

# Projektbericht

## Gebärdensprache Anwendung

**Verfasser:**

Abdalla Mukhaimar (1615092)  
Mahmoud Helmy (1615089)  
Hamza Kaisoun (1615160)

**Betreuer:**

Herr Prof. Dr. Marcard

**Studiengang:**

Ingenieurinformatik Maschinenbau

**Semester:**

WiSe 2023/24

**Abgabedatum:**

21.09.2023

# Inhaltsverzeichnis

Abbildungsverzeichnis .....	3
Einleitung.....	4
Methodik.....	5
Datensammlung und -Vorbereitung.....	6
Datensammlung bzw. Bildersammlung.....	6
Datenvorbereitung bzw. -verarbeitung .....	7
Handausrichtung Festlegung.....	8
Winkel Berechnung.....	11
Drehwinkel Bestimmung .....	12
Modellauswahl und -bewertung.....	15
Modellauswahl.....	15
1. Problem Darstellung und geeigneter ML-Art .....	15
2. Größe des Datensatzes .....	15
3. Anforderung.....	16
4. Geeignete Lernalgorithmen .....	16
4- Experimente .....	17
Hyperparameter-Tuning & Leistung Bewertung:.....	17
Random Forest:.....	17
Support Vector Machine (SVM): .....	21
SGDClassifier (Stochastic Gradient Descent) .....	31
Vergleich der Modelle.....	36
Die Erkennung für beide Hände erweitern.....	36
GUI-Entwicklung.....	38
Fazit.....	40
Englischer Fachbegriffe.....	41
Quellen.....	42

# Abbildungsverzeichnis

Abbildung 1: Benutzeroberfläche für Bilderaufnahmen .....	7
Abbildung 2: Handlandmarks Koordinatennormalisierung .....	8
Abbildung 3: Ein Bild vor und nach der Rotation .....	8
Abbildung 4: Handlandmarks [7] .....	9
Abbildung 5: Handausrichtung nach Oben .....	10
Abbildung 6: Handausrichtung nach rechts .....	10
Abbildung 7: Handausrichtung Erkennungscode „hand_orientation“ .....	11
Abbildung 8: Funktionscode "angle" .....	11
Abbildung 9: Funktionscode "angle_of_rotation" .....	13
Abbildung 10: Drehwinkelbrechungsprinzip mit einer Handausrichtung "Up" und "Down" .....	14
Abbildung 11: Drehwinkelbrechungsprinzip mit einer Handausrichtung "Right" und "Left" .....	14
Abbildung 12: scikit-learn algorithm cheat-sheet .....	17
Abbildung 13: Mean_Test_score (RandomForest) .....	18
Abbildung 14: Mean_Train_score (RandomForest) .....	18
Abbildung 15: Score Difference(Train Score - Test Score) für Random Forest .....	19
Abbildung 16: Konfusionsmatrix (max_depth:30 und n_estimators :100) Random Forest .....	20
Abbildung 17: Konfusionsmatrix (max_depth: 9 und n_estimators :100) Random Forest .....	20
Abbildung 18: Mean Train und Test Score vs. C (LinearSVC) .....	22
Abbildung 19: Train - Test Score Difference vs. C (LinearSVC) .....	22
Abbildung 20: SVC mit C='50' (Classifier Report) .....	23
Abbildung 21: Konfusionsmatrix für C='50'(LinearSVC) .....	24
Abbildung 22: Kleinste Differenz (Trainscore - Testscore) .....	24
Abbildung 23: Bestes LinearSVC Modell mit (C = 100) .....	25
Abbildung 24: Konfusionsmatrix für C='100'(LinearSVC) .....	26
Abbildung 25: Test Score vs C für verschiedene Gamma-werte .....	27
Abbildung 26: Train Score Vs. C für verschiedene Gamma-werte .....	27
Abbildung 27: Mean Train und Test Score vs. C (SVC mit rbf Kernel) .....	28
Abbildung 28: Train - Test Score Difference vs. C für SVM(SVC) .....	29
Abbildung 29: Auflistung der geringsten Unterschiede zwischen Training- und Test-Score .....	29
Abbildung 30: Bestes SVC Modell mit (C = 2.0 und gamma=scale und kernel='rbf').....	30
Abbildung 31: Kofusionsmatrix für die beste SVC Model(SVC mit C='2') .....	30
Abbildung 32: Grid Search Ergebnisse (SGD Classifier): mean_test_score .....	31
Abbildung 33: : Mean Train und Test Score vs. alpha für SGDClassifier .....	32
Abbildung 34: Score Difference(Train- Test) vs. Alpha für SGDClassifier .....	33
Abbildung 35: Auflistung der geringsten Unterschiede zwischen Training- und Test-Score(SGDClassifier) .....	33
Abbildung 36: Bestes SGD_Classifier Modell .....	34
Abbildung 37: Konfusionsmatrix für die SGD_Classifier(alpha=0.0004) .....	34

Abbildung 38: SGD_Classifier mit Alpha =0.0004 (Classifier Report).....	35
Abbildung 39: Aufruf von Modellen im GUI-Code.....	37
Abbildung 40: Linke Hand konfusionsmatrix(SVC mit C= 2 und Gamme='scale') .....	37
Abbildung 41: GUI_Darstellung.....	38
Abbildung 42: predicted character Zähler.....	39

## Einleitung

Wir sind Studierende, die sich leidenschaftlich für die Lösung realer Probleme engagieren. In unserem Studienprojekt möchten wir einen positiven Beitrag leisten, um unsere Gesellschaft inklusiver und verständnisvoller zu machen. Wir glauben daran, dass Bildung und Technologie Menschen miteinander verbinden können, unabhängig von den Herausforderungen, die sie im Leben bewältigen müssen.

Unsere Motivation für dieses Projekt ist einfach: Wir möchten, dass alle Menschen, unabhängig von Hörbeeinträchtigungen oder Gehörlosigkeit, sich frei ausdrücken und verstanden werden können. Kommunikation ist wichtig, und deshalb haben wir uns ein Ziel gesetzt: Wir möchten eine innovative Gebärdensprachanwendung entwickeln. Diese Anwendung soll nicht nur die Gebärdensprache zugänglicher machen, sondern auch das Verständnis und die Interaktion zwischen Hörenden und Gehörlosen fördern. Die Anwendung kann auch für den Lernzweck der Gebärdensprache eingesetzt werden.

Angesichts der Tatsache, dass die Gebärdensprache eine visuelle und räumliche Sprache ist, die in verschiedenen Ländern und Regionen der Welt unterschiedliche Varianten und Dialekte aufweist, ist es aufgrund ihrer Vielfalt und Komplexität oft notwendig, sich auf bestimmte Regeln zu konzentrieren, wenn man über Gebärdensprache spricht oder sie erlernt. Daher haben wir uns in diesem aufregenden Projekt für die amerikanische Gebärdensprache (ASL - American Sign Language) entschieden und eine Anwendung entwickelt, die das Gebärdensprachalphabet erkennt und nutzbar macht. Wir sind davon fest überzeugt, dass dies ein wichtiger Schritt ist, um die Kommunikationslücke zu schließen und die Gehörlosengemeinschaft zu unterstützen. Das Gebärdensprachalphabet bildet die Grundlage für die Gebärdensprache, und seine Anerkennung kann das Verständnis und die Akzeptanz dieser einzigartigen Kommunikationsform erhöhen.

In den nächsten Abschnitten werden wir die Hintergründe unseres Projekts, die angewandten Methoden und die Ergebnisse im Detail erklären. Es bereitet uns große Freude, einen Beitrag dazu zu leisten, eine Welt zu schaffen, in der Gehörlose ihre Gedanken und Gefühle genauso frei teilen können wie jeder andere. Obwohl wir noch nicht ganz dort sind, sind wir doch einen Schritt näher.

# Methodik

In unserem Projekt haben wir verschiedene Technologien und Tools eingesetzt, um unsere Methodik zur Gebärdensprache Erkennung zu realisieren:

- Mediapipe: Wir haben die Mediapipe-Bibliothek für die „Echtzeit-Handerkennung“ genutzt. Sie bietet eine zuverlässige Methode zur Identifizierung von Händen in Videoaufnahmen, was entscheidend für die Erfassung vom Alphabet der Gebärdensprache ist.
- Handlandmarken: Für unseren Ziel haben wir Handlandmarken genutzt. Handlandmarken sind spezifische Punkte oder Bereiche auf der Hand, die wir im Videostream erkannt und verfolgt haben. Sie ermöglichen eine präzise Verfolgung der Handbewegungen und Gesten, was für die korrekte Interpretation vom Alphabet der Gebärdensprache wichtig ist.
- Scikit-learn: Scikit-learn ist eine Bibliothek für Machine Learning und wurde in unserem Projekt eingesetzt, um KI-Modelle zu entwickeln und zu trainieren. Scikit-learn ist einfache zu Integrieren und bietet viele Algorithmen zur Verfügung z. B. wie Klassifikation, Regression, Clustering.
- OpenCV: Wir haben OpenCV verwendet, um den Video-Stream der Kamera zu erfassen und in Echtzeit im GUI-Fenster anzuzeigen. OpenCV ermöglicht die Verarbeitung von Bildern und Videos und ist eine wichtige Technologie für Computer Vision-Anwendungen.
- NumPy: NumPy wurde für numerische Berechnungen verwendet. Es ist eine grundlegende Bibliothek für wissenschaftliches Rechnen in Python und erleichtert die Manipulation von Daten und Matrizen.
- Tkinter: Um die grafische Benutzeroberfläche (GUI) zu erstellen. Haben wir Tkinter genutzt, Tkinter ist eine Standard-Bibliothek in Python und geeignet für die Entwicklung von Benutzeroberflächen.
- Python: Die Hauptprogrammiersprache für das Projekt ist Python. Python ist benutzerfreundlich und bietet eine große Auswahl an Bibliotheken und Frameworks für Machine Learning und Computer Vision.

# Datensammlung und –Vorbereitung

## Datensammlung bzw. Bildersammlung

Hier geht es um die Beschaffung von Bildern für die Erstellung eines Datensatzes für die Handgesten-Erkennung. Hierfür dient das Skript „Collect\_imgs.py“ (Siehe den Source code) zur Erfassung von Handgesten vor einer Kamera und automatisches Speichern der Bilder in Ordnern von A bis Z. Das Hauptziel besteht darin, eine umfangreiche Bildersammlung von qualitativen Bildern zu erstellen, der als Grundlage für die spätere Entwicklung und Evaluation des Handgesten-Erkennungsmodells dient.

Das Skript basiert auf der Nutzung von OpenCV zur Erfassung und Verarbeitung des „Live-Videostreams“ und um ein realistisches Erscheinungsbild anzubieten, wird das Kamerabild um die horizontale Achse gespiegelt. Dies erleichtert dem Benutzer das Zeigen der Handgesten, da er das Gefühl hat, in einen Spiegel zu schauen.

Das Skript verfügt über eine Übersicht der Buchstaben von A bis Z, die in einem 7x4-Gitter auf dem Video angezeigt werden. Jeder Buchstabe repräsentiert einen Ordner für die Handgesten-Bilder. Neben jedem Buchstaben ist eine Zahl zu sehen, diese zeigt die Anzahl der bereits aufgenommenen Bilder pro Buchstaben, sodass der Benutzer jederzeit über den Fortschritt der Bildersammlung informiert ist.

Um eine effiziente Bildersammlung zu gewährleisten, es sind ein paar Kriterien zu berücksichtigen. Zunächst soll eine korrekte Vorführung der Handgesten vor der Kamera demonstriert, um klare und präzise Daten bzw. Bilder zu haben. Jedoch soll eine gute Beleuchtung vorhanden sein, welche die natürliche Beleuchtung für bessere Erkennung der Handlandmarken, anbietet. Darüber hinaus soll nur eine Handgeste vor der Kamera demonstriert werden, um Fehler bzw. Datensatz durch ungeeignete Bilder zu schwächen, zu vermeiden.

Für eine benutzerfreundliche Umgebung ist die Bildaufnahme über die Tastatur möglich. Benutzer zeigt Geste vor, drückt den entsprechenden Buchstaben auf der Tastatur, somit wird das Bild in dem entsprechenden Ordner automatisch gespeichert. Bei einer erfolgreichen Aufnahme wird der Benutzer durch ein sofortiges Feedback „Bildzähler Erhöhung“ informiert (Siehe Abbildung 1).

Der durch „Collect\_imgs.py“ erstellte Bildersammlung spielt eine entscheidende Rolle für die Gesamtleistung des Gestenerkennungssystems. Eine reichhaltige Bildersammlung ermöglicht es dem Modell, verschiedene Gesten zu lernen und diese zuverlässig zu klassifizieren. Durch die Berücksichtigung der oben genannten Kriterien kann die Qualität der Daten garantiert werden, was zu einer höheren Genauigkeit und Robustheit des endgültigen Modells führt.

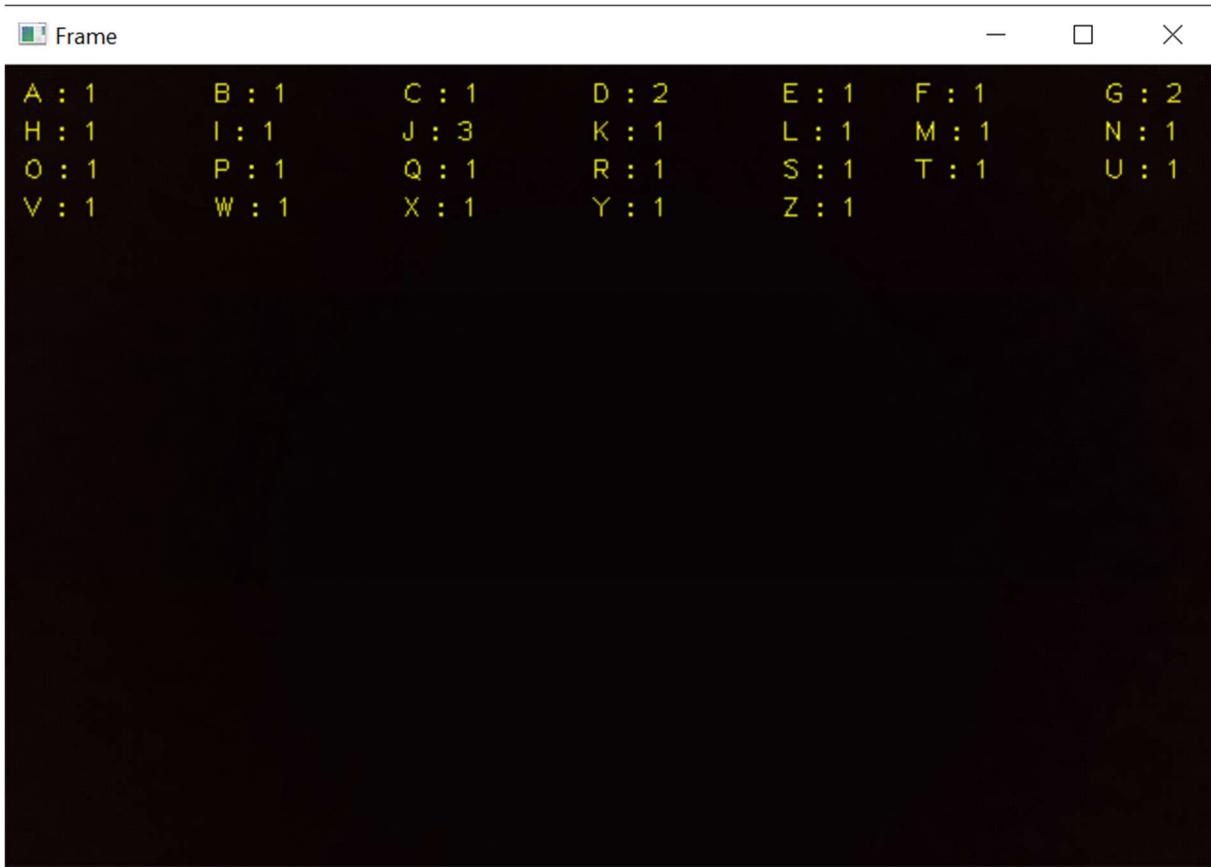


Abbildung 1: Benutzeroberfläche für Bilderaufnahmen

## Datenvorbereitung bzw.-verarbeitung

Für der Datensatzerstellung wird das Skript „Create\_dataset“ (Siehe den Source code) implementiert. In diesem Skript werden mehrere Bibliotheken wie OpenCv, Mediapipe, Numpy und weitere Bibliotheken importiert bzw. verwendet. Das Ziel dieses Skript besteht darin, einen Datensatz für das spätere Training des Machine Learning Models zu generieren. Die Datensatzerstellung gliedert sich in mehreren Schritten. Zuerst werden die Bilder aus dem entsprechenden Verzeichnis ausgelesen und in RGB-Format umgewandelt. Der Grund für die Umwandlung des Bildformats ist, dass die Verarbeitung durch die Mediapipe-Bibliothek reibungslos erfolgen kann. Aus diesen Bildern werden die Handlandmarks mit Hilfe der Mediapipe Bibliothek extrahiert, die zum nächsten Punkt der Bildverarbeitung führen. Da auf den Bildern vorhandene Hände nicht immer eine einheitliche Ausrichtung besitzen, werden die Handlandmarks 0, 9 und 0,13 als essenzielle Referenzpunkte für die Berechnung des Drehwinkels des Bildes verwendet.

Die Berechnung des Drehwinkels und der Drehung des Bildes ist ein wichtiger Schritt. Der zwischen den Orientierungspunkten 0 und 13 gewählte Winkel beeinflusst, wie stark das Bild gedreht wird. Der Zweck dieser Drehung besteht darin, eine konsistente Ausrichtung für alle Frames einer bestimmten Handgeste zu erreichen. Dies trägt zur Konsistenz des Datensatzes

bei und erhöht die Robustheit des resultierenden Modells. Die Berechnungsschritte des Drehwinkels sind wie folgt: zunächst Bestimmung der Richtung der Hand, ob sie nach oben, unten, links oder rechts zeigt. Die Bestimmung der Richtung wird anhand der Landmarks 0 und 9 festgestellt. Aus diesen Informationen und aus dem Winkel zwischen den Landmarks 0 und 13 wird der Drehwinkel berechnet. Zu den Berechnungsschritten werden dafür verwendeten Methoden noch im Laufe der Dokumentation dargestellt und erklärt. Die Bildrotation wird mithilfe der imutils Bibliothek und der Methode „rotate\_bound()“ (Siehe Abbildung 3). Nach der Bildrotation werden die Handlandmarks neu extrahiert und normalisiert, um Unterschiede in Handgrößen und -positionen auszugleichen. Die Normalisierung basiert darauf, die minimalen und maximalen Werte der X- und Y-Koordinaten aus den Handlandmarks Koordinaten zu ermitteln. Dafür werden die X- und Y-Koordinaten von dem entsprechenden minimalen Wert subtrahiert und durch die entsprechende Differenz aus den minimalen und maximalen Werten der X- und Y-Koordinaten geteilt (Siehe Abbildung 2).

```
x = (x - min(x_)) / (max(x_) - min(x_))
y = (y - min(y_)) / (max(y_) - min(y_))
```

Abbildung 2: Handlandmarks Koordinatennormalisierung

Abschließend werden die normalisierten Hand Landmark-Koordinaten in Arrays gespeichert und alle diese erstellten Arrays in einer Pickle-Datei als effizienter Datenspeicher gespeichert.

Der beschriebene Prozess ermöglicht die Erstellung eines qualitativ kohärenten Datensatzes, um ein maschinelles Modell für Handgestenerkennung damit zu trainieren.

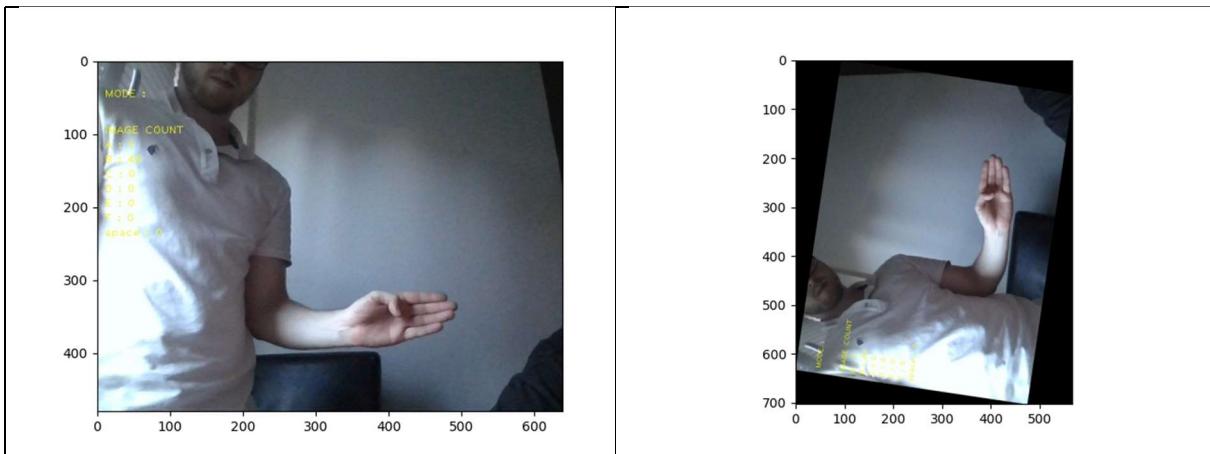


Abbildung 3: Ein Bild vor und nach der Rotation

## Handausrichtung Festlegung

Die Funktion „hand\_orientation“ ist für Handrichtungsbestimmung programmiert. Diese Methode kann anhand zwei Handlandmarks die Handausrichtung bestimmen, welche später

für die Bestimmung des Bilddrehwinkels eingesetzt wird. Die verwendete Landmark-Punkte sind als „Landmarks\_0“ und „Landmark\_9“ definiert (Siehe Abbildung 4). Jedes Landmark wird in X- und Y-Koordinate aufgeteilt. Ein Anstieg (Steigung) „m“ wird berechnet, um das Verhältnis von Höhen- und Breitenunterschieden zwischen den Landmarks zu ermitteln. Ein spezieller Fall ist hier zu betrachten, wenn die Differenz zwischen X9 und X0 kleiner als 0,05 ist, wird dann „m“ auf einen großen Wert (1 Milliarde) gesetzt, um extreme Flachheit zu behandeln. Somit wird die Handausrichtung basierend auf der Steigung „m“ festgelegt (Siehe Abbildung 6):

- Wenn „m“ zwischen 0 und 1 liegt, deutet dies auf eine schräge Linie hin, und „Rechts“ oder „Links“ wird basierend auf der x-Koordinate von Punkt 9 (x9) im Vergleich zu Punkt 0 (x0) bestimmt (Siehe Abbildung 6 „dies gilt auch für links bei 180 Umdrehung“).
- Wenn „m“ größer als 1 ist, handelt es sich um eine steile Linie, und „Oben“ oder „Unten“ wird basierend auf der y-Koordinate von Punkt 9 (y9) im Vergleich zu Punkt 0 (y0) festgelegt (Siehe Abbildung 5 „dies gilt auch für unten bei 180 Umdrehung“).

Die ermittelte Handausrichtung wird als String zurückgegeben, d.h., „Rechts“, „Links“, „Oben“ oder „Unten“.

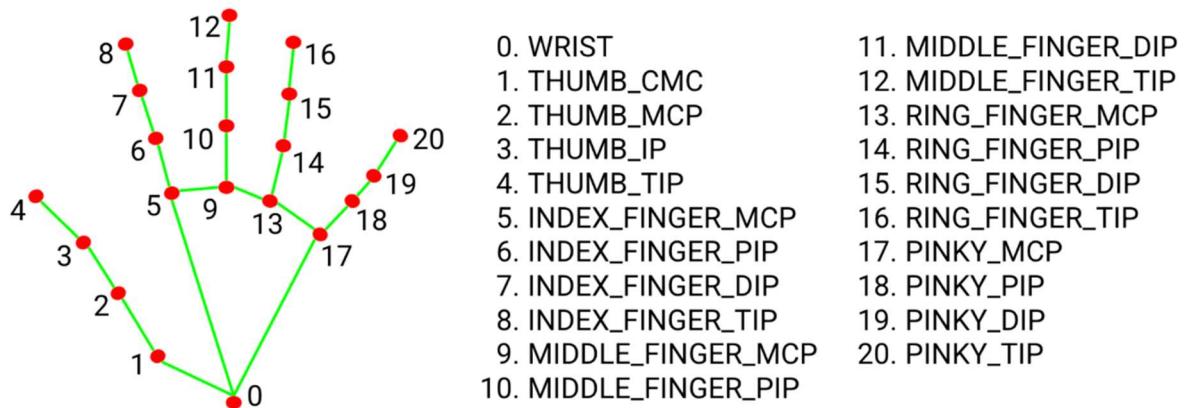


Abbildung 4: Handlandmarks [7]

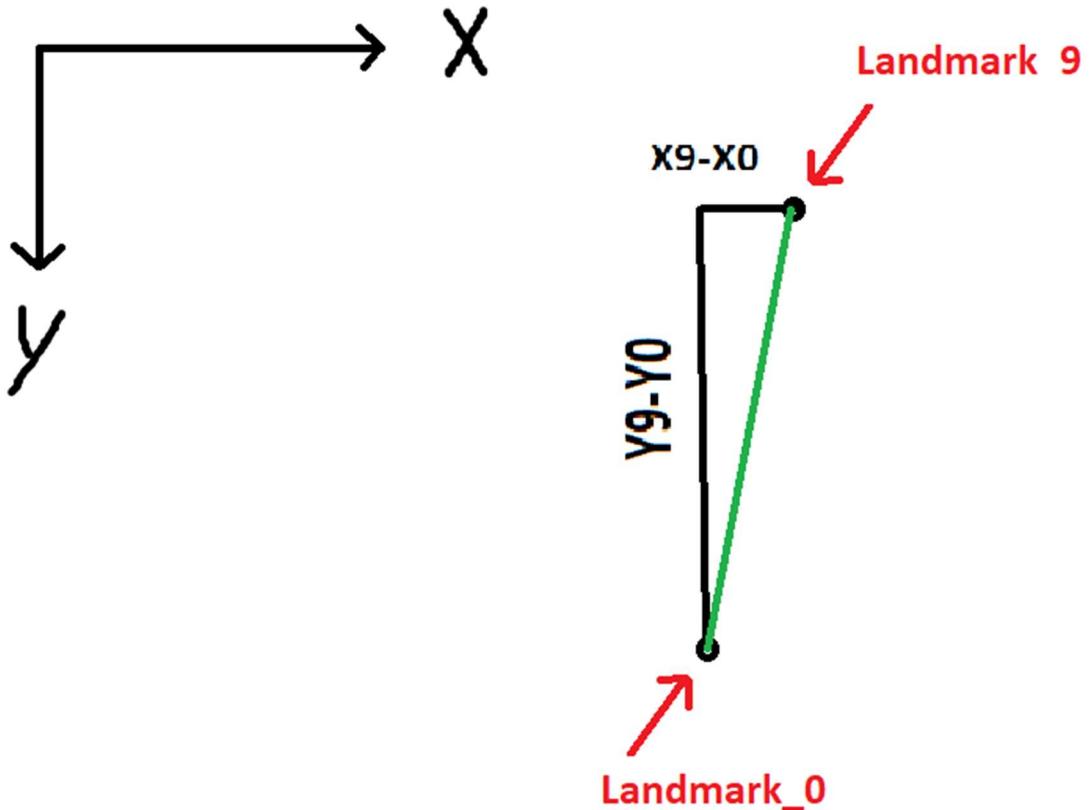


Abbildung 5: Handausrichtung nach Oben

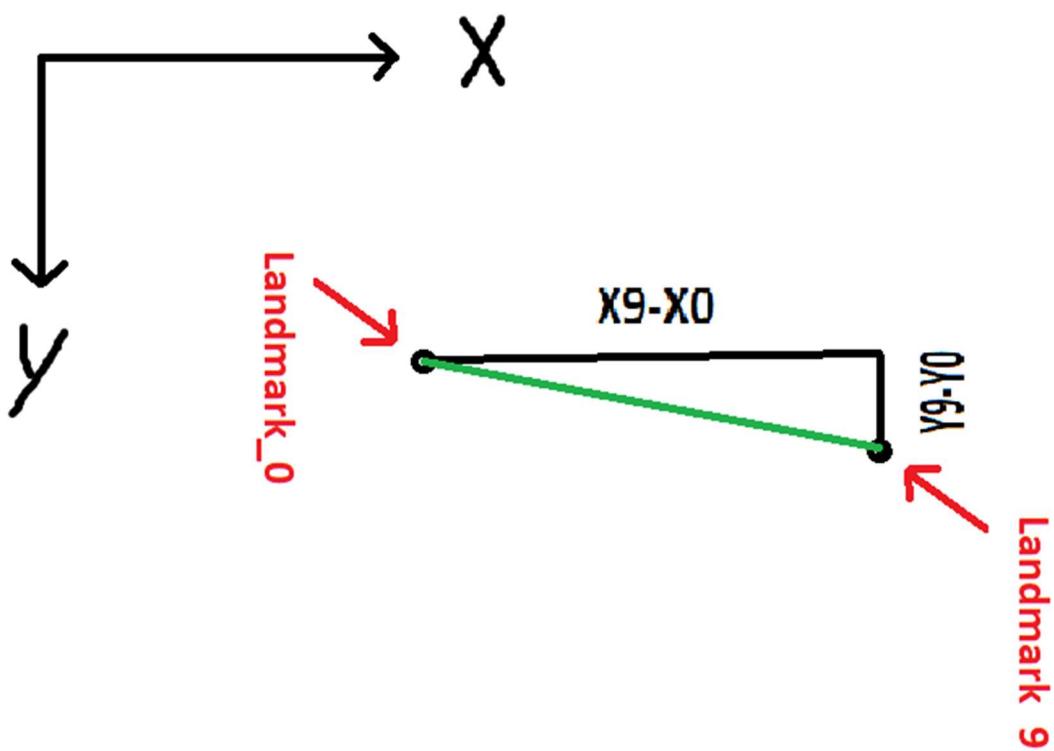


Abbildung 6: Handausrichtung nach rechts

```

9  def hand_orientation(landmark_0, landmark_9):
10     x0, y0 = landmark_0.x, landmark_0.y
11     x9, y9 = landmark_9.x, landmark_9.y
12
13     if abs(x9 - x0) < 0.05: # da tan(0) --> ∞
14         m = 1000000000
15     else:
16         m = abs((y9 - y0) / (x9 - x0))
17
18     if m >= 0 and m <= 1:
19         if x9 > x0:
20             return "Right"
21         else:
22             return "Left"
23     if m > 1:
24         if y9 < y0:
25             return "Up"
26         else:
27             return "Down"
28

```

Abbildung 7: Handausrichtung Erkennungscode „hand\_orientation“

## Winkel Berechnung

Die Funktion „angle“ (Siehe Abbildung 8): ist eine komplementäre Funktion der Funktion „angle\_of\_rotation“ (Siehe Abbildung 9), da in dieser Funktion den Winkel zwischen dem Punkt 0 und dem Ringfinger (Punkt 13) einer Hand berechnet wird (Siehe Abbildung 4). Der Winkel zwischen diesen beiden Orientierungspunkten wird mithilfe trigonometrischer Berechnungen bestimmt. Dies erfolgt mithilfe der Funktion np.arctan(), um das Verhältnis der vertikalen Differenz (y-Koordinaten) zur horizontalen Differenz (x-Koordinaten) zwischen den Landmarks zu berechnen. Die Winkelberechnung erfolgt in Bogenmaß, aus diesem ist eine Winkelumrechnung in Grad notwendig. Für die Winkelumrechnung wird den berechneten Wert in Bogenmaß mit 180/np.pi multipliziert. So wird der Winkel in Grad zurückgegeben (Sehen Sie Abbildung 8).

```

65  # Funktion zur Berechnung des Winkels zwischen Ringfinger und Kleiner Finger
66  def angle():
67      kleiner_finger_punkt = hand_landmarks.landmark[0]
68      ringfinger_punkt = hand_landmarks.landmark[13]
69      angle = np.arctan((ringfinger_punkt.y - kleiner_finger_punkt.y) /
70                         (ringfinger_punkt.x - kleiner_finger_punkt.x)) * 180 / np.pi
71  return angle

```

Abbildung 8: Funktionscode "angle"

## Drehwinkel Bestimmung

Die Funktion „angle\_of\_rotation“ hat die Aufgabe, den Drehwinkel für die Rotation eines Bildes basierend auf der ermittelten Handrichtung und den Winkel zwischen den Handlandmarks „0“ und „13“ zu berechnen. Als Eingabeparameter bekommt diese Funktion zwei. Der erste Parameter ist der Rückgabewert von „hand\_orientation“ (Siehe Abbildung 7), und das ist die Handausrichtung, welche „Up“, „Down“, „Left“ oder „Right“ wäre. Der zweite Parameter ist der berechnete Winkel aus der Funktion „angle“ basierend auf der Handlandmarks „0“ und „13“. Die Drehwinkelberechnung ist je nach Handausrichtung anderes, da pro Handausrichtung zwei Position noch zu betrachten (Sehen Sie Abbildung 9,10 und 11). Abschließend wird der berechnete Drehwinkel zurückgegeben, um das Bild gemäß der Handausrichtung und des Winkels zwischen den Landmarks zu rotieren.

```

30 def angle_of_rotation(hand_orientation,angle_0_13):
31     if hand_orientation == "Up":
32         if hand_landmarks.landmark[9].x > hand_landmarks.landmark[0].x:
33             angle = abs(angle_0_13) - 90
34             return angle
35         else:
36             angle = 90 - abs(angle_0_13)
37             return angle
38
39     elif hand_orientation == "Down":
40         if hand_landmarks.landmark[9].x > hand_landmarks.landmark[0].x:
41             angle = 270 - abs(angle_0_13)
42             return angle
43         else:
44             angle = 90 + abs(angle_0_13)
45             return angle
46
47     elif hand_orientation == "Right":
48         if hand_landmarks.landmark[0].y > hand_landmarks.landmark[9].y:
49             angle = abs(angle_0_13) + 270
50             return angle
51         else:
52             angle = 270 - abs(angle_0_13)
53             return angle
54
55     elif hand_orientation == "Left":
56         if hand_landmarks.landmark[0].y > hand_landmarks.landmark[9].y:
57             angle = 90 - abs(angle_0_13)
58             return angle
59         else:
60             angle = abs(angle_0_13) + 90
61             return angle
62     else:
63         print("Default case")

```

Abbildung 9: Funktionscode "angle\_of\_rotation"

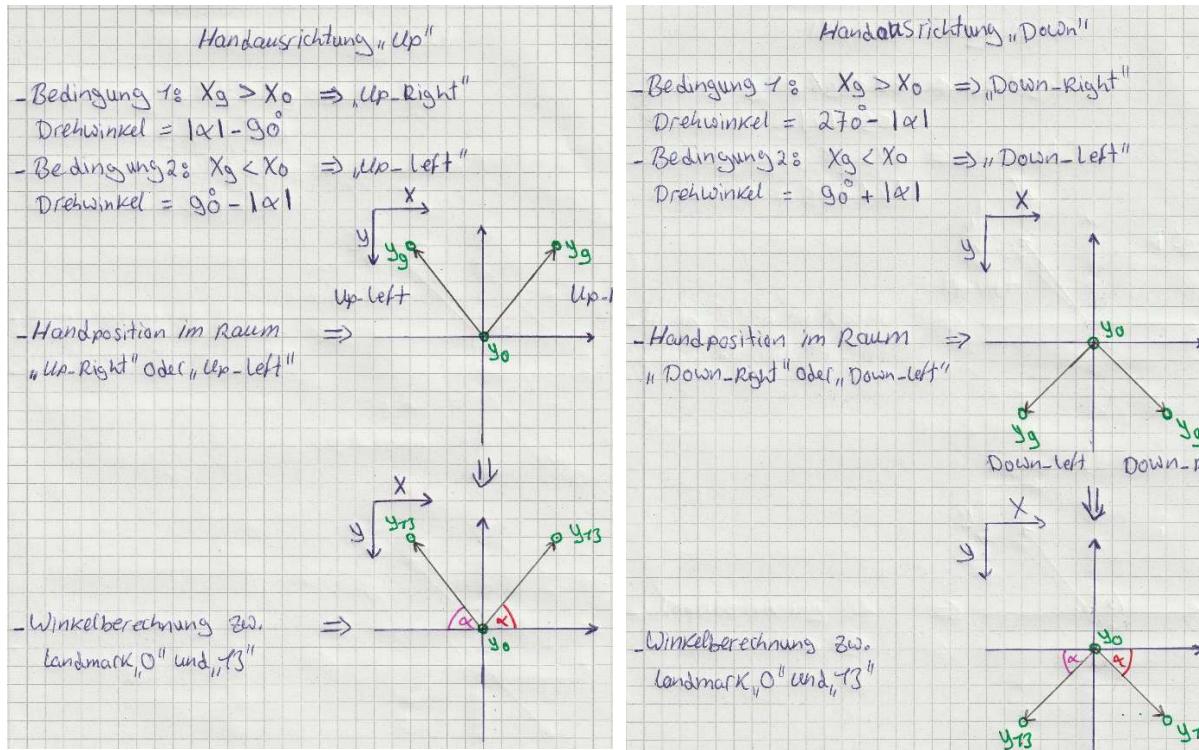


Abbildung 10: Drehwinkelbrechungsprinzip mit einer Handausrichtung "Up" und "Down"

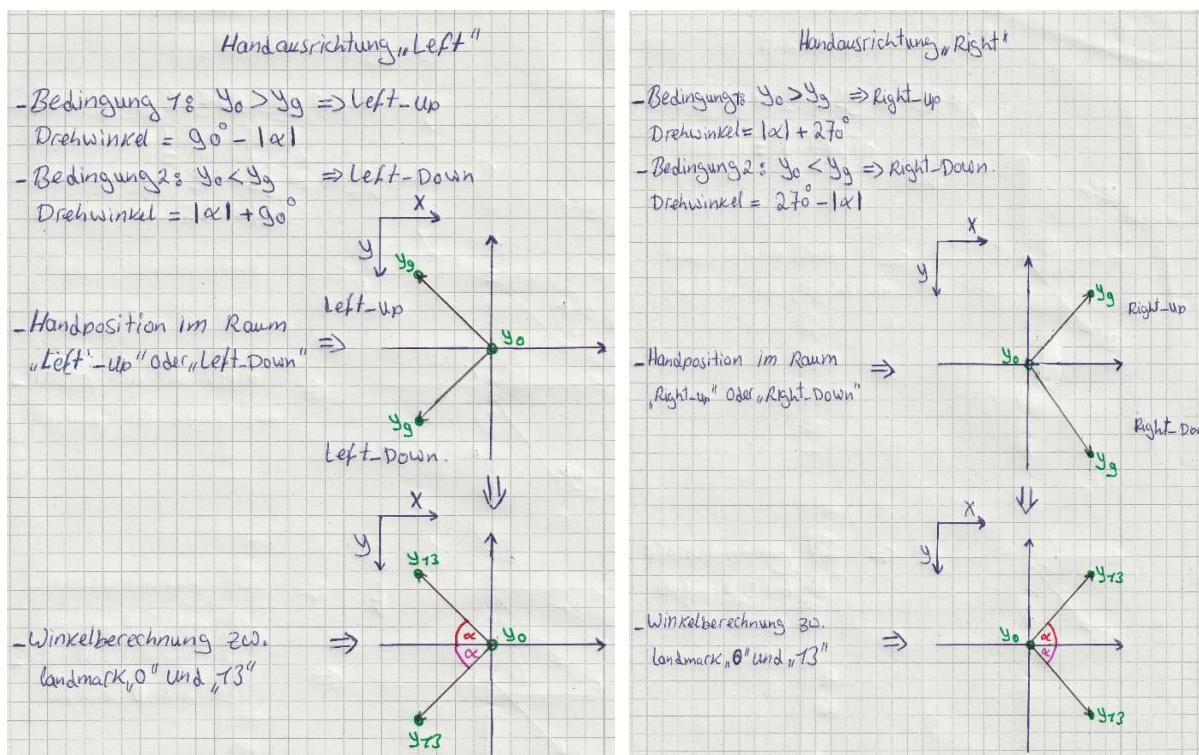


Abbildung 11: Drehwinkelbrechungsprinzip mit einer Handausrichtung "Right" und "Left"

# Modellauswahl und -bewertung

In diesem Abschnitt wird ein Klassifikationsmodell für den zuvor erstellten Datensatz entworfen und analysiert. Zunächst wurden die für das Training relevanten Daten aus der Pickle-Datei geladen und mit Hilfe der NumPy-Bibliothek in NumPy-Arrays umgewandelt. Anschließend werden die Daten in Trainings- und Testsätze aufgeteilt. Der Trainingsprozess wird mit verschiedenen Lernalgorithmen durchgeführt, um die Leistungsfähigkeit der Algorithmen zu beobachten. Anschließend wird die Genauigkeit des Modells auf den Trainingsdaten evaluiert. Um den geeigneten Lernalgorithmus auszuwählen, werden Vorhersagen für die Testdaten mit den verschiedenen Modellen erstellt. Dies geschieht mit Hilfe von gängigen Werkzeugen zur Evaluierung von Modellvorhersagen und zum Aufzeigen von Schwachstellen, wie z.B. der Konfusionsmatrix und dem Klassifikationsreport. Abschließend wird das ausgewählte Modell in einer Pickle-Datei gespeichert, um es für die zukünftige Verwendung aufzubewahren.

## Modellauswahl

### 1. Problem Darstellung und geeigneter ML-Art

Um das passende Modell auszuwählen, müssen wir zunächst verstehen, welches Problem das Modell lösen muss und welche Lernalgorithmen für diesen Fall geeignet sind. In unserem Fall muss das Modell in der Lage sein, anhand numerischer Daten der Handlandmarks zwischen Gebärdensprachbuchstaben zu unterscheiden und zu versuchen, darin Pattern zu finden. Daher ist unser Problem in diesem Fall im Bereich des Machine Learning als **Klassifikation** bekannt. Da es sich in unserem Fall um ein Klassifizierungsproblem handelt und wir über „labeled Data“ verfügen, verwenden wir Supervised Machine Learning.

### 2. Größe des Datensatzes

In unserem Fall enthält der Datensatz etwa 11.000 Zeilen und jede Zeile enthält 42 Merkmale (Features). Die Größe dieses Datensatzes ist mittelgroß, wenn es um Supervised Machine Learning geht. Diese Größe reicht aus, um Modelle effektiv zu trainieren. Beispielsweise der SVM-Algorithmus, der mit einem kleinen Datensatz besser funktioniert als Random Forest, der besser mit größeren Datensatz funktioniert.

„SVM can achieve better results with smaller datasets with higher dimensions, as large datasets may take a longer time“[1]

### 3. Anforderung

- Das erstellte Modell muss möglichst in der Lage sein, Buchstaben mit einer geringen Fehlerquote zu unterscheiden.
- Das Modell muss in der Lage sein, Buchstaben der rechten oder der linken Hand zu erkennen, jedoch nicht beides.
- Das Modell muss über eine ausreichende Genauigkeit verfügen, um trotz Drehung der Hand die Buchstaben erkennen zu können.
- Das Modell sollte logisch sein, das heißt, die angezeigten Ergebnisse sollten logisch und nicht zu optimistisch sein.

### 4. Geeignete Lernalgorithmen

- **Support Vector Machine** : Dieses Modell, genauer gesagt „Linear Support Vector Classification(**svm.LinearSVC**)“ , sollte laut „scikit-learn algorithm cheat-sheet [2]“ gut sein, wenn die Anzahl der Datensätze (Samples) weniger als 100.000 beträgt. Wir werden jedoch auch den nichtlinearen Algorithmus namens „C-Support Vector Classification(**svm.SVC**)“ testen. Da das lineare Beispiel möglicherweise auf einige Probleme stößt, insbesondere bei der Klassifizierung mehrerer Klassen(Multiclass classification)[3]. Außerdem kann SVM mit kleineren Datensätzen mit höheren Dimensionen bessere Ergebnisse erzielen[4] (Siehe Abbildung 12)
- **Random Forest**: Die Verwendung dieses Lernalgorithmus hat Vorteile, unter anderem verhindert er eine Überanpassung(Overfitting) und eignet sich sehr gut für das Klassifizierungsproblem mehrerer Klassen[5].
- **SGDClassifier(stochastic gradient descent)**: Obwohl der Datensatz kleiner ist als für diesen Algorithmus erforderlich. Aber es funktioniert gut für Datensätze mit höheren Dimensionen.

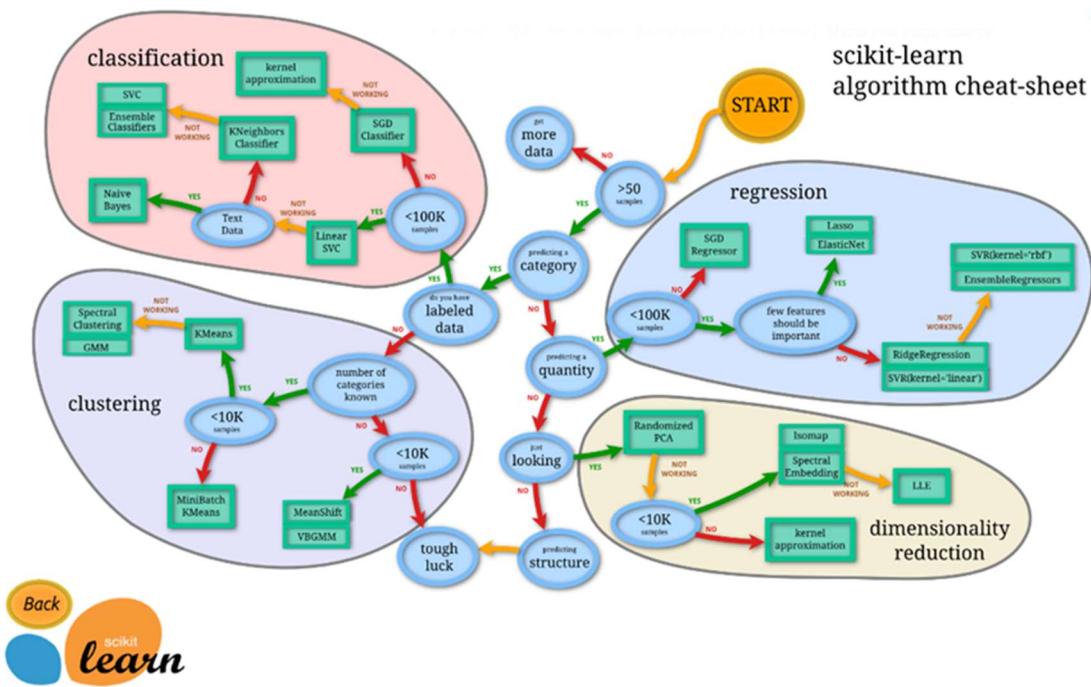


Abbildung 12: scikit-learn algorithm cheat-sheet

#### 4- Experimente

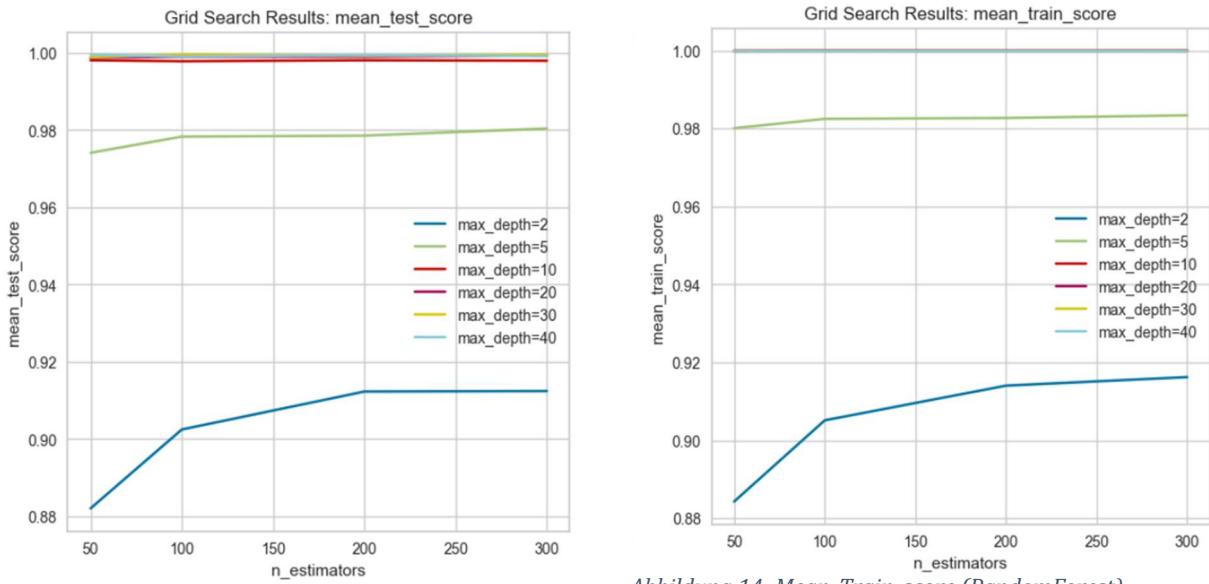
Am Ende muss jedes erstellte Modell separat getestet werden, um seine Leistung zu bewerten und festzustellen, ob es tatsächlich die Anforderungen erfüllt.

### Hyperparameter-Tuning & Leistung Bewertung:

#### Random Forest:

Dieser Lernalgorithmus verfügt über mehrere wichtige Variablen, die zur Verbesserung der Modellleistung beitragen:

- 1- Anzahl der Entscheidungsbäume(n\_estimators): Diese Parameter kann die Effizienz des Modells steigern. Es ist jedoch zu beachten, dass eine sehr große Zahl nicht die richtige Idee ist, sondern der Algorithmus für die Lösung viel komplizierter wird und daher einen viel größeren Rechenaufwand erfordert. Ab einer bestimmten Anzahl von Entscheidungsbäumen wird die Auswirkung auf die Genauigkeit jedoch sehr gering oder gar nicht mehr vorhanden.[6][7]
- 2- maximale Tiefe(max\_depth): Diese Parameter bestimmt die maximale Tiefe des Baums. Dies kann die Genauigkeit des Modells stark beeinflussen. Die Wahl einer großen Tiefe kann jedoch zu einer Überanpassung führen



In Abbildung 13 und 14 ist zu erkennen, wie sich die Test- und Train Score in Abhängigkeit von der maximalen Tiefe und der Entscheidungsbäume verhält. Daher fällt auf:

- 1- Die Anzahl der Bäume über 100 hat keinen großen Einfluss auf die Ergebnisse, insbesondere wenn die Tiefe groß ist. Dies wurde in einer Tiefe von zwei und fünf teilweise beobachtet, in einer Tiefe von 10 bis 40 jedoch vollständig.
- 2- Die Tiefe des Baumes spielt eine sehr große Rolle, und dies wurde bei einer Tiefe von zwei beobachtet, wo die Genauigkeit ungefähr gleich 92% war. Bei einer Tiefe von fünf waren es 98%. Bei darüberhinausgehenden Werten lag die Genauigkeit bei 99 %. Dies weist auf den erheblichen Einfluss der Tiefe auf die Genauigkeit hin und kann zu einer Überanpassung (Overfitting) führen.[8]

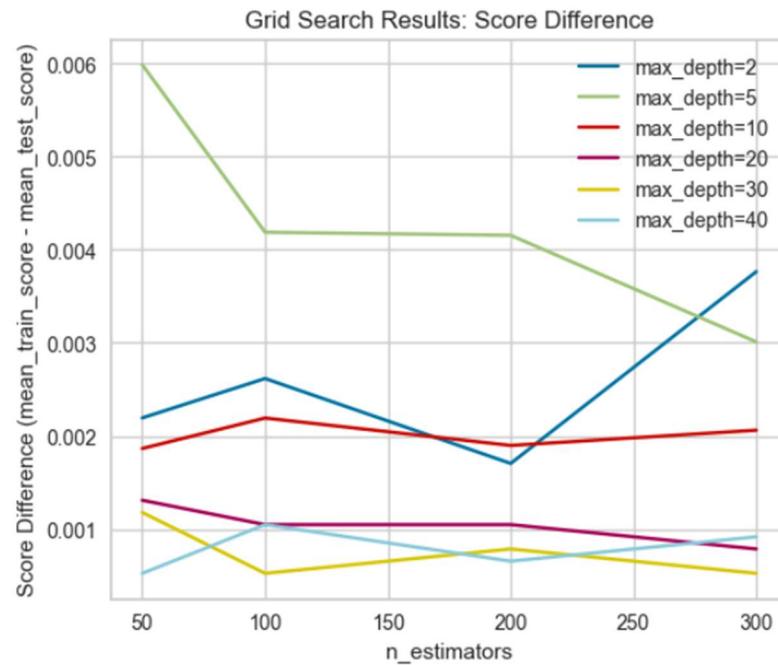


Abbildung 15: Score Difference(Train Score - Test Score) für Random Forest

In Abbildung 15 ist zu kennen, wie groß ist der Unterschied zwischen der Train- und Test-Score. Dies kann ein Hinweis auf Overfitting oder Underfitting sein. Ein kleiner Unterschied kann auf ein gutes, allgemeines Modell hinweisen. Der größte Unterschied besteht darin, dass sich ein Modell nicht wie gewünscht verhält. Allerdings muss das Modell mit hoher Genauigkeit ausprobiert werden, um herauszufinden, ob es wirklich so ist, wie die Ergebnisse zeigen.

## Hyperparameter-Einstellungen

Laut GridSearchCV weist das Modell mit den folgenden Variablen die beste Leistung und Genauigkeit auf: **Max\_depth =30** und **n\_estimators =100** mit einer Accuracy von 100% Und Kreuzvalidierung von 99.869%. Nach der Konfusionsmatrix ist die Genauigkeit sehr hoch und das Modell liefert fast immer die richtigen Ergebnisse, was experimentell nicht nachgewiesen wurde.

Cross-Validation Accuracy : 99.89528795811518 %  
Accuracy : 100.0 %

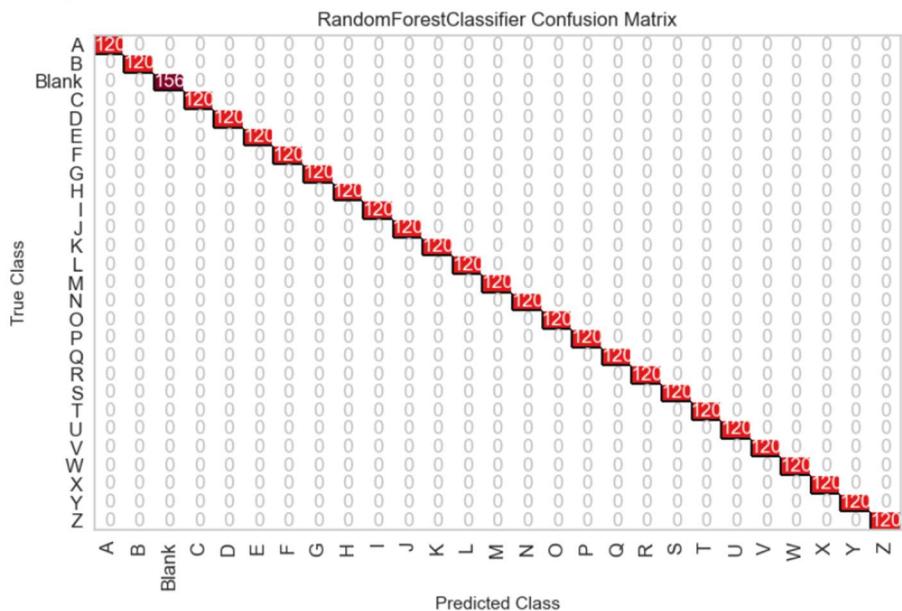


Abbildung 16: Konfusionsmatrix (max\_depth:30 und n\_estimators:100) Random Forest

- Experimentell hat das Modell schlechte Ergebnisse geliefert. Aus diesem Grund hat das Modell mit einer maximalen Tiefe von 9 die besten Ergebnisse mit den folgenden Werten erzielt.

Cross-Validation Accuracy : 99.69897000058211 %  
Accuracy : 99.75579975579976 %

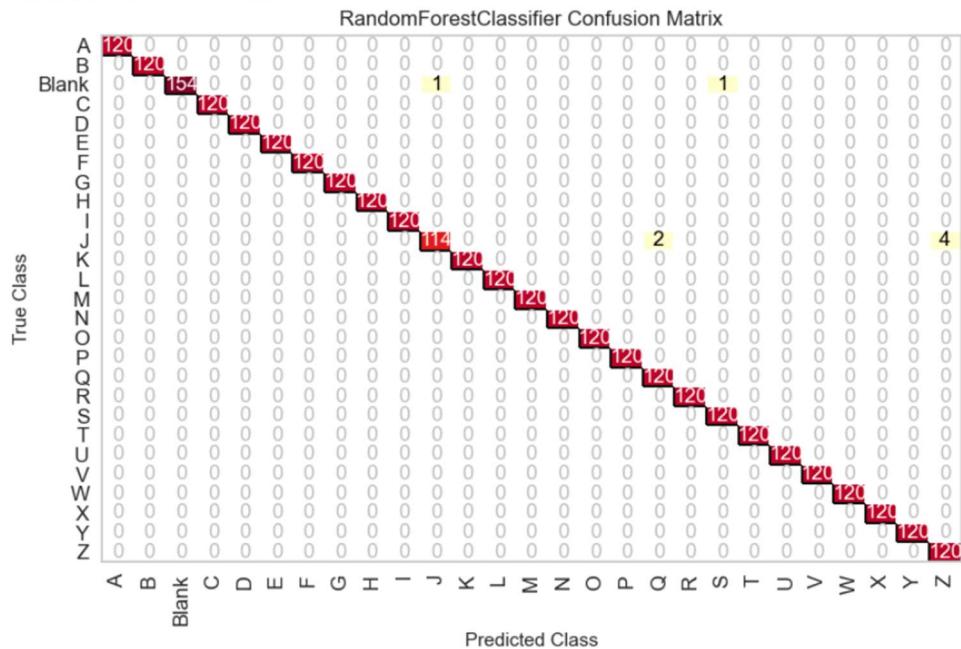


Abbildung 17: Konfusionsmatrix (max\_depth: 9 und n\_estimators :100) Random Forest

## Support Vector Machine (SVM):

Dieser Lernalgorithmus verfügt über wichtigen Parametern, darunter:

- 1- **C** (Regularisierungsparameter): Ein Regularisierungsparameter, der die Toleranz des Modells für die Fehlklassifizierung von Datenpunkten festlegt. Je höher der Wert von Hyperparameters 'C', desto geringer ist die Toleranz. Er spielt eine wichtige Rolle, unabhängig davon, ob es sich um eine lineare oder nichtlineare Entscheidungsgrenze handelt. Allerdings ist Vorsicht geboten, da die Wahl kleiner C-Werte zu einem Underfitting führen kann, und das Gleiche gilt, wenn die Wahl großer C-Werte zu einer Overfitting führen kann [9]
- 2- **Kerneltyp**: Es gibt drei Typen: linear und nicht linear (RBF, Polynom). Die Wahl des Kernels hängt von der Art des Problems ab. Bei großen Datensätzen arbeitet der lineare Kernel schneller als der nichtlineare, während des nicht linearen Kernels die Erstellung komplexer Entscheidungsgrenzen ermöglicht.
- 3- **Gamma**: Dieser Parameter hat einen Einfluss auf die Form der Entscheidungsgrenze, da große Gamma-Werte die Entscheidungsgrenze komplexer machen und umgekehrt. Außerdem ist diese Variable nur bei der nichtlinearen Entscheidungsgrenze von Bedeutung. Allerdings ist Vorsicht geboten, da die Wahl kleiner Gamma -Werte zu einem Underfitting führen kann, und das Gleiche gilt, wenn die Wahl großer Gamma -Werte zu einer Overfitting führen kann.

### *Linear SVM*

Nun wird das lineare Modell mit verschiedenen C-Werten getestet, was zu der folgenden Abbildung führt. (Siehe Abbildung 18)

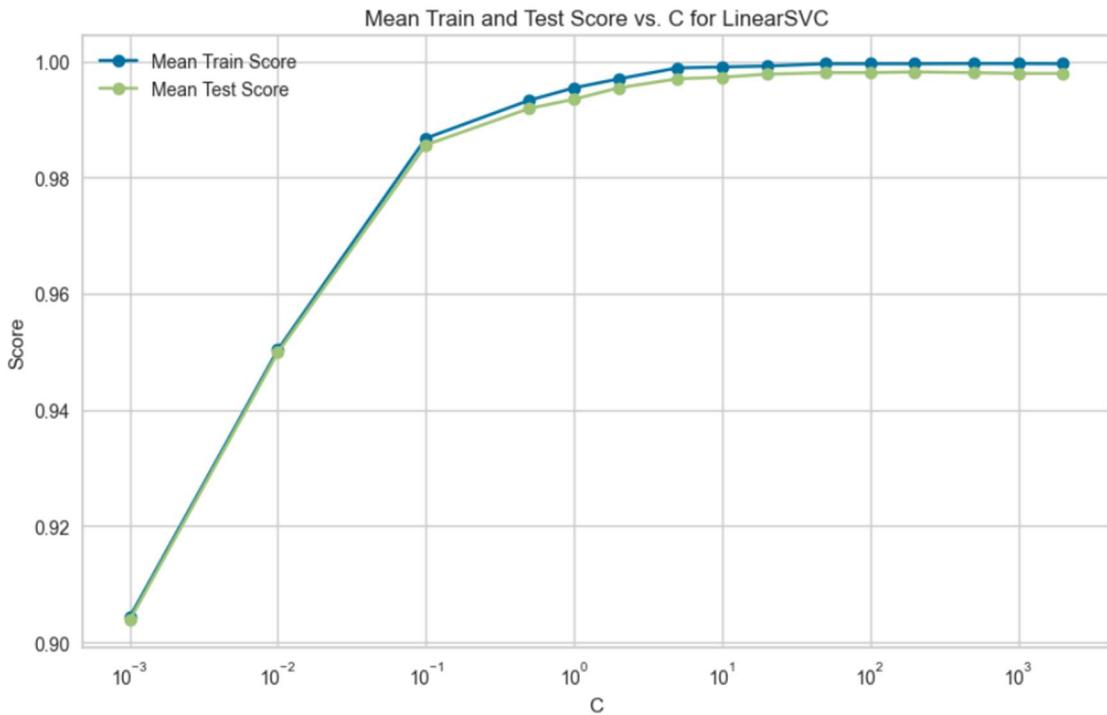


Abbildung 18: Mean Train und Test Score vs. C (LinearSVC)

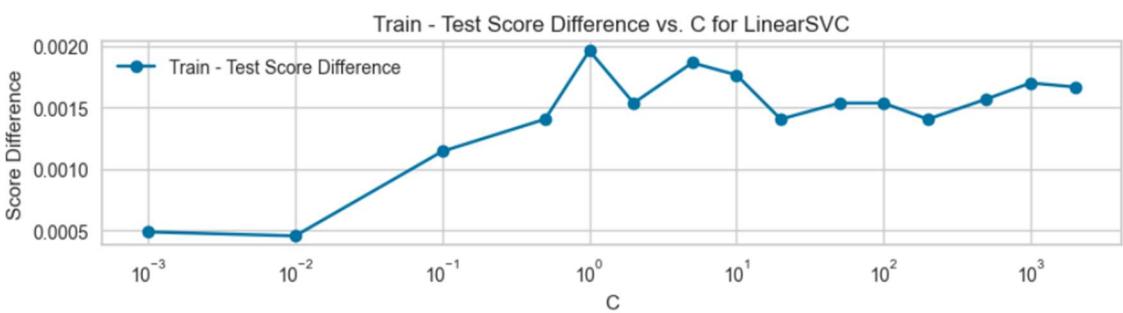


Abbildung 19: Train - Test Score Difference vs. C (LinearSVC)

Folgende Beobachtungen lassen sich aus den Abbildungen 18 und 19 ableiten:

- 1- Für kleine C-Werte [0,001, 0,01, 0,1]: Zunächst muss festgestellt werden, dass das Lernen nicht wie erwartet funktioniert hat: Trotz des sehr geringen Unterschieds zwischen Trainings- und Testscore ist die Genauigkeit nicht ausreichend. Für die oben genannten C-Werte lag die Genauigkeit für die Trainingsdaten zwischen 90,5% und 98,6%.
- 2- Die mittleren C-Werte [0,5, 1, 2, 5, 10] zeigen, dass das Lernen mit den Trainingsdaten effizienter geworden ist, da der Trainingsscore gestiegen ist. Es kann auch festgestellt werden, dass es eine Diskrepanz zwischen den Trainingsdaten und den Testdaten gibt, da die Testpunktzahl gesunken ist. Die Leistung des Modells in dieser Kategorie ist besser als in der ersten Gruppe.

- 3- Für die hohen Werte von C [20, 50, 100, 200, 500, 1000] ist zu beachten, dass das Lernen aus den Trainingsdaten optimal ist und das Verhältnis nahe bei 1 liegt. Der Unterschied in der Genauigkeit zwischen Trainings- und Testdaten ist ebenfalls signifikant, aber geringer als in der zweiten Kategorie. Diese Kategorie deutet darauf hin, dass das Modell stabil sein könnte, da ab einem C-Wert von 20 die Differenz fast gleich geblieben ist und um den Wert 0,0015 schwankt.

## Hyperparameter-Einstellungen (LinearSVC)

Aus der Grid-Suche(rank\_test\_score) geht hervor, dass das Modell den besten TestScore bei einem Hyperparameter 'C' von 50 liefert .

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
A	0.98	1.00	0.99	120
B	1.00	1.00	1.00	120
Blank	1.00	0.97	0.99	156
C	0.99	1.00	1.00	120
D	1.00	1.00	1.00	120
E	1.00	1.00	1.00	120
F	1.00	1.00	1.00	120
G	1.00	1.00	1.00	120
H	1.00	1.00	1.00	120
I	1.00	1.00	1.00	120
J	1.00	0.98	0.99	120
K	1.00	1.00	1.00	120
L	0.99	1.00	1.00	120
M	1.00	1.00	1.00	120
N	1.00	1.00	1.00	120
O	1.00	1.00	1.00	120
P	1.00	1.00	1.00	120
Q	1.00	1.00	1.00	120
R	1.00	1.00	1.00	120
S	0.99	1.00	1.00	120
T	1.00	1.00	1.00	120
U	1.00	1.00	1.00	120
V	1.00	1.00	1.00	120
W	0.98	1.00	0.99	120
X	1.00	1.00	1.00	120
Y	1.00	1.00	1.00	120
Z	1.00	0.99	1.00	120
<b>accuracy</b>			<b>1.00</b>	3276
<b>macro avg</b>		<b>1.00</b>	<b>1.00</b>	<b>3276</b>
<b>weighted avg</b>		<b>1.00</b>	<b>1.00</b>	<b>3276</b>

Abbildung 20: SVC mit C='50' (Classifier Report)

Cross-Validation Accuracy : 99.79059303723133 %  
 Accuracy : 99.78632478632478 %

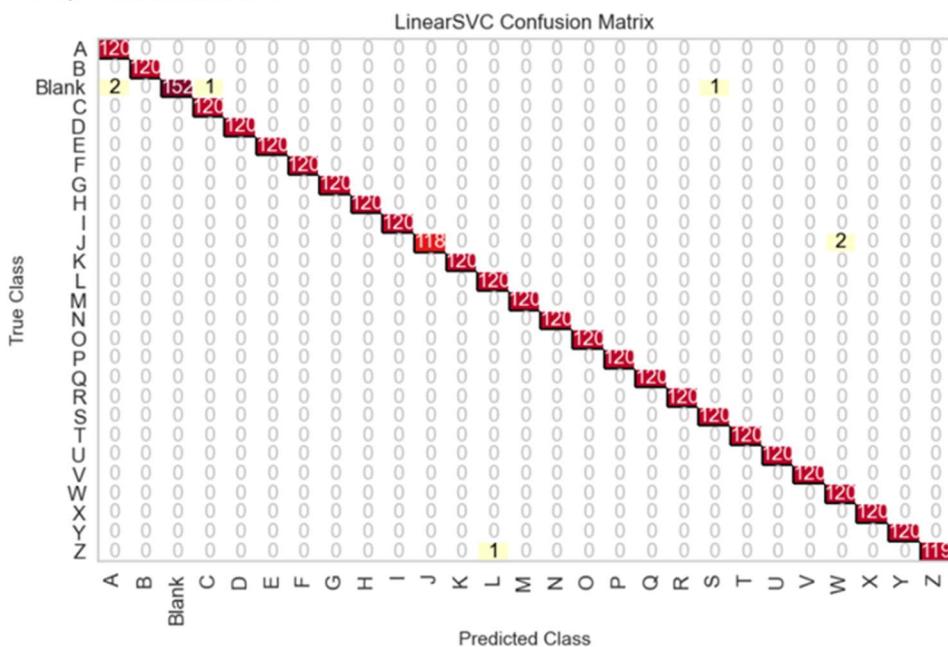


Abbildung 21: Konfusionsmatrix für  $C=50$  (LinearSVC)

Daher wird die oberen Konfusionsmatrix für alle Klassen dargestellt und das Modell experimentell getestet, um zu überprüfen, ob es gute Ergebnisse liefert. Es ist zu erkennen, dass die ausgewählten Algorithmen mit den festgelegten Parametern Schwierigkeiten bei der Erkennung einiger Klassen aufweisen. Experimentelle Tests haben gezeigt, dass das Modell mit diesen Parametern nicht ausreichend stabil ist (siehe Abbildungen 20 und 21).

Das letzte Modell zeigte nicht die beste Leistung, da die Difference zwischen dem Train- und dem Testscore nicht die kleinste ist. Daher werden die berechneten Werte aus Abbildung 19 entnommen, um zu sehen, bei welchen C-Werten die Difference am kleinsten ist. Es ergeben sich folgende Werte.

param_C	mean_test_score	score_difference
2	0.1	0.001145
11	200.0	0.001407
8	20.0	0.001407
3	0.5	0.001407
10	100.0	0.001538

Abbildung 22: Kleinste Differenz (Trainscore - Testscore)

In Abbildung Nr. 23 werden die Kreuzvalidierungsgenauigkeit und die Test-Genauigkeit berechnet, um zu vergleichen, welche die beste Kreuzvalidierungsgenauigkeit ergibt.

Von den fünf unten erwähnten Modellen weist dieses Modell ( $C=100$ ) die beste Kreuzvalidierung auf. Wir haben jedoch alle fünf Modelle manuell getestet und das

Ergebnis war das gleiche. Die Modellnummer bezieht sich auf den Index in der Grid-Suche-Tabelle (linearSVC).

---

```
Model Nr. 2 mit C = 0.1
Cross-Validation Accuracy : 98.49493560791538 %
Accuracy : 98.96214896214897 %
```

```
Model Nr. 11 mit C = 200
Cross-Validation Accuracy : 99.80367348196644 %
Accuracy : 99.75579975579976 %
```

```
Model Nr. 8 mit C = 20
Cross-Validation Accuracy : 99.77749547149524 %
Accuracy : 99.75579975579976 %
```

```
Model Nr. 3 mit C = 0.5
Cross-Validation Accuracy : 99.12312225421948 %
Accuracy : 99.42002442002442 %
```

```
Model Nr. 10 mit C = 100
Cross-Validation Accuracy : 99.80367348196646 %
Accuracy : 99.72527472527473 %
```

Bestes LinearSVC Modell (C = 100 und Cross-Validation Accuracy: 0.9980367348196646)

Abbildung 23: Bestes LinearSVC Modell mit (C = 100)

Cross-Validation Accuracy : 99.80367348196646 %  
Accuracy : 99.72527472527473 %

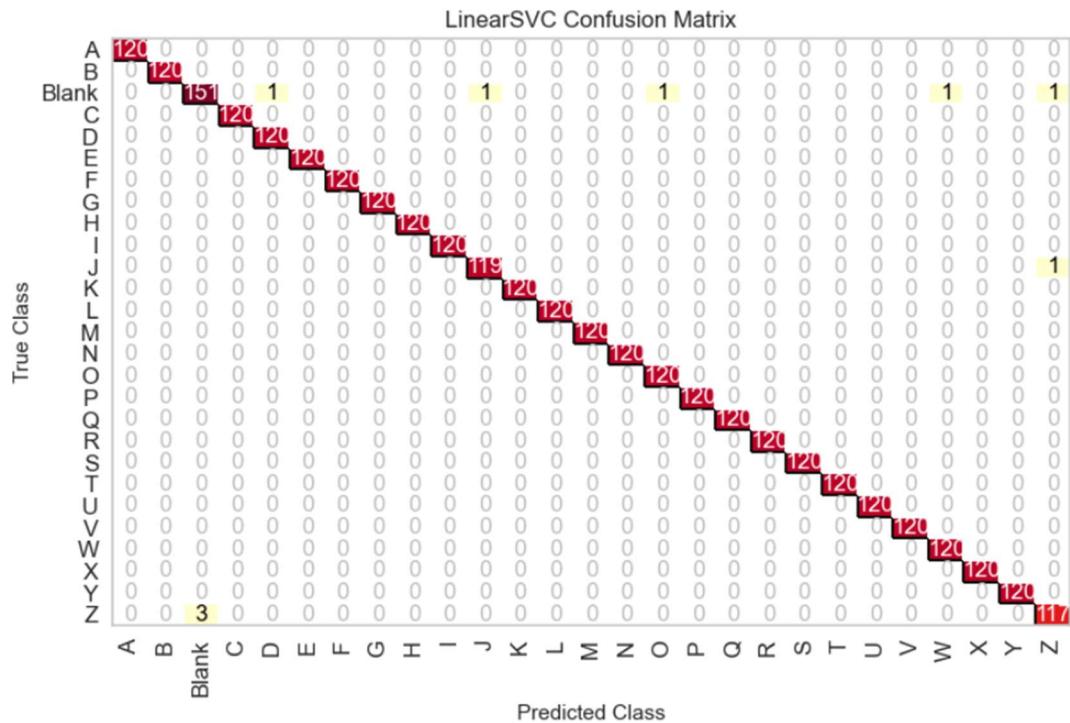


Abbildung 24: Konfusionsmatrix für C='100'(LinearSVC)

Trotzdem, dass das Modell mit  $C='100'$  die beste Validierung aufweist, ist der Unterschied kaum wahrnehmbar, da die Validierung ab dem Wert  $C=20$  fast gleichgeblieben ist. Daher erfüllt das lineare Modell (LinearSVC) nicht die Erwartungen (Siehe Abbildung 23 & 24 )

## Nichtlineare SVM

Gemäß dem "Scikit-learn Algorithmus-Cheat-Sheet" ist ein LinearSVC eine geeignete Wahl für Klassifikationsaufgaben mit weniger als 100.000 Samples und für kleine Datensätze im Allgemeinen. Die linearen Modelle haben akzeptable Ergebnisse geliefert, jedoch nicht wie erwartet. Daher wird nun die nichtlineare SVM (SVC mit rbf Kernel) getestet.

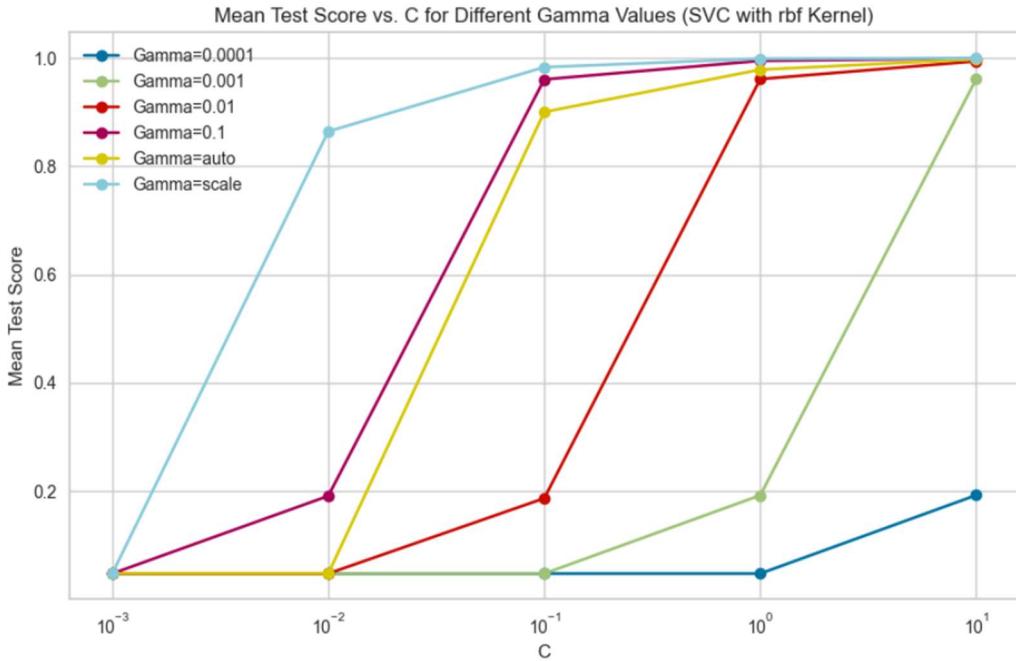


Abbildung 25: Test Score vs C für verschiedene Gamma-werte

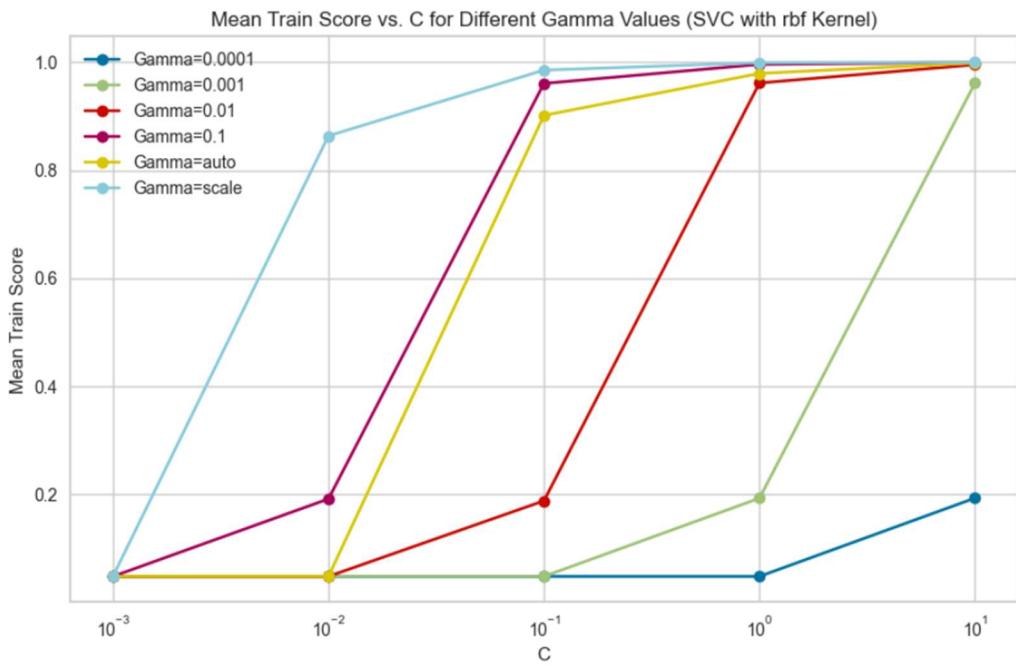


Abbildung 26: Train Score Vs. C für verschiedene Gamma-werte

- In Abbildung 25 wird die SVC mit RBF-Kernel für verschiedene Regulierungsparameter (C) und Gamma-Werte dargestellt. Es lässt sich feststellen, dass der Testscore bei verschiedenen Gamma-Werten schnell gegen den höchsten Wert konvergiert. Dies ist insbesondere bei einem Regulierungsparameter von 0,01 zu erkennen, da der Testscore von 0,0477 auf 0,839 erhöht wurde. Gleiches gilt für den Trainscore (Siehe Abbildung 25 & 26).

- Bezuglich des Regulierungsparameters (C): Je höher dieser ist, desto höher sind Training- und Test-Scores. Es ist außerdem zu beachten, dass bei kleinen Gamma-Werten der C-Wert erhöht werden muss, um höhere Genauigkeit zu erzielen (Siehe Abbildung 25 & 26)
- Aus den beiden Abbildungen ist ersichtlich, dass eine Gamma-Wert Einstellung von 'scale' oder '0,1' und ein C-Wert ab einem Wert von '0,1' die besten Ergebnisse zeigen.

Da der Gamma-Wert von 'scale' die beste Performance zeigte, wurde dieser auf 'scale' festgelegt. Um die linearen Kurven in Abbildung (25 & 26) schmäler und übersichtlicher zu gestalten, wurde die Anzahl der C-Werte um 43 erhöht.

In Abbildung Nr. 27 ist die lineare Kurve der Test- und Trainscore dargestellt. Es kann beobachtet werden, dass das Ausmaß der Übereinstimmung zwischen beiden Kurven besteht. Daher sind die geringfügigen Unterschiede zwischen den beiden Kurven nicht erkennbar.

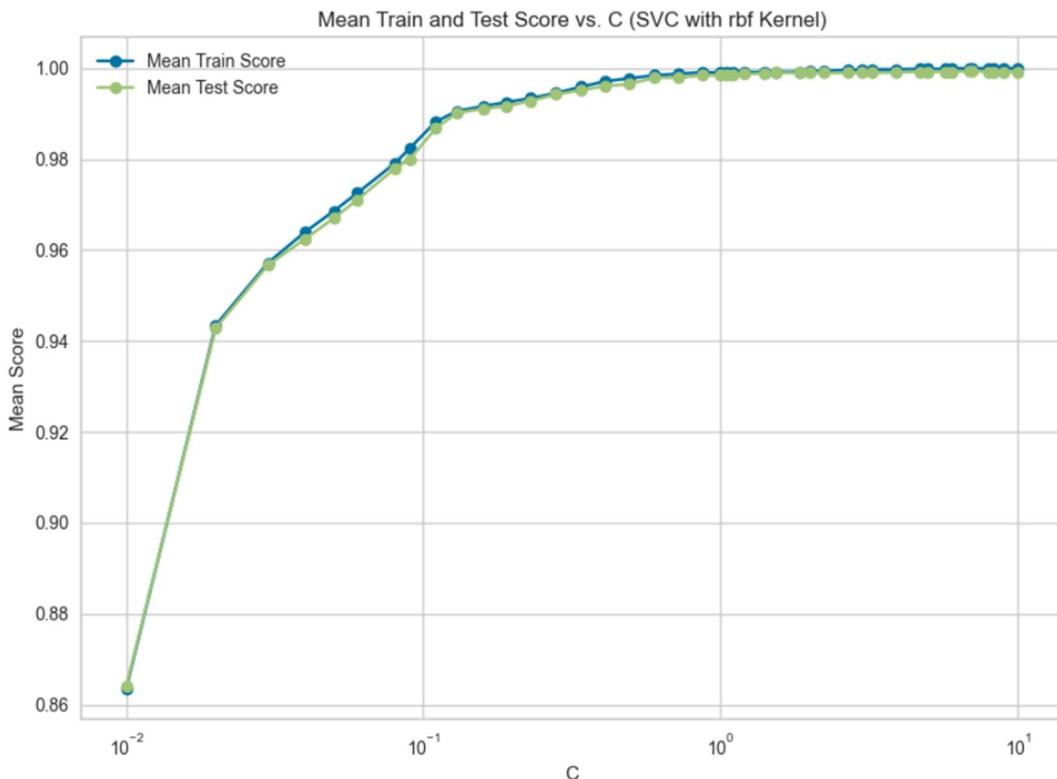


Abbildung 27: Mean Train und Test Score vs. C (SVC mit rbf Kernel)

In der folgenden Abbildung wird der Unterschied zwischen Trainings- und Testscores berechnet und dargestellt. Die Unterschiede sind nun erkennbar. Daher können wir feststellen, wo die minimalen Unterschiede liegen (Siehe Abbildung 28)

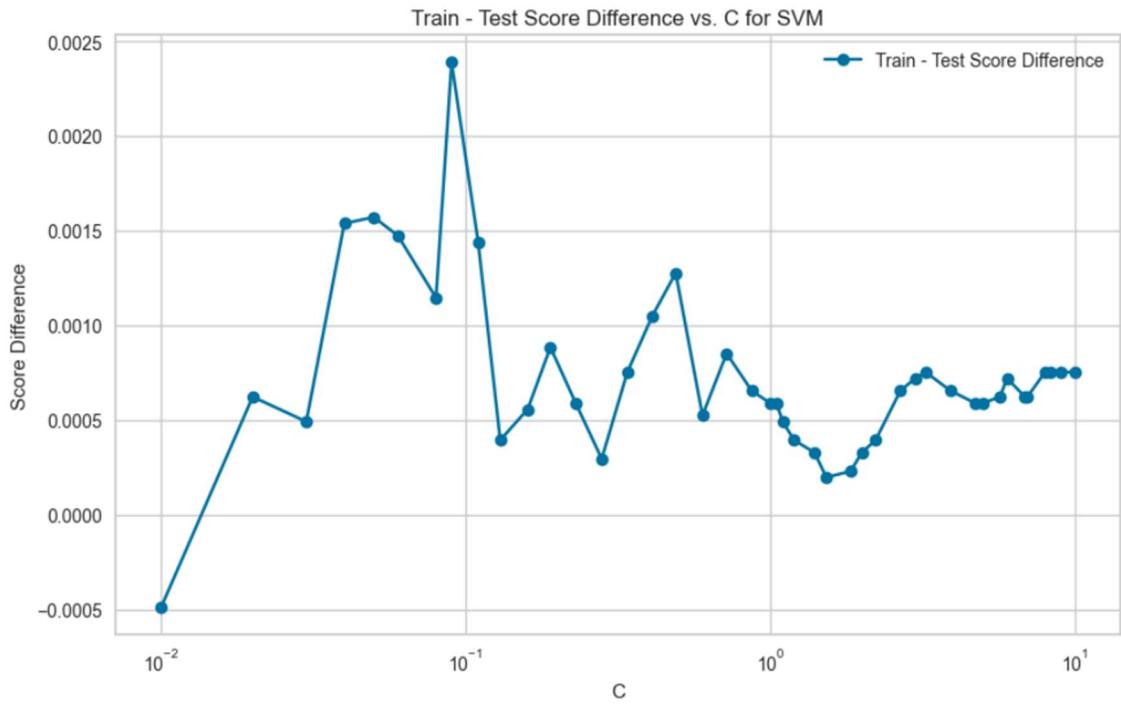


Abbildung 28: Train - Test Score Difference vs. C für SVM(SVC)

Die Tabelle wird aus dem Dataframe extrahiert, welcher für die Erstellung der Abbildung Nr. 27 verwendet wurde. In der Tabelle sind ausschließlich minimale Unterschiede sichtbar, bei Test Scores, die einen Wert von 0,95 oder höher aufweisen. Die Auflistung der Unterschiede erfolgt in aufsteigender Reihenfolge nach der Score Difference.

	param_C	mean_test_score	score_difference
25	1.53	0.998953	0.000196
26	1.84	0.998953	0.000229
13	0.28	0.994242	0.000295
27	2.0	0.998953	0.000327
24	1.4	0.998822	0.000327

Abbildung 29: Auflistung der geringsten Unterschiede zwischen Training- und Test-Score.

Wir haben die Top fünf Modelle, die in Abbildung Nr. 29 gezeigt wurden, getestet und sind zu dem Ergebnis gekommen, dass das Modell mit den Hyperparametern 'C' von '2.0' die beste Leistung erzielt hat und stabiler war als die anderen.

```
Model Nr. 25 mit C = 1.53
Cross-Validation Accuracy : 99.89528795811518 %
Accuracy : 100.0 %
```

```
Model Nr. 26 mit C = 1.84
Cross-Validation Accuracy : 99.90837696335078 %
Accuracy : 99.96947496947497 %
```

```
Model Nr. 13 mit C = 0.28
Cross-Validation Accuracy : 99.39794856166472 %
Accuracy : 99.51159951159951 %
```

```
Model Nr. 27 mit C = 2.0
Cross-Validation Accuracy : 99.90837696335078 %
Accuracy : 99.96947496947497 %
```

```
Model Nr. 24 mit C = 1.4
Cross-Validation Accuracy : 99.89528795811518 %
Accuracy : 100.0 %
```

**Bestes SVC Modell (C = 2.0 und Cross-Validation Accuracy: 0.9990837696335078)**

Abbildung 30: Bestes SVC Modell mit (C = 2.0 und gamma=scale und kernel='rbf')

Dennoch wurde die Genauigkeit mithilfe der Kreuzvalidierung berechnet, um festzustellen, welches der Modelle die höchste Validierungsgenauigkeit aufweist. Es stellte sich heraus, dass das Modell mit C='2' die beste Leistung erbrachte. Es ist auch zu beachten, dass die C-Werte nah beieinander liegen, wodurch die Unterscheidung zwischen der Validierungsgenauigkeit sehr gering ist. Wir kommen zu dem Schluss, dass die Wahl von 'C' zwischen [1.4, 2.0] ziemlich gute Ergebnisse liefert. Die Konfusionsmatrix zeigt, dass die Erkennung gut ist (Siehe Abbildung 30 & 31)

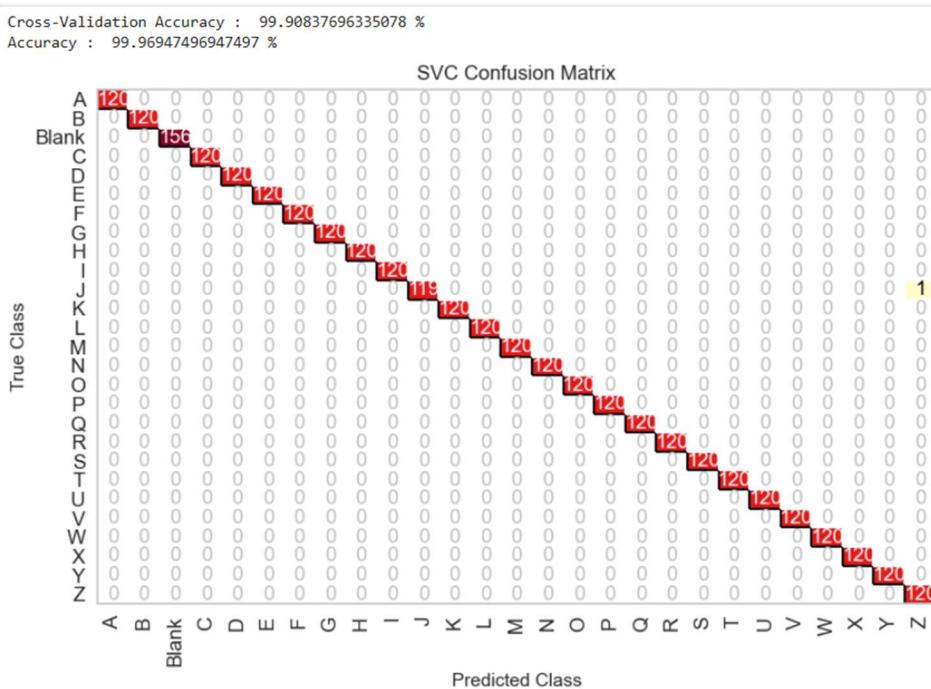


Abbildung 31: Kofusionsmatrix für die beste SVC Model(SVC mit C='2')

## SGDClassifier (Stochastic Gradient Descent)

Nach dem Testen des linearen SVC konnten keine guten Ergebnisse erzielt werden. Aus diesem Grund testen wir nun den Algorithmus Stochastic Gradient Descent, da er ebenfalls ein lineares Klassifikationsverfahren unter Verwendung der gleichen Regularisierung (L2) darstellt. Die Unterschiede liegen hauptsächlich im Algorithmus und seiner Geschwindigkeit. Aus diesem Grund wird auf dem "scikit-learn algorithm cheatsheet" die Nutzung dieses Algorithmus für Datensätze mit mehr als 100.000 Samples empfohlen. Daher werden wir den Algorithmus vergleichen, um festzustellen, welcher lineare Lernalgorithmus für unseren Fall besser geeignet ist.

Die wichtigsten Hyperparameter für diesen Algorithmus sind:[10]

- 1- max\_iter: Die maximale Anzahl der Iterationen über die Trainingsdaten.
  - 2- Loss: Die zu verwendende Verlustfunktion.
  - 3- Alpha: Regularisierungsparameter, Je höher der Wert, desto stärker ist die Regularisierung.
  - 4- **Penalty** : Art der Regularisierung
- Zunächst haben wir die Genauigkeit des Tests(Test Score) mithilfe der Grid-Suche mit verschiedenen Alpha-Werten und Iterationen (max\_iter) dargestellt und dies in dieser Abbildung visualisiert. (Siehe Abbildung 32)

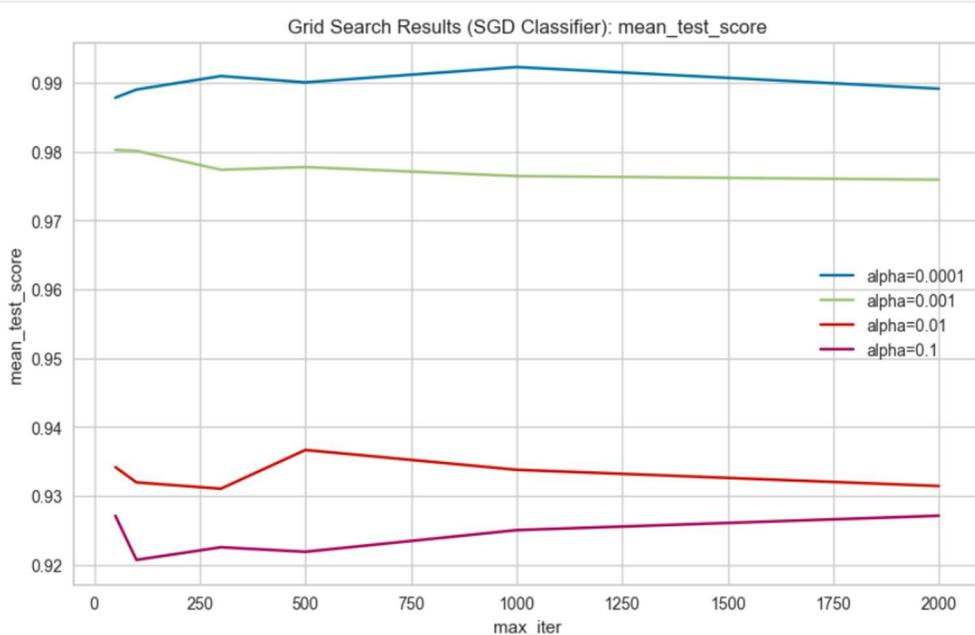


Abbildung 32: Grid Search Ergebnisse (SGD Classifier): mean\_test\_score

- Wir haben festgestellt, dass die Alpha-Werte einen großen Einfluss auf die Testergebnisse haben. Die Testergebnisse sind bei einer Iteration von 500 von etwa 0.925 auf etwa 0,98 gestiegen. Das bedeutet, dass je niedriger der Alpha-Wert ist, desto höher der Testscore ist.

- Die Anzahl der Iterationen spielt keine große Rolle, da sich die Test Score ab einem Wert von 1500 Iterationen kaum verändert hat. Das bedeutet, dass ab einer bestimmten Anzahl an Iterationen das Ergebnis nahezu gleich sein wird.

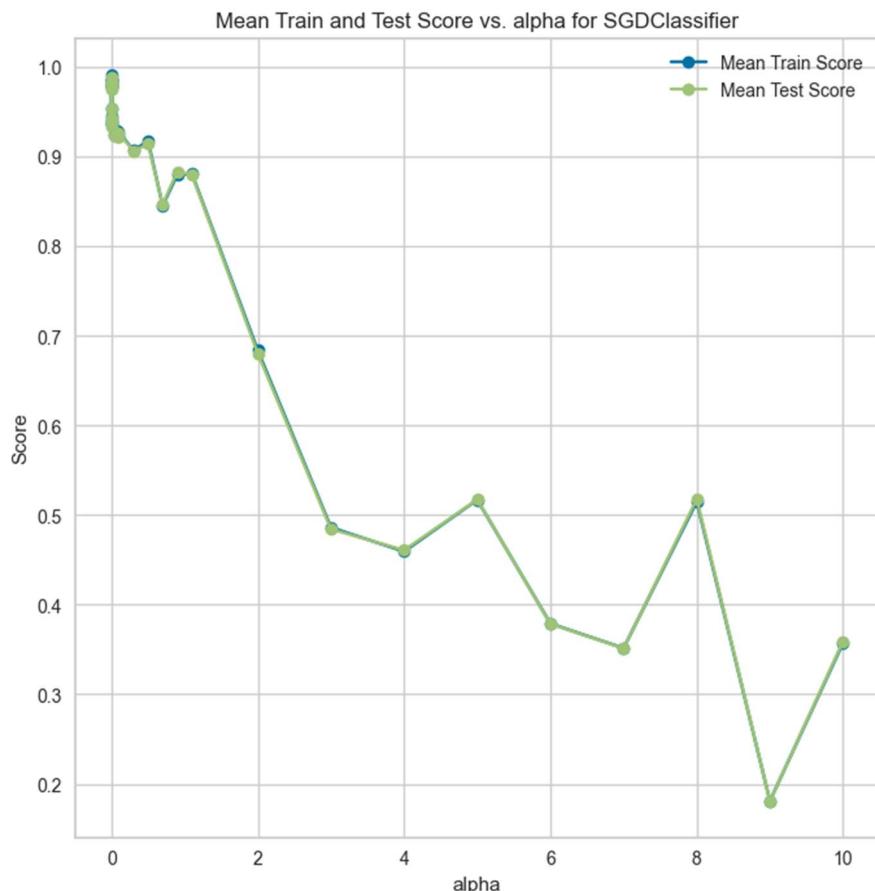


Abbildung 33: Mean Train und Test Score vs. alpha für SGDClassifier

In Abbildung Nr. 33 ist die lineare Kurve der Test- und Trainscore in Abhängigkeit von Alpha dargestellt. Es kann beobachtet werden, dass das Ausmaß der Übereinstimmung zwischen beiden Kurven besteht. Daher sind die geringfügigen Unterschiede zwischen den beiden Kurven nicht erkennbar.

In der folgenden Abbildung wird der Unterschied zwischen Trainings- und Testscores berechnet und dargestellt. Die Unterschiede sind nun erkennbar. Daher können wir feststellen, wo die minimalen Unterschiede liegen (Siehe Abbildung 34).

In der Abbildung kann man feststellen, dass das Modell für höhere Alpha-Werte ein "Overfitting" aufweist, da der Unterschied ( $(\text{Trainscore} - \text{Testscore}) < 0$ ) kleiner als null ist. Für diesen Algorithmus wird zur Reduzierung des Overfitting der Alpha-Wert verkleinert, um genauere Werte zu erzielen. Ab einem Alpha von 0,5 sehen wir, dass die Unterschiede positiv geworden sind. (Siehe Abbildung 34).

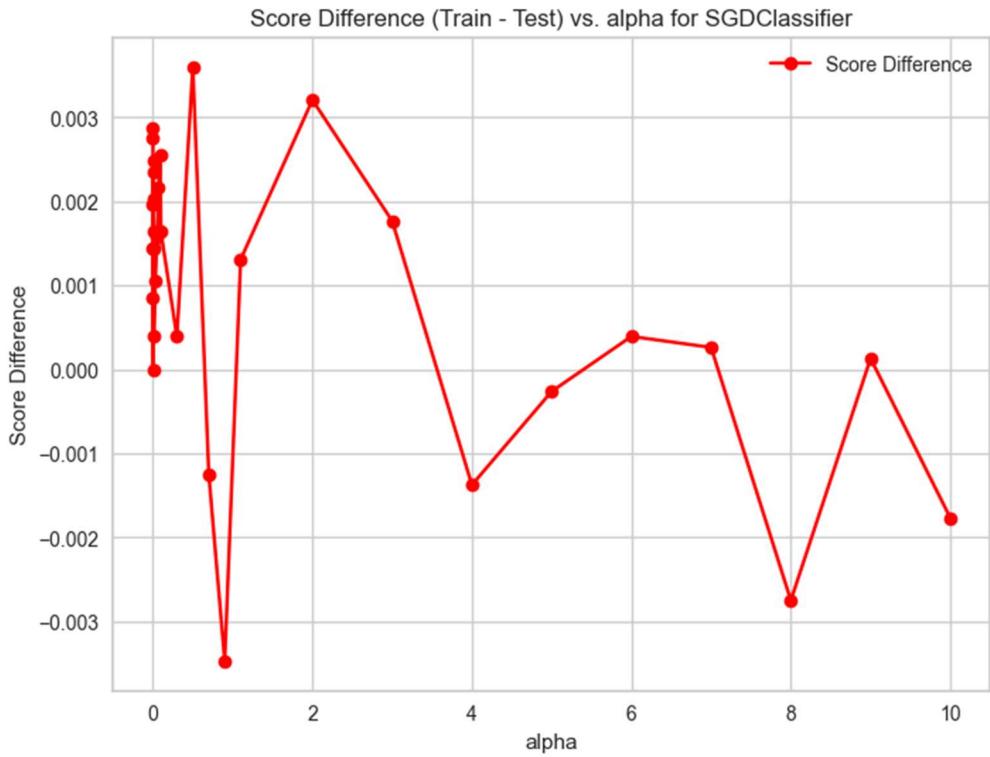


Abbildung 34: Score Difference(Train- Test) vs. Alpha für SGDClassifier

Jetzt muss die fünftkleinste Differenz zwischen Trainings- und Testergebnissen aus der Abbildung 34 gefiltert werden. Es wird analysiert, bei welchen Alpha-Werten dies der Fall ist (Siehe Abbildung 35).

param_alpha	mean_test_score	score_difference
7	0.953409	0.000000
4	0.982725	0.000851
2	0.985735	0.001440
5	0.980238	0.001440
6	0.976312	0.001636

Abbildung 35: Auflistung der geringsten Unterschiede zwischen Training- und Test-Score(SGDClassifier)

Anschließend ist die Kreuzvalidierungsgenauigkeit für alle über 5 Alpha-Werte zu berechnen, sowie die dahingehörige Testgenauigkeit.

```
Model Nr. 7 mit Alpha = 0.003
Cross-Validation Accuracy : 95.19687439006435 %
Accuracy : 95.2991452991453 %
```

```
Model Nr. 4 mit Alpha = 0.0007
Cross-Validation Accuracy : 98.16771903752579 %
Accuracy : 98.19902319902319 %
```

```
Model Nr. 2 mit Alpha = 0.0004
Cross-Validation Accuracy : 98.75663010762261 %
Accuracy : 98.77899877899878 %
```

```
Model Nr. 5 mit Alpha = 0.0009
Cross-Validation Accuracy : 97.8405195881372 %
Accuracy : 97.92429792429792 %
```

```
Model Nr. 6 mit Alpha = 0.001
Cross-Validation Accuracy : 97.65720503083493 %
Accuracy : 98.01587301587301 %
```

Bestes SGDClassifier Modell (Alpha = 0.0004 und Cross-Validation Accuracy: 0.9875663010762261)

Abbildung 36: Bestes SGD\_Classifier Modell

Da das bessere Modell eine Kreuzvalidierungsgenauigkeit von 98,7566% aufweist, ist der Alpha-Wert von 0,0004. Um die Erkennungsschwierigkeiten für jede Klasse darzustellen, muss die Konfusionsmatrix dargestellt werden.

Im Diagramm 37 und 38 ist ersichtlich, dass das Modell Schwierigkeiten bei der Erkennung mehrerer Klassen aufweist. Dies zeigt sich besonders deutlich beim experimentellen Testen. Diese Ergebnisse sind im Klassifizierungsbericht für jeder Klasse in Zahlen dargestellt.

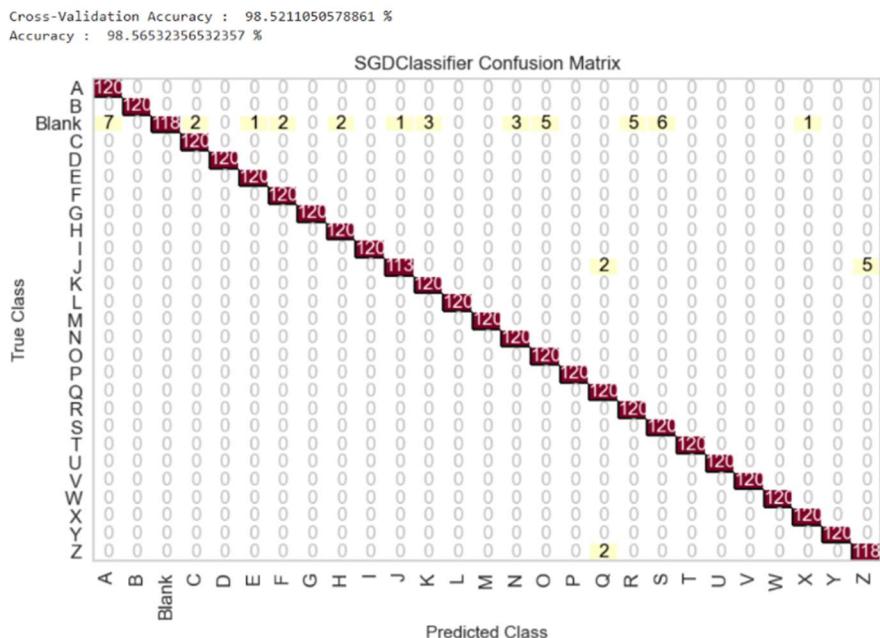


Abbildung 37: Konfusionsmatrix für die SGD\_Classifier(alpha=0.0004)

	precision	recall	f1-score	support
A	0.94	1.00	0.97	120
B	1.00	1.00	1.00	120
Blank	1.00	0.76	0.86	156
C	0.98	1.00	0.99	120
D	1.00	1.00	1.00	120
E	0.99	1.00	1.00	120
F	0.98	1.00	0.99	120
G	1.00	1.00	1.00	120
H	0.98	1.00	0.99	120
I	1.00	1.00	1.00	120
J	0.99	0.94	0.97	120
K	0.98	1.00	0.99	120
L	1.00	1.00	1.00	120
M	1.00	1.00	1.00	120
N	0.98	1.00	0.99	120
O	0.96	1.00	0.98	120
P	1.00	1.00	1.00	120
Q	0.97	1.00	0.98	120
R	0.96	1.00	0.98	120
S	0.95	1.00	0.98	120
T	1.00	1.00	1.00	120
U	1.00	1.00	1.00	120
V	1.00	1.00	1.00	120
W	1.00	1.00	1.00	120
X	0.99	1.00	1.00	120
Y	1.00	1.00	1.00	120
Z	0.96	0.98	0.97	120
accuracy			0.99	3276
macro avg	0.99	0.99	0.99	3276
weighted avg	0.99	0.99	0.98	3276

Abbildung 38: SGD\_Classifier mit Alpha =0.0004 (Classifier Report)

## Vergleich der Modelle

vergleich kriturum	LinearSVC	SVC	SGDClassifier	RandomForestClassifier
Lerngeschwindigkeit	Mäßige Geschwindigkeit	Langsam	Sehr schnell	Sehr langsam
geeignete Datensatz Größe	kleine bis mittlere Größe	jeder Größe	riesigen Datensätzen	mittelgroße bis große
Parameters	C :100	Kernel : rbf C : 2 Gamma :scale	Loss: hinge Alpha: 0.004	max_depth: 30 n_estimators: 100
Genauigkeit	99.756 %	99.969 %	98.779 %	100%
Kreuzvalidierung Genauigkeit	99.8 %	99.908%	98.757%	99.985 %
Experimentelle Leistung	Durchschnittliche Leistung	Hervorragende Leistung	Schlechte Leistung	Unterdurchschnittliche Leistung

Das gewählte Modell ist die nicht-lineare SVC mit den oben genannten Parametern.

## Die Erkennung für beide Hände erweitern.

- Um eine bessere Erkennung auch für die linke Hand zu ermöglichen, haben wir den Algorithmus verwendet, der die beste Leistung gezeigt hat, um ein neues Modell für die linke Hand zu erstellen. Dazu haben wir ein neues Datenset für die linke Hand genutzt, um die beste Leistung zu erreichen.
- Um die Erkennung der linken Hand zu verbessern, wurde ein Algorithmus verwendet, der das beste Ergebnis erzielte, um ein neues Modell für die linke Hand zu erstellen. Um eine bestmögliche Leistung zu erzielen, wurde ein neues Datenset für die linke Hand erstellt und daraufhin ein neues Modell trainiert.
- Um das passende Modell auszuwählen, muss das Programm vorher erkennen, ob es sich um die linke oder rechte Hand handelt. Dann wird die Vorhersage mit dem ausgewählten Modell erfolgen.

```

# Laden der trainierten Modelle für linke und rechte Hand
model_dict_left = pickle.load(open('./model_left.p', 'rb'))
model_left = model_dict_left['model']

model_dict_right = pickle.load(open('./model_right.p', 'rb'))
model_right = model_dict_right['model']

```

Abbildung 39: Aufruf von Modellen im GUI-Code

- Mit den für das Modell der rechten Hand verwendeten Parametern ( $C=2$  und  $\text{Gamma}=\text{'scale'}$ ) ergibt sich die folgende Konfusionsmatrix.

Cross-Validation Accuracy : 100.0 %  
Accuracy : 99.9388379204893 %

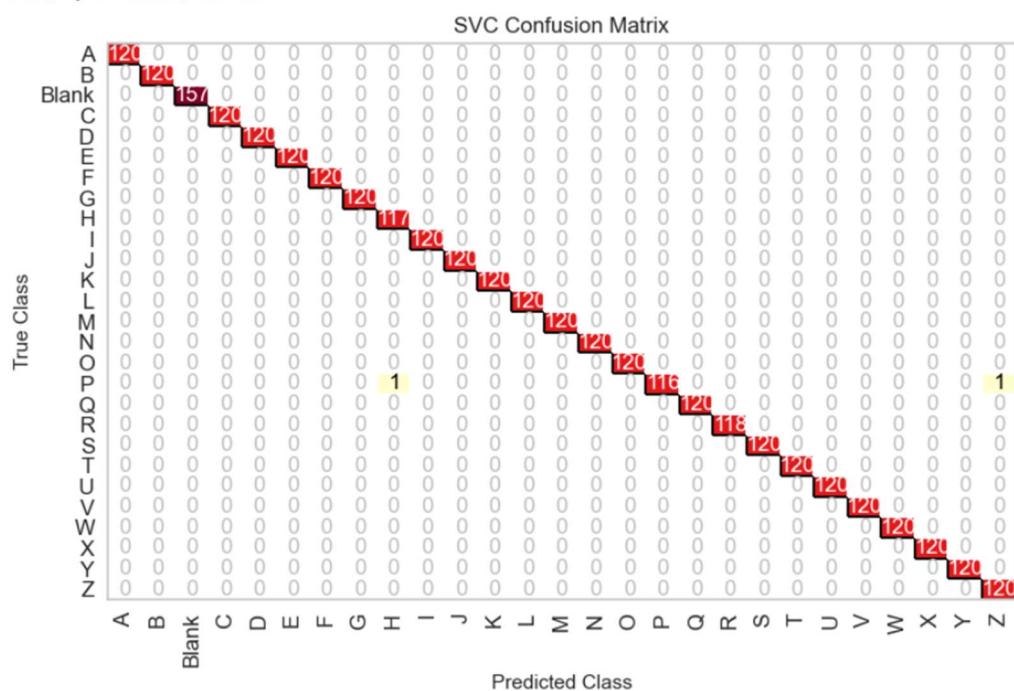


Abbildung 40: Linke Hand konfusionsmatrix(SVC mit  $C=2$  und  $\text{Gamme}=\text{'scale'}$ )

# GUI-Entwicklung

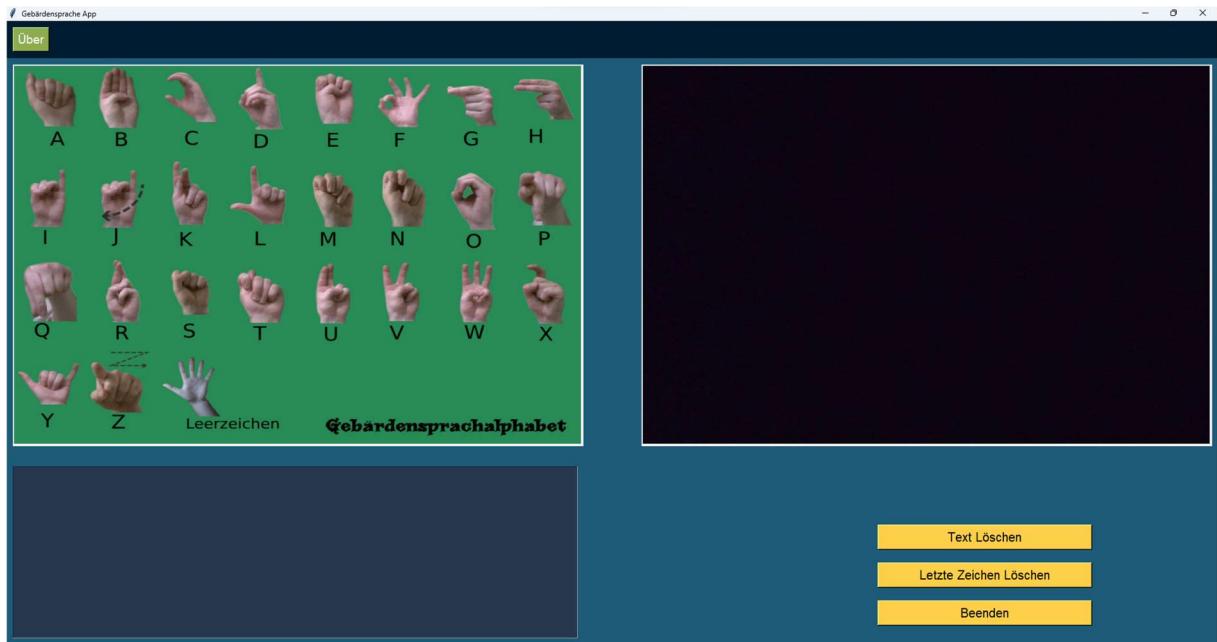


Abbildung 41: GUI\_Darstellung

Unser Hauptziel bei der Entwicklung dieser GUI (grafische Benutzeroberfläche) war es, eine benutzerfreundliche zu schaffen, die es den Benutzern ermöglicht, Gebärdensprachealphabete leicht zu erkennen. Die Anwendung bietet eine intuitive Benutzeroberfläche und nutzt fortschrittliche Technologien wie Computer Vision und KI (Künstliche Intelligenz), um die Gebärdensprachezeichen in Echtzeit zu interpretieren.

Sobald die Anwendung geöffnet wird, ist sie sofort einsatzbereit, um Gebärdensprachealphabete zu erkennen. Die GUI besteht aus vier Hauptkomponenten:

1. Video-Stream: Die Kamera des Geräts erfasst den Videostream und zeigt ihn in Echtzeit im GUI-Fenster an. Die Video-Frames werden kontinuierlich aktualisiert, um die erkannten Gebärdensprachezeichen darzustellen.
2. Bild des Gebärdensprachealphabets: Ein Hintergrundbild mit dem Gebärdensprachealphabet dient als visuelle Referenz für den Benutzer, um die erkannten Zeichen zu überprüfen.
3. Textfeld: Ein Textfeld in der GUI zeigt den erkannten Text an. Dies ermöglicht es dem Benutzer, die interpretierten Zeichen in schriftlicher Form zu sehen.
4. Schaltflächen (Buttons): Die Anwendung verfügt über Schaltflächen für verschiedene Aktionen, um die Benutzerfreundlichkeit zu erhöhen. Dazu gehören:
  - Beenden: Diese Schaltfläche beendet die Anwendung sauber und gibt die Kamera frei, bevor sie geschlossen wird.

- Löschen: Eine Schaltfläche zum Löschen des zuletzt erkannte Buchstabens oder des gesamten Texts.

Um sicherzustellen, dass nur stabile Vorhersagen in den erkannten Text aufgenommen werden, haben wir einen Mechanismus implementiert, der die Vorhersagen über mehrere aufeinanderfolgenden Frames hinweg überwacht. Sobald eine stabile Vorhersage erkannt wird, beispielsweise für 7 aufeinanderfolgende Frames, wird das vorhergesagte Zeichen automatisch zum Text hinzugefügt, und das Textfeld in der GUI wird aktualisiert.

```
if predicted_character==previous_predicted_character:  
    counter +=1  
    if counter>=7:  
        # Aktualisiere die globale Variable mit dem vorhergesagten Buchstaben  
        predicted_text += predicted_character  
        # Rufe die Methode auf, um das Textfeld zu aktualisieren  
        update_textbox()  
        counter=0  
    else:  
        previous_predicted_character=predicted_character  
        counter=0
```

Abbildung 42: predicted character Zähler

## Fazit

Das Entwickeln und Implementieren unserer Gebärdensprache Anwendung war ein besonderes Projekt, das uns die Möglichkeit gab, Technologie und Kreativität miteinander zu verbinden, um die Kommunikation für Gehörlose zu erleichtern und gleichzeitig das Interesse von Hörenden an der Gebärdensprache zu wecken. Die Anwendung erkennt Gebärdensprache Alphabet (ASL) und wandelt sie in Text um.

Abschließend lässt sich sagen, dass die Entwicklung und Implementierung unserer Gebärdensprache- Anwendung ein lehrreiches und erfolgreiches Projekt war. Wir haben wichtige Erkenntnisse aus den technologischen Herausforderungen gewonnen, die wir während des Entwicklungsprozesses bewältigt haben. Die Integration von Technologien wie MediaPipe und Konzepten wie Handlandmarken erforderte anfangs viel Zeit, um zu verstehen, wie sie funktionierten und wie man sie einsetzte, wurde aber schließlich mit Erfolg belohnt. Die Verwendung von Scikit-Learn für das maschinelle Lernen war besonders lehrreich, da wir verschiedene Lernalgorithmen zum Trainieren der Modelle aus dieser Bibliothek und auch andere Funktionen zur Datenvorbereitung verwendet haben. Diese Erfahrungen haben unser technisches Wissen erweitert und unsere Fähigkeiten gestärkt.

Ein besonderer Dank geht an unseren betreuenden **Herr Prof. Dr. Marcard** der uns während des gesamten Projekts unterstützt und inspiriert hat. Dieses Projekt zeigt, dass Technologie dazu beitragen kann, bestehende Barrieren zu überwinden und neue Möglichkeiten für die Kommunikation zu schaffen. Daruf sind wir stolz und sind gespannt auf die Zukunft der Technologie im Bereich der Gebärdenspracheerkennung.

# Englischer Fachbegriffe

**Cross-Validation** (Kreuzvalidierung): ist eine Vorgehensweise zur Bewertung der Leistung eines Algorithmus beim Machine Learning[11]

**Grid-Search** (Rastersuche oder Grid-Suche): Die klassischen Verfahren zur Suche nach optimalen Hyperparametern[12]

**Overfitting**: auch bekannt als Überanpassung, tritt auf, wenn das maschinelle Lernmodell genaue Vorhersagen für Trainingsdaten trifft, aber nicht für neue Daten.[13]

**Classification Report** : Eine Metrik zur Bewertung der Leistung des maschinellen Lernens. Sie wird verwendet, um die Precision, den Recall, den F1-Score und die Support des trainierten Klassifikationsmodells anzuzeigen.[14]

**Precision**(Präzision): ist ein Maß für den Anteil der vorhergesagten positiven Fälle an der Gesamtzahl der positiven Fälle.[15]

**F1-Score**: Der F1-Score ist ein gewichteter Mittelwert aus Präzision und Recall[15]

**Recall**(Erkennungsrate): Der Recall misst die Fähigkeit eines Modells, alle tatsächlich positiven Fälle einer Klasse korrekt zu erkennen.[15]

**True Positives** (TP): Der positive Wert wurde korrekt ermittelt.

**True Negatives** (TN): Der Negativwert wurde korrekt ermittelt.

**False Positives** (FP): Der positive Wert wurde falsch ermittelt.

**False Negatives** (FN): Der Negativwert wurde falsch ermittelt

**Pickle**: Pickle-Datei: ist ein Datenformat zum Speichern von Python-Objekten für die spätere Verwendung. [17]

**Confusion Matrix**(Konfusionsmatrix): liefert eine detailliertere Informationen. Sie zeigt, wie viele Vorhersagen wahr-positiv, wahr-negativ, falsch-positiv und falsch-negativ waren. [16]

# Quellen

- [1]: [Support Vector Machines in Machine Learning \(SVM\): 2023 Guide \(knowledgehut.com\)](#)
- [2]: [Choosing the right estimator — scikit-learn 1.3.0 documentation](#)
- [3]: [API Reference — scikit-learn 1.3.0 documentation](#)
- [4]: [Support Vector Machines in Machine Learning \(SVM\): 2023 Guide \(knowledgehut.com\)](#)
- [5]: [Random Forest vs Decision Tree: Key Differences - KDnuggets](#)
- [6]: [Random Forest Hyperparameter Tuning in Python | Machine learning \(analyticsvidhya.com\)](#)
- [7]: [sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.0 documentation](#)
- [8]: [Mastering Random Forests: A comprehensive guide | by Sandeep Ram | Towards Data Science](#)
- [9]: [SVM RBF Kernel Parameters: Python Examples - Analytics Yogi \(vitalflux.com\)](#)
- [10]: [sklearn.linear\\_model.SGDClassifier — scikit-learn 1.3.0 documentation](#)
- [11]: [3.1. Cross-validation: evaluating estimator performance — scikit-learn 1.3.0 documentation](#)
- [12]: [3.2. Tuning the hyper-parameters of an estimator — scikit-learn 1.3.0 documentation](#)
- [13]: [Was ist Überanpassung? – Erläuterung zu Überanpassung in Machine Learning – AWS \(amazon.com\)](#)
- [14]: [sklearn.metrics.classification\\_report — scikit-learn 1.3.0 documentation](#)
- [15]: [F1 Score in Machine Learning: Intro & Calculation \(v7labs.com\)](#)
- [16]: [Was ist die Konfusionsmatrix? | Data Basecamp](#)
- [17]: <https://docs.python.org/3/library/pickle.html>

## Weitere Quellen

- <https://developers.google.com/mediapipe/framework>
- <https://developers.google.com/mediapipe/solutions/guide>
- [https://developers.google.com/mediapipe/solutions/vision/hand\\_landmarker](https://developers.google.com/mediapipe/solutions/vision/hand_landmarker)
- <https://opencv.org/>
- <https://numpy.org/>
- <https://docs.python.org/3/library/tkinter.html>
- [Welcome to Python.org](#)
- [GitHub - computervisioneng/sign-language-detector-python](#)
- **Scikit-Learn-Dokumentation:** Für Informationen über Modelle für maschinelles Lernen und die im Code verwendeten Bibliotheken (z. B. RandomForestClassifier, SVM, GridSearchCV, SGDClassifier) <https://scikit-learn.org/stable/>
- **Matplotlib-Dokumentation:** Für Informationen zur Erstellung von Visualisierungen und Diagrammen mit Matplotlib
- **Pandas-Dokumentation:** IPandas-Dokumentation
- **YouTube-Tutorials:** "StatQuest mit Josh Starmer", "Data School", "codebasics", "FreeCodeCamp" and „Keith Galli“
- **Udemy-Tutorials:** " Complete Machine Learning & Data Science Bootcamp 2023"