SVM lab

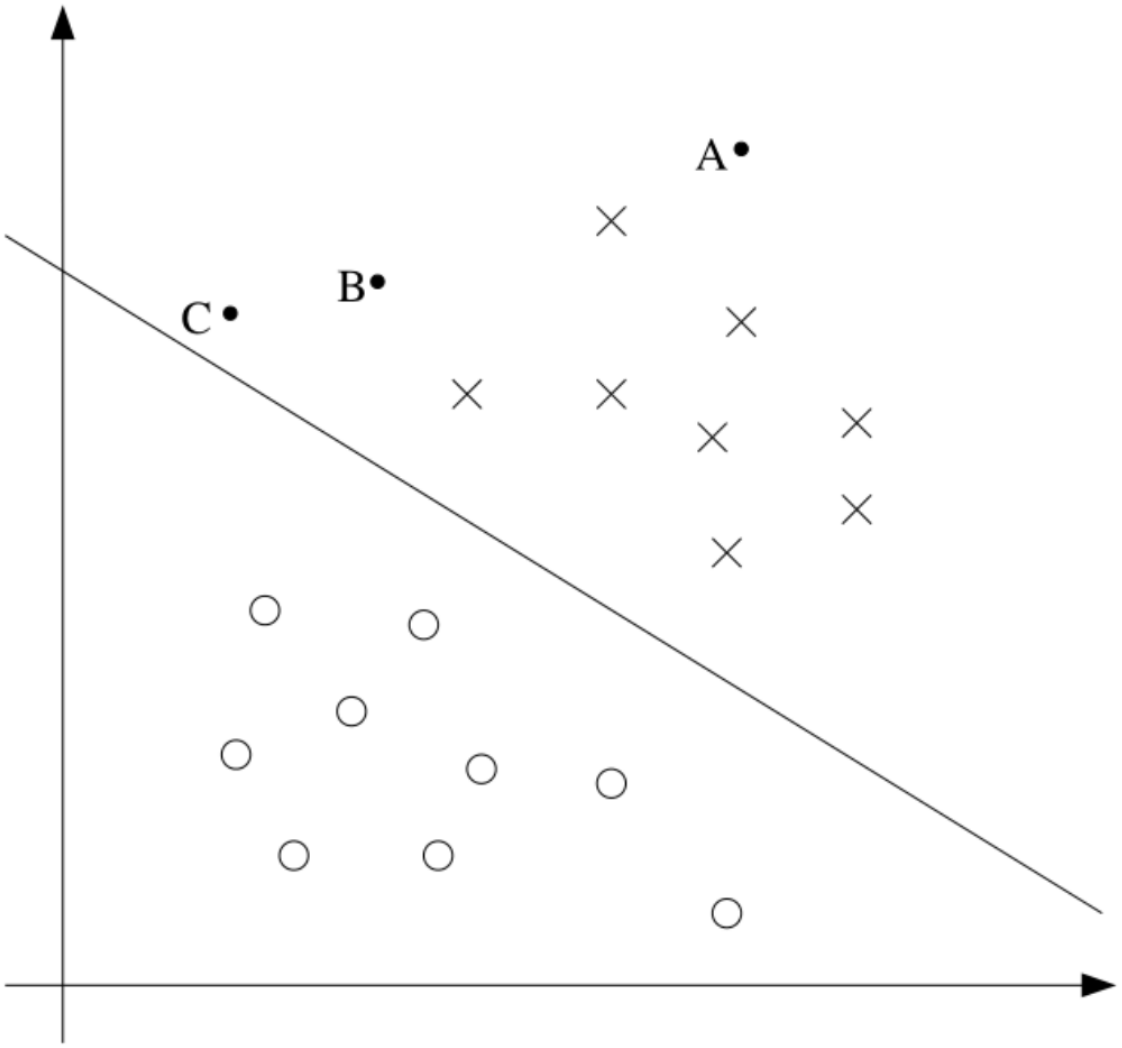
SVM 编程作业报告

22级JOHN班 - 涂文良

前言

首先，我想先略述一下我对于 SVM 的个人理解， 如果有不正确的地方希望助教指出。

SVM 的灵感来源于“分类的自信度”问题， 比如， 对于下图



实线表示分类面的话，显然我们对 A 的分类自信度比 C 要更大。 所以我们得到启发， 希望做到这样一件事： 给定一个训练集， 我们希望找到一个能让我们自信地正确分类样本点的分类面。

我们把一个样本点到分类的距离(functional margin)定义为:

$$\gamma_i = y_i(w^\top x_i + b)$$

然后， 我们希望最小的 functional margin 尽可能大。因此转化为优化问题：

$$\begin{aligned} &\max_{\tau,w,b} \tau \\ \text{s.t.} \quad &\forall i,\ y_i(w^\top X_i + b) \geq \tau \\ &\|w\| = 1 \end{aligned}$$

这个优化问题不是凸的，我们要转化为凸的，先正则化，变成：

$$\begin{aligned} &\max_{\tau,w,b} \frac{\tau}{\|w\|} \\ \text{s.t.} \quad &\forall i,\ y_i(w^\top X_i + b) \geq \tau \end{aligned}$$

之后再转化为:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad &\forall i,\ y_i(w^\top X_i + b) \geq 1 \end{aligned}$$

之后就可以用 Lagrange 乘子法或其他方法求解。

此外， SVM 还可以与 kernel method 相结合， 从而更好地构建一个能拟合数据的 feature map。

SVM 中， 决策边界是由最靠近边界的那个数据点来确定的， 所以把那个数据点叫做支持向量。 它们的 Lagrange 乘子 $\alpha > 0$. 但支持向量的 $\alpha < C$.

Outliers 是那些被错误分类的点或者在间隔边缘里面的点。 SVM with outliers 就是允许一些数据点违反边界的完整性， 从而达到更好的泛化能力。outliers 也满足 $\alpha > 0$.

具体到我们调包使用的 sklearn.svm 而言， C 参数（正则化参数）就是用于控制错误分类样本的惩罚程度的， 可以平衡边界的宽度和分类错误率。 边界的宽度越大， 模型的泛化能力就越好， 但是错误率就会更高。 所以， 更大的 C 值会对违规样本施加更大的惩罚， 从而使得边界的宽度变窄， 但是错误率会降低。 这是一个 tradeoff 的事情。

主要函数

我的主要函数（训练 + 测试）即如下所示， 代码有部分省略：

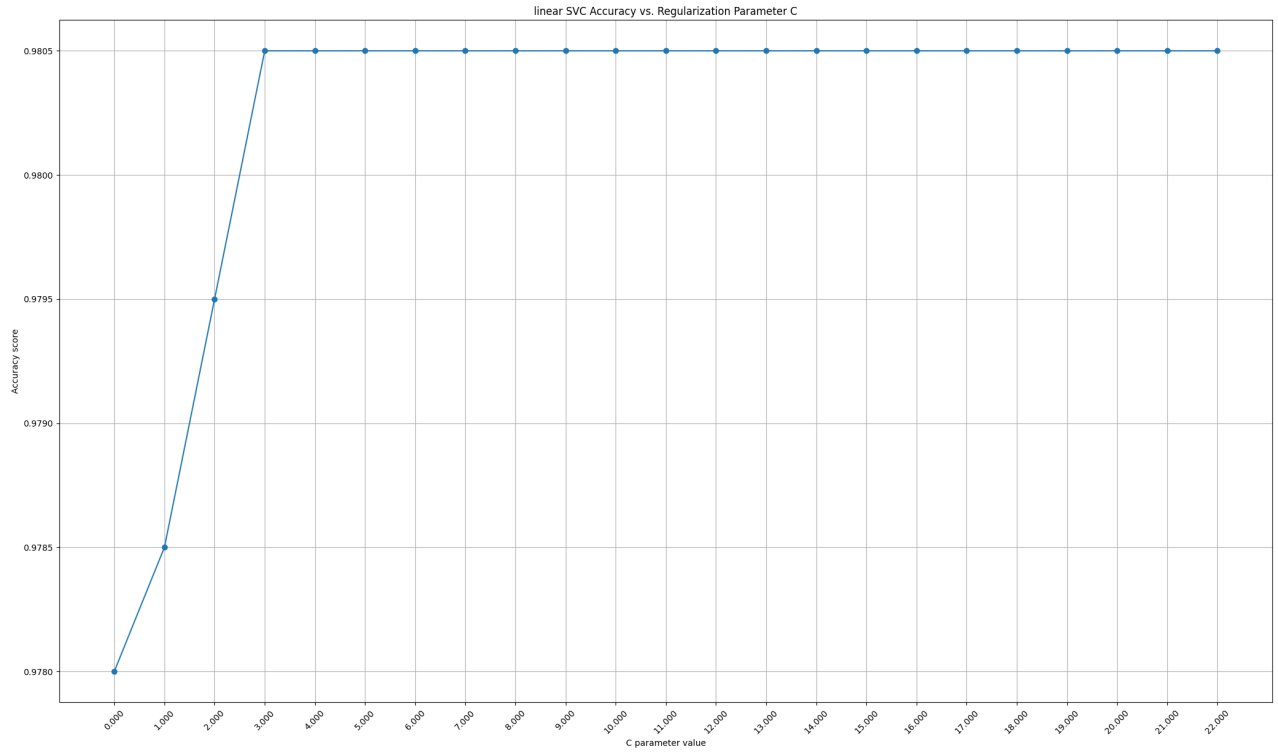
```
# 用于训练模型并生成score列表的函数
# 用于训练模型并生成score列表的函数
def training(mode, H_train, Y_train, H_test, Y_test):
    arg_C_list = []
    # 省略... 填充参数列表 arg_C_list

    score_list = []
    SVM_info_list = []
    indexed_scores_list = []
    for argC in arg_C_list:
        # 每轮选定一个参数进行训练
        svm = SVC(kernel = mode, C = argC)
        # 生成一个SVC模型，使用的核函数由我给定，正则化参数为argC
        svm.fit(H_train, Y_train)
```

```
        # 训练过程
        #print(svm.score(H_test, Y_test))
        score_list.append(svm.score(H_test, Y_test))
        # 填充score列表， 用于输出分类准确度（这里要在测试集上， 千万不能在训练集上测试了）
        svm_info_temp = SVM_info(svm.support_, svm.n_support_, svm.dual_coef_)
        SVM_info_list.append(svm_info_temp)
        #experiment(svm_info_temp, argC)
        # 获取决策函数的值
        decision_scores = svm.decision_function(H_train)
        # 结合分类的标签和决策函数，创建一个复合的列表
        indexed_scores = list(zip(range(len(Y_train)), Y_train, decision_scores))
        indexed_scores_list.append(indexed_scores)

    return score_list, arg_C_list, SVM_info_list, indexed_scores_list
```

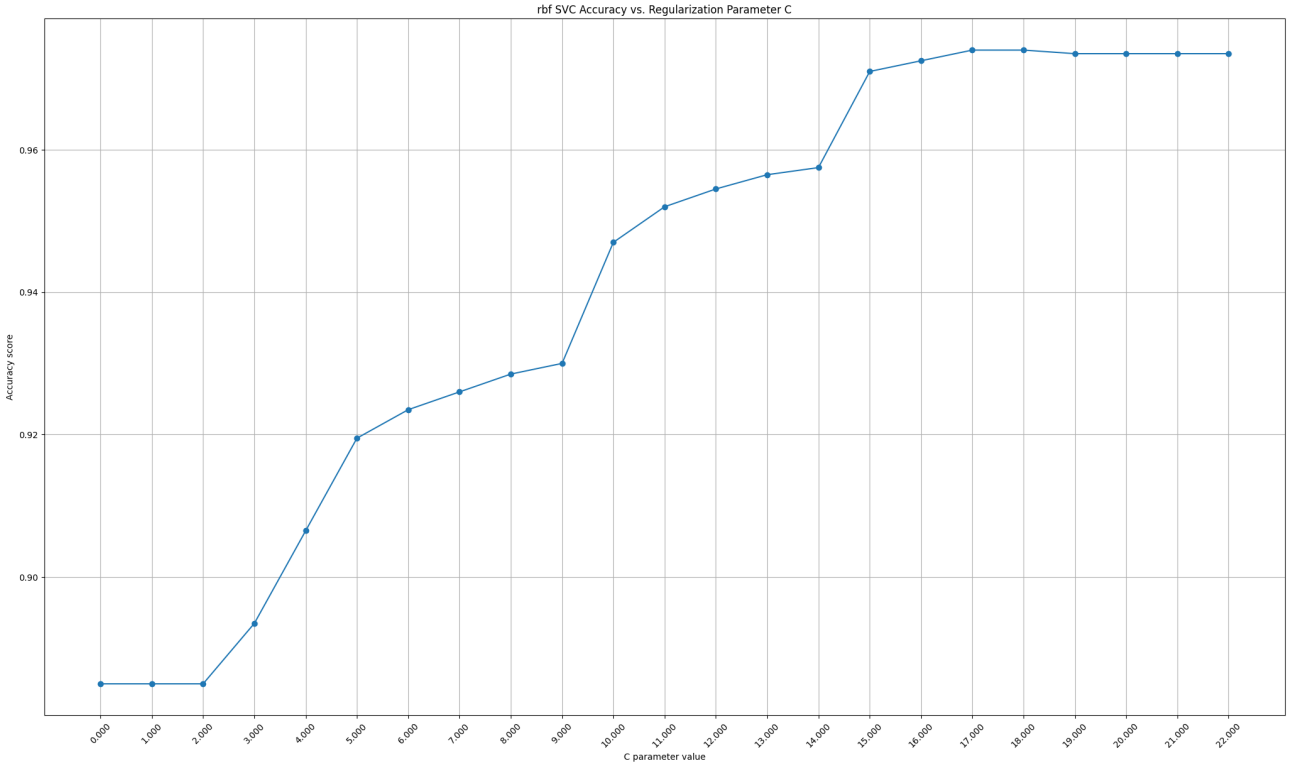
最后画出的图

任务1

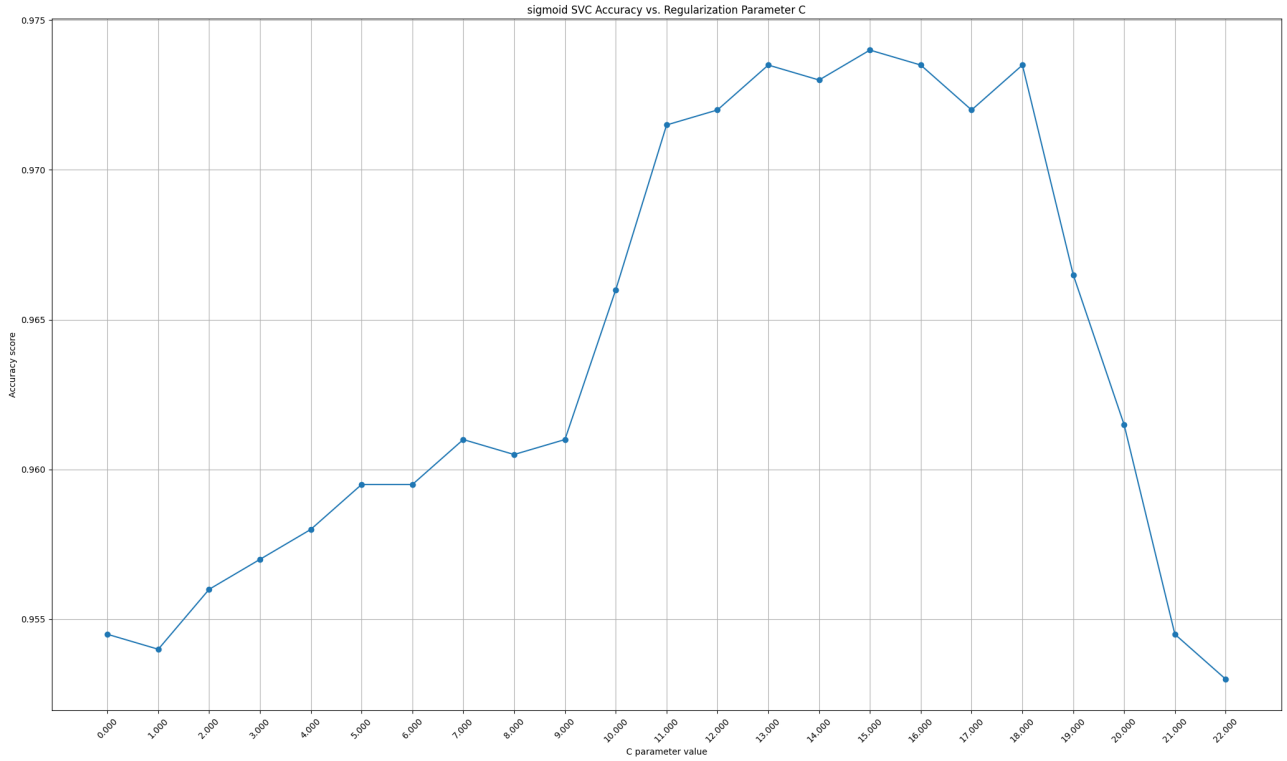
1.Linear:



2.RBF:



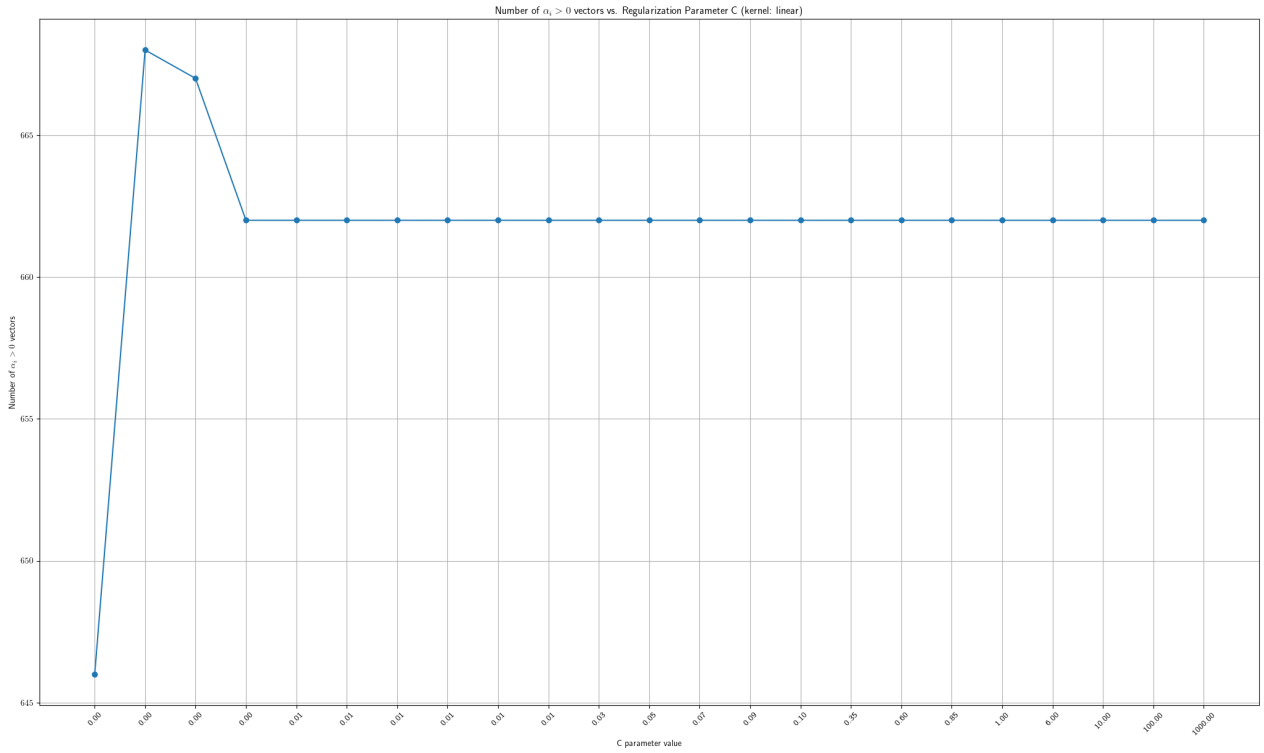
3.sigmoid



任务2

2.1

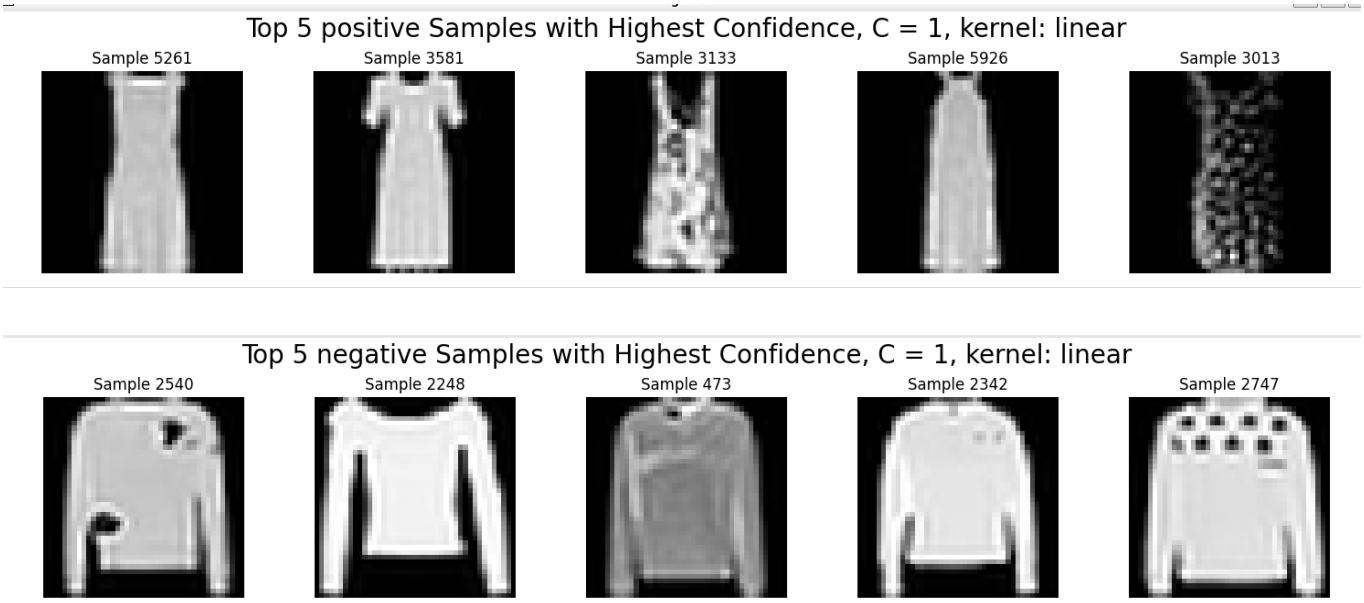
Linear kernel 中， 支持向量个数图：

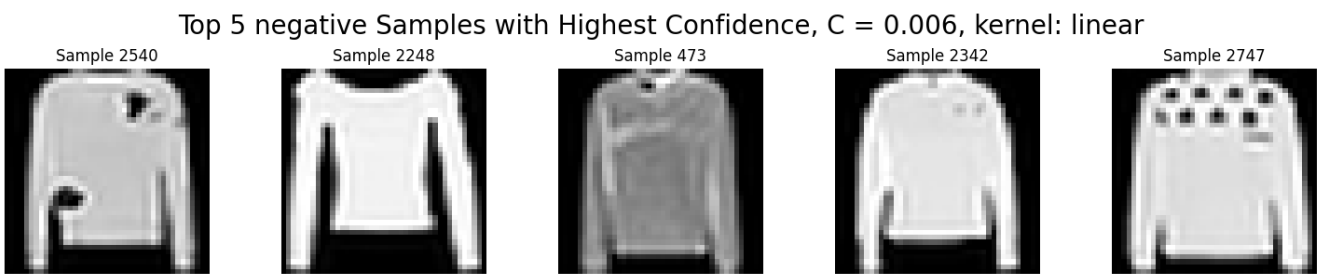


2.2

可视化分类信心最强的5个正样本和5个负样本：

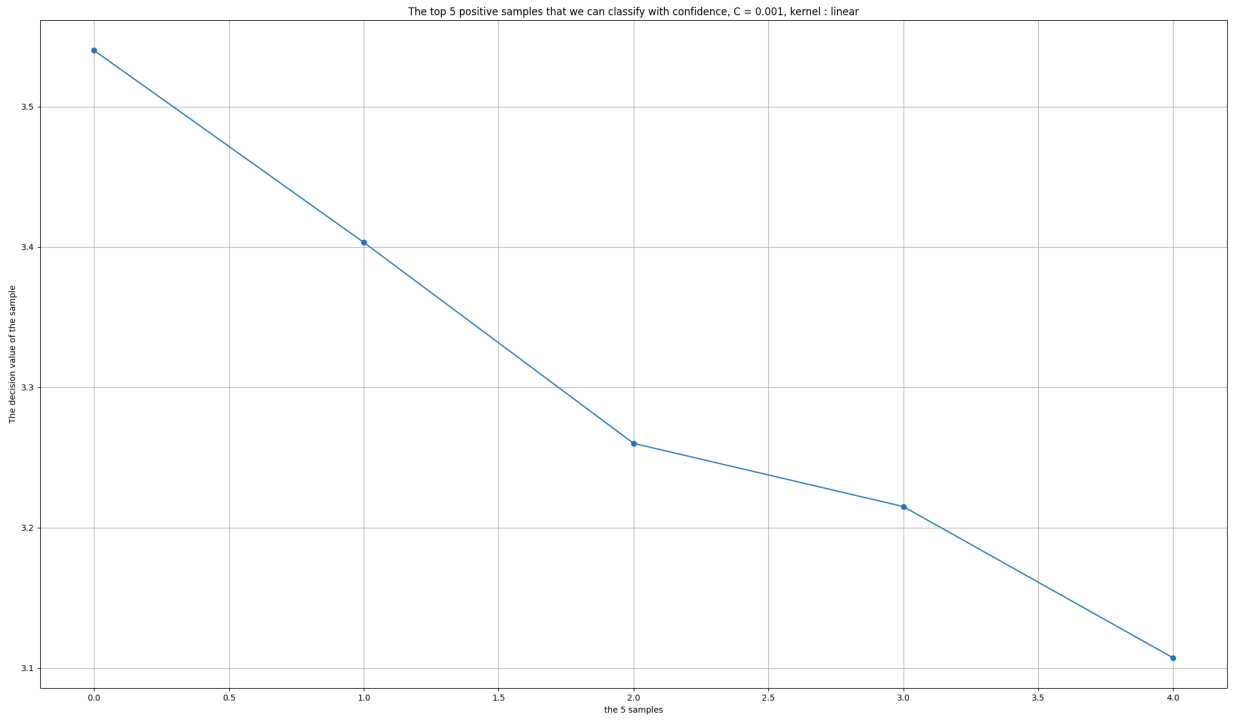
这个我对于每个C都画了一份， 在目录 pictures/confidence里面。 这里可以列举几张图作为例子：

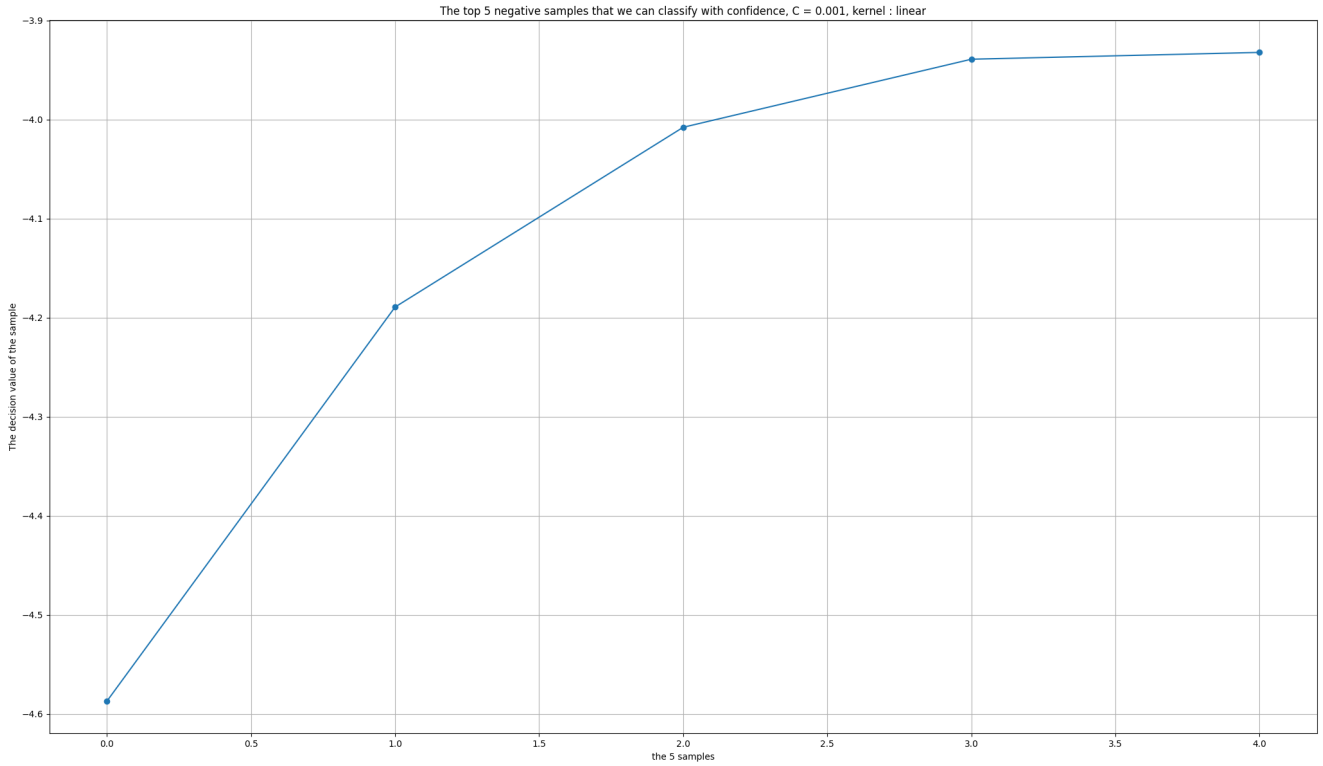




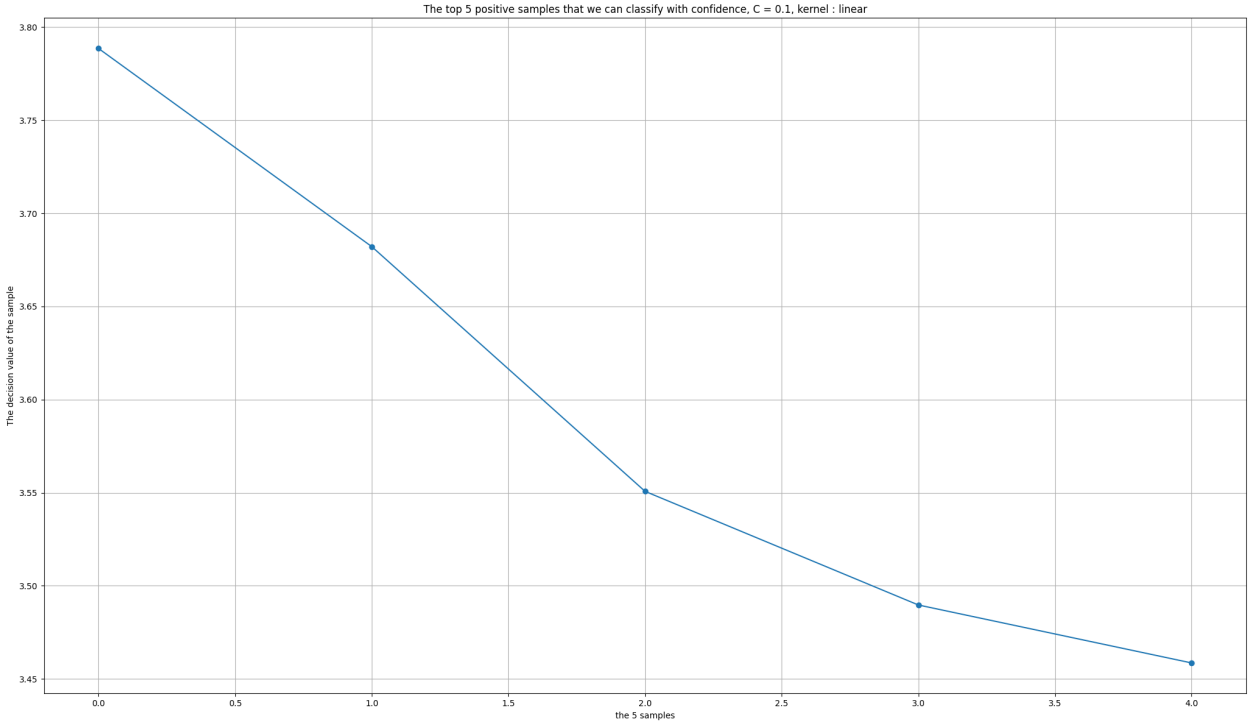
另外， 我还画了这些样本对应的决策值的图像，一同列举展示一下(也在同一目录下)：

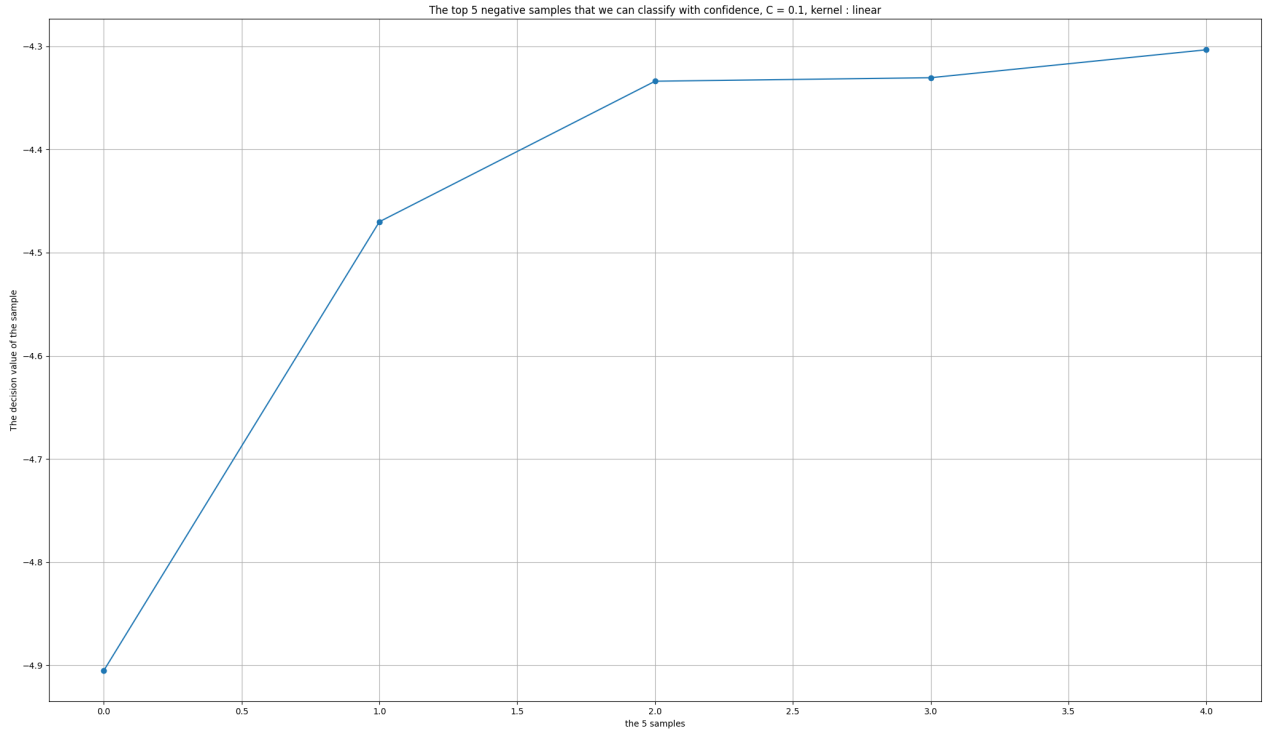
C = 0.001的：




The top 5 negative samples that we can classify with confidence, C = 0.001, kernel : linear
The decision value of the sample
the 5 samples
−3.9
−4.0
−4.1
−4.2
−4.3
−4.4
−4.5
−4.6
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

C = 0.1的:

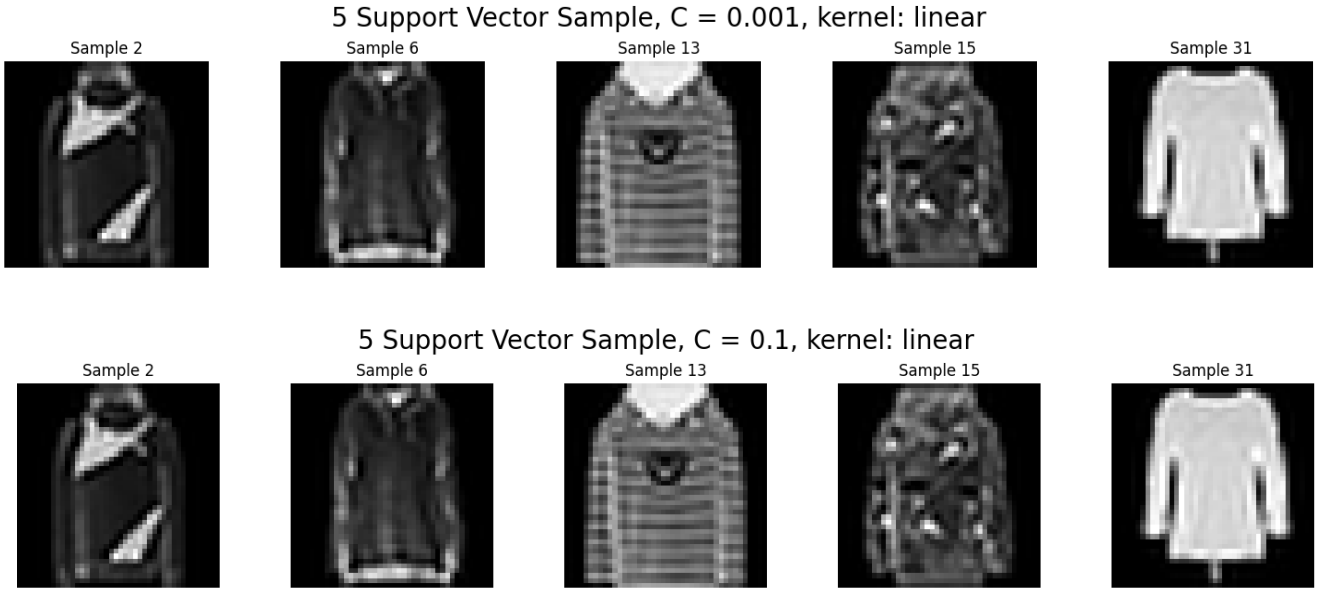

The top 5 positive samples that we can classify with confidence, C = 0.1, kernel : linear
The decision value of the sample
the 5 samples
3.80
3.75
3.70
3.65
3.60
3.55
3.50
3.45
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



2.3

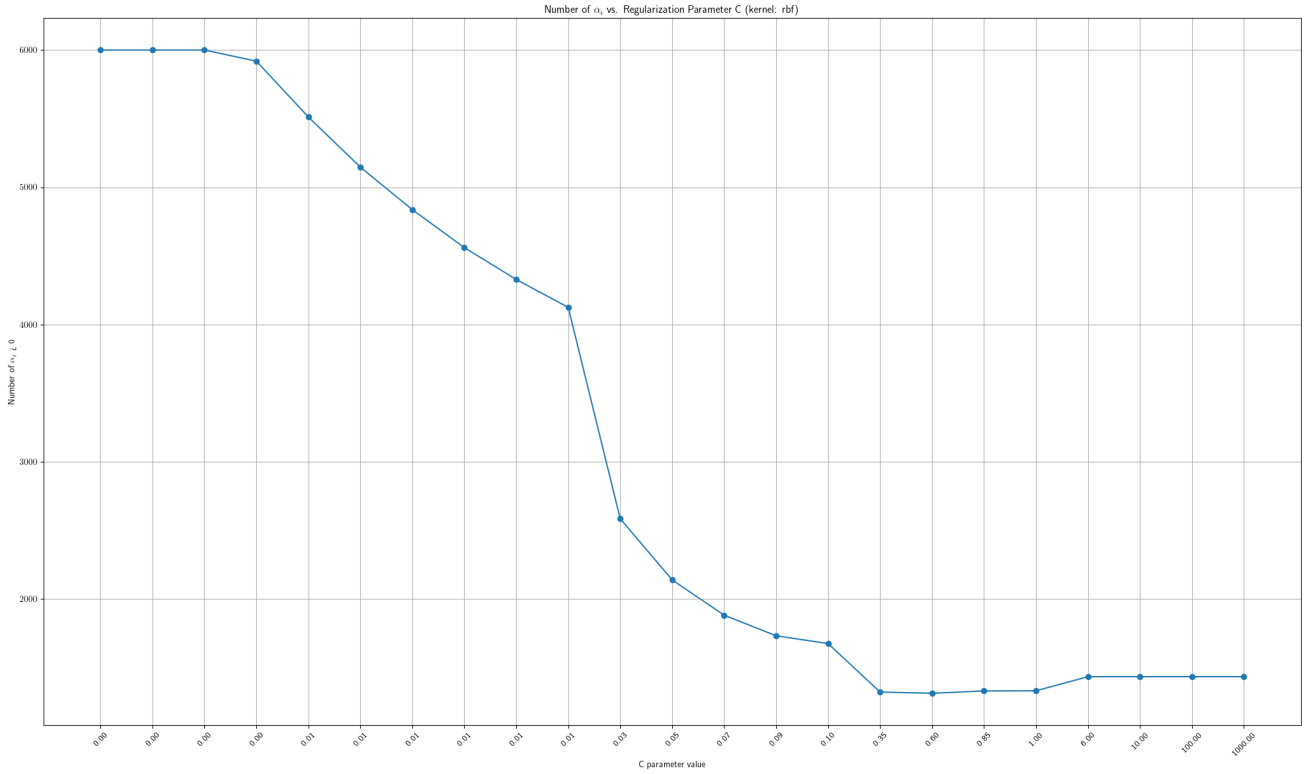
可视化 $\alpha_i > 0$ 的5个支持向量样本:

这个也给每个 C 值都对应画了一份， 在./sv 目录下面。这里列举几个:



任务3

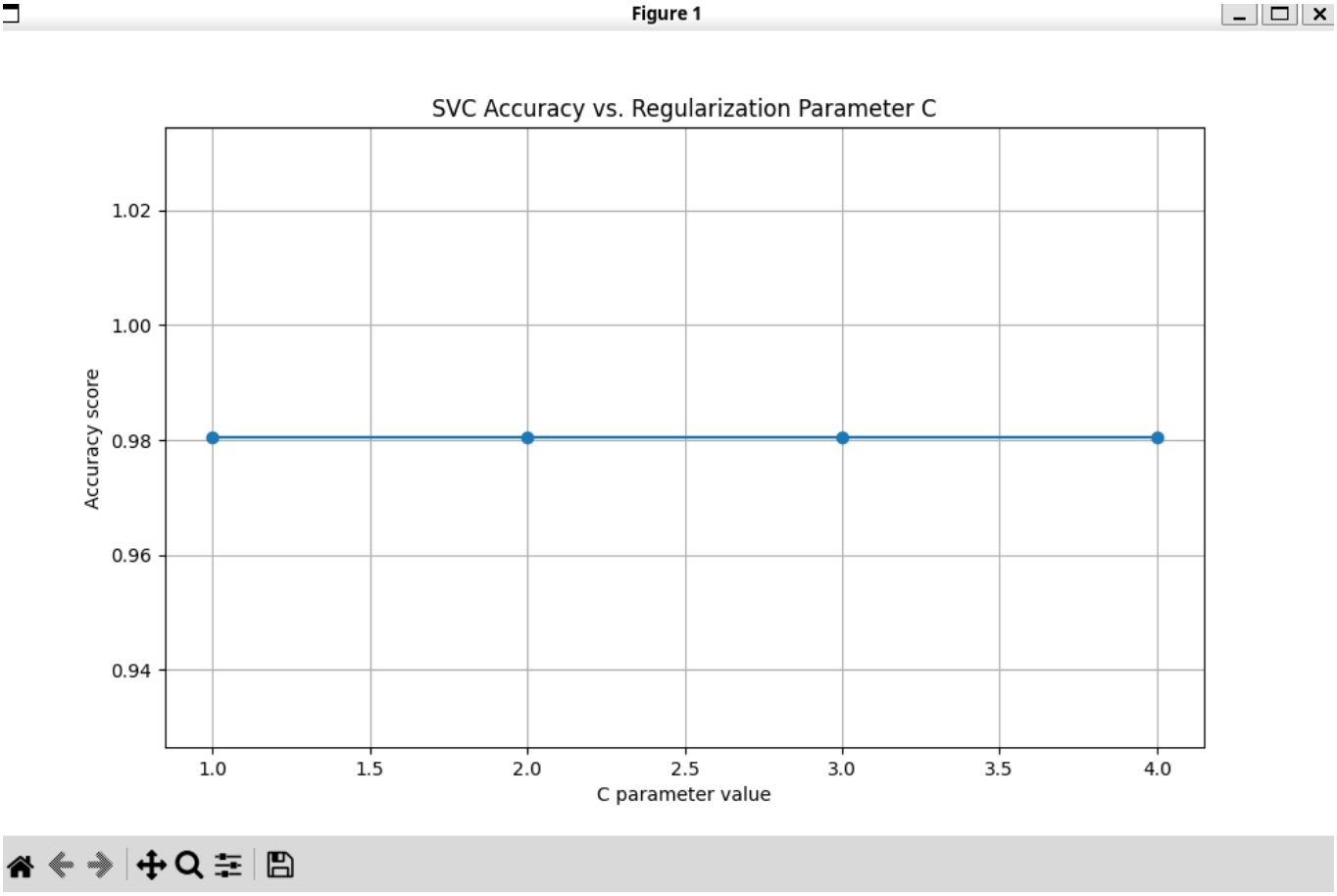
对于 RBF kernel SVM, 不同 C 值下 $\alpha_i > 0$ 的样本数量图:



出现过的卡壳点

1.

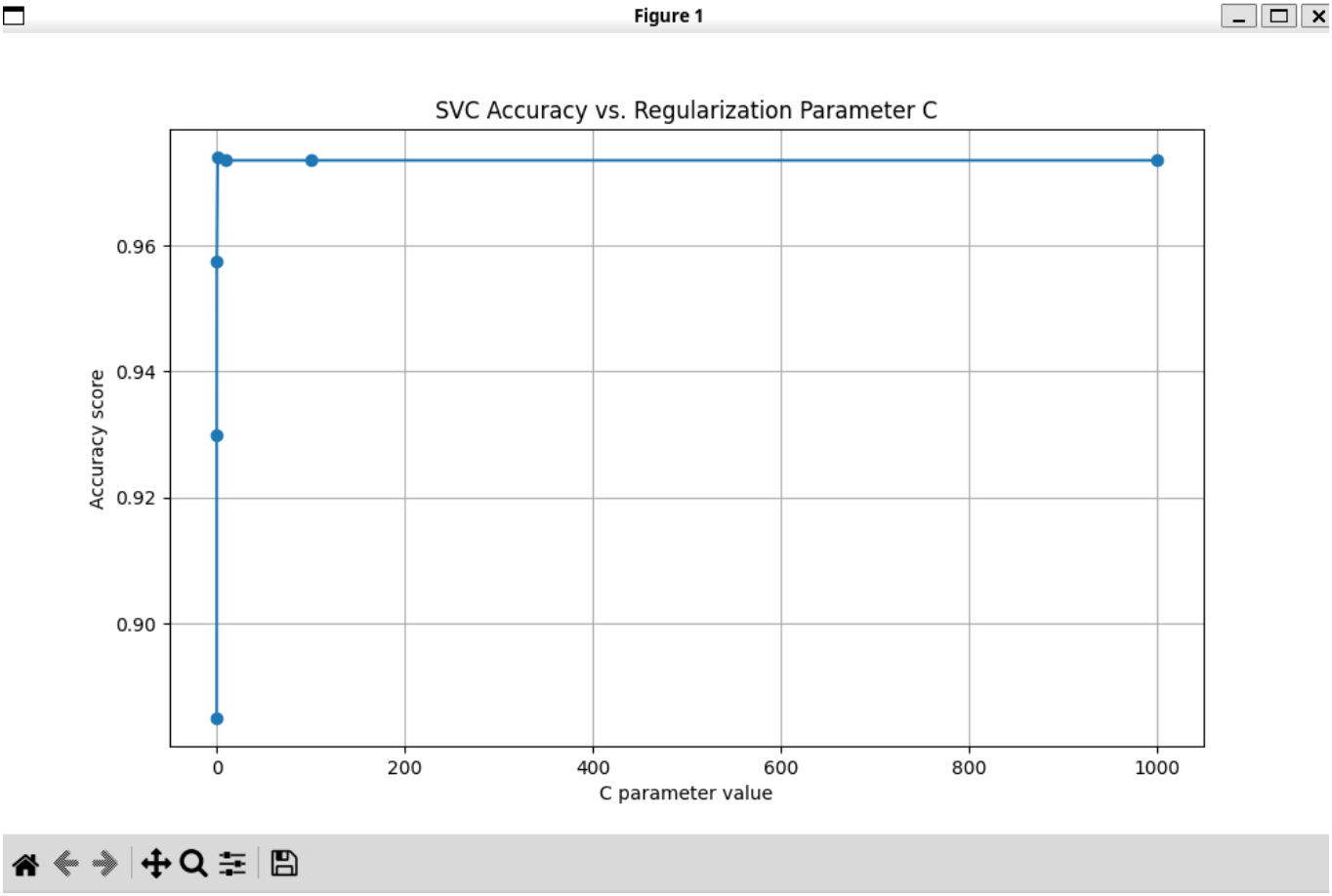
最开始的时候我把 C 选到了[1, 2, 3..., 10] 想测试一下



后来才知道是颗粒度不够， 恰好这个区域内， C (正则化系数) 的改变对分类的影响不大（此时分类准确度都很高了）。

所以我不应该在 [1, 10] 里面细分 C 的取值了。

这我用 [0.001, 0.01, 0.1, 1, 10, 100, 1000：




```
args :  [0.001, 0.01, 0.1, 1, 10, 100, 1000]
score :  [0.885, 0.93, 0.9575, 0.974, 0.9735, 0.9735, 0.9735]
```

我发现 0.001， 0.01， 0.1， 1 这部分区间之间分类准确度差别还挺大。所以我考虑在这里面细分。

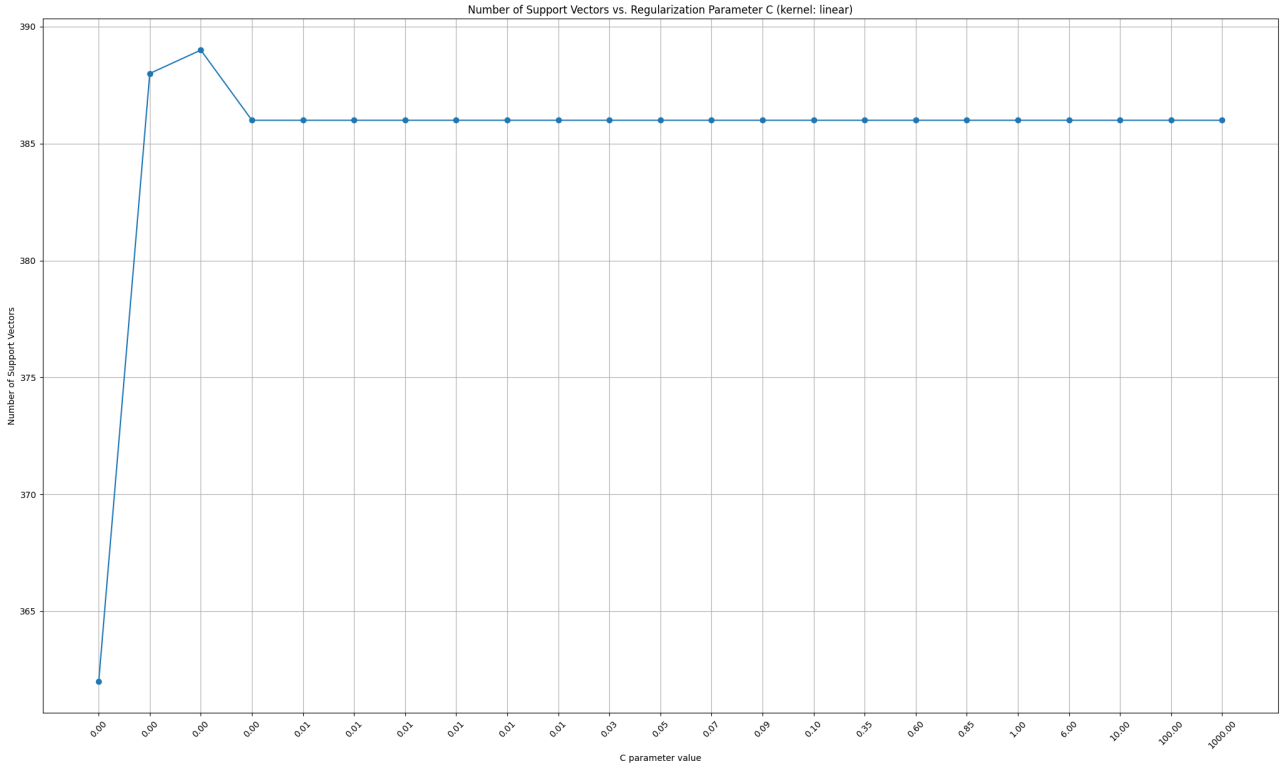
我最后用于生成 C 的参数列表的代码长这样：

```
# 以下两个函数是用于填充参数列表的辅助函数
def fill_arg_C_list(start, end, step, arg_C_list):
    arg_C_list += np.arange(start, end, step).tolist()

def fill_arg_gamma_list(arg_C_list):
    arg_C_list.append(10)
    arg_C_list.append(100)
    arg_C_list.append(1000)

# 用于训练模型并生成score列表的函数
```

```
def training(mode, H_train, Y_train, H_test, Y_test):
    arg_C_list = []
    fill_arg_C_list(0.001, 0.01, 0.001, arg_C_list)
    fill_arg_C_list(0.01, 0.1, 0.02, arg_C_list)
    fill_arg_C_list(0.1, 1, 0.25, arg_C_list)
    fill_arg_C_list(1, 10, 5, arg_C_list)
    fill_arg_gamma_list(arg_C_list)
    # 填充参数列表 arg_C_list
    ###...###
```

2.

最开始画支持向量图的时候画错了，画成了这样：



但是这个算的是， $y_i\alpha_i > 0$ 的样本个数。 实际上用 num_support_vectors_each_class 就行了。

3.

我最开始一直不知道 svm.support / *svm.n_support* 之类的方法， 是只输出支持向量还是输出支持向量 + outliers。

所以我写了个实验函数测试一下：

```
def experiment(SVM_info, C):
    # 用于调试的函数
    print("C : ", C)
    print("support_vector_indices_length : ",
len(SVM_info.support_vector_indices))
    print("support_vector_indices : ", SVM_info.support_vector_indices)
    print("num_support_vectors_each_class[0] : ",
SVM_info.num_support_vectors_each_class[0])
    print("num_support_vectors_each_class[1] : ",
SVM_info.num_support_vectors_each_class[1])
    print("dual_coefs_times_label_length : ",
len(SVM_info.dual_coefs_times_label))
    print("dual_coefs_times_label[0] : ",
SVM_info.dual_coefs_times_label[0])
    sum_coef_C = 0
    for coef in SVM_info.dual_coefs_times_label[0]:
        print("coef : ", coef)
        if(abs(coef - C) < 1e-5):
            sum_coef_C += 1
    print("sum_coef_C : ", sum_coef_C)
```

我的实验结果如下：

```
C :  0.001
support_vector_indices_length :  6000
support_vector_indices :  [   0    1    2 ... 5997 5998 5999]
num_support_vectors_each_class[0] :  3000
num_support_vectors_each_class[1] :  3000
dual_coefs_times_label_length :  1
dual_coefs_times_label[0] :  [-0.001 -0.001 -0.001 ...  0.001  0.001  0.001]
sum_coef_C :  6000
```

可以看出， 它输出的是支持向量 + outliers. 这样就没问题了， 直接拿它来用就行了。

4.

最开始我在可视化分类信心最强的5个样本的时候， 把函数写成了这样：

```
def plot_confidence_vec(args_list, SVM_info_list, indexed_scores_list, mode):
    for i, C in enumerate(args_list):
        indexed_scores = indexed_scores_list[i]
        top_positive_samples = sorted([s for s in indexed_scores if s[1] == 1], key=lambda x: -x[2])[:5]
        top_negative_samples = sorted([s for s in indexed_scores if s[1] == -1], key=lambda x: x[2])[:5]

        plt.figure(figsize=(10, 5))
        plt.suptitle(f'The top 5 samples that we can classify with confidence, C = {C}', fontsize=14)  # 设置整个图表的标题

        # 显示前5个正样本
        for j, (index, label, score) in enumerate(top_positive_samples):
            plt.subplot(2, 5, j + 1)
            plt.imshow(H_train[index].reshape(42, 42), cmap='gray')
            plt.title(f"Pos {j+1}: {score:.2f}")
            plt.axis('off')

        # 显示前5个负样本
        for j, (index, label, score) in enumerate(top_negative_samples):
            plt.subplot(2, 5, 5 + j + 1)
            plt.imshow(H_train[index].reshape(42, 42), cmap='gray')
            plt.title(f"Neg {j+1}: {score:.2f}")
            plt.axis('off')

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
        plt.savefig(f'./pictures/confidence/confidence_{C}.png')
        plt.show()
```

结果matplotlib画不出来。 后来我想到， 这个特征向量是超级高维的，肯定不是直接可视化这个向量。 所以最后我是先从index找到原图像 可视化这个原图像， 然后再画了一份它的decision value作为参考。

总结

通过这次 lab， 我加深了对 SVM 的理解。 真正上手做实验让我不停留在纸面的公式推导，而是在实践中理解每一个算法的本质和强大威力。另外，我对机器学习系统相关知识很感兴趣， 我还打算以后抽时间去看看 SVM 是如何高效实现的。