Machine learning as a way to classify treatment responsiveness of leishmaniasis patiens

A NAIL107 project by Lucie Korená and Lukáš Cakl

Dataset

The dataset was acquired from the paper: Bamorovat M, Sharifi I, Rashedi E, et al. A novel diagnostic and prognostic approach for unresponsive patients with anthroponotic cutaneous leishmaniasis using artificial neural networks.

Plos one. 2021 ;16(5):e0250904. DOI: 10.1371/journal.pone.0250904.

Fetching:

We download the xls formatted dataset:

In [1]:
```
import urllib
import os

datasetUrl = 'https://journals.plos.org/plosone/article/file?type=supplementary&id=i

def downloadData(sourceUrl, localFile, localPath):
    if not os.path.isdir(localPath):
        os.makedirs(localPath)
    localUri = os.path.join(localPath, localFile)
    if not os.path.exists(localUri):
        urllib.request.urlretrieve(sourceUrl, localUri)
        print('Dataset downloaded')
    else:
        print('No download neccessary')

downloadData(datasetUrl, 'dataset.xls', './data')
```

```
No download neccessary
```

And parse it using the pandas library:

In [2]:
```
import pandas as pd
dataset = pd.read_excel("./data/dataset.xls")
dataset
```

Out[2]:

	interior.housing.condition	age	sex	education	duration.of.lesion	number.of.lesion	location.of
0	0	3	0	2	1	1	
1	1	3	1	1	1	1	
2	0	2	0	2	1	1	
3	0	3	1	2	1	2	
4	1	3	1	3	2	1	
...	...	...	...	...	...	...	
167	1	2	1	1	2	1	

	interior.housing.condition	age	sex	education	duration.of.lesion	number.of.lesion	location.of
168	0	2	1	1	2	1	
169	1	5	1	2	3	1	
170	0	2	0	2	2	1	
171	0	1	0	2	2	1	

172 rows × 10 columns

Next we extract our features and targets from the dataset:

In [3]:
```
target = dataset["treatment.outcome"]
target
```

Out[3]:
```
0      0
1      0
2      0
3      0
4      0
      ..
167    1
168    1
169    1
170    1
171    1
Name: treatment.outcome, Length: 172, dtype: int64
```

In [4]:
```
data = dataset.drop(labels="treatment.outcome", axis = 1)
data
```

Out[4]:

	interior.housing.condition	age	sex	education	duration.of.lesion	number.of.lesion	location.of
0	0	3	0	2	1	1	
1	1	3	1	1	1	1	
2	0	2	0	2	1	1	
3	0	3	1	2	1	2	
4	1	3	1	3	2	1	
...	...	...	...	...	...	...	
167	1	2	1	1	2	1	
168	0	2	1	1	2	1	
169	1	5	1	2	3	1	
170	0	2	0	2	2	1	
171	0	1	0	2	2	1	

172 rows × 9 columns

Train / test split

Here we split our dataset into its train and test portions, which will not be intermixed from now on. All our analysis will be done on the train portion of data, while the test will be used for scoring purposes afterwards

In [5]:
```
import sklearn
from sklearn.model_selection import train_test_split
randomSeed = 42 # To maintain reproducibility 42 is used as the random seed

trainingData, testData, trainingTarget, testTarget = sklearn.model_selection.train_t
```

In [6]:
```
trainingData.describe()
```

Out[6]:

	interior.housing.condition	age	sex	education	duration.of.lesion	number.of.le
count	137.000000	137.000000	137.000000	137.000000	137.000000	137.00
mean	0.671533	2.613139	0.532847	1.890511	1.773723	1.15
std	0.471379	1.267514	0.500751	0.734435	0.652965	0.36
min	0.000000	1.000000	0.000000	1.000000	1.000000	1.00
25%	0.000000	2.000000	0.000000	1.000000	1.000000	1.00
50%	1.000000	2.000000	1.000000	2.000000	2.000000	1.00
75%	1.000000	4.000000	1.000000	2.000000	2.000000	1.00
max	1.000000	5.000000	1.000000	3.000000	3.000000	2.00

Dataset exploration

The dataset is explained as follows:

Features	Categories of features	Number of unresponsive cases (%)	Number of responsive cases (%)
Interior housing condition	Suitable[a]	38 (52.78)	69 (69)
	Unsuitable[b]	34 (47.22)	31 (31)
Age (year)	≤ 7	19 (26.39)	25 (25)
	8–15	18 (25)	30 (30)
	16–30	13 (18.05)	25 (25)
	31–50	13 (18.05)	14 (14)
	≥ 50	9 (12.5)	6 (6)
Sex	Female	30 (41.67)	53 (53)
	Male	42 (58.33)	47 (47)
Education	Illiterate	31 (43.05)	26 (26)
	Primary and secondary	31 (43.05)	46 (46)
	High school and university	10 (13.89)	28 (28)
Duration of lesion (month)	≤ 4	7 (9.72)	53 (53)
	5–12	47 (65.28)	45 (45)
	≥ 13	18 (25)	2 (2)
Number of lesions	≤ 2	65 (90.28)	83 (83)
	≥ 3	7 (9.72)	17 (17)
Location of lesion	Hand	25 (34.72)	51 (51)
	Face	38 (52.78)	31 (31)
	Other	9 (12.5)	18 (18)
Treatment course	Incomplete[c]	16 (22.22)	24 (24)
	Complete[d]	56 (77.78)	76 (76)
History of chronic diseases	Yes	22 (30.55)	6 (6)
	No	50 (69.45)	94 (94)
Total		72 (41.86)	100 (58.14)

a: suitable: denotes the patients who live in an appropriate housing condition in terms of building and interior housing, with no cracks and crevices,
b: unsuitable: denotes the patients who live in inappropriate housing conditions in terms of building and interior housing,
c: incomplete treatment: the patients who did not receive a complete course of intramuscular (IM) or intralesional (IL) treatment along with cryotherapy,
d: completed treatment: the patients who received a full course of treatment schedule.

https://doi.org/10.1371/journal.pone.0250904.t001

Before trying out any machine learning methods, we shall explore our dataset using multiple techniques such as LDA, PCA and simple graphing.

PCA

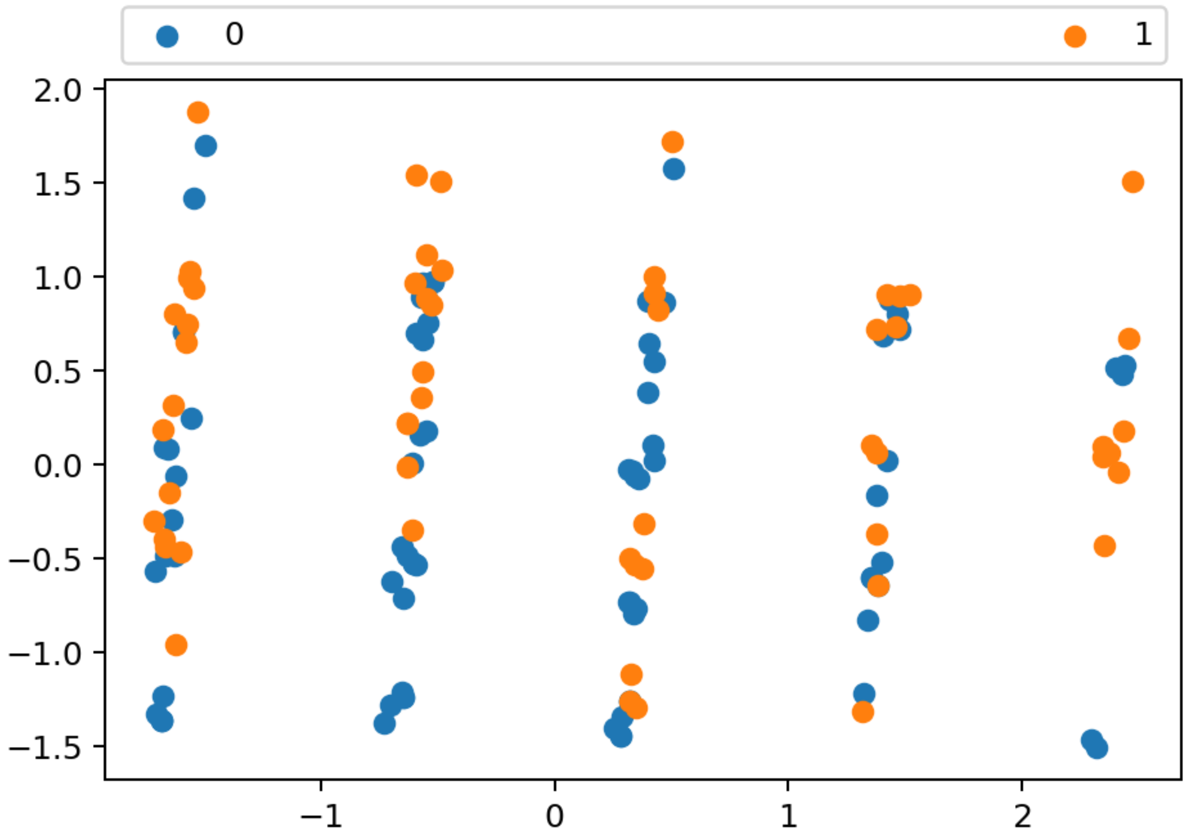
In [7]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pcaTrainingData = pca.fit_transform(trainingData)
print(pca.components_)
print(pca.explained_variance_ratio_)
```

```
[[ 0.0435536   0.99467189 -0.02386315 -0.04091038  0.04211519  0.01445317
   0.03906456 -0.02714484 -0.04735339]
 [-0.04628509 -0.0584617   0.03639587 -0.71321566  0.18454638 -0.00981779
   0.65935463 -0.09141425  0.08473392]]
[0.40502845 0.18613701]
```

We can see that the explained variance in 2 components is still low (~59%), so probably most of the data is important. Also we can see that the second feature (age) gets probably way too much of a value.

In [8]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
fig.dpi = 180

ax.scatter(pcaTrainingData[trainingTarget == 0][:,0], pcaTrainingData[trainingTarget
ax.scatter(pcaTrainingData[trainingTarget == 1][:,0], pcaTrainingData[trainingTarget
ax.legend(bbox_to_anchor=(0, 1, 1, 0), loc="lower left", mode="expand", ncol=2)
```

Out[8]:
```
<matplotlib.legend.Legend at 0x1f071d88fd0>
```



And even the graphed separation is dismal. It is possible that scaling would help as age is blown out of proportion.

LDA

In [9]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

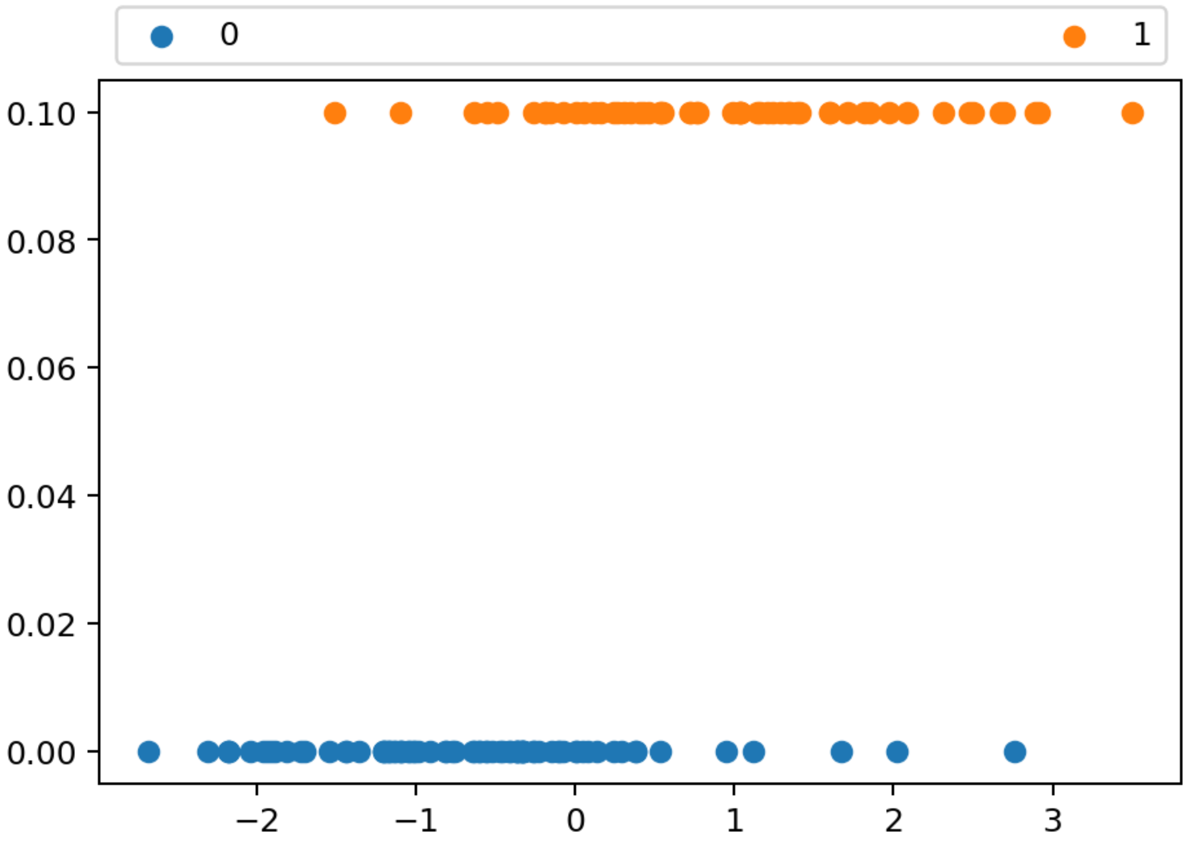
```
lda = LinearDiscriminantAnalysis(store_covariance=True)
lda.fit(trainingData, trainingTarget)
ldaTrainingData = lda.transform(trainingData)
print(lda.coef_)
print(lda.covariance_)
print(lda.explained_variance_ratio_)
```

```
[[-1.53567885  0.29920995  0.59900238 -0.7235833   1.89467649 -1.09865216
   0.12541889  1.40118659  2.11193912]]
[[ 2.11934947e-01  6.96632091e-02 -1.38942246e-02 -1.05647330e-03
   2.15136381e-02  1.40863107e-03  1.72877449e-03  4.32513766e-02
   1.24535792e-02]
 [ 6.96632091e-02  1.58926879e+00 -3.54334742e-02 -2.20322705e-02
   2.68920476e-02  2.69432706e-02  2.86208221e-02 -4.18299398e-02
  -8.52541939e-02]
 [-1.38942246e-02 -3.54334742e-02  2.48834678e-01  3.07625816e-02
  -1.38301959e-02  1.37277500e-02  6.04110642e-02 -7.24484569e-03
  -3.55519273e-02]
 [-1.05647330e-03 -2.20322705e-02  3.07625816e-02  5.05133500e-01
   2.49711871e-03 -2.93507491e-02 -1.87331925e-01  2.41884364e-02
  -2.05772186e-02]
 [ 2.15136381e-02  2.68920476e-02 -1.38301959e-02  2.49711871e-03
   3.18094506e-01 -1.30618517e-02 -7.68344218e-04 -5.57049558e-03
   3.51517480e-02]
 [ 1.40863107e-03  2.69432706e-02  1.37277500e-02 -2.93507491e-02
  -1.30618517e-02  1.26725573e-01 -1.42143680e-02  8.50300935e-03
   3.71366372e-03]
 [ 1.72877449e-03  2.86208221e-02  6.04110642e-02 -1.87331925e-01
  -7.68344218e-04 -1.42143680e-02  4.97983096e-01 -5.97867845e-02
  -3.98578563e-03]
 [ 4.32513766e-02 -4.18299398e-02 -7.24484569e-03  2.41884364e-02
  -5.57049558e-03  8.50300935e-03 -5.97867845e-02  1.77844474e-01
  -2.48671405e-02]
 [ 1.24535792e-02 -8.52541939e-02 -3.55519273e-02 -2.05772186e-02
   3.51517480e-02  3.71366372e-03 -3.98578563e-03 -2.48671405e-02
   1.21294340e-01]]
[1.]
```

LDA seems more promissing. Also the state of chronic diseases, housing, treatment course and duration of lesions seem to have their importance.

In [10]:
```
import numpy as np

fig, ax = plt.subplots()
fig.dpi = 180

ax.scatter(ldaTrainingData[trainingTarget == 0], np.zeros(len(ldaTrainingData[traini
ax.scatter(ldaTrainingData[trainingTarget == 1], np.zeros(len(ldaTrainingData[traini
ax.legend(bbox_to_anchor=(0, 1, 1, 0), loc="lower left", mode="expand", ncol=2)
```

Out[10]:
```
<matplotlib.legend.Legend at 0x1f073e889d0>
```



But the plotting of LDA shows, that the set is not linearly separable

Feature engineering

Certain features in our dataset are represented suboptimally. Interior housing condition, sex, lesion locations, treatment and chronic disease history are all categorical and should be one hot encoded. Furthermore age is represented only by binning into {1, 2, 3, 4, 5} as is education {1, 2, 3} and duration of lesion {1, 2, 3}. Here we should experiment with one hot and scaling and see which is better. As for the number of lesions, that makes sense as an integer, but scaling should help there too.

In [11]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

ct = ColumnTransformer([
        ("standardScale", StandardScaler(), ['age', 'education', 'duration.of.lesion
        ("oneHot", OneHotEncoder(handle_unknown='ignore'), ['interior.housing.condit
    ], remainder='passthrough')
trainingFeatures = ct.fit_transform(trainingData)
testingFeatures = ct.fit_transform(testData)
print(trainingFeatures.shape)
pd.DataFrame(trainingFeatures)
```

```
(137, 15)
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	-0.485509	0.149626	-1.189286	-0.425481	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0
1	-1.277350	0.149626	-1.189286	-0.425481	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0
2	-1.277350	1.516214	0.347810	-0.425481	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0
3	-0.485509	-1.216961	-1.189286	-0.425481	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	-1.277350	0.149626	-1.189286	-0.425481	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
132	-0.485509	1.516214	-1.189286	-0.425481	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0
133	-0.485509	1.516214	1.884906	-0.425481	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0
134	-1.277350	1.516214	-1.189286	-0.425481	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0
135	0.306333	-1.216961	0.347810	-0.425481	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0
136	0.306333	0.149626	0.347810	-0.425481	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0

137 rows × 15 columns

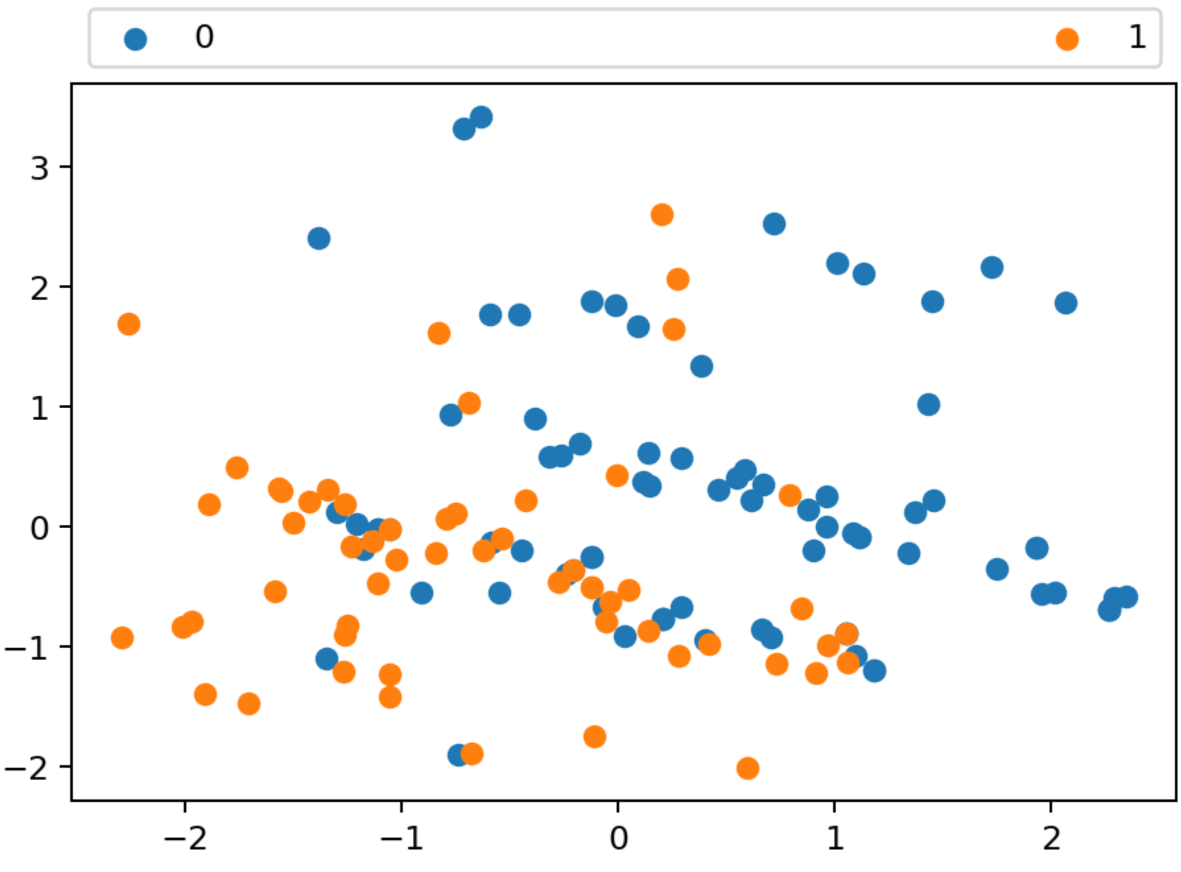
In [12]:

```
pcaTrainingFeatures = pca.fit_transform(trainingFeatures)
print(pca.components_)
print(pca.explained_variance_ratio_)
fig, ax = plt.subplots()
fig.dpi = 180

ax.scatter(pcaTrainingFeatures[trainingTarget == 0][:,0], pcaTrainingFeatures[traini
ax.scatter(pcaTrainingFeatures[trainingTarget == 1][:,0], pcaTrainingFeatures[traini
ax.legend(bbox_to_anchor=(0, 1, 1, 0), loc="lower left", mode="expand", ncol=2)
```

```
[[-0.26382819  0.71013496 -0.57037414  0.06163807 -0.0193238   0.0193238
  -0.04185176  0.04185176  0.21298404 -0.16316421 -0.04981983 -0.02941926
   0.02941926  0.09091035 -0.09091035]
 [ 0.25893623 -0.31143996 -0.46237172  0.76045043 -0.03162667  0.03162667
  -0.05092661  0.05092661 -0.09819937  0.12560611 -0.02740674 -0.00472315
   0.00472315  0.07239275 -0.07239275]]
[0.19962875 0.18475853]
```

Out[12]:

```
<matplotlib.legend.Legend at 0x1f07447d430>
```



Model construction and evaluation

To get the best possible model, we will use grid search alongside stratified crossvalidation. We will then report on the best case for each of the models and validate the results on the test dataset. Finally we will compare our results to the ones available in the original paper.

Also note that certain models (e.g. decision trees, random forests and gradient boosting forests) do best with lower number of features. We will also try those without feature engineering.

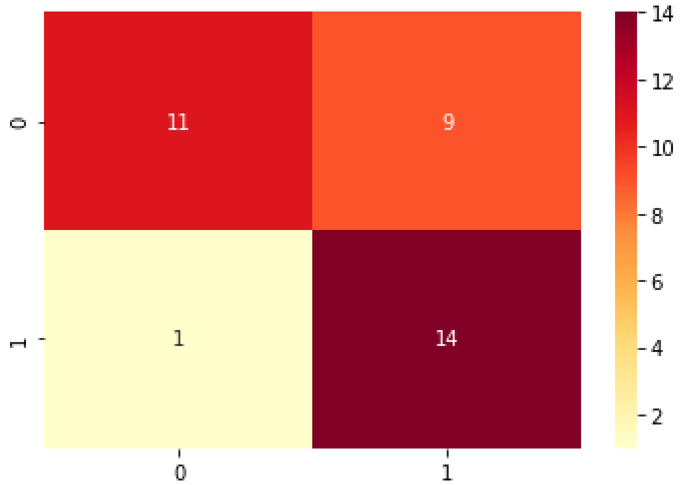
- ### Decision trees

In [82]:
```
import sklearn.model_selection
from sklearn.tree import DecisionTreeClassifier

class_tree = DecisionTreeClassifier()
paramGrid_tree = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': np.linspace(5, 15, 11),
    'min_samples_split': [1.0]
}
crossValidation_tree = sklearn.model_selection.StratifiedKFold()
gridSearch_tree = sklearn.model_selection.GridSearchCV(class_tree, paramGrid_tree, c
gridSearch_tree.fit(trainingData, trainingTarget)
```

Out[82]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=DecisionTreeClassifier(), n_jobs=5,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': array([ 5.,  6.,  7.,  8.,  9., 10., 11., 12.,
13., 14., 15.]),
                         'min_samples_split': [1.0],
                         'splitter': ['best', 'random']})
```

In [83]:
```
import seaborn as sn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

predictions_tree = gridSearch_tree.predict(testData)
accuracy_tree = accuracy_score(testTarget, predictions_tree)
confusion_mat_tree = confusion_matrix(testTarget, predictions_tree)
sn.heatmap(confusion_mat_tree, annot=True, cmap = "YlOrRd")
print("Výsledná accuracy: ", accuracy_tree)
```

```
Výsledná accuracy:  0.7142857142857143
```



Accuracy ~ 71.4%

In [84]:
```python
gridSearch_tree.best_params_
```

Out[84]:
```
{'criterion': 'entropy',
 'max_depth': 15.0,
 'min_samples_split': 1.0,
 'splitter': 'random'}
```

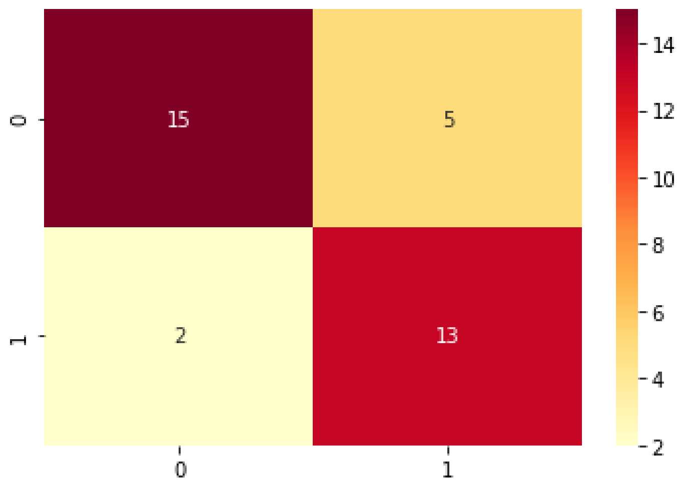
- ### Random Forest:

In [73]:
```python
import sklearn.model_selection
from sklearn.ensemble import RandomForestClassifier

class_random_forest = RandomForestClassifier()
rng = np.random.default_rng()
paramGrid_random_forest = {
    'n_estimators': [75, 100, 125],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 8, 10],
    'class_weight': ['balanced', 'balanced_subsample']
}
crossValidation_random_forest = sklearn.model_selection.StratifiedKFold()
gridSearch_random_forest = sklearn.model_selection.GridSearchCV(class_random_forest,
gridSearch_random_forest.fit(trainingFeatures, trainingTarget)
```

Out[73]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=RandomForestClassifier(), n_jobs=5,
             param_grid={'class_weight': ['balanced', 'balanced_subsample'],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 8, 10],
                         'n_estimators': [75, 100, 125]})
```

In [74]:
```python
import seaborn as sn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

predictions_random_forest = gridSearch_random_forest.predict(testingFeatures)
accuracy_random_forest = accuracy_score(testTarget, predictions_random_forest)
confusion_mat_random_forest = confusion_matrix(testTarget, predictions_random_forest
sn.heatmap(confusion_mat_random_forest, annot=True, cmap = "YlOrRd")
print("Výsledná accuracy: ", accuracy_random_forest)
```

```
Výsledná accuracy:  0.8
```



Accuracy ~ 80%

In [75]:
```python
gridSearch_random_forest.best_params_
```

Out[75]:
```
{'class_weight': 'balanced_subsample',
 'criterion': 'gini',
```

```
 'max_depth': 3,
 'n_estimators': 125}
```

- ### Gradient Boosting

In [164…

```
import sklearn.model_selection
from sklearn.ensemble import GradientBoostingClassifier

class_gradientBoosting = GradientBoostingClassifier()
paramGrid_gradientBoosting = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.1, 0.05, 0.01],
    'n_estimators': [100, 125, 150],
    'max_depth': [2, 3, 5],
    'subsample': [0.5 , 0.75, 1],
    'max_features': [None, 'auto'],
}
crossValidation_gradientBoosting = sklearn.model_selection.StratifiedKFold()
gridSearch_gradientBoosting = sklearn.model_selection.GridSearchCV(class_gradientBoo
gridSearch_gradientBoosting.fit(trainingFeatures, trainingTarget)
```

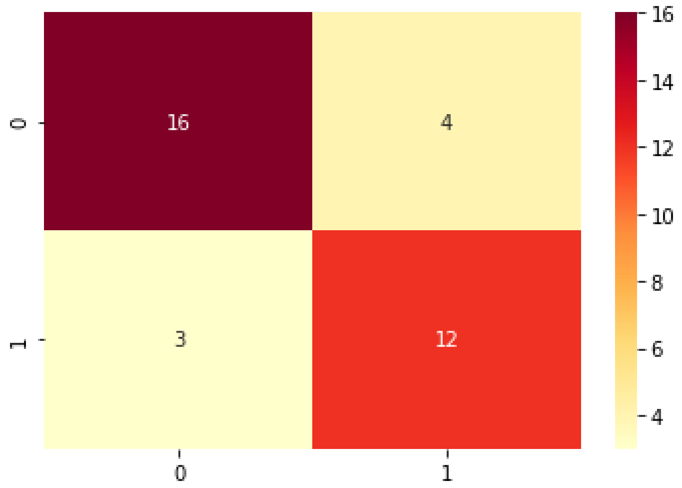
Out[164…

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=GradientBoostingClassifier(), n_jobs=5,
             param_grid={'learning_rate': [0.1, 0.05, 0.01],
                         'loss': ['deviance', 'exponential'],
                         'max_depth': [2, 3, 5], 'max_features': [None, 'auto'],
                         'n_estimators': [100, 125, 150],
                         'subsample': [0.5, 0.75, 1]})
```

In [165…

```
import seaborn as sn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

predictions_gradientBoosting = gridSearch_gradientBoosting.predict(testingFeatures)
accuracy_gradientBoosting = accuracy_score(testTarget, predictions_gradientBoosting)
confusion_mat_gradientBoosting = confusion_matrix(testTarget, predictions_gradientBo
sn.heatmap(confusion_mat_gradientBoosting, annot=True, cmap = "YlOrRd")
print("Výsledná accuracy: ", accuracy_gradientBoosting)
```

```
Výsledná accuracy:  0.8
```



Accuracy ~ 80%

In [166…

```
gridSearch_gradientBoosting.best_params_
```

Out[166…

```
{'learning_rate': 0.01,
 'loss': 'deviance',
 'max_depth': 2,
 'max_features': 'auto',
```

```
 'n_estimators': 150,
 'subsample': 0.75}
```

In [167…

```
pd.DataFrame(gridSearch_gradientBoosting.cv_results_).sort_values("rank_test_score")
```

Out[167…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_loss
232	0.090558	0.011238	0.000798	3.989225e-04	0.01	deviance
304	0.098935	0.009490	0.000799	3.992596e-04	0.01	exponential
228	0.065232	0.005201	0.000997	5.917394e-07	0.01	deviance
301	0.086370	0.005620	0.000998	6.311281e-04	0.01	exponential
163	0.050163	0.004488	0.000000	0.000000e+00	0.05	exponential
...	...	...	...	...	...	...
106	0.206458	0.015179	0.000998	1.181556e-06	0.1	exponential
42	0.164859	0.009749	0.000797	3.986844e-04	0.1	deviance
37	0.129238	0.006191	0.000797	3.985655e-04	0.1	deviance
49	0.149005	0.005307	0.000997	6.305267e-04	0.1	deviance
52	0.196872	0.015367	0.000999	2.015166e-06	0.1	deviance

324 rows × 19 columns

- ### AdaBoostClassifier

In [97]:

```
import sklearn.model_selection
from sklearn.ensemble import AdaBoostClassifier

class_adaboost = AdaBoostClassifier()
rng = np.random.default_rng()
paramGrid_adaboost = {
    'n_estimators': [25, 30, 35],
```

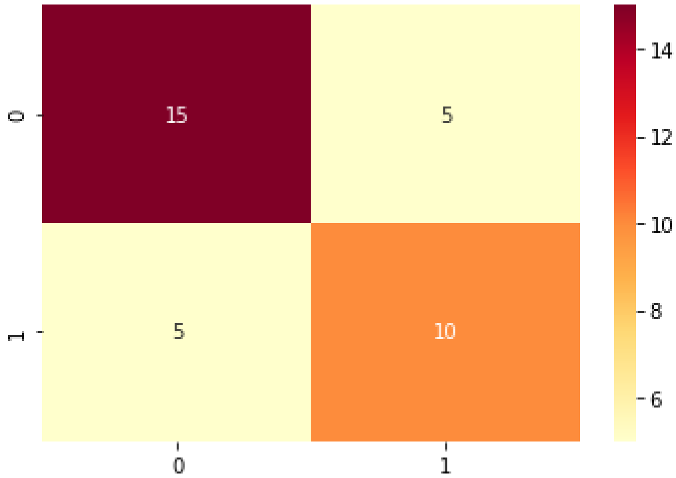
```
    'algorithm': ['SAMME', 'SAMMER.R']
}
crossValidation_adaboost = sklearn.model_selection.StratifiedKFold()
gridSearch_adaboost = sklearn.model_selection.GridSearchCV(class_adaboost, paramGrid
gridSearch_adaboost.fit(trainingFeatures, trainingTarget)
```

```
C:\Users\LukaWolf\AppData\Roaming\Python\Python38\site-packages\sklearn\model_select
ion\_search.py:918: UserWarning: One or more of the test scores are non-finite: [0.7
8095238 0.7952381  0.78809524        nan        nan        nan]
  warnings.warn(
```

Out[97]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=AdaBoostClassifier(), n_jobs=5,
             param_grid={'algorithm': ['SAMME', 'SAMMER.R'],
                         'n_estimators': [25, 30, 35]})
```

In [98]:
```
import seaborn as sn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

predictions_adaboost = gridSearch_adaboost.predict(testingFeatures)
accuracy_adaboost = accuracy_score(testTarget, predictions_adaboost)
confusion_mat_adaboost = confusion_matrix(testTarget, predictions_adaboost)
sn.heatmap(confusion_mat_adaboost, annot=True, cmap = "YlOrRd")
print("Výsledná accuracy: ", accuracy_adaboost)
```

```
Výsledná accuracy:  0.7142857142857143
```



In [99]:
```
gridSearch_adaboost.best_params_
```

Out[99]:
```
{'algorithm': 'SAMME', 'n_estimators': 30}
```

- ### Support vector machine

In [172…
```
from sklearn.svm import SVC

class_svm = SVC()
paramGrid_svm = {
    'kernel': ['linear', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto'],
    'cache_size': [180, 190, 200, 210, 220],
    'decision_function_shape': ['ovo', 'ovr'],
    'class_weight': ['balanced', 'dict']
}
crossValidation_svm = sklearn.model_selection.StratifiedKFold()
gridSearch_svm = sklearn.model_selection.GridSearchCV(class_svm, paramGrid_svm, cv=c
gridSearch_svm.fit(trainingFeatures, trainingTarget)
```

```
C:\Users\LukaWolf\AppData\Roaming\Python\Python38\site-packages\sklearn\model_select
ion\_search.py:918: UserWarning: One or more of the test scores are non-finite: [0.8
```

```
026455   0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
 0.8026455  0.7515873  0.77380952 0.8026455  0.75873016 0.7957672
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan]
  warnings.warn(
```

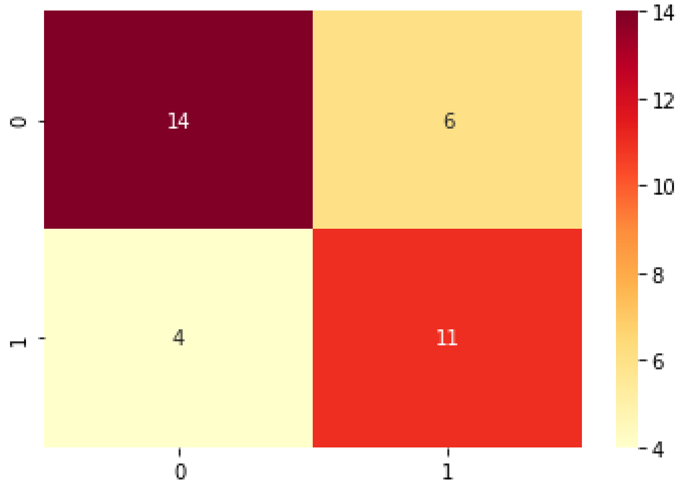
Out[172…

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=SVC(), n_jobs=5,
             param_grid={'cache_size': [180, 190, 200, 210, 220],
                         'class_weight': ['balanced', 'dict'],
                         'decision_function_shape': ['ovo', 'ovr'],
                         'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'rbf', 'sigmoid']})
```

In [173…

```python
import seaborn as sn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

predictions_svm = gridSearch_svm.predict(testingFeatures)
accuracy_svm = accuracy_score(testTarget, predictions_svm)
confusion_mat_svm = confusion_matrix(testTarget, predictions_svm)
sn.heatmap(confusion_mat_svm, annot=True, cmap = "YlOrRd")
print("Výsledná accuracy: ", accuracy_svm)
```

```
Výsledná accuracy:  0.7142857142857143
```



Accuracy ~ 71.4%

In [174…

```python
gridSearch_svm.best_params_
```

Out[174…

```
{'cache_size': 180,
 'class_weight': 'balanced',
 'decision_function_shape': 'ovo',
 'gamma': 'scale',
 'kernel': 'linear'}
```

- ### MLP

In [154…

```
import sklearn.model_selection
from sklearn.neural_network import MLPClassifier

class_mlp = MLPClassifier()
paramGrid_mlp = {
    'hidden_layer_sizes': [(50), (100), (150), (10, 5), (50, 20), (100, 50), (50, 20
    'activation': ['relu', 'logistic'],
    'learning_rate': ['constant', 'adaptive', 'invscaling'],
    'alpha': [0.005 ,0.001, 0.0001]
}
crossValidation_mlp = sklearn.model_selection.StratifiedKFold()
gridSearch_mlp = sklearn.model_selection.GridSearchCV(class_mlp, paramGrid_mlp, cv=c
gridSearch_mlp.fit(trainingFeatures, trainingTarget)
```

```
C:\Users\LukaWolf\AppData\Roaming\Python\Python38\site-packages\sklearn\neural_netwo
rk\_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

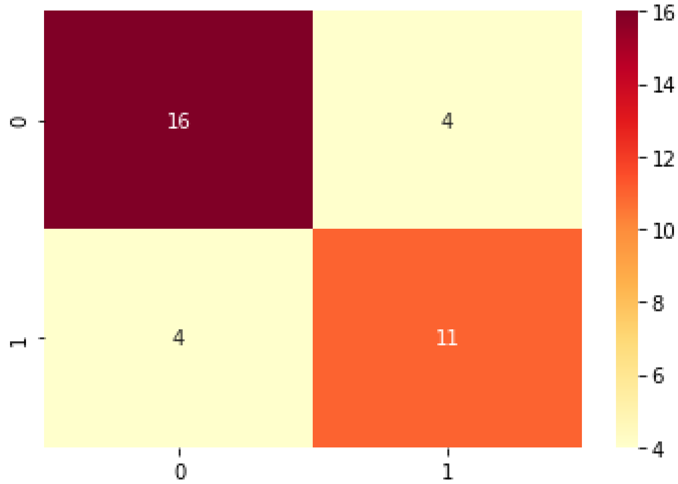
Out[154…

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=MLPClassifier(), n_jobs=5,
             param_grid={'activation': ['relu', 'logistic'],
                         'alpha': [0.005, 0.001, 0.0001],
                         'hidden_layer_sizes': [50, 100, 150, (10, 5), (50, 20),
                                                (100, 50), (50, 20, 10)],
                         'learning_rate': ['constant', 'adaptive',
                                           'invscaling']})
```

In [155…

```
import seaborn as sn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

predictions_mlp = gridSearch_mlp.predict(testingFeatures)
accuracy_mlp = accuracy_score(testTarget, predictions_mlp)
confusion_mat_mlp = confusion_matrix(testTarget, predictions_mlp)
sn.heatmap(confusion_mat_mlp, annot=True, cmap = "YlOrRd")
print("Výsledná accuracy: ", accuracy_mlp)
```

```
Výsledná accuracy:  0.7714285714285715
```



Accuracy ~ 77.1%

In [156…

```
gridSearch_mlp.best_params_
```

Out[156…

```
{'activation': 'logistic',
 'alpha': 0.001,
 'hidden_layer_sizes': 100,
 'learning_rate': 'invscaling'}
```

Conclusions

Below is a table of ranked models:

In [180…

```
pd.DataFrame({
    'Name': ['Decision trees', 'Random forest', 'Gradient boosting', 'ADA boost', 'S
    'Accuracy': [accuracy_tree, accuracy_random_forest, accuracy_gradientBoosting, a
    'Sensitivity': [
        confusion_mat_tree[1][1]/np.sum(testTarget),
        confusion_mat_random_forest[1][1]/np.sum(testTarget),
        confusion_mat_gradientBoosting[1][1]/np.sum(testTarget),
        confusion_mat_adaboost[1][1]/np.sum(testTarget),
        confusion_mat_svm[1][1]/np.sum(testTarget),
        confusion_mat_mlp[1][1]/np.sum(testTarget),
    ],
    'Specificity': [
        confusion_mat_tree[0][0]/np.sum(testTarget==0),
        confusion_mat_random_forest[0][0]/np.sum(testTarget==0),
        confusion_mat_gradientBoosting[0][0]/np.sum(testTarget==0),
        confusion_mat_adaboost[0][0]/np.sum(testTarget==0),
        confusion_mat_svm[0][0]/np.sum(testTarget==0),
        confusion_mat_mlp[0][0]/np.sum(testTarget==0),
    ],
})
```

Out[180…

	Name	Accuracy	Sensitivity	Specificity
0	Decision trees	0.714286	0.933333	0.55
1	Random forest	0.800000	0.866667	0.75
2	Gradient boosting	0.800000	0.800000	0.80
3	ADA boost	0.714286	0.666667	0.75
4	SVM	0.714286	0.733333	0.70
5	MLP	0.771429	0.733333	0.80

My conclusion is that gradient boosting decision trees hit the sweet spot between the relevant metrics. Another possibility would be using the random forest classifier if the alternative treatment is not too dangerous.

Unfortunately we did not hit the same accuracies as the MLP in the paper, but that is to be expected as the paper is rather recent

Classifier	Sensitivity	Specificity	Accuracy	AUC
MLP[a]	90.3%	86%	87.8%	0.88
Multipass LVQ[b]	69.4%	93%	83.1%	0.81
LVQ[c]	81.9%	75%	78%	0.78
SVM[d]	70.8%	88%	80.8	0.79
KNN[e]	33.3%	97%	70.3%	0.65

a: multilayer perceptron,
b: multipass learning vector quantization,
c: learning vector quantization,
d: support vector machine,
e: k nearest neighbors.

https://doi.org/10.1371/journal.pone.0250904.t002

In [ ]: