2022 Spring Machine Learning - Homework 5 (Gaussian process and SVM)

Ta-Yang Lin(林大洋) 310554017

1. Gaussian Process

Code

```
if __name__ == "__main__":
    # init
    beta = 5
    x, y = get_input()
    sample_x = np.linspace(-60, 60, 100).reshape(-1, 1)

    # task 1
    mean, co_var = gaussian_process(x, y, sample_x, beta)
    plot(x, y, sample_x, mean, co_var)

    # task 2
    best_param = optimize(x,y,beta)
    mean, co_var = gaussian_process(x, y, sample_x, beta, best_param)
    plot(x, y, sample_x, mean, co_var, "optimize")
    print("ok")
```

Above code is main function, it shows the processes of gaussian process, as you could see, the function divided three parts, init, task 1 and task 2.

- **Init part:**

```
# init
beta = 5
x, y = get_input()
sample_x = np.linspace(-60, 60, 100).reshape(-1, 1)
```

```
def get_input():
    f = open("./data/input.data")
    data = f.readlines()
    input_data = np.zeros((len(data), 2))
    for i in range(len(data)):
        tmp = data[i].split(' ')
        input_data[i, 0] = tmp[0]
        input_data[i, 1] = tmp[1]
    x = input_data[:, 0].reshape(-1, 1)
    y = input_data[:, 1].reshape(-1, 1)
    return x, y
```

According to spec, beta is 5 and get_input function can get the data points of file, so we got x,y. Because we need to show the result curve of gaussian process, so I sampled 100 data points in range(-60,60) as testing data using `np.linspace`.

- **Task 1:**

```
# task 1
mean, co_var = gaussian_process(x, y, sample_x, beta)
plot(x, y, sample_x, mean, co_var)
```

Basically, `gaussian_process` will return mean and covariance of `sample_x` and `plot` shows the result image, I can observe prediction situation through result image.

```
def rational_quadratic(x1, x2, lengthscale=1.0, var=1.0, a=1.0):
    dis = distance.cdist(x1, x2, 'sqeuclidean')
    return var*(1+(dis/2*a*(lengthscale**2)))**-a
```

$$k(x_a, x_b) = \sigma^2 \left(1 + \frac{\left\Vert x_a - x_b \right\Vert^2}{2\alpha\ell^2}\right)^{-\alpha}$$

With:

- σ^2 the overall variance (σ is also known as amplitude).
- ℓ the lengthscale.
- α the scale-mixture ($\alpha > 0$).

(formula information is cited https://peterroelants.github.io/posts/gaussian-process-kernels/)

`rational_quadratic` is kernel fuction that its formula contains euclidean distance, so I call `scipy.spatial.distance` to calculate the distance, and rational quadratic kernel has three hyperparameters **a**, **lenthscale** and **var**. Defaultly, I set `1.0` all, In task 2 ,I will optimize the hyperpatameter.

```
def gaussian_process(x, y, sample_x, beta, param=None):
    delta = np.identity(len(x))
    if(param is not None):
        C = rational_quadratic(x, x, param[0], param[1], param[2]) +
((beta**-1)*delta)
        mean = np.dot(rational_quadratic(x, sample_x, param[0], param[1],
param[2]).T,
                      np.dot(np.linalg.inv(C), y))
        k_star = rational_quadratic(sample_x, sample_x, param[0], param[1],
param[2])+(beta**-1)
        co_var = k_star - np.dot(rational_quadratic(x, sample_x, param[0],
param[1], param[2]).T,
                                np.dot(np.linalg.inv(C),
rational_quadratic(x, sample_x, param[0], param[1], param[2])))
    else:
        C = rational_quadratic(x, x) + ((beta**-1)*delta)
        mean = np.dot(rational_quadratic(x, sample_x).T,
                    np.dot(np.linalg.inv(C), y))
        k_star = rational_quadratic(sample_x, sample_x)+(beta**-1)
        co_var = k_star - np.dot(rational_quadratic(x, sample_x).T,
                                np.dot(np.linalg.inv(C),
rational_quadratic(x, sample_x)))

    return mean, co_var
```

`gaussian_process` represents Gaussian Process algorithm, it calculates C, mean, covariance using kernel `rational_quadratic`.

```
def plot(x, y, sample_x, mean, co_var,mode=None):
    x = x.reshape(-1)
    y = y.reshape(-1)
    sample_x = sample_x.reshape(-1)
    mean = mean.reshape(-1)
    std = co_var.diagonal()**0.5
    plt.figure()
    plt.scatter(x, y)
    plt.plot(sample_x, mean, c='r')
    up = mean+1.96*std
    low = mean-1.96*std
    plt.fill_between(sample_x, up, low, facecolor='purple', alpha=0.3)
    if(mode is not None):
        plt.savefig(f'./gaussian_process_{mode}.png')
    else:
        plt.savefig("./gaussian_process.png")
```

`plot` is a function that can show the prediction result and draw the line that represents 95% confidence. I used the formula $mean \pm 1.96 * std$

- **Task 2:**

```
# task 2
best_param = optimize(x,y,beta)
mean, co_var = gaussian_process(x, y, sample_x, beta, best_param)
plot(x, y, sample_x, mean, co_var, "optimize")
```

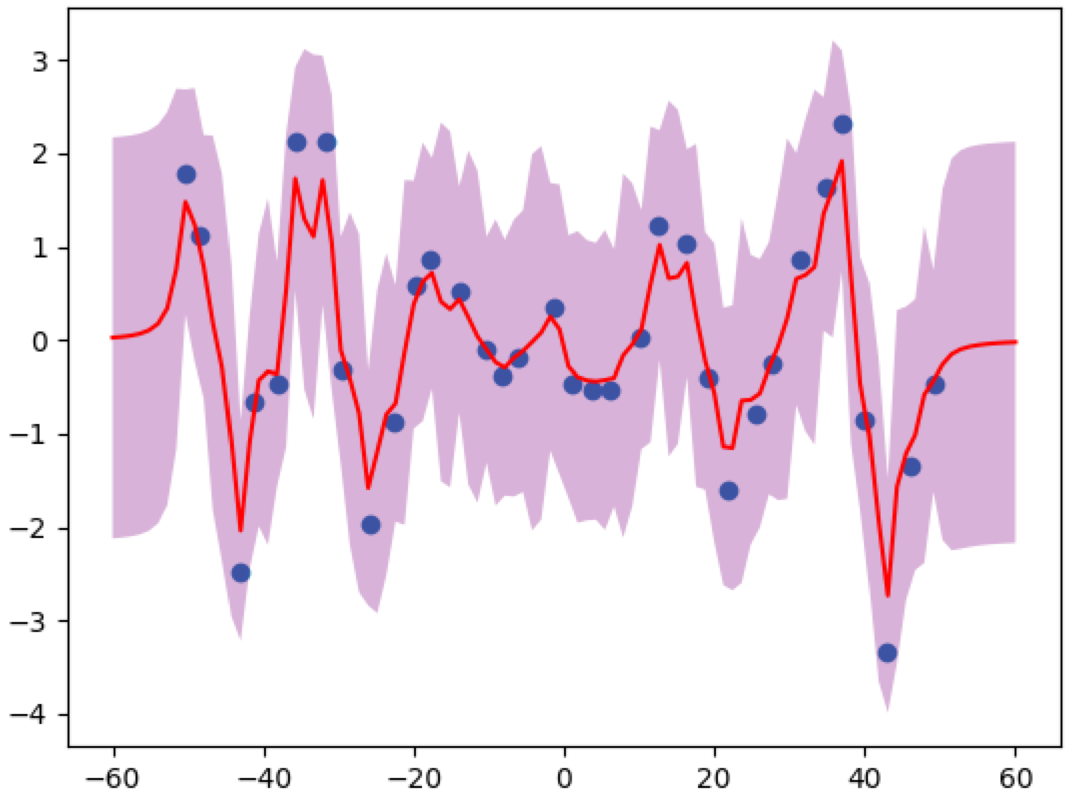
The different with Task 1 is `optimize`,after I optimized the hyperparameters of kernel using `optimize`, I call the same fuction for gaussian process.

```
def optimize(x,y,beta):
    def negative_log_likelihood(param):
        C = rational_quadratic(x, x,param[0],param[1],param[2]) +
((beta**-1)*np.identity(len(x)))
        cost = 0.5*np.log(np.linalg.det(C)) +
0.5*np.dot(y.T,np.dot(np.linalg.inv(C), y)) + len(x)/2 * np.log(np.pi*2)
        return cost[0]
    param = np.ones(3)
    ans = op.minimize(negative_log_likelihood,param)
    return ans.x
```

`optimize` will find the minimum negative marginal log-likelihood, so I defined a cost function `negative_log_likelihood`, the cost fuction will use kernel parameters. Finally I will get the best solution of kernel parameters from `scipy.optimize.minimize`. I can conduct Gaussian process using the kernel parameters and compare the different with result of Task 1.

Experiment

- **Task 1**



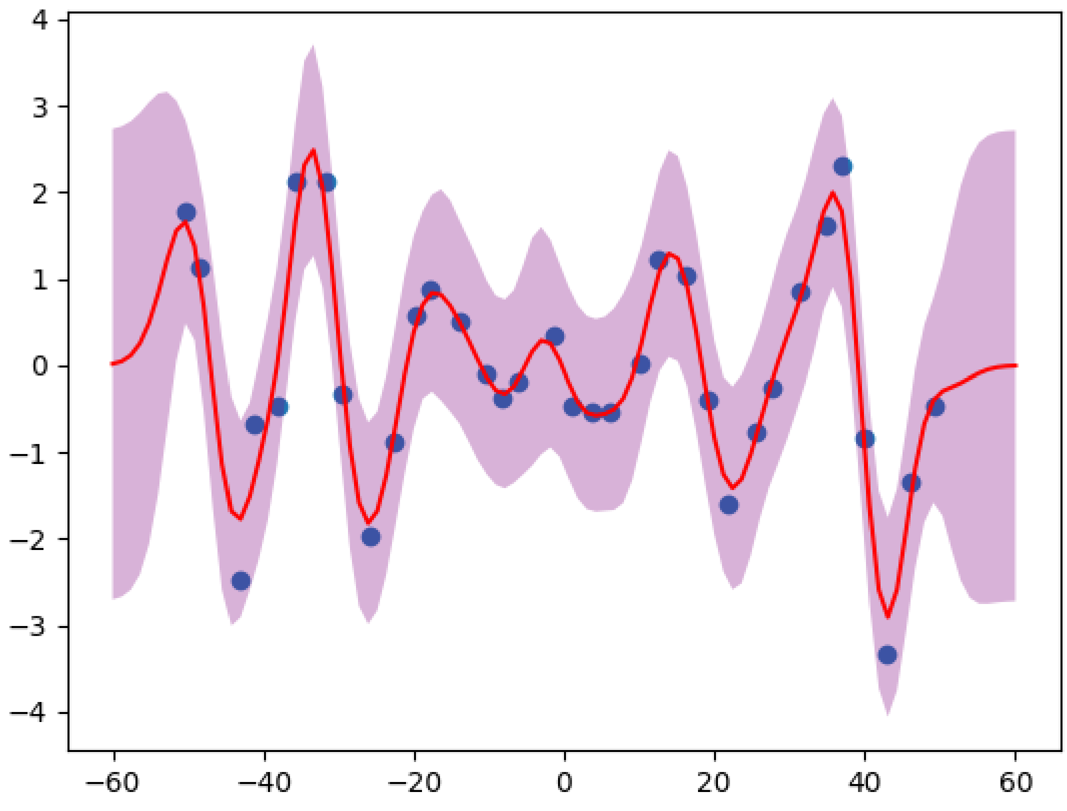
I sampled 100 data points in -60 to 60 as testing data, and training data is the data of file, I set 1.0 for all kernel parameters defaultly. The figure shows the prediction result of Gaussian process, the blue points are training data, red line is mean vector of prediction of testing data and the purple area is 95% confidence area for each mean using covariance. We can note the curve and area change along with blue points, Although we haven't the data that is between any two points, model can predict the value according to probability.

- **Task 2**

After I optimized the kernel parameters, I get the best solution like this:

```
lengthscale = 1.12180617e-03
var = 1.72506419e+00
a = 2.68586654e+02
```

I use the parameters, do Gaussian process again and plot the prediction result.



The figure is different from Task 1, curve is smoother and I find the means are closer to training data because I minimized the negative marginal likelihood. I think this model will be better than task 1, but I concern overfitting in this task, maybe we need to careful.

Discussion

Gaussian process is a kind of kernel-based method, when model predicts results, it will consider all training data, and calculate the probabilities, choosing the kernel is important in this algorithm, but it means we need to have experience with kernel or do some experiments before using the algorithm, it is a threshold for the user. Each kernel has its own parameter, user needs to set it, so optimization is important too, it means we need to spend more time, but it is still uncertain that the optimization result is more suitable for this dataset, maybe it is overfitting. Anyway Gaussian process is a classic algorithm, it is powerful in some cases, so It is worth to be considered, if you need to find an algorithm to predict.

2. SVM on MNIST

Code

```
if __name__ == '__main__':
    args = args_init()
    train_x,train_y,test_x,test_y = get_data()
    if(args.mode == 1 ):
        SVM_1(train_x,train_y,test_x,test_y)
    if(args.mode == 2 ):
        SVM_2(train_x,train_y,test_x,test_y)
    if(args.mode == 3 ):
        SVM_3(train_x,train_y,test_x,test_y)
```

This program is divided three parts, Task 1, Task 2, Task 3, I will introduce them later.

```
def get_data():
    train_x = np.genfromtxt('./data/X_train.csv', delimiter=',',
skip_header = 0)
    train_y = np.genfromtxt('./data/Y_train.csv', delimiter=',',
skip_header = 0)
    test_x = np.genfromtxt('./data/X_test.csv', delimiter=',', skip_header
= 0)
    test_y = np.genfromtxt('./data/Y_test.csv', delimiter=',', skip_header
= 0)
    return train_x,train_y,test_x,test_y
```

First, I need to get the `csv` data, so `get_data` will do that, it returns all data to `main` .

```
def metrics(pred,true):
    TP=0
    FP=0
    TN=0
    FN=0
    for i in range(len(true)):
        if(true[i] == 1):
            if(true[i] == pred[i]):
                TP+=1
            else:
                FN+=1
        else:
            if(true[i] == pred[i]):
                TN+=1
            else:
                FP+=1
    acc = (TP+TN)/len(true)
    recall = TP/(TP+FN)
    precision = TP/(FP+TP)

    return acc,recall,precision
```

Because `libsvm` only offers accuracy metric, I will call `metrics` that calculates `Accuracy` `Recall` `Precision`, and returns them. The way of calculating is that compares the results of prediction and ground truth.

- **Task 1**

```
def SVM_1(train_x,train_y,test_x,test_y):
    kernel = ['linear','poly','rbf']
    for i in range(len(kernel)):
        print("------------------------------")
        print(f'Kernel : {kernel[i]}')
        model = svmutil.svm_train(train_y,train_x,f'-q -t {i}')
        p_label, p_acc, p_val  = svmutil.svm_predict(test_y,test_x,model)
        acc,recall,precision = metrics(p_label,test_y)
        print(acc,recall,precision)
```

In Task 1, I try to use different kernel in SVM, it's easy as you could see.

- **Task 2**

```
def grid_linear(k,train_y,train_x,test_y,test_x):
    rid_search_index = ['cost']
    grid_search = [[1,5,10,20,50,100]]
    for c in range(len(grid_search[0])):
        print(f'kernel: linear , cost: {grid_search[0][c]}')
        model = svmutil.svm_train(train_y,train_x,f'-q -s 0 -t {k} -c
{grid_search[0][c]} -m 10000')
        p_label, p_acc, p_val  = svmutil.svm_predict(test_y,test_x,model)
        acc,recall,precision = metrics(p_label,test_y)
        print(acc,recall,precision)

def grid_poly(k,train_y,train_x,test_y,test_x):
    grid_search_index = ['cost','gamma','coef0','degree']
    grid_search = [[1,20,50],[0.001,0.01,0.1],[0,0.01,0.1],[1,3,10]]
    for c in range(len(grid_search[0])):
        for g in range(len(grid_search[1])):
            for coef in range(len(grid_search[2])):
                for degree in range(len(grid_search[3])):
                    print(f'kernel: poly , cost: {grid_search[0][c]} ,
gamma: {grid_search[1][g]} , coef0: {grid_search[2][coef]} , degree:
{grid_search[3][degree]}')
                    model = svmutil.svm_train(train_y,train_x,f'-q -s 0 -t
{k} -c {grid_search[0][c]} -g {grid_search[1][g]} -r {grid_search[2][coef]}
-m 10000 -d {grid_search[3][degree]}')
                    p_label, p_acc, p_val  =
svmutil.svm_predict(test_y,test_x,model)
                    acc,recall,precision = metrics(p_label,test_y)
                    print(acc,recall,precision)


def grid_rbf(k,train_y,train_x,test_y,test_x):
    grid_search_index = ['cost','gamma']
    grid_search = [[1,20,50],[0.001,0.01,0.1]]
    for c in range(len(grid_search[0])):
        for g in range(len(grid_search[1])):
            print(f'kernel: rbf , cost: {grid_search[0][c]} , gamma:
{grid_search[1][g]} ')
            model = svmutil.svm_train(train_y,train_x,f'-q -s 0 -t {k} -c
{grid_search[0][c]} -g {grid_search[1][g]} -m 10000 ')
            p_label, p_acc, p_val  =
svmutil.svm_predict(test_y,test_x,model)
            acc,recall,precision = metrics(p_label,test_y)
            print(acc,recall,precision)

def SVM_2(train_x,train_y,test_x,test_y):
    kernel = ['linear','poly','rbf']
    for k in range(len(kernel)):
        if k == 0:
            grid_linear(k,train_y,train_x,test_y,test_x)
        elif k==1:
            grid_poly(k,train_y,train_x,test_y,test_x)
        elif k==2:
            grid_rbf(k,train_y,train_x,test_y,test_x)
```

In Task 2, I do the grid search in SVM, there are different hyperparameters for each kernel, so I divided into multi-function. The hyperparametes are C, gamma, degree, coef0, I defines the range of parameters, the program will try to use each setting, finally, find the best setting.

- **Task 3**

```
def SVM_3(train_x,train_y,test_x,test_y):
    grid_search_index = ['cost','gamma']
    grid_search = [[1,20,50],[0.001,1/784,0.01,0.1]]
    for c in range(len(grid_search[0])):
        for g in range(len(grid_search[1])):
            print(f'cost: {grid_search[0][c]} , gamma: {grid_search[1]
[g]}')
            new_train_x,new_test_x =
linear_rbf_kernel(train_x,test_x,grid_search[1][g])
            model = svmutil.svm_train(train_y,new_train_x,f'-q -s 0 -t 4 -c
{grid_search[0][c]}  -m 10000')
            p_label, p_acc, p_val  =
svmutil.svm_predict(test_y,new_test_x,model)
            acc,recall,precision = metrics(p_label,test_y)
```

```
               print(acc,recall,precision)

def linear_rbf_kernel(train_x,test_x,gamma):
    new_train_x = np.zeros((5000,5001))
    new_test_x = np.zeros((2500,5001))
    # training
    linear_train_x = np.dot(train_x,train_x.T)
    rbf_train_x = np.exp((-gamma**2) * distance.cdist(train_x,train_x,
'sqeuclidean'))
    new_train_x[:,1:] = linear_train_x+rbf_train_x
    new_train_x[:,:1] = np.arange(5000)[:,np.newaxis]+1
    # testing
    linear_test_x = np.dot(test_x,train_x.T)
    rbf_test_x = np.exp((-gamma**2) * distance.cdist(test_x,train_x,
'sqeuclidean'))
    new_test_x[:,1:] = linear_test_x + rbf_test_x
    new_test_x[:,:1] = np.arange(2500)[:,np.newaxis]+1

    return new_train_x,new_test_x
```

In Task 3, I design a kernel that it is a mix of linear and rbf, so I use precompute option in `libsvm`, which means I need to precompute my training data and testing data. `linear_rbf_kernel` will calculate them according to kernel formula and add them. I also do grid search in this task, hyperparameters are C and gamma.

- linear kernel :
$$k(x,y) = x^T y$$

- rbf kernel :
$$k(x,y) = exp(-\frac{||x-y||^2}{2\sigma^2})$$

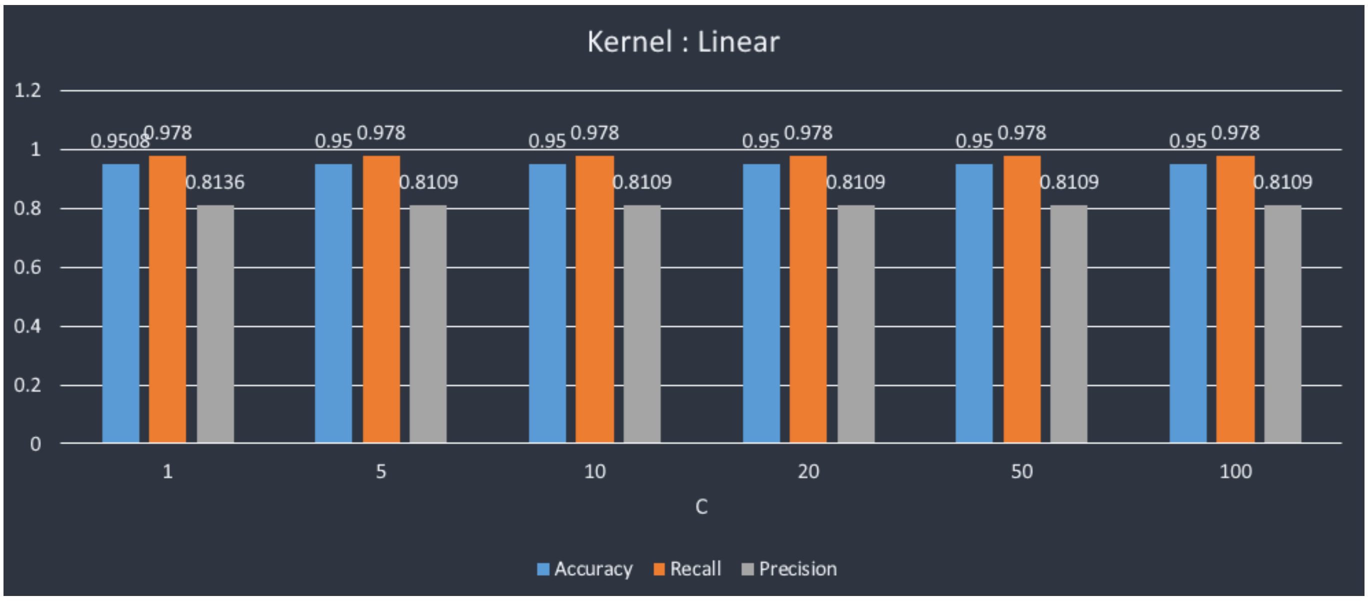
Experiment

- **Task 1**

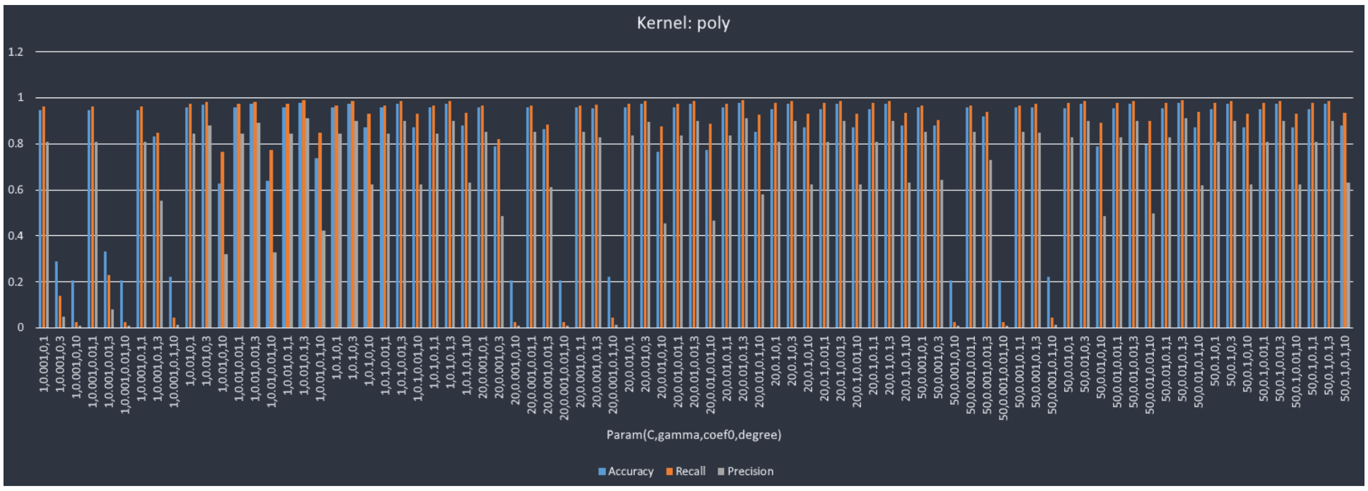
kernel	Accuracy	Recall	Precision
linear	95.08%	97.80%	81.36%
poly	34.68%	42.80%	13.70%
rbf	95.32%	97%	82.62%

We can see the result, rbf is best, linear is No.2, poly's accuracy isn't good, that shows kernel is important in SVM. All the parameters are deafult in this part, so maybe poly is good when we set different parameters, I surprised for poly's accuracy, because poly is more complicated than linear, but the result isn't better than linear.

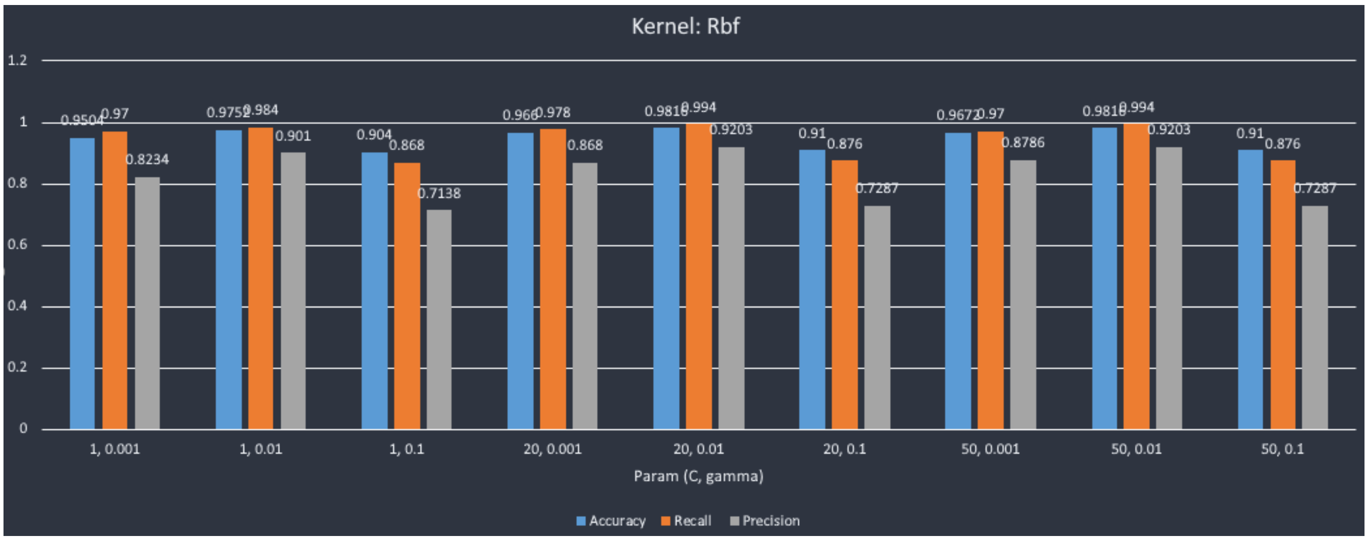
- **Task 2**



In the above figure, we can see the result about linear, there are one parameter C, I try to do grid serarch in range[1,5,10,20,50,100], I find the results are identical, which means C isn't a sensitive parameter. I got good model, Accuracy is 95%, Recall is 97.8%, Precision is 81.09%.

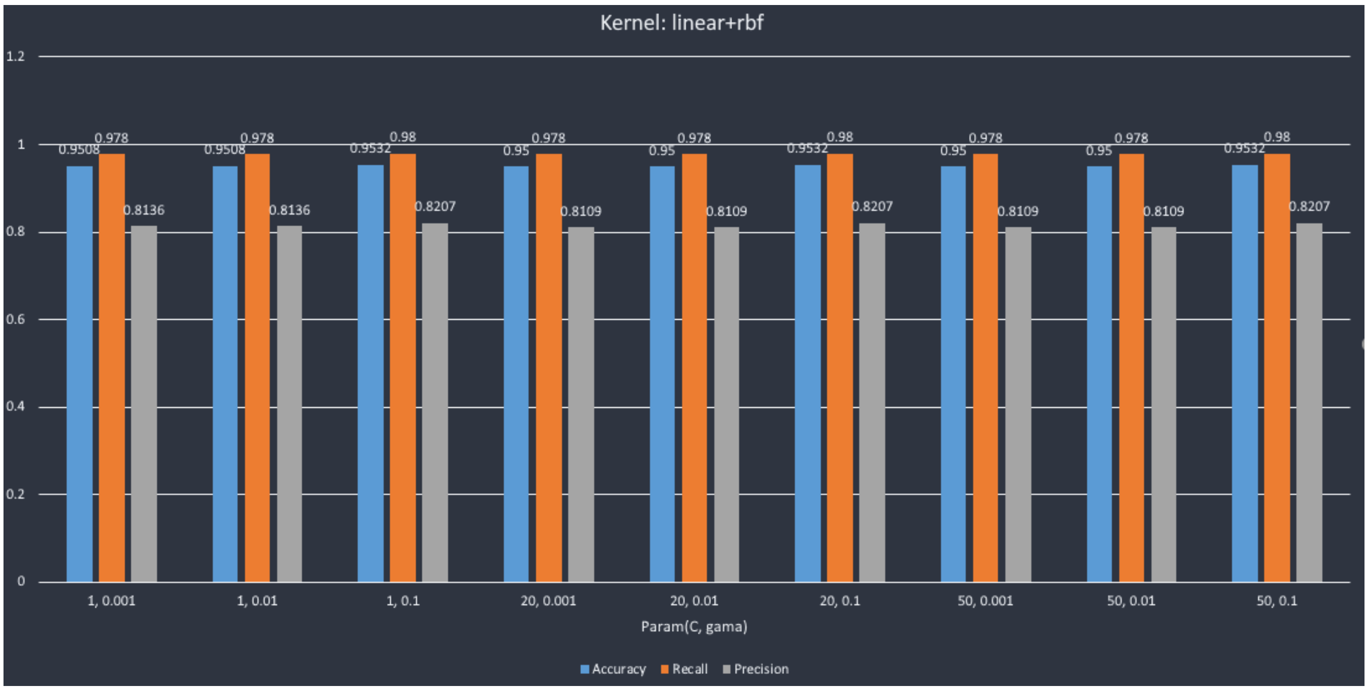


Above figure shows the results about poly kernel, I have four parameters, C, Gamma, coef0, degree, In Task 1's result, poly kernel isn't good, but I can get 97% accuracy over here, which means some parameters are important for poly kernel. I think degree is so important, the default value is 3 in `libsvm`, I discover that 1 is suitable for this data, if we set 3 or 10, in some setting, the results are not good. Finally, I got 97% using poly kernel, it's better than linear, degree is so sensitive for this data, we need to be careful.



The figure shows the results about rbf kernel, Basically, rbf is good kernel for this data, I adjusted two parameters, C and gamma, I can get the good result in mostly setting. Gamma is important for rbf, the result is best using gamma 0.01 than others, it has 98% accuracy and it is best in all the task.

- **Task 3**



The figure shows result of liner+rbf kernel, we can see the scores are good, I try to adjust the parameters, C and gamma, but I find the result won't change, so maybe the parameters aren't important if I use the kernel. I think rbf is better than this kernel, so we don't need to combine the two kernel. Finally I got the 98% recall and 95% accuracy in linear+rbf kernel.

Discussion

SVM is a powerful model, but choosing a kernel is so important, for this data, maybe rbf is suitable because it got the best accuracy and we don't need to adjust parameters too much. In most results, C isn't a sensitive parameter for this data. If you try to use poly kernel, degree is so important, please try to adjust it. In task 3, I try to combine linear and rbf kernel, but I think rbf is better than this mix kernel. Finally, SVM is good method for this data, I can get the good model that has 98% accuracy, 99% recall, 90% precision.