BERICHT

Vorhersage von Überleben und Tod beim Titanic-Unglück

Machine Learning Abschluss-Projekt

Autoren:

xxx, xxx, xxx, Manuela Hebel

Projektzeitraum: 8.12. - 10.12.2020

1. Aufgabenstellung & Zielsetzung

Zur Verfügung statt der Titanic-Datensatz. Dieser beinhaltet verschiedene Daten, z.B. Passagiernamen, Geschlecht des Passagiers, Ticketpreis, usw. sowie die Angabe, ob der entsprechende Mitreisende das Schiffsunglück überlebte oder nicht.

Um einen möglichst aktuellen Anwendungsfall eines Machine Learning-Projekts (Abk.: ML) zu simulieren gehen wir davon aus, dass auch künftig Menschen mit der Titanic reisen und das Unglück noch nicht so weit zurück liegt. Wir nehmen also aktuelle Datensätze an und dass die gleichen Gesetzmäßigkeiten und Zusammenhänge wie damals beim Unglück (z. B. wie stark die Ticket Klasse den sozioökonomischen Status vorhersagt) gelten. Wir möchten ein ML-Modell entwickeln, das auf Basis der vorliegenden Daten vorhersagt, ob jemand bei einem erneuten Unglück überleben oder sterben würde. Für diesen Fall ist es natürlich enorm wichtig, dass die Vorhersage sehr genau mit dem tatsächlichen späteren Eintreten übereinstimmt.

Hierzu sind wir wie folgt vorgegangen:

2. Daten

Dataexploration (Herkunft)

Zunächst wurde die Dokumentation des Datensatzes recherchiert, um die Kodierung zu verstehen (z.B. 1 = survived, 0 = dead, usw.)

Tab. 1: Kaggle Wettbewerb Dokumentation

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Analyse mit Jupyter Notebook

Nach dem Einlesen beider Datensätze stellte sich heraus, dass sich im sog. Test-Datensatz keine y-Werte befinden, weshalb es für supervised learning Algorithmen unbrauchbar ist.

Die Missing-Analyse ergab 3 Features mit fehlenden Werten: Age, Cabin und Embarked.

- Age (177 Missings)
- Cabin (687 Missings)
- Embarked (2 Missings)

Age und Embarked wurden in der Pipeline mit einem Imputer behandelt.

Die Spalte Sex wurde umkodiert in eine binomiale Varible mit 0 und 1. Die Spalten PassangerID, Name, Ticket und Cabin wurden gelöscht. Cabin aufgrund der hohen Anzahl fehlender Werte, die restlichen, da wir glauben, dass sie keinen Vorhersagemehrwert liefern und unsere Analysen deutlich verlangsamen würden.

Überlegungen zu neuen Spalten, von denen aus Zeitgründen abgesehen wurde:

Die Spalten SibSp und Parch enthalten die Anzahl an Siblings/Spouses sowie Parents/Children. Die Aggregation der Spalten SibSp + Parch entspräche der Anzahl der verwandten Mitreisenden. Von dieser Aggregation sehen wir ab, da hier die Information verloren ginge, ob es sich bei den Mitreisenden um Mitreisende einer anderen Generation (Parch) oder derselben Generation (SibSp) handelt. Diese könnten bei der Vorhersage der Überlebensrate relevant sein (z.B. bevorzugte Rettung von Eltern mit Kindern). Vorhersagemodelle wie bspw. die Regression sind jedoch in der Lage auch die Interkation dieser beiden Variablen als Vorhersagewert mit einzubeziehen. Zusätzlich eine aggregierte Spalte hinzuzunehmen bietet aufgrund hoher Interkorrelationen aus statistischer Sicht keinen Mehrwert, sondern verlangsamt vielmehr die Modelle aufgrund zusätzlicher redundanter Spalten.

Aus der Spalte Name könnten folgende Information via string_split einfach extrahiert werden:

- Nachname
- Titel (Mr., Mrs., Miss., Master., Dr.))

Hierüber könnten Rückschlüsse gezogen werden, ob es sich in der Spalte Parch um Kinder oder Eltern handelt und ob es sich um ein Ehepaar oder Geschwister handelt (bei gleichem Nachnamen und SibSp. >0).

Anhand der gemeinsamen Nachnamen und/oder Ticketnummern (welche über die Familienmitglieder hinweg gleich zu sein scheinen) könnten zudem Familien-Cluster gebildet werden, um zu ermitteln, ob große Familienverbände höhere oder niedrigere Überlebenschancen hatten.

Der Count der jeweils gleichen Ticket-Nummern könnten ebenfalls Auskunft über die Gruppengröße der gemeinsam Reisenden geben. Hierzu wäre es jedoch nötig, den Test-Datensatz mit einzubeziehen, da sich scheinbar dieselben Ticketnummern in beiden Datensätzen häufiger befinden.

Data Visualization:

Die Visualisierung der Daten ergab folgende Erkenntnisse:

1. es reisten mehr Männer als Frauen mit
2. Frauen hatten
3. Es gibt 3 vermeintliche Ausreißer im Alter von 35-36 Jahren, die ein besonders teures Ticket gekauft haben. Bei genauerer Betrachtung des Features Fare fiel jedoch auf, dass die Ticket-Fare keinen Pro-Kopf-Ticket-Preis angibt, sondern ein Aggregat über alle Mitreisenden dergleichen Ticketnummer. Manuelles Überschlagen der Daten ergab, dass es sich bei den vermeintlichen Ausreißern um plausible Werte handelt, weshalb sie beibehalten wurden.

Aufgrund der Erkenntnis, dass in Fare ein Aggregat mehreren Variablen darstellt (Anzahl der Mitreisenden, Pro-Kopf-Preis, ggf. weitere) und dass diese ein aufwändigeres Data Preprocessing erfordern würde, wurde das Feature angesichts der limitierten Zeit für die Analyse entfernt.

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	0	3	0	22.0	1	0	S
1	1	1	1	38.0	1	0	C
2	1	3	1	26.0	0	0	S
3	1	1	1	35.0	1	0	S
4	0	3	0	35.0	0	0	S
...	...	...	...	...	...	...	...
886	0	2	0	27.0	0	0	S
887	1	1	1	19.0	0	0	S
888	0	3	1	NaN	1	2	S
889	1	1	0	26.0	0	0	C
890	0	3	0	32.0	0	0	Q

891 rows × 7 columns

Abb. 1: Fertiger Arbeitsdatensatz (full.csv)

3. Vorgehensweise

Methode

Zur Vorgehensweise entschieden wir uns für eine Pipeline bestehend aus mehreren Abschnitten. Grundsätzlich wäre es möglich, alle Einzelschritte (steps) einer Instanz der Pipeline-Klasse zu übergeben. Da wir es jedoch mit kategorialen und metrischen Daten zu tun hatten, musste eine horizontale Splittung der Pipeline für die Steps Imputing, HotEncoding und Scaling erfolgen, die für die PCA Analyse wieder zusammengeführt wurden.

```
Pipeline(steps=[('preprocessor',
                 Pipeline(steps=[('data_transformer',
                                  ColumnTransformer(transformers=[('numerical',
                                                                   Pipeline(steps=[('imputer',
                                                                                    SimpleImputer()),
                                                                                   ('scaler',
                                                                                    MinMaxScaler())]),
                                                                   ['Age',
                                                                    'SibSp',
                                                                    'Parch',
                                                                    'Sex']),
                                                                  ('categorical',
                                                                   Pipeline(steps=[('imputer',
                                                                                    SimpleImputer(strategy='constant')),
                                                                                   ('encoder',
                                                                                    OneHotEncoder(handle_unknown='ignore'))]),
                                                                   ['Pclass',
                                                                    'Embarked'])])),
                                 ('reduce_dim', PCA(n_components=5))])),
                ('classifier',
                 LogisticRegression(C=0.05, max_iter=20, random_state=42))])
```

Abb. 4: Darstellung der Pipeline des finalen Modells unter Nutzung der Logistischen Regression

Die Pipeline lässt sich wie folgt am Beispiel der LogisticRegression darstellen und wurde als interaktiver html code abgelegt:

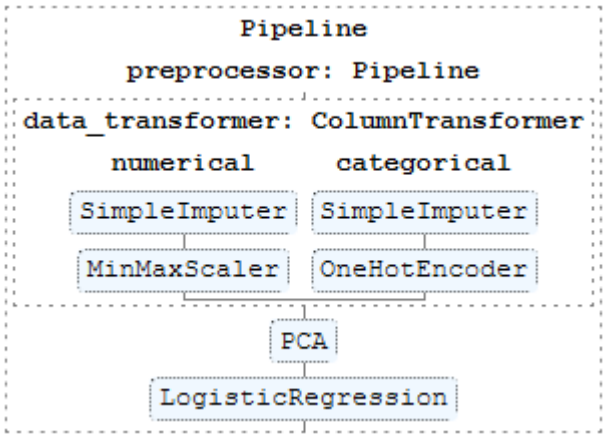


Abb. 2: Übersichtsbeispiel für die Pipeline der LogisticRegresssion

Um die Pipeline flexibel für die Anwendung mehrerer Algorithmen zu gestalten, wurde ein einheitlicher Pipeline-Abschnitt mit einem gemeinsamen Parameter-Space für alle später angewandten Algorithmen erstellt. Dadurch musste später lediglich eine neuer Pipeline-*Abschnitt* für jeden weiteren ML Algorithmus erstellt werden.

Diese Pipeline wurde dann mit verschiedenen Parametern für jeden ML Algorithmus über eine GridSearch getestet, um die optimale Parameterkombination für jeden ML Algorithmus zu identifizieren.

Die Parameter der Vorverarbeitung wurden um die jeweiligen Parameter des GridSearch der zu analysierenden Methode erweitert, z.B.:

```
param_grid_SVC = {**param_grid, **param_grid_SVC}
```

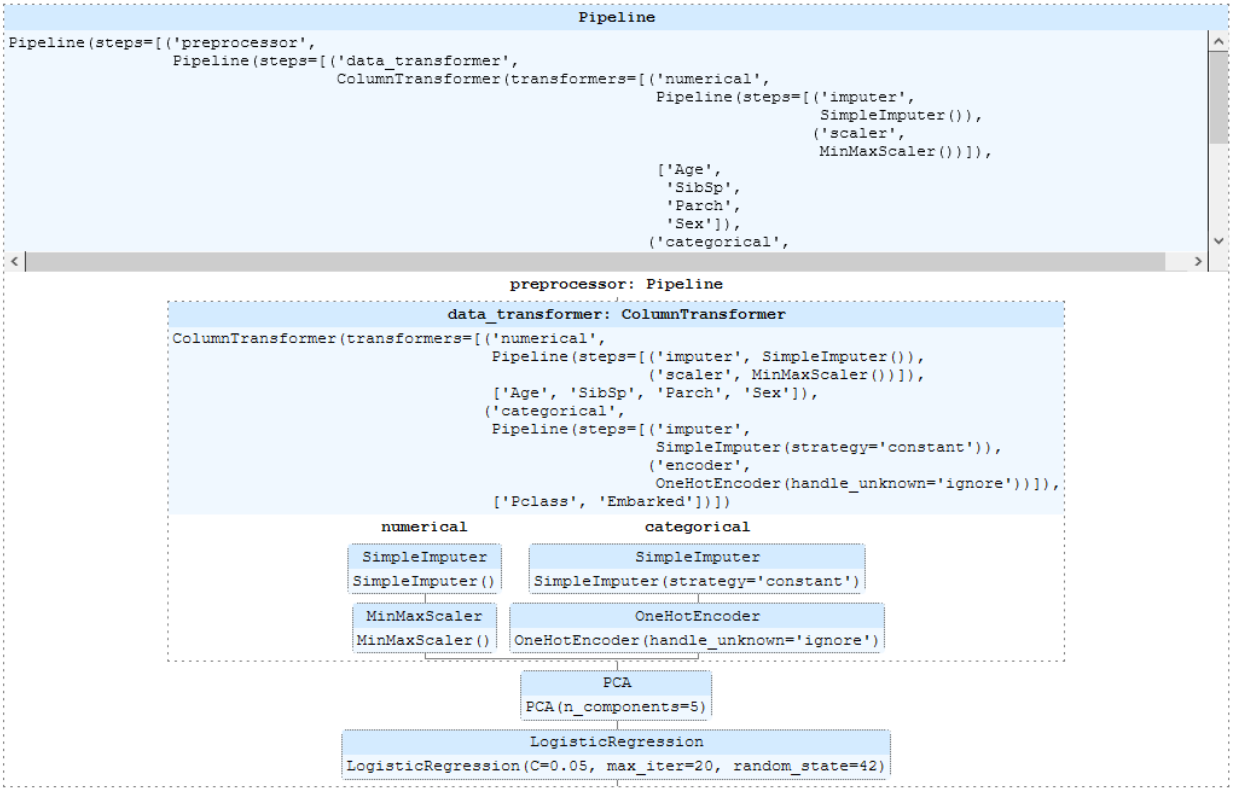


Abb. 3: Detailansicht für LogisticRegression

(Alle HTML-Dateien befinden sich zudem im Projektabgabeordner)

Damit war es möglich, den nachfolgenden Code für alle untersuchten Methoden für das Training, das Predict und die Auswertung gleich zu halten. Aufgrund der begrenzten Zeit wurden keine Klassen verwendet, sondern ein eigener Name für jedes Model (s. auch Ausblick).

Wahl der Parameter

Die Wahl der Parameter erfolgte gemeinsam durch Sicht und Diskussion der einzelnen Parameter in der Gruppe anhand der Dokumentation in scikit learn (https://scikit-learn.org). Anschließend wurde die Multiplizität berechnet und ggfs. Einschränkungen vorgenommen.

Messung, Abschätzung der Laufzeiten

Da wir zusätzlich die Laufzeit berücksichtigten, wurde ggfs. Eine weitere Einschränkung vorgenommen, wenn nach Start der Gridsearch absehbar war, dass die Laufzeit nicht mit dem Ablauf des Projektes zeitlich vereinbar war und aufgrund der begrenzten Zeit gekürzt werden musste.

Tuning und Anpassung der Parameter (Randbereiche)

Je nach Ergebnis der GridsSearch wurden die besten Parameter mit dem Range der Parameter verglichen und Randlagen ggfs. erweitert oder Zwischenwerte für die Parameter hinzugefügt und ein zweiter Lauf vorgenommen.

Dumpen der Modelle

Um die Modelle auch zu einem späteren Zeitpunkt nutzen zu können bzw. diese zur Erstellung unserer Vorhersageoberfläche nutzbar zu machen, wurden alle Modelle unter Nutzung der Library *joblib* „gedumped“ (abgespeichert).

4. Auswahl der Modelle

Folgende singuläre Modelle wurden für die Analyse mit GridSearch ausgewählt und näher betrachtet

- Logistische Regression
- Support Vector Machine
- Neuronales Netz
- Decision Tree

Ausgeschlossen in der Betrachtung wurde ein Bayes Klassifikator, weil wir sowohl kategoriale Daten als auch kontinuierliche haben und in scikit Learn hierzu kein Model direkt verwendet werden kann.

Ein Versuch, dies mit durch den Import eines gemischten Modell vom (from mixed_naive_bayes import MixedNB) scheiterte leider an der verfügbaren Zeit zur Implementierung, weil wir auf technische Schwierigkeiten gestoßen sind und keine Dokumentation zu diesem Package auf der Seite des Autors vorhanden war.

Folgende Modelle wurde als Ensemble getestet:

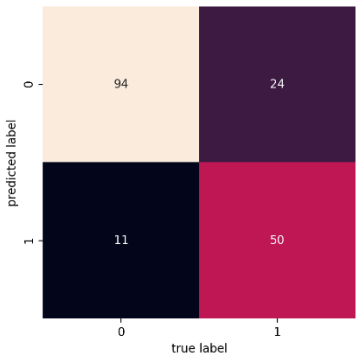
- AdaBoost
- Bagging
- Soft Voting

5. Results

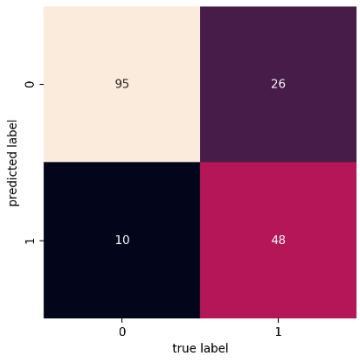
Confusion Matrix

Insgesamt waren alle Modelle im Ergebnis recht ähnlich, so dass es keine klaren Favoriten gab.

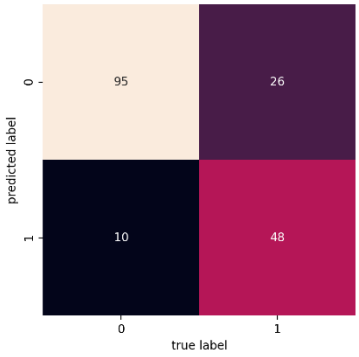
a)



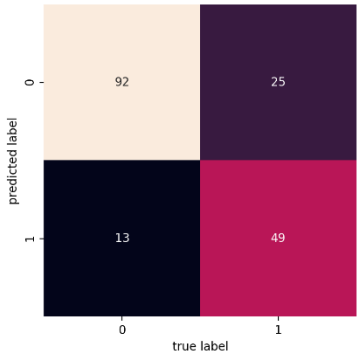
b)



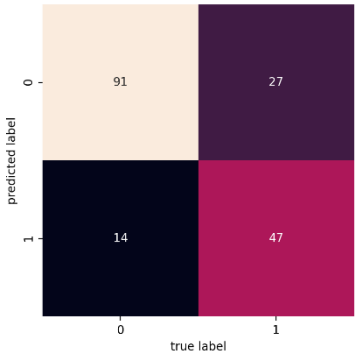
c)



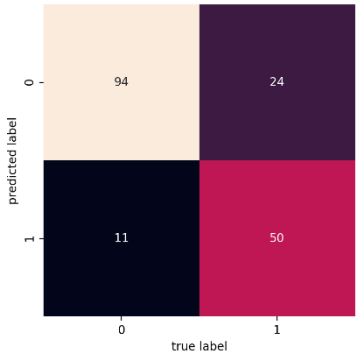
d)



e)



f)



g)

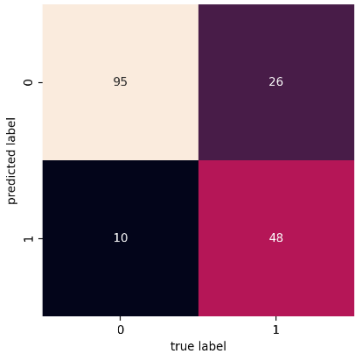


Abb. 5: Konfusionsmatrizen für a) Logistische Regression, b) Support Vector Machine, c) Neuronales Netz, d) Decision Tree, e) AdaBoost, f) Bagging und g) Soft Voting

Classification Reports

Tab. 2: Classification Reports der verschiedenen Modelle

Modell	class/average	precision	recall	f1-score	support
Logistische Regression	0	0,78	0,9	0,84	105
	1	0,82	0,64	0,72	74
	accuracy			0,79	179
	macro average	0,8	0,77	0,78	179
	weighted average	0,8	0,79	0,79	179
Support Vector Machine	0	0,79	0,9	0,84	105
	1	0,83	0,65	0,73	74
	accuracy			0,8	179
	macro average	0,81	0,78	0,78	179
	weighted average	0,8	0,8	0,79	179
Neuronales Netz	0	0,79	0,9	0,84	105
	1	0,83	0,65	0,73	74
	accuracy			0,8	179
	macro average	0,81	0,78	0,78	179
	weighted average	0,8	0,8	0,79	179
Decision Tree	0	0,79	0,88	0,83	105
	1	0,79	0,66	0,72	74
	accuracy			0,79	179
	macro average	0,79	0,77	0,77	179
	weighted average	0,79	0,79	0,78	179
AdaBoost	0	0,77	0,87	0,82	105
	1	0,77	0,64	0,7	74
	accuracy			0,77	179
	macro average	0,77	0,75	0,76	179
	weighted average	0,77	0,77	0,77	179
Bagging	0	0,8	0,9	0,84	105
	1	0,82	0,68	0,74	74
	accuracy			0,8	179
	macro average	0,81	0,79	0,79	179
	weighted average	0,81	0,8	0,8	179
Soft Voting	0	0,79	0,9	0,84	105
	1	0,83	0,65	0,73	74
	accuracy			0,8	179
	macro average	0,81	0,78	0,78	179
	weighted average	0,8	0,8	0,79	179

Jobs und Laufzeiten

Tabelle 3: Gesamtlaufzeit, Anzahl der verarbeiteten Jobs und Laufzeit pro Job für die getesteten Algorithmen und Methoden

Algorithmus/Methode	Runtime in min	Anzahl jobs	ms pro job
Logistische Regression	3.33	9720	20.54
Support Vector Machine	4.45	15552	17.17
Decision Tree	8.84	23328	22.74
Neuronales Netz	39.35	15552	151.83
AdaBoost	1.69	648	156.87
Bagging	13.10	1728	454.76
Soft Voting	176.26	41472	255.01

Bewertung

Die Modelle liefern eine vergleichbare Klassifizierung und die Confusionsmatrix ist sehr ähnlich.
Dementsprechend bringen Ensemble Methoden auch nur eine leichte Verbesserung.
Neuronale Netze brauchen lange und liefern aber kein wirklich besseres Ergebnis, ansonsten sind die Laufzeiten der anderen singulären Modelle vergleichbar und auch die Runtime von AdaBoost und Bagging sind aktzeptabel.

Der Versuch über Voting mit den besten Modellen (SVC, Bagging) mit neuer Variation der Parameter lieferte aufgrund der hohen Ähnlichkeit keinen wirklichen Vorteil, benötigte aber entsprechend viel Zeit und lief über Nacht.

6. Oberfläche für Predicitions

Wir entwickelten zudem eine Oberfläche, über die sich künftige Titanic Passagiere vorhersagen lassen können, ob sie zu den Opfern oder den Überlebenden gehören werden.

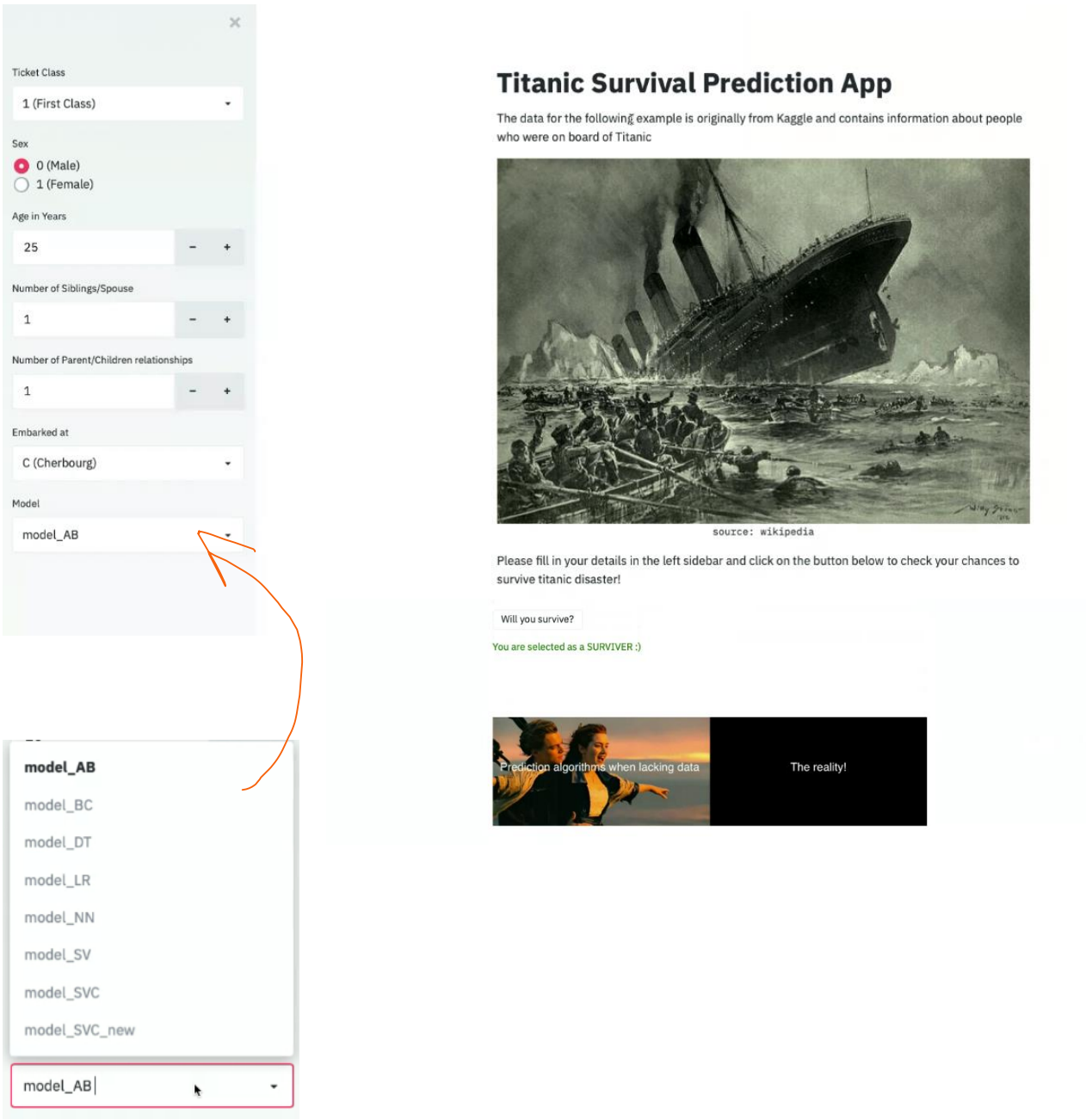


Abb. 5: Oberfläche, entwickelt über das speziell für Machine Learning und Data Science entwickelte Open Source App Framework „Streamlit“

Die Eingabe unserer persönlichen Daten in die Vorhersage-Oberfläche ergab folgendes Ergebnis:

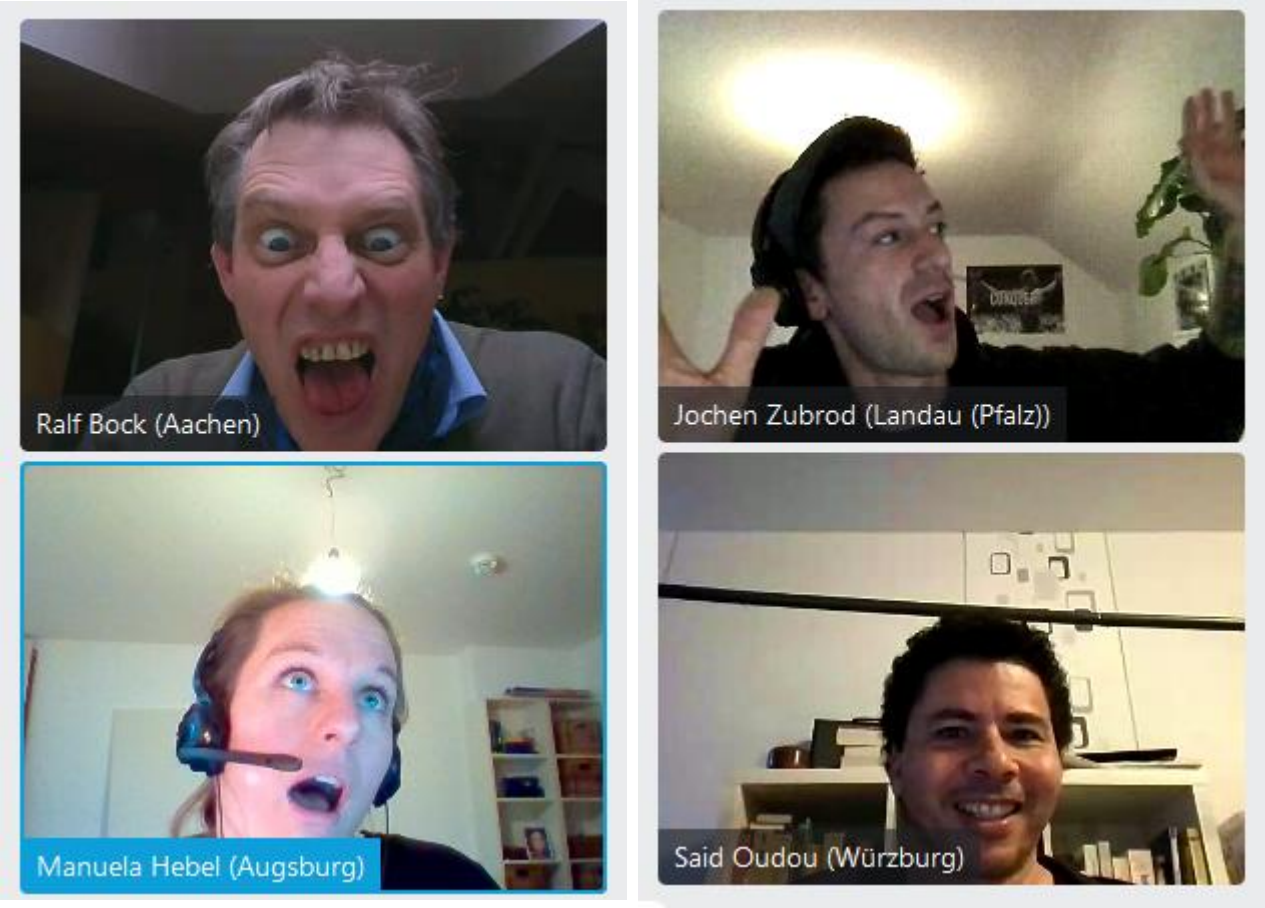


Abb. 6: Reaktionen auf die Modell-Vorhersagen. Aufgrund der Vorhersage scheint Manuela das Erbe dieser Projektarbeit alleine weitertragen müssen…

7. Ausblick

Refactoring

Ein sinnvoller nächster Schritt wäre ein refactoring des Codes. Insbesondere könnten die Modelle z.B. als Liste mit Modelnamen, Model und Parametern definiert werden, z.B.:

```
### Definiere Listen für die verschiedenen Modelle
lmodelname = ['Logistic Regression', 'SVC','Decision Tree','Neural Network',
lmodel = [ LogisticRegression(), SVC(),DecisionTreeClassifier, MLPClassifier(
lparamsp = [
{
    "classifier__penalty": ["l1", "l2", None],
    "classifier__solver": ["lbfgs", "liblinear", "sag", "saga", "newton-cg"],
    "classifier__C": [0.05, 0.1,0.5],
    "classifier__class_weight": ["balanced", None],
    "classifier__max_iter": [20, 25, 30]
    },
{
    'classifier__kernel': ["linear", "poly", "rbf"],
    'classifier__probability': [False, True],
    'classifier__C': [10.0, 1.0, 0.1],
    'classifier__degree': [2, 3, 4],
    'classifier__class_weight': [None, "balanced"],
    'classifier__gamma': ["scale", "auto", 0.0001, 1]
    },
```

Abb. 7: Liste mit Modelnamen, Model und Parametern

Es kann eine Child-Class der Klasse GridSearch mit zusätzlichen eigenen Methoden erstellt und ausgelagert werden, wodurch sich das Hauptprogramm stark vereinfacht und übersichtlicher gestalten lässt.

```
48  ################  Klassen
49  from sklearn.model_selection import GridSearchCV
50  from sklearn.metrics import classification_report
51  from sklearn.metrics import confusion_matrix
52
53  class myGridSearchCV(GridSearchCV):
54      def __init__(self,modelname, model, parameter_space, n_jobs, cv):
55          self.modelname = modelname
56          # super().__init__()
57          self.parameter_space = parameter_space
58          self.n_jobs = n_jobs
59          self.cv = cv
60          self.model = GridSearchCV(model, parameter_space, n_jobs=n_jobs, cv=cv)
61          print(f"Class myGridSearchCV initialized for model {modelname}")
62
63      def fit(self,X,Y):
64          self.model.fit(X,Y)
65
66      def predict(self,Y):
67          yP = self.model.predict(Y)
68          return yP
69
70      def print_model_param(self,verbose=False):
71          md = self.model
72          print(f'Model {self.modelname} Best parameters found:\n', md.best_params_, md.best_score_)
73          print("Mean Score: %0.3f (+/-%0.03f) std" % (md.cv_results_['mean_test_score'][md.best_index_],
74                                                       md.cv_results_['std_test_score'][md.best_index_]))
```

Abb. 8: Bildung einer Child-Class aus der Klasse GridSearchCV

Über eine Schleife im Hauptprogramm könnten die verschiedenen GridSearches sukkzessive durchlaufen werden.

```
l_mapr = []    # Ergebnisliste für accurracy, precision und recall
l_model = []   # Liste der Modelle

for i,mname in enumerate(lmodelname):
    model = myGridSearchCV(mname, lmodel[i], lparamsp[i], ljobs[i], lcv[i])
    model.fit(X_train, y_train)
    model.print_model_param()
    # model.print_model_param(verbose=True)
    y_pred = model.predict(X_test)
    a,p,r = model.get_scores(y_test, y_pred, mname, verbose=True)
    l_mapr.append({'modelname' : mname, 'accuracy' : a, 'precision' : p, 'recall' : r})
    l_model.append({'modelname': mname, 'model' : model})
    #
```

Abb. 9: Schleife im Hauptprogramm zum sukzessiven Durchlaufen der verschiedenen GridSearches

Komplexe Datenaufbereitung

Sinnvoll wäre es auch, nach diesem ersten Ergebnis die Datenaufbereitung komplexer zu gestalten.

So lassen sich z.B. aus dem Namen der Passagiere über die Schlüsselworte („Master“ und „Miss“) in Kombination mit der Anzahl der Eltern bzw. Kindern identifizieren, ob es sich um ein Kind oder einen Erwachsenen handelt, wodurch auch Fehlklassifikationen von „unverheirateten“ Frauen vermieden werden, wenn diese den Titel „Miss“ trugen.

Diese Information (Kind/Erwachsen) ist natürlich stark mit dem Alter korreliert und würde ggfs. durch eine PCA entfernt, könnte allerdings die fehlenden Werte für das Alter besser ergänzen (Impute).

In jedem Fall waren die Überlebenschancen für Kinder durch die übliche Vorgehensweise bei der Rettung von Passagieren („Frauen und Kinder zuerst“) höher als für Erwachsene wie folgende Auswertung zeigt:

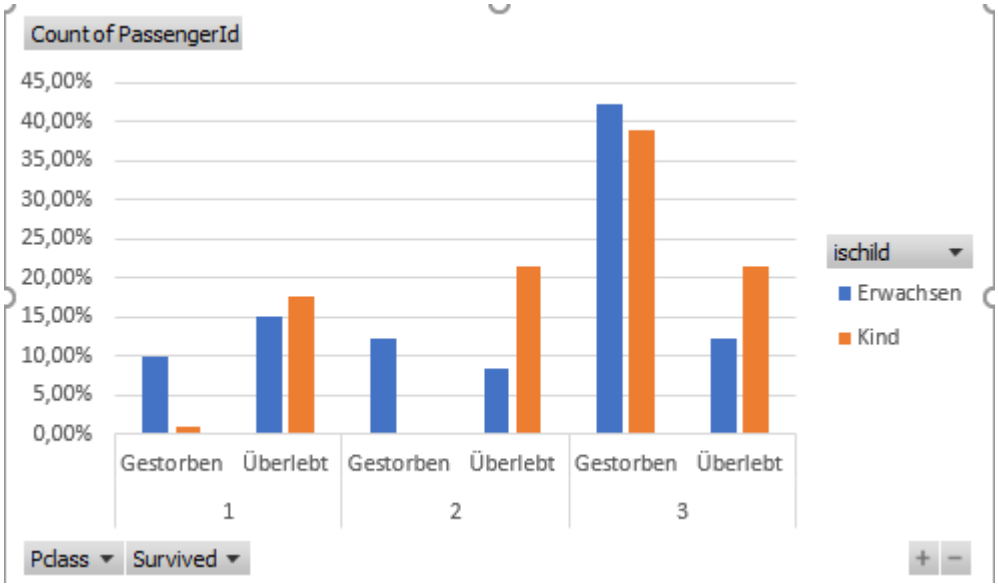


Abb. 10: Überlebenschance für Kinder und Erwachsene nach Price Class

Bekannt sind („Domainwissen“) dass die Tickets der ersten Klasse ab 150$, der zweiten Klasse ab 60$ und der dritten Klasse ab 36$ zu haben waren und dass Kinder (<12 Jahre) z.B. 15$ in der dritten Klasse zu zahlen hatten. (Quelle: https://rms-titanic.fandom.com/de/wiki/Fahrpreise)

Eine erste Auswertung der Trainingsdaten zeigt dies aber nicht:

Average of Fare	Column Labels			
Row Labels	1	2	3	Grand Total
C	104,7185294	25,35833529	11,21408333	59,95414405
Q	90	12,35	11,18339306	13,27602987
S	70,3648622	20,32743902	14,644083	27,0798118
Grand Total	**84,19351636**	**20,66218315**	**13,6755501**	**32,09668088**

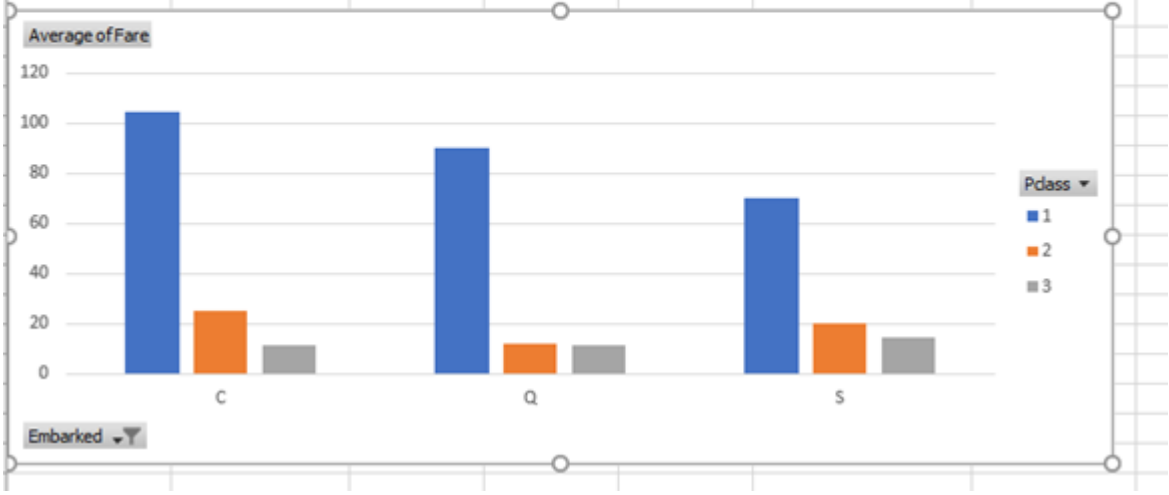


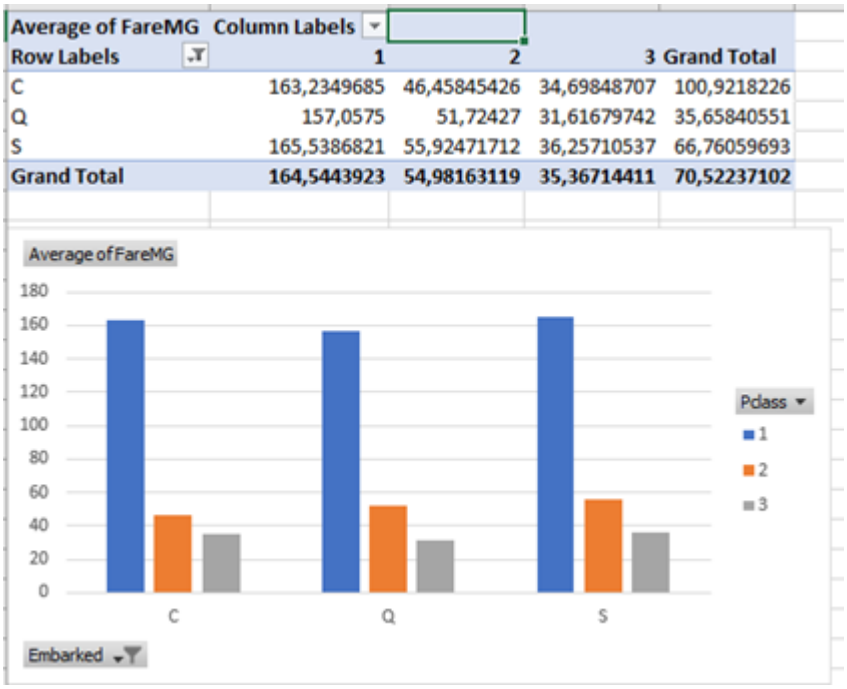
Abb. 11: Ticket-Preis in Abhängigkeit vom Auslaufhafen und der Preis-Klasse
Hinweis: die Abfahrthäfen sind leicht wie folgt zu identifizieren: S = Southhampton, C=Cherbourg, Q=Queenstown)

Abb. 12: Route der Titanic (Quelle: http://www.titanic-stories.de/pages/route.html)

Ebenso müssen die damals üblichen Wechselkurse von Britischen Pfund zu US $ (1 Pfund = 4,87 $) und das irische Pfund = 0.87 Britisches Pfund berücksichtigt werden. Weitere Informationen bieten die TicketID, die in Kombination mit dem Abfahrthafen und Passagierklassen genutzt werden kann, die FARE auf US $ zu normieren

zumal manche Tickets offenbar direkt in US $ bezahlt wurden. Darüber hinaus muss berücksichtigt werden, dass die Preise offenbar pro Ticket sind und dann auf die Gruppengröße unter Beachtung der Kinderanzahl umgerechnet werden muss.

Hierdurch ließe sich durch eine komplexe Transformation der Eingangsdaten eine normierte Fare berechnen, die mit der historisch bekannten Information weitestgehend in Deckung sind:



Average of FareMG	Column Labels			
Row Labels	1	2	3	Grand Total
C	163,2349685	46,45845426	34,69848707	100,9218226
Q	157,0575	51,72427	31,61679742	35,65840551
S	165,5386821	55,92471712	36,25710537	66,76059693
Grand Total	**164,5443923**	**54,98163119**	**35,36714411**	**70,52237102**

Abb. 13: Normierter durchschnittlicher Ticket-Preis in Abhängigkeit vom Auslaufhafen und der Preis-Klasse

Mit diesen Daten haben wir dann versuchsweise den zuvor erfolgreichsten Algorithmus (Support Vector Machine) erneut trainiert:

Tab. 4: Classification Report des erweiterten Modells

Modell	class/average	precision	recall	f1-score	support
Support Vector Machine	0	0,81	0,9	0,86	105
improved	1	0,84	0,7	0,76	74
	accuracy			0,82	179
	macro average	0,83	0,8	0,81	179
	weighted average	0,82	0,82	0,82	179

Das erweiterte Modell führte in der Tat zu einer Verbesserung der Vorhersage im Bereich der richtig vorhergesagten Überlebenden (s. Abb. 13). Das zeigt sich auch in den in Tab. 4 aufgeführten Scores.

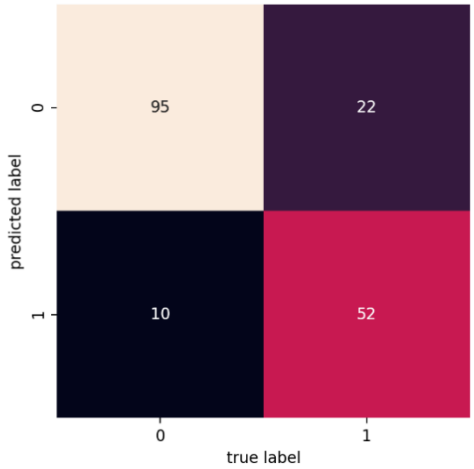


Abb. 14: Konfusions-Matrix des erweiterten Modells

Einbeziehung der separierten Testdaten

Da die Daten aus einem Wettbewerb stammen, waren die Zielwerte aus dem für die Beurteilung des Wettbewerbs verwendeten Testdaten entfernt. Diese wurden von uns nicht benutzt, da wir nur überwachte Verfahren betrachtet haben.
Diese Daten könnten zu zwei Zwecken zur Verbesserung herangezogen werden:

a. Einsatz von unüberwachtem Lernen / Clustering
b. Ergänzung und Konsolidierung der Trainingsdaten, da durch die Separation inkonsistente Daten entstanden sind. (z.B. Anzahl der Mitreisenden im Vergleich zu den Namenslisten)

Sequentielle Kombination von Verfahren

Je nach Zielrichtung (s. Aufgabenstellung) macht es Sinn Verfahren zu kombinieren, die z.B. im ersten Schritt einen hohen Recall erzielen und andere, die eine hohe Precision haben. In unserem Fall waren die Ergebnisse allerdings für alle Modelle zu ähnlich.

8. Lessons learned

Zeitdauer für einzelne Schritte

Wir hatten insgesamt nicht ganz 3 Tage zur Durchführung und Dokumentation des Projekts.

Am ersten Tag für die Analyse der Daten, die Datenvorbereitung und das Aufsetzen der Codestruktur und Pipeline benötigt. Der zweite Tag wurde für die Anwendung der verschiedenen Modellen und der Bestimmung der optimalen Parameter verwendet, so dass die Nacht für den Lauf des Voting-Modells benutzt werden konnte.

Der verbliebene dritte Tag wurde verwendet für die Zusammenstellung und Dokumentation der Ergebnisse, den Aufbaus und Einsatzes einer Oberfläche zur Darstellung des Predict und eines Versuchs der Klassifizierung mit einer Support Vektor Machine mit erweiterten Daten.

Datenvorverarbeitung

Das Verständnis der Daten wächst mit der Analyse und das Vorgehen ist ein iterativer Prozess.

Aufgrund der medialen Bekanntheit des Beispiels waren die Daten intuitiv gut erfassbar und die Datenmenge sehr übersichtlich hinsichtlich Anzahl und Speicherbedarf und sehr gut zu bearbeiten.

Vermutlich macht es Sinn, die im ersten Schritt ausgeschlossenen Daten zu Erzielung eines noch besseren Ergebnisses zu benutzen, da hier noch ungenutzte Information vorliegt.

Dies erfordert jedoch deutlich mehr Zeit und eine intensive Analyse der Daten.

Künftige Empfehlungen

Die Wahl der Modelle ist durch die Verwendung von GridSearch komfortabel durchzuführen aber stark durch die Rechnerleistung und die zur Verfügung stehende Zeit begrenzt. Insbesondere für die Multiplizität der verschiedenen Paramater schnell zu einem zu großen Resourcenbedarf. Hierzu könnten künftig Cloud-Dienste eingebunden werden wie bspw. AWS.

Außerdem scheint eine Kollaboration über Plattformen wie Github sinnvoll.