10. Kernel Methods

Extension Note: Project 2 due date has been extended by 2 days to **July 18 23:59UTC** (Note the UTC time zone).

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 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:

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

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

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

We actually do not need the real mapping $\phi\left(x\right)$, but the inner product between two features after mapping: $\phi\left(x_i\right) \cdot \phi\left(x\right)$, 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 $K\left(x, y\right) = \phi\left(x\right) \cdot \phi\left(y\right)$, 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,

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

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

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

```
        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 - a 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
    """
    return (X @ Y.T + c)**p

```

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

Correct

```
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
    """
    K = X @ Y.transpose()
    K += c
    K **= p
    return K
```

Test results

See full output

CORRECT

See full output

Submit You have used 1 of 20 attempts

Answers are displayed within the problem

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 $K\left(x, y\right)$ for every pair of rows x in X and y in Y.

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

```
        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:
```

```
13            kernel_matrix - (n, m) Numpy array containing the kernel matrix
14    """
15    distance = np.sum(X**2, 1).reshape(X.shape[0], 1) + np.sum(Y**2, 1) - 2 * X @ Y.T
16    return np.exp(-gamma * distance)
17
18
```

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

Correct

```
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
    """
    XTX = np.mat([np.dot(row, row) for row in X]).T
    YTY = np.mat([np.dot(row, row) for row in Y]).T
    XTX_matrix = np.repeat(XTX, Y.shape[0], axis=1)
    YTY_matrix = np.repeat(YTY, X.shape[0], axis=1).T
    K = np.asarray((XTX_matrix + YTY_matrix - 2 * (X @ Y.T)), dtype='float64')
    K *= - gamma
    return np.exp(K, K)
```

Test results

See full output

CORRECT

See full output

Submit You have used 2 of 20 attempts

Answers are displayed within the problem

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

Show Discussion

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