TU Berlin
Machine Learning 1

06 Homework

Learning Theory and Kernels

Jiayun, Fabi, Vincent, Lukas, Paul
December 13, 2020

Exercise 1: Mercer Kernels

(a) Show that the following are Mercer Kernels

i. $k(x, x') = \langle x, x' \rangle$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j \langle x, x' \rangle$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j \sum_{k}^{D} x_i^{(k)} x_j^{(k)} \text{ (The definition of the scalar product is introduced here.)}$$

$$= \sum_{k}^{D} \sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j x_i^{(k)} x_j^{(k)}$$

$$= \sum_{k}^{D} \left(\sum_{i=1}^{N} c_i x_i^{(k)} \right)^2 \geq 0$$

The inner sum is squared and thus the sum is greater or equal to 0 and so it is a Mercer Kernel.

ii. $k(x, x') = f(x) \cdot f(x')$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j f(x) \cdot f(x')$$

$$= \sum_{i=1}^{N} c_i f(x_i) \sum_{j=1}^{N} c_j f\left(x'_j\right)$$

$$= \left(\sum_{i=1}^{N} c_i f(x_i) \right)^2 \geq 0$$

It is a Mercer Kernel.

(b) Show that the following are again Mercer kernels.

1. $k(x, x') = k_1(x, x') + k_2(x, x')$

2. $k(x, x') = k_1(x, x')k_2(x, x')$

Mercer's Condition:
1. for the first kernel construction by summation, because k_1 and k_2 are mercer kernel respectively, so

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j (k_1(x_i, x_j') + k_2(x_i, x_j')) = \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k_1(x_i, x_j') + \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k_2(x_i, x_j') \geq 0$$

it indicates the summation of two M. kernel is M. kernel once again.
2. for the second kernel construction by multiplicity

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k_1(x_i, x_j') \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k_2(x_i, x_j') \geq 0$$

Because both term are non-negative, their multiplicity is then naturally non-negative.
Representer Theorem:
1. for the first kernel construction by summation

$$k(x, x') = k_1(x, x') + k_2(x, x') =< \varphi_1(x), \varphi_1(x') >_F + < \varphi_2(x), \varphi_2(x') >_F$$

For the definition of the inner product is scalar product, and we assume feature map $v\varphi_1$ has M dimension and $v\varphi_2$ has N dimension. we could write it out like:

$$\sum_{i=1}^{M} \varphi_1^i(x)\varphi_1^i(x') + \sum_{i=1}^{N} \varphi_2^i(x)\varphi_2^i(x') = \sum_{k=1}^{M+N} \phi_k(x)\phi_k(x') =< \phi(x), \phi(x') >_F$$

which we regard $\phi(x)$ as condensed feature map consisted from $\varphi_1(x)$ and $\varphi_2(x)$, defined as below:

$$\phi_k(x) = \begin{cases} \varphi_1^i(x), & 1 \leq k \leq M, i = k \\ \varphi_2^i(x), & M+1 \leq k \leq M+N, i = k-M \end{cases} \tag{1}$$

2. for the second kernel construction by multiplicity

$$k(x, x') = k_1(x, x')k_2(x, x') =< \varphi_1(x), \varphi_1(x') >_F < \varphi_2(x), \varphi_2(x') >_F$$

It could be written as

$$\sum_{i=1}^{M} \varphi_1^i(x)\varphi_1^i(x') \sum_{i=1}^{N} \varphi_2^i(x)\varphi_2^i(x') = \sum_{i=1}^{M}\sum_{j=1}^{N} (\varphi_1^i(x)\varphi_2^i(x))(\varphi_1^i(x')\varphi_2^i(x')) = \sum_{k=1}^{MN} \phi_k(x)\phi_k(x') =< \phi(x), \phi(x') >_F$$

Here $\phi_k(x)$ is MN dimensional and $\phi_k(x) = \varphi_1^i(x)\varphi_2^j(x)$ starting from $\varphi_1^1(x)\varphi_2^1(x)$ to $\varphi_1^1(x)\varphi_2^N(x)$ and then until $\varphi_1^M(x)\varphi_2^N(x)$ MN terms in total.

(c) Show using the results above that the polynomial kernel of degree d, where $k(x, x') = (< x, x' > +\vartheta)^d$ and $\vartheta \in R^+$, is a Mercer kernel.

We want to show $k(x, x') = (< x, x' > +\vartheta)^d$, thus we need to use summation rule proved in b.1 section and then multiplicity rule recursively that proved in b.2. We know inner product is M.kernel in a.1 then we need to prove ϑ where $\vartheta \in R^+$ is a M. kernel.

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \vartheta = \vartheta \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j = \vartheta(\sum_{i=1}^{N} c_i)^2 \geq 0$$

So ϑ is M.kernel, according to b.1 $< x, x' > +\vartheta$ is then M.kernel, and $(< x, x' > +\vartheta)^d$ will then recursively be M.kernel by b.2.

(2a)

In order to show that $\phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ is a possible feature map, we used the definition of the polynomial kernel and built the scalar product:

$$\langle \phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \phi \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \rangle = \langle \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}, \begin{pmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{pmatrix} \rangle = x_1^2y_1^2 + 2y_1y_2x_1x_2 + x_2^2y_2^2$$
$$= (\sum_{i=1}^{2} x_iy_i)^2$$

Since the kernel definition is fulfilled for this candidate, it is a possible feature map of the polynomial kernel with dimension 2. $F = \mathbb{R}^3$ as possible feature space is also fulfilled, because of the third dimension of the feature map.

(2b)

Using the plane equation:

$$H : \left[x - p\right] n = 0$$

and using:

$$p = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, n = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \phi(C) = \begin{pmatrix} \cos^2\phi \\ \sqrt{2}\cos\phi\sin\phi \\ \sin^2\phi \end{pmatrix}$$

we can test if $\phi(C)$ lies in the plane H:

$$[\begin{pmatrix} \cos^2\phi \\ \sqrt{2}\cos\phi\sin\phi \\ \sin^2\phi \end{pmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \cos^2\phi - 1 + \sin^2\phi = 0$$

Thus a plane exists in which the image of the unit circle C lies.

(2c)

In order to find a point P in F that does not lie in the $\phi(A)$, we first calculate $\phi(A)$:

$$\phi(A) = \begin{bmatrix} t^2 \\ \sqrt{2}ts \\ s^2 \end{bmatrix},$$

with t, s $\in \mathbb{R}$

As the value t^2 can only hold positive values we can construct e.g. the point:

$$P = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix},$$

P lies in F but not in $\phi(A)$.

(2d)

$$(\sum_{i=1}^{d} x_i y_i)^2 = x_1^2 y_1^2 + \cdots + x_d^2 y_d^2 + \sum_{i=1}^{d} \sum_{j=1}^{d} x_i y_j x_j y_i$$

This follows for the feature map to be:

$$\phi \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} = \begin{bmatrix} x_i^2 \\ \vdots \\ x_d^2 \\ \sqrt{2}x_1x_2 \\ \vdots \\ \sqrt{2}x_1x_d \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_2x_d \\ \vdots \\ \sqrt{2}x_{d-1}x_d \end{bmatrix}$$

sheet06_lukas

December 13, 2020

[1]:
```
import numpy as np
import sklearn
%matplotlib inline
```

1 Learning curves and error bounds

In this exercise, we test the performance of a classifier as we vary its complexity and the number of data points used for training, and compare the error with two bounds: the Hoeffdings bound, and the VC bound. As a first step, we load some artificial dataset where the two classes are 1000-dimensional Gaussians of same (isotropic) covariance and of different means.

[2]:
```
import utils
X,T = utils.getdata()
print(X.shape,T.shape)
```

```
(20000, 1000) (20000,)
```

We consider the perceptron as a training algorithm.

[3]:
```
import sklearn.linear_model
model = sklearn.linear_model.Perceptron()
```

1.0.1 Measuring the error (15 P)

The first task is to build a function that estimates the classification error (the 0/1 loss) when training the data on N examples (sampled randomly from the dataset), and observing only the first d dimensions of the data. Your function should output the training error and the test error (measured on the points that have not been used for training). In order to get robust error estimates, run the procedure 10 times with different training/test splits, and compute the average training and test error over the 10 splits.

[4]:
```
def geterr(N,d,X,T):

    # ----------------------------
    # TODO: replace by your code
    # ----------------------------
```

```
    etrain, etest=0,0
    for _ in range(10):
        trainind = np.random.randint(X.shape[0], size=(N,))
        #print(len(trainind)==N)
        testind = np.array([i for i in range(0,X.shape[0]) if i not in␣
↪trainind])
        Xtrain=X[trainind,:d]
        Ttrain=T[trainind]
        Xtest=X[testind,:d]
        Ttest=T[testind]
        model.fit(Xtrain, Ttrain)
        etrain += np.count_nonzero(Ttrain != model.predict(Xtrain))/len(Xtrain)
        etest += np.count_nonzero(Ttest != model.predict(Xtest))/len(Xtest)
    etrain/=10
    etest/=10
    # ------------------------------

    return etrain,etest
```

1.0.2 Bounding the test error (15 P)

We would like to compare the test error estimates with error bounds seen during the lecture. We consider in particular the Hoeffding's bound given by

$$R[f] \leq R_{\text{emp}}[f] + \sqrt{\frac{\log(2/\delta)}{2N}}$$

which makes the assumption that the function f was not learned, but predefined. Note that this assumption does not hold in practice, and we therefore expect the Hoeffding's bound to not be a bound of the test error. We will verify this in the subsequent experiments.

We also consider the VC bound, which does not have this limitation and is therefore a true bound of the test error. The VC bound is given by:

$$R[f] \leq R_{\text{emp}}[f] + 2\sqrt{2\frac{h_{\mathcal{F}}(\log(2N/h_{\mathcal{F}}) + 1) + \log(2/\delta)}{N}}$$

where $h_{\mathcal{F}}$ is the VC-dimension that measures the model complexity, and we recall that $h_{\mathcal{F}} = d + 1$ for the set of linear models.

In the following, we would like to implement these two bounds, so that they can be compared to the test error.

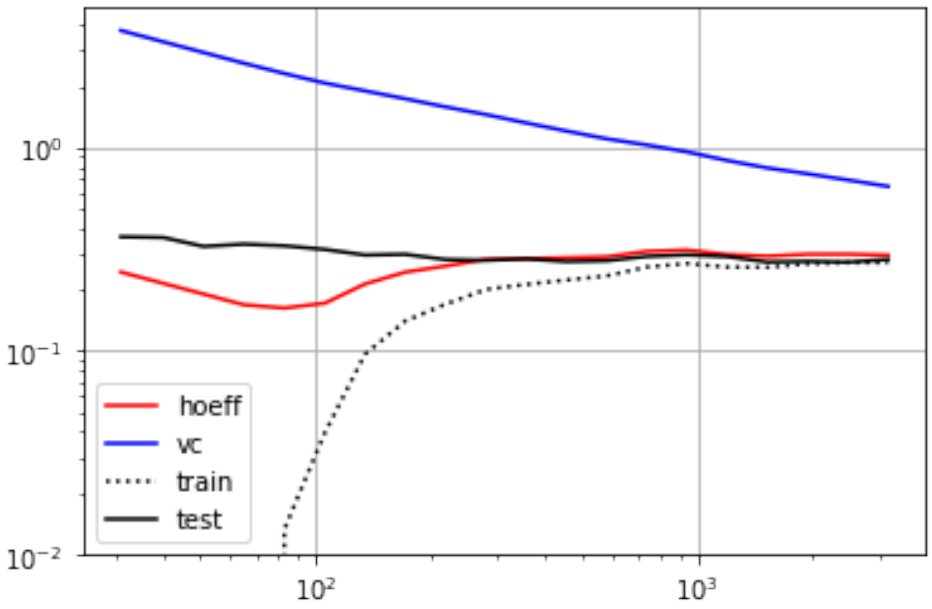
[5]:
```
def hoeffding(Etrain,N,delta=0.05):

    # ------------------------------
    # TODO: replace by your code
    # ------------------------------
    val=Etrain+np.sqrt(np.log(2/delta)/(2*N))
```

```
    # ----------------------------

    return val

def vc(Etrain,N,d,delta=0.05):

    # ----------------------------
    # TODO: replace by your code
    # ----------------------------
    val=Etrain+2*np.sqrt(2*(d+1+np.log(2/delta))/N)
    # ----------------------------

    return val
```

Once these functions have been implemented, test your implementation by running the following two experiments, where we plot the training and test error and the two bounds when varying different parameters.

In the first experiment, we first keep the number of observed dimensions d fixed and vary the number of data points N used for training.

[6]:
```
Ns = np.logspace(1.5,3.5,20).astype('int')
utils.getcurves(X,T,zip(Ns,[50]*len(Ns)),Ns,geterr,hoeffding,vc)
```

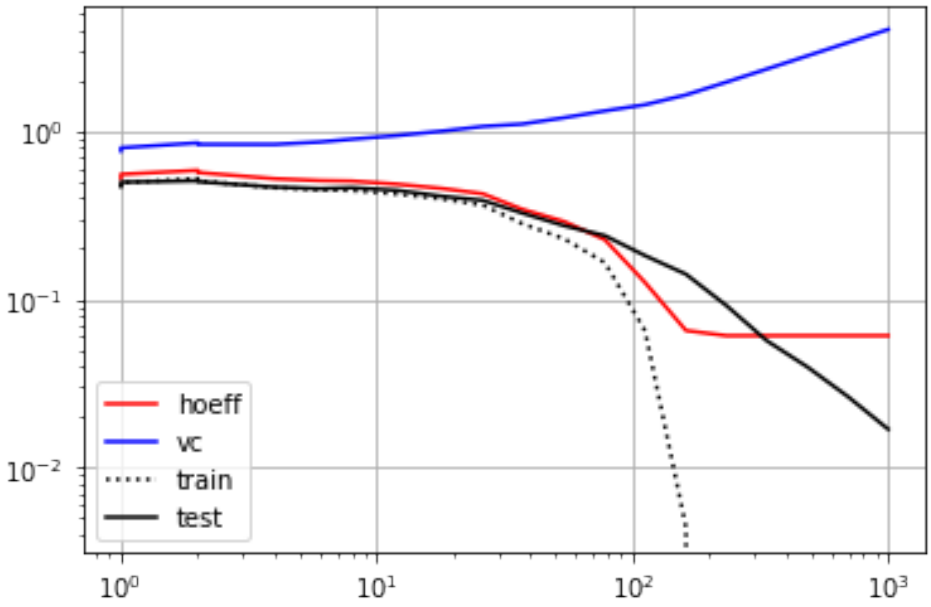


We observe that the Hoeffding's bound is indeed for some values of N below the test error. There-

fore, it is not a bound of the test error. Instead, the VC bound is always above the test error. While the bound is loose (it never actually reaches an error below 1) it still correctly predicts the benefit of including more data for improving the test error.

In the second experiment, we keep the number of training points N fixed and we vary the number of dimensions d used for training.

[7]:
```
ds = np.logspace(0,3,20).astype('int')
utils.getcurves(X,T,zip([500]*len(ds),ds),ds,geterr,hoeffding,vc)
```



Here again, the Hoeffding's bound is not a bound of the test error as it produces values below that error. Here, the VC bound varies in opposite direction to the test error. This can be due to the fact that we have set the VC-dimension to that of a general d-dimensional classifier, without taking in to account other aspects such as margin, which has a tendency to increase as we observe more dimensions.