Ostfalia
Hochschule für angewandte Wissenschaften
Wolfenbüttel

Fakultät Informatik

Robin Müller, 70460120, Pit-Aurel Ehlers, 70453261

Python Machine Learning: scikit-learn

Mobilität Track B

Betreuer:
Prof. Dr. Claus Führner

Salzgitter
Suderburg
Wolfsburg

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist.

Wolfenbüttel, den 6. Februar 2020

Kurzfassung

In dieser Seminararbeit soll ein Einblick in die Welt des Machine Learning gegeben werden, wobei ein Oberthema Mobilität darstellt. Um das Thema am besten zu durchdringen wird zunächst ein Überblick über die vielfältigen Themenbereichen, Problemstellungen und Herangehensweisen geliefert. Außerdem werden zu jeder der drei Problematiken (Classification, Regression und Clustering) oberflächlich einige Algorithmen erklärt, um diese in der späteren Anwendung besser zu verstehen. Als letzter Punkt des ersten Teils dieser Arbeit wird SciKit-Learn vorgestellt: eine Python-Bibliothek die einen simplen Einstieg in Machine Learning verspricht. Mithilfe der von SciKit bereitgestellten Tools wird im zweiten Teil der Arbeit zunächst ein Experiment aus dem Buch “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems” (Aurélien Géron, 2019) nachgestellt, bei dem die Wohnungspreise in Kalifornien mit Machine Learning vorhergesagt werden sollen. Anschließend wird das Experiment um einige Aspekte erweitert, um vergleichen zu können, welche Algorithmen am besten sind. Nach, aber auch schon während, der Projektdurchführung hat sich herausgestellt, dass ein falscher Ansatz verfolgt wurde. Es lässt sich kein bester oder schlechtester Algorithmus definieren, vielmehr muss genau überlegt werden, welche Herangehensweise für die vorliegende Aufgabe die richtige ist, um das beste Ergebnis zu erhalten.

Abstract

This project paper offers a insight into the world of machine learning with some relatedness to the field of mobility. To give the best understanding about machine learning, initially there will be a brief overview about the diverse topics, different approaches, strategies and complexities. Besides you can find some shallow explanation of algorithms for the three basic problematics (classification, regression, clustering) to have a better understanding for them, while using them in the second part of our paper. The last paragraph of the first part of this paper is an introduction to scikit-learn, which is a python-library that offers functions and methods to help with machine learning projects. The second part of this paper is a whole machine learning project with the purpose to compare different kinds of algorithms, using scikit's tools. Therefor an experiment from the book "Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems" (Aurélien Géron, 2019) was postpositioned and extended, to make a comparison. After, but even while, the execution of the project it became more and more clear that an incorrect approach has been used. Those algorithms cannot reasonably be compared to each other, because every different complexity demands a very different kind of strategy to master it.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ML Machine Learning

SVM Support Vector Machine

RMSE Root Mean Spuare Error

Abbildungsverzeichnis

1 Einleitung

Diese Seminararbeit beschäftigt sich mit häufigen Problemstellungen des Machine Learnings. Des weiteren wird sich mit den verschiedenen Methoden beschäftigt, welche Scikit zur Lösung dieser Probleme bereitstellt. Ziel dieser Arbeit ist es dem Leser ein besseres Verständnis für Machine Learning zu vermitteln. Es soll ihm auch näher bringen, welche Methoden von Scikit für verschiedene Problemstellungen am besten sind. Im ersten Teil der Seminararbeit wird zunächst ein generelles Verständnis für Machine Learning(ML) geschaffen. Im zweiten Teil werden verschiedene Scikit Funktionalitäten anhand des Beispiels “Boston housing Dataset” angewandt. Im letzten Teil der Arbeit sollen die verschiedenen Algorithmen miteinander verglichen werden.

2 Motivation

Machine-Learning wird in der heutigen Zeit immer wichtiger und entwickelt sich immer weiter.[1] Machine-Learning findet sich schon heute in vielen Bereichen unseres Lebens wieder, wie z.B. in der Automobilindustrie. Vor allem im Komplex des Autonomen Fahrens steht die Industrie vor zahlreichen Herausforderungen, die es noch zu meistern gilt und von denen einige nur mit Hilfe von ML gelöst werden können. Eine einfache Aufgabe ist dabei noch das Erkennen von Straßenschildern. Eine Software zu schreiben, die Schilder unter allen möglichen Wetterbedingungen, Winkeln, Lichtverhältnissen etc. zuverlässig erkennt stellt eine unfassbar große Herausforderung dar, die sich jedoch mit ML relativ einfach meistern lässt, indem ein System auf entsprechend großen Datasets trainiert wird. Einen einfachen Einstieg in das Thema ML bieten Projekte wie SciKit-Learn. SciKit-Learn macht es jedem, der grundlegende Programmierkenntnisse besitzt, möglich selbst mit ML zu experimentieren, ohne die Algorithmen selbst implementieren zu müssen.

2.1 Zielsetzung

Ziel der Arbeit ist es, ein grundlegendes Verständnis für Machine Learning Projekte und Problematiken zu entwickeln und zu vermitteln.

3 Einführung Machine Learning

Um sich mit Machine Learning auseinanderzusetzen, muss zunächst die Theorie verstanden werden, welche sich dahinter verbirgt. Anders als bei herkömmlicher Programmierung, in der ein Programm und Daten einen Output erzeugt, leitet man beim Machine Learning aus gesammelten Daten und Outputs ein Programm her [2, S. 3]. Dies wird in der unten stehenden Grafik verdeutlicht.

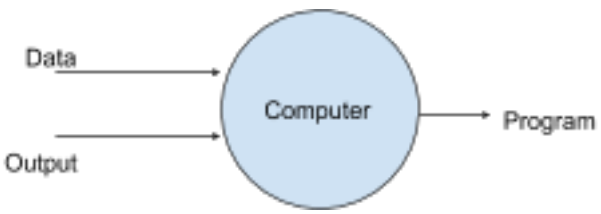

Abbildung 3.1: Machine Learning

[2, S. 3]

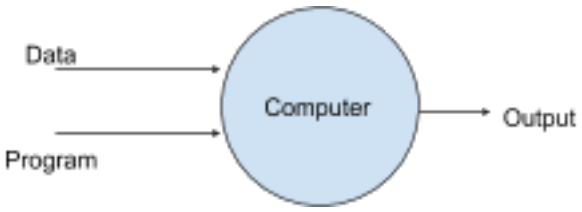

Abbildung 3.2: Herkömmliche Programmierung

[2, S. 3]

4 Einführung verschiedener Problemstellungen

Nicht nur bei den Algorithmen gibt es verschiedene Unterteilungen, auch die Probleme, welche man mit Machine Learning (ML) lösen möchte, lassen sich in 3 Hauptkategorien unterteilen.

4.1 Classification

In diesen Fällen versucht man mit hilfe von ML zu ermitteln in was für eine Kategorie ein Objekt gehört[3]. Dies realisiert man, indem man das System mit einem Dataset der möglichen Kategorien trainiert. Das Ergebnis eines Classification Problems ist entweder ein diskreter oder ein kontinuierlicher Wert [2, S. 4]. Spamfilter wären ein Beispiel eines Classification Problem, in dem Mails beispielsweise in die Kategorien Spam und kein Spam eingeteilt werden könnten.

4.1.1 Support Vector Machine (SVM)

Ziel des SVM Algorithmus ist es, eine Hyperebene(Ebene mit n-1 Dimensionen) in einen n-dimensionalen Raum zu legen (wobei n die Anzahl der Features ist), sodass alle Datensätze einer Klasse auf einer Seiter der Hyperebene liegen und alle Datensätze einer anderen Klasse auf der anderen Seite. Dann kann für neue Daten aufgrund ihrer Position in Beziehung zu Ebene bestimmt werden, welcher Klasse diese angehört [2, S. 177f.].

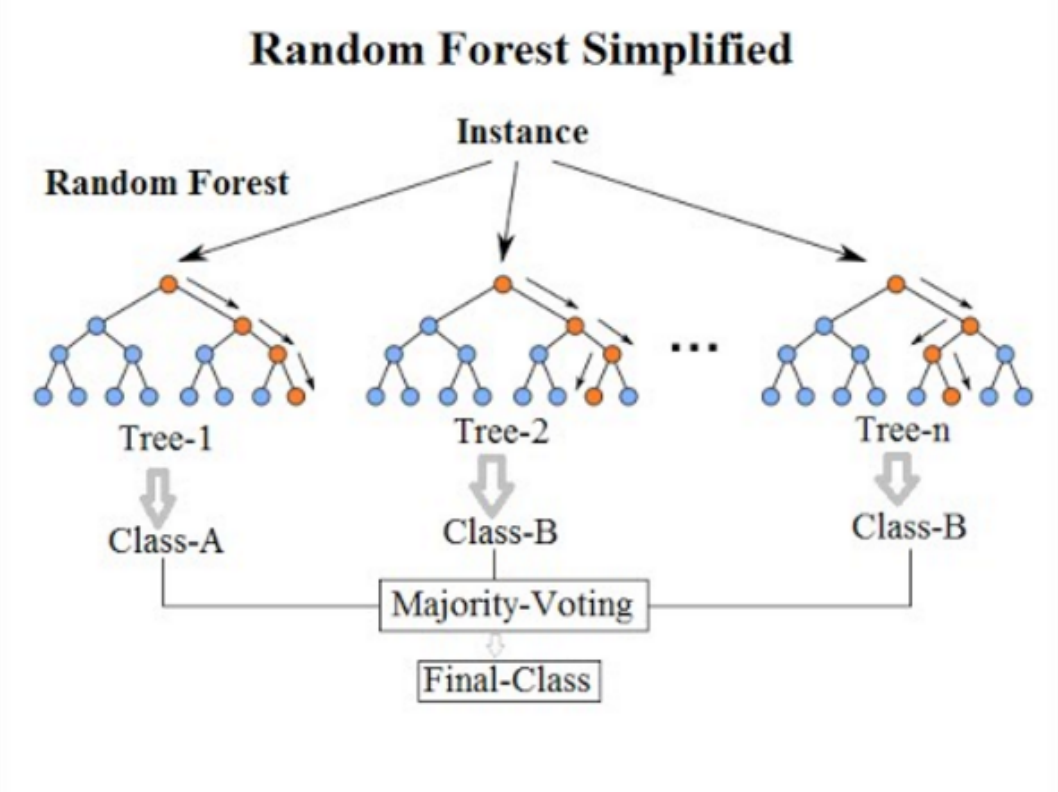

Abbildung 4.1: Diagramm: Random Forrest

4.1.2 Random Forest Classifier

«««< HEAD Bei diesem Algorithmus werden aus dem Trainingsset zufällig Teilmengen genommen,mit denen verschiedene Modelle trainiert werden[1, S. 256]. Diese verschiedenen Modelle werden durch Decisiontrees trainiert. Später kommt es durch ein “Mehrheitsvotum” zu einer finalen Einordnung der Klasse[4]. Dies wird in folgender Abbildung verdeutlicht.

======= Bei diesem Algorithmus werden aus dem Trainingsset zufällige Teilmengen genommen,mit denen dann mehrere Decisiontrees trainiert werden. Daher auch der Name "Random Forest", da dieser aus vielen verschiedenen Bäumen besteht[1, p. 256]. Bekommt dieser Algorithmus einen Input gibt er ihn an jeden einzelnen Tree weiter, jeder für sich fällt dann unabhängig seine eigene Entscheidung. Später kommt es durch ein “Mehrheitsvotum” der einzelnen Trees zu einer finalen Einordnung der Klasse, wodurch dieser Algorithmus deutlich weniger anfällig für Overfitting ist als ein einzelner Decisiontree[4]. [Grafik Random Forest] »»»> 1256aa2475e4a5bcfd169577550135e3e87f4ce8

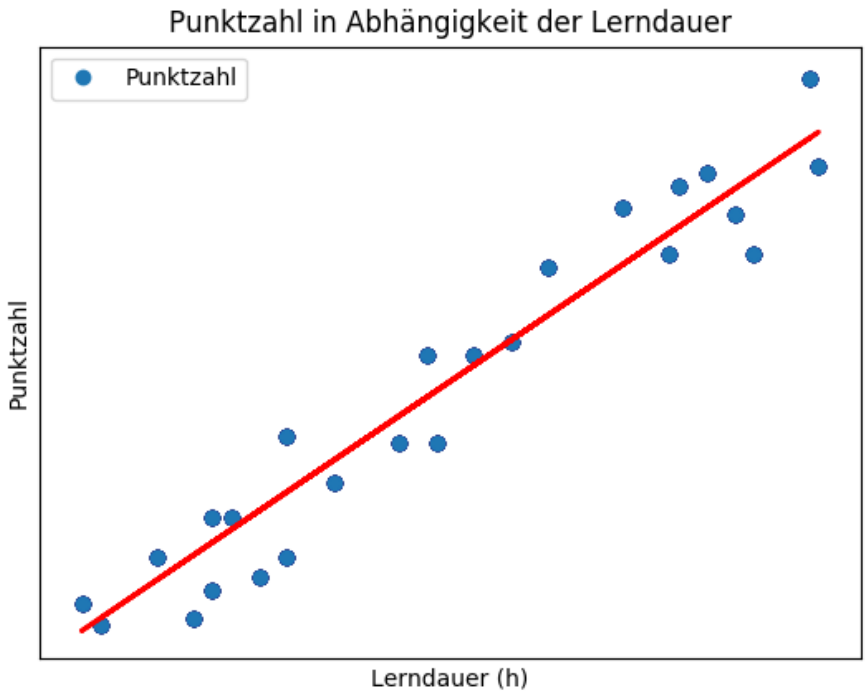

Abbildung 4.2: Diagramm: Regression

4.2 Regression

Hier wird versucht mit ML eine Prognose zu machen. Diese Prognose entsteht, indem die KI verschiedene Variablen in Beziehung setzt und dadurch eine Vorhersage ermittelt. Als Ergebnis liefert die KI in diesem Fall einen kontinuierlichen Output [2, S. 4.]. Ein typisches Beispiel für Regression ist beispielsweise die Vorhersage der erreichten Punktzahl in einem Test im Verhältnis zur Lerndauer in Stunden. Dies wird in folgender Abbildung verdeutlicht.

4.2.1 Lineare Regression

Gilt als ein sehr einfacher Algorithmus, bei dem versucht wird eine lineare Annäherung zwischen einer Abhängigen und einer Unabhängigen Variablen zu ermitteln[2, S. 100.].

4.2.2 Decisiontree Regression

Decisiontrees werden bei classification und regression Problemen eingesetzt. Ziel dieses Algorithmus ist es, von einem Dataset Regeln zu lernen und anhand dieser Vorhersagen zu machen.Der Decisontree wird genauer, wenn dieser größer wird oder die Auswahlmöglichkeiten (decisions) komplexer werden[5].

4.2.3 Random Forest Regression

Dieser Algorithmus funktioniert wie der vorher schon genannte Random Forest Classifier, nur mit dem unterschied, dass am ende keine Finale Klasse entsteht, sonder ein kontinuirlicher Output. Um diesen zu erreichen wird anders als beim Random Forest Classifier bei der Abstimmung kein Majority-Voting durchgeführt, sondern es wird der Mittelwert der Ergebnisse von den verschiedenen Decision Trees gebildet.

4.3 Clustering

Bei Clustering Problemen wird versucht mit hilfe von ML verschiedene Daten aus einem Dataset in verschiedene Gruppierungen einzuordnen[2, S. 5.]. Clustering Probleme treten dann auf, wenn z.B. jemand eine Kundenanalyse durchführen möchte. Solch eine Kundenanalyse kann aufschlussreich sein, um das Kaufverhalten verschiedener Arten von Kunden zu analysieren.

4.3.1 K-Means

K-Means wird bei einem unlabeled Dataset angewendet und versucht in diesem K Gruppen zu finden. Dazu werden zunächst k zufällige Datenpunkte ausgewählt und dann anhand dieser die restlichen Daten zu den Clustern zugeordnet. Anschließend wird der Mittelwert(Mean) jedes Clusters bestimmt und die Daten werden anhand der neuen Means zu Clustern zugeordnet. Dieser Vorgang wird wiederholt, bis sich die Mittelwerte nicht mehr ändern. Bei K-Means werden beliebig viele solcher Modelle erstellt, und am Ende das verwendet, welches die geringste Varianz bietet[2, S. 222.].

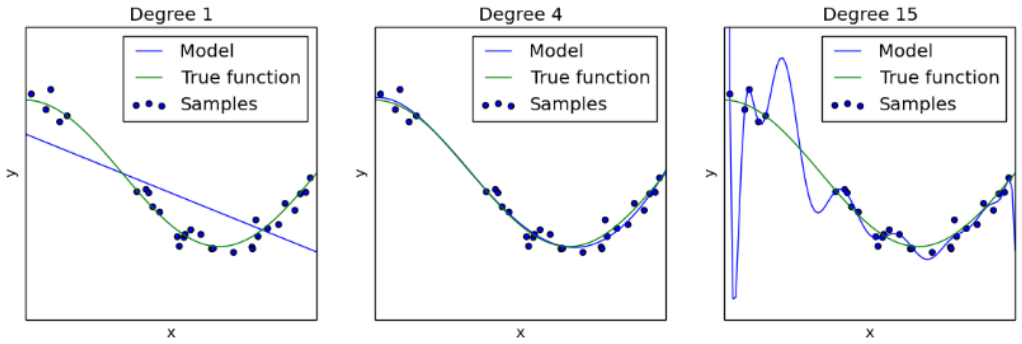

Abbildung 4.3: Overfitting Underfitting

4.4 Overfitting/Underfitting

Damit man durch die genannten Algorithmen ein vernünftiges Ergebnis erzielen kann muss man beachten, dass die KI nicht overfittet oder underfittet, da es so zu unbrauchbaren Ergebnissen kommt.

4.4.1 Overfitting

Beim Overfitting wurde das Modell so trainiert, dass es versucht jeden einzelnen Punkt aus den Trainingsdaten zu berücksichtigen. So werden auch unwichtige schwankungen der Daten beachtet. Bei Modellen die overfitten ist das größte Problem, dass dieses Modell nicht für neue Daten geeignet ist[2, S. 214.].

4.4.2 Underfitting

Anders als beim Overfitting passiert bei Underfitting im Prinzip das Gegenteil und das Modell schafft es nicht den Verlauf der Daten zu erfassen[2, S. 214.]. Ziel ist es ein Modell zu finden, welches sich zwischen Overfitting und Underfitting befindet. Die unten abgebildeten Graphen sind jeweils ein Beispiel für Underfitting, Overfitting und einem Modell, welches erstrebenswert ist. Das Modell Degree 1 ist das Modell, welches underfittet, das Modell Degree 4 ist das angestrebte und das Degree 15 das overfittete Modell.

$$RMSE(X, h) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2}$$

Abbildung 4.4: Berechnung des Root Mean Square Error

4.5 Performance Measure

Um die Genauigkeit der verschiedenen Algorithmen vergleichen zu können, muss eine Formel zur Bestimmung der Fehler gewählt werden. Die übliche Methode ist der “Root Mean Spuare Error”. Diese gibt einen Ausblick auf die Anzahl an Fehler, die das System typischerweise macht, wobei größere Fehler auch schwerer gewichtet werden[6, S. 39.].

5 SciKit-Learn

Scikit-Learn ist ein Projekt, welches 2007 von David Cournapeau gestartet wurde[3]. Ziel dieses Projektes ist es Machine-Learning für jeden zu vereinfachen, indem sie verschiedene Python-Libraries erstellen. Da die Algorithmen des Machine-Learning nur durch sehr komplex Berechnungen zu realisieren sind, stellt scikit verschiedene Libraries zur verfügung, welche diese komplexen Berechnungen für den Programmierer übernimmt, so dass oftmals nur wenige Funktionsaufrufe nötig sind, um eine KI zu erstellen. Aber scikit besteht nicht nur aus Funktionen, welche komplexe Berechnungen durchführen, es gibt auch Funktionen, wie z.B. train_est_split(), welche festlegt, welche Daten zum trainieren und welche zum Testen genommen werden.

6 Beispielprojekt

6.1 Projektvorstellung

Um die beschriebenen Herangehensweisen aus dem vorherigen Kapitel besser verstehen und nachvollziehen zu können, soll in diesem Kapitel ein eigenes Projekt durchgeführt werden. Dieses wird uns sowohl mit der Umsetzung der oben beschriebenen Konzepte bekannt machen, also auch eine grundlegende Einführung in die Funktion von SciKit-Learn ermöglichen. Um einen Überblick über dieses umfassende Thema zu bekommen, werden zunächst die drei verschiedenen Herangehensweisen(Classification, Regression und Clustering) jeweils mit zwei verschiedenen Algorithmen implementiert. So werden die Unterschiede und Gemeinsamkeiten herausgearbeitet, um ein Vergleich der einzelnen Implementierungen zu ermöglichen. Hierbei muss natürlich bedacht werden, dass sich für jedes Dataset andere Algorithmen und Herangehensweisen als am besten geeignet erweisen: Um die Werte einer linearen Funktion zu bestimmen eignet sich vermutlich eher eine Regression als eine Classification oder Clustering. Doch genau diese Fragestellungen sollen in diesem Projekt behandelt und beantwortet werden. Zunächst wird dazu ein Experiment aus “Hands-On Machine Learning with SciKit-Learn” nachgestellt, und dann mit eigenen Ansätzen weitergearbeitet.

6.2 Datensatz

Für das Projekt wurde der “California Housing Prices” Datensatz gewählt. Dieser stammt aus R. Kelly Pace’s and Ronalds Berry’s “Sparse Spatial Autoregressions” Statistics & Probability Letters 33 no. 3 (1997). Wie der Name schon vermuten lässt handelt es sich bei diesen Daten um die Preise für eine Haus in Kalifornien in Abhängigkeit von dessen Lage. Die Werte stammen aus 1990 und sind somit nicht

mehr relevant, dennoch bieten sich viele Möglichkeiten an diesen Daten zu lernen und zu experimentieren, für das geplante Projekt also die perfekten Voraussetzungen.

6.3 Vorbereiten der Entwicklungsumgebung

Auf dem für das Projekt zur Verfügung stehenden Computer muss Python mit folgenden Modulen installiert sein:

- jupyter
- matplotlib
- numpy
- pandas
- scipy
- scikit-learn

Zusätzlich wird das Modul virtualenv installiert um das Projekt in einer isolierten Umgebung durchführen zu können. Als letztes wird der Jupyter-Server gestartet.

7 Projektdurchführung

Das Jupyter-Notebook das zur Durchführung dieses Projekts angelegt wurde, kann unter https://github.com/robbmue/MLSKLSeminar eingesehen werden.

In diesem Kapitel werden als erstes die Daten aus dem Datensatz importiert und aufgearbeitet. Dann werden die Daten visualisiert und mithilfe der, aus der Visualisieren gewonnenen, Kentnisse in eine sogenannte Pipeline überführt, welche die Aufbereitung der Daten übernimmt. Anschließend können dann unterschiedliche Algorithmen aus den Feldern Regression und Classification mit Diesen trainiert und verglichen werden, eine Überprüfung der Ergebnisse, mithilfe von Cross-Validation, und eine abschließende Evaluation durchgeführt werden.

7.1 Vorbereitung der Daten 1

Zunächst muss der Datensatz, der verwendet werden soll in das System importiert werden, wofür der vorbereiteten Datensatz aus “Hands on Machine Learning” verwenden konnte[7]. Die Daten müssen anschließend entpackt werden und können dann mithilfe von pandas eingelesen werden. Anschließend können Informationen über den Datensatz angezeigt werden. Diese finden sich im der Abbildung Attribut Datensatz. Dort lässt sich erkennen, dass der Datensatz 20640 Einträge enthält. Eine Ausnahme bildet dabei das Attribut “total_bedrooms”. Diese Problem wird in Kapitel 6.3 Vorbereitung der Daten 2 behandelt. Um eine genauere Vorstellung des Datensatzes zu bekommen, mit dem gearbeitet wird, kann außerdem ein Histogramm erstellt werden. Dieses findet sich in der Abbildung Histogramm. Dann muss dann das zur Verfügung stehende Dataset in zwei aufgesplittet werden. Ein Richtwert für die Aufteilung ist 80/20: 80% der Daten befinden sich nach dem Split im train_set und 20% im test_set. Zuletzt soll das Income-Feature in Kategorien eingeteilt werden, da dieses Feature für das Endergebnis eine besonders große Rolle

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Abbildung 7.1: Attribute Datensatz

spielt. Hier wurde eine Aufteilung in 5 Schichten gewählt, wobei jede ein Einkommen von +$15,000 repräsentiert. Also $0 - $15,000, $15,000 - $30,000 usw. wobei die letzte Kategorie nach oben nicht begrenzt ist, also >$60,000. Ein Histogramm dieser Kategorien ist das Histogramm Verteilung Income. Mithilfe dieser Kategorien und des StratisfiedShuffleSplits, den scikit-learn zur Verfügung stellt, können Train und Test Sets gebildet werden, die die Verteilung der Einkommensschichten repräsentieren können. Diese werden strat_train_set und strat_test_set genannt.

7.2 Visualisierung der Daten

Um ein noch besseres Verständnis für die Daten zu bekommen werden diese auf verschiedene Arten visualisiert. Als Erstes als die reine Verteilung unserer Daten auf den Raum Kalifornien. Jedes Datum wird mit einem Punkte repräsentiert, dicht besiedelte Gebiete werden hervorgehoben. Da als X- und Y-Achsen die Koordinaten gewählt wurden, wird hier die Form Kaliforniens abgebildet. Dies ist in der Grafik Räumliche Verteilung der Datensätze wiederzufinden.

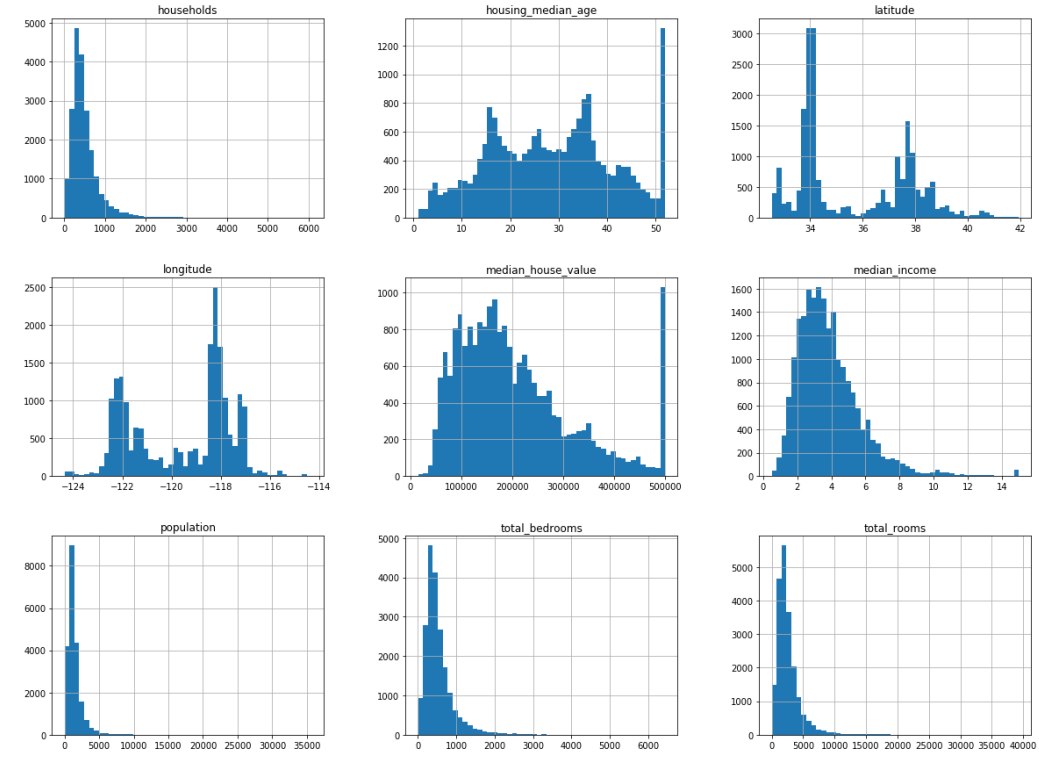

Abbildung 7.2: Histogramm

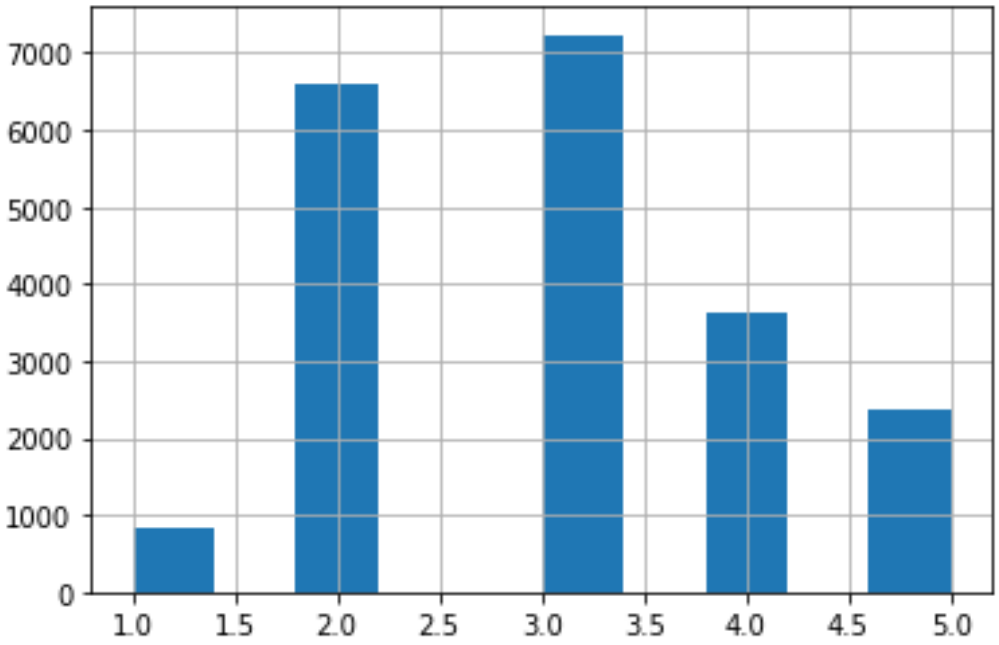

Abbildung 7.3: Histogramm: Verteilung Income

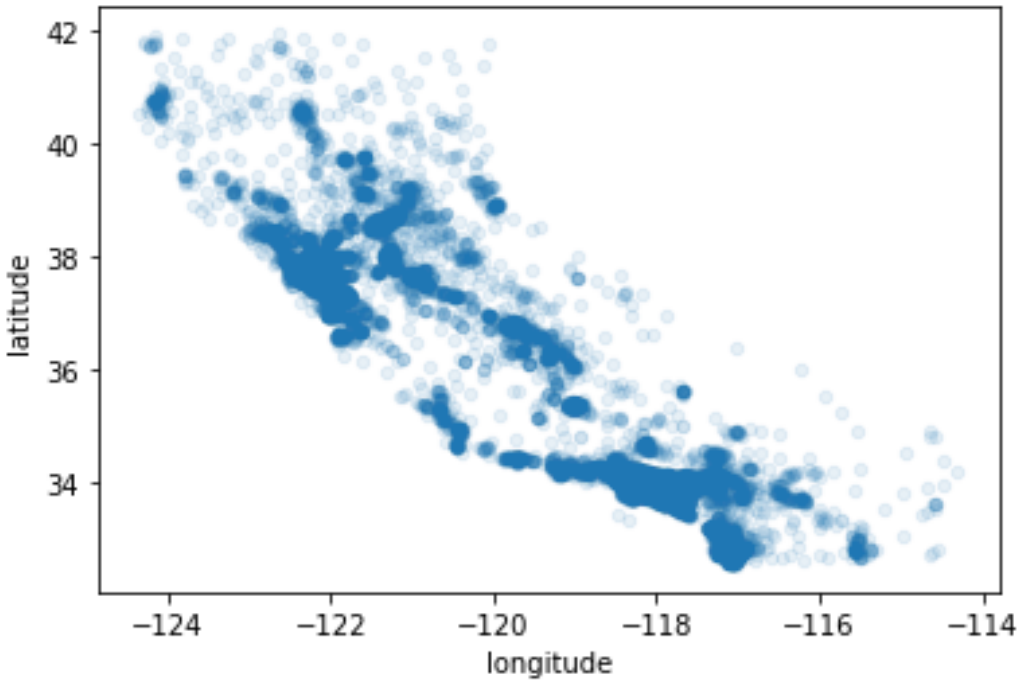

Abbildung 7.4: Räumliche Verteilung der Datensätze

Damit die Muster, die dieser Grafik zugrunde liegen noch besser verstanden werden können, werden sie in eine Jet-Grafik umgewandelt. Rot symbolisiert dabei die hohen Grundstückspreise und Blau die niedrigen.

7.3 Vorbereitung der Daten 2

Mithilfe dieser Informationen können die Daten weiterverarbeitet werden. Dafür konnten die von scikit-learn implementierten Module Pipeline, StandardScaler, ColumnTransformer und OneHotEncoder genutzt werden, auf deren Funktion hier nicht weiter eingegangen werden soll, das der Schwerpunkt des Projekts auf dem Vergleich der Machine-Learning-Algorithmen liegen soll und nicht auf dem Vorbereiten der Daten. Deswegen an dieser Stelle nur ein kurzer Überblick über die durchgeführten Schritte:

1. Die nicht vorhandenen total_bedrooms Werte werden mit dem Mittelwerte aller total_bedrooms befüllt

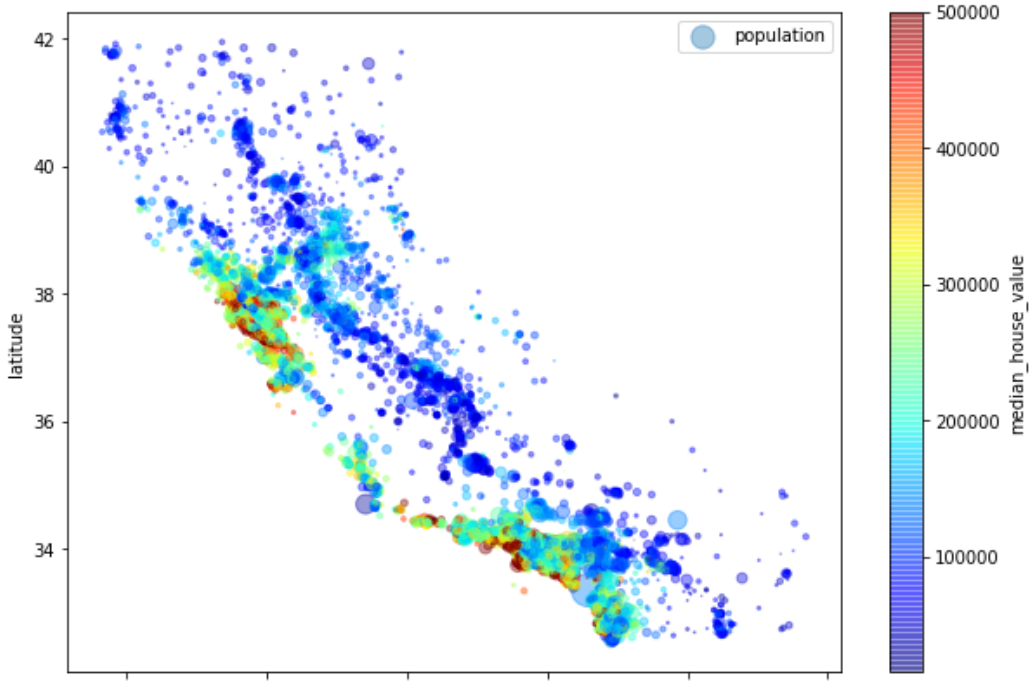

Abbildung 7.5: Hier lässt sich erkennen, dass für unser vorhaben vor allem die Attribute Lage und Bevölkerungsdichte wichtig sein werden. Mithilfe dieser Visualisierungen lassen sich also wichtige Informationen herausarbeiten.

```
from sklearn.linear_model import LinearRegression  LinearRegression: <class 'sklearn.linear_model._ba

lin_reg = LinearRegression()  LinearRegression: <class 'sklearn.linear_model._base.LinearRegression'>
lin_reg.fit(housing_prepared, housing_lables)  lin_reg: LinearRegression(copy_X=True, fit_intercept=T
```

Abbildung 7.6: Code: Lineare Regression

```
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)  lin_reg: LinearRegression
lin_mse = mean_squared_error(housing_lables, housing_predictions)  housing_lables:
lin_rmse = np.sqrt(lin_mse)  lin_mse: 4709829587.97112
print(lin_rmse)  lin_rmse: 68628.19819848922
```

Abbildung 7.7: Code: Evaluation Regression

2. Es werden aus den vorhandenen Attributen neue, sinnvollere Attribute gebildet. Zum Beispiel ist das Attribut rooms_per_household deutlich sinnvoller als die einzelnen Attribute total_roooms oder total_households pro District.

3. Alle Werte werden standardisiert, was bedeutet dass der Mittelwert eines Attributes immer 0 ist, und die restliches Werte entsprechend um diesen gebildet werden.

4. Auf den nicht numerischen Wert ocean_proximity wird der OneHotEncoder angewandt: Es wird ein Binärer Wert erzeugt, der den verschiedenen Kategorien entsprechend viele Stellen erhält. Für jede Kategorie wird dann also einer dieser Werte zu 1.

7.4 Durchführung Linear Regression

Folgendes Code Beispiel soll veranschaulichen wie simpel es mithilfe von scikit-learn ist, einen Machine-Learning Algorithmus zu trainieren: Mit diesen wenigen Zeilen wurde zunächst das entsprechende Modul importiert, dann ein entsprechendes Objekt erzeugt und trainiert. Auch die Evaluierung des Systems ist nicht viel komplexer:

```
from sklearn.tree import DecisionTreeRegressor  DecisionTreeRegressor: <class 'skle

tree_reg = DecisionTreeRegressor()  DecisionTreeRegressor: <class 'sklearn.tree._cl
tree_reg.fit(housing_prepared, housing_lables)  tree_reg: DecisionTreeRegressor(ccp
housing_predictions = tree_reg.predict(housing_prepared)  tree_reg: DecisionTreeReg
tree_mse = mean_squared_error(housing_lables, housing_predictions)  housing_lables:
tree_rmse = np.sqrt(tree_mse)  tree_mse: 0.0
tree_rmse  tree_rmse: 0.0
```

Abbildung 7.8: Code: Decision Tree Regression

7.5 Durchführung Decision Tree Regression & Random Forest Regression

Auch für diese Alternative bietet scikit-learn eine Implementation:
Random Forest ist genauso zu implementieren, deswegen hier nicht abgebildet.

7.6 Cross-Validation

Beim validieren des Decision-Trees fällt natürlich auf, dass nach dem Abgleich der Ergebnisse ein Root Mean Spuare Error (RMSE) von 0 vorliegt, also jeder Preis auf den Cent genau richtig vorhergesagt wurde. Wie kann das sein? Hier liegt die Vermutung nahe, dass der Algorithmus aufgrund der gelieferten Daten dahingehend trainiert wurde, dass er deutlich overfitted. Mithilfe von SciKit-Learns cross-Validierungs Funktion konnte diese Vermutung einfach kontrolliert werden: Die Funktion cross_val_score bildet aus einem gegebenen Datensatz beliebig viele sogenannte Folds. Danach wird entsprechend oft der ausgewählte Algorithmus mithilfe aller Folds, bis auf einen trainiert und an dem ausgelassenen Fold validiert. So entsteht ein Array mit den pro Fold festgestellten Fehlern und damit kann der durchschnittliche Fehler sowie auch die Standardabweichung bestimmt werden.

7.7 Erster Vergleich

Mithilfe dieser Funktionen lassen sich schon einige wichtige Werte für den Vergleich der Unterschiedlichen Algorithmen finden. Als dritter Algorithmus wurde

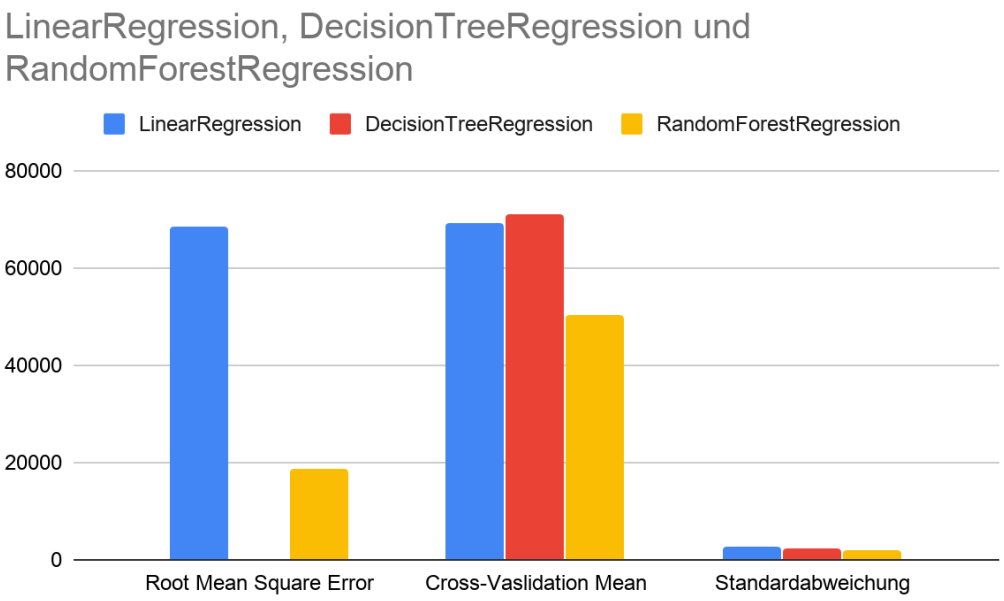

Abbildung 7.9: Statistik: Vergleich verschiedener Algorithmen bei der die y-Achse $ entspricht

RandomForestRegressor ausgewählt und der Übersicht hinzugefügt:
Hier lässt sich erkennen, dass für den Algorithmus RandomForestRegression Der Fehler in der Standardabweichung sowie bei der Cross-Validation am kleinsten war, dieser Algorithmus scheint also bisher die genauesten Ergebnisse zu liefern. Dass der Decision-Tree keinen Fehler bei der RMSE-Berechnung hat, liegt also tatsächlich am Overfitting.

7.8 Evaluation

An dieser Stelle wird nun auf das zum Beginn des Projekts erstellte Test-Set zurück gegriffen. Dieses wird genau wie zuvor mit der full_pipeline vorbereitet, und in Lables und Variablen gesplittet. Mit dem zuvor ausgewählten und trainierten Modell (hier also das RandomForestRegression) wird eine Vorhersage der Median_House_Values erstellt und mit den Lables verglichen. Unter Zuhilfenahme der RMSE-Formal aus Abschnitt 5.3 kann auch hier der RMSE berechnet werden. Das Ergebnis beträgt in dem Experiment 48.464, ein annehmbarer Fehlerwert für die Größenordnung in der wir uns bewegen! Es kann also mithilfe unseren Systems

```
[[-0.32020187  0.68517302  1.68863487  2.73541594  3.85529689  5.27566208
   8.31246632  6.30325389  7.30369092  9.31753875]]
```

Abbildung 7.10: decision_funktion

```
print(cross_val_score(svm_clf,housing_prepared, housing_labels_classes, cv=3, scoring="accuracy"))
```

Abbildung 7.11: Code: cross_val_score 1

der Preis eines Hauses auf circa $50.00 richtig geschätzt werden. Der Algorithmus RandomForestRegression scheint hierbei die beste Wahl zu sein.

7.9 Vergleich mit Classification

Um die verschiedenen Herangehensweisen sinnvoll vergleichen zu können müssen für den Classification-Ansatz zunächst erstmal Klassen definiert werden, in welche die Preise eingeordnet werden sollen. Da der Regression-Ansatz ergeben hat, dass Preise durchschnittlich mit einer Genauigkeit von ± $50.000 bestimmt werden können, sollen hier also Preisklassen von jeweils $50.000 gebildet werden, somit sollten die Häuser gut in eine richtige Preiskategorie eingestuft werden können. Da nur 156 Preise in die Kategorie <$50.000 fallen wird diese mit der nächst höheren zu der Kategorie <$100.000 zusammengefasst. Als Labels für unsere Daten dienen nun also Integers von 1-10, wobei 10 die höchste Preiskategorie von >$500000 repräsentiert. Da nun zwischen 10 verschiedenen Klassen unterschieden werden soll, muss ein mulitclass Classification Algorithmus verwendet werden. Dafür wurde zunächst der Support Vector Machine Algorithmus gewählt. Das Verfahren zum Trainieren der Algorithmen ist analog zu dem vorherigen, nur muss hier auf die Auswahl der Labels als Y-Train-Set geachtet werden. Beim predicten eines Preises wird dann eine sogenannte decision_funktion erstellt, mit deren Hilfe sich ein Objekt zu einer Preisklasse zuordnen lassen können soll. Hier ein Beispiel für solch eine Funktion:
In diesem Beispiel ist klar erkennbar, dass der Wert der die Klasse 10 repräsentiert am höchsten ist. Das Objekt sollte aber eigentlich der Klasse 5 zugeordnet werden. Hier liegt also ein Fehler vor. Um die Präzision genau zu bestimmen wird wieder mit dem cross_val_score gearbeitet.
Die Präzision dieses Algorithmus ist mit 47% nicht besonders gut, weswegen das

```
[0.47329215 0.47020349 0.47202035]
```

Abbildung 7.12: Code: cross_val_score 2

```
Support Vector Machine mit 100er Klasses
[0.64135174 0.63571948 0.64335029]


Random Forest Classifier mit 100er Klassed
[0.51998547 0.52307413 0.50563227]
```

Abbildung 7.13: Ergebnis nach anpassung der Preiskategorien. SVM hat eine Präzision von 64% und SGD 52%

Problem noch einmal mit einem Classification Algorithmus angegangen wird, der sich bereits bewährt hat: Dem SDGClassifier bzw. dem Random Forest Classifier. Mit diesem Ansatz ließ sich jedoch nur ein Präzision von 35% erreichen. Um zu zufriedenstellenden Ergebnissen zu gelangen muss also ein anderer Weg gewählt werden.
Eine sinnvolle Lösung erschien es uns zu sein, die Preiskategorien anzupassen. Im nächsten Anlauf werden nun Klasses von jeweils $100.000 benutzt um das System zu trainieren. Es werden also 6 Kategorien erstellt und als Label-Set definiert, mit denen die Algorithmen erneut trainiert und getestet werden. Die Ergebnisse folgen:

Bei einer Gegenüberstellung der Ergebnisse des Regressions Ansatzes gegenüber dem Classification Ansatzes lässt sich folgendes festhalten:

- Mit der Regression ließ sich ein RMSE von $48.464 erreichen
- Mit der Classification ließ sich mit 64% Präzision eine Einordnung in Klassen von jeweils $100.000 erreichen

7.10 Projektauswertung

Ein direkter Vergleich dieser Werte ist kaum sinnvoll zu vollziehen. Der Grund dafür ist uns während der Durchführung des Projekts immer klarer geworden: Einen direkten Vergleich zwischen Classification- und Regression-Algorithmen ist weder möglich noch sinnvoll. Für jede Problemstellung ist ein anderer Ansatz zu wählen:

In unserem Beispiel liegt ein klarer Fall von Regression vor, denn es soll ein möglichst genauer Preis für das Haus bestimmt werden. Durch einen Workaround wurde versucht das Problem auch für den Classification Ansatz aufzubereiten, mit diesem erzwungenen Ansatz konnten allerdings keine besonders guten Ergebnisse erzielt werden. Der Clustering-Ansatz ist für den Datensatz so unpassend, dass dieser aus unserem Experiment ausgeschlossen wurde. Ohne das Labeln der Daten werden die Häuser wahrscheinlich in Kategorien wie "Entfernung zum Strand" eingeteilt werden, da diese im Datensatz die klarsten Anzeichen für Klassen zeigen.
Jeder Ansatz hat seine Daseinsberechtigung, und dass einige Algorithmen besser abschneiden als andere, kann auch in der Natur der Problemstellung begründet sein. Die ersten und wichtigsten Schritte bei einem Machine-Learning Projekt sollten unsere Meinung also folgende sein:

- Ziel genau definieren
  Im Beispiel: Sollen genaue Preise geschätzt werden? Sollen Preisklassen gebildet werden?

- Visualisierung der Daten
  Hier können schon erste wichtige Erkenntnisse gesammelt werden, die ohne die Veranschaulichung vielleicht übersehen werden können.

- Analyse der Daten
  Können schon ohne Machine-Learning Zusammenhänge erkannt werden? Wenn ja sollten diese, falls sie für das Projekt sinnvoll erscheinen, unbedingt berücksichtigt werden.

- Auswahl der Herangehensweise
  Mit den Informationen die in den ersten Schritten gesammelt worden, sollte es einfach sein sich für die richtige Methode und den passenden Algorithmus zu entscheiden.

Unter Berücksichtigung dieser Gesichtspunkte sollte man meistens zu einer erfolgreichen Algorithmus-Auswahl und damit auch zu einem sinnvollen Projektergebnis gelangen.

8 Resümee

Beim Schreiben dieser Seminararbeit und bei der Durchführung des dazu erdachten Experiments, sahen wir uns immer wieder mit der Vielfältigkeit des Machine-Learnings konfrontiert. Unser ursprünglicher Ansatz, der Vergleich verschiedener Algorithmen aus verschiedenen Ansätzen erwies sich mit dem Voranschreiten des Projekts als immer weniger sinnvoll. Stattdessen haben wir wertvolle Erkenntnisse im Umgang mit Machine Learning Projekten erlangt: Es gibt keine allgemeingültige beste Lösung, sondern es muss genau überlegt werden welcher Ansatz für das vorliegende Problem der richtige ist. SciKit-Learn unterstützt einen dabei mit zahlreichen Funktionen und Möglichkeiten, die es uns ermöglicht haben die Daten einfach aufzubereiten und zu visualisieren. Diese Tools machen es uns relativ leicht das Problem einzuordnen und dementsprechend die korrekte Herangehensweise zu selektieren. Bei unserem Projekt ist der sinnvollste Ansatz die Regression, daraus lassen sich allerdings keinerlei Rückschlüsse auf andere Problematiken ziehen. Mithilfe der gewonnenen Erkenntnisse lässt sich allerdings im Bezug auf das Thema Mobilität eine wichtige Aussage treffen: Ein solch umfassendes Thema wie zum Beispiel Autonomes Fahren muss sich aller Instrumente bedienen um ansatzweise abgedeckt werden zu können. Es folgen einige Exempel um diese Aussage zu verdeutlichen. Als erste Komplexität wollen wir die Erkennung von Straßenschildern nennen. Hier müssen keine unbekannten oder neuen Schilder gedeutet werden, es sind alle Schilder bekannt und können klar gelabelt werden. Hier sollte also unbedingt mit Classification gearbeitet werden um das bestmögliche Ergebnis zu erzielen. Ein Präzedenzfall für Regression wiederum ist die Berechnung der besten Geschwindigkeit, bei der die Gefährdung minimal ist aber auch der Verkehrsfluss nicht beeinträchtigt wird. Diese ist abhängig vom Abstand zum vorherfahrenden Automobil, der zulässigen Geschwindigkeit, dem Zustand der Fahrbahn und vielen weiteren Faktoren. Aufgrund all dieser Daten muss ein genauer Wert festgelegt

werden, es wird also klar, dass Regression hier die sinnvollste Methodik ist. All diese Erkenntnisse konnten wir vor allem mithilfe unseres Projekts erlangen. Dieses hat zwar nicht zu den Ergebnissen geführt, die wir uns anfangs erhofft haben, hat dafür aber zu einem tieferen Verständnis für Deep-Learning und dessen Diversität in Ansätzen und Problemen geführt und war deswegen für uns von allergrößter Bedeutung und Nutzen.

Literaturverzeichnis

[1] M. Bowles, *Machine Learning with Spark and Python Essentia Techniques For Predictive Analytics*. John Wiley & Sons, Inc., 2020.

[2] W.-M. Lee, *Python Machine Learning*. John Wiley & Sons, Inc., 2019.

[3] "SciKitLearn," https://scikit-learn.org/stable/index.html, accessed: 2020-02-02.

[4] "Medium random forest," https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d, accessed: 2020-02-02.

[5] "SciKitLearn tree," https://scikit-learn.org/stable/modules/tree.html, accessed: 2020-02-02.

[6] A. Géron, *Hands on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2019.

[7] "Dataset california housing prices," https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/housing/housing.tgz, accessed: 2020-01-02.