Predicting neurological outcome in patients with a severe postanoxic encephalopathy

Primary Topic: Data Mining, Secondary Topic: Time Series
Course: 2021-1B – Group: 89 – Submission Date: 2021-01-31

Adam Meijer
University of Twente
a.j.meijer@student.utwente.nl

Liselot Goris
University of Twente
l.c.goris@student.utwente.nl

ABSTRACT

Every year 176,000 patients in Europe are admitted to the intensive care unit with a postanoxic coma after cardiac arrest, 40% of these patients will progress into a vegetative state. Early prediction of good or poor neurological outcome is important to provide the right care for the patients. In this study a machine learning model is designed to predict poor or good outcome (based on the cerebral performance category score) from the 12 hours and 24 hours EEG data of patients after cardiac arrest. Several classifiers were evaluated, the random forest classifier gave the best accuracy for this dataset. In order to improve the accuracy of the model hyperparameter tuning was applied. The k-best chi-square score was used for feature selection. For the 12 hours data the highest accuracy was reached with 26 features and for the 24 hours data the highest accuracy was reached with 35 features selected reaching an AUC score of 0.854. For poor outcome the sensitivity was 33% at a specificity of 100%, for good outcome the sensitivity was 34% at a specificity of 95%. The 24 hours data did not have a significantly better outcome. It was concluded that machine learning in combination with qEEG is a promising tool for predicting good and poor outcome of comatose patients after cardiac arrest, however the model should be further improved before it can be used in practice.

KEYWORDS

qEEG features, machine learning, postanoxic coma

1 INTRODUCTION

Every year 176,000 patients in Europe are admitted to the Intensive Care Unit with a postanoxic coma after cardiac arrest. 80% of the patients who initially survive cardiac arrest are comatose, 40% of these patients will progress into a vegetative state [7]. Early prediction of neurological outcome is vital to prevent futile treatment and to be able to provide the right care for patients with a high probability of good recovery.
Electroencephologram (EEG) is the standard technique to determine cerebral activity. Machine learning might be helpful to predict the outcome of the patient's health. The goal of this project is to fit a model to the EEG data to predict the patient's health outcome based on EEG features by using machine learning classifiers. The research question would be: 'How accurate is the machine learning model to predict poor or good outcome from the EEG data of patients with a postanoxic coma after cardiac arrest?'
The first goal is to find the right classifier for the data, thereafter it will be investigated which features need to be selected to get the highest accuracy in predicting good or poor outcome. The last goal is to find out how the prediction changes when EEG data of different hours after cardiac arrest is used, by evaluating the differences between the area under the curve (AUC) of the 12 hours data and the 24 hours data.

2 BACKGROUND

In this project the dataset is evaluated on various quantitative EEG (qEEG) features to predict the neurological outcome. The Cerebral Performance Category Score (CPC) is a score to evaluate the patient's neurological outcome. Good outcome is characterised by a CPC of 1 (good cerebral performance) and 2 (moderate cerebral disability), and bad outcome by a CPC of 3 (severe cerebral disability), 4 (coma or vegetative state) and 5 (brain death) [3].

Several earlier studies have been working on predicting neurological outcome by using qEEG features and machine learning. Tjepkema et al. (2013) [9] combined five qEEG features into a single number, the Cerebral Recovery Index (CRI) to predict the neurological outcome at 24 hours after cardiac arrest, this study came to a sensitivity of 55% for poor outcome and 25% for good outcome at a specificity of 100%. With this study it was shown that 'Quantitative EEG analysis can reduce the time needed to review long-term EEG and makes the analysis more objective.' [9] In a later study Tjepkema et al (2017) [8] used nine qEEG features for the prediction of neurological outcome. In this study a sensitivity of 56% and 65% for poor outcome and 63% and 58% for good outcome was reached, at a specificity of 100% at 12 hours and 24 hours after cardiac arrest. [8]
A year later, Nagaraj et al (2018) [5] used forty-four qEEG features in combination with a revised CRI and predicted poor outcome with a sensitivity of 66% and 60%, and good outcome with a sensitivity of 72% and 40%, at 12 hours and 24hours after cardiac arrest with a specificity of 95%. [5]
The results of the studies mentioned above suggest that it is possible to efficiently monitor patient outcome after cardiac arrest with qEEG features and machine learning algorithms.

3 APPROACH

3.1 Description of dataset

Two EEG datasets are used, one of 12 hours after cardiac arrest and one of 24 hours after arrest. The datasets were provided by Prof. Dr. ir. Michel J.A.M. van Putten. The datasets consist of the neurological outcome of patients given by the CPC score, and 44 different EEG features, a description is given in table 1. The CPC score is grouped into two categories, good (CPC scores 1 and 2, denoted as 1) and poor (CPC scores 3-5, denoted as 0). The features are either time

	Feature name	Domain		Feature name	Domain
1	Patient Outcome		24	Beta_tot	Freq
2	Nonlinear Energy	Time	25	Alpha_delta	Freq
3	Activity	Time	26	Theta_delta	Freq
4	Mobility	Time	27	Spindle_delta	Freq
5	Complexity	Time	28	Beta_delta	Freq
6	RMS Amplitude	Time	29	Alpha_theta	Freq
7	Kurtosis	Time	30	Spindle_theta	Freq
8	Skewness	Time	31	Beta_theta	Freq
9	Mean AM	Time	32	Fhtife1	Freq
10	Std AM	Time	33	Fhtife2	Freq
11	Skew AM	Time	34	Fhtife3	Freq
12	Kurt AM	Time	35	Fhtife4	Freq
13	BSR	Time	36	Sef	Freq
14	Delta	Freq	37	Df	Freq
15	Theta	Freq	38	Svd_ent	Entropy
16	Alpha	Freq	39	H_spec	Entropy
17	Spindle	Freq	40	SE	Entropy
18	Beta	Freq	41	Saen	Entropy
19	Total	Freq	42	Abs(renyi)	Entropy
20	Delta_tot	Freq	43	Abs(shan)	Entropy
21	Theta_tot	Freq	44	Perm_entr	Entropy
22	Alpha_tot	Freq	45	FD	Entropy
23	Spindle_tot	Freq			

Table 1: qEEG features and domain

domain features, frequency domain features or entropy domain features. The data is normalised before the classifiers are applied, using the standard normalisation method:

$$\text{Normalised data} = \frac{data - data_{min}}{data_{max} - data_{min}}$$

3.2 Explanation of classifiers

The datasets are processed using Python. To decide which classifier to use to evaluate the data, different classifiers are tested and the results are shown in a ROC curve. The classifiers used are:

- Decision Tree Classifier: uses a decision tree to go from observations to a conclusion. The tree root is the start, where the data is split on the feature that results in the largest information gain. This process is repeated until the leaves are pure. Overfitting should be prevented, which sometimes results in impurity of the final leaves.
- Decision Tree plus Ada Boost Classifier: The AdaBoost detects the weak points in the decision tree and fits a new decision tree to improve performance [6].
- Support Vector Classifier (SVC): each data item is plotted as a point in n-dimensional space (n is the number of features). Classification is performed by finding the hyper-plane that differentiates the two classes very well. The support vectors are the coordinates of individual observations. The Support Vector Classifier is the borderline which best segregates the two classes. The classification of the SVC is accurate however it suffers from a large amount of computation [4].
- Multi-layer Perceptron classifier: a classifier that relies on an underlying Neural Network to perform the task of classification, in this case with 100 neurons in 100 hidden layers, an ADAM optimiser and a ReLU activation function.
- Random forest classifier: a random forest is a combination of decision trees where each tree depends on the values of a random vector sampled independently, and with the same distribution for all trees in the forest. A random forest corrects for overfitting of the training set, which is an advantage over the Decision Tree Classifier [1].

3.3 Feature selection

A random state of 42 is used for the random forest classifier and when splitting the data into test and training sets.

Step 1: Hyperparameters. In order to tune the random forest classifier, hyperparameter tuning is applied. With this technique the training data is split into cross validation values. After splitting the datasets, the parameters of the classifier will be randomly tested inside a range, and the parameters that give the highest accuracy will be obtained. The hyperparameters that are used to tune the model are:

- Number of decision trees (n estimators)
- Maximum depth of the tree (max dept)
- Splitting criteria (criteria)
- Minimum number of samples required to be at a leaf node (min samples leaf)
- Minimum number of samples required to split an internal node (min samples split)
- Number of features to consider when looking for the best split (max features)

To get the best parameters, a Random Hyperparameter Grid technique is be applied. The python SKlearn model called RandomizedSearchCV is used for this. The cross validation value is set to 2 and the number of iterations to 200. To avoid overfitting, the decision tree is pruned, the value for max dept of the decision tree is set to 15, higher values cause overfitting.

Step 2: Feature selection using k-best chi-square. Since the dataset contains 44 different features a feature selection method is applied to select the best features for classification. For each feature a chi-square score is calculated and sorted from high to low. Thereafter, these features are used to train the random forest classifier. Starting with only the best feature, following with the best and second best, up to selecting all features. For each iteration the accuracy is measured to display how many features, ordered on chi-square score, give the highest accuracy.

4 EXPERIMENTS

4.1 Results of different classifiers (ROC curves)

The ROC curves of the five different classifiers is shown in figure 1. The area under the curve (AUC) is 0.72 for the decision tree classifier, 0.83 for the SVC, 0.83 for the MLP classifier, 0.84 for the

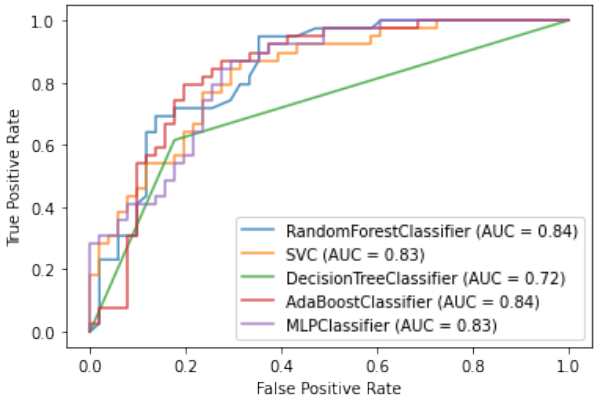

Figure 1: ROC curves of different classifiers

AdaBoostClassifier and 0.91 for the random forest classifier. Based on these results we choose to use the Random Forest Classifier.

4.2 Results of feature selection

Based on the results of the hyperparameter technique, optimal values for the random forest classifier are found for both 12 hour and 24 hour data separate. Hyperparameter tuning improved the accuracy only by a small fraction, but it gave good ranges about which parameters to change and by how much. To get the optimal results the parameters are further tuned by hand, resulting in the following values. For the 12 hour data, the criterion is set to entropy. The minimal samples in a leaf to 1 and the minimal samples to split an internal node is set to 15. The max dept of a tree is set to 15, the number of decision trees is set to 150 and the maximal features is set to log2. For the 24 hour data, the number of decision trees, max dept of a tree, maximal features, minimal samples in a leaf and minimal samples to split a node are set to the same values of the 12 hour data. Only one parameter differs, the criterion is set to gini index. With these parameters the accuracy of the random forest classifier improved from 78% till 81%.

Figure 2 shows graphs of the k-best features for the random forest classifier. For both 12 hour and 24 hour data the accuracy does not exceed 80%. With 26 features selected the highest accuracy is achieved for the 12 hour data, see table 2. The highest accuracy for the 24 hours data is achieved with 35 features selected, see table 3.

4.3 Results of 12 hours and 24 hours data

For the 12 hour data the AUC value is 0.854 and for 24 hour data the AUC is 0.855. Looking at figure 4, for poor outcome at a 100% specificity, the sensitivity is 33%. For good outcome, shown in figure 3, at a specificity 95% the sensitivity is 34%.

Looking at figure 6, for the 24 hour data and a poor outcome at 100% specificity, the sensitivity is 13% and for good outcome, shown in figure 5, at a specificity 95% the sensitivity is 38%.

These ROC curves were obtained with the random forest classifier, by combining chi-square feature selection and hyperparameter tuning. For the 12 hour data 26 features were selected and for the 24 hour data 35 features were selected.

	Feature name		**Feature name**
1	BSR	24	SkewAM
2	MeanAM	25	Beta_tot
3	Activity	26	Abs(renyi)
4	Total		
5	Nonlinear Energy		
6	Alpha		
7	stdAM		
8	KurtAM		
9	Theta		
10	Abs(shan)		
11	Delta		
12	RMS Amplitude		
13	Fhtife4		
14	Beta_theta		
15	Kurtosis		
16	Beta_delta		
17	Fhtife2		
18	Df		
19	Fhtife1		
20	Spindle		
21	Spindle_delta		
22	Spindle_theta		
23	Skewness		

Table 2: Highest chi-square features of 12 hour data

	Feature name		**Feature name**
1	stdAM	24	Beta
2	BSR	25	Beta_tot
3	Abs(shan)	26	Alpha_theta
4	KurtAM	27	Abs(renyi)
5	Kurtosis	28	Theta_delta
6	Theta	29	Theta_tot
7	Beta_theta	30	Mobility
8	Fhtife4	31	Spindle_tot
9	MeanAM	32	H_spec
10	Skewness	33	FD
11	RMS Amplitude	34	Alpha_delta
12	SkewAM	35	Fhtife3
13	Total		
14	Fhtife2		
15	Activity		
16	Beta_delta		
17	Spindle_theta		
18	Alpha		
19	Nonlinear energy		
20	Delta		
21	Fhtife1		
22	Spindle_delta		
23	Df		

Table 3: Highest chi-square features of 24 hour data

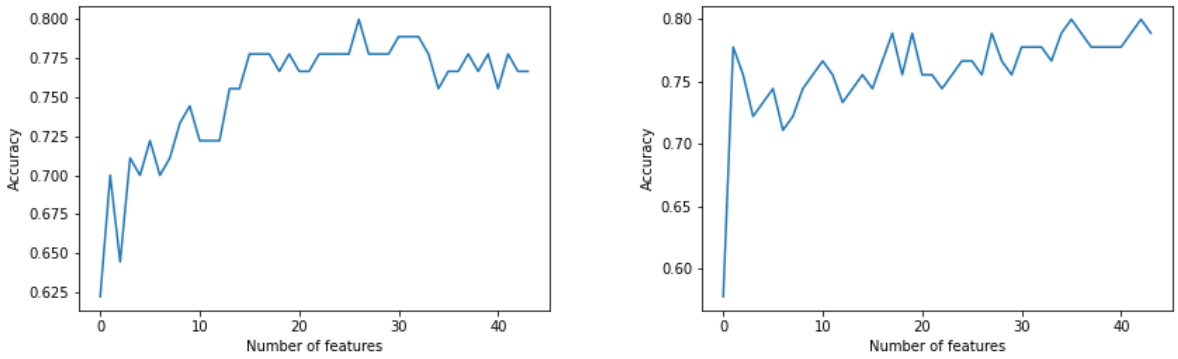

Figure 2: Graph of the accuracy prediction for 12 hours data (left) and 24 hours data (right) on the k-best chi-square selected features

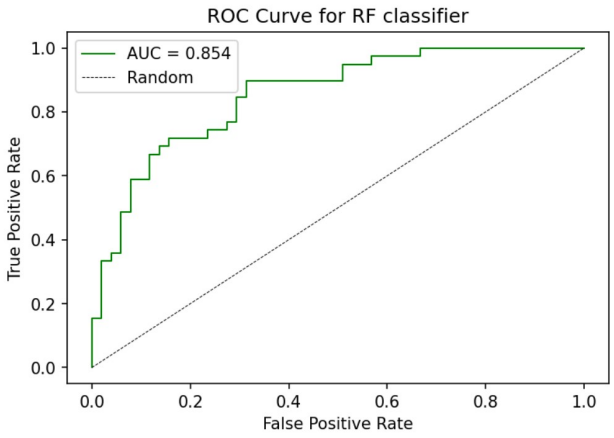

Figure 3: ROC curve of good outcome, 12hour data

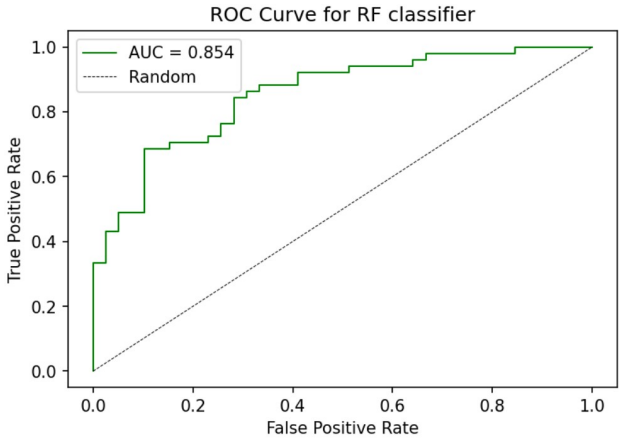

Figure 4: ROC curve of poor outcome, 12hour data

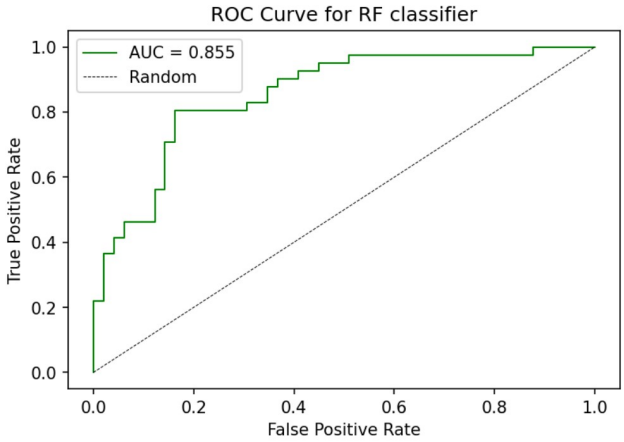

Figure 5: ROC curve of good outcome, 24hour data

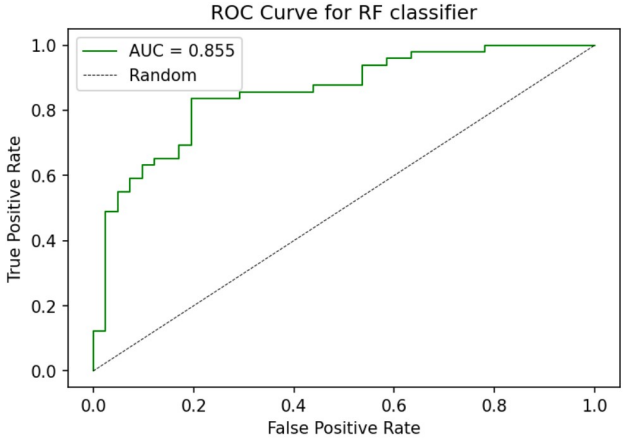

Figure 6: ROC curve of poor outcome, 24hour data

5 DISCUSSION

The experiments show that the random forest classifier performs best with our dataset (AUC = 0.84), the random forest model minimises overfitting. It must be said that the performances of the AdaBoost, SVC and the MLP were very close to the random forest. The fact that the performances of these classifiers were so close to each other around an AUC of 0.83/0.84 most likely means that the data has a certain amount of variance that cannot be overcome by these classifiers. More data would help to reduce variance and a classifier with more predictive power could improve the performance as well.

The results of our model show a relatively low sensitivity (about 34%) with a specificity of 100%, the sensitivity should be improved before the model can be used in practice. Improvements can come from training the model with more data or give weights to certain features that are more important. Earlier studies ([5], [8], [9]) combined qEEG features in a single number, the Cerebral Recovery Index (CRI), to quantify and grade EEG data of patients after cardiac arrest and assist in prediction of both poor and good outcome. A similar number like the CRI could be used which is calculated by giving qEEG features different weights depending on their importance for predicting neurological outcome, and thereby increasing the accuracy of the prediction.

The AUC score for 12 hour data and 24 hour data is roughly the same, respectively 0.854 and 0.855. An AUC performance of 0.85 is pretty decent, a value between 0.80 and 0.90 is seen as good [2]. The highest AUC score is gained by combining the chi-square feature selection method with hyperparameter tuning of the random forest classifier. The hyperparameter tuning gave an accuracy of 81%. This was an increase of 3% relative to the standard parameters. Adding feature selection increased the AUC with 0.01. However, with only 20 features selected the results are already close to maximal performance. Therefore, it is not fully necessary to use all 44 features, since they improve the prediction with a very small fraction. Especially when it is complicated and more expensive to obtain certain features, it is sufficient to only use 20 features and already get good results.

Looking at the 20 highest chi-square features, the feature called stdAM is clearly the best feature. This is a time domain feature, the time domain features predominate the the top 20. The most important entropy feature is the calculated Shannon entropy (Abs(shan)) and the most important frequency feature is the calculated beta theta. The most prominent difference between both highest scoring features of the highly important features calculated with chi-square and those calculated with random forest, is that the stdAM is by far the most important chi-square feature in the 24 hour data, but it does not score high for random forest. It is likely that the improvement of the AUC with chi-square feature selection goes along with selecting this feature as important, since the other features largely correspond. For the 12 hour data the time domain feature meanAM is very important and does not score high for random forest.

It can be concluded that making an EEG 12 hours after cardiac arrest is sufficient to predict neurological outcome, an extra EEG after 24h is not necessary, since the performance of both datasets are roughly the same. Predicting the outcome in an early stage is favourable for the patient and his/her loved ones, as it will sooner be clear what the outcome will be, and care for the patient can be adjusted based on the outcome.

6 CONCLUSION

In this project a random forest classifier model was used to predict neurological outcome in comatose patients after cardiac arrest. The model was tuned by selecting the hyperparameters that gave the highest accuracy. After k-best chi-square feature selection it could be concluded that only after selecting 26 features in the 12 hour data the highest accuracy was already achieved. For the 24 hours data this accuracy was reached with 35 features. By combining hyperparameter tuning and feature selection, the highest accuracy was reached, the model accuracy was roughly the same for the 12 hours and 24 hours data. All in all, this study gives promising results for the use of machine learning for predicting good and poor outcome of comatose patients after cardiac arrest based on EEG, however the model should be further improved before it can be used in practice.

REFERENCES

[1] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[2] Thomas G and MD Tape. 2001. *Interpreting Diagnostic Tests*. University of Nebraska Medical Center. https://doi.org/10.7326/0003-4819-135-1-200107030-00043

[3] Erich L Kiehl, Alex M Parker, Ralph M Matar, Matthew F Gottbrecht, Michelle C Johansen, Mark P Adams, Lori A Griffiths, Steven P Dunn, Katherine L Bidwell, Venu Menon, et al. 2017. C-GRA pH: A Validated Scoring System for Early Stratification of Neurologic Outcome After Out-of-Hospital Cardiac Arrest Treated With Targeted Temperature Management. *Journal of the American Heart Association* 6, 5 (2017), e003821.

[4] KW Lau and QH Wu. 2003. Online training of support vector classifier. *Pattern Recognition* 36, 8 (2003), 1913–1920.

[5] Sunil B Nagaraj, Marleen C Tjepkema-Cloostermans, Barry J Ruijter, Jeannette Hofmeijer, and Michel JAM van Putten. 2018. The revised Cerebral Recovery Index improves predictions of neurological outcome after cardiac arrest. *Clinical neurophysiology* 129, 12 (2018), 2557–2566.

[6] Robert E Schapire. 2013. Explaining adaboost. In *Empirical inference*. Springer, 37–52.

[7] Mario Stanziano, Carolina Foglia, Andrea Soddu, Francesca Gargano, and Michele Papa. 2011. Post-anoxic vegetative state: imaging and prognostic perspectives. *Functional neurology* 26, 1 (2011), 45.

[8] Marleen C Tjepkema-Cloostermans, Jeannette Hofmeijer, Albertus Beishuizen, Harold W Hom, Michiel J Blans, Frank H Bosch, and Michel JAM Van Putten. 2017. Cerebral recovery index: reliable help for prediction of neurologic outcome after cardiac arrest. *Critical care medicine* 45, 8 (2017), e789–e797.

[9] Marleen C Tjepkema-Cloostermans, Fokke B van Meulen, Gjerrit Meinsma, and Michel JAM van Putten. 2013. A Cerebral Recovery Index (CRI) for early prognosis in patients after cardiac arrest. *Critical care* 17, 5 (2013), 1–11.

A PYTHON SCRIPT

```
# -*- coding: utf-8 -*-
"""Programming_Project_EEG

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/17cSeAj0ecueWVKIvAP66gGIPj8ZR0O-n
"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
from itertools import cycle
import seaborn as sns

# sklearn
from sklearn import datasets
from sklearn import metrics
from sklearn.metrics import RocCurveDisplay
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import plot_roc_curve
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc, precision_recall_curve, average_precision_score
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.ensemble import ExtraTreesClassifier

from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV

#from accstats import confusion_matrix

# matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import cnames

# %matplotlib inline

from google.colab import drive
drive.mount('/content/drive')

"""Loading data"""

data12 = pd.read_csv('/content/drive/MyDrive/Data_Science/Project/featuresNEW_12hrs.csv', sep=';', decimal=',')
data24 = pd.read_csv('/content/drive/MyDrive/Data_Science/Project/featuresNEW_24hrs.csv', sep=';', decimal=',')
data_combined = pd.concat([data12, data24])

#Normalize data
normalized_data12 = (data12-data12.min())/(data12.max()-data12.min())
normalized_data24 = (data24-data24.min())/(data24.max()-data24.min())
normalized_data_combined = (data_combined-data_combined.min())/(data_combined.max()-data_combined.min())

#select the data to be used, normalized_data12, normalized_data24, normalized_data_combined
data = normalized_data12

outcome_y = data['Patient Outcome']
features_X = data.loc[:, data.columns != 'Patient Outcome']
target_names = list(data.columns)

#features_X['Nonlinear energy']

# train test split the data (X, y)
X_train, X_test, y_train, y_test = train_test_split(features_X, outcome_y, test_size=0.3, random_state=42) #, random_state=42
```

```

#some confidence interval
#sns.lineplot(data=data, ci=95)

"""Fit different classifiers to the data"""

# fit random forest classifier
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)

# fit SVC classifier
svc = SVC(random_state=42)
svc.fit(X_train, y_train)

# fit Decision tree classifier
dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)

# fit AdaBoost classifier
adb_clf = AdaBoostClassifier(random_state=42)
adb_clf.fit(X_train, y_train)

# fit MLP classifier
mlp_clf = MLPClassifier(random_state=42)
mlp_clf.fit(X_train, y_train)

y_pred = rf_clf.predict(X_test)

"""ROC curves of the data using different classifiers"""

""" Straightforward ROC curve """
ax = plt.gca()
rfc_disp = plot_roc_curve(rf_clf, X_test, y_test, ax=ax, alpha=0.8)
svc_disp = plot_roc_curve(svc, X_test, y_test, ax=ax, alpha=0.8)
dt_clf = plot_roc_curve(dt_clf, X_test, y_test, ax=ax, alpha=0.8)
adb_clf = plot_roc_curve(adb_clf, X_test, y_test, ax=ax, alpha=0.8)
mlp_clf = plot_roc_curve(mlp_clf,X_test, y_test, ax=ax, alpha=0.8)
plt.show()

print(rfc_disp)

"""Hyperparameter tuning of the random forest classifier"""


from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num = 20)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt', 'log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [5, 10, 15]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4, 6, 8]
# Method of selecting samples for training each tree
bootstrap = [True, False]

criterion = ['gini', 'entropy']

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': [5, 10],
               'min_samples_split':
                   min_samples_split,
               'min_samples_leaf':
                   min_samples_leaf,
               'bootstrap': bootstrap,
               'criterion': criterion}
print(random_grid)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 2 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_hyp = RandomForestClassifier(random_state=42)

rf_random = RandomizedSearchCV(estimator = rf_hyp, param_distributions = random_grid, n_iter = 200, cv = 2, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
151
152 rf_random.best_params_
153
154 """Creating Classification Reports"""
155
156 def evaluate(model, test_features,
        test_labels):
157     predictions = model.predict(
            test_features)
158     CM = confusion_matrix(test_labels,
            predictions)
159     CR = classification_report(test_labels,
            predictions)
160     print('Performance')
161     print(CM)
162     print(CR)
163     return
164
165 rf_clf.fit(X_train, y_train)
166 base_accuracy = evaluate(rf_clf, X_test,
        y_test)
167
168 best_random = rf_random.best_estimator_
169 print(rf_random.best_params_)
170 random_accuracy = evaluate(best_random,
        X_test, y_test)
171
172 rf_best = RandomForestClassifier(
        n_estimators=150, min_samples_leaf=1,
        min_samples_split=15, max_depth=10,
        max_features='log2', criterion='entropy'
        , random_state=42)
173 rf_best.fit(X_train, y_train)
174 best_accuracy = evaluate(rf_best, X_test,
        y_test)
175
176 from sklearn.ensemble import
        RandomForestClassifier
177 rf_best = RandomForestClassifier(
        n_estimators=150, min_samples_leaf=1,
        min_samples_split=15, max_depth=15,
        max_features='log2', criterion='entropy'
        , random_state=42)
178 rf_best.fit(X_train, y_train)
179
180 # get the probability distribution
181 probas = rf_best.predict_proba(X_test)
182 # get false and true positive rates
183 fpr, tpr, _ = roc_curve(y_test, probas[:,1],
         pos_label=1)
184
185 # get area under the curve
186 roc_auc = auc(fpr, tpr)
187
188 # PLOT ROC curve
189 plt.figure(dpi=150)
190 plt.plot(fpr, tpr, lw=1, color='green',
        label=f'AUC = {roc_auc:.3f}')
191 plt.plot([0,1], [0,1], '--k', lw=0.5, label=
        'Random')
192 plt.title('ROC Curve for RF classifier')
193 plt.xlabel('False Positive Rate')
194 plt.ylabel('True Positive Rate')
195 plt.xlim([-0.05, 1.05])
196 plt.ylim([-0.05, 1.05])
197 plt.legend()
198 plt.show()
199
200 # get the probability distribution
201 probas = rf_best.predict_proba(X_test)
202 # get false and true positive rates
203 fpr, tpr, _ = roc_curve(y_test, probas[:,0],
         pos_label=0)
204
205 # get area under the curve
206 roc_auc = auc(fpr, tpr)
207
208 # PLOT ROC curve
209 plt.figure(dpi=150)
210 plt.plot(fpr, tpr, lw=1, color='green',
        label=f'AUC = {roc_auc:.3f}')
211 plt.plot([0,1], [0,1], '--k', lw=0.5, label=
        'Random')
212 plt.title('ROC Curve for RF classifier')
213 plt.xlabel('False Positive Rate')
214 plt.ylabel('True Positive Rate')
215 plt.xlim([-0.05, 1.05])
216 plt.ylim([-0.05, 1.05])
217 plt.legend()
218 plt.show()
219
220 """Chi-square feature selection"""
221
222 #apply SelectKBest class to extract top 10
        best features
223 max_features = len(features_X)
224
225 X = abs(X_train)  #independent columns
226 y = y_train    #target column i.e price
        range
227
228 feature_list = []
229 for i in range(1, 45):
```

```
    #apply SelectKBest class to extract top 10 best features
    bestfeatures = SelectKBest(score_func=chi2, k=i)
    fit = bestfeatures.fit(X,y)
    dfscores = pd.DataFrame(fit.scores_)
    dfcolumns = pd.DataFrame(X.columns)

    #concat two dataframes for better visualization
    featureScores = pd.concat([dfcolumns, dfscores],axis=1)
    featureScores.columns = ['Specs','Score']  #naming the dataframe columns

    best_features = featureScores.nlargest(i,'Score').Specs
    #print(X_train[best_features])
    rf_clf.fit(X_train[best_features], y_train)
    y_pred = rf_clf.predict(X_test[best_features])

    CR_dict = classification_report(y_test, y_pred, output_dict=True)
    feature_list.append(CR_dict['accuracy'])

best_feature_n = feature_list.index(max(feature_list))
print('Number of features for best performance:', best_feature_n)

plt.plot(feature_list)
plt.xlabel('Number of features')
plt.ylabel('Accuracy')
#plt.savefig('k-best_24hour.png')
plt.show()

n_features = best_feature_n

print(featureScores.nlargest(n_features,'Score'))  #print 10 best features
best_features = featureScores.nlargest(n_features,'Score').Specs
#print(X_train[best_features])

"""ROC curves good and bad outcome, feature selection and hyperparameters combined


"""

rf_best = RandomForestClassifier(n_estimators=150, min_samples_leaf=1, min_samples_split=15, max_depth=15, max_features='log2', criterion='entropy', random_state=42)
rf_best.fit(X_train[best_features], y_train)

# get the probability distribution
probas = rf_best.predict_proba(X_test[best_features])
# get false and true positive rates
fpr, tpr, _ = roc_curve(y_test, probas[:,1], pos_label=1)

# get area under the curve
roc_auc = auc(fpr, tpr)

# PLOT ROC curve
plt.figure(dpi=150)
plt.plot(fpr, tpr, lw=1, color='green', label=f'AUC = {roc_auc:.3f}')
plt.plot([0,1], [0,1], '--k', lw=0.5, label='Random')
plt.title('ROC Curve for RF classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.legend()
plt.show()

# get the probability distribution
probas = rf_best.predict_proba(X_test[best_features])
# get false and true positive rates
fpr, tpr, _ = roc_curve(y_test, probas[:,0], pos_label=0)

# get area under the curve
roc_auc = auc(fpr, tpr)

# PLOT ROC curve
plt.figure(dpi=150)
plt.plot(fpr, tpr, lw=1, color='green', label=f'AUC = {roc_auc:.3f}')
plt.plot([0,1], [0,1], '--k', lw=0.5, label='Random')
plt.title('ROC Curve for RF classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
```

```
plt.legend()
plt.show()

"""Feature importance"""

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(18).plot(kind='barh')
plt.show()
```