Trainingsdaten aus und projiziert die Daten dann in den neuen Merkmalsraum. Das gibt euch Aufschluss darüber, wie schwierig euer Klassifikationsproblem ist beziehungsweise wie gut eure Features geeignet sind, um die Klassen zu unterscheiden.

Visualisiert euch die Features von unterschiedlichen Klassen von Verkehrsschildern. (Ihr könnt hier auch mehr als nur zwei Klassen auswählen, um euch mehr Klassen auf einmal zu visualisieren.) Ihr könnt sowohl die OpenCV- als auch die scikit-learn-Bibliothek dafür nutzen.

Hinweise zur Klassifikation:

Für die Klassifikation könnt ihr den in der Vorbereitungsübung kennengelernten SVM-Klassifikator einsetzen. Informationen zur Verwendung des SVM-Klassifikators könnt ihr auch im OpenCV 3 computer vision with Python cookbook finden.

Hinweise zur Evaluation:

Zur Beurteilung der Klassifikationsleistung des Klassifikators könnt ihr die CCR auf den Validierungsdaten berechnen und euch die Konfusionsmatrix anschauen. Dafür könnt ihr beispielsweise die accuracy_score-Funktion, die confusion_matrix-Funktion, f1_score-Funktion der scikit-learn-Bibliothek nutzen. Experimentiert ein wenig und vergleicht die Ergebnisse für verschiedene Klassen und Konfigurationen eures Klassifikators.

Erstellung der Lernkurve: Das Ziel ist, zu schauen, wie sich die Fehler auf den Trainings- und Validierungsdaten mit der Menge der genutzten Trainingsdaten verändern. Dies wird Aufschluss darüber geben, wie ihr eure Klassifikationsleistung am effektivsten verbessern könnt. Man kann daraus ablesen, ob z.B. die Features ungeeignet sind oder das Modell des Klassifikators zu simpel ist, oder ob z.B. mehr Trainingsdaten helfen würden um die Klassifikationsleistung zu verbessern. Zur Erstellung der Lernkurve könnt ihr die learning_curve-Funktion der scikit-learn-Bibliothek nutzen.