

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

# Contents

<b>1. Introduction</b>	<b>3</b>
<b>2. Theory</b>	<b>3</b>
2.1. The Standard Model of Particle Physics . . . . .	3
2.2. Cosmic Rays . . . . .	4
<b>3. The IceCube Neutrino Observatory</b>	<b>4</b>
<b>4. Multivariate Selection and Machine Learning</b>	<b>6</b>
4.1. Machine Learning Algorithms for Classification . . . . .	7
4.2. Quality Parameters . . . . .	8
<b>5. Analysis</b>	<b>9</b>
5.1. Data Preparation . . . . .	9
5.2. Forward Selection . . . . .	9
5.3. Training the Classifiers . . . . .	10
<b>6. Discussion</b>	<b>11</b>
<b>A. Appendix</b>	<b>12</b>
A.1. Code used for Analysis . . . . .	12

# 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] is the theory that describes all known fundamental interactions except gravity. These are the strong, weak, and electromagnetic interactions. It also contains and classifies all known elementary particles, which 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.

The six leptons are also 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 boson 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 ( $\approx 89\%$ ) and  $\alpha$ -particles ( $\approx 10\%$ ). The remaining percent are contributions from heavier nuclei. This radiation can achieve energies up to  $10^{20}$  eV and the energy spectrum can approximately be described by a power law

$$\frac{d\Phi}{dE} = \Phi_0 E^\gamma, \quad (1)$$

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 either come from astrophysical sources directly or be produced in atmospheric interactions of primary CRs. The latter 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  $\propto E^{-3.7}$ .

Prompt muons and neutrinos are produced in the decay of  $D$  mesons and  $\Lambda_c$  baryons, which can be produced at high energies. These have very short lifetimes and therefore do not lose any energy before decaying, leading to an energy spectrum  $\propto E^{-2.7}$  for the muons and neutrinos.

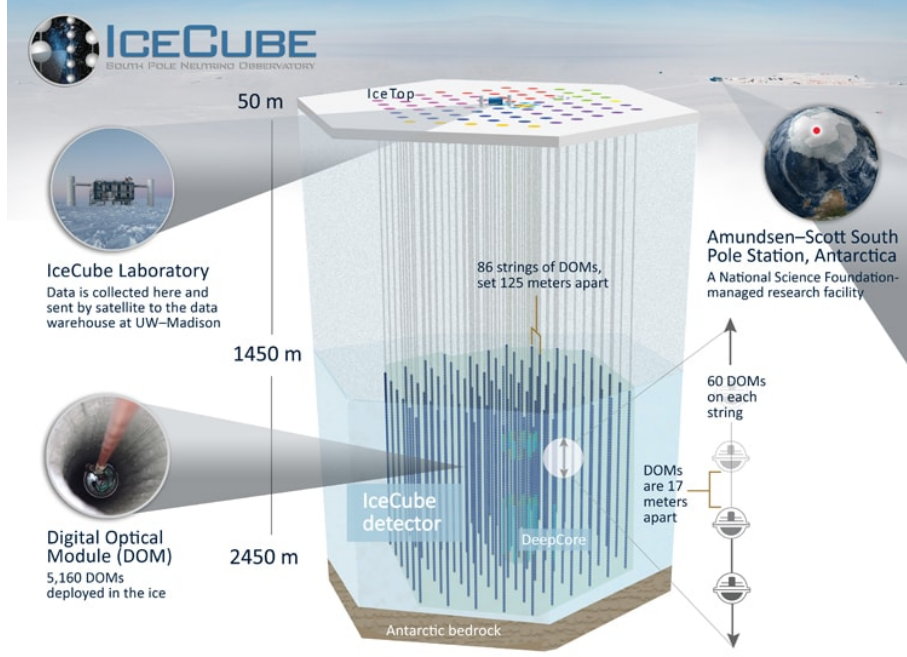
It is expected that cosmological sources of accelerated hadrons also emit photons and neutrinos. These are not deflected in the galactic and extragalactic magnetic fields due to their lack of electric charge. Photons are significantly affected by absorption and emission processes though, leading to a loss of the information about their source directions. Neutrinos on the other hand are able to pass through optically dense media due to their small cross section. Therefore the arrival directions of neutrinos point back to their sources.

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 pole. The experiment consists of 5160 digital optical modules (DOMs) on 86 strings located 1450 m–2450 m under the surface as seen in Figure 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

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



**Figure 1:** Schematic of the IceCube Experiment[6].

where  $c_0$  is the speed of light in the vacuum and  $n$  is the refractive index of the medium. The seven strings of DeepCore, as well as the PMTs on each string, are spaced more densely, resulting in a lower energy threshold of 10 GeV in contrast to a 100 GeV energy threshold of the rest of the In-Ice Array. 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)

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

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. The minimum ionizing nature of muons in combination with its long lifetime in comparison to tau leptons allows the muons to travel through the detector volume while continually emitting Cherenkov radiation. The wavefronts of the emitted Cherenkov radiation interfere, resulting in a Cherenkov cone along the track

of the muon.

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 angular resolution. In comparison, muon tracks have a worse energy resolution, but a higher angular resolution. Planet earth can be used as a shield for atmospheric muons since these are absorbed. Therefore muons that come from below the detector have to come from muon neutrinos. To reject atmospheric muons, a cut on the zenith angle can be performed. This leads to an enhanced detector volume since also muons from neutrino interactions outside 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

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

The attribute selection is stable against statistical fluctuations with a value close to 1.

### 4.1. Machine Learning 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.

### Naive-Bayes-Learner

This learner is based on the Bayes theorem of conditional probabilities

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

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

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

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 based on the  $k$  nearest neighbors of an object. The distance between two objects is measured in the phase space of all considered attributes. There are a variety of ways to define the distance, but a simple possibility would be the euclidian distance. The training of the classifier solely consists of saving the training data. The classification principle is depicted in Figure 2. An object is classified as the class, which the majority of the  $k$  nearest neighbors are in.

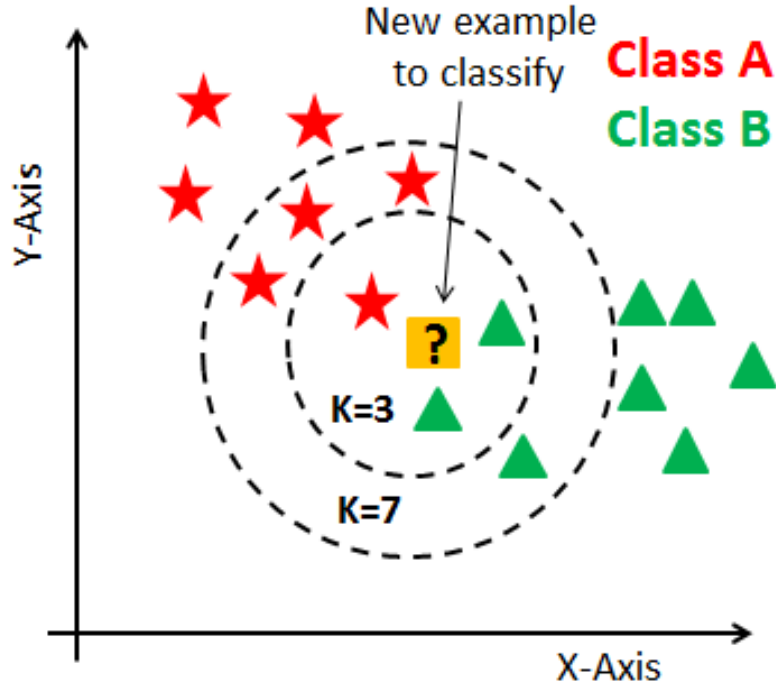
The prediction is highly dependent on the choice of  $k$ . kNN Classifiers with low  $k$  are sensitive to statistical fluctuations and those with high  $k$  can have worse predictions, because they include objects that are really far away. The latter is especially a problem when the training data is sparse or not distributed uniformly.

### 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. For each node the best split is determined from a random subset of  $k$  attributes. 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. 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.



**Figure 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.

## 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

$$\text{purity } p = \frac{tp}{tp + fp}$$

$$\text{efficiency } r = \frac{tp}{tp + fn}.$$

Here  $tp$  denotes the number of *true positive* predictions, i.e. correctly predicted as signal, whereas  $tn$  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 positives 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 the 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 and 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].

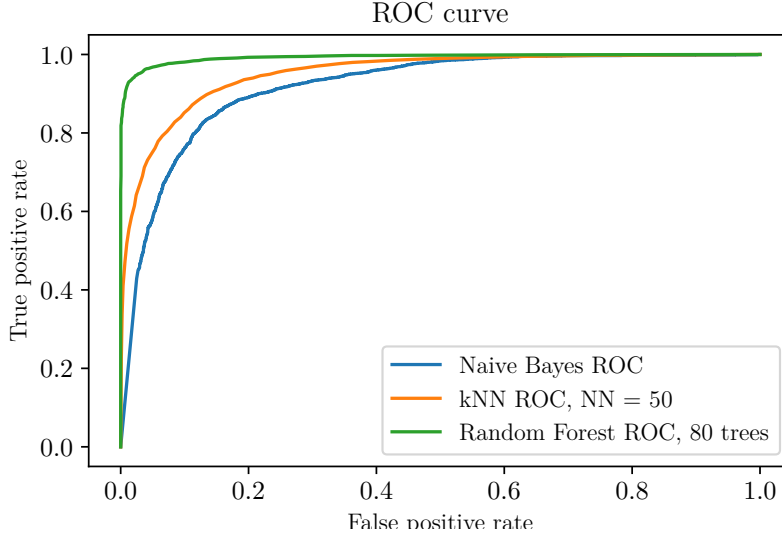
For this the dataset is sequentially reduced to

$N \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$  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 achieved 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 separation of the signal data from the background data the classifiers `RandomForestClassifier`, `KNeighborsClassifier` and `GaussianNB` from `sklearn` are



**Figure 3:** Determined ROC curves for the used classifiers.

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  $N_{\text{neighbours}} = 50$ .

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].

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

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	$0.7970 \pm 0.0528$	$0.7980 \pm 0.0662$	$0.6618 \pm 0.0212$

## 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 was achieved 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 achieves an efficiency above, a purity below and a jaccard index 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 too many neighbours leading to too 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 separates the signal from the background most successfully and is therefore best fitted to the task, assuming that computing time does not play a role in the selection of the algorithm.

## References

- [1] Georges Aad et al. “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), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].
- [2] R. Abbasi et al. “IceTop: The surface component of IceCube”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 700 (2013), pp. 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 et al. “The design and performance of IceCube DeepCore”. In: *Astroparticle Physics* 35.10 (2012), pp. 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 et al. “First year performance of the IceCube neutrino telescope”. In: *Astroparticle Physics* 26.3 (2006), pp. 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), pp. 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/> (visited on 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*. Feb. 8, 2018. URL: <https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn> (visited on 06/05/2022).
- [9] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

## A. Appendix

### A.1. Code used for Analysis

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # sklearn libraries
```

```

6 from sklearn.model_selection import train_test_split, cross_val_score
7 from sklearn import ensemble
8 from sklearn.feature_selection import SelectKBest, f_classif
9 from sklearn.neighbors import KNeighborsClassifier
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.metrics import roc_curve, precision_score, accuracy_score,
    jaccard_score
12
13 df_signal = pd.read_csv("data/signal.csv", delimiter=";")
14 df_background = pd.read_csv("data/background.csv", delimiter=";")
15
16
17 # drop columns with key words that are associated with MC Simulation
18
19 df_signal = df_signal.drop(df_signal.filter(regex="MC").columns, axis=1)
20 df_signal = df_signal.drop(df_signal.filter(regex="Weight").columns, axis=1)
21 df_signal = df_signal.drop(df_signal.filter(regex="Corsika").columns, axis=1)
22 df_signal = df_signal.drop(df_signal.filter(regex="I3EventHeader").columns,
    axis=1)
23 df_signal = df_signal.drop(df_signal.filter(regex="end").columns, axis=1)
24 df_signal = df_signal.drop(df_signal.filter(regex="start").columns, axis=1)
25 df_signal = df_signal.drop(df_signal.filter(regex="time").columns, axis=1)
26 df_signal = df_signal.drop(df_signal.filter(regex="NewID").columns, axis=1)
27
28 df_background.drop(df_background.filter(regex="MC").columns, axis=1, inplace=
    True)
29 df_background.drop(df_background.filter(regex="Weight").columns, axis=1,
    inplace=True)
30 df_background.drop(df_background.filter(regex="Corsika").columns, axis=1,
    inplace=True)
31 df_background.drop(df_background.filter(regex="I3EventHeader").columns, axis
    =1, inplace=True)
32 df_background.drop(df_background.filter(regex="end").columns, axis=1, inplace
    =True)
33 df_background.drop(df_background.filter(regex="start").columns, axis=1,
    inplace=True)
34 df_background.drop(df_background.filter(regex="time").columns, axis=1,
    inplace=True)
35 df_background.drop(df_background.filter(regex="NewID").columns, axis=1,
    inplace=True)
36
37
38 df_signal.replace({np.inf: np.nan, -np.inf: np.nan}, value=None, inplace=True
    )
39 df_background.replace({np.inf: np.nan, -np.inf: np.nan}, value=None, inplace=
    True)
40 # convert all infinities into NaN
41
42 df_signal.dropna(axis=1, how="any", inplace=True)
43 df_background.dropna(axis=1, how="any", inplace=True)
44 # delete NaN from dataframes
45
46 col_signal = df_signal
47 col_background = df_background

```

```

48 # save columns in sets
49
50 for feature in col_signal:
51     if feature not in col_background:
52         df_signal.drop(feature, axis=1, inplace=True)
53
54 for feature in col_background:
55     if feature not in col_signal:
56         df_background.drop(feature, axis=1, inplace=True)
57 # remove columns from eother Dataframe that are not in the other dataframe
58
59 data = [df_signal, df_background]
60 df = pd.concat(data)
61 # merge both dataframes into oner dataframe
62
63 df = df.loc[:, df.apply(pd.Series.nunique) != 1]
64 # remove all columns where every value in the column is equal
65
66 X = df.drop("label", axis=1)
67 y = df["label"]
68 X_train, X_test, y_train, y_test = train_test_split(
69     X, y, test_size=0.2, random_state=15, shuffle=True
70 )
71 # split the datafrme in a test- and a train-set whlie shuffeling the data
72
73 print(len(X.columns))
74 # print the number of features
75
76 # next the optimal number of attributes will be determined by looking for the
    number with the highest Jaccard index with the naive bayes clasifier
77
78 N_att = np.linspace(10, 100, 10, dtype=int).tolist()
79 # numbers of attributes to be tested
80
81 jac = []
82 # list that will be used to save the respective jaccard index
83
84 # loop over attribute numbers
85 for att in N_att:
86     X_new = SelectKBest(score_func=f_classif, k=att)
87     # generate instence to select best features
88
89     d_fit = X_new.fit(X_train, y_train)
90     # generate scores for features
91
92     scores = d_fit.scores_
93     # scores of features.
94
95     args_max = np.argsort(scores)[::-1]
96     # save scores largest to lowest score into an array
97
98     attributes = []
99     for i in range(att):
100         attributes.append(X_train.columns.tolist()[args_max[i]])

```

```

101     # saving impotant attributes into a list
102
103     X_train_short = X_train.loc[:, attributes]
104     X_test_short = X_test.loc[:, attributes]
105     # remove unimportant attributes
106
107     NB_clf = GaussianNB()
108     # generate instance of Naive-Bayes classifier
109
110     NB_clf.fit(X_train_short, y_train)
111     # train on data
112
113     NB_Jscore = jaccard_score(np.array(y_test), NB_clf.predict(X_test_short))
114     jac.append(NB_Jscore)
115     # calculate and save jaccard index
116
117     print("Jaccard-Score: ", NB_Jscore, " with ", att, " features")
118     # status update
119
120     jac_max = max(jac)
121     jac_maxpos = jac.index(jac_max)
122     att = N_att[jac_maxpos]
123     print(att, " attributes will be used")
124     # save best number of attributes
125
126
127     # now the data will be prepared by removing unimportant attributes
128
129     X_new = SelectKBest(score_func=f_classif, k=att)
130     d_fit = X_new.fit(X_train, y_train)
131     # generate scores to select features
132
133     scores = d_fit.scores_
134     # get the scores
135
136     args_max = np.argsort(scores)[::-1]
137     # save scores largest to lowest scores into array
138
139     features = []
140     for i in range(att):
141         features.append(X_train.columns.tolist()[args_max[i]])
142     # save only important features
143
144     X_train = X_train.loc[:, features]
145     X_test = X_test.loc[:, features]
146     # remove unimprtant features
147
148
149     # the next step is the implementation of the random forest classifier
150
151     # find the optimal number of trees
152
153     N_trees = np.linspace(10, 150, 15, dtype=int).tolist()
154     # list with the numbers of trees to be tested

```

```

155
156 jac = []
157 # array to save jaccard index
158
159 for tree in N_trees:
160     RFClf = ensemble.RandomForestClassifier(n_estimators=tree)
161     # generate classifier with tree trees
162
163     RFClf.fit(X_train, y_train)
164     # train classifier
165
166     rfc_Jscore = jaccard_score(y_test, RFClf.predict(X_test))
167     # calculate score
168
169     jac.append(rfc_Jscore)
170     # save jaccard index
171
172     print("jaccard score, RFC: ", rfc_Jscore, " with ", tree, " trees")
173     # status update
174
175
176 jac_max = max(jac)
177 jac_maxpos = jac.index(jac_max)
178 trees = N_trees[jac_maxpos]
179 # save everithing related toi the maximal value
180
181 print("Maximal index, RFC: ", jac_max, " at ", jac_maxpos)
182 print("With ", trees, "trees")
183 # print the result
184
185
186 plt.plot(N_trees, jac)
187 plt.yscale("log")
188 plt.xlabel("Number of trees")
189 plt.ylabel("Jaccard Score")
190 plt.title("Jaccard-Score, RFC")
191 # plt.show()
192 # make a beautiful plot to see if there is a trend
193
194 # <<<<<< HEAD
195 # #plt.savefig('/plots/RF_test.pdf')
196 # ||||| 340fc38
197 # plt.savefig('/plots/RF_test.pdf')
198 # =====
199 plt.savefig("plots/RF_test.pdf")
200 # >>>>>> 333782f0034d33d89ecaac4534123250a69b7cf3
201 plt.clf()
202
203 # finally the classifier will be used with the optimal trees
204
205 RFClf = ensemble.RandomForestClassifier(n_estimators=trees)
206 # generate classifier with 100 trees
207
208 # RFClf.get_params()

```



```

209 # returns parameters of estimator in dictionary?
210
211 RFClf.fit(X_train, y_train)
212 # train classifier
213
214 y_pred = RFClf.predict_proba(X_test)
215 # predict values for the test set
216
217 y_pred = y_pred[:, 1]
218 # congratulations, its a vector
219
220 fpr1, tpr1, thr1 = roc_curve(y_test, y_pred)
221 # get estimates of false positive rate, true positive rate and thr
222
223
224 cv_score_rfc_eff = cross_val_score(RFClf, X, y, cv=5, scoring="recall")
225 print(
226     "Efficiency: %0.4f (+/- %0.4f)"
227     % (cv_score_rfc_eff.mean(), cv_score_rfc_eff.std() * 2)
228 )
229
230 cv_score_rfc_rein = cross_val_score(RFClf, X, y, cv=5, scoring="precision")
231 print(
232     "Purity: %0.4f (+/- %0.4f)"
233     % (cv_score_rfc_rein.mean(), cv_score_rfc_rein.std() * 2)
234 )
235
236 cv_score_rfc_J = cross_val_score(RFClf, X, y, cv=5, scoring="jaccard")
237 print(
238     "Jaccard Index: %0.4f (+/- %0.4f)"
239     % (cv_score_rfc_J.mean(), cv_score_rfc_J.std() * 2)
240 )
241 # efficiency, precision and jaccard index with cross-validation
242
243 # now with kNN classifier
244
245 # estimate best number for neighbours
246
247 N_neighbours = np.linspace(10, 50, 10, dtype=int).tolist()
248 # list with the numbers of trees to be tested
249
250 jac = []
251 # list to save jaccard index
252
253 for neigh in N_neighbours:
254
255     kNN_clf = ensemble.RandomForestClassifier(n_estimators=neigh)
256     # generate classifier with neigh neighbours
257
258     kNN_clf.fit(X_train, y_train)
259     # train classifier
260
261     kNN_Jscore = jaccard_score(y_test, kNN_clf.predict(X_test))
262     # calculate score

```

```

263
264     jac.append(kNN_Jscore)
265     # berechne den jaccard score und speicher ihn ab
266
267     print("jaccard score, kNN: ", kNN_Jscore, " with ", neigh, " neighbours")
268     # status update
269
270     jac_max = max(jac)
271     jac_maxpos = jac.index(jac_max)
272     neighbours = N_neighbours[jac_maxpos]
273     # save everithing related the the maximal value
274
275     print("Maximal index, kNN: ", jac_max, " at ", jac_maxpos)
276     print("With ", neighbours, " neighbours")
277     # print the result
278
279
280     plt.plot(N_neighbours, jac)
281     plt.yscale("log")
282     plt.xlabel("Number of Neighbours")
283     plt.ylabel("Jaccard Score")
284     plt.title("Jaccard-Score, kNN")
285     # plt.show()
286     # make a beautiful plot to see if there is a trend
287
288     # <<<<<< HEAD
289     # #plt.savefig('/plots/kNN_test.pdf')
290     # ||||| 340fc38
291     # plt.savefig('/plots/kNN_test.pdf')
292     # =====
293     plt.savefig("plots/kNN_test.pdf")
294     # >>>>>> 333782f0034d33d89ecaac4534123250a69b7cf3
295
296     plt.close()
297
298     # kNN learning
299
300     knn_clf = KNeighborsClassifier(n_neighbors=neighbours)
301     knn_clf.fit(X_train, y_train)
302     PRED_knn = knn_clf.predict_proba(X_test)
303     PRED_knn = PRED_knn[:, 1]
304     fpr2, tpr2, thr2 = roc_curve(y_test, PRED_knn)
305
306
307     cv_score_knn_eff = cross_val_score(knn_clf, X, y, cv=5, scoring="recall")
308     print(
309         "Efficiency: %0.4f (+/- %0.4f)"
310         % (cv_score_knn_eff.mean(), cv_score_knn_eff.std() * 2)
311     )
312
313     cv_score_knn_rein = cross_val_score(knn_clf, X, y, cv=5, scoring="precision")
314     print(
315         "Purity: %0.4f (+/- %0.4f)"
316         % (cv_score_knn_rein.mean(), cv_score_knn_rein.std() * 2)

```

```

317 )
318
319 cv_score_knn_J = cross_val_score(knn_clf, X, y, cv=5, scoring="jaccard")
320 print(
321     "Jaccard Index: %0.4f (+/- %0.4f)"
322     % (cv_score_knn_J.mean(), cv_score_knn_J.std() * 2)
323 )
324
325
326 # finally the Naive-Bayes classifier
327
328 NB_clf = GaussianNB()
329 NB_clf.fit(X_train, y_train)
330 NB_pred = NB_clf.predict_proba(X_test)
331 NB_pred = NB_pred[:, 1]
332 fpr3, tpr3, thr3 = roc_curve(y_test, NB_pred)
333
334
335 cv_score_nb_eff = cross_val_score(NB_clf, X, y, cv=5, scoring="recall")
336 print(
337     "Efficiency: %0.4f (+/- %0.4f)" % (cv_score_nb_eff.mean(),
338     cv_score_nb_eff.std() * 2)
339 )
340
341 cv_score_nb_rein = cross_val_score(NB_clf, X, y, cv=5, scoring="precision")
342 print(
343     "Purity: %0.4f (+/- %0.4f)"
344     % (cv_score_nb_rein.mean(), cv_score_nb_rein.std() * 2)
345 )
346
347 cv_score_nb_J = cross_val_score(NB_clf, X, y, cv=5, scoring="jaccard")
348 print(
349     "Jaccard Index: %0.4f (+/- %0.4f)" % (cv_score_nb_J.mean(), cv_score_nb_J
350     .std() * 2)
351 )
352
353 # all into one happy plot
354
355 plt.plot(fpr3, tpr3, label="Naive Bayes ROC")
356 plt.plot(fpr2, tpr2, label="kNN ROC, NN = {}".format(neighbours))
357 plt.plot(fpr1, tpr1, label="Random Forest ROC, {} trees".format(trees))
358 plt.xlabel("False positive rate")
359 plt.ylabel("True positive rate")
360 #plt.yscale("log")
361 plt.title("ROC curve")
362 plt.legend(loc="best")
363 # plt.show()
364
365 # <<<<<< HEAD
366 # #plt.savefig('/plots/ROC.pdf')
367 # ||||| 340fc38
368 # plt.savefig('/plots/ROC.pdf')
369 # =====

```

```

369 plt.savefig("plots/ROC.pdf")
370 # >>>>>> 333782f0034d33d89ecaac4534123250a69b7cf3
371 # plt.close()

```

$$F(x) = \int_{x_{\min}}^x N x'^{-n} dx' \quad (2)$$

$$\Leftrightarrow F(x) = \frac{N}{n+1} (x_{\min}^{1-n} - x^{1-n}) \quad (3)$$

$$\Leftrightarrow F(x) = \frac{x_{\min}^{1-n} - x^{1-n}}{x_{\min}^{1-n} - x_{\max}^{1-n}}. \quad (4)$$