NYCU Introduction to Machine Learning, Homework 4

109550198 , 卜銳凱

Part. 1, Coding (50%):
For this coding assignment, you are required to implement some fundamental parts of the Support Vector Machine Classifier using only NumPy. After that, train your model and tune the hyperparameter on the provided dataset and evaluate the performance on the testing data.

(50%) Support Vector Machine

Requirements:

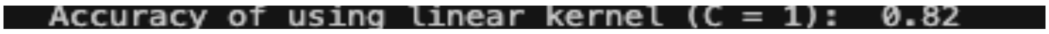
- Implement the *gram_matrix* function to compute the Gram matrix of the given data with an argument **kernel_function** to specify which kernel function to use.
- Implement the *linear_kernel* function to compute the value of the linear kernel between two vectors.
- Implement the *polynomial_kernel* function to compute the value of the polynomial kernel between two vectors with an argument **degree**.
- Implement the *rbf_kernel* function to compute the value of the rbf kernel between two vectors with an argument **gamma**.

Tips:

- Your functions will be used in the SVM classifier from scikit-learn like the code below.
  svc = SVC(kernel='precomputed')
  svc.fit(gram_matrix(X_train, X_train, your_kernel), y_train) y_pred =
  svc.predict(gram_matrix(X_test, X_train, your_kernel))
- For hyperparameter tuning, you can use any third party library's algorithm to automatically find the best hyperparameter, such as GridSearch. In your submission, just give the best hyperparameter you used and do not import any additional libraries/packages.

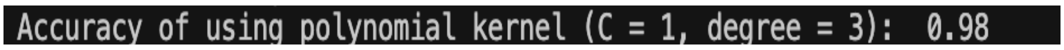
Criteria:

1. (10%) Show the accuracy score of the testing data using *linear_kernel*. Your accuracy score should be higher than 0.8.



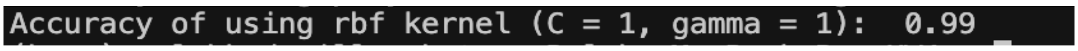
```
Accuracy of using linear kernel (C = 1):  0.82
```

2. (20%) Tune the hyperparameters of the *polynomial_kernel*. Show the accuracy score of the testing data using *polynomial_kernel* and the hyperparameters you used.



```
Accuracy of using polynomial kernel (C = 1, degree = 3):  0.98
```

3. (20%) Tune the hyperparameters of the *rbf_kernel*. Show the accuracy score of the testing data using *rbf_kernel* and the hyperparameters you used.



```
Accuracy of using rbf kernel (C = 1, gamma = 1):  0.99
```

(next page)

The following table is the grading criteria for question 2 and 3:

Points	Testing Accuracy
20 points	**0.98 <= acc**
15 points	**0.90 <= acc < 98**
10 points	**0.85 <= acc < 0.90**
5 points	**0.8 <= acc < 0.85**
0 points	**acc < 0.8**

Part. 2, Questions (50%):

1. (20%) Given a valid kernel $k_1(x, x')$, prove that the following proposed functions are or are not valid kernels. If one is not a valid kernel, give an example of $k(x, x')$ that the corresponding K is not positive semidefinite and shows its eigenvalues.
   a. $k(x, x') = k_1(x, x') + exp(x^T x')$
   b. $k(x, x') = k_1(x, x') - 1$
   c. $k(x, x') = exp(\|x - x'\|^2)$
   d. $k(x, x') = exp(k_1(x, x')) - k_1(x, x')$

a. $k(x,x') = k_1(x,x') + exp(x^T x')$
This proposed function is a valid kernel. To prove this, let's consider the sum of valid kernels: If $k_1(x,x')$ is a valid kernel, then for any set of points $\{x_1,x_2,\ldots,x_n\}$, the corresponding Gram matrix K_1 with entries $K_{1ij} = k_1(xi,xj)$ is positive semidefinite.

Additionally, $\exp(x^T x')$ represents the exponentiation of an inner product, and by Mercer's theorem, the exponentiation of a valid kernel remains a valid kernel.

Therefore, the sum of two valid kernels remains a valid kernel. The resulting Gram matrix K=K_1 + $\exp(x^T x)$ will also be positive semidefinite for any set of points.

b. $k(x,x') = k_1(x,x')^{-1}$

(next page)

This proposed function is not necessarily a valid kernel. Taking the inverse of a valid kernel does not guarantee a valid kernel. For instance, consider the kernel $\boldsymbol{k_1(x,x')=x^Tx' + c}$, where c is a constant term. This kernel is valid as it satisfies Mercer's conditions.

However, taking its inverse $k(x,x')=1/(x^Tx'+c)$ does not ensure positive semidefiniteness for all possible sets of points. Therefore, this function does not always yield a valid kernel.

c. $k(x,x') = \exp(\|x-x'\|^2)$
This proposed function is a valid kernel. The function represents the exponentiation of the squared Euclidean distance between x and x'. The squared distance is a valid kernel, and exponentiating it maintains the validity as per Mercer's theorem

d. $k(x, x') = \exp(k_1(x, x')) - k_1(x, x')$

This proposed function is not necessarily a valid kernel. The exponentiation of a valid kernel does not always guarantee positive semidefiniteness, especially when subtracting another valid kernel. The resulting function might fail the positive semidefinite property for certain sets of points.

For such a scenario, consider a valid kernel $k_1(x, x')=x^Tx'$. The resulting function $k(x, x') = \exp(x^Tx')-x^Tx'$ might not satisfy positive semidefiniteness for some choices of points.

(next page)

2. (15%) One way to construct kernels is to build them from simpler ones. Given three possible “construction rules”: assuming $K_1(x, x')$ and $K_2(x, x')$are kernels

then so are

   a. (scaling) $f(x)K_1(x, x')f(x'),\ f(x) \in R$

   b. (sum) $K_1(x, x') + K_2(x, x')$

   c. (product) $K_1(x, x')K_2(x, x')$

Use the construction rules to build a normalized cubic polynomial kernel:

$$K(x, x') = \left(1 + \left(\frac{x}{||x||}\right)^T \left(\frac{x'}{||x'||}\right)\right)^3$$

You can assume that you already have a constant kernel $K_0(x, x') = 1$ and a

linear kernel $K_1(x, x') = x^T x'$. Identify which rules you are employing at each

step.

to construct the normalized cubic polynomial kernel, let's go through the process step-by-step, employing the given construction rules:

Apply the Sum Rule: Since you have a constant kernel $K_0(\mathbf{x},\mathbf{x}')=1$ and a linear kernel $K_1(\mathbf{x},\mathbf{x}')=\mathbf{x}^T\mathbf{x}'$, use the sum rule to combine them: $K_2(\mathbf{x},\mathbf{x}')=K_0+K_1=1+\mathbf{x}^T\mathbf{x}'$

Apply the Product Rule: Now apply the product rule to K_2 with itself three times to construct a cubic polynomial kernel: $K_3(\mathbf{x},\mathbf{x}')=K_2\cdot K_2\cdot K_2=(1+\mathbf{x}^T\mathbf{x}')^3$

Normalize the Kernel: The goal is to normalize each vector $\mathbf{x}$ and $\mathbf{x}'$ by their respective norms. The normalization factor is a scaling function $f(\mathbf{x}) = \frac{1}{\|\mathbf{x}\|}$. Apply this function to each occurrence of $\mathbf{x}$ and $\mathbf{x}'$ in K_3 using the scaling rule:

$$K(\mathbf{x},\mathbf{x}') = \left(1 + \left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right)^T \left(\frac{\mathbf{x}'}{\|\mathbf{x}'\|}\right)\right)^3$$

By using these steps, you follow the construction rules to build the normalized cubic polynomial kernel as required. Each step uses a construction rule on the kernels obtained from the previous step until the final form is achieved.

3. (15%) A social media platform has posts with text and images spanning multiple topics like news, entertainment, tech, etc. They want to categorize posts into these topics using SVMs. Discuss two multi-class SVM formulations: \`One-versus-one\` and \`One-versus-the-rest\` for this task.

   a. The formulation of the method [how many classifiers are required]

      One-versus-One: Involves training a single classifier for each pair of classes. If there are N classes, you need N(N−1)/2 classifiers. For example, for 4 classes, you need 6 classifiers.

   b. Key trade offs involved (such as complexity and robustness).

      Key trade-offs for One-versus-One include increased training time due to the larger number of classifiers but potentially more accurate classification since each classifier specializes in distinguishing between two classes only. However, it can be computationally expensive for large N.

   c. If the platform has limited computing resources for the application in the inference phase and requires a faster method for the service, which method is better.

      For One-versus-the-rest, only N classifiers are needed, where each is trained to distinguish a single class from all remaining classes. The trade-off here is in the potential for less precise classification due to imbalance in class representation but with the benefit of reduced computational complexity.

      Given limited computing resources and a need for speed in the inference phase, One-versus-the-rest might be the better approach because it requires fewer classifiers, which can speed up the prediction process.