Schwerpunktspraktikum Teilchenphysik und Detektoren

Analysis of IceCube Data

Felix Wersig
felix.wersig@tu-dortmund.de

Noah Behling
noah.behling@tu-dortmund.de

Durchführung: 25.04.2022 Abgabe: 09.05.2022

TU Dortmund – Fakultät Physik

Inhaltsverzeichnis

1. Introduction

In this lab course neutrino selection with Monte-Carlo-simulated data from the IceCube neutrino observatory is done. To efficiently select neutrinos the data first need to be preprocessed. Afterwards a selection of attributes will be used for separation via machine learning algorithms.

2. Theory

In this Chapter a brief introduction to the standard model of particle physics (SM) and its important aspects regarding cosmic rays is given.

2.1. The Standard Model of Particle Physics

The SM [7] describes the currently known elementary particles that all matter consists of and all fundamental interactions except gravity. The elementary particles are divided into fermions and bosons. Fermions have spin 1/2 and can be further divided into quarks and leptons. There are six quarks called *up* (u), *down* (d), *charm* (c), *strange* (s), *top* (t) and *bottom* (b). These fall into three generations

$$\begin{pmatrix} u \\ d \end{pmatrix}, \begin{pmatrix} c \\ s \end{pmatrix}, \begin{pmatrix} t \\ b \end{pmatrix}.$$

The *up-type* quarks u, c and t have an electric charge $Q = \frac{2}{3}$ while the *down-type* quarks d, s and b have an electric charge $Q = -\frac{1}{3}$. Additionally, quarks come in three different color charges red, green, and blue and their respective anticolours.
Also the six leptons are categorized into three generations consisting of one of the charged leptons electron, muon and tau and their respective left-handed neutrino:

$$\begin{pmatrix} e^- \\ \nu_e \end{pmatrix}, \begin{pmatrix} \mu^- \\ \nu_\mu \end{pmatrix}, \begin{pmatrix} \tau^- \\ \nu_\tau \end{pmatrix}.$$

The electric charge of the charged leptons is -1 while neutrinos are neutral.
Each class of fermions is divided into generations since they show the same physical behaviour, but have a higher mass in comparison to lower generations.

Bosons have an integer spin and are distinguished between gauge bosons and the Higgs boson. Gauge bosons have spin 1 and are mediators of the fundamental interactions. Photons mediate the electromagnetic interaction and couple to electric charge, gluons the mediators of the strong interaction and couple to color charge and the $W^\pm$ and the Z^0 bosons mediate the weak interaction.
The Higgs however has a spin 0 and does not act as a mediator. It is the newest addition to the SM and was discovered 2012 at the LHC[1].

2.2. Cosmic Rays

The earth is constantly bombarded with highly energetic particles. These cosmic rays (CR) mostly consist of protons ($\approx 85\,\%$) and α-particles ($\approx 10\,\%$), but also heavy nuclei and electrons can be observed. This radiation can achieve energies up to $10^{20}\,\mathrm{eV}$ and the energy spectrum can approximately be described by a power law

$$\frac{\mathrm{d}\Phi}{\mathrm{d}E} = \Phi_0 E^{\gamma}, \tag{1}$$

where $\gamma \approx -2.7$ is the spectral index.
Due to the electric charge of the CR particles they are deflected in the galactic and extragalactic magnetic fields resulting in an isotropic flux from all directions.

The muons and neutrinos detected by IceCube can come from either atmospheric or astrophysical sources. First can be separated into prompt and conventional muons and neutrinos regarding their energetic spectrum. Conventional muons and neutrinos come from pions and kaons formed when the primary CRs interact with the earth's atmosphere. Due to their comparatively large lifetimes these particles lose energy before decaying into muons and neutrinos resulting in a energy spectrum $\propto E^{-3.7}$. But at high energies also D mesons and Λ_c baryons are produced. These have very short lifetimes and therefore do not lose any energy before decaying. This leads to an energy spectrum $\propto E^{-2.7}$.
It is expected that cosmological sources of accelerated hadrons also emit photons and neutrinos. These are not deflected in the galactic and extragalactic magnetic fields due to their lack of electric charge resulting in their arrival directions pointing back to their sources. Due to the small cross section of neutrinos, they are also able to pass through dense dust clouds and other optically dense media which would absorb the emitted photons.
Under the assumption of shock acceleration [5], a spectral index $\gamma \approx 2$ is expected.

3. The IceCube Neutrino Observatory

Consisting of the In-Ice Array [4], IceTop [2] and DeepCore [3], the IceCube experiment is used to study neutrinos and muons. It is located at the Amundsen-Scott South Pole Station near the geographical south pople. The experiment consists of 5160 digital optical modules (DOMs) on 86 strings located 1450 m–2450 m under the surface as seen in Abbildung 1. The DOMs are photomultipliers (PMTs) in a vacuumised sphere of glass. These are used to detect the Cherenkov radiation emitted by muons, electrons and tau leptons. Cherenkov radiation is emitted when a relativistic particle travels through a medium with a speed greater than the speed of light in the medium

$$c_n = \frac{c_0}{n},$$

where c_0 is the speed of light in the vaccum and n is the refractive index of the medium. The seven strings of DeepCore are as well as their PMTs are spaced more densely, resulting

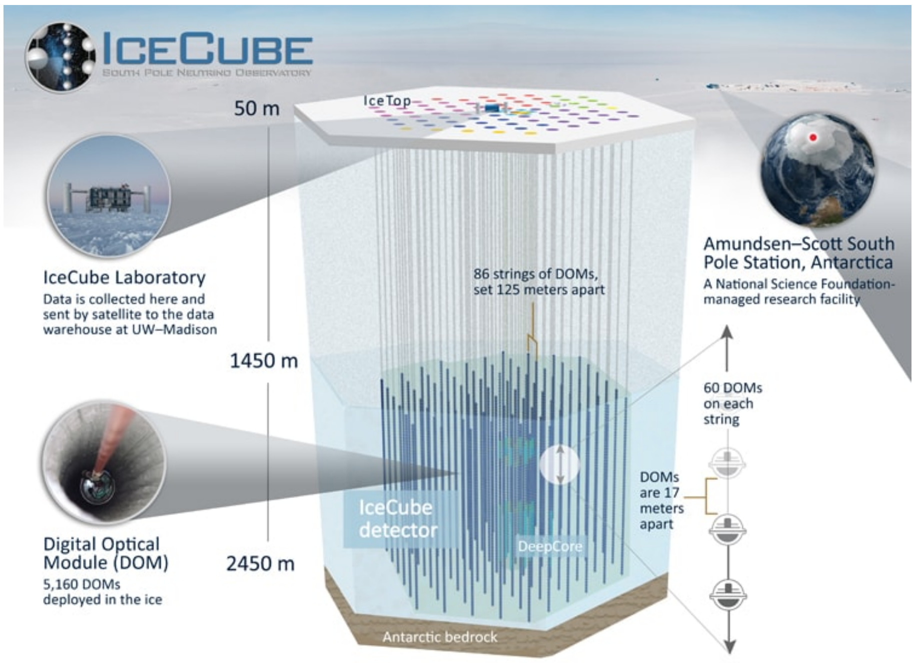
ICECUBE
SOUTH POLE NEUTRINO OBSERVATORY
50 m
IceTop
IceCube Laboratory
Data is collected here and sent by satellite to the data warehouse at UW–Madison
86 strings of DOMs, set 125 meters apart
Amundsen–Scott South Pole Station, Antarctica
A National Science Foundation-managed research facility
1450 m
60 DOMs on each string
DOMs are 17 meters apart
IceCube detector
DeepCore
Digital Optical Module (DOM)
5,160 DOMs deployed in the ice
2450 m
Antarctic bedrock

Abbildung 1: Schematic of the IceCube Experiment[6].

in a lower energy threshold of 10 GeV in contrast to a 100 GeV energy threshold of the rest of the In-Ice Arry. With the IceTop experiment located at the surface of the south pole air showers are studied. It also functions as a veto for the In-Ice Array. Neutrinos are measured indirectly via the charged (CC) and neutral currents (NC)

$$\nu_l(\bar{\nu}_l) + A \rightarrow l^{\pm} + X \qquad \text{(CC)}$$
$$\nu_l(\bar{\nu}_l) + A \rightarrow \nu_l + X \qquad \text{(NC)}$$

where A is a particle the neutrino interacts with and X is a resulting particle shower. Since electrons are highly ionising, they quickly deposit all their energy resulting in a sphere of Cherenkov radiation. The same can be observed for tau leptons because of their short lifetime. Therefore it cannot be distinguished between electron and tau neutrinos, except for the energy of a tau neutrino exceeding 1 PeV since then the cascade of the CC interaction and that of the tau lepton decay are spatially separated. The signature of NC interactions of all lepton flavors induced by the hadron shower is also similar to that of electrons in the CC channel. Cherenkov radiation emitted by muons is too weak to be detected, but a muon produces e^+e^- pairs along its way emitting detectable Cherenkov radiation. This results in a cone-like signature for muons.

For rejection of atmospheric muons, so called *starting events* are used. Here the outer layers of the detector are used as a veto leading to a smaller effective detector volume, where all neutrino flavors contribute in the same way, meaning that a majority of events

are NC processes and CC electron and tau neutrino interactions. These have good energy resolution, but lack a good angle resolution.
Muon tracks have a worse energy resolution, but a higher angle resolution and depending on their energy a higher range allowing to use the earth as shielding, because only muons from neutrino interactions can come from below the detector. To reject atmospheric muons, a cut on the zenith angle can be performed also leading to an enhanced detector volume since also muons from neutrino interactions outisde the detector can be considered. Assuming a perfect reconstruction algorithm, that cut would reject all atmospheric muons. While this is not the case, the cut on the zenith angle still enhances the signal to background ratio from $1 : 10^6$ to $1 : 10^3$. To further improve background rejection machine learning algorithms can be utilised, which is the aim of this analysis.

4. Multivariate Selection and Machine Learning

To save computation time, it is useful to study only a subset of the attributes available since not all of them contain the same information value. A method to find a subset of valuable attributes while limiting computation time is forward selection, where iteratively the attribute is added to the model that results in the best prediction in combination with the attributes already chosen. Hereby usually an F-test is performed to evaluate the quality of the prediction. it is noteworthy that the selection of attributes is not objective, but depends on the learning algorithm that is used.
The Jaccard index is a measure of the stability of the attribute selection against statistical fluctuations in the learning algorithm. It is defined as

$$J(F_a, F_b) = \frac{|F_a \cup F_b|}{|F_a \cap F_b|},$$

containing information on the similarity of two sets F_a and F_b. To study the stability of an attribute selection the selection is performed l times on l subsets of the data. The mean Jaccard index is then calculated as

$$\hat{J} = \frac{2}{l\,(l-1)} \sum_{i=1}^{l} \sum_{j=i+1}^{l} J\left(F_i, F_j\right) .$$

Here a value near one corresponds to a stable attribute selection against statistical fluctuations in the learning algorithm.

4.1. Machine Learnring Algorithms for Classification

Multivariate learning algorithms are known to perform better than classification than simple cuts, because they can also consider correlations in the attributes. Since there are multiple concepts and implementations of classifiers on the market a selection of these is evaluated in this analysis to find the optimal classifier.

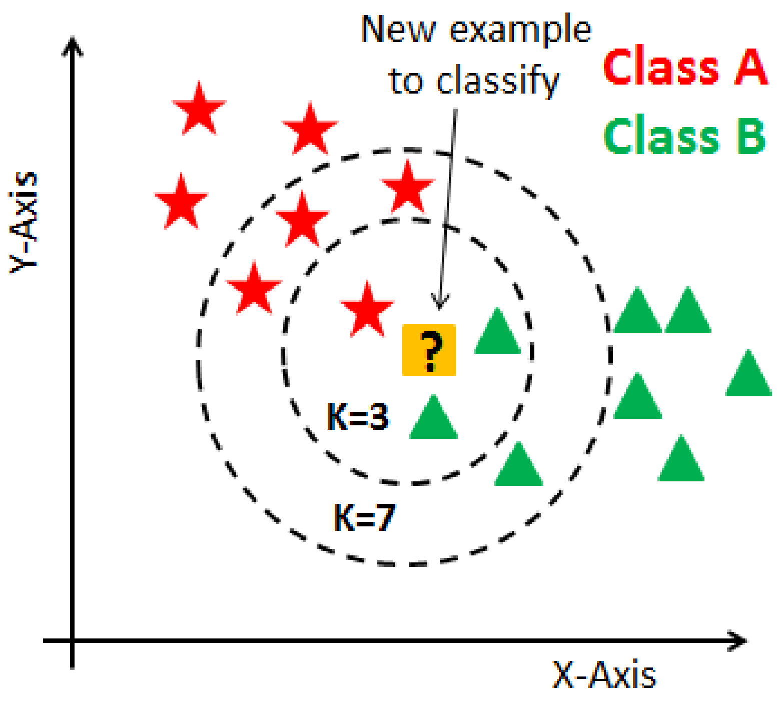

Abbildung 2: Schematic of a kNN Classifier's prediction dependency on k[8]. $k = 3$ would result in a prediction of class B and $k = 7$ would lead to a prediction of class A.

Naive-Bayes-Learner

This learner is based on the Bayes theorem of conditional probabilities

$$p\left(A|B\right) = \frac{p\left(B|A\right)p\left(A\right)}{p\left(B\right)},$$

where B depicts an attribute and A is the class affiliation with A representing signal and $\bar{A}$ background. Considering multiple attributes that are assumed to be only correlated with the classification, the quantity

$$Q = \prod_{i=1}^{n} \frac{p\left(B_i|A\right)}{p\left(B_i|\bar{A}\right)}$$

gives a measure of probability of an event being signal if $Q > 1$ or background if $Q < 1$.

kNN Classifier

The k nearest neighbors (kNN) classifier is a classifier that makes its prediction by calculating the mean value of the k nearest events as depicted in Abbildung 2. The prediction is highly dependent on k. kNN Classifiers with low k are sensitive to statistical fluctuations and those with high k can have worse predictions due to too many different classes in a bunch.

Random Forest Classifier

The random forest classifier is based on the binary decision tree. A decision tree consists of nodes at which a cut on one attribute is performed splitting the data. This is done subsequently until a specified depth is reached or if a node only contains one class which is then called a leaf. To prevent overfitting the mean value of a variety of trees is calculated. In a common implementation each tree is trained on a subset of the whole data and the k attributes to be considered for the best cut are chosen randomly. The confidence

$$c = \frac{1}{N}\sum_{i=1}^{N} P_i \, , P_i \in \{0, 1\}$$

of the prediction then is the arithmetic mean of the decisions P_i of the single trees.

4.2. Quality Parameters

The quality of separation of two classes can be evaluated with a variety of quality parameters. A common measure are the

$$\begin{aligned} \text{purity } p &= \frac{tp}{tp + fp} \\ \text{efficiency } r &= \frac{tp}{tp + fn} \, . \end{aligned}$$

Here tp denotes the number of *true positive* predictions, i.e. correctly predicted as signal, wheres tn and fn denote the number of *true negative* and *false negative* predictions, i.e. correctly and falsely, respectively, predicted background events.

To estimate an error for these parameters so called cross validation is used. here the data is split into n subsets of which $n - 1$ are used for training and the remaining is used for validation. This is done n times so that each subset is used as validation set. For each iteration the quality parameters above are calculated and a mean value and standard deviation on these can be given.

Another common measure of the quality of binary classifiers are receiver operating characteristics (ROC) curves which show the ratio of true positives or true positive rate (tpr) against the ratio of false positives or false positive rate (fpr). A perfect classification would result in a horizontal line at $tpr = 1$ while a totally random classification can be depicted as a diagonal line from $(0, 0)$ to $(1, 1)$ meaning equally many false positves and true positives would be predicted. The quality of the ROC curve can be described via the area under curve (AUC) score. Totally random classification would give AUC $= 0.5$ while a perfect classifier would score AUC $= 1$.

5. Analysis

5.1. Data Preparation

The data used consists of a dataset containing signals generated in a Monte-Carlo simulation and one dataset containing background signals also generated in a Monte-Carlo simulation.
As the initial step of the analysis the data is prepared. First all Monte-Carlo truths, event identification numbers and weights are removed so that only the simulated measurements and event-labels which will later be used for supervised learning are left. In the next step all simulated measurement with missing or infinite values are removed from the data. Furthermore all columns that contain te same value in every line are deleted. Finally in both the simulated signal- and backgrounddata attributes which are not contained in the other are removed und both datasets are combined to one.
Finally the data is split in a test- and training-dataset while 80 % of the data is used for training and 20 % for testing.

5.2. Forward Selection

In the next step of the analysis the number of features is reduced by determining the most important features using the `SelectKBest` with *f_classif* from the `sklearn` library and removing all other features [9].
For this the dataset is sequentially reduced to
$N \in \{10, 20, 30, 40, 40, 50, 60, 70, 80, 90, 100\}$ features and for each number of features the jaccard index is determined for the Naive-Bayes Classifier `GaussianNB` [9]. In the end the number of features that yields the highest jaccard index is selected for the analysis. In the run used for the analysis the best jaccard index was archieved for 50 features. The best 50 features are determined using `SelectKBest` and all other features are removed from the data.

5.3. Training the Classifiers

For the seperation of the singal data from the background data the classifiers `RandomForestClassifier`, `KNeighborsClassifier` and `GaussianNB` from `sklearn` are used. in the case of `RandomForestClassifier` and `KNeighborsClassifier` the optimal number of trees and neighbours is determined analogously to the number of attributes by trying different values and choosing the value with the largest jaccard index. The optimal values are $N_{\text{trees}} = 80$ and $N_{\text{neighbours}} = 50$ [9].
All classifiers are trained on the training data and the jaccard index, purity and efficiency are determined by cross validation using the method `cross_val_score` [9]. The values are listed in table 1.
Finally a receiver operating characteristic is generated utilising the function `roc_curve` and pictured in figure 3 [9].

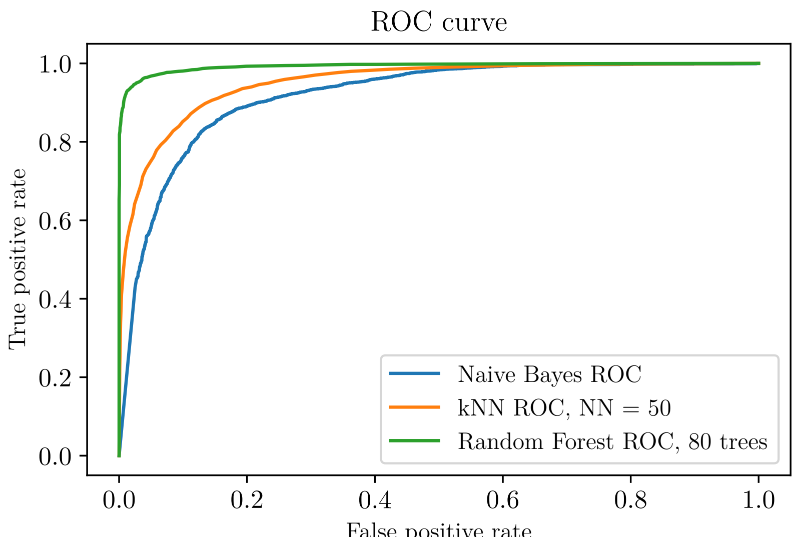

Abbildung 3: ROC curves for the three differrent classifiers.

Classifier	Efficiency	Purity	Jaccard index
RandomForest	$0{,}9525 \pm 0{,}0085$	$0{,}9651 \pm 0{,}0829$	$0{,}9187 \pm 0{,}0757$
KNeighborsClassifier	$0{,}8540 \pm 0{,}0215$	$0{,}7661 \pm 0{,}0621$	$0{,}6769 \pm 0{,}0407$
Naive-Bayes	$0{,}7970 \pm 0{,}0528$	$0{,}7980 \pm 0{,}0662$	$0{,}6618 \pm 0{,}0212$

Tabelle 1: Efficiency, purity and Jaccard index of the three classifiers applied to the data set with error estimated by cross validation [9].

6. Discussion

The errors of the values in table 1 are all of the same order of magnitude. The best result in efficiency, purity and the jaccard index in table 1 was archieved with the Random Forest classifier while the ROC curve is also closest to the ideal curve.
It is clear from figure 3 that the second closest curve to the ideal curve is that of the kNN classifier while it archieves an efficiency above, a purity below and a jaccard index in table 1 above that of the Naive-Bayes classifier. The performance of the kNN classifier being so close to the Naive-Bayes classifier could be caused by the use of to many neighbours leading to to many classes in a bunch and leading to worse predictions. Therefore it would be interesting for further investigation if the used algorithm would perform better if the criterium for selecting the optimal number of used neighbours is changed.
Overall it is clear that the Random Forest classifier seperates the signal from the background most successfully and is therfore best fitted to the task, assuming that computing time does not play a role in the selection of the algorithm.

Literatur

[1] Georges Aad u. a. „Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC“. In: *Phys. Lett. B* 716 (2012), S. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[2] R. Abbasi u. a. „IceTop: The surface component of IceCube“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 700 (2013), S. 188–220. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2012.10.067. URL: https://www.sciencedirect.com/science/article/pii/S016890021201217X.

[3] R. Abbasi u. a. „The design and performance of IceCube DeepCore“. In: *Astroparticle Physics* 35.10 (2012), S. 615–624. ISSN: 0927-6505. DOI: https://doi.org/10.1016/j.astropartphys.2012.01.004. URL: https://www.sciencedirect.com/science/article/pii/S0927650512000254.

[4] A. Achterberg u. a. „First year performance of the IceCube neutrino telescope“. In: *Astroparticle Physics* 26.3 (2006), S. 155–173. ISSN: 0927-6505. DOI: https://doi.org/10.1016/j.astropartphys.2006.06.007. URL: https://www.sciencedirect.com/science/article/pii/S0927650506000855.

[5] ENRICO Fermi. „On the Origin of the Cosmic Radiation“. In: *Phys. Rev.* 75 (8 Apr. 1949), S. 1169–1174. DOI: 10.1103/PhysRev.75.1169. URL: https://link.aps.org/doi/10.1103/PhysRev.75.1169.

[6] *IceCube*. URL: https://icecube.wisc.edu/science/icecube/ (besucht am 05. 05. 2022).

[7] *Introduction to Elementary Particles*. John Wiley & Sons, Ltd, 1987. ISBN: 9783527618460. DOI: https://doi.org/10.1002/9783527618460. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527618460.

[8] Avinash Navlani. *KNN Classification Tutorial using Scikit-learn*. 8. Feb. 2018. URL: https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn (besucht am 05. 06. 2022).

[9] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

A. Appendix

A.1. Code used for Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# sklean libraries
from sklearn.model_selection import train_test_split, cross_val_score
```

```python
from sklearn import ensemble
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, precision_score, accuracy_score,
    jaccard_score

df_signal = pd.read_csv("data/signal.csv", delimiter=";")
df_background = pd.read_csv("data/background.csv", delimiter=";")


# drop columns with key words that are associated with MC Simulation

df_signal = df_signal.drop(df_signal.filter(regex="MC").columns, axis=1)
df_signal = df_signal.drop(df_signal.filter(regex="Weight").columns, axis=1)
df_signal = df_signal.drop(df_signal.filter(regex="Corsika").columns, axis=1)
df_signal = df_signal.drop(df_signal.filter(regex="I3EventHeader").columns,
    axis=1)
df_signal = df_signal.drop(df_signal.filter(regex="end").columns, axis=1)
df_signal = df_signal.drop(df_signal.filter(regex="start").columns, axis=1)
df_signal = df_signal.drop(df_signal.filter(regex="time").columns, axis=1)
df_signal = df_signal.drop(df_signal.filter(regex="NewID").columns, axis=1)

df_background.drop(df_background.filter(regex="MC").columns, axis=1, inplace=
    True)
df_background.drop(df_background.filter(regex="Weight").columns, axis=1,
    inplace=True)
df_background.drop(df_background.filter(regex="Corsika").columns, axis=1,
    inplace=True)
df_background.drop(df_background.filter(regex="I3EventHeader").columns, axis
    =1, inplace=True)
df_background.drop(df_background.filter(regex="end").columns, axis=1, inplace
    =True)
df_background.drop(df_background.filter(regex="start").columns, axis=1,
    inplace=True)
df_background.drop(df_background.filter(regex="time").columns, axis=1,
    inplace=True)
df_background.drop(df_background.filter(regex="NewID").columns, axis=1,
    inplace=True)


df_signal.replace({np.inf: np.nan, -np.inf: np.nan}, value=None, inplace=True
    )
df_background.replace({np.inf: np.nan, -np.inf: np.nan}, value=None, inplace=
    True)
# convert all infinities into NaN

df_signal.dropna(axis=1, how="any", inplace=True)
df_background.dropna(axis=1, how="any", inplace=True)
# delete NaN from dataframes

col_signal = df_signal
col_background = df_background
# save columns in sets
```

```

for feature in col_signal:
    if feature not in col_background:
        df_signal.drop(feature, axis=1, inplace=True)

for feature in col_background:
    if feature not in col_signal:
        df_background.drop(feature, axis=1, inplace=True)
# remove columns from eother Dataframe that are not in the other dataframe

data = [df_signal, df_background]
df = pd.concat(data)
# merge both dataframes into oner dataframe

df = df.loc[:, df.apply(pd.Series.nunique) != 1]
# remove all columns where every value in the column is equal

X = df.drop("label", axis=1)
y = df["label"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=15, shuffle=True
)
# split the datafrme in a test- and a train-set whlie shuffeling the data

print(len(X.columns))
# print the number of features

# next the optimal number of attributes will be determined by looking for the
     number with the highest Jaccard index with the naive bayes clasifier

N_att = np.linspace(10, 100, 10, dtype=int).tolist()
# numbers of attributes to be tested

jac = []
# list that will be used to save the respective jaccard index

# loop over attribute numbers
for att in N_att:
    X_new = SelectKBest(score_func=f_classif, k=att)
    # generate instence to select best features

    d_fit = X_new.fit(X_train, y_train)
    # generate scores for features

    scores = d_fit.scores_
    # scores of features.

    args_max = np.argsort(scores)[::-1]
    # save scores largest to lowest score into an array

    attributes = []
    for i in range(att):
        attributes.append(X_train.columns.tolist()[args_max[i]])
    # saving impotant attributes into a list
```

```python

    X_train_short = X_train.loc[:, attributes]
    X_test_short = X_test.loc[:, attributes]
    # remove unimportant attributes

    NB_clf = GaussianNB()
    # generate instance of Naive-Bayes classifier

    NB_clf.fit(X_train_short, y_train)
    # train on data

    NB_Jscore = jaccard_score(np.array(y_test), NB_clf.predict(X_test_short))
    jac.append(NB_Jscore)
    # calculate and save jaccard index

    print("Jaccard-Score: ", NB_Jscore, " with ", att, " features")
    # status update

jac_max = max(jac)
jac_maxpos = jac.index(jac_max)
att = N_att[jac_maxpos]
print(att, " attributes will be used")
# save best number of attributes


# now the data will be prepared by removing unimportant attributes

X_new = SelectKBest(score_func=f_classif, k=att)
d_fit = X_new.fit(X_train, y_train)
# generate scores to select features

scores = d_fit.scores_
# get the scores

args_max = np.argsort(scores)[::-1]
# save scores largest to lowest scores into array

features = []
for i in range(att):
    features.append(X_train.columns.tolist()[args_max[i]])
# save only important features

X_train = X_train.loc[:, features]
X_test = X_test.loc[:, features]
# remove unimprtant features


# the next step is the implementation of the random forest classifier

# find the optimal number of trees

N_trees = np.linspace(10, 150, 15, dtype=int).tolist()
# list with the numbers of trees to be tested

```

```
jac = []
# array to safe jaccard index

for tree in N_trees:
    RFClf = ensemble.RandomForestClassifier(n_estimators=tree)
    # generate classifier with tree trees

    RFClf.fit(X_train, y_train)
    # train classifier

    rfc_Jscore = jaccard_score(y_test, RFClf.predict(X_test))
    # calculate score

    jac.append(rfc_Jscore)
    # save jaccard index

    print("jaccard score, RFC: ", rfc_Jscore, " with ", tree, " trees")
    # status update


jac_max = max(jac)
jac_maxpos = jac.index(jac_max)
trees = N_trees[jac_maxpos]
# save everithing related toi the maximal value

print("Maximal index, RFC: ", jac_max, " at ", jac_maxpos)
print("With ", trees, "trees")
# print the result


plt.plot(N_trees, jac)
plt.yscale("log")
plt.xlabel("Number of trees")
plt.ylabel("Jaccard Score")
plt.title("Jaccard-Score, RFC")
# plt.show()
# make a beautiful plot to see if there is a trend

# <<<<<<< HEAD
# #plt.savefig('/plots/RF_test.pdf')
# ||||||| 340fc38
# plt.savefig('/plots/RF_test.pdf')
# =======
plt.savefig("plots/RF_test.pdf")
# >>>>>>> 333782f0034d33d89ecaac4534123250a69b7cf3
plt.clf()

# finally the classifier will be used with the optimal trees

RFClf = ensemble.RandomForestClassifier(n_estimators=trees)
# generate classifier with 100 trees

# RFClf.get_params()
# returns parameters of estimator in dictionary?
```

```python

RFClf.fit(X_train, y_train)
# train classifier

y_pred = RFClf.predict_proba(X_test)
# predict values for the test set

y_pred = y_pred[:, 1]
# congratulations, its a vector

fpr1, tpr1, thr1 = roc_curve(y_test, y_pred)
# get estimates of false positive rate, true positive rate and thr


cv_score_rfc_eff = cross_val_score(RFClf, X, y, cv=5, scoring="recall")
print(
    "Efficiency: %0.4f (+/- %0.4f)"
    % (cv_score_rfc_eff.mean(), cv_score_rfc_eff.std() * 2)
)

cv_score_rfc_rein = cross_val_score(RFClf, X, y, cv=5, scoring="precision")
print(
    "Purity: %0.4f (+/- %0.4f)"
    % (cv_score_rfc_rein.mean(), cv_score_rfc_rein.std() * 2)
)

cv_score_rfc_J = cross_val_score(RFClf, X, y, cv=5, scoring="jaccard")
print(
    "Jaccard Index: %0.4f (+/- %0.4f)"
    % (cv_score_rfc_J.mean(), cv_score_rfc_J.std() * 2)
)
#  efficiency, precision and jaccard index with cross-validation

# now with kNN classifier

# estimate best number for neighbours

N_neighbours = np.linspace(10, 50, 10, dtype=int).tolist()
# list with the numbers of trees to be tested

jac = []
# list to save jaccard index

for neigh in N_neighbours:

    kNN_clf = ensemble.RandomForestClassifier(n_estimators=neigh)
    # generate classifier with neigh neighbours

    kNN_clf.fit(X_train, y_train)
    # train classifier

    kNN_Jscore = jaccard_score(y_test, kNN_clf.predict(X_test))
    # calculate score

    jac.append(kNN_Jscore)
    # berechne den jaccard score und speicher ihn ab

    print("jaccard score, kNN: ", kNN_Jscore, " with ", neigh, " neighbours")
    # status update

jac_max = max(jac)
jac_maxpos = jac.index(jac_max)
neighbours = N_neighbours[jac_maxpos]
# save everithing related the the maximal value

print("Maximal index, kNN: ", jac_max, " at ", jac_maxpos)
print("With ", neighbours, " neighbours")
# print the result


plt.plot(N_neighbours, jac)
plt.yscale("log")
plt.xlabel("Number of Neighbours")
plt.ylabel("Jaccard Score")
plt.title("Jaccard-Score, kNN")
# plt.show()
# make a beautiful plot to see if there is a trend

# <<<<<<< HEAD
# #plt.savefig('/plots/kNN_test.pdf')
# ||||||| 340fc38
# plt.savefig('/plots/kNN_test.pdf')
# =======
plt.savefig("plots/kNN_test.pdf")
# >>>>>>> 333782f0034d33d89ecaac4534123250a69b7cf3

plt.close()

# kNN learning

knn_clf = KNeighborsClassifier(n_neighbors=neighbours)
knn_clf.fit(X_train, y_train)
PRED_knn = knn_clf.predict_proba(X_test)
PRED_knn = PRED_knn[:, 1]
fpr2, tpr2, thr2 = roc_curve(y_test, PRED_knn)


cv_score_knn_eff = cross_val_score(knn_clf, X, y, cv=5, scoring="recall")
print(
    "Efficiency: %0.4f (+/- %0.4f)"
    % (cv_score_knn_eff.mean(), cv_score_knn_eff.std() * 2)
)

cv_score_knn_rein = cross_val_score(knn_clf, X, y, cv=5, scoring="precision")
print(
    "Purity: %0.4f (+/- %0.4f)"
    % (cv_score_knn_rein.mean(), cv_score_knn_rein.std() * 2)
)
```

```python

cv_score_knn_J = cross_val_score(knn_clf, X, y, cv=5, scoring="jaccard")
print(
    "Jaccard Index: %0.4f (+/- %0.4f)"
    % (cv_score_knn_J.mean(), cv_score_knn_J.std() * 2)
)


# finally the Naive-Bayes classifier

NB_clf = GaussianNB()
NB_clf.fit(X_train, y_train)
NB_pred = NB_clf.predict_proba(X_test)
NB_pred = NB_pred[:, 1]
fpr3, tpr3, thr3 = roc_curve(y_test, NB_pred)


cv_score_nb_eff = cross_val_score(NB_clf, X, y, cv=5, scoring="recall")
print(
    "Efficiency: %0.4f (+/- %0.4f)" % (cv_score_nb_eff.mean(),
    cv_score_nb_eff.std() * 2)
)

cv_score_nb_rein = cross_val_score(NB_clf, X, y, cv=5, scoring="precision")
print(
    "Purity: %0.4f (+/- %0.4f)"
    % (cv_score_nb_rein.mean(), cv_score_nb_rein.std() * 2)
)

cv_score_nb_J = cross_val_score(NB_clf, X, y, cv=5, scoring="jaccard")
print(
    "Jaccard Index: %0.4f (+/- %0.4f)" % (cv_score_nb_J.mean(), cv_score_nb_J
    .std() * 2)
)


# all into one happy plot

plt.plot(fpr3, tpr3, label="Naive Bayes ROC")
plt.plot(fpr2, tpr2, label="kNN ROC, NN = {}".format(neighbours))
plt.plot(fpr1, tpr1, label="Random Forest ROC, {} trees".format(trees))
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
#plt.yscale("log")
plt.title("ROC curve")
plt.legend(loc="best")
# plt.show()

# <<<<<<< HEAD
# #plt.savefig('/plots/ROC.pdf')
# ||||||| 340fc38
# plt.savefig('/plots/ROC.pdf')
# =======
plt.savefig("plots/ROC.pdf")
```

```
# >>>>>>> 333782f0034d33d89ecaac4534123250a69b7cf3
# plt.close()
```