Icecube

October 27, 2020

[1]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from skfeature.function.information_theoretical_based import MRMR

pd.set_option('display.max_columns', None)
```

[2]:
```
signal = pd.read_csv("signal.csv", sep = ";")
bkg = pd.read_csv("background.csv", sep = ";")
```

[3]:
```
signal = signal.drop(signal.filter(regex='MC').columns, axis=1)
signal = signal.drop(signal.filter(regex='Weight').columns, axis=1)
signal = signal.drop(signal.filter(regex='Corsika').columns, axis=1)
signal = signal.drop(signal.filter(regex='I3EventHeader').columns, axis=1)
signal = signal.drop(signal.filter(regex='end').columns, axis=1)
signal = signal.drop(signal.filter(regex='start').columns, axis=1)
signal = signal.drop(signal.filter(regex='time').columns, axis=1)
signal = signal.drop(signal.filter(regex='NewID').columns, axis=1)
signal = signal.drop('label', axis=1)
```

[4]:
```
bkg = bkg.drop(bkg.filter(regex='MC').columns, axis=1)
bkg = bkg.drop(bkg.filter(regex='Weight').columns, axis=1)
bkg = bkg.drop(bkg.filter(regex='Corsika').columns, axis=1)
bkg = bkg.drop(bkg.filter(regex='I3EventHeader').columns, axis=1)
bkg = bkg.drop(bkg.filter(regex='end').columns, axis=1)
bkg = bkg.drop(bkg.filter(regex='start').columns, axis=1)
bkg = bkg.drop(bkg.filter(regex='time').columns, axis=1)
bkg = bkg.drop(bkg.filter(regex='NewID').columns, axis=1)
bkg = bkg.drop('label', axis=1)
```

[5]:
```
signal.replace([np.inf, -np.inf], np.nan)
signal.dropna(axis = 'columns', inplace = True)
signal = signal.drop(signal.std()[(signal.std() == 0)].index, axis=1)
#signal.dropna(inplace = True)
```

```
bkg.replace([np.inf, -np.inf], np.nan)
bkg.dropna(axis = 'columns', inplace = True)
bkg = bkg.drop(bkg.std()[(bkg.std() == 0)].index, axis=1)
#bkg.dropna(inplace = True)
```

[6]:
```
bcol = bkg.columns
scol = signal.columns
```

[7]:
```
for att in scol:
    if att not in bcol:
        signal.drop(att, axis=1, inplace = True)

for att in bcol:
    if att not in scol:
        bkg.drop(att, axis=1, inplace = True)
```

[8]:
```
import scipy.io
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate
from sklearn import svm
from sklearn.metrics import jaccard_score
```

[9]:
```
# print(len(signal.columns))
# print(len(bkg.columns))
```

[10]:
```
sig_label = np.zeros(signal.shape[0])
bkg_label = np.ones(bkg.shape[0])
```

[11]:
```
combined_df = pd.concat([signal, bkg], ignore_index=True)
combined_label = np.append(sig_label, bkg_label)
```

[12]:
```
combined_df.insert(114, 'label', combined_label)
shuffled = combined_df.sample(frac = 1)
```

[13]:
```
y = shuffled['label']
X = shuffled.drop('label', axis=1)
```

[14]:
```
from sklearn import (
    ensemble, linear_model, neighbors, svm, tree, naive_bayes,
    gaussian_process, neural_network, dummy)
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.base import clone
from tqdm import tqdm
model = ensemble.RandomForestClassifier(n_estimators=100)
model.get_params()
from sklearn.model_selection import train_test_split
```

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[16]:
```
from sklearn.feature_selection import SelectKBest, chi2, mutual_info_classif,␣
↪f_classif
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import r2_score, roc_auc_score, roc_curve
import numpy as np
from sklearn.model_selection import train_test_split
import pandas as pd
from scipy.spatial import distance
from sklearn.metrics.cluster import v_measure_score
from sklearn.neighbors import KNeighborsClassifier

# Anzahl der features die wir nehmen wollen
N_feat = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85,␣
↪90, 95, 100]
nn = 20
# jac = []
# for feat in N_feat:
#     X_new = SelectKBest(score_func=f_classif, k=feat)
#     d_fit = X_new.fit(X_train, y_train)
#     # generiere scores die die güte des features angeben
#     scores = d_fit.scores_
# #     print(scores)
#     # sortiere nach größe...
#     sorted_scores = sorted(scores, reverse=True)
#     args_max = np.argsort(scores)[::-1]
#     # print(args_max)
#     features = []
#     for i in range(feat):
#         features.append(X.columns.tolist()[args_max[i]])
# #     print(features)
#     # werfe aus den trainingsdaten und testdaten alle features bis auf die␣
↪wichtigsten raus
#     X_train_sclief = X_train.loc[:, features]
#     X_test_sclief = X_test.loc[:, features]

#     knn_clf = KNeighborsClassifier(n_neighbors=nn)
#     knn_clf.fit(X_train_sclief, y_train)
#     PRED_knn = knn_clf.predict_proba(X_test_sclief)
#     PRED_knn = PRED_knn[:, 1]
#     fpr2, tpr2, thr2 = roc_curve(y_test, PRED_knn)

#     knn_precision = precision_score(np.array(y_test), knn_clf.
↪predict(X_test_sclief))
```

```
#     knn_eff = accuracy_score(np.array(y_test), knn_clf.predict(X_test_sclief))
# #     print('KNN accuracy score(sklearn) = ', knn_eff)
# #     print('KNN precision score(sklearn) = ', knn_precision)
#     knn_Jscore = jaccard_score(np.array(y_test), knn_clf.
 ↪predict(X_test_sclief))
#     jac.append(knn_Jscore)
# jac_max = max(jac)
# jac_maxpos = jac.index(jac_max)
# print(jac)
# print(jac_max)
# print(jac_maxpos)
```

[17]:
```
N_feat = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85,␣
 ↪90, 95, 100]
# feats = N_feat[jac_maxpos]
feats = 50
X_new = SelectKBest(score_func=f_classif, k=feats)
d_fit = X_new.fit(X_train, y_train)
# generiere scores die die güte des features angeben
scores = d_fit.scores_
# sortiere nach größe...
sorted_scores = sorted(scores, reverse=True)
args_max = np.argsort(scores)[::-1]
# print(args_max)
# lese die N_feat wichtigsten features aus und speichere sie weg
features = []
for i in range(feats):
    features.append(X.columns.tolist()[args_max[i]])
# print(features)
# werfe aus den trainingsdaten und testdaten alle features bis auf die␣
 ↪wichtigsten raus
X_train = X_train.loc[:, features]
X_test = X_test.loc[:, features]
```

[18]:
```
# trainiere den lerner
model.fit(X_train, y_train)
# sage die labels vorher
y_pred = model.predict_proba(X_test)
y_pred = y_pred[:, 1]
fpr1, tpr1, thr1 = roc_curve(y_test, y_pred)
```

[19]:
```
from sklearn.model_selection import cross_val_score
#print(roc_auc_score(y_test, y_pred))
#print(r2_score(y_test, y_pred))

RFC_precision = precision_score(y_test, model.predict(X_test))
RFC_eff = accuracy_score(y_test, model.predict(X_test))
```

```
print('RFC accuracy score(sklearn) = ', RFC_eff)
print('RFC precision score(sklearn) = ', RFC_precision)
rfc_Jscore = jaccard_score(y_test, model.predict(X_test))
print('jaccard score, RFC: ', rfc_Jscore)

cv_score_rfc_eff = cross_val_score(model, X, y, cv=5, scoring='recall')
print("Effizienz: %0.4f (+/- %0.4f)" % (cv_score_rfc_eff.mean(),␣
 ↪cv_score_rfc_eff.std() * 2))
cv_score_rfc_rein = cross_val_score(model, X, y, cv=5, scoring='precision')
print("Reinheit: %0.4f (+/- %0.4f)" % (cv_score_rfc_rein.mean(),␣
 ↪cv_score_rfc_rein.std() * 2))
cv_score_rfc_J = cross_val_score(model, X, y, cv=5, scoring='jaccard')
print("Jaccard Index: %0.4f (+/- %0.4f)" % (cv_score_rfc_J.mean(),␣
 ↪cv_score_rfc_J.std() * 2))
```

```
RFC accuracy score(sklearn) =  0.9635
RFC precision score(sklearn) =  0.9571038928837095
jaccard score, RFC:  0.9296724470134875
Effizienz: 0.9770 (+/- 0.0052)
Reinheit: 0.9556 (+/- 0.0060)
Jaccard Index: 0.9351 (+/- 0.0024)
```

[20]:
```
class KNN:
    def __init__(self, k):
        self.k = k


    def euc(self, a, b):
        return distance.euclidean(a, b)


    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train


    def predict(self, X):
        pred = []
        # row_count = 0
        for row in X: # iteriere durch jedes event
            # label = orte mit kleinstem abstand
            label = self.closest_points(row, self.X_train)
            # pred.append(prediction)
            N_sig = 0
            N_bg = 0
            for i in label:
                if self.y_train[i] == 0:
```

```
                    N_sig += 1
                else:
                    N_bg += 1
            if N_sig >= N_bg: # wenn label = signal ist dann 0 ( weil 0 =␣
↪signal heisst)
                pred.append(0)
            else:
                pred.append(1) # sonst 1 appenden da 1 = bg
            # if row_count % 10000 == 0:
                # print(row_count)
            # row_count += 1
        return pred


    def closest_points(self, row, X):
        index = [] # speichere hier die besten indizes
        distances = [] # alle distanzen
        distances = distance.cdist([row], X, 'euclidean')

        sort_dist = np.argsort(distances) # sortiere die distancen und gibt die␣
↪k kleinsten zurueck
        for j in range(self.k):
            index.append(sort_dist[0][j])

        return index
```

[21]:
```
from sklearn.preprocessing import normalize
nn = 20
knn = KNN(nn)
# daten muessen np arrays sein
X_trn = np.array(X_train)
y_trn = np.array(y_train)
X_t = np.array(X_test)
y_t = np.array(y_test)

knn_fit = knn.fit(X_trn, y_trn)
knn_pred = knn.predict(X_t)
```

[22]:
```
knn_roc_score = roc_auc_score(y_t, knn_pred)
#print('roc_auc score (knn): ', knn_roc_score)

y_true = y_t.tolist()

tp = 0
fp = 0
fn = 0
tn = 0
```

```
for i in range(len(y_true)):
    if knn_pred[i] == y_true[i] and knn_pred[i] == 0:
        tp += 1
    if knn_pred[i] == 0 and y_true[i] == 1:
        fp += 1
    if knn_pred[i] == 1 and y_true[i] == 0:
        fn += 1
    if knn_pred[i] == y_true[i] and knn_pred[i] == 1:
        tn += 1

# print('tp: ', tp, 'fp: ', fp, 'fn: ', fn, 'tn:', tn)

Eff = tp / (tp + fn)
P = tp / (tp + fp)
S = tp / np.sqrt(tp + tn)
accuracy = (tp + tn) / (tp + fp + tn + fn)

print('for kNN:')
print('accuracy score(sklearn) = ', accuracy_score(knn_pred, y_true))
print('Eff: ', Eff)
print('Purity: ', P)
print('Signifikanz: ', S)
print('Accuracy: ', accuracy)
```

```
for kNN:
accuracy score(sklearn) =  0.88675
Eff:  0.9057448395921412
Purity:  0.8735907891580715
Signifikanz:  43.24086264010846
Accuracy:  0.88675
```

[23]:
```
# knn von sklearn:
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_neighbors=nn)
knn_clf.fit(X_trn, y_trn)
PRED_knn = knn_clf.predict_proba(X_t)
PRED_knn = PRED_knn[:, 1]
fpr2, tpr2, thr2 = roc_curve(y_t, PRED_knn)
```

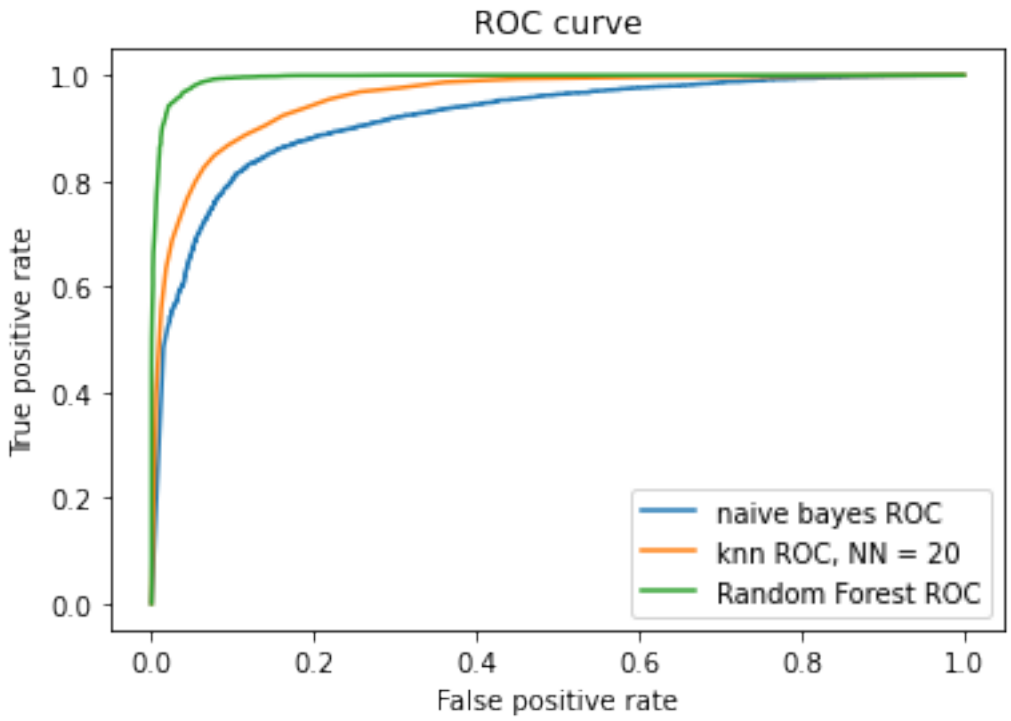
[24]:
```
knn_precision = precision_score(y_true, knn_clf.predict(X_t))
knn_eff = accuracy_score(y_true, knn_clf.predict(X_t))
print('KNN accuracy score(sklearn) = ', knn_eff)
print('KNN precision score(sklearn) = ', knn_precision)
knn_Jscore = jaccard_score(y_true, knn_clf.predict(X_t))
print('jaccard score, kNN: ', knn_Jscore)

cv_score_knn_eff = cross_val_score(knn_clf, X, y, cv=5, scoring='recall')
```

```
print("Effizienz: %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("Reinheit: %0.4f (+/- %0.4f)" % (cv_score_knn_rein.mean(),␣
 ↪cv_score_knn_rein.std() * 2))
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))
```

```
KNN accuracy score(sklearn) =  0.88675
KNN precision score(sklearn) =  0.9010702166536152
jaccard score, kNN:  0.7921064708581919
Effizienz: 0.7421 (+/- 0.0072)
Reinheit: 0.8505 (+/- 0.0065)
Jaccard Index: 0.6564 (+/- 0.0041)
```

[25]:
```
# naive bayes:
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train, y_train)
NB_pred = clf.predict_proba(X_test)
NB_pred = NB_pred[:, 1]
fpr3, tpr3, thr3 = roc_curve(y_test, NB_pred)
```

[27]:
```
plt.figure(1)
plt.plot(fpr3, tpr3, label='naive bayes ROC')
plt.plot(fpr2, tpr2, label='knn ROC, NN = {}'.format(nn))
plt.plot(fpr1, tpr1, label='Random Forest ROC')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.savefig("/home/nbreer/Pictures/roc_curve.pdf")
```



[28]:
```
NB_precision = precision_score(y_true, clf.predict(X_test))
NB_eff = accuracy_score(y_true, clf.predict(X_test))
print('NB accuracy score(sklearn) = ', NB_eff)
print('NB precision score(sklearn) = ', NB_precision)
NB_Jscore = jaccard_score(y_true, clf.predict(X_test))
print('jaccard score, NB: ', NB_Jscore)

cv_score_nb_eff = cross_val_score(clf, X, y, cv=5, scoring='recall')
print("Effizienz: %0.4f (+/- %0.4f)" % (cv_score_nb_eff.mean(), cv_score_nb_eff.
 ↪std() * 2))
cv_score_nb_rein = cross_val_score(clf, X, y, cv=5, scoring='precision')
print("Reinheit: %0.4f (+/- %0.4f)" % (cv_score_nb_rein.mean(),␣
 ↪cv_score_nb_rein.std() * 2))
cv_score_nb_J = cross_val_score(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))
```

```
NB accuracy score(sklearn) =  0.825
NB precision score(sklearn) =  0.7811205580989754
jaccard score, NB:  0.719044752157335
Effizienz: 0.8080 (+/- 0.0283)
Reinheit: 0.8014 (+/- 0.0142)
Jaccard Index: 0.6730 (+/- 0.0112)
```