Protokoll zu:

Übung 1

16. Dezember 2022

Gruppe 01:

Losbichler Maximilian (11815736)
Kittel Thomas (12002136)
Schutting Georg (01601919)

TU Wien

Department für Geodäsie und Geoinformation

Inhalt

Einführung

Die Aufgabe dieser Übung war es, anhand eines 64 km² abdeckenden Satellitenbildes der Sentinel-2-Reihe einen Klassifikator zu programmieren, der auf dem Bild Flächen mit Waldbedeckung von Flächen ohne Waldbedeckung unterscheiden kann. Das Untersuchungsgebiet durfte dabei selbst gewählt werden.

Die Angabe bestand aus zwei Teilen – jeweils mit Unterpunkten – als schrittweise Hilfestellung zur Erreichung des Ziels. Aus Konsistenzgründen und zur besseren Übersicht ist auch dieses Protokoll in einen Teil 1 und einen Teil 2 aufgeteilt.

Ersterer befasst sich mit der Erzeugung eines hochauflösenden RGB-Komposits sowie eines Falschfarbenbildes unter Verwendung des NIR[1]-Kanals aus den am Server gespeicherten Satellitenbildern. Außerdem sollten aus den Images noch die Wolken mithilfe medianweiser Mittelung über zeitlich aufeinanderfolgende Bilder herausgefiltert werden.

Teil 2 beschreibt schließlich die Vorgehensweise und Resultate bei der Anwendung und Programmierung verschiedener Klassifikatoren. Es mussten zunächst manuell Wald- und Nicht-Wald-Gebiete in dem in Teil 1 gewählten Bild ausgewiesen werden. Dann sollten mindestens 3 Klassifikatoren auf deren Treffsicherheit getestet werden – einmal am multispektralen Bild und einmal nur mit einem NDVI[2]-Kanal. Und schlussendlich sollte dann ein eindimensionaler Klassifikator selbst programmiert und auf den NDVI trainiert und getestet werden.

Teil 1

1.1 Lösungsansätze

Ziel des ersten Unterpunkts (1a) war es, ein hochauflösendes RGB-Komposit mit gutem Kontrast und 10m x 10m Auflösung zu erzeugen. Der Bildausschnitt zeigt ein Gebiet westlich von Wien mit dem Aufnahmedatum 20.7.2017. Um die Datenabfrage zu erleichtern setzten wir zunächst eine zeitliche Einschränkung für den Datacube, wir betrachteten also nur Bilder zwischen 1.6.2017 und 1.10.2017. Eine weitere Einschränkung war, dass nur Bilder aus jenem Ordner herangezogen wurden, der die Bilder von Wien und seiner Umgebung enthält (E052N016T1).

Zuerst legten wir die Spektralbänder B02(blau), B03(grün) und B04(rot) übereinander, um ein RGB-Komposit zu erstellen. Eine Verstärkung des Kontrasts erreichten wir mit der Anwendung der *auto_clip* Funktion zur Entfernung von Extremwerten. Es musste eine Kopie als Argument

[1] NIR = Nahinfrarot
[2] NDVI = Normalized Difference Vegetation Index

an *auto_clip* übergeben werden, da der Array überschrieben wurde. Die Funktion *normalize* interpolierte schließlich alle Werte von dem Bildarray auf den Bereich [0,1] ab.

Als nächstes (1b) erzeugten wir ein Falschfarbbild unter Verwendung des NIR-Kanals. Dabei wurde dasselbe Bild wie in der vorrangegangenen Aufgabe verwendet, allerdings interessierte uns jetzt nur der Kanal B08. Die Funktion *auto_clip* wurde wieder verwendet um den Kontrast im Bild zu erhöhen. Mit der Funktion *flatten* wurde die Dimension des Arrays verringert, um sie für die *hist* Funktion verwertbar zu machen.

Nun sollte das in 1a gewählte Bild noch Median-gemittelt werden, um die Wolken herauszufiltern (1c). Dazu wählten wir 5 zeitlich aufeinanderfolgende Bilder im gewählten Teilgebiet aus. Wir konnten jedoch nicht einfach 5 aufeinanderfolgende Tage nehmen, denn der Satellit überfliegt das gegebene Gebiet nicht jeden Tag, zudem streifen einige Überflüge das Untersuchungsgebiet nur und führen somit zu schwarzen Flecken auf den Bildern. Deshalb nahmen wir die Auswahl manuell vor, über die Durchsicht einiger Dutzend Bilder, mit folgenden Bildern als Ergebnis:

- 20.07.2017
- 30.07.2017
- 09.08.2017
- 29.09.2017
- 08.09.2017

Als nächstes importierten wir alle 5 ausgewählten Bilder in Python und speicherten sie in separaten Variablen. Für jedes der 9 Spektralbänder, die zur Analyse zur Verfügung standen, legten wir alle 5 Images übereinander und bildeten mithilfe von *numpy* den Median über jedes Pixel. Somit ergaben sich 9 Median-gefilterte Spektralaufnahmen, die nun wieder zu einem gemittelten Foto zusammengefügt werden konnten.

Für eine schöne Ausgabe sind nur die RGB-Bänder B02, B03 und B04 wichtig, deshalb fügten wir die Median-gefilterten Versionen dieser Bänder zusammen und bearbeiteten das resultierende Bild zwecks Ansehnlichkeit noch mithilfe der Funktionen zur Kontrasterhöhung, wie bereits beschrieben.

Schlussendlich musste die Grafik noch exportiert werden, um in einem späteren Schritt die Wald- und Nicht-Wald-Flächen darauf definieren zu können. Hierbei machten wir anfangs den Fehler, dass wir das Bild einfach als jpeg exportierten, nicht ahnend, dass dadurch wertvolle geografische Information verloren ging, die später Wiederimportieren zu großen Schwierigkeiten führen sollte.

Es sollte einige Korrespondenz und eine Feedbackstunde brauchen, bis wir herausfanden, dass das erstellte Image mithilfe der Codechunks als geotiff exportiert werden musste, um die im Bild hinterlegten Koordinaten weiterhin zu speichern – vor allem im richtigen Koordinatensystem.

1.2 Ergebnisse

Abbildung 1 zeigt die Histogramme der 3 Farbkanäle des originalen und des bearbeiteten RGB-Komposits. Ein Histogramm zeigt die statistische Häufigkeit der Tonwerte des Bildes an. Bei den bearbeiteten RGB-Kanälen sind die Maximalwerte geringer als bei den originalen RGB-Kanälen. Außerdem sieht man in Abbildung 2, dass das bearbeitete RGB-Komposit deutlich heller ist und einen besseren Kontrast aufweist, als das originale Bild.

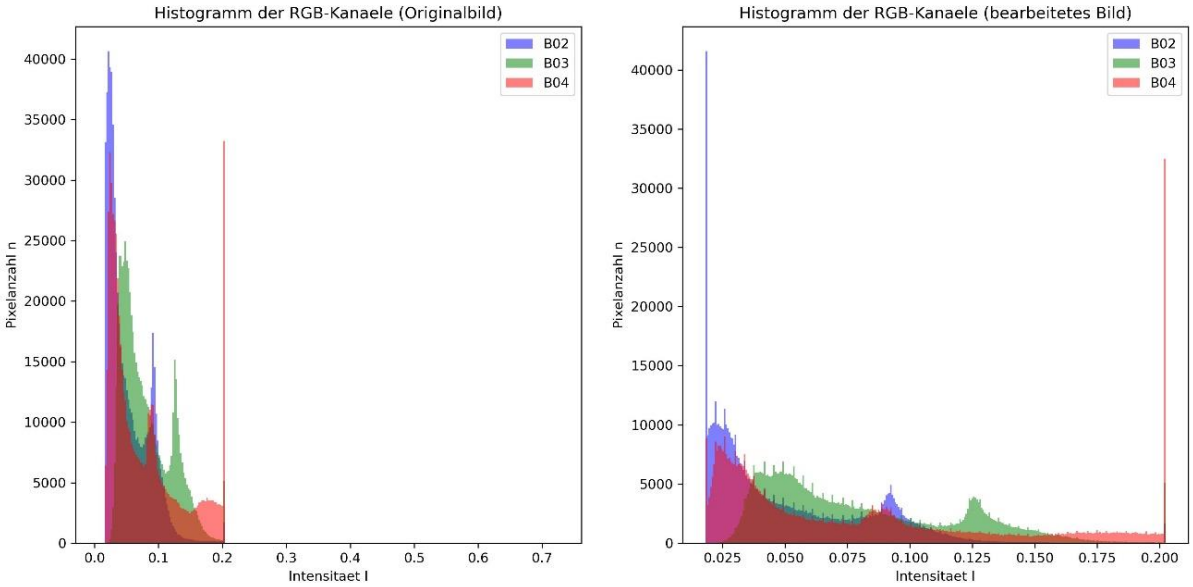

Abbildung 1: Vergleich der Histogramme

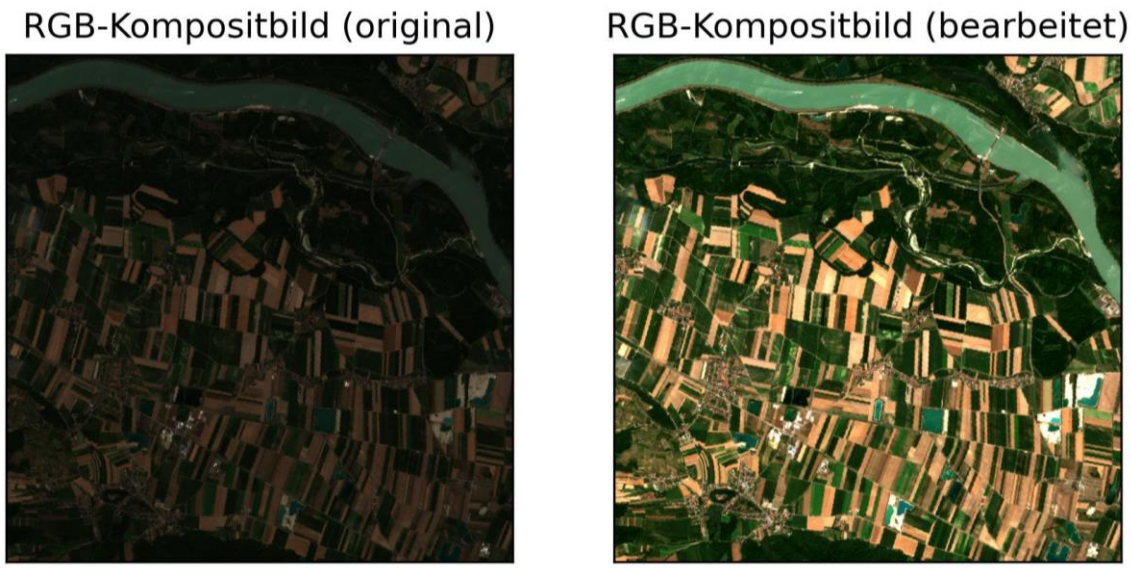

Abbildung 2: Vergleich der RGB-Kompositbilder

In Abbildung 3 sind die Histogramme des originalen und des bearbeiteten NIR-Kanals zu sehen. Wieder trat nach Anwendung der Funktion *auto_clip* eine Verringerung der Maximalwerte auf. Analog zu 1a zeigt Abbildung 4 anschließend den Vergleich zwischen originalem und bearbeitetem Falschfarbbild unter Verwendung der colormap *seismic*.

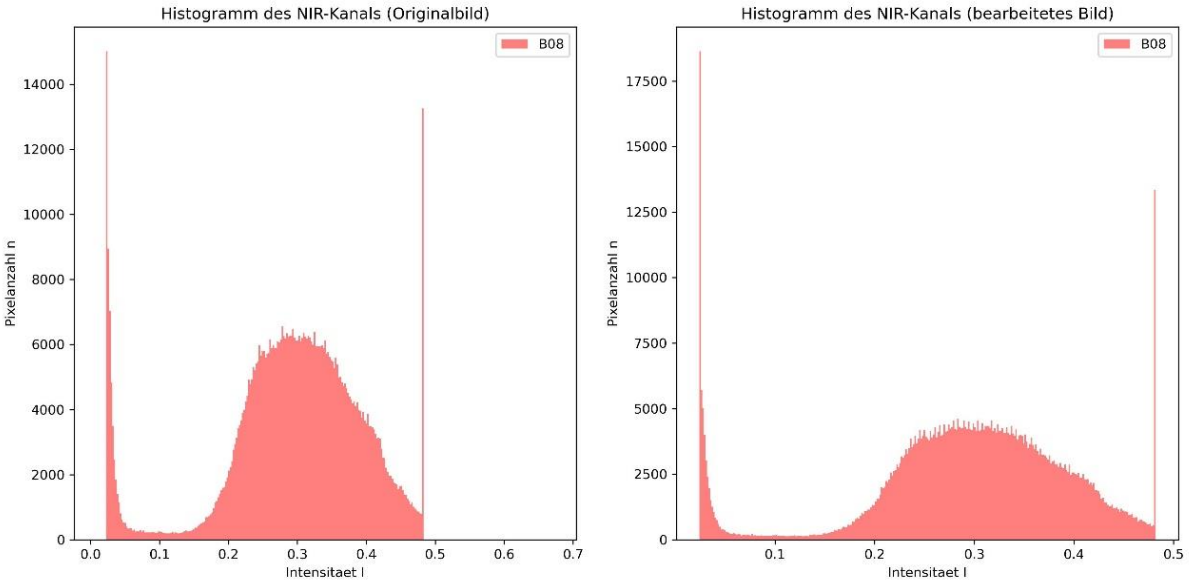

Abbildung 3: Vergleich der Histogramme

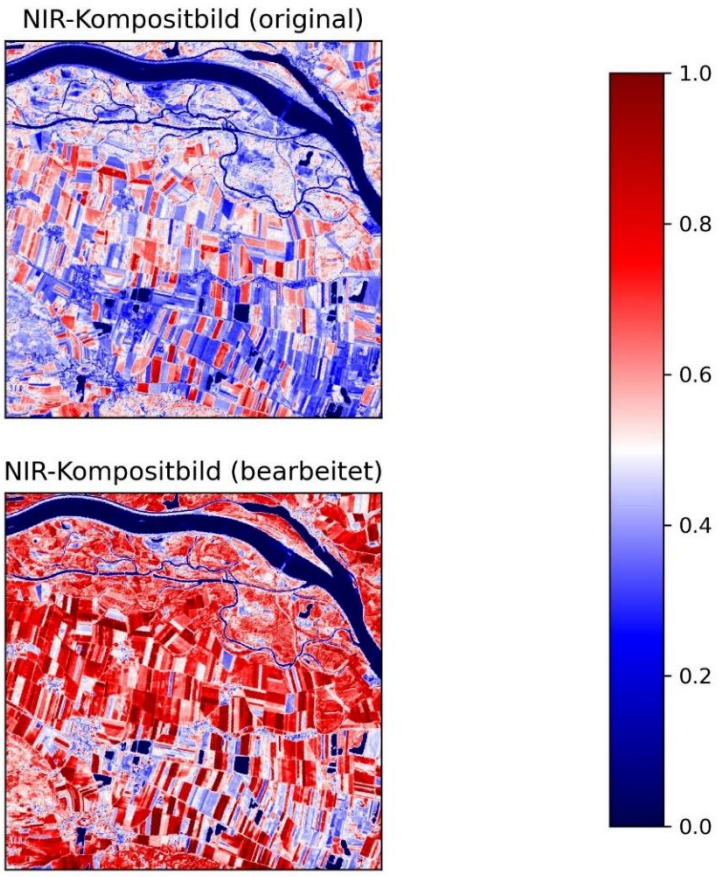

Abbildung 4: Vergleich der NIR-Kompositbilder

Das Resultat der Median-Filterung (siehe Abbildung 5) zeigt, dass alle (wenn auch spärlich vorhandenen) Wolken am linken Bildrand verschwunden sind. Ein amüsanter Nebeneffekt des Filters war zudem, dass auch die in Bewegung befindlichen Schiffe auf der Donau entfernt wurden.

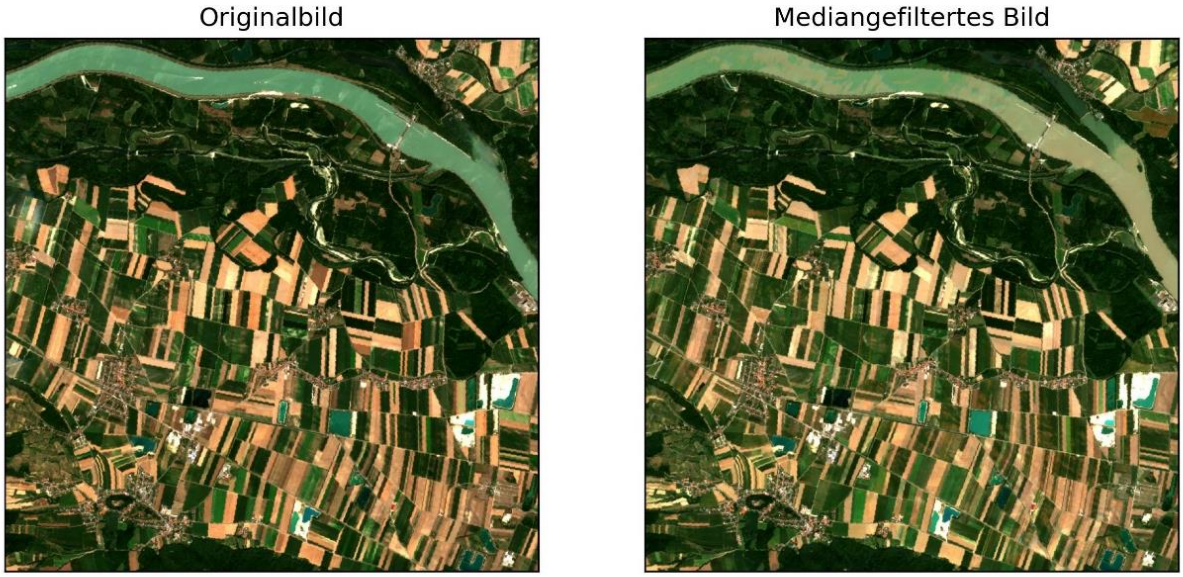

Abbildung 5: Vergleich Originalbild mit mediangefiltertem Bild

Teil 2

2.1 Lösungsansätze

Im Teil 2 ging es nun um den Aufbau eines Klassifikators, der Waldflächen auf den Satellitenbildern erkennen können sollte.

Zu Beginn (2a) war es notwendig, dass wir selbst händisch einige Flächen als Wald- bzw. Nicht-Wald auswiesen, um dem Mustererkennungsalgorithmus später eine Datenbasis übergeben zu können. Das machten wir in QGIS, indem wir das in Teil 1 exportierte Median-gefilterte Bild als Hintergrund verwendeten und darauf in einem zweiten und dritten Layer Polygone zeichneten, die Flächen mit Wald und Flächen mit anderen Landnutzungskategorien bedeckten, wie Abbildung 6 zeigt.

Für Nicht-Wald wurden die vier Landnutzungsklassen Acker, Wasser, Straße/Schiene und Siedlung zusammengefasst. Anteilsmäßig am größten waren Äcker in unserem Bild vertreten, weswegen die Acker-Polygone am größten von allen Nicht-Wald-Polygonen gezeichnet wurden. Das Gegenteil trifft auf die Klasse Straße/Schiene zu. Das diente dazu, dem Algorithmus klarzumachen, von welcher Landnutzungsklasse er am meisten erwarten durfte und welche eher selten vorkommt.

Wald-Polygone zogen wir sowohl im Auenwald direkt neben der Donau als auch im weniger wassernahen Wald im unteren Bereich des Bildes ein, um möglichst alle verschiedenen Waldformen im Bild gut zu repräsentieren.

Nun musste für den Klassifikator aus den Polygonen eine geeignete gleich große Trainings- und Testmenge definiert werden, welche anschließend als Trainingsgrundlage für 3 verschiedene, in der Angabe vorgegebene, und einen weiteren, in 2d selbst zu implementierenden Klassifikator für eindimensionale Merkmale, verwendet bzw. weiterverarbeitet werden sollte (2b).

Hierzu lasen wir anfangs die beiden zuvor als geojson-Datei aus QGIS exportierten Polygon-Daten in Python ein und verschnitten diese in jeweils separaten Schritten mit Hilfe der Code-Chunks mit dem ausgewählten Bild. Außerdem entfernten wir die leeren Polygone aus den beiden Dictionaries und berechneten für die Filterung der „cloudy“ Pixel die jeweiligen zeitweisen Mittelwerte über jedes der 9 Bänder.

Anschließend entfernten wir für die Wald- und Nicht-Wald-Polygone die für den Klassifikator irrelevanten räumlichen Koordinaten und verketteten die übrigen Reihen zu jeweils einem Array, ohne NaN-Werte.

Um die Input- und Outputvariablen (X und y) für den Trainingsklassifikator zu erhalten, erstellten wir für den Wald-Array einen mit 1ern gefüllten Vektor derselben Länge und für den Nicht-Wald-Array einen mit 0ern gefüllten Vektor, ebenfalls mit der Länge des Arrays. Für die Input-Variable X wurden die Wald- und Nicht-Wald-Arrays und für die Output-Variable y die 1er- und 0er-Vektoren zusammengefügt.

Mit der Funktion *train_test_split* aus der scikit-learn-Bibliothek teilten wir die beiden Variablen mithilfe eines Zufallsgenerators im Verhältnis 0,5 in eine Trainings- und eine Testmenge auf, welche wir jetzt für die drei in scikit-learn als Funktionen definierten Klassifikatoren *Gaussian Naive Bayes*, *SVM* und *Random Forest* verwenden konnten.

Zum Trainieren eines jeden Klassifikators wurden die Trainingsmengen von X und y genutzt und zum Testen der trainierten Klassifikator-Funktionen fütterten wir diese mit dem Testset von X und verglichen das Output-y mit dem vorher definierten Testset von y.

Diesen Vergleich konnten wir mithilfe der Funktionen *classification_report* und *confusion_matrix* anstellen, die jeweils die User’s Accuracy, die Producer’s Accuracy und die Overall Accuracy des Klassifikationsprozesses ausgaben.

Die nächste Aufgabe war (2c), den NDVI für unser Teilbild zu berechnen und den zuvor implementierten *Gaussian Naive Bayes* Klassifikator (NBK) auf diesen Wert zu trainieren und zu testen. Für die Berechnung des NDVIs definierten wir, wie in den Code-Chunks vorgegeben, die Funktion:

$$NDVI = \frac{NIR - Red^3}{NIR + Red}$$

Formel 1: NDVI-Formel

Anschließend setzten wir die bei 2b ermittelte transponierte Input-Variable X mit den jeweiligen Bändern (B08 – NIR-Kanal und B04 – Rot-Kanal) in Formel 1 ein und erstellten wieder mit der Funktion *train_test_split* eine zufällige Trainings- und eine Testmenge, mit welchen wir dann den NBK trainierten und auf seine Treffsicherheit testeten.

Bei den letzten beiden Aufgaben des zweiten Teils war ein NBK für eindimensionale Merkmale selbst zu implementieren, auf welchen dann der errechnete NDVI des Trainingsgebietes trainiert werden musste. Anschließend musste man diese Ergebnisse interpretieren und die Leistungsfähigkeit des selbstprogrammierten Klassifikators vergleichen.

Hierfür kombinierten wir die in den Code-Chunks vorgegebene Klasse „MyGNBClassifier“ mit den jeweiligen Formeln von Naive Bayes, welche wir dem Dokument *naiveBayes.pdf* von TUWEL entnommen haben. Danach trainierten wir unseren Klassifikator mit den beiden Variablen aus 2c aus und ließen das Test-Ergebnis über die Funktion *classification_report* auswerfen, auf das wir im Ergebnisabschnitt näher eingehen werden.

2.2 Ergebnisse

Abbildung 6 zeigt links die gezeichneten Polygone vor dem Median-gefilterten Bild als Hintergrund. Die Wald-Polygone sind hellbraun eingefärbt, während die Nicht-Wald-Polygone hellgrün sind. Rechts ist die Polygonanordnung in Wald- und Nicht-Wald-Polygone zu sehen. Man erkennt die großzügigen Acker-Polygone sowie die kleiner gehaltenen Siedlungs-, Verkehrswege- und Wasser-Polygone.

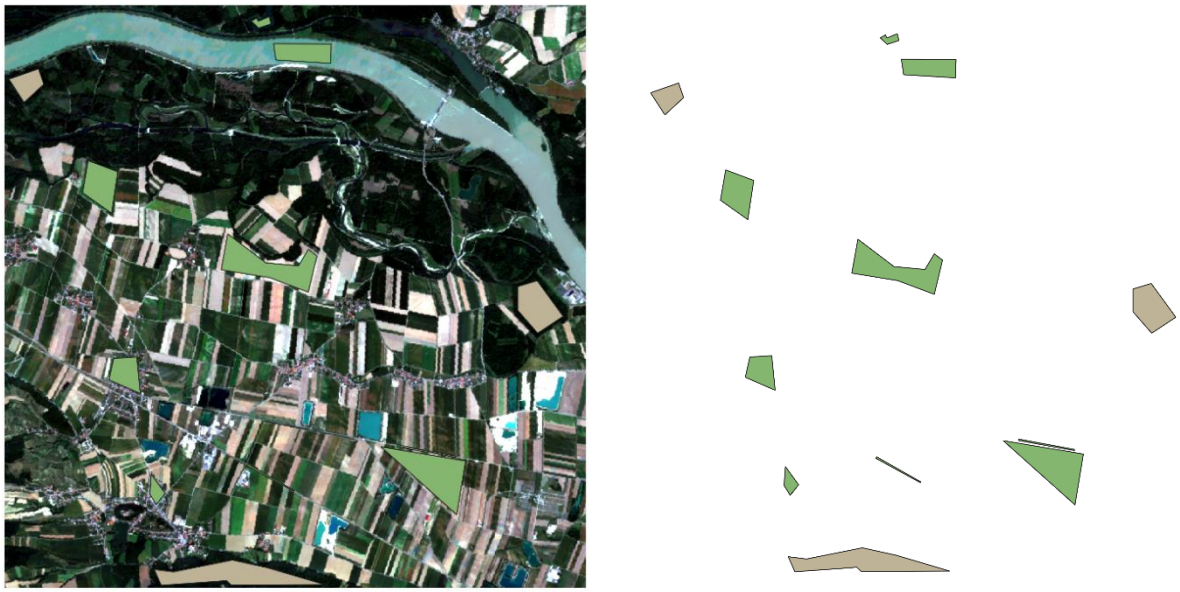

Abbildung 6: Polygone über mediangefiltertem Bild

[3] Red = rotes Spektralband

Die Abbildungen 7, 8 und 9 veranschaulichen die Ergebnisse der Test-Runs mit den zuvor trainierten Klassifikatoren *SVM, Random Forest* und *Gauss Naive Bayes.* In Form von Konfusionsmatrizen lassen sich jeweils die User‘s Accuracy (UA), Producer’s Accuracy (PA) und Overall Accuracy (OA) ablesen.

Gauss Naive Bayes		Referenz			
		Wald	Nicht-Wald	Summe Klassierung	Users Accuracy
Klassierung	Wald	4222	832	5054	0,835
	Nicht-Wald	79	6914	6993	0,989
	Summe Referenz	4301	7746	12047	
	Producers Accuracy	0,982	0,893		
Overall Accuracy			0,924		

Abbildung 7: Konfusionsmatrix des GNB Klassifikators

Wie man an Abbildung 7 erkennen kann, ist beim *NBK* die PA des Nicht-Waldes und die UA des Waldes unter 90 %, dadurch befindet sich die Overall Accuracy “nur” bei 92,4 %. Verglichen mit den Ergebnissen der beiden nachfolgenden Klassifikatoren ist der NBK somit der ungenaueste Klassifikator. Dies spiegelt sich auch in den Plots aus Abbildung 10 der Klassifikatoren wider.

SVM		Referenz			
		Wald	Nicht-Wald	Summe Klassierung	Users Accuracy
Klassierung	Wald	4285	51	4336	0,988
	Nicht-Wald	16	7695	7711	0,998
	Summe Referenz	4301	7746	12047	
	Producers Accuracy	0,996	0,993		
Overall Accuracy			0,994		

Abbildung 8: Konfusionsmatrix des SVM Klassifikators

Die Konfusionsmatrix des *SVM* Klassifikators in Abbildung 8 zeigt zumindest bei den Ergebnissen eine etwas höhere Genauigkeit bei UA, PA und OA als der vorherige Gauss Naive Bayes Klassifikator. Da die Genauigkeiten nahe 100 % liegen, kann man davon ausgehen, dass die meisten Teile des Waldes und Nicht-Waldes richtig ausgegeben werden.

Random Forest		Referenz			
		Wald	Nicht-Wald	Summe Klassierung	Users Accuracy
Klassierung	Wald	4276	21	4297	0,995
	Nicht-Wald	25	7725	7750	0,997
	Summe Referenz	4301	7746	12047	
	Producers Accuracy	0,994	0,997		
Overall Accuracy			0,996		

Abbildung 9: Konfusionsmatrix des Random Forest Klassifikators

Abbildung 9 zeigt beim *Random Forest* Klassifikator eine ähnlich hohe, wenn nicht sogar minimal bessere, Genauigkeit als beim *SVM* Klassifikator. Die Overall Accuracy hat sich hier um ca. 0,02 % verbessert.

Ein weiterer Vergleich der Klassifikatoren wird in Abbildung 10 mittels Subplots veranschaulicht. In gelber Farbe sind die als Wald erkannten und in schwarz/lila die als Nicht-Wald erkannten Gebiete dargestellt.

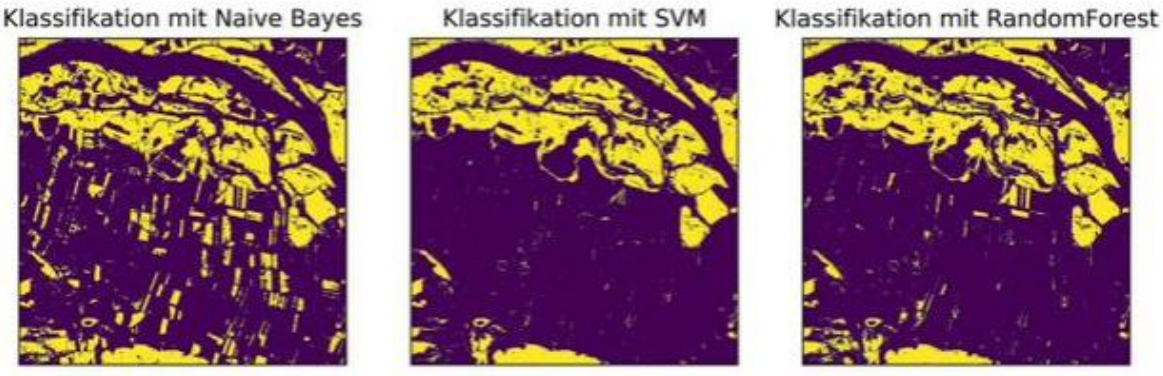

Abbildung 10: Vergleich der Klassifikatoren

In den 3 Plots erkennt man jetzt, dass sich die auf den ersten Blick minimal erscheinenden Abweichungen in den Konfusionsmatrizen doch sehr signifikant voneinander unterscheiden. So werden beim *NBK* noch einige Felder und Wiesen als Wald erkannt, die sowohl bei *SVM* als auch bei *Random Forest* großteils richtig herausgefiltert werden.

Interessanterweise fällt beim genaueren Vergleich zwischen *SVM* und *Random Forest* auf, dass ersterer zwar alle Felder richtig herausfiltert, jedoch auch nicht alle Wälder erkennt und dass zweiterer zwar alle Wälder erkennt, aber auch einige wenige Nicht-Wälder als Wald ausgibt.

NBK Scikit-learn		Referenz			
		Wald	Nicht-Wald	Summe Klassierung	Users Accuracy
Klassierung	Wald	4187	1075	5262	0,796
	Nicht-Wald	114	6671	6785	0,983
	Summe Referenz	4301	7746	12047	
	Producers Accuracy	0,973	0,861		
	Overall Accuracy		0,901		

NBK für eindim. Merkmale		Referenz			
		Wald	Nicht-Wald	Summe Klassierung	Users Accuracy
Klassierung	Wald	4300	1800	6100	0,705
	Nicht-Wald	1	5946	5947	1,000
	Summe Referenz	4301	7746	12047	
	Producers Accuracy	1,000	0,768		
	Overall Accuracy		0,851		

Abbildung 11: Vergleich der Konfusionsmatrizen des NBK

Die Konfusionsmatrizen in Abbildung 11 vergleichen schließlich den aus der scikit-learn-Bibliothek implementierten *NBK* mit dem selbst implementierten *NBK* für eindimensionale Merkmale basierend auf einen zuvor berechneten NDVI des Trainingsgebietes. Wie man an den beiden OAs erkennen kann, sind beide *NBKs* ungenau, wobei der selbst implementierte *NBK* etwas unbrauchbarere Ergebnisse als sein scikit-learn-Pendant wiedergibt. Man sieht bei der Producer's Accuracy, dass er 100 % des Waldgebietes, allerdings aber nur 76,8 % des Nicht-Waldes richtig erkennt. Dies führt im Endeffekt dazu, dass 23,2 % des Nicht-Waldes als Wald ausgegeben werden. Der scikit-learn-*NBK* hingegen gibt 13,9 % des Nicht-Waldes als Wald aus.

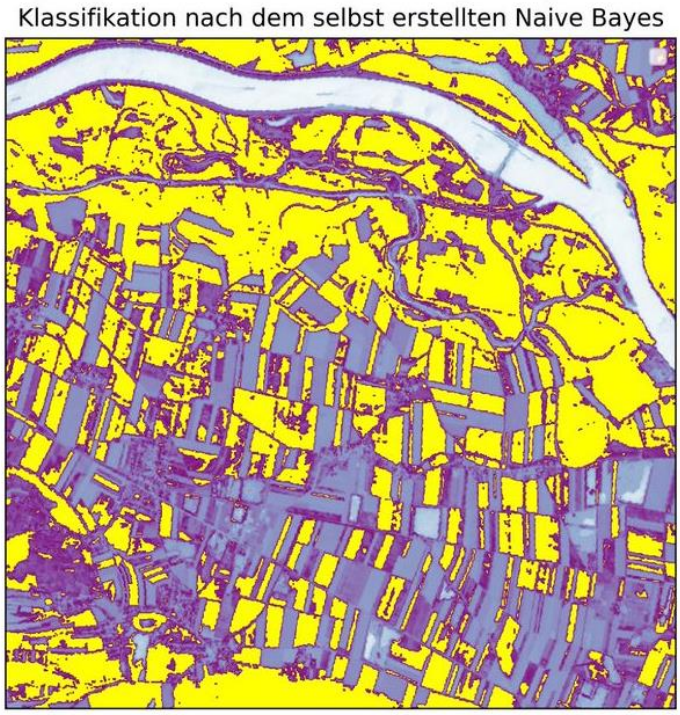

Abbildung 12: Selbst erstellter NBK

Abbildung 12 zeigt schlussendlich noch grafisch die vom selbst implementierten *NBK* als Wald erkannten Bereiche des Teilbildes in gelber Farbe. So erkennt man besser, dass wie bereits erwähnt 100 % des echten Waldes und ca. 25 % des Nicht-Waldes als Wald ausgegeben wurden.

Zusammenfassend kann man sagen, dass sich weder der von scikit-learn noch der selbst implementierte *NBK* auf NDVI-Basis zur Walderkennung eignen aufgrund deren Neigung, auch andere Vegetationsformen wie Wiesen als Wald zu erkennen. Sollte man einen der beiden dennoch verwenden, so ist ersterer zu bevorzugen. Am besten eignen sich die Klassifikatoren *SVM* und *Random Forest*, wobei unserer Meinung nach der *Random Forest* Klassifikator etwas bessere Ergebnisse liefert, da dieser alle Teile des Waldes und nur minimale Teile des Nicht-Waldes als Wald ausgibt.

Abbildungsverzeichnis

Literaturverzeichnis

Tuwel-Kurs Angewandte Fernerkundung
Scikit-learn-Bibliothek: https://scikit-learn.org/