Theoretical and practical metagenomic approaches to viral discovery

Practical Session: Practical Introduction to Machine Learning and SVMs

Kevin Lamkiewicz, Manja Marz

23.10.2019

European Virus Bioinformatics Center

WHOAMI

WHOAMI

- PhD student at Manja's Lab since 2017
- Interested in (RNA) viruses
- Focus on RNA secondary structures and ncRNAs
- Experience with Machine Learning, RNA structures, assemblies, alignments, SNP calling, ...

Introducing scikit-learn

BUILDING A SUPPORT VECTOR MACHINE WITH PYTHON3

```
1 #!/usr/bin/env python3
2
3 # to import scikit-learn we write the following
4 import sklearn
5
```

BUILDING A SUPPORT VECTOR MACHINE WITH PYTHON3

```
1 #!/usr/bin/env python3
2
3 # to import scikit-learn we write the following
4 import sklearn
5
6 # but actually we need this:
7 from sklearn import svm
8
```

BUILDING A SUPPORT VECTOR MACHINE WITH PYTHON3

```
#!/usr/bin/env python3

# to import scikit-learn we write the following
import sklearn

# but actually we need this:
from sklearn import svm

# now we create our SVM
classifier = svm.SVC(kernel='linear')

```

Building a support vector machine with Python3

```
#!/usr/bin/env python3

# to import scikit-learn we write the following
import sklearn

# but actually we need this:
from sklearn import svm

# now we create our SVM
classifier = svm.SVC(kernel='linear')

# Done.
```

UNDERSTANDING THE TWO LINES OF CODE

```
1 from sklearn import svm
```

- make all functions of the `svm` class available

UNDERSTANDING THE TWO LINES OF CODE

```
1 from sklearn import svm
```

- make all functions of the `svm` class available

```
1 classifier = svm.SVC(kernel='linear')
```

- create a classifier object with a linear kernel

Applying machine learning to real biological data

Our virus toy-data

```
1 import csv
2 from sklearn import svm
3 from sklearn.metrics import confusion_matrix, accuracy_score
4 # read and parse the dataset
5 data = []
6 target = []
```

Our virus toy-data

```
1 import csv
2 from sklearn import svm
3 from sklearn.metrics import confusion_matrix, accuracy_score
4 # read and parse the dataset
5 data = []
6 target = []
7 with open('virus.csv', 'r') as inputStream:
```

Our virus toy-data

```
1  import csv
2  from sklearn import svm
3  from sklearn.metrics import confusion_matrix, accuracy_score
4  # read and parse the dataset
5  data = []
6  target = []
7  with open('virus.csv', 'r') as inputStream:
8    reader = csv.reader(inputStream, delimiter=',')
9    for idx, row in enumerate(reader):
```

Our virus toy-data

```
1  import csv
2  from sklearn import svm
3  from sklearn.metrics import confusion_matrix, accuracy_score
4  # read and parse the dataset
5  data = []
6  target = []
7  with open('virus.csv', 'r') as inputStream:
8    reader = csv.reader(inputStream, delimiter=',')
9    for idx, row in enumerate(reader):
10     if idx == 0:
11       continue
```

Our virus toy-data

```
1  import csv
2  from sklearn import svm
3  from sklearn.metrics import confusion_matrix, accuracy_score
4  # read and parse the dataset
5  data = []
6  target = []
7  with open('virus.csv', 'r') as inputStream:
8    reader = csv.reader(inputStream, delimiter=',')
9    for idx, row in enumerate(reader):
10     if idx == 0:
11       continue
12   data.append(row[:4])
13   target.append(row[4])
```

Let's make a Machine Learning model

```
1 import csv
2 from sklearn import svm
3 from sklearn.metrics import confusion_matrix, accuracy_score
4
```

LET'S MAKE A MACHINE LEARNING MODEL

```
1 import csv
2 from sklearn import svm
3 from sklearn.metrics import confusion_matrix, accuracy_score
4
5 # create a classifer model via svm and train it with the data
6 svmLinear = svm.SVC(kernel='linear').fit(data, target)
7
```

LET'S MAKE A MACHINE LEARNING MODEL

```
1  import csv
2  from sklearn import svm
3  from sklearn.metrics import confusion_matrix, accuracy_score
4
5  # create a classifer model via svm and train it with the data
6  svmLinear = svm.SVC(kernel='linear').fit(data, target)
7
8  # predict with our model
9  prediction = svmLinear.predict(data)
10
```

LET'S MAKE A MACHINE LEARNING MODEL

```
import csv
from sklearn import svm
from sklearn.metrics import confusion_matrix, accuracy_score

# create a classifer model via svm and train it with the data
svmLinear = svm.SVC(kernel='linear').fit(data, target)

# predict with our model
prediction = svmLinear.predict(data)

# check performance of our prediction and model
confusion_matrix(target, prediction)
accuracy_score(target, prediction)
```

EASY, RIGHT?

Problems...!?

EASY, RIGHT?

Problems...!?

I used the same data for training and prediction (testing). This is very bad practice and should be avoided at all cost.

Our classifier learns based on the given training data - obviously, if the test data is identical to the training data, the model performs well.

We have to split our training set...

SPLITTING THE DATA IN TRAINING AND TEST SETS

```
1 from sklearn.model_selection import train_test_split
2
```

SPLITTING THE DATA IN TRAINING AND TEST SETS

```
1  from sklearn.model_selection import train_test_split
2
3  # here, we split our data set into different parts - for training and testing
4  # the method train_test_split() returns a tuple with 4 elements,
5  # which are stored in the appropriate variables, respectively.
6  data_train, data_test, target_train, target_test = train_test_split(
7          data, target, test_size=0.33)
8
```

SPLITTING THE DATA IN TRAINING AND TEST SETS

```
1  from sklearn.model_selection import train_test_split
2
3  # here, we split our data set into different parts - for training and testing
4  # the method train_test_split() returns a tuple with 4 elements,
5  # which are stored in the appropriate variables, respectively.
6  data_train, data_test, target_train, target_test = train_test_split(
7          data, target, test_size=0.33)
8
9  # train the model on the split data
10 svmLinear = svm.SVC(kernel='linear').fit(data_train, target_train)
11 # predict the targets of the "new data"
12 prediction = svmLinear.predict(data_test)
13
```

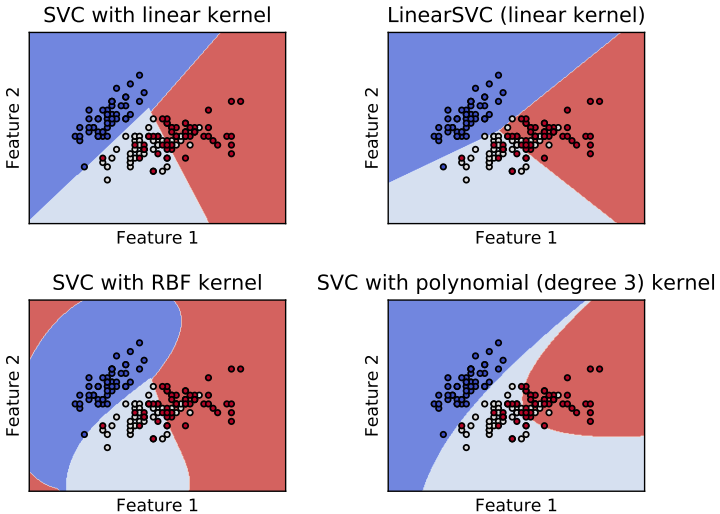
SPLITTING THE DATA IN TRAINING AND TEST SETS

```
1  from sklearn.model_selection import train_test_split
2
3  # here, we split our data set into different parts - for training and testing
4  # the method train_test_split() returns a tuple with 4 elements,
5  # which are stored in the appropriate variables, respectively.
6  data_train, data_test, target_train, target_test = train_test_split(
7          data, target, test_size=0.33)
8
9  # train the model on the split data
10 svmLinear = svm.SVC(kernel='linear').fit(data_train, target_train)
11 # predict the targets of the "new data"
12 prediction = svmLinear.predict(data_test)
13
14 # get the performance of this model
15 confusion_matrix(target_test, prediction)
16 accuracy_score(target_test, prediction)
```

SVM Kernel and other Hyperparamter

DIFFERENT KERNELS MAY YIELD DIFFERENT RESULTS...

DIFFERENT KERNELS MAY YIELD DIFFERENT RESULTS...



LET US CLASSIFY HANDWRITTEN DIGITS

```
1 from sklearn import svm, datasets
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 from sklearn.model_selection import train_test_split
4 # get the data
5 digits = datasets.load_digits()
6
```

LET US CLASSIFY HANDWRITTEN DIGITS

```
1  from sklearn import svm, datasets
2  from sklearn.metrics import confusion_matrix, accuracy_score
3  from sklearn.model_selection import train_test_split
4  # get the data
5  digits = datasets.load_digits()
6
7  # split everything
8  data_train, data_test, target_train, target_test = train_test_split(
9          data, target, test_size=0.33)
10 # train the model with RBF kernel on the training set
11 digitSVM = svm.SVC(kernel='rbf').fit(data_train, target_train)
12
```

LET US CLASSIFY HANDWRITTEN DIGITS

```
1  from sklearn import svm, datasets
2  from sklearn.metrics import confusion_matrix, accuracy_score
3  from sklearn.model_selection import train_test_split
4  # get the data
5  digits = datasets.load_digits()
6
7  # split everything
8  data_train, data_test, target_train, target_test = train_test_split(
9          data, target, test_size=0.33)
10 # train the model with RBF kernel on the training set
11 digitSVM = svm.SVC(kernel='rbf').fit(data_train, target_train)
12
13 prediction = digitSVM.predict(data_test)
14 confusion_matrix(target_test, prediction)
15 accuracy_score(target_test, prediction)
```

WELL, THAT DIDN'T WORK...

WELL, THAT DIDN'T WORK...

```
1 # train the model with linear kernel on the training set
2 digitSVM = svm.SVC(kernel='linear').fit(data_train, target_train)
3
```

WELL, THAT DIDN'T WORK...

```
# train the model with linear kernel on the training set
digitSVM = svm.SVC(kernel='linear').fit(data_train, target_train)

prediction = digitSVM.predict(data_test)
confusion_matrix(target_test, prediction)
accuracy_score(target_test, prediction)
```

CHANGING THE DEGREE OF OUR KERNEL FUNCTION

A linear kernel is essentially the same as a polynomial kernel with degree 1. So, what happens, if we change this parameter?

CHANGING THE DEGREE OF OUR KERNEL FUNCTION

A linear kernel is essentially the same as a polynomial kernel with degree 1. So, what happens, if we change this parameter?

```
# init a for loop to change the hyperparamter 'degree'
for d in range(1,11):
  # train different SVMs with a different kernel each time
  digitSVM = svm.SVC(kernel='poly', degree=d, gamma='auto').
    fit(data_train, target_train)
  prediction = digitSVM.predict(data_test)
  # some output things
  print(f"Degree of polynomial kernel: {d}")
  print(f"Accuracy of digit classification:
    {accuracy_score(target_test, prediction)}")
```

Coffee Break