KunitzProject_Tornisiello

June 11, 2020

1 This notebook contains the python script used to handle the datasets,compute the MCC, chose the thresholds, compute the accuracy, generate confusion matrixes, ROC curves and strip-plot

[17]:
```
#creation of a pandas series that contains the UniProt ID and the class␣
↪(1=postive, 0=negative)
import pandas as pd
import numpy as np
from Bio import SeqIO
from Bio import SearchIO

#domain series creation
domain = {}
for record in SeqIO.parse("./negatives_kunitz2.fasta", "fasta"):
    ID_n = record.id.split("|")[1]
    domain[ID_n] = 0
for record in SeqIO.parse("./positives_clean_kunitz2.fasta", "fasta"):
    ID_p = record.id.split("|")[1]
    domain[ID_p] = 1
domain = pd.Series(domain, dtype='category')
domain
```

[17]:
```
Q4R8P0    0
P03949    0
Q9NPB9    0
P31937    0
Q9FFR3    0
         ..
P0DJ49    1
Q90W98    1
P0DJ68    1
P0DJ65    1
P0DJ77    1
Length: 562233, dtype: category
Categories (2, int64): [0, 1]
```

[18]:
```
#creation of a pandas series containing the UniProtID and the full sequence␣
↪E-value
hmm_results = SearchIO.read("./hmmsearch_output_kunitz.txt", "hmmer3-tab")
evalue = {}
for hit in hmm_results:
    ID = hit.id.split("|")[1]
    evalue[ID] = hit.evalue
evalue = pd.Series(evalue)
evalue
```

[18]:
```
Q868Z9    1.200000e-194
O76840    2.400000e-179
Q02445     8.500000e-68
P84875     6.000000e-67
O54819     4.300000e-66
               …
W4VS46     1.000000e+00
W5U5X5     1.000000e+00
X5CFH4     1.000000e+00
X5CWH9     1.000000e+00
X5IWT5     1.000000e+00
Length: 269835, dtype: float64
```

[19]:
```
#creation of a unique pandas dataframe containing ID, class and full seq E-value
dataframe = pd.DataFrame({'EVALUE': evalue, 'CLASS': domain})
dataframe['EVALUE'].fillna(10, inplace = True)
```

[20]:
```
#split of the dataframe in train and test subsets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import seaborn as sns
#full seq e-value=A
test_size = 0.5
seed = 42
X_trainA, X_testA, Y_trainA, Y_testA = train_test_split(dataframe.EVALUE,␣
↪dataframe.CLASS, test_size=test_size, random_state=seed)
X_trainA_2D = X_trainA.values.reshape(-1, 1)
X_testA_2D = X_testA.values.reshape(-1, 1)
```

[21]:
```
#computation of MCC for each E-value (full sequence) threshold (ranging from␣
↪1e-20 to 1)
from sklearn.metrics import matthews_corrcoef

mcc_list = []
for i in range(-20, 1, 1):
    Y_pred = X_trainA.apply(lambda x: 1 if x<10**(i) else 0)
    mcc_i = matthews_corrcoef(Y_trainA, Y_pred)
```

```
    mcc_list.append((i, mcc_i))
    Y_pred = Y_pred.iloc[0:0]
print(mcc_list)
```

```
[(-20, 0.9315898563939514), (-19, 0.9647700079360332), (-18,
0.9777274801272762), (-17, 0.9841423793035657), (-16, 0.9905158248411561), (-15,
0.9905158248411561), (-14, 0.9905158248411561), (-13, 0.9968486118555193), (-12,
0.9968486118555193), (-11, 0.9968486118555193), (-10, 0.9968486118555193), (-9,
0.9968486118555193), (-8, 0.9968486118555193), (-7, 0.9968486118555193), (-6,
0.9968683278055732), (-5, 0.9757400772256043), (-4, 0.9424462450536518), (-3,
0.7057965382493548), (-2, 0.28152877879163685), (-1, 0.06365695343897643), (0,
0.025608909085265894)]
```

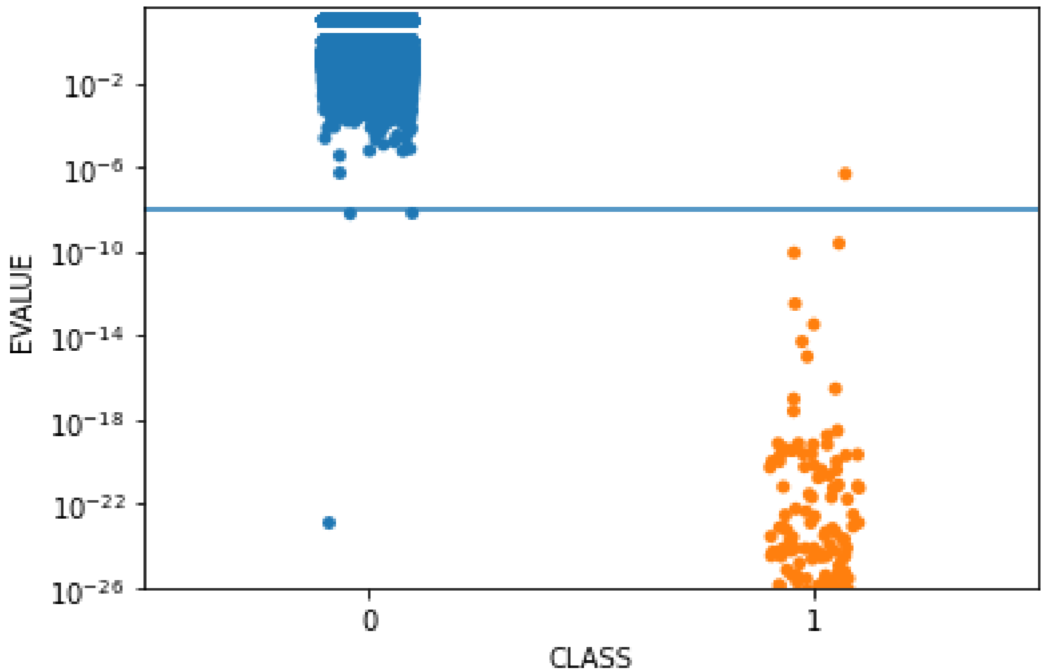
[22]:

```
#best th=10e-9 full sequence evalue
#application on test subset and computation of the stripplot
Y_predA = X_testA.apply(lambda x: 1 if x<10e-9 else 0)
mismatch = Y_predA.loc[Y_testA != Y_predA]
false_pos = mismatch.loc[mismatch == 1].index
false_neg = mismatch.loc[mismatch == 0].index
print('false positives are:', false_pos)
print('false negatives are:', false_neg)
striplot = sns.stripplot(y = X_testA, x = Y_testA)
striplot.set(yscale='log', ylim=(10e-27,40))
striplot.axhline(10e-9)
```

```
false positives are: Index(['P56409', 'P84555', 'G3LH89'], dtype='object')
false negatives are: Index(['D3GGZ8'], dtype='object')
```

[22]:

```
<matplotlib.lines.Line2D at 0x7f2a9668a820>
```



[23]:
```
import sklearn
sklearn.metrics.accuracy_score(Y_testA, Y_predA)
```

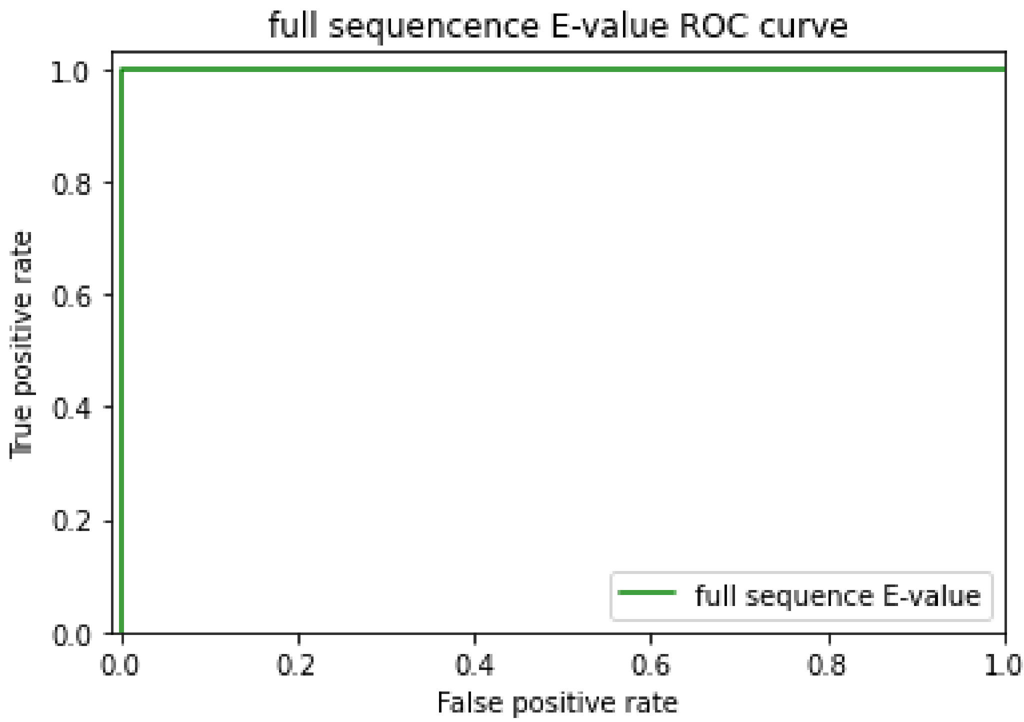
[23]: 0.9999857710490649

[24]:
```
#computation of confusion matrix full sequence evalue
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_testA, Y_predA)
```

[24]:
```
array([[280934,      3],
       [     1,    179]])
```

[13]:
```
#computation of ROC curve and AUC for full sequence e-value
import sklearn
from sklearn import metrics
import matplotlib.pyplot as plt
Y_score = [- values for values in X_testA]
fprA, tprA, thA = sklearn.metrics.roc_curve(Y_testA, Y_score)
aucA = metrics.auc(fprA, tprA)
#plot ROC curve
plt.title('full sequencence E-value ROC curve')
plt.plot(fprA, tprA, color='green', label='full sequence E-value')
plt.legend(loc='lower right')
plt.xlim([-0.01, 1])
plt.ylim([0, 1.03])
```

```
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
aucA
```



[13]: 0.9999999375550024

1.1 Repeating the same process for best domain E-value

[25]:
```
#evalue-best domain pandas series
hmm_results = SearchIO.read("./hmmsearch_output_kunitz.txt", "hmmer3-tab")
evalue = {}
for hit in hmm_results:
    ID = hit.id.split("|")[1]
    evalue[ID] = hit[0].evalue
evalue = pd.Series(evalue)
evalue
```

[25]:
```
Q868Z9    1.400000e-22
O76840    5.100000e-23
Q02445    2.300000e-26
P84875    1.400000e-25
```

```
O54819    1.600000e-25
               …
W4VS46    2.600000e-01
W5U5X5    1.000000e+00
X5CFH4    1.000000e+00
X5CWH9    1.000000e+00
X5IWT5    1.000000e+00
Length: 269835, dtype: float64
```

[26]:
```python
#creation of the dataframe
dataframe = pd.DataFrame({'EVALUE': evalue, 'CLASS': domain})
dataframe['EVALUE'].fillna(10, inplace = True)
```

[27]:
```python
#splitting in train and test subsets
#best domain e-value=B
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import plot_confusion_matrix
import seaborn as sns

test_size = 0.5
seed = 42
X_trainB, X_testB, Y_trainB, Y_testB = train_test_split(dataframe.EVALUE,␣
 ↪dataframe.CLASS, test_size=test_size, random_state=seed)
X_trainB_2D = X_trainB.values.reshape(-1, 1)
X_testB_2D = X_testB.values.reshape(-1, 1)
```

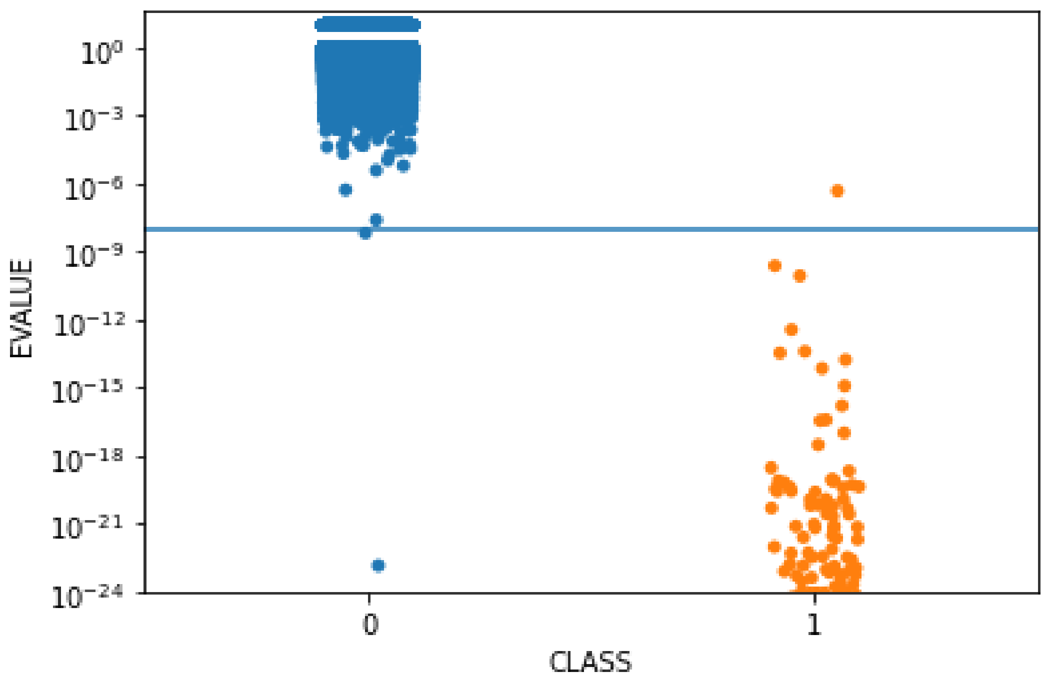
[28]:
```python
#computation of MCC for best domain evalue for each treshold (ranging from␣
 ↪1e-20 to 1)
mcc_list = []
for i in range(-20, 1, 1):
    Y_predB = X_trainB.apply(lambda x: 1 if x<10**(i) else 0)
    mcc_i = matthews_corrcoef(Y_trainB, Y_predB)
    mcc_list.append((i, mcc_i))
    Y_predB = Y_predB.iloc[0:0]
print(mcc_list)
```

```
[(-20, 0.9248112761244277), (-19, 0.9647700079360332), (-18,
0.9777274801272762), (-17, 0.9841423793035657), (-16, 0.9905158248411561), (-15,
0.9905158248411561), (-14, 0.9905158248411561), (-13, 0.9936872519621153), (-12,
0.9968486118555193), (-11, 0.9968486118555193), (-10, 0.9968486118555193), (-9,
0.9968486118555193), (-8, 0.9968486118555193), (-7, 0.9968486118555193), (-6,
0.9968683278055732), (-5, 0.9906921829851045), (-4, 0.9586601834225883), (-3,
0.7941936341055966), (-2, 0.3942139614918003), (-1, 0.12099208023473058), (0,
0.03803554166269458)]
```

[29]:
```
#best th= 10e-9 best domain evalue
#on test
Y_predB = X_testB.apply(lambda x: 1 if x<10e-9 else 0)
mismatch = Y_predB.loc[Y_testB != Y_predB]
false_pos = mismatch.loc[mismatch == 1].index
false_neg = mismatch.loc[mismatch == 0].index
print('false positives are:', false_pos)
print('false negatives are:', false_neg)
striplot = sns.stripplot(y = X_testB, x = Y_testB)
striplot.set(yscale='log', ylim=(10e-25,40))
striplot.axhline(10e-9)
```

```
false positives are: Index(['P84555', 'G3LH89'], dtype='object')
false negatives are: Index(['D3GGZ8'], dtype='object')
```

[29]: `<matplotlib.lines.Line2D at 0x7f2ab6f35700>`



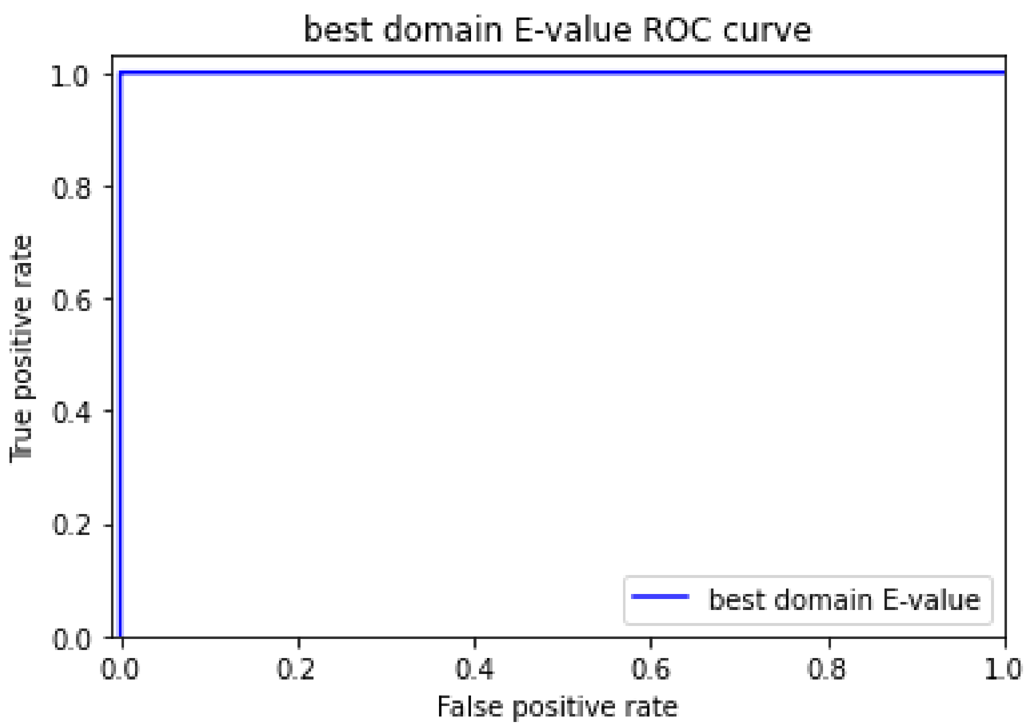
[30]:
```
sklearn.metrics.accuracy_score(Y_testB, Y_predB)
```

[30]: 0.9999893282867988

[31]:
```
#confusion matrix best domain
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_testB, Y_predB)
```

```
[31]: array([[280935,      2],
             [     1,    179]])
```

```
[32]: #computation of ROC curve and ACU for best domain e-value
import sklearn
from sklearn import metrics
import matplotlib.pyplot as plt
Y_score = [- values for values in X_testB]
fprB, tprB, thB = sklearn.metrics.roc_curve(Y_testB, Y_score)
aucB = metrics.auc(fprB, tprB)
#plot ROC curve
plt.title('best domain E-value ROC curve')
plt.plot(fprB, tprB, color='blue', label='best domain E-value')
plt.legend(loc='lower right')
plt.xlim([-0.01, 1])
plt.ylim([0, 1.03])
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
aucB
```




```
[32]: 0.9999987640566312
```

1.2 Switching train and test datasets

[33]:
```
X_trainA2 = X_testA
Y_trainA2 = Y_testA
X_testA2 = X_trainA
Y_testA2 = Y_trainA
X_trainA2_2D = X_testA_2D
X_testA2_2D = X_trainA_2D
```

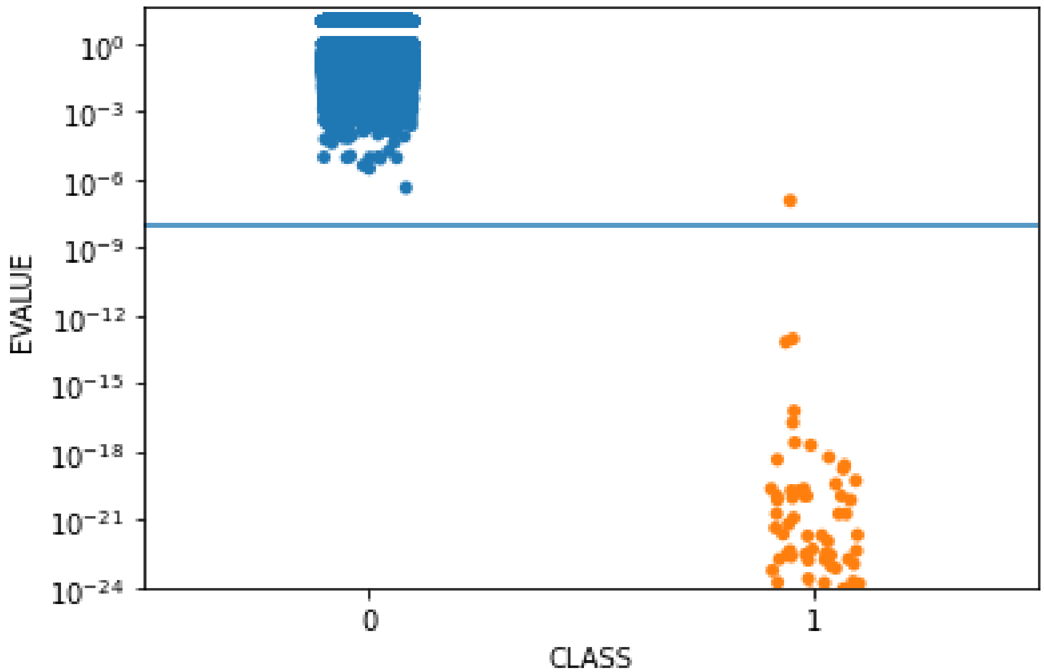
[34]:
```
#computation of MCC for each E-value (full sequence) threshold (ranging from␣
↪1e-20 to 1)
mcc_list = []
for i in range(-20, 1, 1):
    Y_predA2 = X_trainA2.apply(lambda x: 1 if x<10**(i) else 0)
    mcc_i = matthews_corrcoef(Y_trainA2, Y_predA2)
    mcc_list.append((i, mcc_i))
    Y_predA2 = Y_predA2.iloc[0:0]
print(mcc_list)
```

```
[(-20, 0.9158807286433993), (-19, 0.9632067522378225), (-18,
0.9689603822452433), (-17, 0.9746801343755602), (-16, 0.9775274921644231), (-15,
0.9803666014361492), (-14, 0.9831975336286215), (-13, 0.9860203591550949), (-12,
0.9888351474246505), (-11, 0.9888351474246505), (-10, 0.9916419668621258), (-9,
0.9944408849275421), (-8, 0.988958270501271), (-7, 0.988958270501271), (-6,
0.9890636688407966), (-5, 0.9784781776441691), (-4, 0.9370008776681715), (-3,
0.7158968993989808), (-2, 0.28962180946405724), (-1, 0.06735798253370169), (0,
0.027216163769347174)]
```

[35]:
```
#th= 10e-9 full seq evalue
#on test
Y_predA2 = X_testA2.apply(lambda x: 1 if x<10e-9 else 0)
mismatch = Y_predA2.loc[Y_testA2 != Y_predA2]
false_pos = mismatch.loc[mismatch == 1].index
false_neg = mismatch.loc[mismatch == 0].index
print('false positives are:', false_pos)
print('false negatives are:', false_neg)
striplot = sns.stripplot(y = X_testA2, x = Y_testA2)
striplot.set(yscale='log', ylim=(10e-25,40))
striplot.axhline(10e-9)
```

```
false positives are: Index([], dtype='object')
false negatives are: Index(['062247'], dtype='object')
```

[35]:
```
<matplotlib.lines.Line2D at 0x7f2a9c3d7f10>
```



[33]:
```
sklearn.metrics.accuracy_score(Y_testA2, Y_predA2)
```

[33]: 0.9999928856004553

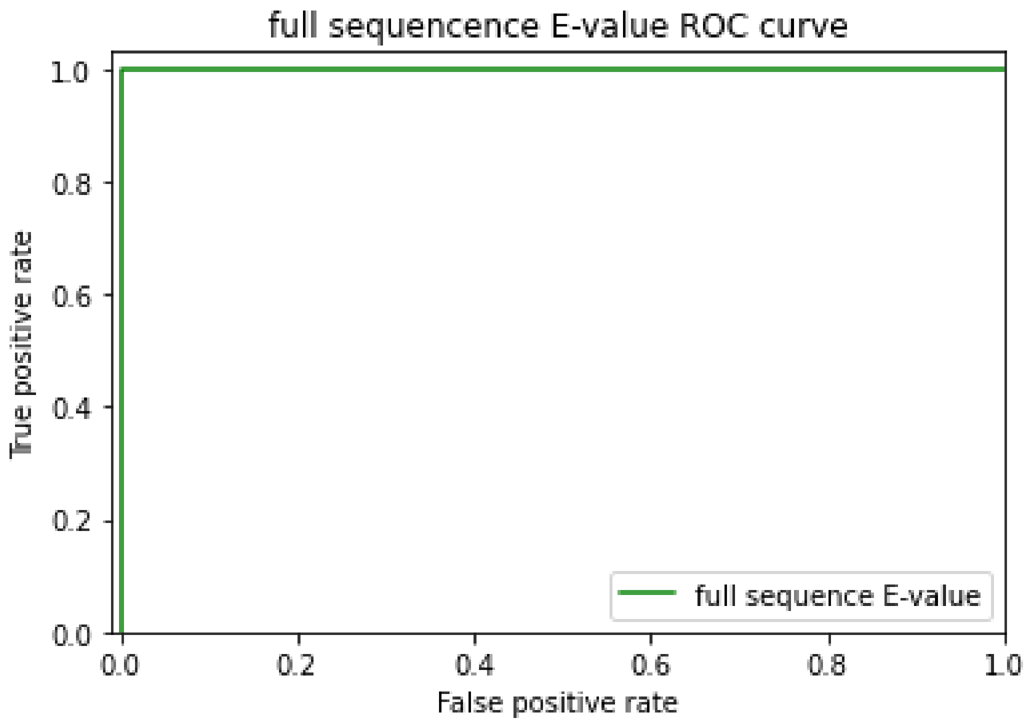
[34]:
```
#confusion matrix full seq
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_testA2, Y_predA2)
```

[34]:
```
array([[280943,      2],
       [     0,    175]])
```

[66]:
```
#computation of ROC curve and ACU for full seq E-value
import sklearn
from sklearn import metrics
import matplotlib.pyplot as plt

Y_score = [- values for values in X_testA2]
fprA2, tprA2, thA2 = sklearn.metrics.roc_curve(Y_testA2, Y_score)
aucA2 = metrics.auc(fprA2, tprA2)
#plot ROC curve
plt.title('full sequencence E-value ROC curve')
plt.plot(fprA2, tprA2, color='green', label='full sequence E-value')
plt.legend(loc='lower right')
plt.xlim([-0.01, 1])
plt.ylim([0, 1.03])
```

```
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
aucA2
```



[66]: 0.9999990440426826

[36]:
```
X_trainB2 = X_testB
Y_trainB2 = Y_testB
X_testB2 = X_trainB
Y_testB2 = Y_trainB
X_trainB2_2D = X_testB_2D
X_testB2_2D = X_trainB_2D
```

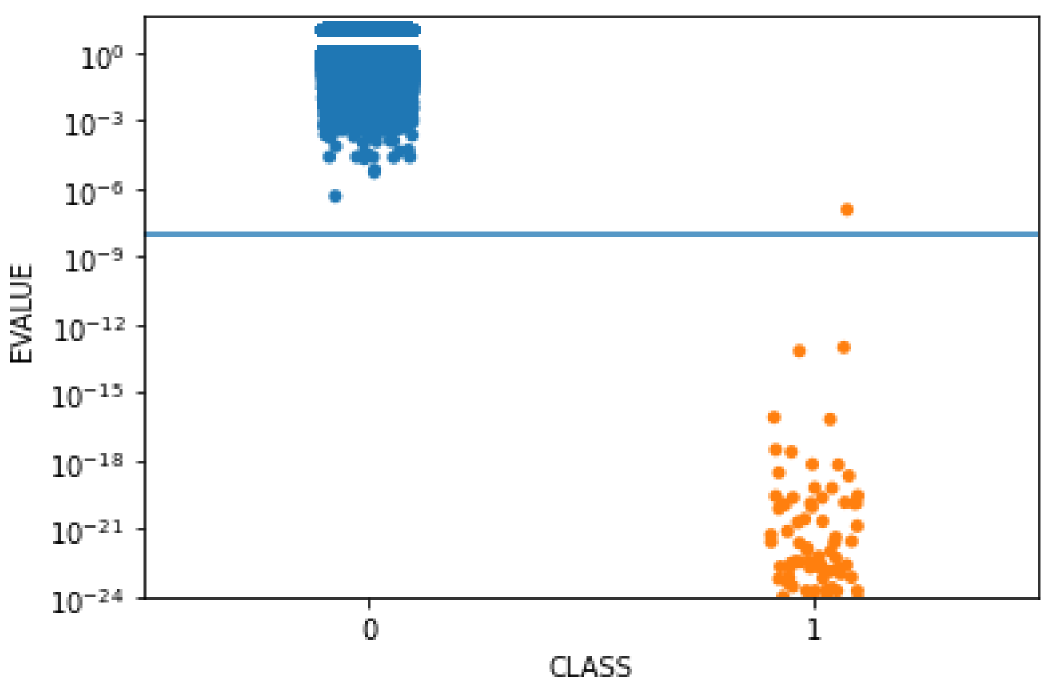
[37]:
```
#computation of MCC for best domain evalue for each treshold (ranging from␣
↪1e-20 to 1)
mcc_list = []
for i in range(-20, 1, 1):
    Y_predB2 = X_trainB2.apply(lambda x: 1 if x<10**(i) else 0)
    mcc_i = matthews_corrcoef(Y_trainB2, Y_predB2)
    mcc_list.append((i, mcc_i))
    Y_predB2 = Y_predB2.iloc[0:0]
print(mcc_list)
```

```
[(-20, 0.9036629750857237), (-19, 0.9515953976201543), (-18,
0.9574186337489125), (-17, 0.960317042931942), (-16, 0.9689603822452433), (-15,
0.9718244555860576), (-14, 0.9775274921644231), (-13, 0.9860203591550949), (-12,
0.9888351474246505), (-11, 0.9888351474246505), (-10, 0.9916419668621258), (-9,
0.9944408849275421), (-8, 0.9916882211371817), (-7, 0.988958270501271), (-6,
0.9890636688407966), (-5, 0.9837282487156767), (-4, 0.9486495289071499), (-3,
0.8016410149004117), (-2, 0.4081633118463328), (-1, 0.12760347838834654), (0,
0.040281293086292774)]
```

[38]:
```
#best th= 10e-9 best domain evalue
#Th applied on test
Y_predB2 = X_testB2.apply(lambda x: 1 if x<10e-9 else 0)
mismatch = Y_predB2.loc[Y_testB2 != Y_predB2]
false_pos = mismatch.loc[mismatch == 1].index
false_neg = mismatch.loc[mismatch == 0].index
print('false positives are:', false_pos)
print('false negatives are:', false_neg)
striplot = sns.stripplot(y = X_testB2, x = Y_testB2)
striplot.set(yscale='log', ylim=(10e-25,40))
striplot.axhline(10e-9)
```

```
false positives are: Index([], dtype='object')
false negatives are: Index(['062247'], dtype='object')
```

[38]: <matplotlib.lines.Line2D at 0x7f2a9c3d7e80>



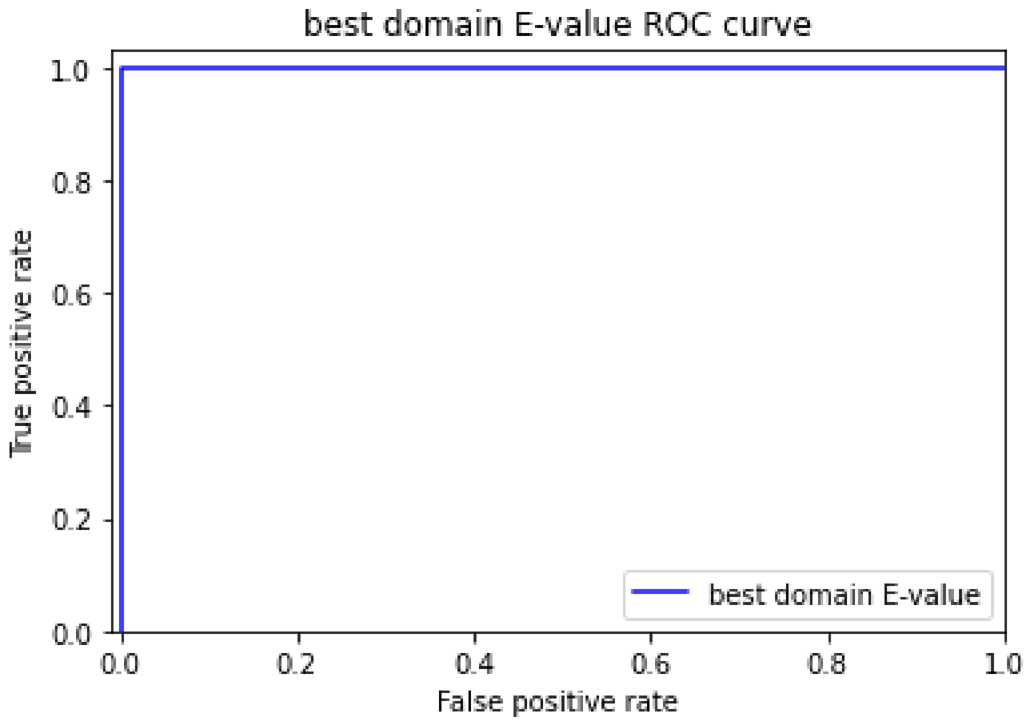
[39]:
```
sklearn.metrics.accuracy_score(Y_testB2, Y_predB2)
```

[39]: 0.9999964427496123

[40]:
```
#confusion matrix best domain
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_testB2, Y_predB2)
```

[40]:
```
array([[280957,      0],
       [     1,    158]])
```

[41]:
```
#computation of ROC curve and ACU for best domain e-value
import sklearn
from sklearn import metrics
import matplotlib.pyplot as plt
Y_score = [- values for values in X_testB2]
fprB2, tprB2, thB2 = sklearn.metrics.roc_curve(Y_testB2, Y_score)
aucB2 = metrics.auc(fprB2, tprB2)
#plot ROC curve
plt.title('best domain E-value ROC curve')
plt.plot(fprB2, tprB2, color='blue', label='best domain E-value')
plt.legend(loc='lower right')
plt.xlim([-0.01, 1])
plt.ylim([0, 1.03])
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
aucB2
```



[41]: 0.9999987796289563

[ ]: