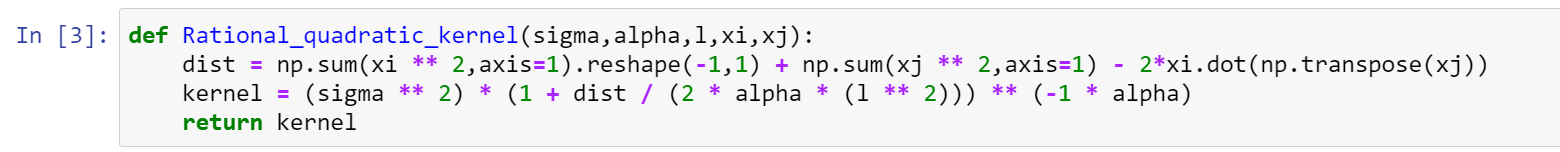
**ML HW05 Report**

1. Gaussian Process
2. Code

Part 1. Apply Gaussian Process Regression

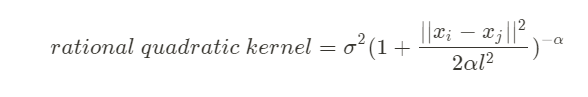


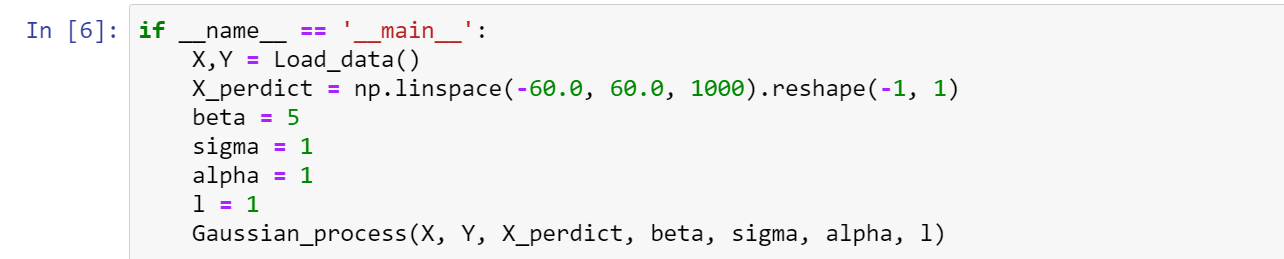
First, I load the data with this function and reshape NumPy array to n\*1 where the n is the number of x in input data.



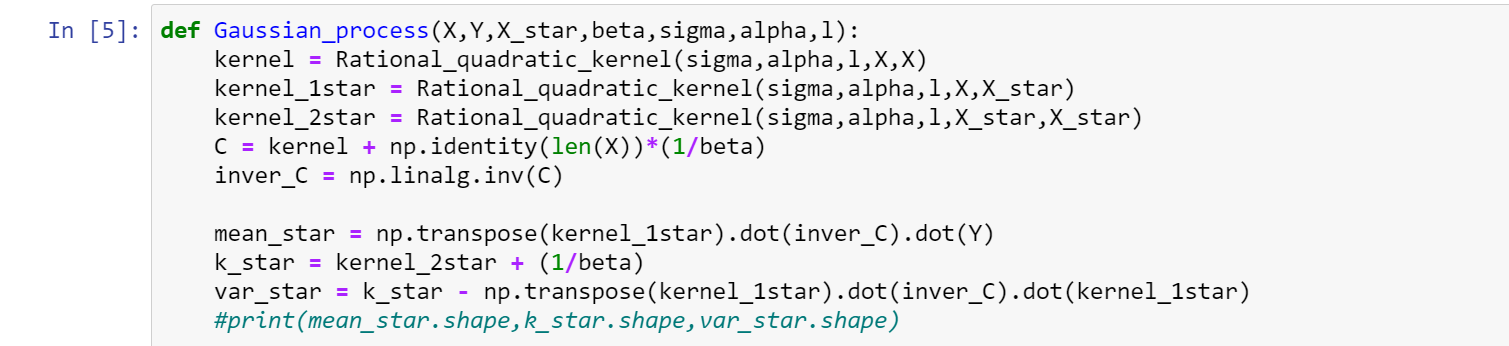
Use rational quadratic kernel as a kernel to compute the similarity between xi

and xj, where sigma, alpha, l are the parameter of kernel. We can search rational quadratic kernel format on Wiki.

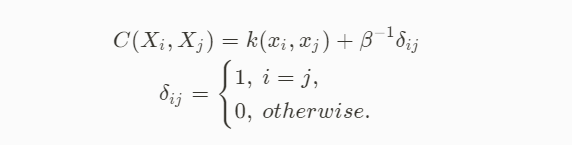




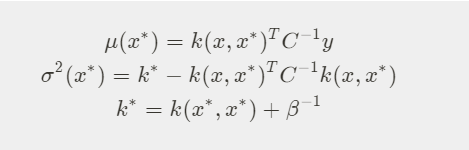
X and Y are training data, beta is the reciprocal of variance of the noise function (error function). X\_predict is the testing data. Because Spec said that we want to predict the distribution from X= -60 to X=+60, we space the number over the interval. I simply set the kernel parameter to 1

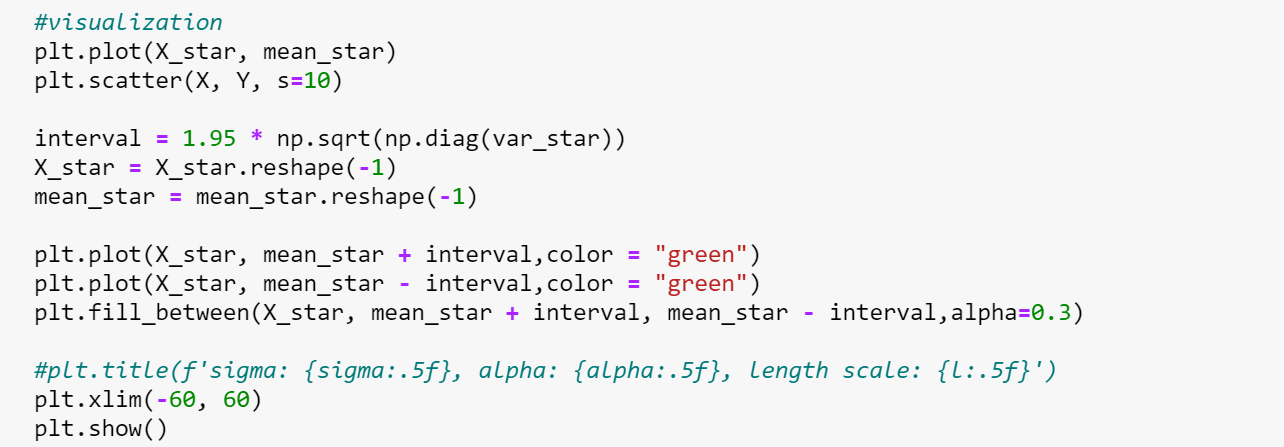


Then I can implement Gaussian Process. I solve the covariance matrix C first, it’s just like the format in the slide (picture below). k is kernel function, beta is the reciprocal of variance, and delta is Kronecker delta.



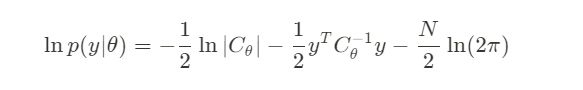
For the prediction, we can compute the mean and variance with the formula below. x is training data and x\* is testing data.



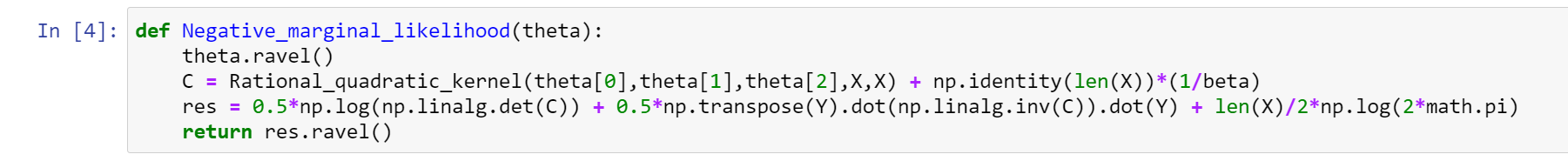


After computing the mean and variance, we can plot the result.

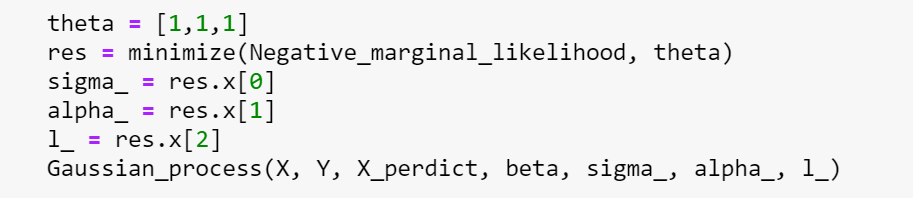
Part 2. Optimize the kernel parameters



We want to optimize the kernel parameters by minimizing negative marginal log-likelihood. Here is the format above.



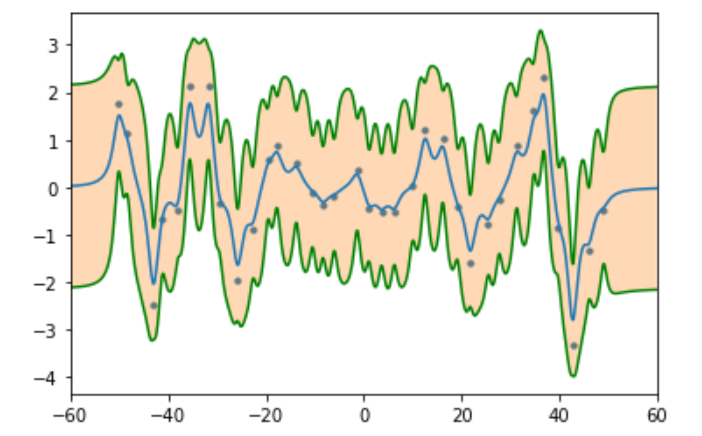
theta is kernel parameter



And we use scipy.optimize to minimize the marginal log-likelihood. We set all the kernel parameters to 1 for the initial guess.

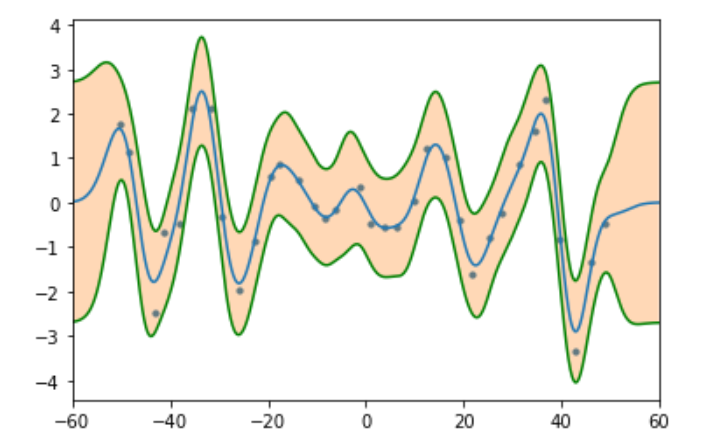
1. Result

Part 1. Apply Gaussian Process Regression.



Part 2. Optimize the kernel parameters

Result

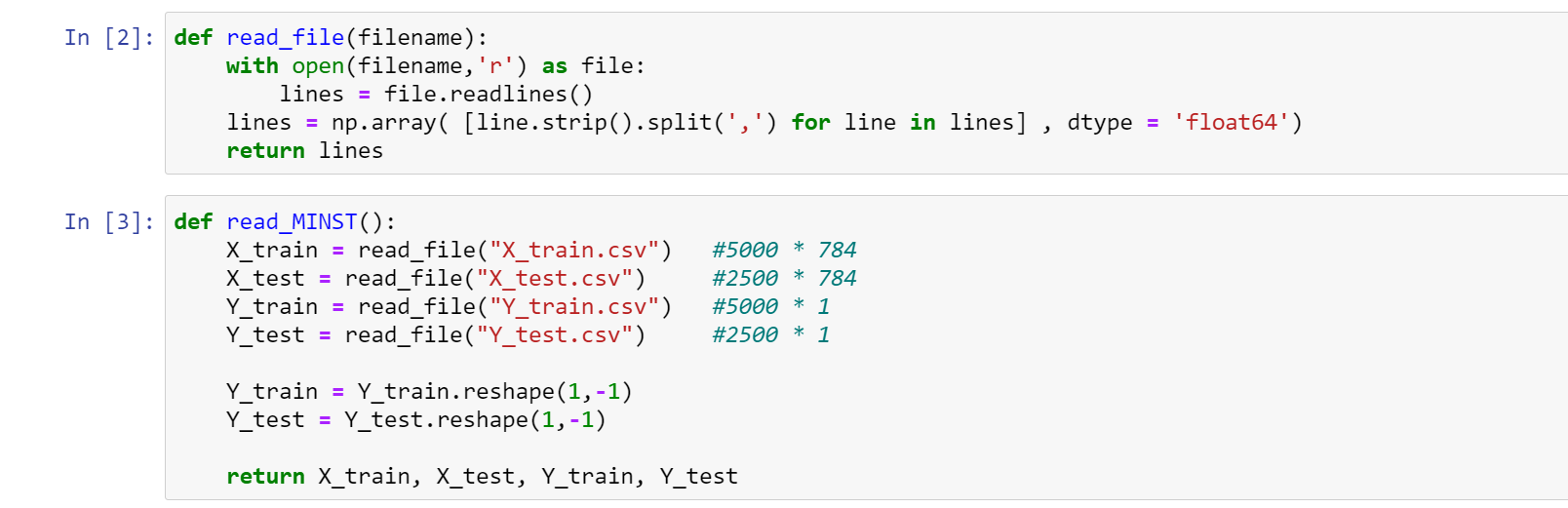


1. Discussion & observations
2. result 2 is obviously better than result 1
3. When l in kernel parameter is high 🡪 underfitting

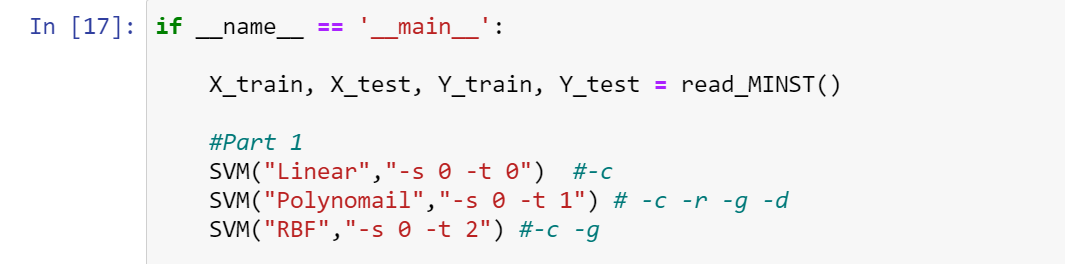
l in kernel parameter is low 🡪 overfitting

1. It may show bad result if we have bad parameters. The worst case of overfitting is multiple impulse function. It may fit the training data, bit it is not the general function we are trying to get.
2. SVM
3. Code

Part 1. Different kernel functions to SVM classifier

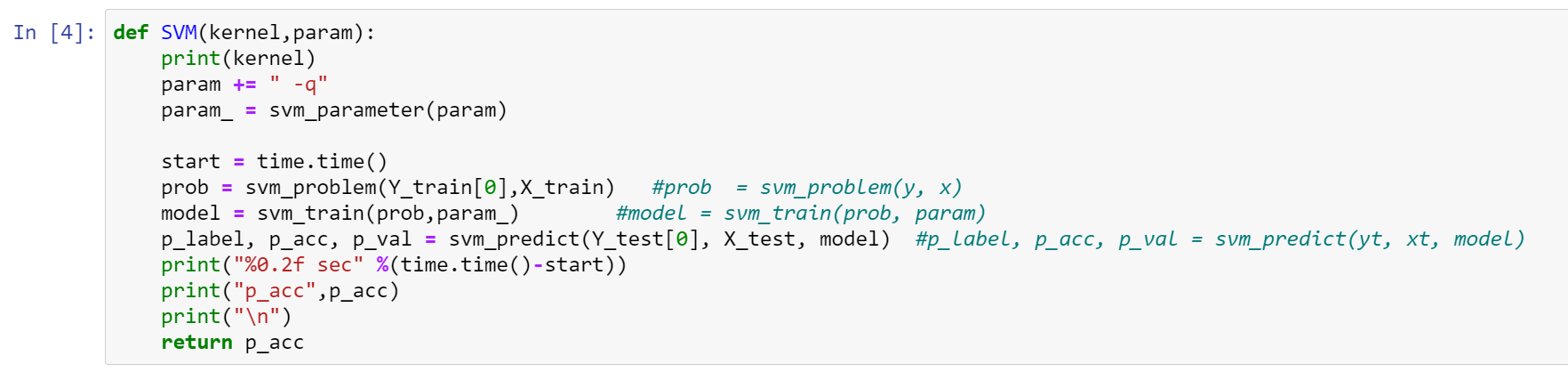


Use read\_MINST to read .csv and return X\_train, X\_test, Y\_train, Y\_test as NumPy array.



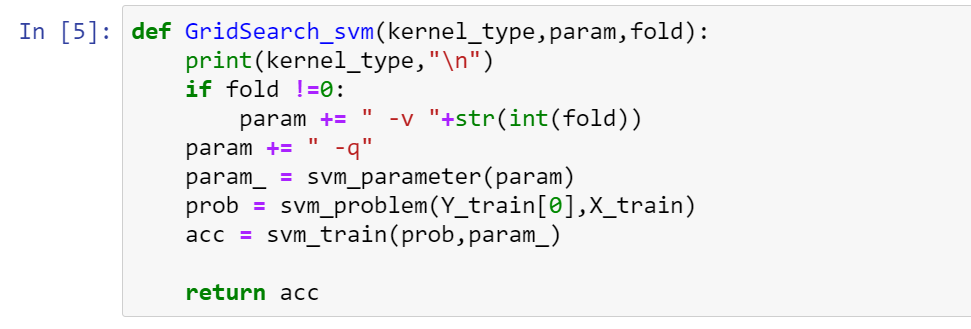
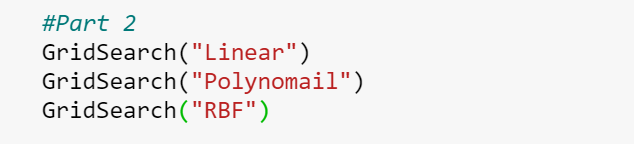
param record the parameter of SVM training. -s: SVM type and 0 is default as C-SVC.

-t: 0,1,2 is used to choose kernel function. 0,1,2 is correspond to Linear, Polynomial, RBF.



In this code, -q means not to print while training. We can use the function which is imported by libsvm.svmutil. svm\_predict return three term, which are predicted label, acc, val. In the accuracy include three value: accuracy, mean square error, and squared correlation coefficient.

Part 2. Grid search



Within GridSearch(), I defined some parameters those will use in different kernel\_type. After GridSearch() went through all parameter, we will get the best parameter store in max\_param.

Some new parameters:

-c: cost, set C in C\_SVC

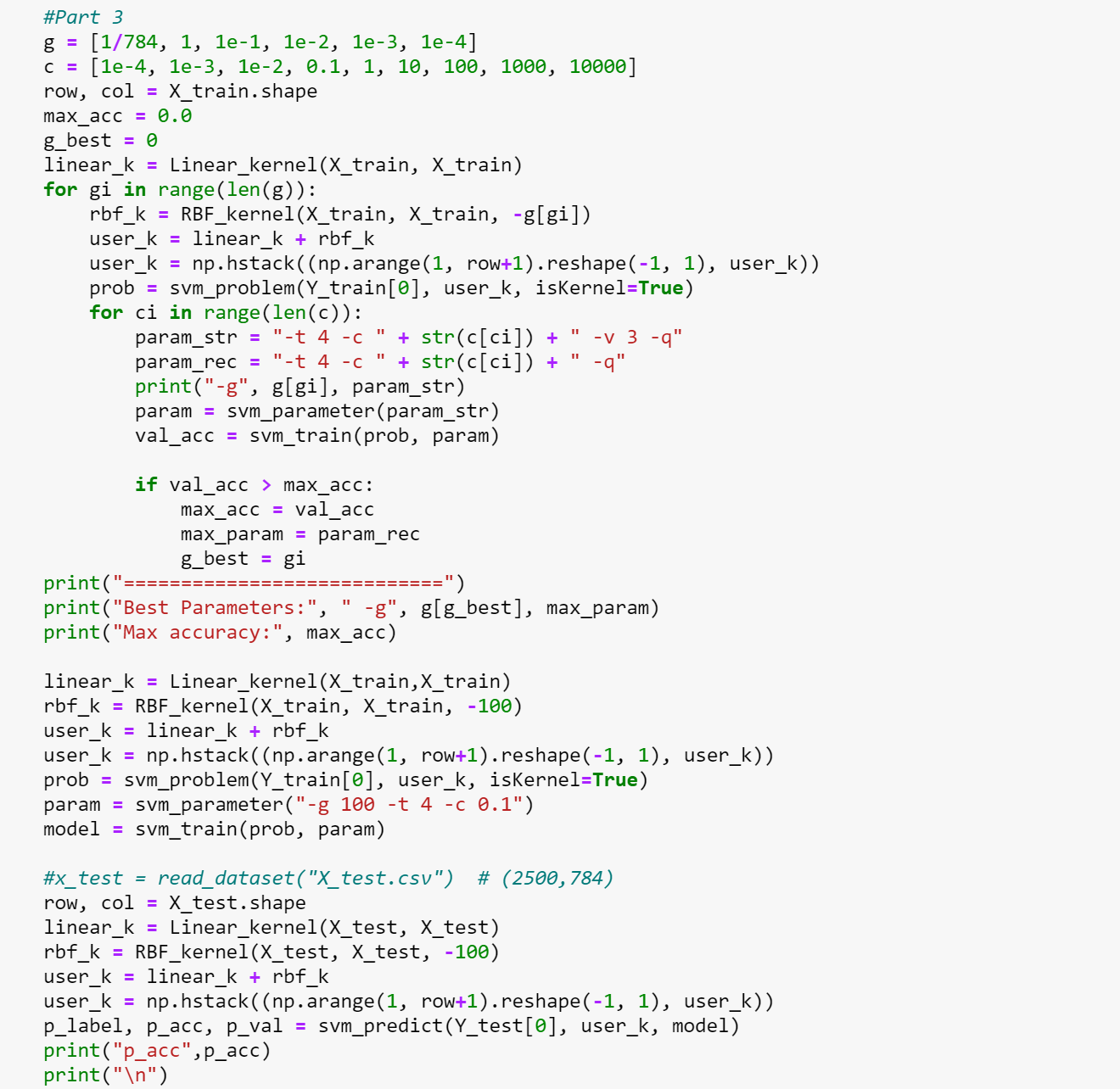
-d: degree, set degree in kernel function

-g: gamma, default is 1/num\_features (num\_features is 784 in this case)

-r: coef0

-v: n, n-fold cross validation, without -v means not using cross validation

Part 3. Linear kernel + RBF kernel



libsvm.svmutil supports user-defined kernel, we need to precomputed the data and set isKernel = True. We also need to do GridSearch to get the best parameter. Use the linear + RBF as a kernel and use best parameter and train it again.

1. Result

Part 1. Apply different kernel functions to SVM classifier

|  |  |  |
| --- | --- | --- |
| Kernel | Testing Acc(%) | Default parameters |
| Linear | 95.08 |  |
| Polynomial | 34.68 | -g 1/784 -r 0 -d 3 |
| RBF | 95.32 | -g 1/784 |

Part 2. Grid search

|  |  |  |  |
| --- | --- | --- | --- |
| Kernel | Cross validation Acc(%) | Testing Acc(%) | parameters |
| Linear | 96.86 | 95.96 | -c 0.01 |
| Polynomial | 98.24 | 97.72 | -c 0.1 -r 2 -g 1.0 -d 2 |
| RBF | 97.61 | 97.12 | -c 0.01 -g 10.0 |

Part 3. Linear kernel + RBF kernel

|  |  |  |  |
| --- | --- | --- | --- |
| Kernel | Cross validation Acc(%) | Testing Acc(%) | parameters |
| Linear + RBF | 97.06 | 34.2 | -g 0.01 -t 4 -c 0.01 |

1. Discussion & observations
2. The highest accuracy is SVM with RBF kernel and the lowest one is SVM with polynomial kernel function.
3. Testing result(%) may be worse than cross validation result, because the best parameter with training data may not be the best in testing data.
4. Linear kernel doesn’t need any parameter, so it takes minimum times to compute.
5. Polynomial kernel takes maximum times while doing compute since it need to fine-tuned lots of parameters.
6. RBF kernel is a well-used kernel for its great ability in classification. It take less time than linear, but RBF get better testing accuracy in result.
7. Parameter c is control the tolerance to the model, if c value is higher, it means that the less tolerance the model has a error. The result may too fit to the training data.