

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

```
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
9 # now we create our SVM
10 classifier = svm.SVC(kernel='linear')
11
```

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
9 # now we create our SVM
10 classifier = svm.SVC(kernel='linear')
11
12 # 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

THE IRIS TEST DATA SET

```
1 from sklearn import svm, datasets  
2 from sklearn.metrics import confusion_matrix, accuracy_score  
3 # load the iris dataset  
4 iris = datasets.load_iris()  
5
```

THE IRIS TEST DATA SET

```
1 from sklearn import svm, datasets
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 # load the iris dataset
4 iris = datasets.load_iris()
5
6 # parse the important fields
7 data = iris.data
8 target = iris.target
9
```

THE IRIS TEST DATA SET

```
1 from sklearn import svm, datasets
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 # load the iris dataset
4 iris = datasets.load_iris()
5
6 # parse the important fields
7 data = iris.data
8 target = iris.target
9
10 # create a classifier model via svm and train it with the data
11 svmLinear = svm.SVC(kernel='linear').fit(data, target)
12
```

THE IRIS TEST DATA SET

```
1 from sklearn import svm, datasets
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 # load the iris dataset
4 iris = datasets.load_iris()
5
6 # parse the important fields
7 data = iris.data
8 target = iris.target
9
10 # create a classifier model via svm and train it with the data
11 svmLinear = svm.SVC(kernel='linear').fit(data, target)
12
13 # predict with our model
14 prediction = svmLinear.predict(data)
15
```

THE IRIS TEST DATA SET

```
1 from sklearn import svm, datasets
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 # load the iris dataset
4 iris = datasets.load_iris()
5
6 # parse the important fields
7 data = iris.data
8 target = iris.target
9
10 # create a classifier model via svm and train it with the data
11 svmLinear = svm.SVC(kernel='linear').fit(data, target)
12
13 # predict with our model
14 prediction = svmLinear.predict(data)
15
16 # check performance of our prediction and model
17 confusion_matrix(target, prediction)
18 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
```

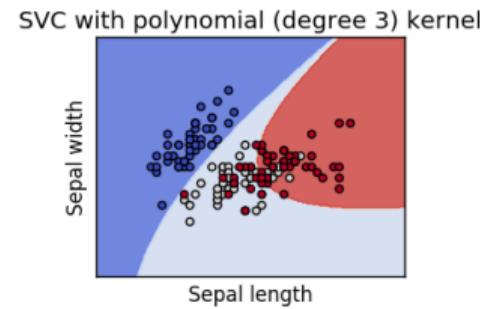
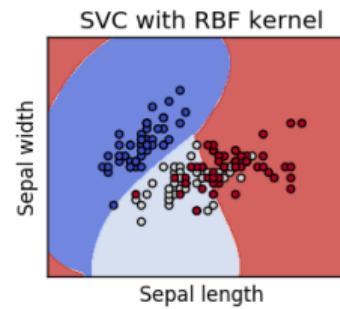
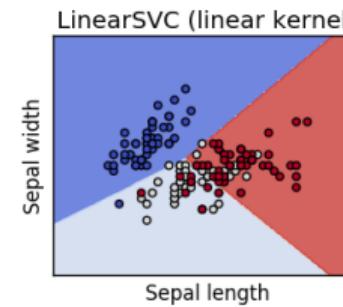
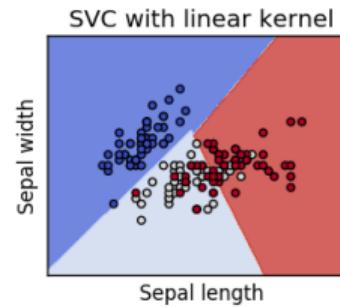
SPLITTING THE DATA IN TRAINING AND TEST SETS

```
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...

```
1 # train the model with linear kernel on the training set
2 digitSVM = svm.SVC(kernel='linear').fit(data_train, target_train)
3
4 prediction = digitSVM.predict(data_test)
5 confusion_matrix(target_test, prediction)
6 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?

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

COFFEE BREAK

