10. Kernel Methods

As you can see, implementing a direct mapping to the high-dimensional features is a lot of work (imagine using a 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 θ can be represented by a linear combination of features:

$$\theta = \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \phi\left(x^{(i)}\right)$$

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

$$\theta_j = \sum_{i=1}^{n} \alpha_j^{(i)} y^{(i)} \phi\left(x^{(i)}\right)$$

$$h\left(x\right) = \frac{1}{\sum_{j=1}^{k} e^{[\theta_j \cdot \phi(x)/\tau] - c}} \begin{bmatrix} e^{[\theta_1 \cdot \phi(x)/\tau] - c} \\ e^{[\theta_2 \cdot \phi(x)/\tau] - c} \\ \vdots \\ e^{[\theta_k \cdot \phi(x)/\tau] - c} \end{bmatrix}$$

,

$$h(x) = \frac{1}{\sum_{j=1}^{k} e^{[\sum_{i=1}^{n} \alpha_j^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/\tau] - c}} \begin{bmatrix} e^{[\sum_{i=1}^{n} \alpha_1^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/\tau] - c} \\ e^{[\sum_{i=1}^{n} \alpha_2^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/\tau] - c} \\ \vdots \\ e^{[\sum_{i=1}^{n} \alpha_k^{(i)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)/\tau] - c} \end{bmatrix}$$

, 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

1/1 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,

$$\phi(x) = \langle x_d^2, \ldots, x_1^2, \sqrt{2}x_d x_{d-1}, \ldots, \sqrt{2}x_d x_1, \sqrt{2}x_{d-1} x_{d-2}, \ldots, \sqrt{2}x_{d-1} x_1, \ldots, \sqrt{2}x_2 x_1, \sqrt{2c}x_d, \ldots, \sqrt{2c}x_1, c \rangle$$

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

```
def polynomial_kernel(X, Y, c, p):
    """
        Compute the polynomial kernel between two matrices X and Y::
            K(x, y) = (<x, y> + c)^p
        for each pair of rows x in X and y in Y.

        Args:
            X - (n, d) NumPy array (n datapoints each with d features)
            Y - (m, d) NumPy array (m datapoints each with d features)
            c - an coefficient to trade off high-order and low-order terms (scalar)
            p - the degree of the polynomial kernel

        Returns:
            kernel_matrix - (n, m) Numpy array containing the kernel matrix
    """
    # YOUR CODE HERE
```

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

Correct

Test results

Hide output

CORRECT

Test: output

Output:

```
X: [[0.50850525]
 [0.80810771]
 [0.39895306]
 [0.04672001]
 [0.22629073]
 [0.27818553]
 [0.33150097]
 [0.50979713]
 [0.3663641 ]
 [0.30753055]
 [0.86995534]
 [0.09893425]
 [0.34155625]
 [0.27805316]
 [0.46428983]
 [0.39758288]]
Y: [[0.02639479]
 [0.76975143]
 [0.21656199]
 [0.27850756]
 [0.37931437]
 [0.68877365]
 [0.7229831 ]
 [0.89907204]
 [0.7222399 ]
 [0.48238328]
 [0.58872238]
 [0.04278369]
 [0.62694135]
 [0.50205871]
 [0.54204886]
 [0.29242258]]
c: 4
d: 4
Submission output: [[259.45333595 371.8955203  285.3771768  294.22669197 309.06592057
  358.14168432 363.90466045 394.67698636 363.77872608 324.81139948
  341.67928418 261.61507036 347.89930791 327.88422711 334.19686215
  296.24256778]
 [261.50426828 456.38942714 303.82826452 318.66415282 343.96055771
  431.08695322 441.64416716 499.08663536 441.41277279 371.35213017
  401.29496084 264.9663172  412.48672939 376.76156415 387.93811054
  322.07016761]
 [258.70641569 344.14200074 278.8448848  285.65182257 296.99453556
  333.93233262 338.21733494 360.92997355 338.12380769 308.93766316
  321.63358968 260.3976317  326.29080858 311.25788117 316.01395597
  287.19787101]
 [256.3158362  265.3313811  258.59999558 259.3473295  260.566961
  264.33790001 264.75725989 266.92376307 264.74814402 261.8184033
  263.11427075 256.5120905  263.5811871  262.05781    262.54490792
  259.51542799]
```

```
[257.53248945 303.59027734 268.77796955 272.51939316 278.6913155
 298.2942809  300.52307451 312.19376959 300.47452182 285.10952506
 291.84691305 258.48749038 294.29730165 286.34722587 288.87525633
 273.36518525]
[257.88489991 315.37934417 271.77450283 276.41778576 284.10088826
 308.68972339 311.50266709 326.28732664 311.44135258 292.1208729
 300.57206605 259.06048759 303.65382077 293.67094962 296.8404698
 277.46895198]
[258.24733423 327.84534997 274.87908963 280.46630355 289.74017681
 319.65031863 323.09337454 341.25822479 323.01828015 299.45802651
 309.73833657 259.65016228 313.49703015 301.3405586  305.19412808
 281.73300283]
[259.46215356 372.23249407 285.45488617 294.32894766 309.21042998
 358.43479611 364.21604032 395.08854144 364.08970502 325.00216227
 341.92112084 261.62945228 348.16035666 328.08417948 334.41585466
 296.35048437]
[258.48453761 336.19435727 276.92352115 283.13758198 293.47284715
 326.97362139 330.84551237 351.32246587 330.76103221 304.32998118
 315.8445072  260.03629632 320.06166657 306.4363282  310.75113722
 284.54772946]
[258.08433785 322.19582101 273.48000601 278.64064144 287.19448583
 314.68697719 317.84292867 334.46501862 317.77411465 296.14237561
 305.59165624 259.38492227 309.04241214 297.87397766 301.41683482
 279.80986573]
[261.92915655 475.48529179 307.74591577 323.89262627 351.51688823
 447.43510641 459.12912321 522.9903726  458.87266205 381.55070735
 414.5136192  265.66211262 426.86704405 387.49592318 399.79337217
 327.60541453]
[256.66916008 276.0594804  261.529128   263.12702795 265.74300193
 273.89554671 274.80815296 279.54181578 274.78830245 268.43779611
 271.23951918 257.08531093 272.25181466 268.95454197 270.00712216
 263.48697398]
[258.31573205 330.23723971 275.46758222 281.23480976 290.81308394
 321.74976056 325.3151244  344.13847972 325.2373537  300.85715574
 311.49032005 259.76148752 315.37995048 302.80371602 306.78917725
 282.54267801]
[257.88400053 315.348843   271.76682761 276.40778893 284.08699025
 308.66286619 311.47428356 326.25078119 311.4130024  292.10282565
 300.54956428 259.05902479 303.62967424 293.65209157 296.81994512
 277.45842576]
[259.15168349 360.49964143 282.72707581 290.74295912 304.15045426
 348.21765001 353.367302   380.7840135  353.25482296 318.33288939
 333.47931712 261.12319777 339.05289364 321.09564996 326.76631832
 292.56678822]
[258.69708409 343.80503719 278.76390147 285.5457769  296.84583324
 333.63754093 337.90493335 360.52210871 337.81179186 308.74288899
 321.38860091 260.382432   326.02709192 311.05402364 315.79134731
 287.08607609]]
```

Test: size

Output:

```
Output size: (15, 15)
```

Hide output

Submit You have used 2 of 20 attempts

✔ Correct (1/1 point)

Gaussian RBF Kernel

1/1 point (graded)

Another common used Knernel is the Gaussian RBF kenel. Similarly, write a function `rbf_kernel` that takes in two matrix 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

```
def rbf_kernel(X, Y, gamma):
    """
        Compute the Gaussian RBF kernel between two matrices X and Y::
            K(x, y) = exp(-gamma ||x-y||^2)
        for each pair of rows x in X and y in Y.

        Args:
            X - (n, d) NumPy array (n datapoints each with d features)
            Y - (m, d) NumPy array (m datapoints each with d features)
            gamma - the gamma parameter of gaussian function (scalar)

        Returns:
            kernel_matrix - (n, m) Numpy array containing the kernel matrix
    """
    # YOUR CODE HERE
    n, m = X.shape[0], Y.shape[0]
```

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

Correct

Test results

Hide output

CORRECT

Test: output

Output:

```
X: [[0.52009799 0.33276938 0.42125613 0.60833721]
 [0.63681678 0.31999943 0.05405923 0.83100665]
 [0.4102819  0.99294353 0.02802847 0.55893175]
 [0.86683841 0.13349928 0.38415034 0.90254971]
 [0.0539113  0.86359485 0.89690824 0.63431305]
 [0.04469732 0.08131647 0.84941345 0.9747802 ]
 [0.09435112 0.32814925 0.22581371 0.23698263]
 [0.94533382 0.91536782 0.56659626 0.60617055]
 [0.96055376 0.54993516 0.82225866 0.13439779]
 [0.05021793 0.14804559 0.59366726 0.52419078]
 [0.66446213 0.26918429 0.88382138 0.07526064]
 [0.52760462 0.59630909 0.15472622 0.55947617]]
Y: [[0.48210932 0.47426103 0.00981111 0.42829309]
 [0.66953065 0.40539054 0.90534885 0.10439846]
 [0.51716543 0.15275132 0.35379114 0.81364837]
 [0.04202855 0.96426873 0.84726547 0.90178465]
 [0.81890828 0.43859331 0.42855636 0.67330293]
 [0.05856177 0.58690659 0.04608773 0.32357872]
 [0.23149459 0.5507729  0.77693495 0.44885076]
 [0.24017568 0.79046077 0.68216106 0.16295543]
 [0.81297699 0.48832983 0.64407587 0.10332951]
 [0.75349644 0.80946848 0.85368719 0.60275183]
 [0.57006613 0.96085723 0.96705456 0.80693637]
 [0.39040672 0.3825822  0.04981811 0.88685114]]
gamma: 0.5914668003360491
Submission output: [[0.87634517 0.73702013 0.95428163 0.58900204 0.93991847 0.74423215
  0.84599348 0.72048204 0.78250358 0.75786734 0.6476978  0.87031636]
 [0.88222127 0.47433157 0.92466173 0.43611359 0.88197436 0.67554193
  0.59207654 0.48613374 0.57452877 0.57191882 0.47763137 0.96070839]
 [0.84158754 0.4398235  0.59130171 0.57857692 0.68171598 0.81575905
  0.62285307 0.67896396 0.55224399 0.61022405 0.5634784  0.75241958]
 [0.68925664 0.54650489 0.92518798 0.39162348 0.91515282 0.46122691
  0.57422291 0.4216375  0.61021389 0.63038933 0.51512875 0.78882555]
 [0.50225036 0.59781789 0.53832676 0.95137217 0.55787947 0.58827362
  0.89997394 0.83329187 0.53330933 0.74609513 0.83217985 0.5137822 ]
 [0.45019476 0.47570305 0.7440148  0.62859458 0.55513491 0.45665951
  0.72782184 0.48364783 0.39803652 0.50026248 0.52431941 0.60223947]
 [0.8600184  0.6171012  0.71870137 0.48153006 0.63467804 0.93809877
  0.7814242  0.76689246 0.64748382 0.49346574 0.41158567 0.7249099 ]
 [0.64141757 0.65993918 0.60370707 0.55860777 0.85398785 0.47883507
  0.65650735 0.65218937 0.76236889 0.92574512 0.81610718 0.57430975]
 [0.55974334 0.93513224 0.54211272 0.38704024 0.75378314 0.42336865
  0.68792133 0.70240322 0.96611182 0.8224042  0.62501081 0.40794645]
 [0.68363473 0.65199613 0.80850673 0.59663193 0.65135891 0.72970995
  0.87058597 0.70662715 0.59519264 0.55159862 0.50644206 0.70207499]
 [0.56548227 0.98830449 0.60080169 0.39861588 0.69404722 0.48267848
  0.78107533 0.74393936 0.92689674 0.71002659 0.54390877 0.42612736]
 [0.9678926  0.61310703 0.83687506 0.56410287 0.88965977 0.84360569
  0.7487816  0.71986343 0.72632002 0.70671913 0.60279114 0.89758213]]
```

Test: size

Output:

```
Output size: (12, 12)
```

Hide output

Submit You have used 1 of 20 attempts

✔ Correct (1/1 point)

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`, `compute_probabilities`, `run_gradient_descent_iteration`.

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.

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

Show all posts ▼ by recent activity ▼

?	[Staff] Please check graders for kernel functions For polynomial_kernel, my output seems to match what's expected exactly, but is marked wrong. For rbf_kernel, identical submissions were graded inconsistently -- the first r...	8
?	Implementing the softmax regression using kernelized features I am trying to implement the softmax regression using kernelized features. Please give me the explanation on alpha^{i} in above formulation. In the lecture, alpha^{i} means t...	7
?	Testing the kernel functions Is there any way to test these functions locally? main.py has no tests that run them.	3
☑	What are the other directories in the mnist project? `part2-mnist part2-nn part2-twodigit` - is this something the MIT students would have a chance to work with, or these are just unfinished folders and we will work with their ...	2

Learn About Verified Certificates