Course

<u>Progress</u>

<u>Dates</u>

Discussion

Resources



Project due Oct 21, 2020 05:29 IST Completed

As you can see, implementing a direct mapping to the high-dimensional features is a lot of work (imagine using an even higher dimensional feature mapping.) This is where the kernel trick becomes useful.

Recall the kernel perceptron algorithm we learned in the lecture. The weights heta can be represented by a linear combination of features:

$$heta = \sum_{i=1}^n lpha^{(i)} y^{(i)} \phi\left(x^{(i)}
ight)$$

In the softmax regression fomulation, we can also apply this representation of the weights:

$$heta_{j} = \sum_{i=1}^{n} lpha_{j}^{(i)} y^{(i)} \phi\left(x^{(i)}
ight).$$

$$h\left(x
ight) = rac{1}{\sum_{j=1}^{k}e^{\left[heta_{j}\cdot\phi\left(x
ight)/ au
ight]-c}}egin{bmatrix} e^{\left[heta_{j}\cdot\phi\left(x
ight)/ au
ight]-c}\ dots\ e^{\left[heta_{k}\cdot\phi\left(x
ight)/ au
ight]-c}\ dots\ e^{\left[heta_{k}\cdot\phi\left(x
ight)/ au
ight]-c} \end{bmatrix}$$

$$h\left(x
ight) = rac{1}{\sum_{j=1}^{k} e^{\left[\sum_{i=1}^{n} lpha_{j}^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/ au
ight] - c}} egin{bmatrix} e^{\left[\sum_{i=1}^{n} lpha_{1}^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/ au
ight] - c} \ & arepsilon^{\left[\sum_{i=1}^{n} lpha_{2}^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/ au
ight] - c} \ & dots \ & e^{\left[\sum_{i=1}^{n} lpha_{k}^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/ au
ight] - c} \end{bmatrix}$$

We actually do not need the real mapping $\phi(x)$, but the inner product between two features after mapping: $\phi(x_i) \cdot \phi(x)$, where x_i is a point in the training set and x is the new data point for which we want to compute the probability. If we can create a kernel function $K(x,y) = \phi(x) \cdot \phi(y)$, for any two points x and y, we can then kernelize our softmax regression algorithm.

You will be working in the files part1/main.py and part1/kernel.py in this problem

Implementing Polynomial Kernel

1.0/1 point (graded)

In the last section, we explicitly created a cubic feature mapping. Now, suppose we want to map the features into d dimensional polynomial space,

$$\phi\left(x\right) = \langle x_{d}^{2}, \dots, x_{1}^{2}, \sqrt{2}x_{d}x_{d-1}, \dots, \sqrt{2}x_{d}x_{1}, \sqrt{2}x_{d-1}x_{d-2}, \dots, \sqrt{2}x_{d-1}x_{1}, \dots, \sqrt{2}x_{2}x_{1}, \sqrt{2c}x_{d}, \dots, \sqrt{2c}x_{1}, c \rangle$$

Write a function $[polynomial_kernel]$ that takes in two matrix X and Y and computes the polynomial kernel K(x,y) for every pair of rows x in X and y in Y.

Available Functions: You have access to the NumPy python library as np

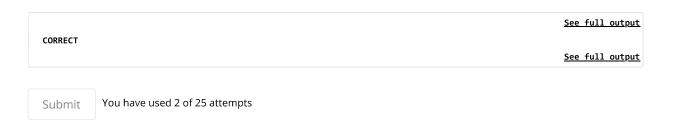
1 def polynomial_kernel(X, Y, c, p):
2 """

```
Compute the polynomial kernel between two matrices X and Y::
 4
               K(x, y) = (\langle x, y \rangle + c)^p
           for each pair of rows x in X and y in Y.
 6
7
 8
               X - (n, d) NumPy array (n datapoints each with d features)
9
               Y - (m, d) NumPy array (m datapoints each with d features)
10
               c - a coefficient to trade off high-order and low-order terms (scalar)
11
               p - the degree of the polynomial kernel
12
13
           Returns:
14
               kernel_matrix - (n, m) Numpy array containing the kernel matrix
15
       # VOUR CODE HERE
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results



Gaussian RBF Kernel

1.0/1 point (graded)

Another commonly used kernel is the Gaussian RBF kenel. Similarly, write a function $[rbf_kernel]$ that takes in two matrices X and Y and computes the RBF kernel K(x,y) for every pair of rows x in X and y in Y.

Available Functions: You have access to the NumPy python library as np

```
1 def rbf_kernel(X, Y, gamma):
3
           Compute the Gaussian RBF kernel between two matrices X and Y::
4
               K(x, y) = \exp(-\text{gamma } ||x-y||^2)
 5
           for each pair of rows x in X and y in Y.
 6
 7
           Args:
8
               X - (n, d) NumPy array (n datapoints each with d features)
9
               Y - (m, d) NumPy array (m datapoints each with d features)
10
               gamma - the gamma parameter of gaussian function (scalar)
11
12
           Returns:
13
               kernel_matrix - (n, m) Numpy array containing the kernel matrix
14
15
       # YOUR CODE HERE
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

```
CORRECT

See full output

Submit You have used 1 of 25 attempts
```

Now try implementing the softmax regression using kernelized features. You will have to rewrite the

softmax_regression function in softmax.py, as well as the auxiliary functions compute_cost_function,

How does the test error change?

In this project, you have been familiarized with the MNIST dataset for digit recognition, a popular task in computer vision.

You have implemented a linear regression which turned out to be inadequate for this task. You have also learned how to use scikit-learn's SVM for binary classification and multiclass classification.

Then, you have implemented your own softmax regression using gradient descent.

Finally, you experimented with different hyperparameters, different labels and different features, including kernelized features.

In the next project, you will apply neural networks to this task.

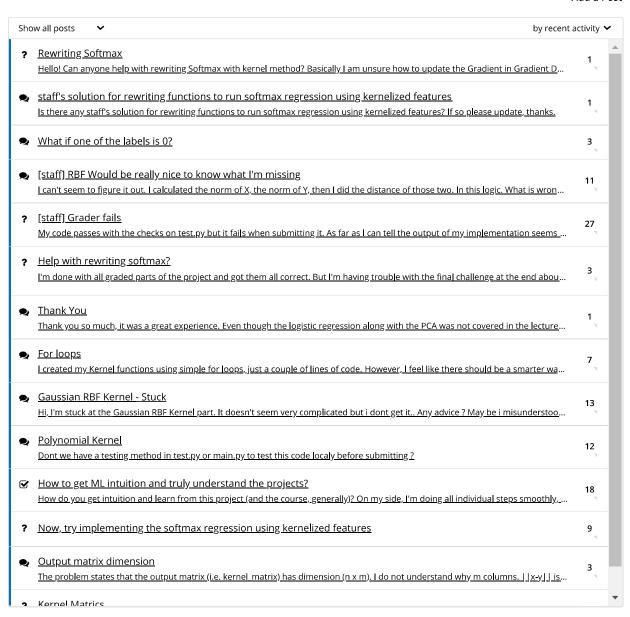
compute_probabilities, run_gradient_descent_iteration.

Discussion

Hide Discussion

Topic: Unit 2 Nonlinear Classification, Linear regression, Collaborative Filtering (2 weeks):Project 2: Digit recognition (Part 1) / 10. Kernel Methods

Add a Post



Previous

Next >

© All Rights Reserved



edX

About

Affiliates

edX for Business

Open edX

Careers

<u>News</u>

Legal

Terms of Service & Honor Code

Privacy Policy

Accessibility Policy

Trademark Policy

<u>Sitemap</u>

Connect

<u>Blog</u>

Contact Us

Help Center

Media Kit

Donate















© 2020 edX Inc. All rights reserved.

深圳市恒宇博科技有限公司 <u>粤ICP备17044299号-2</u>