Non-Time-Series-based Random-Forest for Land Cover Classification

ALEXEJ SHABAS

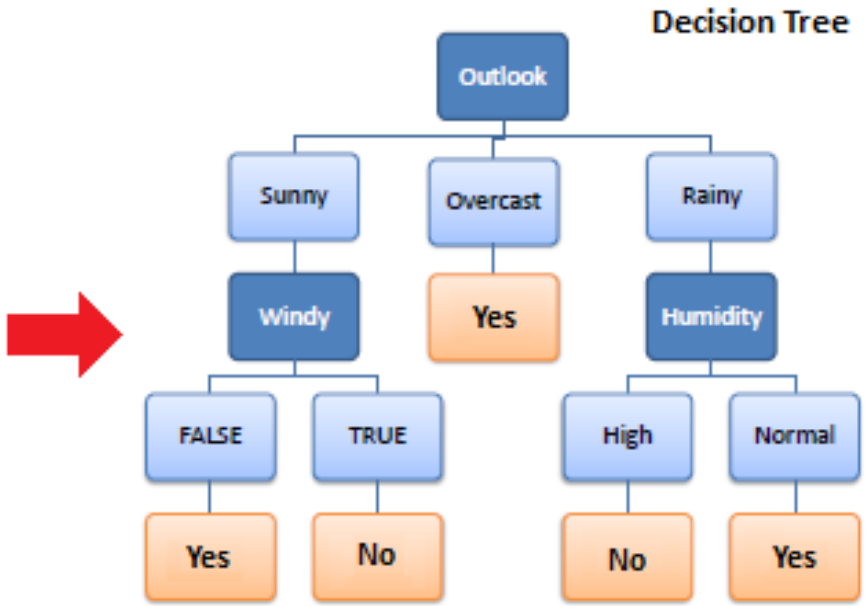
Gliederung

1. Random Forest
2. Mein Ansatz: Non-Time-Series-based
    1. Statistische Features
    2. Autoencoder Features
3. Fazit
4. Quellen

Random Forest – Decision Tree 1

- Random Forest besteht aus mehreren Decision Trees

Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



Grafik entnommen aus: *https://www.saedsayad.com/decision_tree.htm*

Random Forest – Decision Tree 2

- Top-down gebildet
  - Dataset in Teilmenge zerlegt
  - Teilmenge enthält homogene Samples
- Homogenität durch Entropy berechnet
  - 0 → Daten sind komplett homogen z.B. nur Samples mit Label 1
  - 1 → Daten sind gleichmäßig verteilt z.B. 50% Label 0 und 50% Label 1

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

9/14

$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)
= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971
= 0.693

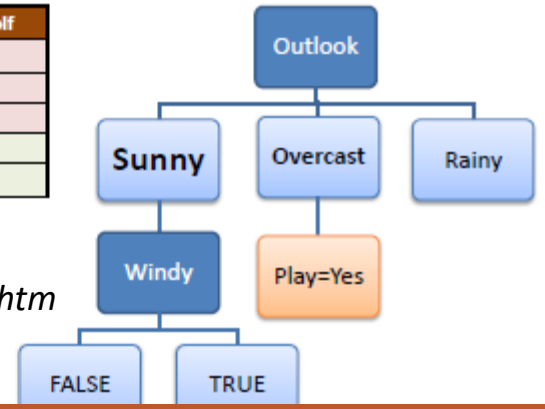
-((2/5)*log2(2/5))-((3/5)*log2(3/5))

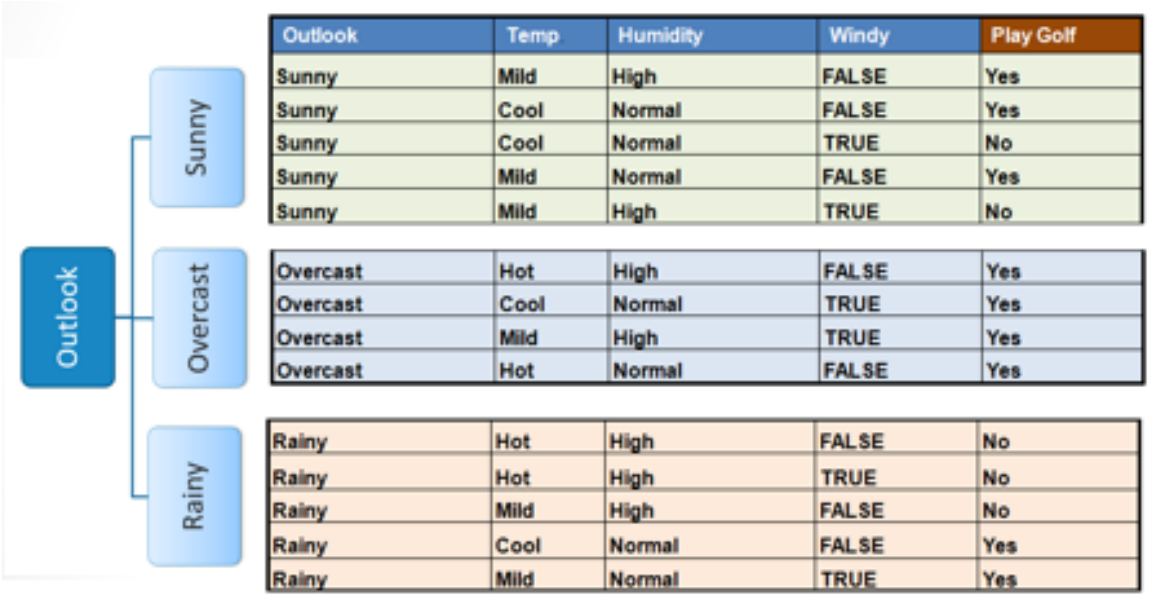
Grafiken entnommen aus:
https://www.saedsayad.com/decision_tree.htm

Random Forest – Decision Tree 3

- Für jedes Feature Entropy berechnen
- Information Gain: Totale Entropy – Feature Entropy
  - z.B. Information Gain für Feature „Outlook“: 0.94 – 0.693 = 0.247
- Feature mit höchstem Information Gain für Split verwenden
  - Dataset wird in homogene Teilmengen zerlegt
- Branch mit Entropy 0 → Leaf
- Branch mit Entropy >0 → rek. Aufruf auf Subset

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



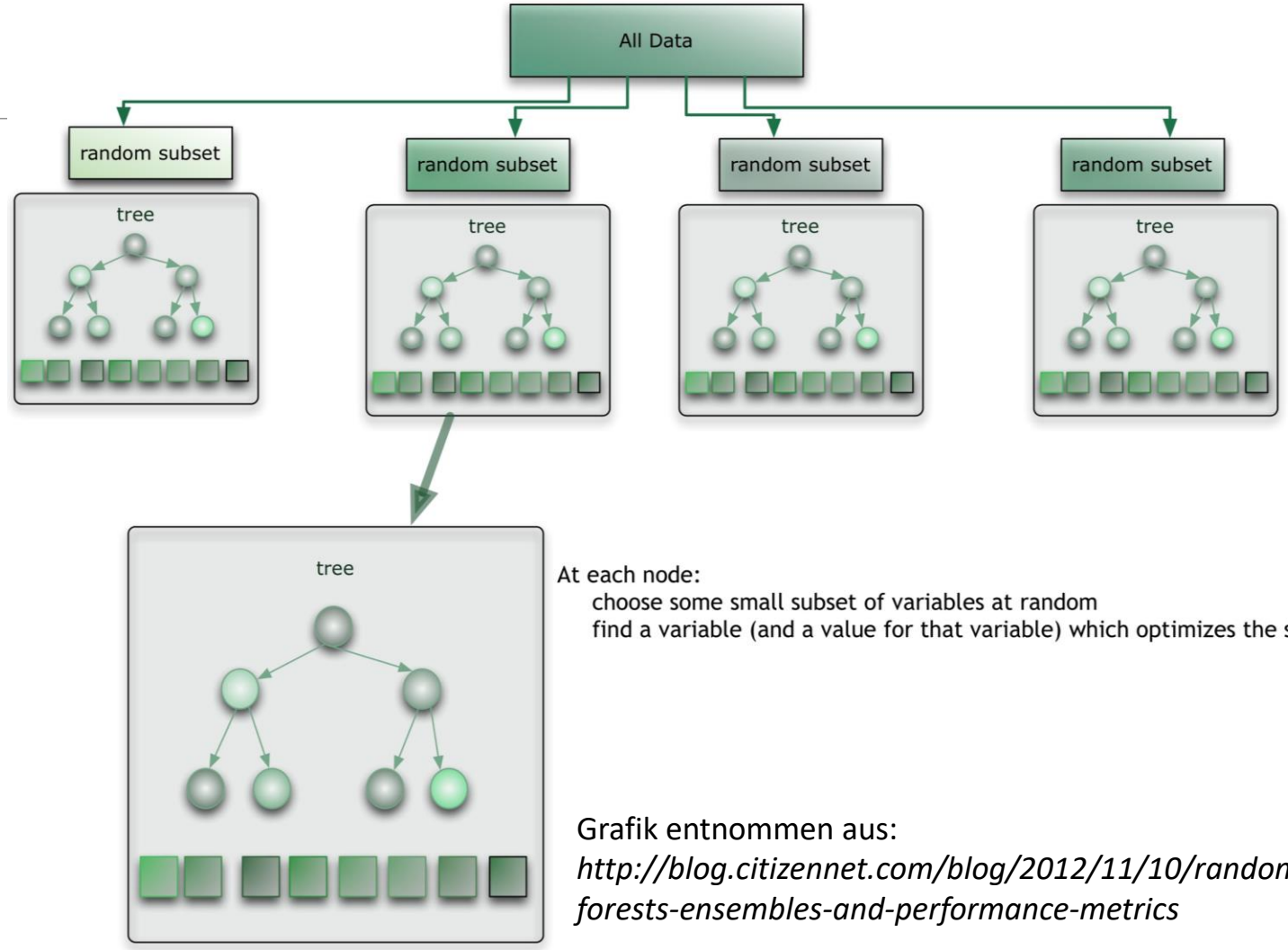


Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

Grafiken entnommen aus:
https://www.saedsayad.com/decision_tree.htm

Random Forest

- Forest: mehrere DTs
- Random:
  - Zufällige Teilmenge pro DT
  - Zufällige Features pro Knoten
    - D.h. zufällige Featureteilmenge
- Klassifizierung über „Abstimmung“



Grafik entnommen aus:
http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics

Random Forest – Vorteile/Nachteile

- Vorteile:
  - Einfach zu verwenden
  - wenig Overfitting
  - Parallel trainierbar
- Nachteile:
  - Nicht leicht zu interpretieren (Black Box)
  - Rechenintensiv

Mein Ansatz - Was bedeutet genau „Non-Time-Series-based“

- Temporale Abhängigkeit wird nicht explizit modelliert
- Jeder Zeitpunkt einer Zeitreihe wird als ein eigenständiges Feature interpretiert
- → Feature-Engineering
  - Statistische Feature
  - Autoencoder Feature
- Missing Values: Lineare Interpolation

Statistische Features

- Idee:
  - Statistische Werte einer Zeitreihe berechnen und als Features verwenden
  - z.B. Durchschnitt, Min, Max, …
- TSFresh Framework verwendet
  - Daten in ein bestimmtes Format umwandeln
  - Statistische Werte, die berechnet werden sollen, spezifizieren

sampleId (Zeilen-Nr)	Zeit	Zeitreihe 1 z.B. NIR	… Zeitreihe n z.B. RED
0	0	65	l
…	…	…	…
0	45	73	m
1	0	174	w
1	1	164	e

Statistische Features – TSFresh 1

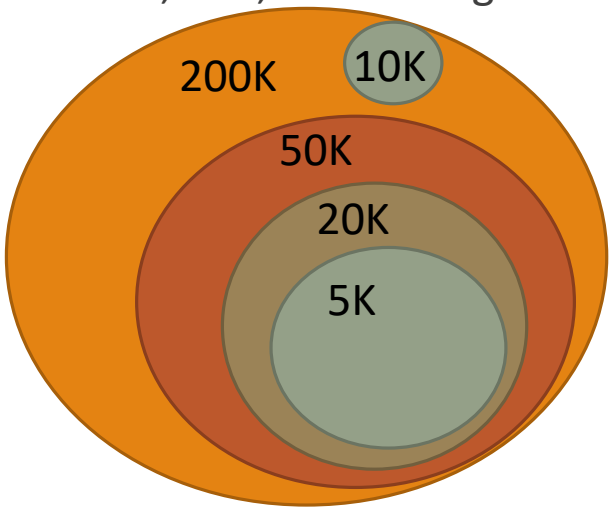
- Meine Idee: mehrere Zeitreihen ableiten
  - NDVI
  - saisonale Zeitreihen von Nir, Red, Green und NDVI
    - Frühling: erste 12 Zeitpunkte
    - Sommer: weitere 19 Zeitpunkte
    - Herbst: letze 15 Zeitpunkte
  - Moving Average mit unterschiedlichen Window-Sizes (3, 9, 20)
- Zeitreihen:
  - (nir, red, green, ndvi), (nirMA3, redMA3, greenMA3, ndviMA3), (nirMA9, redMA9, greenMA9, ndviMA9), (greenMA20, ndviMA20) → 14 Zeitreihen
  - Frühling, Sommer, Herbst jeweils: (nir, red, green, ndvi), (nirMA3, redMA3, greenMA3, ndviMA3) → 8 ZR
  - gesamt ca. 823MB (4 Dateien) bei 200K Subsample

Statistische Features – TSFresh 2

- 17 statistische Werte berechnet:
  - Mean, median, variance, variance_larger_than_standard_deviation, standard_deviation, count_above_mean, …
- In TSFresh viele weitere Berechnungen möglich, die man parametrisieren muss
- Anzahl potenziell möglicher Features: 92(ndviMA3 und greenMA3 Zeitreihen) + 238 (statistische Werte) + 3*136 (saisonale statistische Werte) = 738
- TSFresh Featurereduktion: 238 →226, 136 → 129 (Frühling), 136 → 130 (Sommer), 136 → 132 (Herbst)
  - Gesamt 709 Features
  - Hypothesentest, der Signifikanz jedes generierten Features checkt
  - Featurereduktion auf 200K Subsample trainiert, da sonst RAM voll (16GB RAM) → TSFresh verbrauch viele Resourcen
  - 200K Features Datei: 1.33GB
  - Ca. 2,5 Stunden (i5-6500, 4 Kerne 3.2 Ghz)

Statistische Features – Subsampling

- 200K für Featurereduktion-training
  - Top 8300 Samples pro Klasse mit geringster Anzahl der Missing Values
  - gesamt 189.881 Samples
- Weitere Subsamples für RF-Tuning
  - 10K Validation Set
  - 50K, 20K, 5K Training Sets



Statistische Features – RF Tuning

- Parameter:
  - "criterion": ["entropy", "gini"]
  - "max_depth" : [None, 10, 100]
  - "min_samples_split" : [2, 10]
  - "min_samples_leaf": [1, 10]
  - "max_features": ["sqrt", "log2"]
  - "class_weight": ["balanced_subsample", "balanced"]
- 2 Modelle trainiert auf 5K, 20K und 50K trainiert und besten Parameter angeschaut
  - 709 Features
  - PCA mit 98% Varianz auf 709 Features → 190 Features
  - Beide Modelle auf 10K Validation Set verglichen

Statistische Features – RF Tuning 2

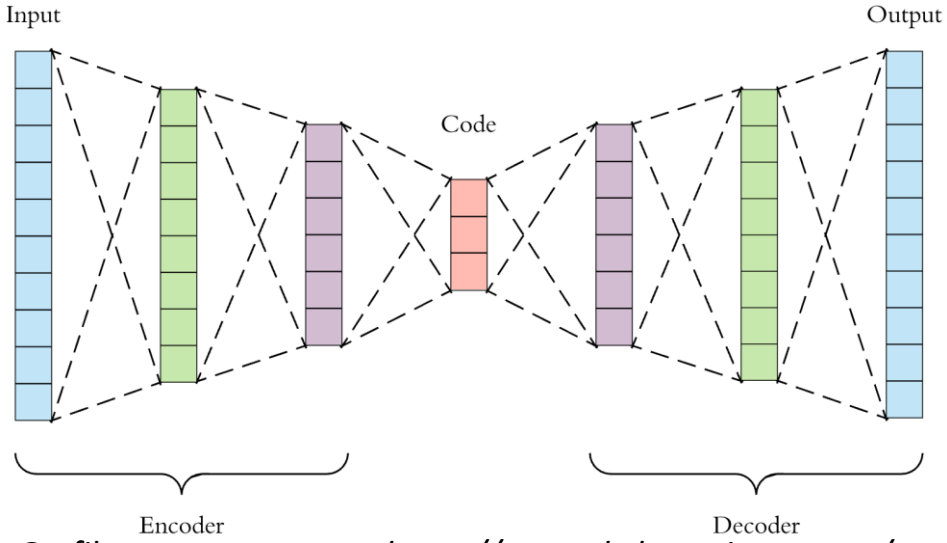
- 709 Features Modell immer um ca. 0.10 besser
- Immer gleiche beste Parameter:
  - "criterion": ["gini"]
  - "max_depth" : [None]
  - "min_samples_split" : [2]
  - "min_samples_leaf": [1]
  - "max_features": ["sqrt"]
  - "class_weight": ["balanced"]
- → für alle weiteren RF-Modelle diese Parameter verwendet

Statistische Features - Kaggle

- 709 Features Modell auf 200K trainiert mit 200 Trees → 0.62120
- 900K Sample generiert durch Splitting in 200K Sets, Featureextrahierung aus 200K Sets und Merging zu 900K Set (6.3GB)
  - Je Klasse Top geringste Anzahl an Missing Values
- 709 Features Modell auf 900K Subsample trainiert mit 200 Trees → 0.65850
- 190 Features (PCA) auf 900K Subsample mit 200 Trees → 0.53460
- → PCA verschlechtert das Resultat
- → Vergrößerung des Datasets bringt nicht viel

Autoencoder

- DNN
- Input = Output
- Encoder: lernt komprimierte Repräsentation
  - → Features
- Decoder
  - aus Komprimierung Eingabe rekonstruieren
- Unsupervised



Grafik entnommen aus: *https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798*

- Mein Ansatz supervised d.h. Outputvektor = MovingAverage(Inputvektor) + 24 Label-Knoten

Autoencoder Architekturen

- 5 Netze trainiert
- Jeweils ein DNN pro NIR, RED, GREEN, NDVI
  - 46 → 18 → 10 → 18 → 70 (46+24)
  - 10 Features pro Zeitreihe → gesamt 40 Features
- Ein DNN mit NIR, RED, GREEN kombiniert
  - 138 → 104 → 70 → 104 → 162 (138+24)
  - Gesamt 70 Features

Autoencoder Parameter

- NDVI Netz mit unterschiedlichen Parametern trainiert
  - Fest: 100 Epochen , 30% (von 3M.) Validation Data, 700.000 Batch, Loss: mean squared error
  - Gesucht: Aktivierungsfunktion (relu, sigmoid, tanh) und Gradientenverfahren (rmsprop, adam, sgd)
  - Rmsprop:
    - tanh, ... ,sigmoid: loss: 0.0176 - val_loss: 0.0176
    - relu, ..., tanh: loss: 0.0171 - val_loss: 0.0170
    - sigmoid, ..., sigmoid: loss: 0.0197 - val_loss: 0.0197
    - relu , ..., relu: loss: 0.1452 - val_loss: 0.1451
    - tanh, ..., tanh: loss: 0.0164 - val_loss: 0.0163 → am besten
  - Adam:
    - tanh, ..., tanh: loss: 0.0195 - val_loss: 0.0195
  - Sgd:
    - tanh, ..., tanh: loss: 0.1965 - val_loss: 0.1956
- Kaggle: NDVI Netz mit Rmsprop, tanh + RF 200 Trees und 10 Features → 0.57609

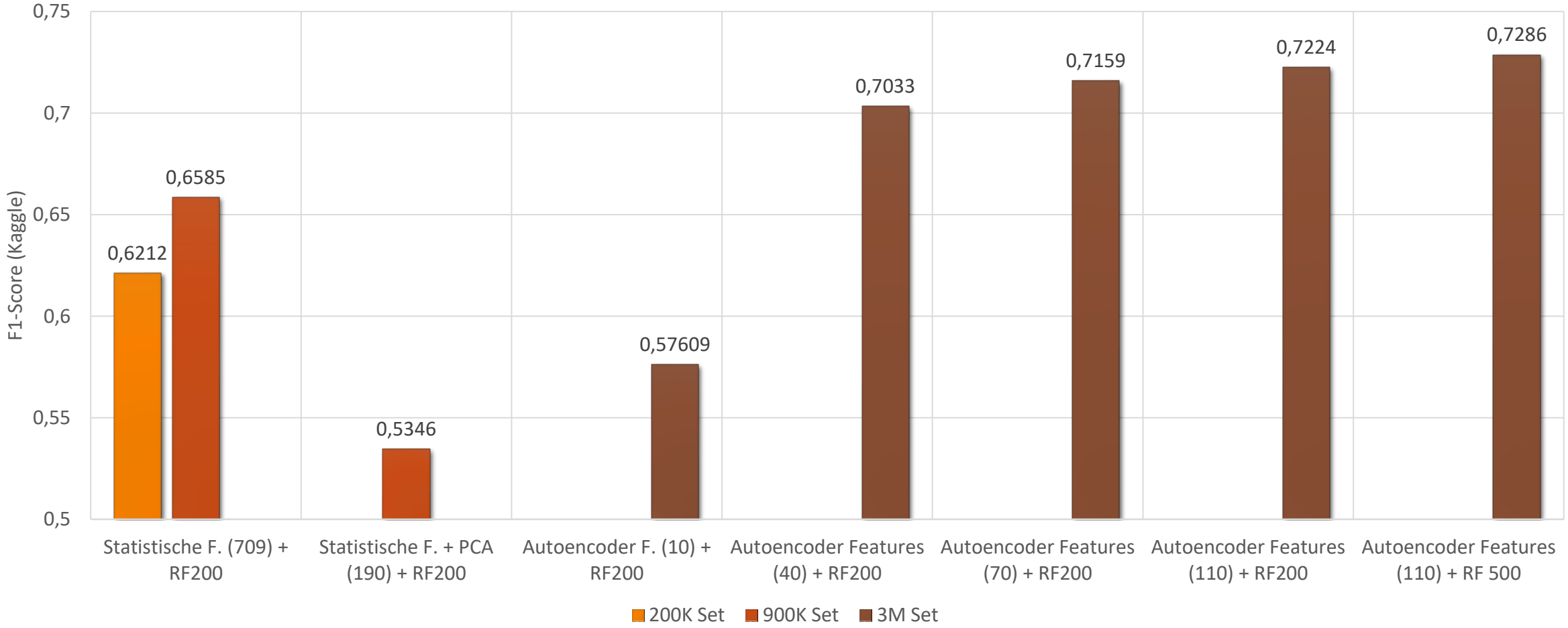
Autoencoder Training

- ca. 3M Dataset
  - Pro Klasse Top kleinste Anzahl der Missing Values
- GPU: GeForce GTX 1060 6GB
- Alle 5 Netze mit Batchsize 1024 und 250 Epochen trainiert
  - Nach ca. 100 Epochen kaum Änderung in Loss
  - Green loss: 0.0080
  - Red loss: 0.0077
  - Ndvi loss: 0.0078
  - Nir loss: 0.0082
  - Nir, red, Green loss: 0.0020
  - Pro Netz ca. 60 Min. Trainingszeit

Autoencoder - Kaggle

- RF 200 Trees (23 Depth) + 40 Features: 0.70330
- RF 200 Trees (23 Depth) + 70 Features: 0.71590
- RF 200 Trees (23 Depth) + 110 Features: 0.72240
- RF 500 Trees + 110 Features: 0.72860
- RF Trainingszeit: jeweils 3-4 Stunden

Übersicht der Modelle



Fazit

- Autoencoder Features deutlich besser als statistische Features

Quellen

- http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics
- Gómez, Cristina, Joanne C. White, and Michael A. Wulder. "Optical remotely sensed time series data for land cover classification: A review." ISPRS Journal of Photogrammetry and Remote Sensing 116 (2016): 55-72.
- https://medium.com/regen-network/remote-sensing-indices-389153e3d947
- https://github.com/blue-yonder/tsfresh
- https://www.saedsayad.com/decision_tree.htm