Week 6 Lab

The questions below are due on Sunday October 22, 2017; 11:00:00 PM.

This lab is divided into two parts:

1. Checkoff on kernels
2. Implementing kernel methods

Part 1 will require a checkoff with a staff member.

A code and data folder that will be useful for doing this lab can be found here (https://introml.mit.edu/__STATIC__/fall17/labs/lab06/code_for_lab6.py.zip). Download this to your computer.

1) CHECKOFF ON KERNELS

1.1) Radial Basis Kernel

One very popular and useful kernel is the radial basis kernel (also often called the Gaussian kernel); it is defined as

$$K(x, z) = \exp\left(-\beta \|x - z\|^2\right) = \exp\left(-\beta (x - z) \cdot (x - z)\right).$$

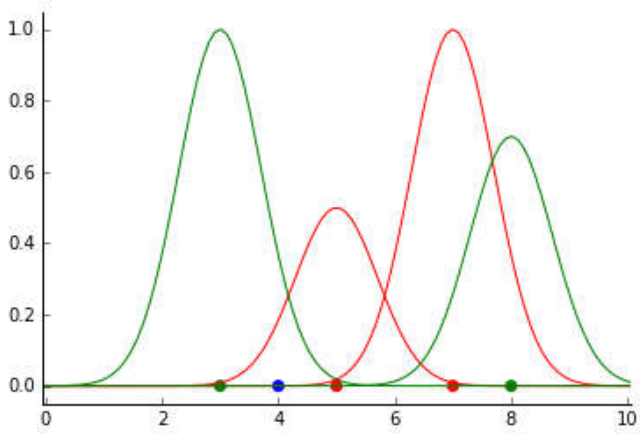
The radial basis kernel is maximized when $x = z$ and decreases exponentially with the distance between x and z. Note that the radial basis kernel is isotropic, and decays with distance regardless of direction, with that decay rate governed by the parameter β. That is, a larger value of β means that the kernel value decreases faster with increasing distance between x and z.

The first thing to notice is the form of the predictor that we get when we use the radial-basis (or Gaussian) kernel. If we want to make a prediction for a new input point z, we make a weighted combination of the labels of the training examples, with the weights set by nearness to the to input point and by the α_i's determined from the training algorithm:

$$pred(z; X, Y, \alpha) = sign\left(\sum_{i=1}^{n} \alpha_i y^{(i)} \exp\left(-\beta \|x^{(i)} - z\|^2\right)\right)$$

1) Given the training data and alpha values below, what prediction would you make for $z = 4$?

```
X = np.array([[5, 7, 3, 8]])
Y = np.array([[-1, -1, 1, 1]])
alpha = np.array([[.5, 1, 1, .7]])
beta = 1
```



In the above figure, we show the radial basis kernel $K(x, z)$ as a function of z for each of the x's in the set of four training points, colored green for training points with positive labels and red for ones with negative labels. To compute the prediction for the test point $z = 4$ (the blue point), we first compute the kernel values $K(5, 4)$, $K(7, 4)$, $K(3, 4)$ and $K(8, 4)$, and then use these kernel values, along with the vector α determined from a training algorithm, to determine a weighted sum of training point labels. The sign of the weighted sum is then the predicted value.

a)

What is floating point value of the weighted combination of labels for the above training set and the test point $z = 4$ (before you take the `sign`).

0.38381638955625663

Ask for Help

b)

Enter the predicted label:

1

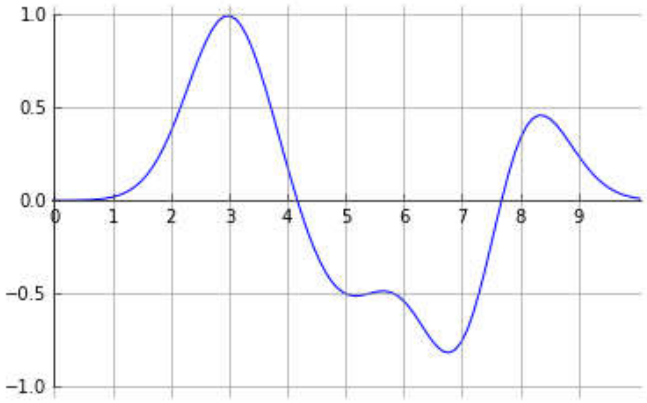
Ask for Help

2)

Is the above training data linearly separable? No

Ask for Help

It's instructive to look at the sum of weighted labels, before taking the sign. We'll plot that, as a function of x in the plot below.



3) What are the predictions from this hypothesis (described by the data and α) for the training data?

Enter a list of labels for the training data [3, 5, 7, 8]

```
[1, -1, -1, 1]
```

Ask for Help

4)

Does this hypothesis (described by the data and α) correctly predict the training data? Yes

Ask for Help

Describe the separator "encoded" by the data and α values, by indicating the points where the boundaries between positive and negative regions of the x space occur. You only have to be accurate to within 0.2.

5)

Enter a list of floats (in numerical order)

```
[-0.5, 2] #theta, theta_0
```

Ask for Help

1.2) Discrete Feature vectors

This basic idea, of weighting the training points' labels based on the their distance from the test point, is very intuitive. But it is kind of hard to interpret it as the dot product of a feature vector. To understand it this way, we will start by making a discrete approximation to the trainsformed feature vector $\phi(x)$.

We will assume our data values are all between $l = 0$ and $u = 10$, and for now, arbitrarily discretize the range into $m = 10$ segments of width w (in this example, w = 1). Then, let $\phi(x)$ be an m-dimensional vector whose i^{th} component is given by

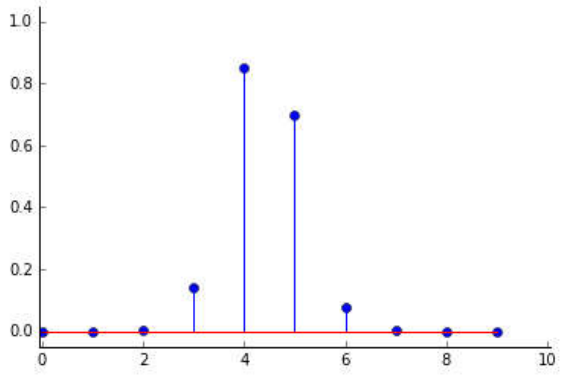
$$\phi(x)_i = \exp\left(-\beta \|x - iw\|^2\right)$$

for $i \in (1, \ldots, m)]$.

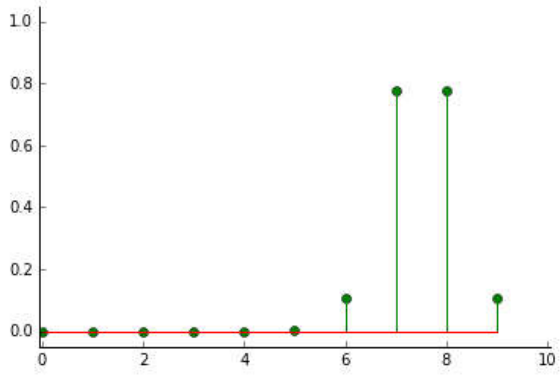
Below are plots of two example feature vectors, as well as a procedure for computing them:

```
def dgphi(x, l = 0, m = 10, u = 10, beta = 1):
    w = (u - l) / m
    return np.array([np.exp(-beta * (x - w*i)**2) for i in range(m)]).reshape(-1,1)
```

1)



2)



The left plot corresponds to `dgphi(4.4)`; the right plot corresponds to `dgphi(7.5)`.

What is the feature vector $\phi(2)$? It's okay to use code to compute it, doing it by hand is too tedious.

Enter a Python list of 10 values.

```
[1.83156389e-02, 3.67879441e-01, 1.00000000e+00, 3
```

Ask for Help

Now, let's think about the associate kernel $K(x, z) = \phi(x) \cdot \phi(z)$. What is $K(2, 3)$? $K(2, 5)$?

Enter a Python list of two values: [a, b], where a=$K(2,3)$ and b=$K(2,5)$

```
[0.7492393 , 0.0137228]
```

Ask for Help

1.3) Radial Basis Function (RBF) Kernel

It might seem surprising that we can think of the RBF kernel as acting like the dot products of the m-dimensional vectors we generated above, but we can. For example, examining the plots above should make it clear that $\phi(x) \cdot \phi(x')$ is maximized when $x = x'$, and that the dot product decreases as the distance between x and x' increases, just like $K(x, x')$ for the RBF kernel.

So, we can think, now, of the RBF kernel as being the "dot product" between two infinite feature "functions". If the original dimension is 1, then it is analogous to the above example, except that we have to let the spacing, w, become infinitely fine and allow the lower and upper limits on m go to negative and positive infinity. In that limit, the "dot product" becomes an integral:

$$K(x,z) = \int_{w=-\infty}^{\infty} \exp\left(-\beta(x-w)(x-w)\right) \exp\left(-\beta(z-w)(z-w)\right) dw$$

which happens to come out just right (up to a constant):

$$K(x,z) = \frac{\sqrt{\pi}}{\sqrt{\beta}} \exp\left(-\beta(x-z)(x-z)\right)$$

This integral identity is far from trivial, and is well outside 6.036. But it justifies a simple intuition, that using the RBF kernel is like using a feature vector that is an infinite-dimensional transformation of x, one that looks like the Gaussian function centered on x.

What would have to change in order for this kernel to work in a higher-dimensional input space (where x and z are d-dimensional column vectors)?

- [ ] We would have to design a new kernel function.
- [x] We could replace the expression $(x-z)(x-z)$ by $(x-z)^T(x-z)$

We could change it to something of the form

- [x] $$\exp\left(-\beta \sum_{j=1}^{d}((x_j - z_j)(x_j - z_j))\right)$$

We could change it to something of the form

- [ ] $$\exp\left(-\beta \prod_{j=1}^{d}((x_j - z_j)(x_j - z_j))\right)$$

Ask for Help

1.4) String Theory

We're interested in generalizing linear regression to the setting in which the input to our regressor will be strings of arbitrary length, and the output will be a real number. We will explore the extension of ordinary least-squares regression to use a kernel function.

Pat claims the following function is a kernel:

$$K(x,z) = \sum_{\beta \in \text{alphabet}} (\text{no. occurrences of } \beta \text{ in x})(\text{no. occurrences of } \beta \text{ in z})$$

where the alphabet is the set of Roman characters 'a' through 'z'. We will perform a kernelized regression, finding parameters α_i, so that the predictions are of the form:

$$y(x) = \sum_{i=1}^{N} \alpha_i K(x^{(i)}, x).$$

Answer the following questions assuming the training data is:

x	y
``abalone''	10
``xyzygy''	1
``zigzag''	3

1) What is the feature vector associated with Pat's kernel?

- ○ A vector of integer counts of how many times each letter pair occurs in the word.
- ◉ A vector of integer counts of how many times each letter occurs in the word.
- ○ A vector of products of counts of how many times each letter in a pair occurs in the word.
- ○ A vector of products of counts of how many times each letter occurs in the word.

Ask for Help

2) Determine an expression for $y(\text{“ziggy”})$ in terms of the α_i parameters.

You can use the symbols `alpha_1`, `alpha_2` and `alpha_3` as well as constants.

$y('ziggy') =$ `alpha_1*0 + alpha_2*6 + alpha_3*7`

Ask for Help

3) The vector α can be determined by solving a system of equations of the form $A\alpha = b$. Give the numerical values for matrix A and vector b. a)

Enter the value in A as a list of lists (each representing a row of A).

`[[9.0, 0.0, 2.0],[0.0, 12.0, 4.0],[2.0, 4.0, 10.0]`

Ask for Help

b)

Enter a list of numbers representing b

`[10, 1, 3]`

Ask for Help

1.5) Experimenting with kernel methods

We will be looking at two kernel methods described in the notes: kernel perceptron and kernel linear regression using both the polynomial kernel and the radial basis kernel.

Do not click on View Answer in this section since that will prevent you from looking at new plots

1.5.1) Kernel Perceptron

The function `t1`, by default, computes and displays the output of the kernel perceptron using a polynomial kernel of the specified order, which defaults to 1. Try changing the order of the polynomial to see the effect on the separator.

```
1 def run():
2     return t1(order=1)
3
```

Ask for Help

1. What is the minimum order required to achieve separation of the input data?

The function `t2`, by default, computes and displays the output of the kernel perceptron using a radial basis kernel with the specified β, which defaults to 0.01. Try changing the beta to see the effect on the separator.

```
1 def run():
2     return t2(beta=0.01)
3
```

Ask for Help

1. What is the minimum value of β to achieve separation of the input data?
2. Explain what is happening when $\beta = 100$?

1.5.2) Kernel Regression

The function `t3`, by default, computes and displays (in orange) the output of the kernel linear regression using a polynomial kernel with the specified order, which defaults to 1, and the specified lambda, which defaults to 0.001. You will also see a display (in green) of the regression with polynomial features of the same order (and no regularization) that we saw in Lab 4.

Try changing the order and the lambda to see the effect on the regressor.

```
1 def run():
2     return t3(order=1, lam=0.001)
3
```

Ask for Help

1. What is the relationship between the two curves as lambda approaches 0?

The function `t4`, by default, computes and displays (in orange) the output of the kernel linear regression using a radial basis kernel with the specified beta, which defaults to 0.01, and the specified lambda, which defaults to 0.001.

Try changing the beta and the lambda to see the effect on the regressor.

```
1 def run():
2     return t4(beta=0.01, lam=0.001)
3
```

Ask for Help

1. How does the regression behave for β near 0?
2. How does the regression behave for β large?

1.6) Checkoff

Checkoff 1:

Have a check-off conversation with a staff member, to explain your answers.

Ask for Help | Ask for Checkoff

2) IMPLEMENTING KERNEL METHODS

We will now implement these two kernel methods.

2.1) Kernel Functions

Implement the kernel functions for polynomial kernel of a given order n:

$$K(x, z) = (x^T z + 1)^n$$

and for radial basis (Gaussian) kernel with given β.

$$K(x, z) = \exp\left(-\beta(x - z)^T(x - z)\right)$$

`x` and `z` are d-dimensional column vectors. The output should be a float.

```
def k_poly(x, z, n=1):
    return (np.dot(x.T, z) + 1)**n


def k_gauss(x, z, beta=1):
    return math.exp(-beta*(np.dot((x-z).T, x-z)))

```

Ask for Help

2.2) Kernel Perceptron

The kernel perceptron (`kernel_perceptron`) is called as follows:

- `X`: a dxn data matrix
- `Y`: a 1xn vector of labels
- `k`: a kernel function of two arguments, each of which is a column vector
- `T`: number of iterations through the data

It returns:

- alphas: a 1xn vector

```
def kernel_perceptron(X, Y, k, T =10):
    d, n = x.shape
    alphas = np.zeros(n).reshape(1,n)
    counter = 0
    while(counter < T):
        for i in range(n):
            counter+=1
            if Y[i]*predict(X, Y, alpha, X[:,[i]], k) <= 0:
                alphas[i]+=1
    return alphas

```

Ask for Help

2.3) Kernel Linear Regression

The kernel linear regression function (`kernel_lin_reg`) is called as follows:

- `X`: a dxn data matrix
- `Y`: a 1xn vector of targets
- `lam`: a float, the value of the regularization parameter
- `k`: a kernel function of two arguments, each of which is a column vector

It returns:

- `alphas`: a 1xn vector

You will need to compute the Gram matrix $K(i, J)$ for the specified kernel function. The function `gram` takes:

- `X`: a dxn data matrix
- `k`: a kernel function of two arguments, each of which is a column vector

It returns:

- `K`: an nxn matrix

You should also write the prediction function `pred`:

- `X`: a dxn data matrix
- `alphas`: a 1xn vector
- `x`: a dx1 data point
- `k`: a kernel function of two arguments, each of which is a column vector

It returns:

- a float

```
1 def gram(X, k):
2     d, n = X.shape
3     return np.vstack((np.apply_along_axis(k, 0, X, X[:,[i]]) for
4
5 def kernel_lin_reg(X, Y, lam, k):
6     d, n = X.shape
7     kernel_matrix = gram(X,k) + lam*np.eye(n) #nXn
8     return np.dot(np.linalg.inv(kernel_matrix), Y.T)
9 def predict_regress(X, alpha, x, k):
10    return float(np.dot(np.apply_along_axis(k, 0, X, x), alpha.T)
11
12
```

Ask for Help