Theoretical and practical metagenomic approaches to viral discovery

Practical Session: Random Forest Classifier and viral application

Kevin Lamkiewicz, Manja Marz

23.10.2019

European Virus Bioinformatics Center

How to: Implement the Random Forest Classifier

Random Forest

```
1 from sklearn.ensemble import RandomForestClassifier
2 # we know this already...
3 with open('virus.csv', 'r') as inputStream:
4 ...
```

Random Forest

```
1  from sklearn.ensemble import RandomForestClassifier
2  # we know this already...
3  with open('virus.csv', 'r') as inputStream:
4  ...
5
6  # create the classifier object
7  # n_estimators: number of trees for the forest
8  # max_depth: maximum depth of one tree
9  rfc = RandomForestClassifier(n_estimators=100, max_depth=2)
10 rfc = rfc.fit(data, target)
```

Training our model to identify viral elements

We want to identify viral pre-miRNAs

Our task:

In the `miRBase` database are currently around 320 viral pre-miRNAs. We want to use them in to train a machine learning model that can distinguish between viral pre-miRNAs and other sequences.

We want to identify viral pre-miRNAs

Our task:

In the `miRBase` database are currently around 320 viral pre-miRNAs. We want to use them in to train a machine learning model that can distinguish between viral pre-miRNAs and other sequences.

I already prepared...

- a file with all precursors
- a file with our negative data
- some utilities we need (reading the data, ...)

We want to identify viral pre-miRNAs

Our task:

In the `miRBase` database are currently around 320 viral pre-miRNAs. We want to use them in to train a machine learning model that can distinguish between viral pre-miRNAs and other sequences.

I already prepared...

- a file with all precursors
- a file with our negative data
- some utilities we need (reading the data, ...)
- Features?

EXERCISE

EXERCISE

Your task

Have a look at the `rf_classifier.py` script. Try to understand as much of it as possible.

Then, create a Random Forest Classifier and train the model with the data – `precursor.fa` and `negative_set.fa`

Let us discuss our results

First the basics

```
1 # first we need to translate the sequences into something
2 # that can be used by the machine learning algorithm:
3
```

First the basics

```
1  # first we need to translate the sequences into something
2  # that can be used by the machine learning algorithm:
3
4  import numpy as np
5
```

First the basics

```
1  # first we need to translate the sequences into something
2  # that can be used by the machine learning algorithm:
3
4  import numpy as np
5
6  # read in the data
7  trainingsData = read_training_set(positive, negative)
8
```

First the basics

```
# first we need to translate the sequences into something
# that can be used by the machine learning algorithm:

import numpy as np

# read in the data
trainingsData = read_training_set(positive, negative)

# transform the sequence into features
trainingSet, targets = transform_data(trainingsData)
```

Function Definitions

```
1  def transform_data(data):
2      # this will be our data structure
3      trainingSet = []
4      targets = []
5
```

Function Definitions

```
1  def transform_data(data):
2      # this will be our data structure
3      trainingSet = []
4      targets = []
5
6      # iterate over all sequences
7      for dataPoint in data:
8        sequence = dataPoint[0]
9        group = dataPoint[1]
10       # extract the features
11       length = len(sequence)
12       ...
13
```

Function Definitions

```
1  def transform_data(data):
2      # this will be our data structure
3      trainingSet = []
4      targets = []
5
6      # iterate over all sequences
7      for dataPoint in data:
8        sequence = dataPoint[0]
9        group = dataPoint[1]
10       # extract the features
11       length = len(sequence)
12       ...
13
14       # store everything in our array
15       vector = [length, ...]
16       trainingSet.append(vector)
17       targets.append(group)
18
```

Function Definitions

```
1  def transform_data(data):
2      # this will be our data structure
3      trainingSet = []
4      targets = []
5
6      # iterate over all sequences
7      for dataPoint in data:
8        sequence = dataPoint[0]
9        group = dataPoint[1]
10       # extract the features
11       length = len(sequence)
12       ...
13
14       # store everything in our array
15       vector = [length, ...]
16       trainingSet.append(vector)
17       targets.append(group)
18
19     # return the data within a numpy array
20     return(np.array(trainingSet, dtype=float), targets)
```

PERFORMANCE OF OUR MODEL

```
# we know this already. we split the data into training and test sets
data_training, data_test, target_training, target_test =
     train_test_split(trainingSet, targets, test_size=0.2)

# creating the random forest classifier and already fitting the training data
rfc = RandomForestClassifier(n_estimators=100, max_depth=2).
      fit(data_training, target_training)
# prediction of the test set
prediction = rfc.predict(data_test)
# output the confusion matrix to evaluate the model
print()
print(metrics.confusion_matrix(target_test, prediction))
print(metrics.accuracy_score(target_test, prediction))
```

COFFEE BREAK