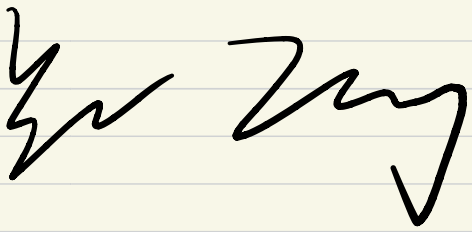
I certify that all solutions are entirely my own words and that I have not looked at another student's solutions.
I have given credits to all external solutions I consulted.



Collaborators: Simon Zhan.

Weijia Zeng.

Q1.

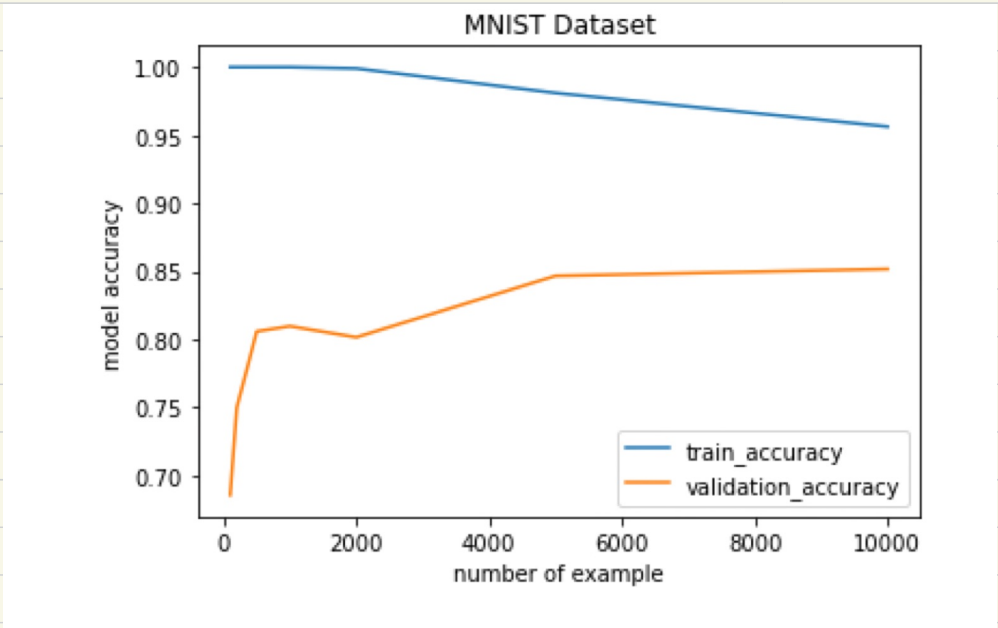
```
##################################
# Question 1: Data Partitioning #
##################################
mnist_val_size = 10000
spam_val_size = round(spam_data['training_data'].shape[0]*0.2)
cifar_val_size = 5000
validation_sizes = [mnist_val_size, spam_val_size, cifar_val_size]

def partition(data, labels, validation_size):
    # output: training_data, training_label, validation_data, validation_label
    if data.shape[0] != labels.shape[0]:
        raise Exception("data size mismatch with label size!")
    indices = np.arange(data.shape[0])
    np.random.shuffle(indices)
    data_shuffle = data[indices]
    labels_shuffle = labels[indices]
    return data_shuffle[validation_size:], labels_shuffle[validation_size:],␣
 ↪data_shuffle[:validation_size], labels_shuffle[:validation_size]
```

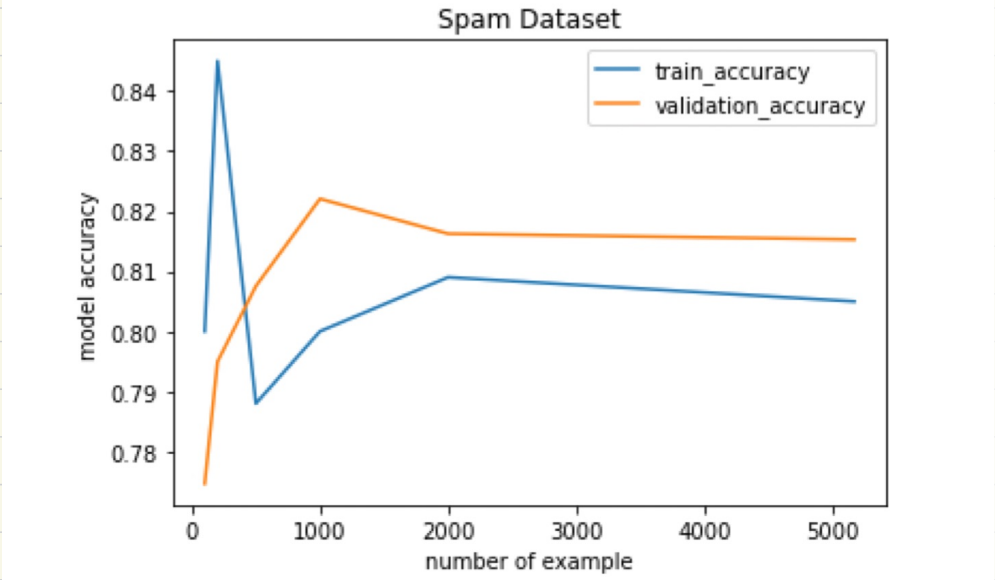
```
mnist_train = partition(mnist_data['training_data'],␣
 ↪mnist_data['training_labels'], mnist_val_size)
mnist_train[2].shape
```

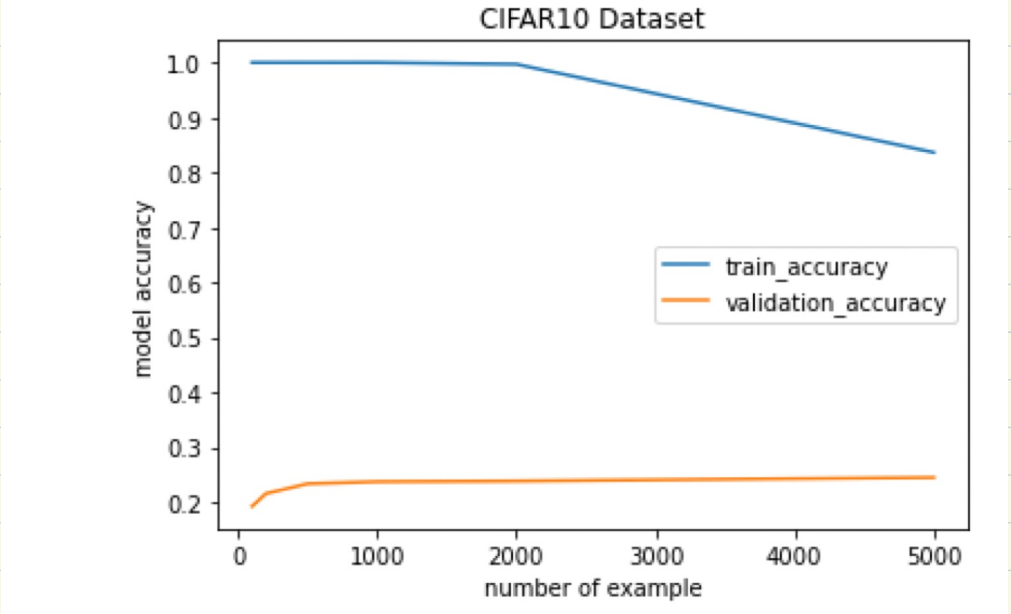
```
(10000, 784)
```

↳ Partition is achieved by randomly shuffling the indices of input list then divide into two parts.

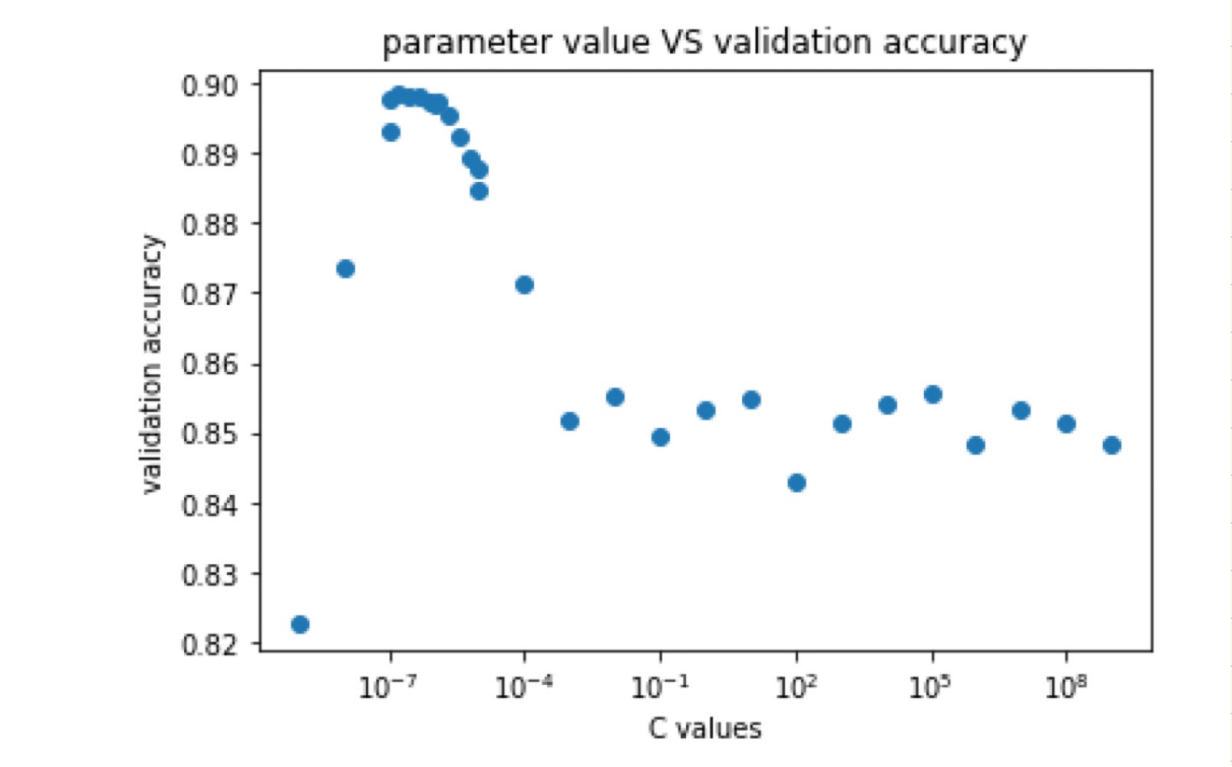
Q2、


MNIST Dataset
model accuracy
number of example
train_accuracy
validation_accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
2000
4000
6000
8000
10000


Spam Dataset
model accuracy
number of example
train_accuracy
validation_accuracy
0.84
0.83
0.82
0.81
0.80
0.79
0.78
0
1000
2000
3000
4000
5000


CIFAR10 Dataset
model accuracy
number of example
train_accuracy
validation_accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
1000
2000
3000
4000
5000

Q3



- C values tried:

```
[1.00000000e-09 1.00000000e-08 1.00000000e-07 1.00000000e-06
 1.00000000e-05 1.00000000e-04 1.00000000e-03 1.00000000e-02
 1.00000000e-01 1.00000000e+00 1.00000000e+01 1.00000000e+02
 1.00000000e+03 1.00000000e+04 1.00000000e+05 1.00000000e+06
 1.00000000e+07 1.00000000e+08 1.00000000e+09 1.00000000e-07
 1.66810054e-07 2.78255940e-07 4.64158883e-07 7.74263683e-07
 1.29154967e-06 2.15443469e-06 3.59381366e-06 5.99484250e-06
 1.00000000e-05]
```

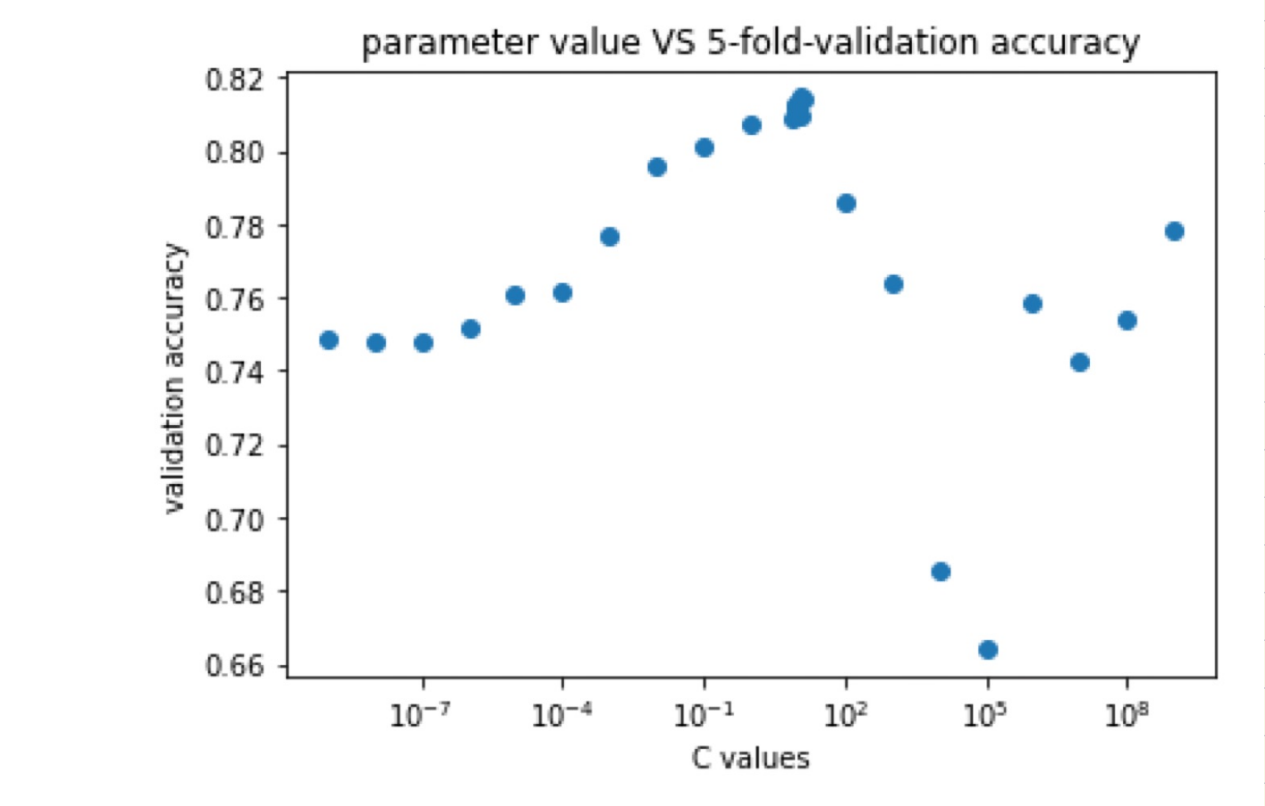
- Corresponding Accuracies:

```
[0.8227 0.8736 0.8931 0.8969 0.8878 0.8712 0.852  0.8553 0.8495 0.8532
 0.8547 0.8432 0.8515 0.8542 0.8556 0.8482 0.8533 0.8516 0.8484 0.8977
 0.8983 0.8982 0.8979 0.8975 0.8974 0.8956 0.8922 0.8892 0.8848]
```

- Best C Value:

```
highest c value is: 1.668100537200059e-07,with accuracy: 0.8983
```

Q4



- C values tried:

```
[1.00000000e-09 1.00000000e-08 1.00000000e-07 1.00000000e-06
 1.00000000e-05 1.00000000e-04 1.00000000e-03 1.00000000e-02
 1.00000000e-01 1.00000000e+00 1.00000000e+01 1.00000000e+02
 1.00000000e+03 1.00000000e+04 1.00000000e+05 1.00000000e+06
 1.00000000e+07 1.00000000e+08 1.00000000e+09 7.94328235e+00
 8.36030694e+00 8.79922544e+00 9.26118728e+00 9.74740226e+00
 1.02591437e+01 1.07977516e+01 1.13646367e+01 1.19612833e+01
 1.25892541e+01]
```

- Corresponding Accuracies:

```
[0.74883936 0.74806436 0.74806586 0.7517467  0.7608264  0.76198918
 0.77668022 0.79582485 0.80143059 0.80704174 0.81225614 0.78597165
 0.76429176 0.68538372 0.66415253 0.75871107 0.74225175 0.75426438
 0.77862548 0.80916361 0.80955438 0.81206253 0.81225764 0.81187042
 0.81129164 0.81360992 0.81457648 0.80954373 0.8141872 ]
```

- Best C Value:

```
highest c value is: 11.364636663857247,with 5-fold-accuracy: 0.8145764770741643
```

Q5.

Kaggle account name: Jeffrey Zhangyc 789.

MNIST Accuracy:	0.98810	Rank #8
SPAM Accuracy:	0.80126	
CIFAR 10 Accuracy:	0.58850	Rank #14.

I first tried to use the c-value tuning & k-fold validation with entire train set as the training sample size through out training process. It worked fine with MNIST and SPAM but CIFAR took forever to train. So I tried to remove the k-fold part for CIFAR but tuning time for 1 c-value still takes few hours. Then I realized that the training sample size was too large so I reduced the training size of C-value tuning part to 8000 for MNIST and 4000 for CIFAR. Then the training time reduced to about 20 min for about 10 c values.

My focus was on Mnist & Cifar dataset. I tried to tune C twice: first on a log scale from 10^{-7} to 10^{5} then another 10 c values equally spaced between ± 5 around the best C found in first round. This helped to find a more accurate C so accuracy increased a bit.

The most effective technique was to change the kernel used by SVM. I tried "linear", "poly", and "rbf" with all implementations described above, and "rbf" was the most effective kernel which increased accuracy to about 0.96 for Mnist and 0.48 for Cifar.

After finding best kernel I raised to train size for Mnist to 15000 and Cifar to 7000 to find a C for large sample size.

Finally, I tried to preprocessing the data by using HOG (skimage.feature.hog), where I convert Mnist vector to 28×28, cifar to 32×32×3 (not directly reshape) and feed in to hog with pixel compression from 2×2 to 5×5. The best result obtained when pixel compression was 4×4, and use this as input for SVM.

Increasing traing sample size on tuning process didn't improve the accuracy by a lot so I just kept at medium level for tuning process, but final model was trained on entire dataset.

Q6

a. $f(w,\alpha) = |w|^2 - \sum_{i=1}^{n} \lambda_i (y_i(x_i \cdot w + \alpha) - 1)$

$$\frac{\partial f(w,\alpha)}{\partial w} = (w' \cdot w + w \cdot w') - \left(\sum_{i=1}^{n} \frac{\partial \lambda_i (y_i(x_i \cdot w + \alpha) - 1)}{\partial y_i(x_i \cdot w + \alpha) - 1} \frac{\partial y_i(x_i \cdot w + \alpha) - 1}{\partial x_i \cdot w + \alpha} \frac{\partial x_i w \alpha}{\partial w} \right)$$

$$0 = 2w - \sum_{i=1}^{n} \lambda_i \cdot y_i \cdot x_i$$

$$\boxed{w = \frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i}, \quad \text{also } \frac{\partial f'(w,\alpha)}{\partial w^2} = 2 > 0 \text{ so it's minimum.}$$

$$\frac{\partial f(w,\alpha)}{\partial \alpha} = \sum_{i=1}^{n} \frac{\partial \lambda_i (y_i(x_i \cdot w + \alpha) - 1)}{\partial y_i(x_i \cdot w + \alpha) - 1} \frac{\partial y_i(x_i \cdot w + \alpha) - 1}{\partial x_i \cdot w + \alpha} \frac{\partial x_i w \alpha}{\partial \alpha}$$

$$\boxed{0 = \sum_{i=1}^{n} \lambda_i \cdot y_i}$$

$$f(w,\alpha) = \frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i \cdot \frac{1}{2} \sum_{j=1}^{n} \lambda_j y_j x_j - \sum_{i=1}^{n} \lambda_i \left(y_i \left(x_i \cdot \frac{1}{2} \sum_{j=1}^{n} \lambda_j y_j x_j + \alpha \right) - 1 \right)$$

$$= \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j - \sum_{i=1}^{n} \lambda_i y_i \alpha + \sum_{i=1}^{n} \lambda_i$$

$$= -\frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j - \alpha \underbrace{\sum_{i=1}^{n} \lambda_i y_i}_{0} + \sum_{i=1}^{n} \lambda_i$$

$$= \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j$$

subject to $\sum_{i=1}^{n} \lambda_i y_i = 0$. (come from partial derivative of α)

b.

$$r(x) = \begin{cases} +1 & \text{if } w \cdot x + \alpha \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$w^* = \frac{1}{2} \sum_{i=1}^{n} \lambda_i^* y_i x_i$$

$$r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2} \sum_{i=1}^{n} \lambda_i^* y_i x_i \cdot x \geq x \\ -1 & \text{otherwise.} \end{cases}$$

c. $\lambda_i^*(y_i(X_i \cdot w^* + \alpha^*) - 1) = 0 \quad \forall i \in \{1, \dots, n\}$

$$1 = y_i(X_i \cdot w^* + \alpha^*) \quad \forall \lambda_i^* > 0.$$

Since we are trying to find $\min_{w,\alpha} |w^2|$ subject to

$$y_i(X_i \cdot w + \alpha) \geq 1, \quad \forall i \in \{1, \dots, n\}$$

for hard-margin SVM

so this implies the points corresponding to $\lambda_i^* > 0$ is on the edge of this margin, and provides the optimal solution to the problem.

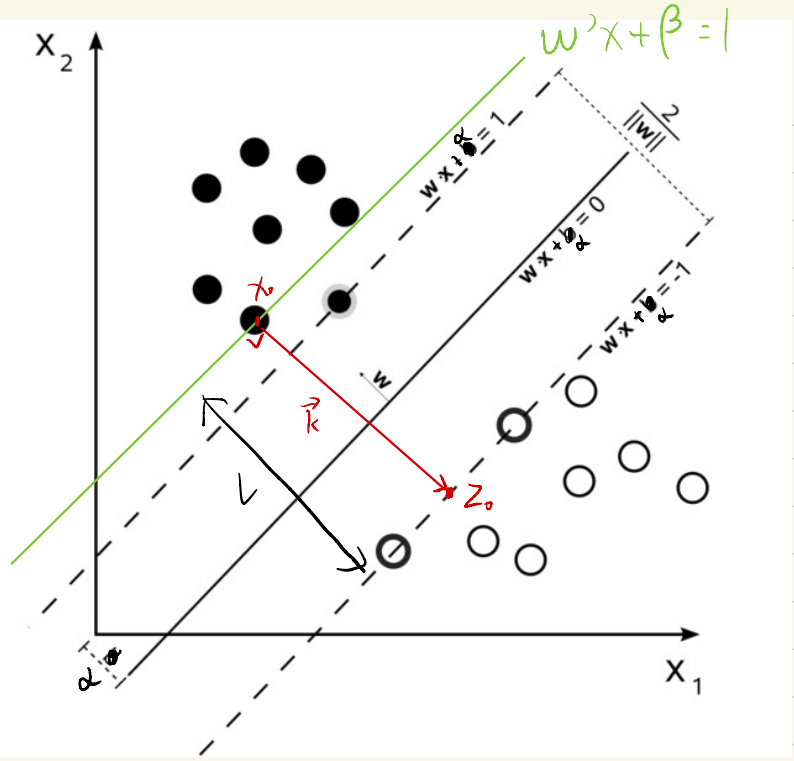
d. Because $\lambda_i \geq 0$, so if $\lambda_i^* \neq 0$, $\lambda_i^* = 0$, then $\lambda_i(y_i(X_i w + \alpha) - 1)$ is 0 so it does not have a minimizing effect on equation 3, so these points can remove condition $y_i(X_i w^* + \alpha^*) \geq 1$ may not be satisfied, so the corresponding point (X_i, Y_i) cannot be used to optimize the parameters of SVM.

Therefore, the supporting vectors are the only points needed as they satisfy the optimization condition and defines the edge of classification.

e. Proof: At least one support vector exist for each class, +1, -1.

Contradiction Assumption: There is no support vector for either of the class +1.

condition for support vector: $y_i(X_i \cdot w + \alpha) = 1$



Credit: Piazza @ 44_f15. Jason Zang.

Since we assume there is no support vector for +1 class, then the margin for +1 class could be shift further from the original margin. Now the distance between two margins increases, and since the distance is determined by $\frac{2}{|w|}$ (reference: svm-tutorial.com/2015/06/svm-understanding-math-part-3/) the new w' defining the distance is smaller than w.

Since there is no support vector for +1 class, we don't have $x_i \cdot w + \alpha = 1$,

so $x_i \cdot w + \alpha > 1$, and the new margin is now $x_i \cdot w + \beta = 1$, represented by the green line.

↳ $x_i \cdot w + \alpha - \varepsilon = 1$ where $\alpha > \beta$ & $\beta = \alpha - \varepsilon$, $\varepsilon > 0$

Let L be the length between the two planes defining the new margins.

Let $\vec{k}$ be the normal vector pass through these hyperplanes with magnitude L, where x_0 & z_0 are points of intersection for $\vec{k}$ with the planes, so $wx_0 + \beta = 1$, $wz_0 + \alpha = -1$.

Since w is also the normal vector to both planes, (this prove is also inspired by the reference link page above).

$\vec{k} = -L \cdot \frac{\vec{w}}{|\vec{w}|}$, $L = \frac{2}{|w'|}$ → $\vec{k}$ has opposite direction with $\vec{w}$

also, $z_0 = x_0 + k$

↳ $wz_0 + \alpha = -1$

$w(x_0 + k) + \alpha = -1$

$$w\left(x_0 - l\cdot\frac{w}{|w|}\right)+\alpha=-1$$

$$wx_0 - l\cdot\frac{|w|^2}{|w|}+\alpha=-1$$

$$\underbrace{(wx_0+\alpha-\varepsilon)}_{+1} - l\cdot|w|+\varepsilon=-1$$

$$-l\cdot|w|+\varepsilon=-2$$

$$l=\frac{2+\varepsilon}{|w|}$$

$$l=\frac{2\left(1+\frac{\varepsilon}{2}\right)}{|w|}$$

$$l=\frac{2}{\left|\frac{\vec{w}}{1+\frac{\varepsilon}{2}}\right|}$$

Since we know that the distance between two hyperplanes must be $l=\frac{2}{|w|}$,

so we found a new w' where $w'=\frac{w}{1+\frac{\varepsilon}{2}}$,

then the actual α' that draws the actual hyperplane will be

$$\begin{cases} x_i\cdot w'+\alpha'=1 \\ x_i\cdot w+\alpha-\varepsilon=1 \end{cases} \Rightarrow \begin{cases} x_i\cdot\frac{w}{1+\frac{\varepsilon}{2}}+\alpha'=1 \\ x_i\cdot w=1+\varepsilon-\alpha \end{cases}$$

$$\hookrightarrow x_i w=\cancel{1}+\frac{\varepsilon}{2}-\alpha'\left(1+\frac{\varepsilon}{2}\right)=\cancel{1}+\varepsilon-\alpha$$

$$+\alpha'\left(1+\frac{\varepsilon}{2}\right)=\frac{\varepsilon}{2}+\alpha$$

$$\alpha'=\frac{\alpha-\frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}}$$

$$\hookrightarrow \alpha'<\alpha$$

Test this on the -1 class which support vector exist, where $x_i w+\alpha=-1$ in this case

$$x_i w'+\alpha'=x_i\frac{w}{1+\frac{\varepsilon}{2}}+\frac{\alpha-\frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}}$$

$$=\frac{(x_i w+\alpha)-\frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}}$$

$$=\frac{-1-\frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}}$$

$$=-\frac{1+\frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}}$$

$$=-1 \longrightarrow \text{Constrains on } -1 \text{ class satisfied.}$$

so $w' < w$, $\alpha' < \alpha$, and all other constrains are not violated,

this means that w & α is not the optimal solution to the original SVM problem under our assumption, so the assumption must be false.

This proof is symmetrical since choosing -1 class for the contradictory assumption and following the same steps (all steps will be identical and symmetric) will give the same conclusion.

Appendix.

question_code

January 27, 2021

[1]:
```
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
for data_name in ["mnist", "spam", "cifar10"]:
    data = io.loadmat("data/%s_data.mat" % data_name)
    print("\nloaded %s data!" % data_name)
    fields = "test_data", "training_data", "training_labels"
    for field in fields:
        print(field, data[field].shape)
```

```
loaded mnist data!
test_data (10000, 784)
training_data (60000, 784)
training_labels (60000, 1)

loaded spam data!
test_data (5857, 32)
training_data (5172, 32)
training_labels (5172, 1)

loaded cifar10 data!
test_data (10000, 3072)
training_data (50000, 3072)
training_labels (50000, 1)
```

[2]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn import svm
from sklearn.metrics import accuracy_score # This is the only function you are␣
 ↪allowed to use.
from scipy import io
import csv
import os
```

```
import argparse
import warnings
warnings.simplefilter('ignore')

def load_data(name):
    # Return the specified dataset by NAME
    return io.loadmat("data/%s_data.mat" % name)
mnist_data = load_data('mnist')
spam_data = load_data('spam')
cifar_data = load_data('cifar10')
all_data = [mnist_data, spam_data, cifar_data]
```

[3]:
```
len(mnist_data['training_data'])
```

[3]: 60000

[4]:
```
##################################
# Question 1: Data Partitioning #
##################################
mnist_val_size = 10000
spam_val_size = round(spam_data['training_data'].shape[0]*0.2)
cifar_val_size = 5000
validation_sizes = [mnist_val_size, spam_val_size, cifar_val_size]

def partition(data, labels, validation_size):
    # output: training_data, training_label, validation_data, validation_label
    if data.shape[0] != labels.shape[0]:
        raise Exception("data size mismatch with label size!")
    indices = np.arange(data.shape[0])
    np.random.shuffle(indices)
    data_shuffle = data[indices]
    labels_shuffle = labels[indices]
    return data_shuffle[validation_size:], labels_shuffle[validation_size:],␣
 ↪data_shuffle[:validation_size], labels_shuffle[:validation_size]
```

[5]:
```
mnist_train = partition(mnist_data['training_data'],␣
 ↪mnist_data['training_labels'], mnist_val_size)
mnist_train[2].shape
```

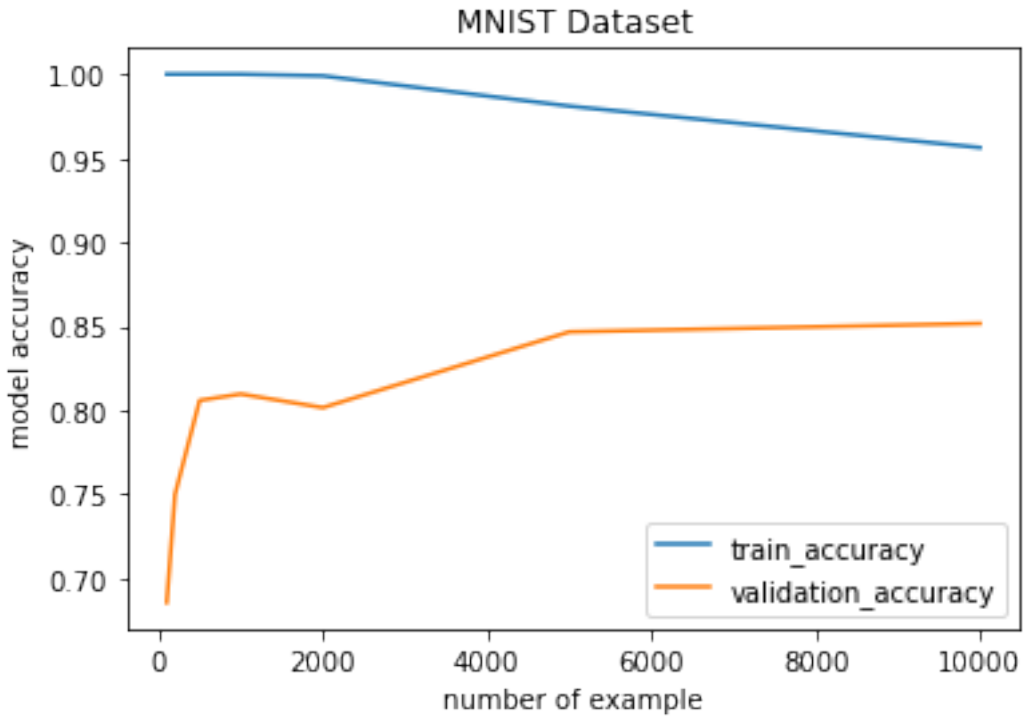
[5]: (10000, 784)

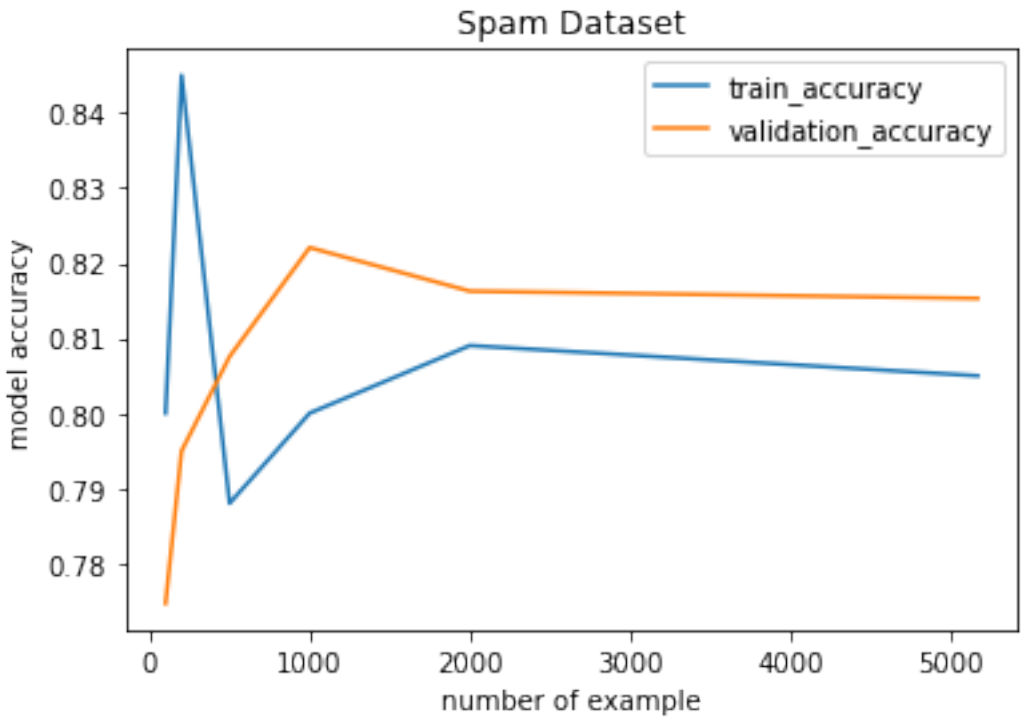
[6]:
```
##################################
# Question 2: SVMs               #
##################################

def train(X, Y, c = 1.0):
    # create, train, and return and SVM Model
```

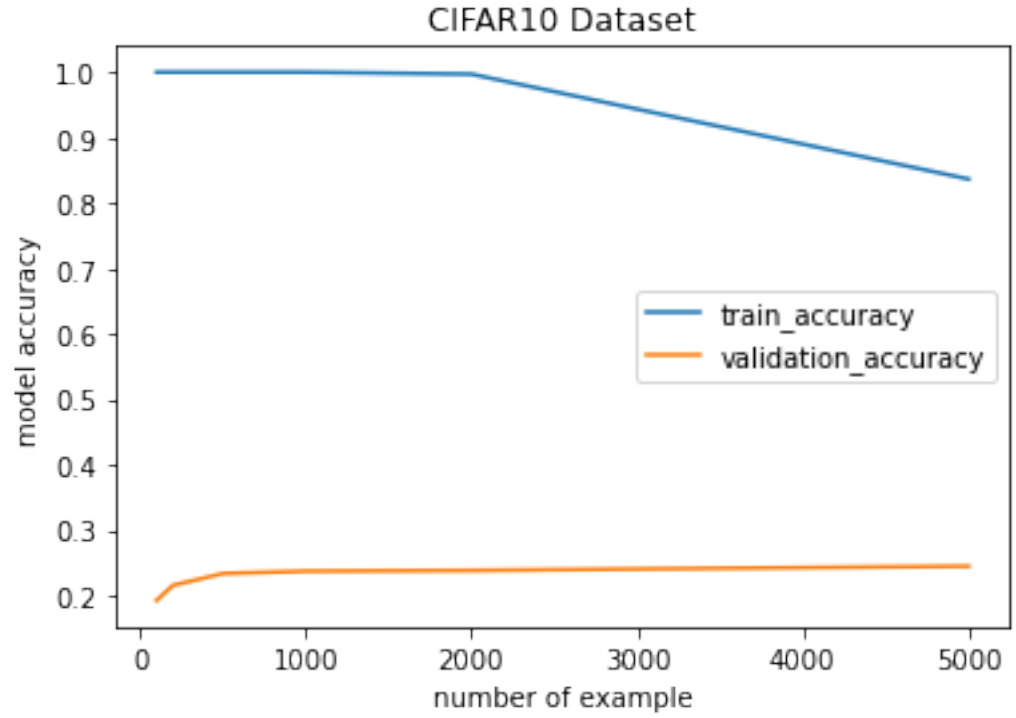
```
    #clf = svm.LinearSVC(C=c)
    clf = svm.LinearSVC(C=c)
    clf.fit(X, Y)
    return clf

def num_examples_experiment(X_train, Y_train, X_val, Y_val, num_examples_arr,␣
↪name):
    # train an svm for each number of examples.
    # Evaluate the training and validation performance
    # plot the results.
    train_acc = []
    val_acc = []
    for example in num_examples_arr:
        indices = np.arange(X_train.shape[0])
        np.random.shuffle(indices)
        X = X_train[indices]
        Y = Y_train[indices]
        model = train(X[:example], Y[:example])
        train_acc.append(accuracy_score(Y[:example], model.predict(X[:
↪example])))
        val_acc.append(accuracy_score(Y_val, model.predict(X_val)))
    plt.figure()
    plt.plot(num_examples_arr, train_acc, label = 'train_accuracy')
    plt.plot(num_examples_arr, val_acc, label = 'validation_accuracy')
    plt.xlabel('number of example')
    plt.ylabel('model accuracy')
    plt.title(name)
    plt.legend()
    plt.show()
    return

def main_q2():
    # Run all of the code for question 3.
    MNIST_NUM_EXAMPLES_ARR = [100, 200, 500, 1000, 2000, 5000, 10000]
    SPAM_NUM_EXAMPLES_ARR = [100, 200, 500, 1000, 2000,␣
↪spam_data['training_data'].shape[0]] # Figure out the number of examples in␣
↪the dataset or handle this case.
    CIFAR_NUM_EXAMPLES_ARR = [100, 200, 500, 1000, 2000, 5000]

    # YOUR CODE HERE
    all_examples = [MNIST_NUM_EXAMPLES_ARR, SPAM_NUM_EXAMPLES_ARR,␣
↪CIFAR_NUM_EXAMPLES_ARR]
    all_names = ['MNIST Dataset','Spam Dataset', 'CIFAR10 Dataset']
    for i in range(3):
        dataset = all_data[i]
        data = partition(dataset['training_data'], dataset['training_labels'],␣
↪validation_sizes[i])
```

```
        num_examples_experiment(data[0], data[1], data[2], data[3],␣
↪all_examples[i], all_names[i])
    return
```

[30]:
```
main_q2()
```




Spam Dataset
train_accuracy
validation_accuracy
model accuracy
0.84
0.83
0.82
0.81
0.80
0.79
0.78
0
1000
2000
3000
4000
5000
number of example

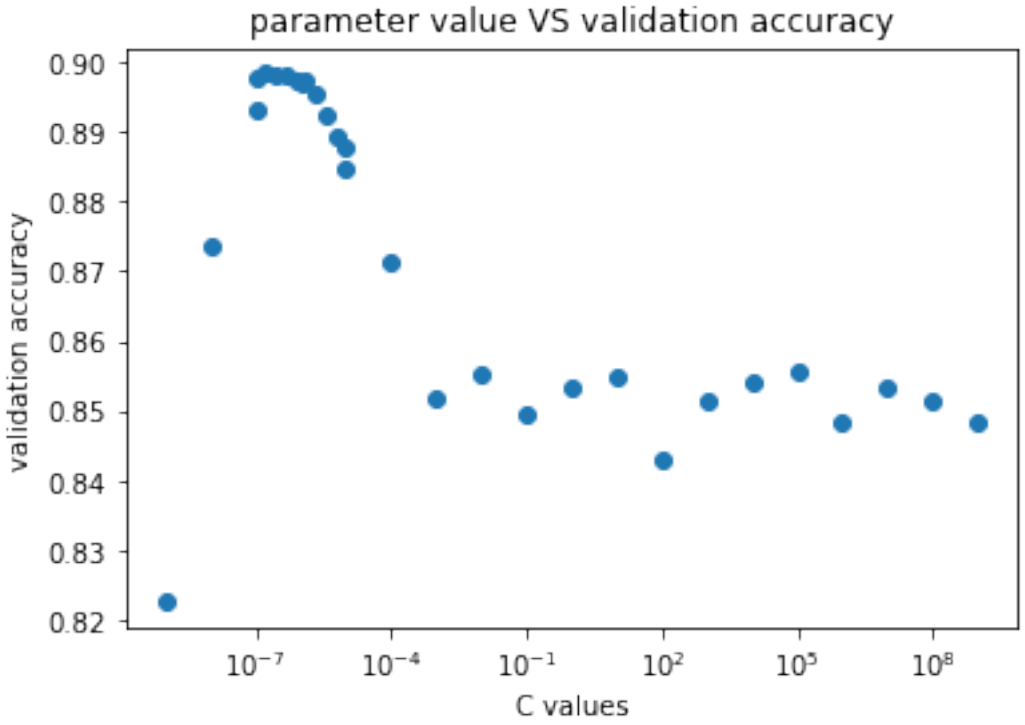

CIFAR10 Dataset
train_accuracy
validation_accuracy
model accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
1000
2000
3000
4000
5000
number of example

[7]:
```
#################################
# Question 3: Hyperparameters   #
#################################

def hyperparameter_search(X_train, Y_train, X_val, Y_val, parameter_values):
    val_acc = []
    for param in parameter_values:
        model = train(X_train, Y_train, param)
        val_acc.append(accuracy_score(Y_val, model.predict(X_val)))

    return val_acc

def main_q3():
    #c_values = [0.0001, 0.1, 0.5, 1.0, 2.0, 3.0, 5.0, 7.5, 10.0, 20.0, 50.0,␣
 ↪100]
    c_values = np.logspace(-9, 9, num=19)
    dataset = mnist_data
    size = 10000
    data = partition(dataset['training_data'], dataset['training_labels'],␣
 ↪mnist_val_size)
    val_acc_1 = hyperparameter_search(data[0][:size], data[1][:size], data[2],␣
 ↪data[3], c_values)
    c_val = c_values[np.argmax(val_acc_1)]

    new_c_values = np.logspace(np.log10(c_val)-1, np.log10(c_val)+1, num=10)
    data = partition(dataset['training_data'], dataset['training_labels'],␣
 ↪mnist_val_size)
    val_acc_2 = hyperparameter_search(data[0][:size], data[1][:size], data[2],␣
 ↪data[3], new_c_values)
    c_val = new_c_values[np.argmax(val_acc_2)]

    val_acc = np.append(val_acc_1, val_acc_2)
    params = np.append(c_values, new_c_values)

    print(params)
    print(val_acc)

    plt.figure()
    plt.xscale('log')
    plt.scatter(params, val_acc)
    plt.xlabel('C values')
    plt.ylabel('validation accuracy')
    plt.title('parameter value VS validation accuracy')
    plt.show()
    print('highest c value is: ' + str(c_val) + ',with accuracy: ' +␣
 ↪str(max(val_acc)))
```

```
    return
```

[8]:
```
main_q3()
```

```
[1.00000000e-09 1.00000000e-08 1.00000000e-07 1.00000000e-06
 1.00000000e-05 1.00000000e-04 1.00000000e-03 1.00000000e-02
 1.00000000e-01 1.00000000e+00 1.00000000e+01 1.00000000e+02
 1.00000000e+03 1.00000000e+04 1.00000000e+05 1.00000000e+06
 1.00000000e+07 1.00000000e+08 1.00000000e+09 1.00000000e-07
 1.66810054e-07 2.78255940e-07 4.64158883e-07 7.74263683e-07
 1.29154967e-06 2.15443469e-06 3.59381366e-06 5.99484250e-06
 1.00000000e-05]
[0.8227 0.8736 0.8931 0.8969 0.8878 0.8712 0.852  0.8553 0.8495 0.8532
 0.8547 0.8432 0.8515 0.8542 0.8556 0.8482 0.8533 0.8516 0.8484 0.8977
 0.8983 0.8982 0.8979 0.8975 0.8974 0.8956 0.8922 0.8892 0.8848]
```



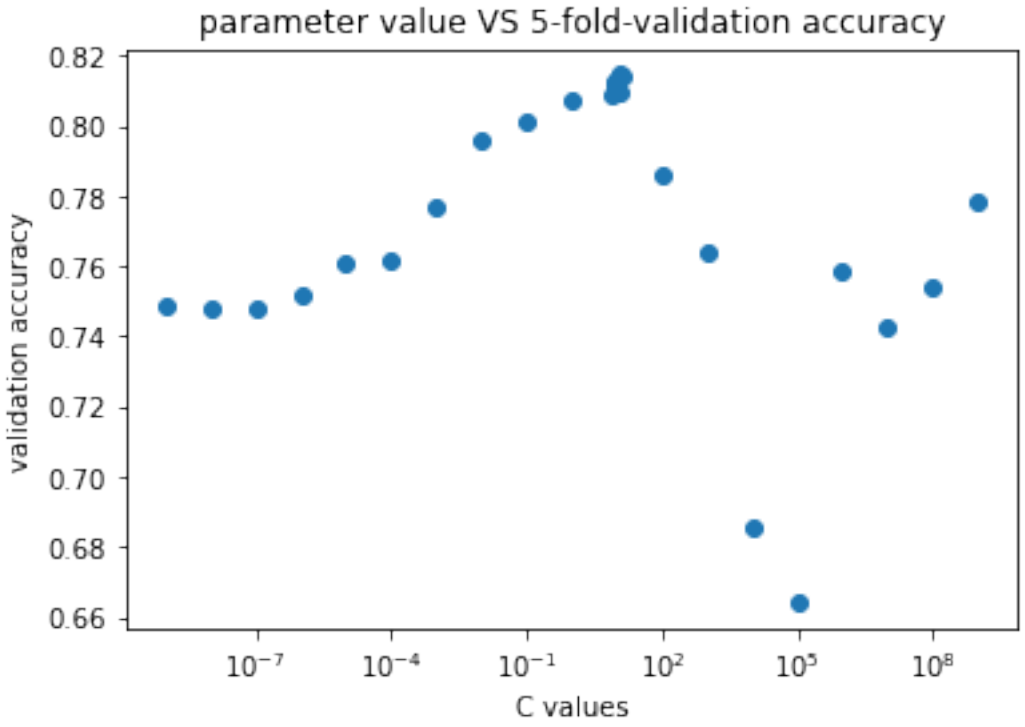

```
highest c value is: 1.668100537200059e-07,with accuracy: 0.8983
```

[9]:
```
###################################
# Question 4: KFold CrossValid  #
###################################

def k_fold_cross_validation(X_train, Y_train, k, parameter_values):
```

```
    val_acc = []
    for param in parameter_values:
        indices = np.arange(X_train.shape[0])
        np.random.shuffle(indices)
        x_shuffle = X_train[indices]
        y_shuffle = Y_train[indices]
        k_acc = []
        for i in range(k):
            start = int(np.floor(i*len(X_train)/k))
            end = int(np.floor((i+1)*len(X_train)/k))
            train_x = np.append(x_shuffle[:start], x_shuffle[end:], axis=0)
            train_y = np.append(y_shuffle[:start], y_shuffle[end:], axis=0)
            val_x = x_shuffle[start:end]
            val_y = y_shuffle[start:end]
            model = train(train_x, train_y, param)
            k_acc.append(accuracy_score(val_y, model.predict(val_x)))
        val_acc.append(np.mean(k_acc))

    return val_acc

def main_q4():
    c_values = np.logspace(-9, 9, num=19)
    dataset = spam_data
    val_acc_1 = k_fold_cross_validation(dataset['training_data'],␣
 ↪dataset['training_labels'], 5, c_values)
    c_val = c_values[np.argmax(val_acc_1)]

    new_c_values = np.logspace(np.log10(c_val)-0.1, np.log10(c_val)+0.1, num=10)
    val_acc_2 = k_fold_cross_validation(dataset['training_data'],␣
 ↪dataset['training_labels'], 5, new_c_values)
    c_val = new_c_values[np.argmax(val_acc_2)]

    val_acc = np.append(val_acc_1, val_acc_2)
    params = np.append(c_values, new_c_values)

    print(params)
    print(val_acc)
    plt.figure()
    plt.xscale('log')
    plt.scatter(params, val_acc)
    plt.xlabel('C values')
    plt.ylabel('validation accuracy')
    plt.title('parameter value VS 5-fold-validation accuracy')
    plt.show()
    print('highest c value is: ' + str(c_val) + ',with 5-fold-accuracy: ' +␣
 ↪str(max(val_acc)))
    return
```

[10]:
```
main_q4()
```

```
[1.00000000e-09 1.00000000e-08 1.00000000e-07 1.00000000e-06
 1.00000000e-05 1.00000000e-04 1.00000000e-03 1.00000000e-02
 1.00000000e-01 1.00000000e+00 1.00000000e+01 1.00000000e+02
 1.00000000e+03 1.00000000e+04 1.00000000e+05 1.00000000e+06
 1.00000000e+07 1.00000000e+08 1.00000000e+09 7.94328235e+00
 8.36030694e+00 8.79922544e+00 9.26118728e+00 9.74740226e+00
 1.02591437e+01 1.07977516e+01 1.13646367e+01 1.19612833e+01
 1.25892541e+01]
[0.74883936 0.74806436 0.74806586 0.7517467  0.7608264  0.76198918
 0.77668022 0.79582485 0.80143059 0.80704174 0.81225614 0.78597165
 0.76429176 0.68538372 0.66415253 0.75871107 0.74225175 0.75426438
 0.77862548 0.80916361 0.80955438 0.81206253 0.81225764 0.81187042
 0.81129164 0.81360992 0.81457648 0.80954373 0.8141872 ]
```




```
highest c value is: 11.364636663857247,with 5-fold-accuracy: 0.8145764770741643
```

[14]:
```
###################################
# Question 5: Kaggle              #
###################################

# For kaggle, do whatever you like!
```

```
def kaggle_train(X, Y, c = 1.0, kernel_name='linear'):
    # create, train, and return and SVM Model
    #clf = svm.LinearSVC(C=c)
    print("here we train")
    if kernel_name == 'linear':
        clf = svm.LinearSVC(C=c)
    else:
        clf = svm.SVC(kernel=kernel_name, C=c)
    clf.fit(X, Y)
    return clf

def no_k_fold(X_train, Y_train, parameter_values, kernel_name, size):
    data = partition(X_train, Y_train, size)
    val_acc = []
    for param in parameter_values:
        model = kaggle_train(data[0][:10000], data[1][:10000], param,␣
 ↪kernel_name)
        val_acc.append(accuracy_score(data[3], model.predict(data[2])))
    return val_acc

def k_fold_param_tuning_with_kernel(X_train, Y_train, k, parameter_values,␣
 ↪kernel_name):
    val_acc = []
    for param in parameter_values:
        indices = np.arange(X_train.shape[0])
        np.random.shuffle(indices)
        x_shuffle = X_train[indices]
        y_shuffle = Y_train[indices]
        k_acc = []
        for i in range(k):
            start = int(np.floor(i*len(X_train)/k))
            end = int(np.floor((i+1)*len(X_train)/k))
            train_x = np.append(x_shuffle[:start], x_shuffle[end:], axis=0)
            train_y = np.append(y_shuffle[:start], y_shuffle[end:], axis=0)
            val_x = x_shuffle[start:end]
            val_y = y_shuffle[start:end]
            model = kaggle_train(train_x, train_y, param, kernel_name)
            k_acc.append(accuracy_score(val_y, model.predict(val_x)))
        val_acc.append(np.mean(k_acc))

    return val_acc

def kaggle_predict(dataset, kernel_name, k_fold_or_not = False, size=None):
    c_values = np.logspace(-2, 3, num=6)
    if k_fold_or_not:
        val_acc_1 = k_fold_param_tuning_with_kernel(dataset['training_data'],␣
 ↪dataset['training_labels'], 5, c_values, kernel_name)
```

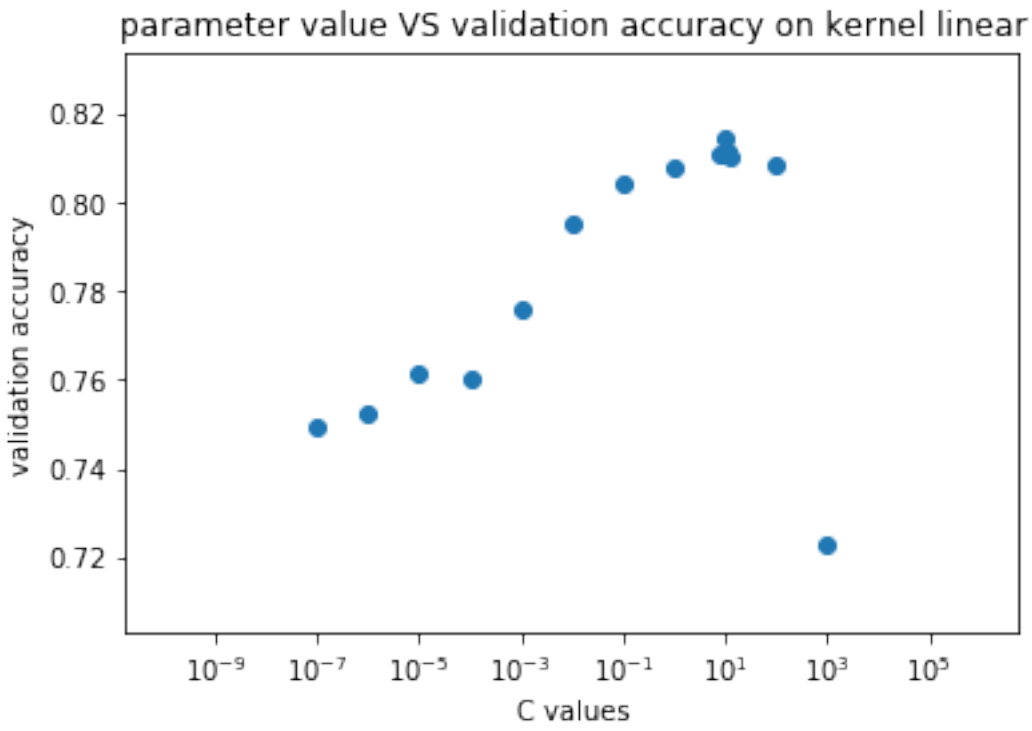
```
    else:
        val_acc_1 = no_k_fold(dataset['training_data'],␣
↪dataset['training_labels'], c_values, kernel_name, size)
    c_val = c_values[np.argmax(val_acc_1)]

    new_c_values = np.logspace(np.log10(c_val)-0.2, np.log10(c_val)+0.2, num=9)
    if k_fold_or_not:
        val_acc_2 = k_fold_param_tuning_with_kernel(dataset['training_data'],␣
↪dataset['training_labels'], 5, new_c_values, kernel_name)
    else:
        val_acc_2 = no_k_fold(dataset['training_data'],␣
↪dataset['training_labels'], new_c_values, kernel_name, size)
    c_val = new_c_values[np.argmax(val_acc_2)]

    val_acc = np.append(val_acc_1, val_acc_2)
    params = np.append(c_values, new_c_values)

    plt.figure()
    plt.xscale('log')
    plt.scatter(params, val_acc)
    plt.xlabel('C values')
    plt.ylabel('validation accuracy')
    plt.title('parameter value VS validation accuracy on kernel '+kernel_name)
    plt.show()
    print('highest c value is: ' + str(c_val) + ',with 5-fold-accuracy: ' +␣
↪str(max(val_acc)))
    final_model = kaggle_train(dataset['training_data'],␣
↪dataset['training_labels'], c_val, kernel_name)
    return final_model.predict(dataset['test_data'])

def q5_kaggle(name, kernel):
    sizes = {'mnist': 5000, 'spam':round(spam_data['training_data'].shape[0]*0.
↪2), 'cifar':3000}
    size = sizes[name]
    if name == 'mnist':
        print('working on MNIST dataset...')
        prediction = kaggle_predict(mnist_data, kernel, False, size)
        print('finish MNIST dataset')
    elif name == 'spam':
        print('working on SPAM dataset...')
        prediction = kaggle_predict(spam_data, kernel,True)
        print('finish spam dataset')
    else:
        print('working on CIFAR dataset...')
        prediction = kaggle_predict(cifar_data, kernel, False, size)
        print('finish CIFAR dataset')
```

```
    return prediction
```

[9]:
```
spam_prediction = q5_kaggle('spam', 'linear')
```

```
working on SPAM dataset…
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```

here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train



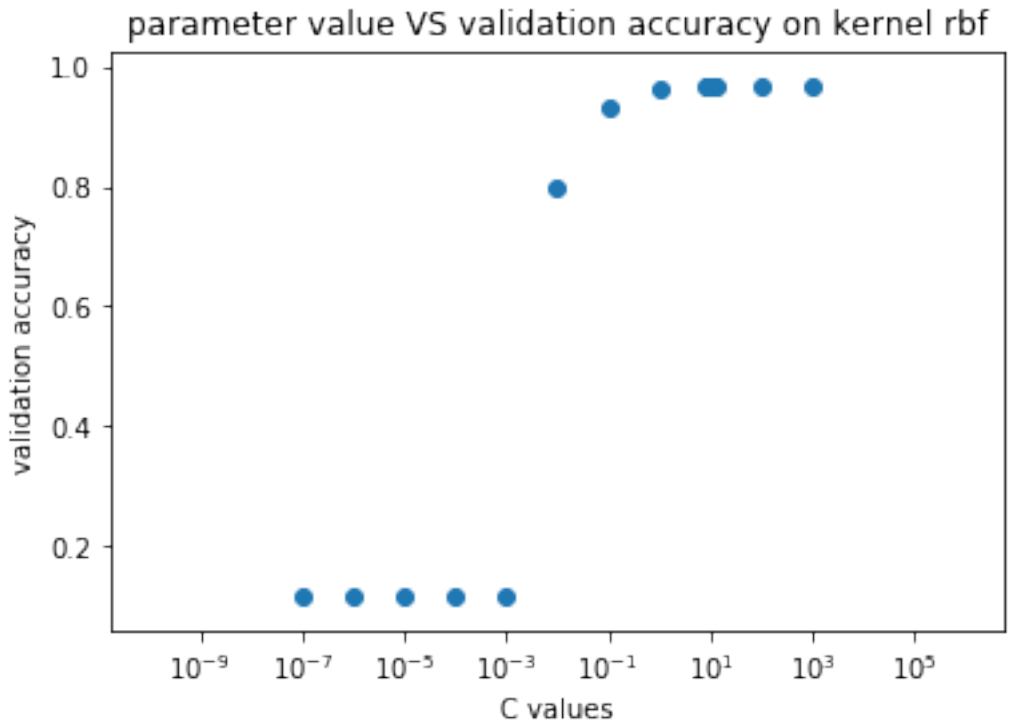

```
highest c value is: 9.261187281287935,with 5-fold-accuracy: 0.8141928068847587
here we train
finish spam dataset
```

[14]:
```
import pandas as pd
import numpy as np

# Usage results_to_csv(clf.predict(X_test))
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission.csv', index_label='Id')

#results_to_csv(spam_prediction)
```

[11]:
```
mnist_prediction = q5_kaggle('mnist', 'rbf')
```

```
working on MNIST dataset…
here we train
here we train
here we train
here we train
here we train
```

```
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```



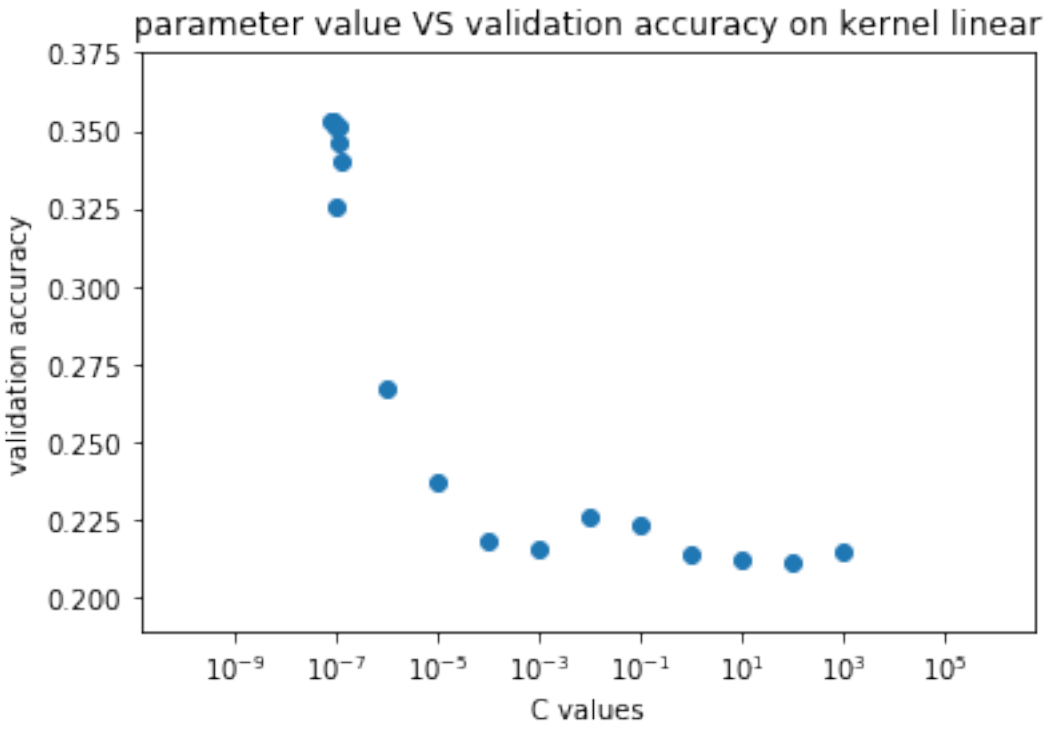

```
highest c value is: 7.943282347242816,with 5-fold-accuracy: 0.9692
here we train
finish MNIST dataset
```

[12]:
```
results_to_csv(mnist_prediction)
```

[6]:
```
cifar_prediction = q5_kaggle('cifar', 'linear')
```

```
working on CIFAR dataset…
here we train
```

```
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```

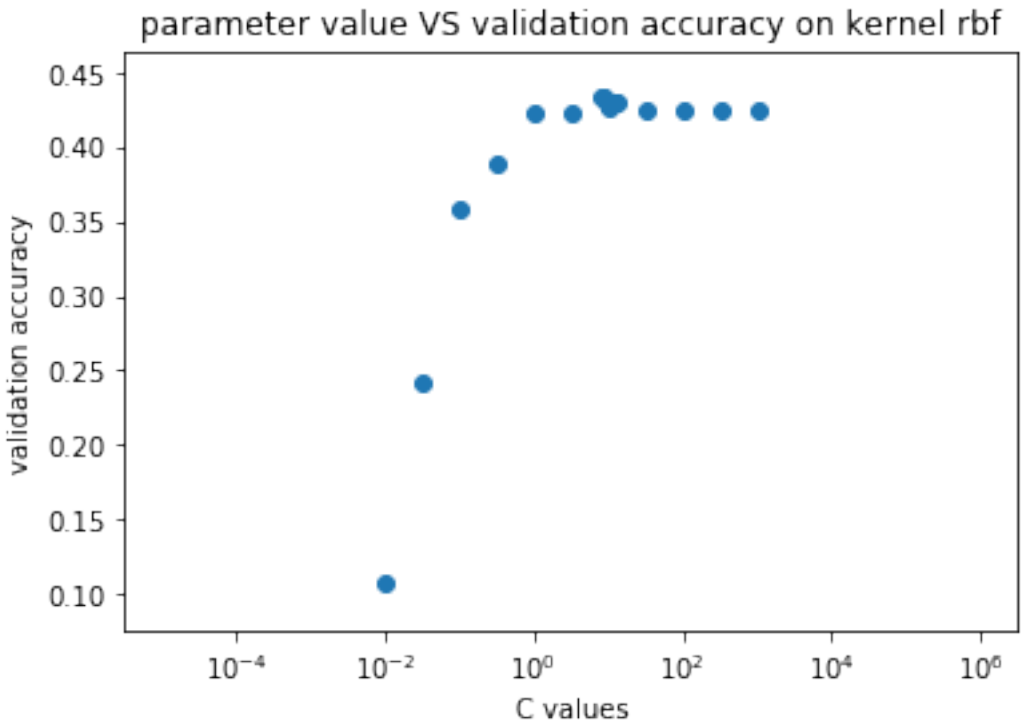



```
highest c value is: 7.943282347242822e-08,with 5-fold-accuracy: 0.3535
here we train
finish CIFAR dataset
```

[9]:
```
results_to_csv(cifar_prediction)
```

[8]:
```
cifar_prediction = q5_kaggle('cifar', 'rbf')
```

```
working on CIFAR dataset…
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```



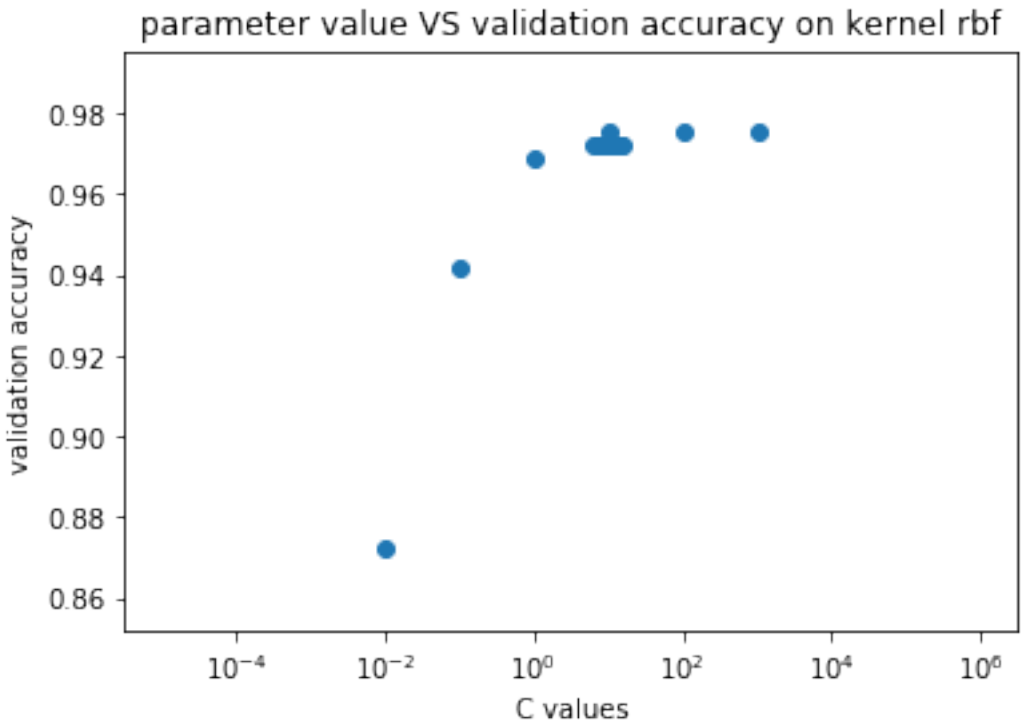

```
highest c value is: 8.576958985908941,with 5-fold-accuracy: 0.4335
here we train
```

```
finish CIFAR dataset
```

[10]:
```
results_to_csv(cifar_prediction)
```

[12]:
```
mnist_prediction = q5_kaggle('mnist', 'rbf')
```

```
working on MNIST dataset…
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```



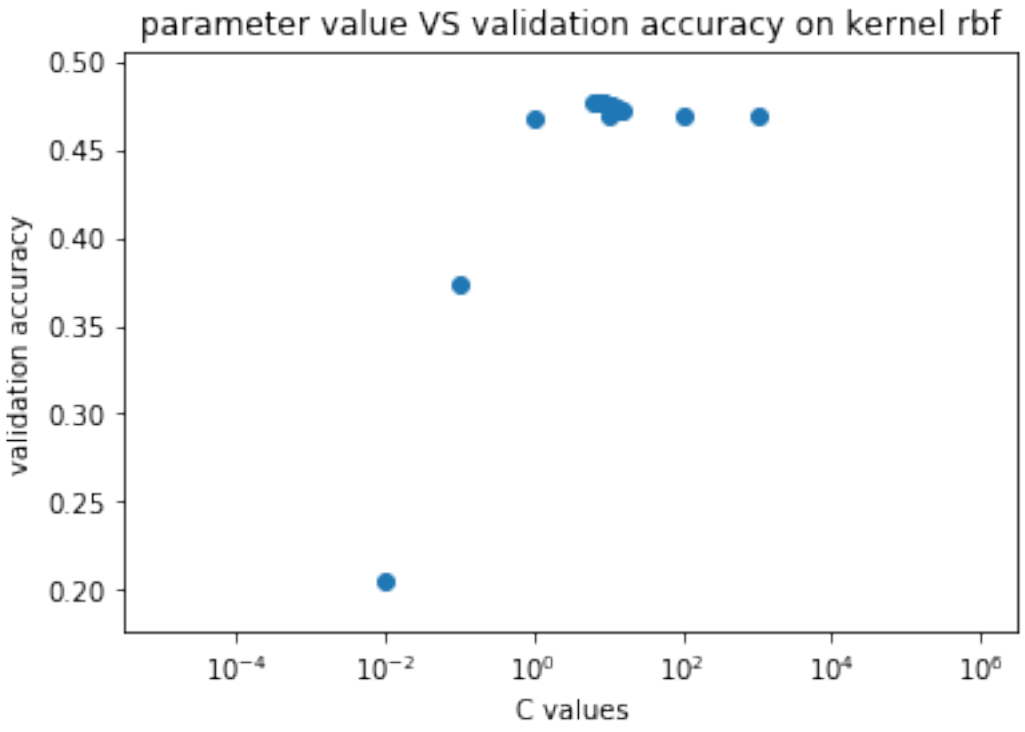

```
highest c value is: 8.912509381337454,with 5-fold-accuracy: 0.9752
```

```
here we train
finish MNIST dataset
```

[13]:
```
results_to_csv(mnist_prediction)
```

[15]:
```
cifar_prediction = q5_kaggle('cifar', 'rbf')
```

```
working on CIFAR dataset…
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```



```
highest c value is: 7.07945784384138,with 5-fold-accuracy: 0.4776666666666667
here we train
finish CIFAR dataset
```

[16]:
```
results_to_csv(cifar_prediction)
```

[21]:
```
from skimage.feature import hog

def mnist_hogging(image):
    old_image = np.reshape(image, (28,28))
    fd, new_image = hog(old_image, orientations=8, pixels_per_cell=(2, 2),␣
 ↪cells_per_block=(1, 1),
                        visualize=True, feature_vector=True)
    return fd.tolist()

def cifar_hogging(image):
    test_image = np.reshape(image, (3, 1024))
    test_image = [np.reshape(i, (32,32)).tolist() for i in test_image]
    image = np.dstack(test_image)
    fd= hog(image, orientations=8, pixels_per_cell=(2,2),
                    cells_per_block=(1, 1), feature_vector=True,␣
 ↪multichannel=True)
    return fd.tolist()

def kaggle_train(X, Y, c = 1.0, kernel_name='linear'):
    # create, train, and return and SVM Model
    #clf = svm.LinearSVC(C=c)
    print("here we train")
    if kernel_name == 'linear':
        clf = svm.LinearSVC(C=c)
    else:
        clf = svm.SVC(kernel=kernel_name, C=c)
    clf.fit(X, Y)
    return clf

def partition(data, labels, validation_size):
    # output: training_data, training_label, validation_data, validation_label
    if data.shape[0] != labels.shape[0]:
        raise Exception("data size mismatch with label size!")
    indices = np.arange(data.shape[0])
    np.random.shuffle(indices)
    data_shuffle = data[indices]
    labels_shuffle = labels[indices]
    return data_shuffle[validation_size:], labels_shuffle[validation_size:],␣
 ↪data_shuffle[:validation_size], labels_shuffle[:validation_size]

def no_k_fold(X_train, Y_train, parameter_values, kernel_name, size):
```

```
    data = partition(X_train, Y_train, size)
    val_acc = []
    for param in parameter_values:
        model = kaggle_train(data[0][:12000], data[1][:12000], param,␣
↪kernel_name)
        val_acc.append(accuracy_score(data[3], model.predict(data[2])))
    return val_acc

def kaggle_predict(dataset, kernel_name, k_fold_or_not = False, size=None):
    c_values = np.logspace(-2, 3, num=6)
    if k_fold_or_not:
        val_acc_1 = k_fold_param_tuning_with_kernel(dataset['training_data'],␣
↪dataset['training_labels'], 5, c_values, kernel_name)
    else:
        val_acc_1 = no_k_fold(dataset['training_data'],␣
↪dataset['training_labels'], c_values, kernel_name, size)
    c_val = c_values[np.argmax(val_acc_1)]

    new_c_values = np.logspace(np.log10(c_val)-0.3, np.log10(c_val)+0.3, num=11)
    if k_fold_or_not:
        val_acc_2 = k_fold_param_tuning_with_kernel(dataset['training_data'],␣
↪dataset['training_labels'], 5, new_c_values, kernel_name)
    else:
        val_acc_2 = no_k_fold(dataset['training_data'],␣
↪dataset['training_labels'], new_c_values, kernel_name, size)
    c_val = new_c_values[np.argmax(val_acc_2)]

    val_acc = np.append(val_acc_1, val_acc_2)
    params = np.append(c_values, new_c_values)

    plt.figure()
    plt.xscale('log')
    plt.scatter(params, val_acc)
    plt.xlabel('C values')
    plt.ylabel('validation accuracy')
    plt.title('parameter value VS validation accuracy on kernel '+kernel_name)
    plt.show()
    print('zoomed c ranges'+str(new_c_values))
    print('highest c value is: ' + str(c_val) + ',with 5-fold-accuracy: ' +␣
↪str(max(val_acc)))
    final_model = kaggle_train(dataset['training_data'],␣
↪dataset['training_labels'], c_val, kernel_name)
    return final_model.predict(dataset['test_data'])
    #return None

def q5_kaggle(name, kernel):
```

```
    sizes = {'mnist': 4000, 'spam':round(spam_data['training_data'].shape[0]*0.
↪2), 'cifar':3000}
    size = sizes[name]
    if name == 'mnist':
        print('working on MNIST dataset...')
        prediction = kaggle_predict(hog_mnist_data, kernel, False, size)
        print('finish MNIST dataset')
    elif name == 'spam':
        print('working on SPAM dataset...')
        prediction = kaggle_predict(spam_data, kernel,True)
        print('finish spam dataset')
    else:
        print('working on CIFAR dataset...')
        prediction = kaggle_predict(hog_cifar_data, kernel, False, size)
        print('finish CIFAR dataset')
    return prediction
```

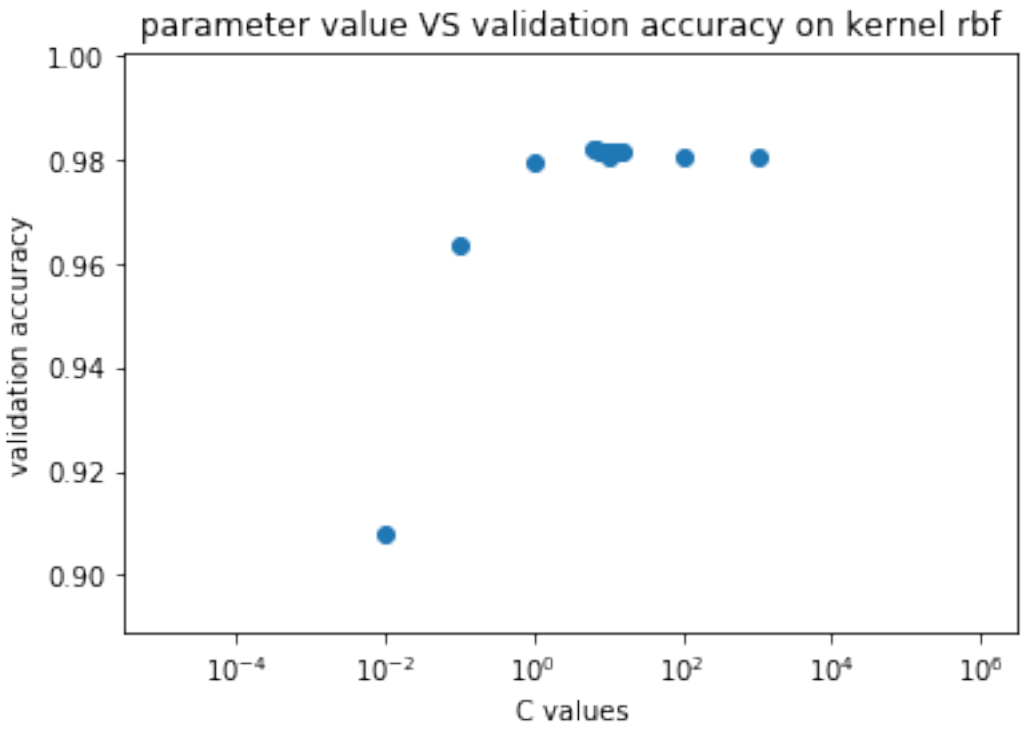
[18]:
```
hog_mnist_data_train = np.array([mnist_hogging(i) for i in␣
↪mnist_data['training_data']])
```

[19]:
```
hog_mnist_data_test = np.array([mnist_hogging(i) for i in␣
↪mnist_data['test_data']])
```

[20]:
```
hog_mnist_data = {'training_data':hog_mnist_data_train, 'training_labels':
↪mnist_data['training_labels'], 'test_data':hog_mnist_data_test}
```

[33]:
```
mnist_prediction = q5_kaggle('mnist', 'rbf')
```

```
working on MNIST dataset…
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```




```
highest c value is: 6.309573444801933,with 5-fold-accuracy: 0.982
here we train
finish MNIST dataset
```

[34]:
```
results_to_csv(mnist_prediction)
```

[35]:
```
len(mnist_prediction)
```

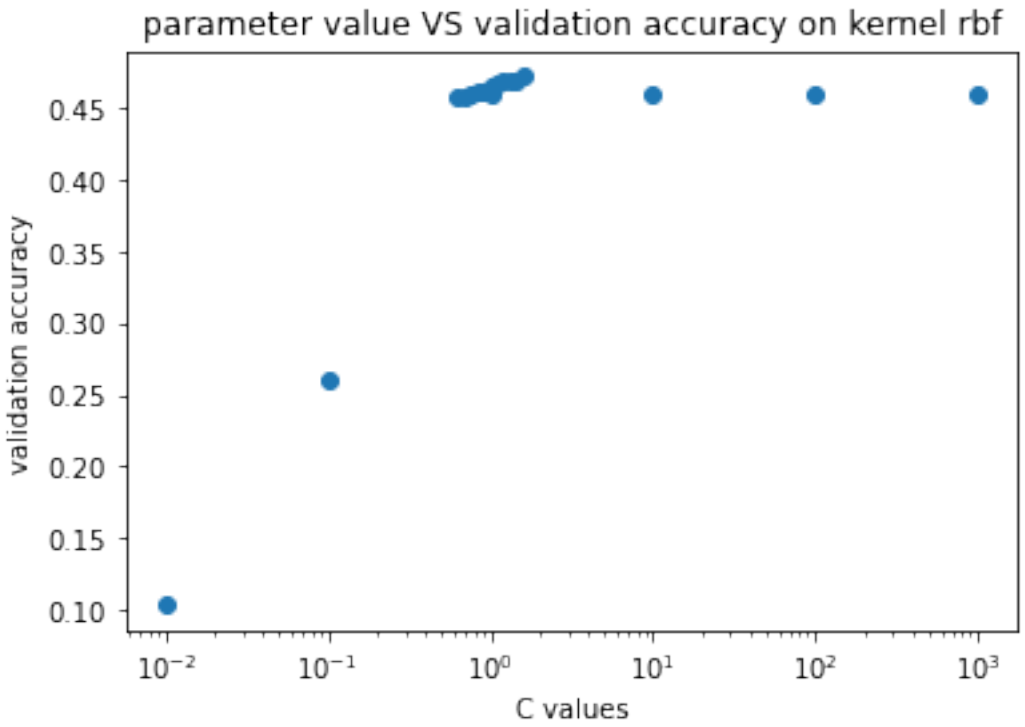
[35]:
```
10000
```

[6]:
```
hog_cifar_data_train = np.array([cifar_hogging(i) for i in␣
 ↪cifar_data['training_data']])
hog_cifar_data_test = np.array([cifar_hogging(i) for i in␣
 ↪cifar_data['test_data']])
hog_cifar_data = {'training_data':hog_cifar_data_train, 'training_labels':
 ↪cifar_data['training_labels'], 'test_data':hog_cifar_data_test}
```

[7]:
```
cifar_prediction = q5_kaggle('cifar', 'rbf')
```

```
working on CIFAR dataset…
here we train
here we train
here we train
here we train
```

```
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```

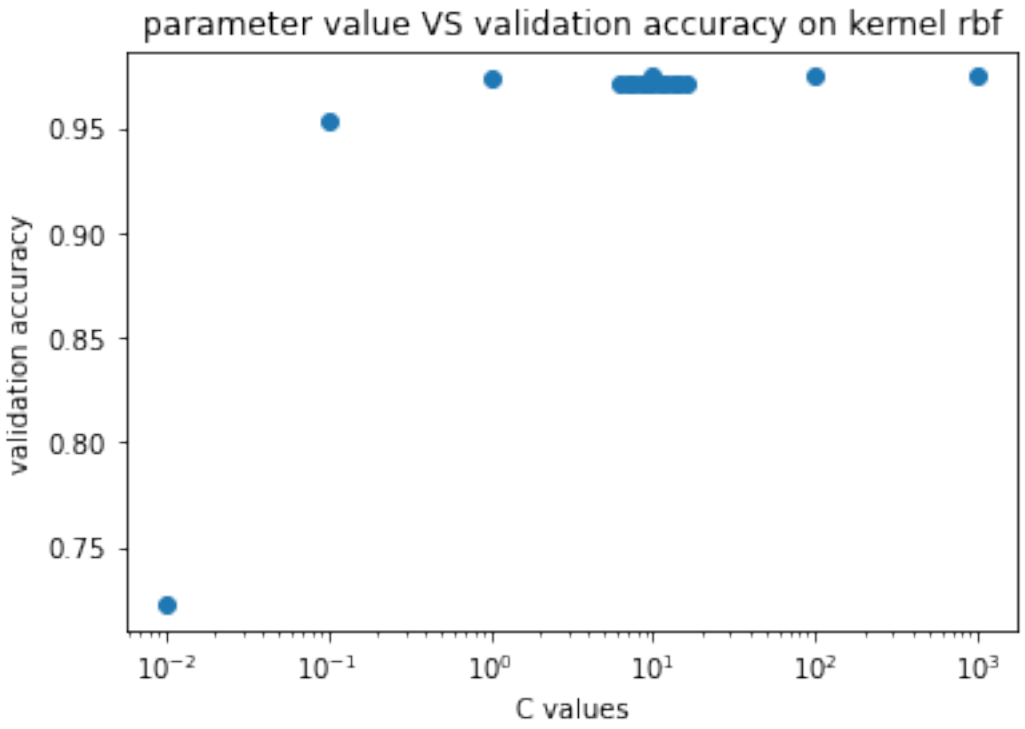



```
zoomed c ranges[0.63095734 0.69183097 0.75857758 0.83176377 0.91201084 1.
 1.0964782  1.20226443 1.31825674 1.44543977 1.58489319]
highest c value is: 1.5848931924611136,with 5-fold-accuracy: 0.472
here we train
finish CIFAR dataset
```

[12]:
```
mnist_prediction = q5_kaggle('mnist', 'rbf')
```

```
working on MNIST dataset…
here we train
```

```
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```

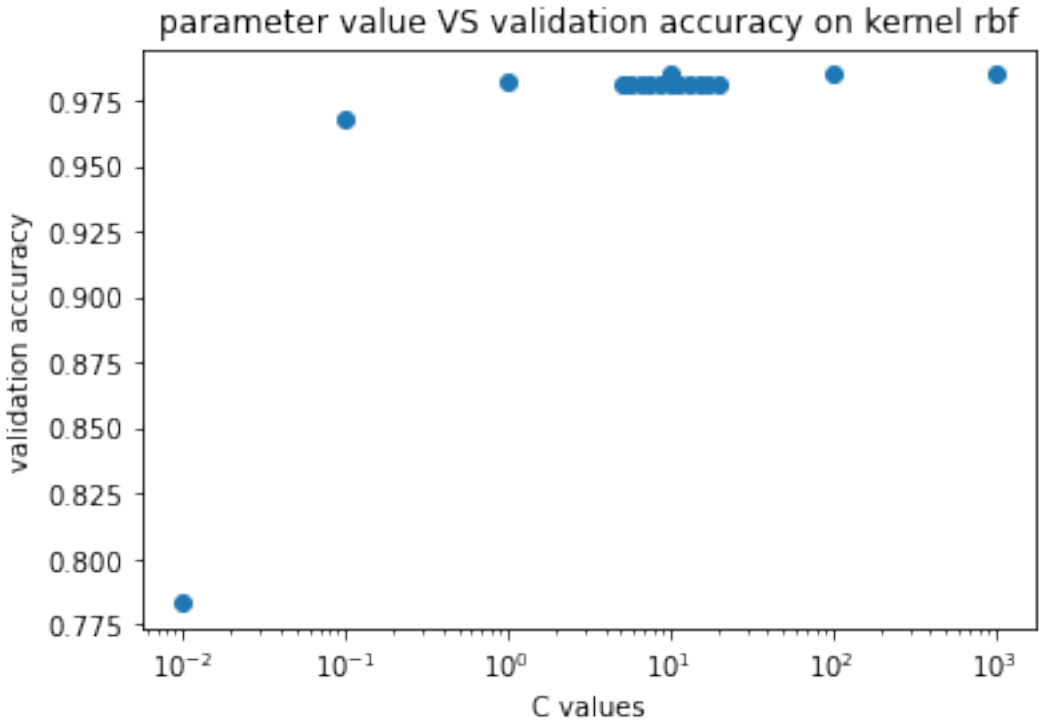



```
zoomed c ranges[ 6.30957344  6.91830971  7.58577575  8.31763771  9.12010839 10.
 10.96478196 12.02264435 13.18256739 14.45439771 15.84893192]
highest c value is: 6.309573444801933,with 5-fold-accuracy: 0.9745
here we train
finish MNIST dataset
```

[15]:
```
results_to_csv(mnist_prediction)
```

[22]:
```
mnist_prediction = q5_kaggle('mnist', 'rbf')
```

```
working on MNIST dataset…
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
here we train
```




```
zoomed c ranges[ 5.01187234  5.75439937  6.60693448  7.58577575  8.7096359  10.
 11.48153621 13.18256739 15.13561248 17.37800829 19.95262315]
highest c value is: 5.011872336272722,with 5-fold-accuracy: 0.98475
```

```
here we train
finish MNIST dataset
```

[23]:
```
results_to_csv(mnist_prediction)
```

[ ]:
```

```