NYCU Introduction to Machine Learning, Homework 4

**110550126 曾家祐**

**Part. 1, Coding (50%)**:

For this coding assignment, you are required to implement some fundamental parts of the [Support Vector Machine Classifier](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) 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](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Polynomial_kernel) between two vectors with an argument **degree**.
* Implement the *rbf\_kernel* function to compute the value of the [rbf kernel](https://en.wikipedia.org/wiki/Radial_basis_function_kernel) between two vectors with an argument **gamma**.

**Tips:**

* Your functions will be used in the SVM classifier from [scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) 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](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html). 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.



1. (20%) Tune the hyperparameters of the *polynomial\_kernel*. Show the accuracy score of the testing data using *polynomial\_kernel* and the hyperparameters you used.



1. (20%) Tune the hyperparameters of the *rbf\_kernel*. Show the accuracy score of the testing data using *rbf\_kernel* and the hyperparameters you used.



**Part. 2, Questions (50%):**

1. (20%) Given a valid kernel , prove that the following proposed functions are or are not valid kernels. If one is not a valid kernel, give an example of that the corresponding is not positive semidefinite and shows its eigenvalues.

Suppose that ϕ(x) = x 🡪 it’s kernal function is 🡪 it is valid kernal function

and the exponential tern of a valid kernal will bbe valid kernal 🡪 is valid kernal.

since and are both valid kernal, by construction rules of sum 🡪 is valid kernal.

may not be a valid kernal.

consider x = [1 0 ]*T*, x’=[0 1]T

* K = A screenshot of a computer

  Description automatically generated and the eigenvalue = 1 and -1
* K is not positive semidefinite
* This kernel is not an valid kernel

may not be a valid kernal.

consider x = [1 0 ]*T*, x’=[3 0]T

* K = A screenshot of a computer

  Description automatically generated and the eigenvalue = 1+exp(4) and 1-exp(4)
* K is not positive semidefinite
* This kernel is not an valid kernel

By Taylor’s expansions 🡪

1 is valid kernal and 🡪, … are all valid kernal.

since the every term on the right side is valide kernal 🡪 are also valid kernal.

1. (15%) One way to construct kernels is to build them from simpler ones. Given

three possible “construction rules”: assuming and are kernels then so are

* 1. (scaling)
  2. (sum)
  3. (product)

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

You can assume that you already have a constant kernel = 1 and a linear kernel . Identify which rules you are employing at each step.

et and

1. scaling : f(x) = 1/() 🡪 from k2 and f(x)

-> we get 🡪

2.sum: we add k1 and the kernel we construct from scaling

->

3. product: power is a kind of power and from the kernal from sum rule we get we can product it self another two times

🡪)\*()\*)

🡪

We get the normalized cubic polynomial kernel by the construction rules.

1. (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.
   1. The formulation of the method [how many classifiers are required]

1. One-versus-one(OvO):

For N classes, we need to train N\*(N-1)/2 binaryclassifiers. Each classifier is for each pair of classes, (ex: for three class A,B,C we need to train Avs B, B vs C, A vs C. During prediction, each classifier votes for a class, and the class with the most votes is assigned.

2. One-versus-the-rest(OvR):

For N classes, we need to train N classifiers. one for each class against the rest of the classes, , (ex: for three class A,B,C we need to train Avs non-A, B vs non-B, C vs C). During prediction, the class with the highest confidence score from any of the N classifiers is assigned.

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

1. One-versus-one(OvO):

Num Number of Classifiers:

K\*(K-1)/2, which can be computationally expensive if there are many classes.

Training Time:

Training K\*(K-1)/2, but each classifier will be faster , since for each classifier the dataset is smaller.

Prediction Time:

Prediction involves comparing the outputs K\*(K-1)/2 of classifiers, which can be slower than OvR for large K.

2. One-versus-the-rest(OvR):

Num Number of Classifiers:

K, which can be more efficient when the number of classes is large.

Training Time for each classifier:

Training K classifiers, but on larger datasets compared to OvO.

Prediction Time:

Prediction involves comparing the outputs K\*(K-1)/2 of classifiers, which can be slower than OvR for large K.

* 1. 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.

I think OVR is better than OVO, since OvR involves evaluating only K classifiers during prediction, making it computationally more efficient compared to the K\*(K-1)/2 classifiers in OvO.