1.01-RC-Analyse

November 15, 2024

Verschiedene Ansätze um die Daten zu analysieren.

[1]:
```
import numpy as np
import pandas as pd
import scipy
import sys
import os
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score

sys.path.append(os.path.dirname(os.getcwd()))
from src.load_covid19 import load_clean_covid19

df = load_clean_covid19()

# # male cannot be pregnant
# df.loc[df.SEX == 'male', 'PREGNANT'] = False
# # zuhause kann nicht beatmet werde & intubiert werden
# df.loc[df.PATIENT_TYPE=='athome','ICU']=False
# df.loc[df.PATIENT_TYPE=='athome','INTUBED']=False

# print(df.isna().sum())
# bool_columns = ['PNEUMONIA', 'PREGNANT', 'DIABETES', 'COPD', 'ASTHMA',␣
 ↪'INMSUPR',
#                 'HIPERTENSION', 'CARDIOVASCULAR', 'RENAL_CHRONIC',␣
 ↪'OTHER_DISEASE', 'OBESITY', 'TOBACCO',
#                 'INTUBED', 'ICU', 'DIED']
# for i in bool_columns:
#     df[i] = df[i].fillna(False)

# df.isna().sum()
```

```
Dataset already exists at s:\SynologyDrive\SynologyDrive\Uni\Master 2\3.
Semester\Projektpraktikum Web Science\covid-19-risiko-
erkennung\src\..\data\raw\covid19-dataset. Skipping download.
Saving clean dataset to: s:\SynologyDrive\SynologyDrive\Uni\Master 2\3.
```

```
Semester\Projektpraktikum Web Science\covid-19-risiko-
erkennung\data\interim\covid-data-clean.csv
Saved
Loading clean dataset from: s:\SynologyDrive\SynologyDrive\Uni\Master 2\3.
Semester\Projektpraktikum Web Science\covid-19-risiko-
erkennung\data\interim\covid-data-clean.csv
```

[3]:
```
df.PATIENT_TYPE.unique()
```

[3]:
```
[1, 2]
Categories (2, int64): [1, 2]
```

[4]:
```
df['SEX'] = df['SEX'].replace('female', 0).replace('male', 1)
df['PATIENT_TYPE'] = df['PATIENT_TYPE'].replace('returned home', 0).
 ↪replace('hospitalization', 1)


df['AT_RISK'] = df['DIED']+df['INTUBED']+df['ICU']


df.AT_RISK = df.AT_RISK.apply(lambda x: 1 if x > 0 else 0)
df = df.drop(columns = ['DIED', 'INTUBED', 'ICU'])


# train(90%), test(5%)
train, test = train_test_split(df, test_size=0.1, shuffle=True)

# df.to_csv("../data/raw/covid19-dataset/Covid Data2.csv")

train_y = train.AT_RISK.to_numpy()
train_x = train.drop(columns = ['AT_RISK']).to_numpy()


test_y = test.AT_RISK.to_numpy()
test_x = test.drop(columns = ['AT_RISK']).to_numpy()


def get_scores(y_test, y_pred):
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    return acc, f1, prec, rec
```

KNN

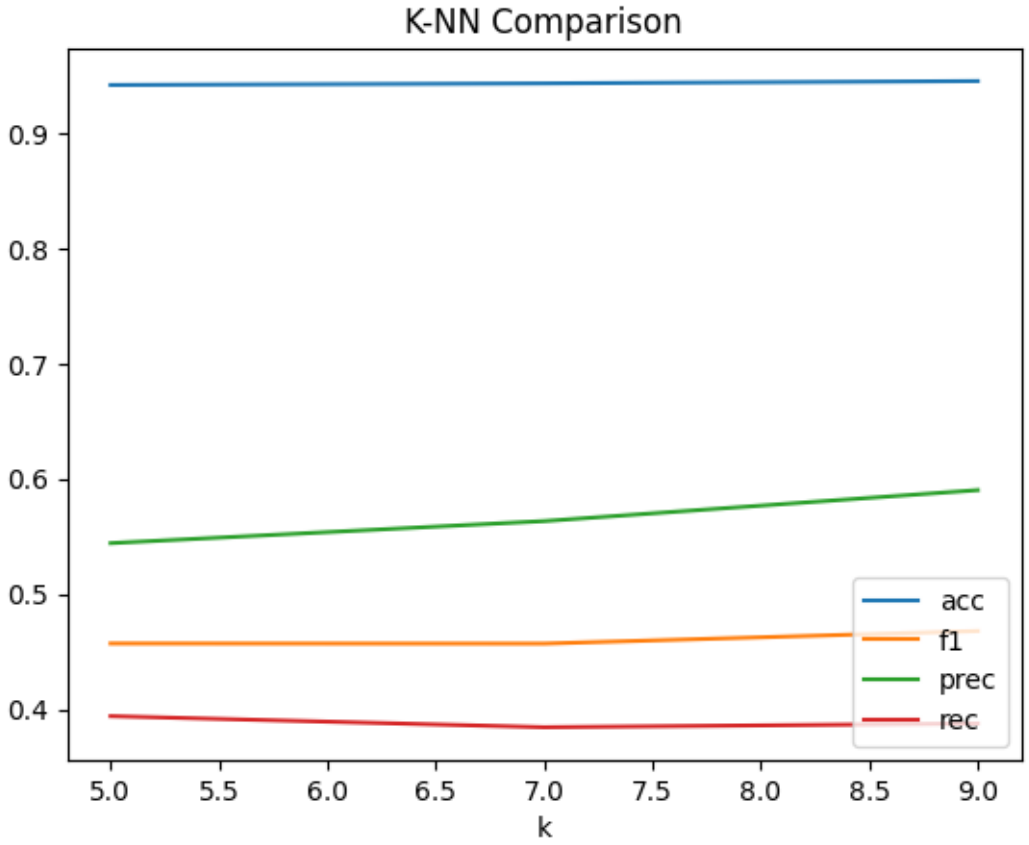
[5]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

```
# from sklearn.metrics import classification_report
# from sklearn.metrics import confusion_matrix


# undersampling the train set
undersampling = 10000
train_xus = train_x[:undersampling]
train_yus = train_y[:undersampling]
test_xus = test_x[:undersampling]
test_yus = test_y[:undersampling]



ks = [5,7,9]
# ks = [1,3,5,7,9]
accs = []
f1s = []
precs = []
recs = []

for k in ks:
    knn_model = KNeighborsClassifier(n_neighbors=k)

    knn_model.fit(train_xus, train_yus)
    # Scale the features using StandardScaler
    # scaler = StandardScaler()
    # test_x_std = scaler.fit_transform(validation_x)
    # X_test = scaler.transform(X_test)

    y_pred = knn_model.predict(test_xus)
    acc, f1, prec, rec = get_scores(test_yus, y_pred)
    accs.append(acc)
    f1s.append(f1)
    precs.append(prec)
    recs.append(rec)
    print(f"k: {k} - {acc}")
```

```
k: 5 - 0.9416
k: 7 - 0.943
k: 9 - 0.945
```

[7]:
```
import matplotlib.pyplot as plt

plt.title('K-NN Comparison')
plt.errorbar(ks, accs, label="acc")
```

```
plt.errorbar(ks, f1s, label="f1")
plt.errorbar(ks, precs, label="prec")
plt.errorbar(ks, recs, label="rec")
plt.legend(loc ='lower right')
plt.xlabel('k')
plt.show()
```



Decision Tree

[9]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree


# undersampling the train set
undersampling = 10000
train_xus = train_x[:undersampling]
train_yus = train_y[:undersampling]
test_xus = test_x[:undersampling]
test_yus = test_y[:undersampling]
```

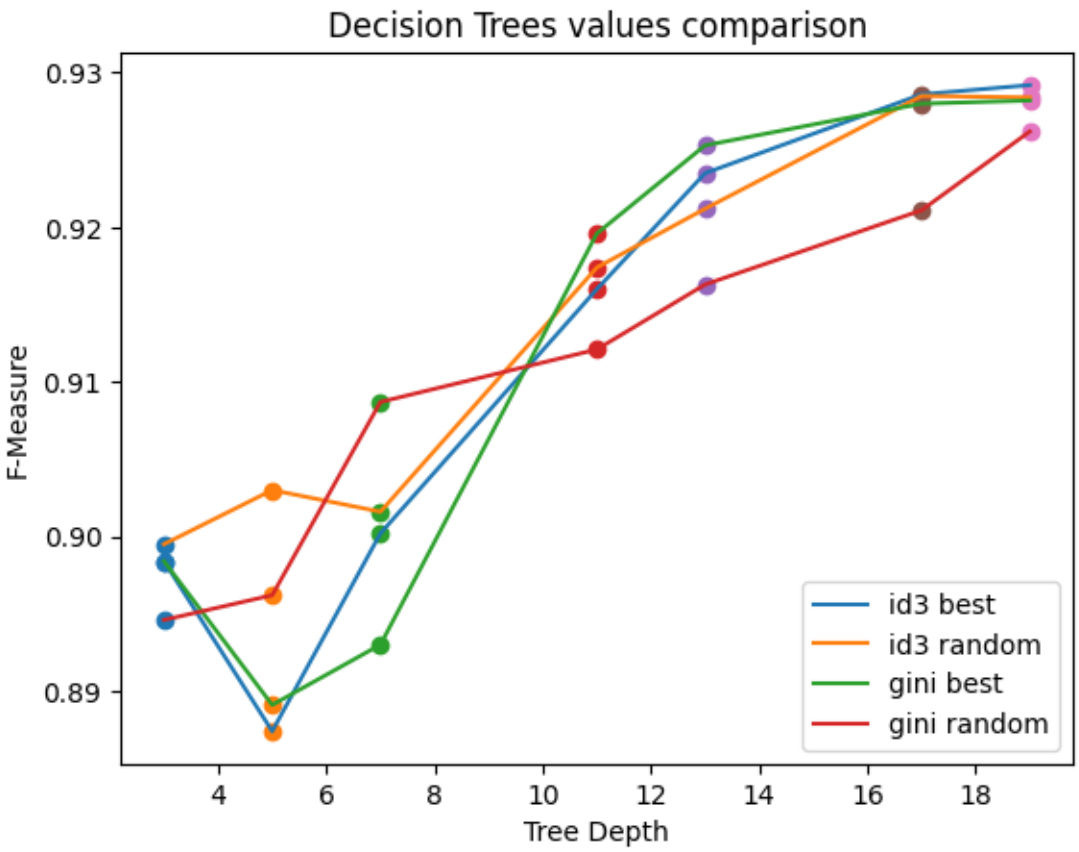
```
heuristics = ["entropy", "gini"]
splitters = ["best", "random"]
max_depth = [3, 5, 7, 11, 13, 17, 19]
dt_fmeasures = {}
dt_best_result = 0
dt_best = None

for d in max_depth:
    dt_fmeasures[d] = np.zeros(len(heuristics)*len(splitters))
    i=0
    for heuristic, splitter in [(heuristic, splitter) for heuristic in␣
 ↪heuristics for splitter in splitters]:
        # create a Decision Tree classifier instance and compute the prediction
        dt_classifier = DecisionTreeClassifier(criterion=heuristic,␣
 ↪splitter=splitter, max_depth=d, class_weight='balanced')
        dt_classifier.fit(train_xus, train_yus)
        y_pred = dt_classifier.predict(test_xus)

        # calculate F-measures and save best model
        acc, f1, prec, rec = get_scores(test_yus, y_pred)
        dt_fmeasures[d][i] = acc
        print(f"{d}:{heuristic} {splitter}: f1: {f1}")
        i=i+1

        if f1 > dt_best_result:
            dt_best_result = f1
            dt_best = dt_classifier
```

```
3:entropy best: f1: 0.5423423423423424
3:entropy random: f1: 0.54421768707483
3:gini best: f1: 0.5423423423423424
3:gini random: f1: 0.5323868677905945
5:entropy best: f1: 0.5125541125541125
5:entropy random: f1: 0.5450281425891182
5:gini best: f1: 0.5192891200693541
5:gini random: f1: 0.5336927223719676
7:entropy best: f1: 0.5288007554296507
7:entropy random: f1: 0.5264677574590952
7:gini best: f1: 0.5109689213893968
7:gini random: f1: 0.5450921773791729
11:entropy best: f1: 0.5248868778280543
11:entropy random: f1: 0.547645125958379
11:gini best: f1: 0.540045766590389
11:gini random: f1: 0.5251215559157212
13:entropy best: f1: 0.5245494095711623
13:entropy random: f1: 0.5407925407925408
```

```
13:gini best: f1: 0.5239005736137667
13:gini random: f1: 0.5153445280833816
17:entropy best: f1: 0.5027855153203342
17:entropy random: f1: 0.5165652467883706
17:gini best: f1: 0.4857142857142857
17:gini random: f1: 0.45847632120796156
19:entropy best: f1: 0.48917748917748916
19:entropy random: f1: 0.450920245398773
19:gini best: f1: 0.45770392749244715
19:gini random: f1: 0.462882096069869
```

[11]:
```
print(dt_best_result)



l1, l2, l3, l4 = [], [], [], []
for k in dt_fmeasures.keys():
    fmeasures = dt_fmeasures[k]
    plt.scatter([k] * len(fmeasures), fmeasures)
    l1.append(dt_fmeasures[k][0])
    l2.append(dt_fmeasures[k][1])
    l3.append(dt_fmeasures[k][2])
    l4.append(dt_fmeasures[k][3])

plt.errorbar(max_depth, l1, label ='id3 best')
plt.errorbar(max_depth, l2, label ='id3 random')
plt.errorbar(max_depth, l3, label ='gini best')
plt.errorbar(max_depth, l4, label ='gini random')
plt.legend(loc ='lower right')
plt.title('Decision Trees values comparison')
plt.xlabel('Tree Depth')
plt.ylabel('F-Measure')
plt.show()

# fig = plt.figure(figsize=(25,20))
# _ = tree.plot_tree(dt_classifier,
#                    #    feature_names=iris.feature_names,
#                    #    class_names=iris.target_names,
#                       filled=True)
```

```
0.547645125958379
```

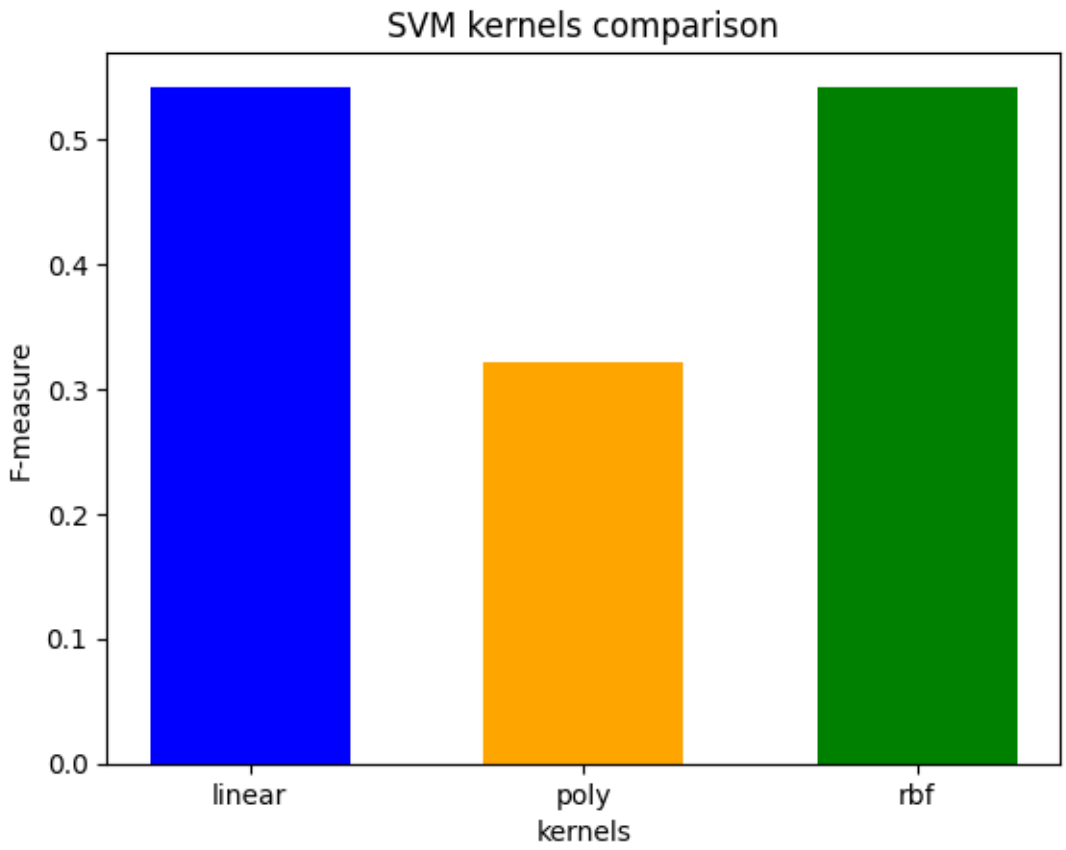


SVM

[12]:
```
from sklearn.svm import SVC



# undersampling the train set
undersampling = 10000
train_xus = train_x[:undersampling]
train_yus = train_y[:undersampling]
test_xus = test_x[:undersampling]
test_yus = test_y[:undersampling]


kernels = ["linear", "poly", "rbf"]
svm_fmeasures = {}
svm_best_result = 0

for kernel in kernels:
    # create a SVM classifier instance and compute the prediction
```

```
    svm_classifier = SVC(kernel=kernel, degree=8, class_weight='balanced')
    svm_classifier.fit(train_xus, train_yus)
    y_pred = svm_classifier.predict(test_xus)

    # calculate F-measures and save best model
    acc, f1, prec, rec = get_scores(test_yus, y_pred)
    svm_fmeasures[kernel] = f1
    print(f"{kernel}:{f1}")
    if svm_fmeasures[kernel] > svm_best_result:
        svm_best_result = svm_fmeasures[kernel]
        svm_best_model = svm_classifier



plt.bar(svm_fmeasures.keys(), svm_fmeasures.values(), color=['blue', 'orange',␣
 ↪'green', 'red'], width=.6)
plt.title('SVM kernels comparison')
plt.xlabel('kernels')
plt.ylabel('F-measure')
plt.show()
```

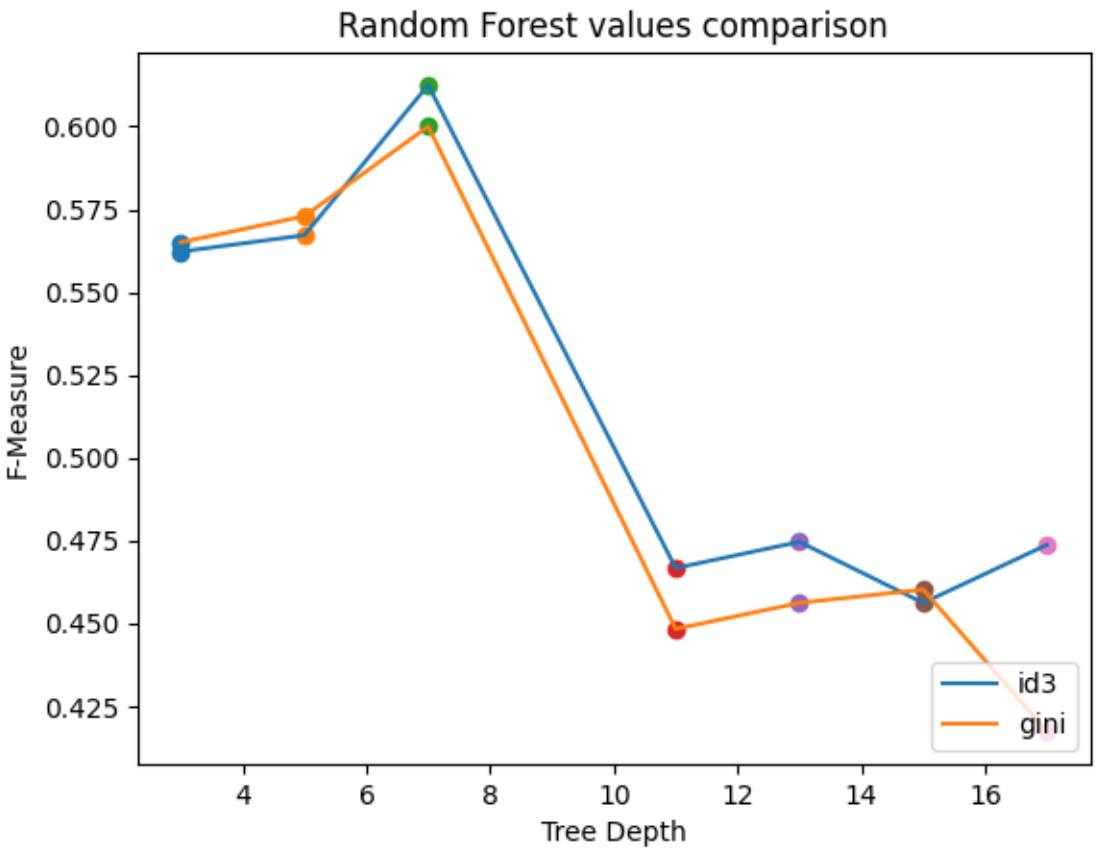
```
linear:0.54182156133829
poly:0.3215728521411258
rbf:0.54182156133829
```



Random Forrest

[13]:
```
from sklearn.ensemble import RandomForestClassifier



# undersampling the train set
undersampling = 1000
train_xus = train_x[:undersampling]
train_yus = train_y[:undersampling]
test_xus = test_x[:undersampling]
test_yus = test_y[:undersampling]

# set Random Forest parameters
heuristics = ["entropy", "gini"]
max_depth = [3, 5, 7, 11, 13, 15, 17]
rf_fmeasures = {}
rf_best_result = 0
for j in range(5):
    for d in max_depth:
```

```
        rf_fmeasures[d] = np.zeros(len(heuristics))
        for (i, heuristic) in enumerate(heuristics):
            # create a Random Forest classifier instance and compute the␣
 ↪prediction
            rf_classifier = RandomForestClassifier(n_estimators=100,␣
 ↪criterion=heuristic, max_depth=d, class_weight='balanced')
            rf_classifier.fit(train_xus, train_yus)
            y_pred = rf_classifier.predict(test_xus)

            # calculate F-measures and save best model
            acc, f1, prec, rec = get_scores(test_yus, y_pred)
            rf_fmeasures[d][i] = f1
            if rf_fmeasures[d][i] > rf_best_result:
                rf_best_result = rf_fmeasures[d][i]
                rf_best_model = rf_classifier


# plot the results
l1, l2 = [], []
for k in rf_fmeasures.keys():
    fmeasures = rf_fmeasures[k]
    plt.scatter([k] * len(fmeasures), fmeasures)
    l1.append(rf_fmeasures[k][0])
    l2.append(rf_fmeasures[k][1])

# create line for each  models
plt.errorbar(max_depth, l1, label ='id3')
plt.errorbar(max_depth, l2, label ='gini')
plt.legend(loc ='lower right')
plt.title('Random Forest values comparison')
plt.xlabel('Tree Depth')
plt.ylabel('F-Measure')
plt.show()
```


Random Forest values comparison
F-Measure
0.600
0.575
0.550
0.525
0.500
0.475
0.450
0.425
4
6
8
10
12
14
16
Tree Depth
id3
gini