| | # # Berechnet die Konfusionsmatrix conf_mat = confusion_matrix(y_true, y_pred) |
|------|--|
| | # # Überführung in einen DataFrame für eine bessere Übersichtlichkeit df_index = pd.MultiIndex.from_tuples([('Wahrer Wert', cn) for cn in class_names]) |
| | <pre>dr_index = pd.MultiIndex.from_tuples([('Prognose des Modells', cn) for cn in class_names]) df_conf_mat = pd.DataFrame(conf_mat, index=df_index, columns=df_cols)</pre> |
| | return df_conf_mat, classification_report(y_true, y_pred) |
| | 1.2.2 Funktion zur Betrachtung der ROC-Kurve # Definition einer Funktion, welche auf Basis eines gegeben Modells und zweier zusammengehöriger |
| | # DataFrames die receiver operating characteristic curve (ROC-Curve) visualisiert # # Argumente: |
| | # - X: DataFrame auf welchem die Prognose durchgefürt werden soll (ohne die Zielgröße) # - y_true: Zum DataFrame X gehörige Werte der Zielgröße # - model: Modell auf Basis dessen die ROC-Curve berechnet werden soll # |
| | <pre>def roc_curve_func(X, y_true, model):</pre> |
| | # # Berechnung FPR, TPR und AUC auf Basis des Modells y_score = model.predict_proba(X)[:,1] |
| | <pre>FPR, TPR, _ = roc_curve(y_true, y_score) AUC = auc(FPR, TPR)</pre> |
| | # # Darstellung als Grafik plt.figure() |
| | <pre>plt.plot(FPR, TPR, color='red', lw=2, label='ROC-Kurve (AUC = %0.5f)' % AUC) plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='') plt.xlim([-0.005, 1.0]) plt.ylim([0.0, 1.05])</pre> |
| | plt.ylabel('False Positive Rate') plt.ylabel('True Positive Rate') plt.title('ROC-Kurve') |
| | plt.legend(loc="lower right") plt.show() |
| | 1.2.3 Funktion für das Submitten |
| | # Definition einer Funktion, welche das Submitten der Prognose auf dem Testdatensatz erleichert # # Argumente: |
| | # - model: Modell auf Basis dessen die Prognose erfolgt # - c: #> Wenn None, dann wird die Prognose ohne die Berücksichtung von c vorgenommen # - None dann wird Prognose ohne die Berücksichtung von c vorgenommen |
| | #> Wenn != None, dann wird Prognose mit der Berücksichtung von c vorgenommen # - save: #> Wenn False, dann werden die prognostizierten Daten nicht gespeichert #> Wenn True dann werden die prognostizierten Daten nicht gespeichert |
| | #> Wenn True, dann werden die prognostizierten Daten als .csv gespeichert # - manu_name: #> Wenn None, dann wird ein nicht eindeutiger Standardname als Bezeichnung der .csv gewählt #> Wenn None, dann wird die au eneighemnde .csv mit einem timesterm wensehen |
| | #> Wenn != None, dann wird die zu speichernde .csv mit einem timestamp versehen # def submit(model, c=None, save=False, manu_name=False): |
| | # # Testdatensatz einlesen |
| | <pre>X_test = pd.read_csv('test.csv', index_col=0) #</pre> |
| | # Prognosewerte auf Sensordaten des Testdatensatzes und unter # Berücksichtung von c erzeugen if c != None: |
| | <pre>predicted_test = (model.predict_proba(X_test) >= c)[:,1].astype(int) # # Prognosewerte auf Sensordaten des Testdatensatzes erzeugen</pre> |
| | # Prognosewerte day Sensordaten des lestattensatzes erzeugen # ohne c zu Berücksichtigen if c == None: predicted_test = model.predict(X_test) |
| | <pre># # Submissiondatensatz einlesen und prognostizierte Werte hineinschreiben submission = pd.read_csv('sample_submission.csv') submission['Fehlerhaft'] = predicted_test</pre> |
| | # # In .csv speichern, wenn save=True if save == True: |
| | # # Standardnamen wählen, wenn manu_name == False if manu_name == False: submission.to_csv('./predicted_values.csv', index=False) |
| | # # Standardnamen mit timestamp kombinieren, wenn manu_name == True |
| | <pre>if manu_name == True: import datetime now = datetime.datetime.now() name = now.strftime('%Y-%m-%dT%H%M%S') + ('-%02d' % (now.microsecond / 10000))</pre> |
| | submission.to_csv('./predicted_values_' + str(name) + '.csv', index=False) return submission.head(), submission.loc[submission['Fehlerhaft'] == 1] |
| _ | 1.2.4 Funktion zum Filtern von Quantilen |
| [5]: | # Definition einer Funktion, welche einen gegeben DataFrame # um untere und obere Quantile beschneiden kann |
| | # # Argumente: # - orignal_df: DataFrame welcher bearbeitet werden soll |
| | # - quantile_low: Unteres Quantil bis zu welchem orignal_df beschnitten werden soll # - quantile_high: Oberes Quantil welchem orignal_df beschnitten werden soll # - colum_to_drop: Spalte des orignal_df, welche während des Vorgangs gedroppt werden soll |
| | <pre># def filter_my_df(orignal_df, quantile_low, quantile_high, colum_to_drop):</pre> |
| | # # Spalte "colum_to_drop" aus dem Datensatz entfernen df_filtered = orignal_df.loc[:, orignal_df.columns != colum_to_drop] |
| | # Quantil-DataFrame erzeugen quant_df = df_filtered.quantile([quantile_low, quantile_high]) # Quantil-DataFrame auf orignal_df anweden |
| | <pre>df_filtered = df_filtered.apply(lambda x: x[(x>quant_df.loc[quantile_low,x.name]) &</pre> |
| | # # Spalte "Fehlerhaft" dem gefiltertem DataFrame wieder anfügen df_filtered = pd.concat([orignal_df.loc[:,colum_to_drop], df_filtered], axis=1) |
| | # Aus Beschneidung resultierende NaN-Werte bereinigen df_filtered.dropna(inplace=True) return df_filtered |
| | 1.3 Datensatz einlesen (bereinigigen) und betrachten |
| 1 | 1.3.1 Datensatz einlesen # |
| 2021 | <pre># Datensatz einlesen data = pd.read_csv('train.csv', index_col=0)</pre> |
| | 1.3.2 Optionale Datensatzbereinigung |
| | # # Datensatz unterteilen |
| | <pre>df_fehlerfrei = data.loc[data['Fehlerhaft'] == 0] df_fehlerhaft = data.loc[data['Fehlerhaft'] == 1] """</pre> |
| | n n n + + + + + + + + + + + + + + + + + |

KNeighborsClassifier_v004

February 1, 2020

© Thomas Robert Holy 2019 Version 1.0 Visit me on GitHub: https://github.com/trh0ly Kaggle Link: https://www.kaggle.com/c/dda-p2/leaderboard

1 KNeighborsClassifier

from sklearn.pipeline import Pipeline

from sklearn.metrics import auc

import matplotlib.pyplot as plt

from sklearn.svm import SVC

Werte von c betrachtet werden.

import datetime as dt

1.2 Hilfsfunktionen

#-----# Argumente:

if c != None:

if c == None:

- c:

from sklearn.linear_model import LogisticRegression from sklearn.preprocessing import StandardScaler from sklearn.preprocessing import RobustScaler from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import confusion_matrix from sklearn.metrics import classification_report from sklearn.metrics import roc_curve, roc_auc_score

from sklearn.ensemble import RandomForestClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.preprocessing import PolynomialFeatures from sklearn.model_selection import GridSearchCV from sklearn.model_selection import StratifiedKFold from sklearn.model_selection import cross_val_score from sklearn.model_selection import train_test_split

from IPython.core.display import display, HTML from scipy.spatial.distance import euclidean

1.2.1 Funktion zur Betrachtung der Konfusionsmatrix

 ${\tt from \ sklearn.metrics.pairwise \ import \ manhattan_distances}$

- y_true: Zum DataFrame X gehörige Werte der Zielgröße

[2]: # Definition einer Funktion, welche eine Konfusionsmatrix und einen Klassifikationsreport # zurückgibt. Die Konfusionsmatrix kann, wenn ein Wert für c gegeben ist, für beliebige

- X: DataFrame auf welchem die Prognose durchgefürt werden soll (ohne die Zielgröße)

---> Wenn None, dann wird die Konfusionsmatrix ohne die Einbeziehung von c bestimmt # ---> Wenn != None, dann wird die Konfusionsmatrix in Abhängigkeit von c bestimmt

- class_names: Bezeichnung für die Spalten des Dataframes (default=['0', '1'], mit 0 = negativ und 1 = positiv)

- model: Modell auf Basis dessen die Konfusionsmatrix berechnet werden soll

def get_confusion_matrix(X, y_true, model, class_names=['0', '1'], c=None):

Vorgelagerte Berechnung falls ein Wert für c gegeben ist # und die Konfusionsmatrix für ein gegebenes c anpasst

pred_probability = model.predict_proba(X) pred_probability = pred_probability >= c y_pred = pred_probability[:, 1].astype(int)

Wenn kein Wert für c gegeben, dann führe Prognose

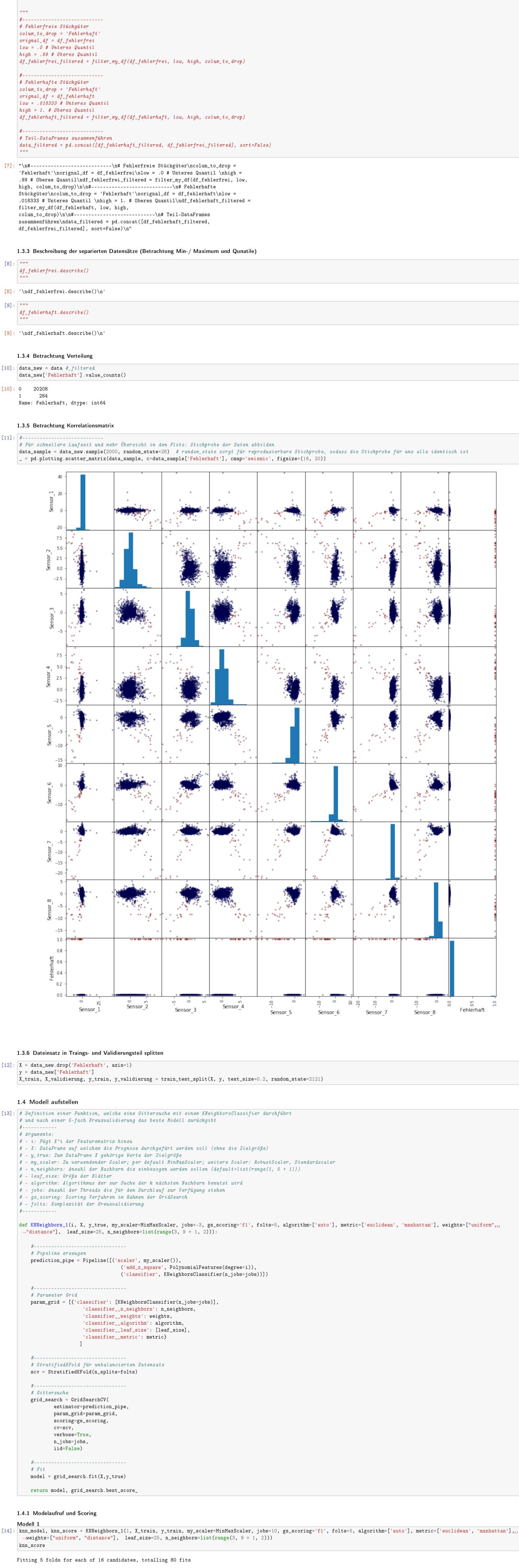
lediglich auf Basis des Modells durch

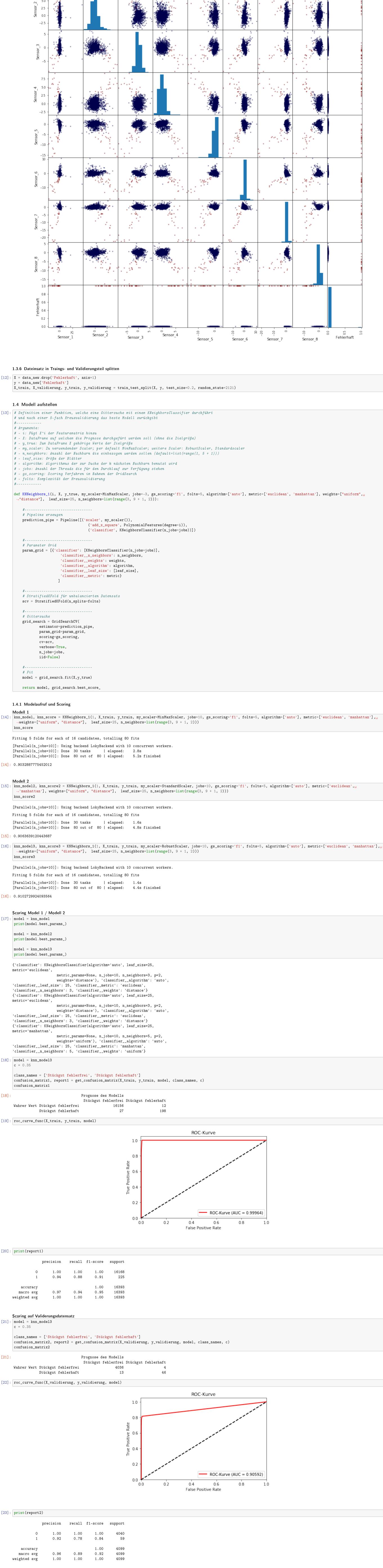
y_pred = model.predict(X)

1.1 Package Import

import pandas as pd

[1]: import numpy as np





1.5 Submit

c = 0.35

 $model = knn_model3$

 $confusion_matrix3$

c)\nconfusion_matrix3\n"

1.5.2 Submit der Prognose

 $submission_fehlerhaft$

[26]: '\nsubmission_fehlerhaft\n

manu_name=True) \nsubmission_head \n'

 $submission_head$

11 11 11

nnn

[25]: """

[26]: """

HHHH

[24]:

1.5.1 Kontrolle Modellwahl (Modell 1 oder 2) anhand der Konfusionsmatrix

 $confusion_matrix3, \ report3 = get_confusion_matrix(X_validierung, \ y_validierung, \ model, \ class_names, \ c)$

class_names = ['Stückgut fehlerfrei', 'Stückgut fehlerhaft']

'Stückgut fehlerhaft']\nconfusion_matrix3, report3 =

[24]: "\nmodel = knn_model3\nc = 0.35\n\nclass_names = ['Stückgut fehlerfrei',

[25]: '\nsubmission_head, submission_fehlerhaft = submit(model, c, save=True,

1.5.3 Ausgabe DataFrame mit als defekt klassifizierten Stückgütern im Testdatensatz

get_confusion_matrix(X_validierung, y_validierung, model, class_names,

 $submission_head, \ submission_fehlerhaft \ = \ submit (model, \ c, \ save=True, \ manu_name=True)$