

Student Name: Leo-Adventure  
Student ID: 666



## Introduction to Machine Learning(CS189)

### Homework 1 Write-Up

---

#### 1. Honor Code

I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted.

Signature : Leo-Adventure

## 2. Data Partitioning

The code for Data Partitioning is shown as below.

```
1
2 import numpy as np
3 from sklearn.svm import SVC
4 from sklearn import metrics
5 import matplotlib.pyplot as plt
6
7 if __name__ == "__main__":
8     # For the MNIST dataset, write code that sets aside 10,000 training images as a
9         validation set.
10
11     mnist_data = np.load("../data/mnist-data.npz")
12     mnist_train = mnist_data["training_data"]
13     mnist_train_label = mnist_data["training_labels"]
14     mnist_state = np.random.get_state()
15     np.random.shuffle(mnist_train)
16     np.random.set_state(mnist_state)
17     np.random.shuffle(mnist_train_label)
18     mnist_validation = mnist_train[:9999]
19     mnist_validation_label = mnist_train_label[:9999]
20     mnist_train = mnist_train[10000:]
21     mnist_train_label = mnist_train_label[10000:]
22     # For the spam dataset, write code that sets aside 20% of the training data as
23         a validation set.
24
25     spam_data = np.load("../data/spam-data.npz")
26     spam_train = spam_data["training_data"]
27     spam_train_label = spam_data["training_labels"]
28     state = np.random.get_state()
29     np.random.shuffle(spam_train)
30     np.random.set_state(state)
31     np.random.shuffle(spam_train_label)
32     len_spam_train = int(len(spam_train)*0.8)
33     len_spam_validation = len(spam_train) - len_spam_train
34     spam_validation = spam_train[:len_spam_validation - 1]
35     spam_validation_label = spam_train_label[:len_spam_validation-1]
36     spam_train = spam_train[len_spam_validation:]
37     spam_train_label = spam_train_label[len_spam_validation:]
38     # For the CIFAR-10 dataset, write code that sets aside 5,000 training images as
39         a validation set.
40
41     cifar_data = np.load("../data/cifar10-data.npz")
42     cifar_train = cifar_data["training_data"]
43     cifar_train_label = cifar_data["training_labels"]
44     state = np.random.get_state()
45     np.random.shuffle(cifar_train)
46     np.random.set_state(state)
47     np.random.shuffle(cifar_train_label)
48
49     cifar_validation = cifar_train[:4999]
50     cifar_validation_label = cifar_train_label[:4999]
51
52     cifar_train = cifar_train[5000:]
53     cifar_train_label = cifar_train_label[5000:]
```

3. Support Vector Machines: Coding

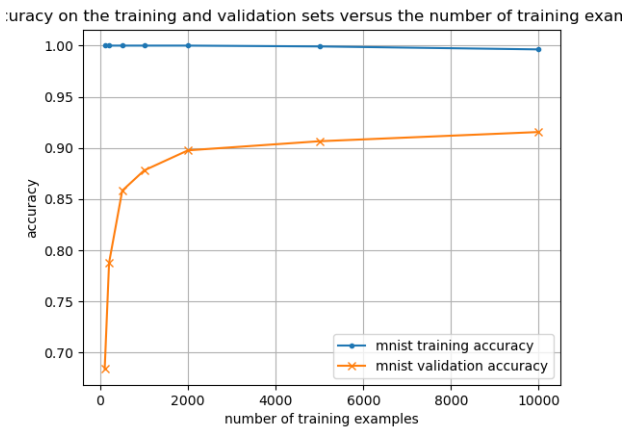


Figure 1: The SVM model training result of mnist dataset

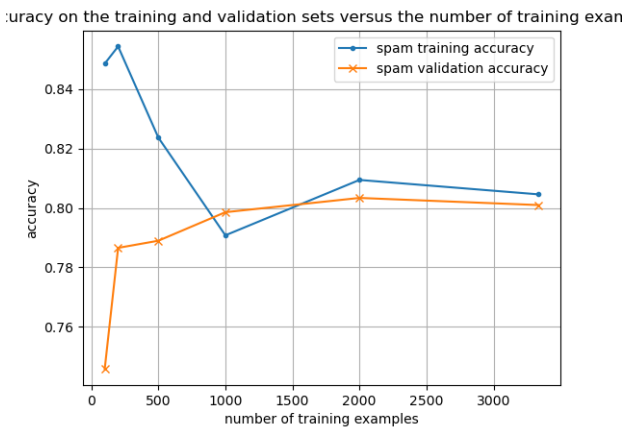


Figure 2: The SVM model training result of spam dataset

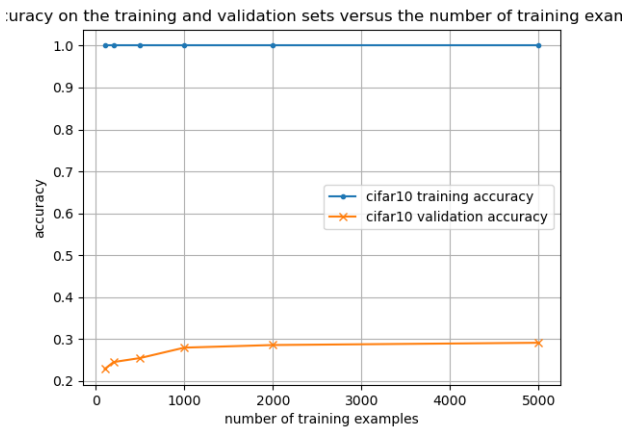


Figure 3: The SVM model training result of CIFAR-10 dataset

#### 4. Hyperparameter Tuning

The relationship between the C value we chose and the accuracy is shown in table 4

C value	5.00E-05	1.00E-04	1.50E-04	2.00E-04	2.50E-04	3.00E-04	3.50E-04	4.00E-04	4.50E-04	5.00E-04
Accuracy	0.574	0.744	0.817	0.842	0.854	0.861	0.866	0.871	0.874	0.876

The C value I chose and the result can be plotted as figure 4

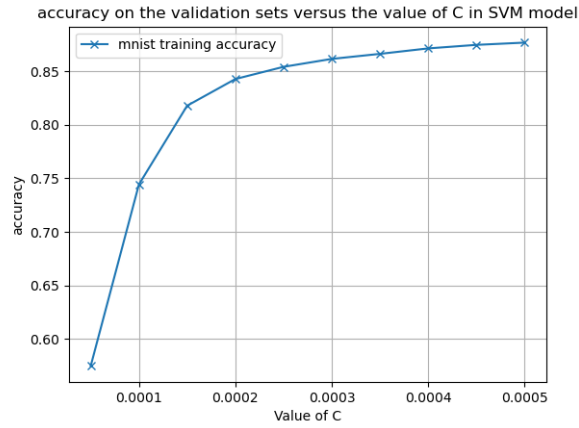


Figure 4: The accuracy score of mnist dataset based on different C

The best C here is 5e-4.

## 5. K-Fold Cross Validation

The C value I chose and the corresponding accuracy can be plotted as table5

C value	1	2	3	4	5	6	7	8
Accuracy	0.7951	0.7983	0.7985	0.7975	0.7971	0.7978	0.798	0.7978

The C value I chose and the result can be plotted as figure 5

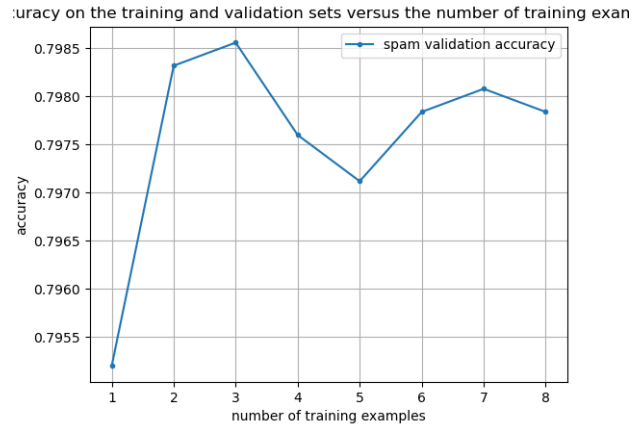


Figure 5: The C value and the corresponding validation accuracy

The best C value here is 3, the corresponding accuracy is 0.7985.

The Code of this part is listed in the Code Appendix "Code For K-Fold Validation".

6. My Kaggle name is "Leo-Adventure".

For the Mnist Competition: My Kaggle score is 0.97233. The Code for Mnist Competition is shown in the "Code For Kaggle MNIST".

For the SPAM Competition: My Kaggle score is 0.82333. The Code for Mnist Competition is shown in the "Code For Kaggle SPAM".

For the CIFAR10 Competition: My Kaggle score is 0.45366. The Code for Mnist Competition is shown in the "Code For Kaggle CIFAR-10".

I have use the K-Fold validation to tune the C of the SVM model, and it helps me improve my accuracy by provide me a proper C value. What's more, I use the GridSearchCV function of sklearn to help me tune a more proper C value. Though the training speed slow down a lot, it is useful in enhance the accuracy.

## 7. Theory of Hard-Margin Support Vector Machines

(a) Denote

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

To derive the  $w$  and  $\alpha$  that make the  $L$  obtain the maximum, we need to calculate the partial derivative of  $w$  and  $\alpha$ .

$$\frac{\partial L}{\partial w} = 2w - \sum_{i=1}^n \lambda_i y_i X_i = 0 \implies w = \frac{1}{2} \sum_{i=1}^n \lambda_i y_i X_i.$$

$$\frac{\partial L}{\partial \alpha} = - \sum_{i=1}^n \lambda_i y_i = 0 \implies \sum_{i=1}^n \lambda_i y_i = 0.$$

Then bring the value of  $w$  and  $\alpha$  into the formula, we can get

$$L(w, \alpha, \lambda_i) = \left(\frac{1}{2}\right)^2 \left(\sum_{i=1}^n \lambda_i y_i X_i\right) \left(\sum_{j=1}^n \lambda_j y_j X_j\right) - \left(\sum_{i=1}^n \frac{1}{2} \sum_{i=1}^n \lambda_i y_i X_i \lambda_i y_i X_i + \sum_{i=1}^n \lambda_i y_i \alpha - \sum_{i=1}^n \lambda_i\right)$$

Then we could get the dual optimization problem.

$$\max_{\lambda_i \geq 0} \sum_{i=1}^n \lambda_i - \frac{1}{4} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j X_i X_j$$

Which subject to  $\sum_{i=1}^n \lambda_i y_i = 0$ , which can be derived from the partial derivative of  $\alpha$

(b) From (a), we obtained that  $w = \frac{1}{2} \sum_{i=1}^n \lambda_i y_i X_i$ , if we know the values  $\lambda_i^*$  and  $\alpha^*$ , we can represent the  $w$  and  $\alpha$  in  $wx + \alpha$  by  $\lambda_i^*$  and  $\alpha^*$ , so that  $wx + \alpha \geq 0$  becomes  $\alpha^* + \frac{1}{2} \sum_{i=1}^n \lambda_i^* y_i X_i \cdot x \geq 0$

(c) Every point that is not a supporting vector has  $\lambda_i^* = 0$ , they will not effect the decision boundary. But considering those points that make  $\lambda_i^* > 0$ , which are supporting vector constructing the decision margin.

(d) Since the support vectors lie on or closest to the decision boundary, they are the most essential or critical data points in the training set.

(e)

## 8. Code Appendix

- Code for SVM

```
1 import numpy as np
2 from sklearn.svm import SVC
3 from sklearn import metrics
4 import matplotlib.pyplot as plt
5
6 if __name__ == "__main__":
7     # For the MNIST dataset, write code that sets aside 10,000 training images
8     # as a validation set.
9     mnist_data = np.load("../data/mnist-data.npz")
10    mnist_train = mnist_data["training_data"]
11    mnist_train_label = mnist_data["training_labels"]
12
13    mnist_state = np.random.get_state()
14    np.random.shuffle(mnist_train)
15    np.random.set_state(mnist_state)
16    np.random.shuffle(mnist_train_label)
17
18    mnist_validation = mnist_train[:9999]
19    mnist_validation_label = mnist_train_label[:9999]
20
21    mnist_train = mnist_train[10000:]
22    mnist_train_label = mnist_train_label[10000:]
23
24    svm_model = SVC(kernel='linear')
25    size_arr = [100, 200, 500, 1000, 2000, 5000, 10000]
26    nsamples, n1, n2, n3 = mnist_validation.shape
27    mnist_validation = mnist_validation.reshape(nsamples, n1*n2*n3)
28    val_acc_arr = []
29    train_acc_arr = []
30    for i in range(len(size_arr)):
31        mnist_train_section = mnist_train[:size_arr[i]-1]
32        mnist_train_label_section = mnist_train_label[:size_arr[i]-1]
33
34        # Since the fit function can only accept 2 dimension dataset, so here
35        # we use reshape to make it 2D
36        nsamples, n1, n2, n3 = mnist_train_section.shape
37        mnist_train_section = mnist_train_section.reshape(nsamples, n1*n2*n3)
38
39        svm_model.fit(mnist_train_section, mnist_train_label_section)
40
41        mnist_predict = svm_model.predict(mnist_train_section)
42
43        mnist_accuracy = metrics.accuracy_score(y_true=
44                                                    mnist_train_label_section, y_pred=
45                                                    mnist_predict)
46
47        train_acc_arr.append(mnist_accuracy)
48
49        mnist_predict = svm_model.predict(mnist_validation)
50
51        mnist_accuracy = metrics.accuracy_score(y_true=mnist_validation_label,
52                                                    y_pred=mnist_predict)
53
54        # print("accu = ")
55        # print(mnist_accuracy)
56        val_acc_arr.append(mnist_accuracy)
57
58    # plt.plot(size_arr, train_acc_arr, label='mnist training accuracy', marker
```



```

54         = ".")
# plt.plot(size_arr, val_acc_arr, label='mnist validation accuracy', marker
55         = "x")
# plt.xlabel('number of training examples')
56 # plt.ylabel('accuracy')
57 # plt.title("accuracy on the training and validation sets versus the number
of training examples")
58 # plt.grid(True)
59 # plt.legend()
60 # plt.show()
61
62
63
64 # For the spam dataset, write code that sets aside 20\% of the training
data as a validation set.
65 spam_data = np.load("../data/spam-data.npz")
66 spam_train = spam_data["training_data"]
67 spam_train_label = spam_data["training_labels"]
68
69 state = np.random.get_state()
70 np.random.shuffle(spam_train)
71 np.random.set_state(state)
72 np.random.shuffle(spam_train_label)
73
74 len_spam_train = int(len(spam_train)*0.8)
75 len_spam_validation = len(spam_train) - len_spam_train
76
77 spam_validation = spam_train[:len_spam_validation - 1]
78 spam_validation_label = spam_train_label[:len_spam_validation-1]
79
80 spam_train = spam_train[len_spam_validation:]
81 spam_train_label = spam_train_label[len_spam_validation:]
82
83 size_arr = [100, 200, 500, 1000, 2000, len_spam_train]
84
85 spam_train_acc_array = []
86 spam_validation_acc_array = []
87
88 for i in range(len(size_arr)):
89     spam_train_section = spam_train[:size_arr[i] - 1]
90     spam_train_label_section = spam_train_label[:size_arr[i] - 1]
91     svm_model.fit(spam_train_section, spam_train_label_section)
92
93     spam_train_pred = svm_model.predict(spam_train_section)
94     spam_train_acc = metrics.accuracy_score(y_true=spam_train_label_section
, y_pred=spam_train_pred)
95     spam_train_acc_array.append(spam_train_acc)
96
97     spam_val_pred = svm_model.predict(spam_validation)
98     spam_val_acc = metrics.accuracy_score(y_true=spam_validation_label,
y_pred = spam_val_pred)
99     spam_validation_acc_array.append(spam_val_acc)
100
101 # plt.plot(size_arr, spam_train_acc_array, label='spam training accuracy',
marker=".")
102 # plt.plot(size_arr, spam_validation_acc_array, label='spam validation
accuracy', marker="x")
103 # plt.xlabel('number of training examples')
104 # plt.ylabel('accuracy')
105 # plt.title("accuracy on the training and validation sets versus the number
of training examples")

```

```

106 # plt.grid(True)
107 # plt.legend()
108 # plt.show()
109
110 # For the CIFAR-10 dataset, write code that sets aside 5,000 training
      images as a validation set.
111
112 cifar_data = np.load("../data/cifar10-data.npz")
113 cifar_train = cifar_data["training_data"]
114 cifar_train_label = cifar_data["training_labels"]
115
116 state = np.random.get_state()
117 np.random.shuffle(cifar_train)
118 np.random.set_state(state)
119 np.random.shuffle(cifar_train_label)
120
121 cifar_validation = cifar_train[:4999]
122 cifar_validation_label = cifar_train_label[:4999]
123
124 cifar_train = cifar_train[5000:]
125 cifar_train_label = cifar_train_label[5000:]
126
127 size_arr = [100, 200, 500, 1000, 2000, 5000]
128 cifar_train_acc_array = []
129 cifar_validation_acc_array = []
130 for i in range(len(size_arr)):
131     cifar_train_section = cifar_train[:size_arr[i]-1]
132     cifar_train_label_section = cifar_train_label[:size_arr[i]-1]
133
134     svm_model.fit(cifar_train_section, cifar_train_label_section)
135
136     cifar_train_pred = svm_model.predict(cifar_train_section)
137     cifar_train_acc = metrics.accuracy_score(y_true=
      cifar_train_label_section, y_pred =
      cifar_train_pred)
138     cifar_train_acc_array.append(cifar_train_acc)
139
140     cifar_validation_pred = svm_model.predict(cifar_validation)
141     cifar_validation_acc = metrics.accuracy_score(y_true =
      cifar_validation_label, y_pred=
      cifar_validation_pred)
142     cifar_validation_acc_array.append(cifar_validation_acc)
143
144 plt.plot(size_arr, cifar_train_acc_array, label='cifar10 training accuracy',
      marker=".")
145 plt.plot(size_arr, cifar_validation_acc_array, label='cifar10 validation
      accuracy', marker="x")
146 plt.xlabel('number of training examples')
147 plt.ylabel('accuracy')
148 plt.title("accuracy on the training and validation sets versus the number
      of training examples")
149
150 plt.grid(True)
151 plt.legend()
152 plt.show()

```

- Code for Hyper-parameter Tuning

```

1 import numpy as np
2 from sklearn.svm import SVC
3 from sklearn import metrics
4 import matplotlib.pyplot as plt
5
6 if __name__ == "__main__":
7     # For the MNIST dataset, write code that sets aside 10,000 training images
8     # as a validation set.
9
10    mnist_data = np.load("../data/mnist-data.npz")
11    mnist_train = mnist_data["training_data"]
12    mnist_train_label = mnist_data["training_labels"]
13
14    mnist_state = np.random.get_state()
15    np.random.shuffle(mnist_train)
16    np.random.set_state(mnist_state)
17    np.random.shuffle(mnist_train_label)
18
19    mnist_validation = mnist_train[:9999]
20    mnist_validation_label = mnist_train_label[:9999]
21
22    mnist_train = mnist_train[10000:]
23    mnist_train_label = mnist_train_label[10000:]
24
25    c_arr = [5e-5, 1e-4, 1.5e-4, 2e-4, 2.5e-4, 3e-4, 3.5e-4, 4e-4, 4.5e-4, 5e-4]
26
27    mnist_train = mnist_train[:10000]
28    mnist_train_label = mnist_train_label[:10000]
29
30    nsamples, n1, n2, n3 = mnist_validation.shape
31    mnist_validation = mnist_validation.reshape(nsamples, n1*n2*n3)
32
33    nsamples, n1, n2, n3 = mnist_train.shape
34    mnist_train = mnist_train.reshape(nsamples, n1*n2*n3)
35
36    val_acc_arr = []
37    for i in range(len(c_arr)):
38        svm_model = SVC(kernel="linear", C=c_arr[i])
39        svm_model.fit(mnist_train, mnist_train_label)
40        mnist_pred = svm_model.predict(mnist_validation)
41        accuracy = metrics.accuracy_score(y_true=mnist_validation_label, y_pred
42                                         =mnist_pred)
43
44        val_acc_arr.append(accuracy)
45
46    print(val_acc_arr)
47    plt.plot(c_arr, val_acc_arr, label='mnist training accuracy', marker="x")
48    plt.xlabel('Value of C')
49    plt.ylabel('accuracy')
50    plt.title("accuracy on the validation sets versus the value of C in SVM
51              model")
52
53    plt.grid(True)
54    plt.legend()
55    plt.show()

```

- Code for K-Fold Cross Validation

```

1 import numpy as np
2 from sklearn.svm import SVC
3 from sklearn import metrics
4 import matplotlib.pyplot as plt
5
6 if __name__ == "__main__":
7     spam_data = np.load("../data/spam-data.npz")
8     spam_train = spam_data["training_data"]
9     spam_train_label = spam_data["training_labels"]
10
11     state = np.random.get_state()
12     np.random.shuffle(spam_train)
13     np.random.set_state(state)
14     np.random.shuffle(spam_train_label)
15
16     len_spam_train = len(spam_train)
17     len_partition = int(len_spam_train / 5)
18     c_array = [1, 2, 3, 4, 5, 6, 7, 8]
19     acc_arr = []
20     for j in range(len(c_array)):
21         svm_model = SVC(kernel="linear", C=c_array[j])
22         sub_acc_arr = []
23         for i in range(5):
24             training_set = spam_train[len_partition*(i):len_partition*(i+1)-1]
25             training_set_label = spam_train_label[len_partition*(i):
26                                                         len_partition*(i+1)-1]
27
28             j = (i+1)%5
29             validation_set = spam_train[len_partition*(j):len_partition*(j+1)-1]
30
31             validation_set_label = spam_train_label[len_partition*(j):
32                                                         len_partition*(j+1)-1]
33
34             svm_model.fit(training_set, training_set_label)
35             pred = svm_model.predict(validation_set)
36             acc = metrics.accuracy_score(y_true=validation_set_label, y_pred=
37                                         pred)
38
39             sub_acc_arr.append(acc)
40         num_arr = np.array(sub_acc_arr)
41         avg_val = np.mean(num_arr)
42         acc_arr.append(avg_val)
43
44     print(acc_arr)
45
46     plt.plot(c_array, acc_arr, label='spam validation accuracy', marker=".")
47     plt.xlabel('number of training examples')
48     plt.ylabel('accuracy')
49     plt.title("accuracy on the training and validation sets versus the number
50               of training examples")
51
52     plt.grid(True)
53     plt.legend()
54     plt.show()
55     \label{k_fold_py}
56     \caption{Code for K-Fold Validation}

```

- Code for Kaggle MNIST

```

1 import numpy as np
2 from sklearn.svm import SVC
3 from sklearn import metrics
4 import matplotlib.pyplot as plt
5 import pandas as pd
6 from sklearn.model_selection import KFold
7 from sklearn.model_selection import GridSearchCV
8
9 # Usage: results_to_csv(clf.predict(X_test))
10 def results_to_csv(y_test):
11     y_test = y_test.astype(int)
12     df = pd.DataFrame({'Category': y_test})
13     df.index += 1 # Ensures that the index starts at 1
14     df.to_csv('submission.csv', index_label='Id')
15
16 if __name__ == "__main__":
17     # For the MNIST dataset, write code that sets aside 10,000 training images
18                                     as a validation set.
19
20     mnist_data = np.load("../data/mnist-data.npz")
21     mnist_train = mnist_data["training_data"]
22     mnist_train_label = mnist_data["training_labels"]
23     mnist_test = mnist_data["test_data"]
24
25     mnist_state = np.random.get_state()
26     np.random.shuffle(mnist_train)
27     np.random.set_state(mnist_state)
28     np.random.shuffle(mnist_train_label)
29
30     mnist_validation = mnist_train[:9999]
31     mnist_validation_label = mnist_train_label[:9999]
32
33     mnist_train = mnist_train[10000:]
34     mnist_train_label = mnist_train_label[10000:]
35
36     # c_arr = [1.05e-2]
37
38     mnist_train = mnist_train[:15000]
39     mnist_train_label = mnist_train_label[:15000]
40
41     nsamples, n1, n2, n3 = mnist_validation.shape
42     mnist_validation = mnist_validation.reshape(nsamples, n1*n2*n3)
43
44     nsamples, n1, n2, n3 = mnist_train.shape
45     mnist_train = mnist_train.reshape(nsamples, n1*n2*n3)
46
47     nsamples, n1, n2, n3 = mnist_test.shape
48     mnist_test = mnist_test.reshape(nsamples, n1*n2*n3)
49
50     # svm_model = SVC(kernel="linear", C=c_arr[0]) 93.7%
51     params = {"C": [0.1, 1, 10, 100, 1000]}
52
53     folds = KFold(n_splits = 5, shuffle = True, random_state = 4)
54
55     # instantiating a model with cost=1
56     model = SVC()
57
58     # set up grid search scheme
59     # note that we are still using the 5 fold CV scheme we set up earlier
60     svm_model = GridSearchCV(estimator = model, param_grid = params,
61                               scoring= 'accuracy',

```

```

60         cv = folds,
61         verbose = 1,
62         return_train_score=True)
63     print("here")
64     # svm_model = SVC(C=10, gamma=0.001, kernel="rbf")
65     svm_model.fit(mnist_train, mnist_train_label)
66     print("after")
67     pred = svm_model.predict(mnist_validation)
68     acc = metrics.accuracy_score(y_true=mnist_validation_label, y_pred=pred)
69     print("mnist accuracy = ", str(acc))
70     pred = svm_model.predict(mnist_test)
71     results_to_csv(pred)
72

```

- Code for Kaggle SPAM

```

1 import numpy as np
2 from sklearn.svm import SVC
3 from sklearn import metrics
4 import matplotlib.pyplot as plt
5 import pandas as pd
6 from sklearn.model_selection import KFold
7 from sklearn.model_selection import GridSearchCV
8
9 # Usage: results_to_csv(clf.predict(X_test))
10 def results_to_csv(y_test):
11     y_test = y_test.astype(int)
12     df = pd.DataFrame({'Category': y_test})
13     df.index += 1 # Ensures that the index starts at 1
14     df.to_csv('spam_submission.csv', index_label='Id')
15
16 if __name__ == "__main__":
17     spam_data = np.load("../data/spam-data.npz")
18     spam_train = spam_data["training_data"]
19     spam_train_label = spam_data["training_labels"]
20     spam_test = spam_data["test_data"]
21     print(spam_test.shape)
22
23     # state = np.random.get_state()
24     # np.random.shuffle(spam_train)
25     # np.random.set_state(state)
26     # np.random.shuffle(spam_train_label)
27
28     # len_spam_train = len(spam_train)
29     # len_partition = int(len_spam_train / 5)
30     state = np.random.get_state()
31     np.random.shuffle(spam_train)
32     np.random.set_state(state)
33     np.random.shuffle(spam_train_label)
34
35     len_spam_train = int(len(spam_train)*0.8)
36     len_spam_validation = len(spam_train) - len_spam_train
37
38     spam_validation = spam_train[:len_spam_validation - 1]
39     spam_validation_label = spam_train_label[:len_spam_validation-1]
40
41     spam_train = spam_train[len_spam_validation:]
42     spam_train_label = spam_train_label[len_spam_validation:]
43     params = {"C": [0.1, 1, 10, 100, 1000]}
44
45     folds = KFold(n_splits = 5, shuffle = True, random_state = 4)
46
47     # instantiating a model with cost=1
48     model = SVC()
49
50     # set up grid search scheme
51     # note that we are still using the 5 fold CV scheme we set up earlier
52     svm_model = GridSearchCV(estimator = model, param_grid = params,
53                             scoring= 'accuracy',
54                             cv = folds,
55                             verbose = 1,
56                             return_train_score=True)
57     # svm_model = SVC(kernel="linear", C=c)
58     svm_model.fit(spam_train, spam_train_label)
59     pred = svm_model.predict(spam_validation)
60     acc = metrics.accuracy_score(y_true=spam_validation_label, y_pred=pred)

```

```
61     print(acc)
62     pred = svm_model.predict(spam_test)
63     results_to_csv(pred)
64
65
```



- Code for Kaggle CIFAR-10

```

1 import numpy as np
2 from sklearn.svm import SVC
3 from sklearn import metrics
4 import matplotlib.pyplot as plt
5 import pandas as pd
6 from sklearn.model_selection import KFold
7 from sklearn.model_selection import GridSearchCV
8
9 # Usage: results_to_csv(clf.predict(X_test))
10 def results_to_csv(y_test):
11     y_test = y_test.astype(int)
12     df = pd.DataFrame({'Category': y_test})
13     df.index += 1 # Ensures that the index starts at 1
14     df.to_csv('cifar_submission.csv', index_label='Id')
15
16
17 if __name__ == "__main__":
18     cifar10_data = np.load("../data/cifar10-data.npz")
19     cifar10_train = cifar10_data["training_data"]
20     cifar10_train_label = cifar10_data["training_labels"]
21     cifar10_test = cifar10_data["test_data"]
22     print(cifar10_test.shape)
23
24     # len_cifar10_train = len(cifar10_train)
25     # len_partition = int(len_cifar10_train / 5)
26     state = np.random.get_state()
27     np.random.shuffle(cifar10_train)
28     np.random.set_state(state)
29     np.random.shuffle(cifar10_train_label)
30
31     # len_cifar10_train = int(len(cifar10_train)*0.8)
32     # len_cifar10_validation = len(cifar10_train) - len_cifar10_train
33
34     cifar10_validation = cifar10_train[:4999]
35     cifar10_validation_label = cifar10_train_label[:4999]
36
37     cifar10_train = cifar10_train[5000:10000]
38     cifar10_train_label = cifar10_train_label[5000:10000]
39     params = {"C": [0.1, 1, 10, 100, 1000]}
40
41     folds = KFold(n_splits = 5, shuffle = True, random_state = 4)
42
43     # instantiating a model with cost=1
44     model = SVC()
45
46     # set up grid search scheme
47     # note that we are still using the 5 fold CV scheme we set up earlier
48     svm_model = GridSearchCV(estimator = model, param_grid = params,
49                             scoring= 'accuracy',
50                             cv = folds,
51                             verbose = 1,
52                             return_train_score=True)
53     # svm_model = SVC(kernel="linear", C=c)
54     print("here")
55     svm_model.fit(cifar10_train, cifar10_train_label)
56     print("after fit")
57     # pred = svm_model.predict(cifar10_validation)
58     # print("after")
59     # acc = metrics.accuracy_score(y_true=cifar10_validation_label, y_pred=pred
60 )

```

```
60     # print(acc)
61     pred = svm_model.predict(cifar10_test)
62     results_to_csv(pred)
63
64
```