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.

```python

import numpy as np
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt

if __name__ == "__main__":
    # For the MNIST dataset, write code that sets aside 10,000 training images as a validation set.
    mnist_data = np.load("../data/mnist-data.npz")
    mnist_train = mnist_data["training_data"]
    mnist_train_label = mnist_data["training_labels"]
    mnist_state = np.random.get_state()
    np.random.shuffle(mnist_train)
    np.random.set_state(mnist_state)
    np.random.shuffle(mnist_train_label)
    mnist_validation = mnist_train[:9999]
    mnist_validation_label = mnist_train_label[:9999]
    mnist_train = mnist_train[10000:]
    mnist_train_label = mnist_train_label[10000:]
    # For the spam dataset, write code that sets aside 20\% of the training data as a validation set.
    spam_data = np.load("../data/spam-data.npz")
    spam_train = spam_data["training_data"]
    spam_train_label = spam_data["training_labels"]
    state = np.random.get_state()
    np.random.shuffle(spam_train)
    np.random.set_state(state)
    np.random.shuffle(spam_train_label)
    len_spam_train = int(len(spam_train)*0.8)
    len_spam_validation = len(spam_train) - len_spam_train
    spam_validation = spam_train[:len_spam_validation - 1]
    spam_validation_label = spam_train_label[:len_spam_validation-1]
    spam_train = spam_train[len_spam_validation:]
    spam_train_label = spam_train_label[len_spam_validation:]
    # For the CIFAR-10 dataset, write code that sets aside 5,000 training images as a validation set.
    cifar_data = np.load("../data/cifar10-data.npz")
    cifar_train = cifar_data["training_data"]
    cifar_train_label = cifar_data["training_labels"]
    state = np.random.get_state()
    np.random.shuffle(cifar_train)
    np.random.set_state(state)
    np.random.shuffle(cifar_train_label)

    cifar_validation = cifar_train[:4999]
    cifar_validation_label = cifar_train_label[:4999]

    cifar_train = cifar_train[5000:]
    cifar_train_label = cifar_train_label[5000:]
```

3. Support Vector Machines: Coding

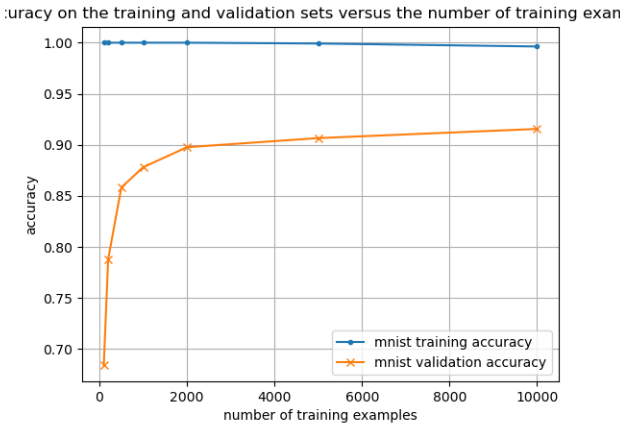

Figure 1: The SVM model training result of mnist dataset

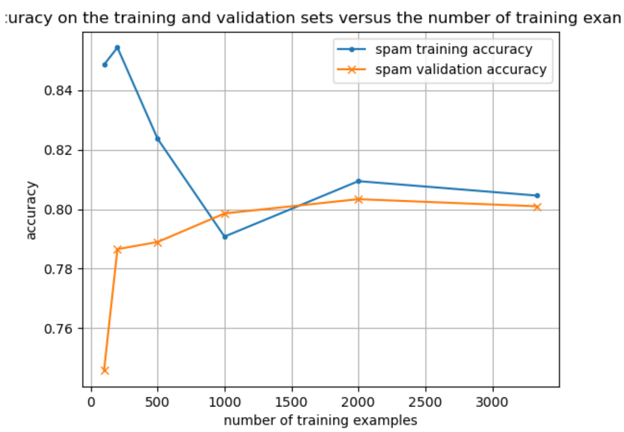

Figure 2: The SVM model training result of spam dataset

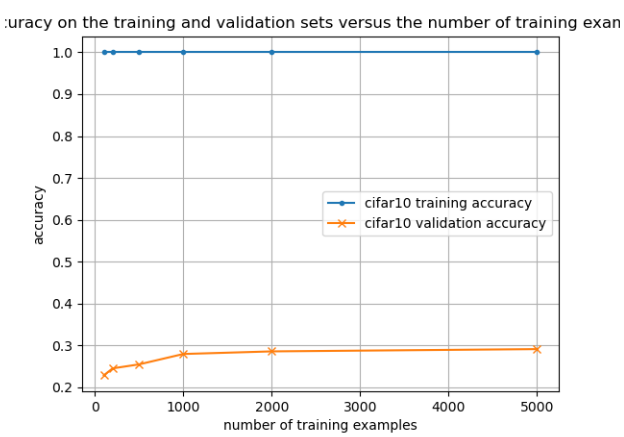

Figure 3: The SVM model training result of CIFAR-10 dataset

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

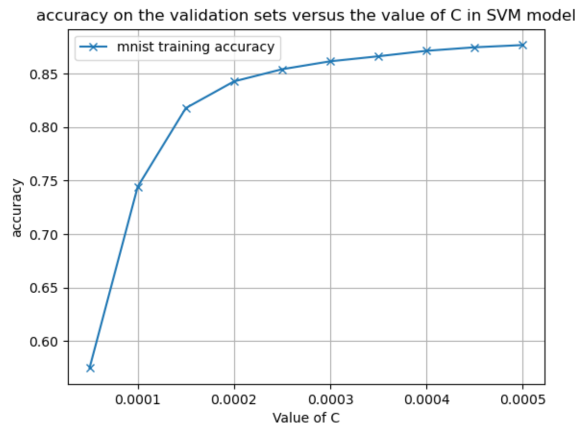

Figure 4: The accuracy score of mnist dataset based on different C

   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

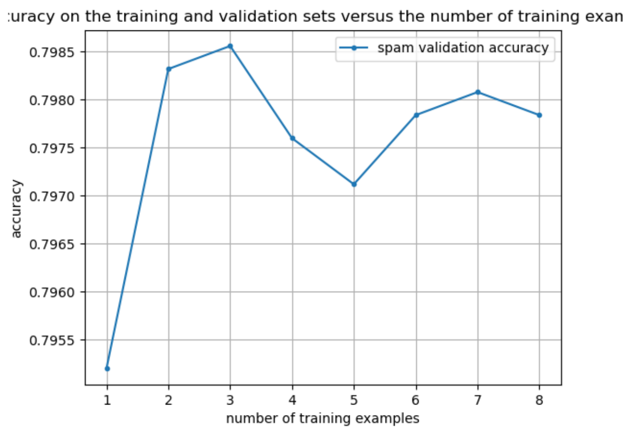

Figure 5: The C value and the corresponding validation accuracy

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 α that make the L obtain the maximum, we need to calculate the partial derivative of w and α.

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

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

Then bring the value of w and α into the formula, we can get

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

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 α

(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 λ_i^* and α^*, we can represent the w and α in $wx + \alpha$ by λ_i^* and α^*, 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

```
import numpy as np
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt

if __name__ == "__main__":
    # For the MNIST dataset, write code that sets aside 10,000 training images as a validation set.
    mnist_data = np.load("../data/mnist-data.npz")
    mnist_train = mnist_data["training_data"]
    mnist_train_label = mnist_data["training_labels"]

    mnist_state = np.random.get_state()
    np.random.shuffle(mnist_train)
    np.random.set_state(mnist_state)
    np.random.shuffle(mnist_train_label)

    mnist_validation = mnist_train[:9999]
    mnist_validation_label = mnist_train_label[:9999]

    mnist_train = mnist_train[10000:]
    mnist_train_label = mnist_train_label[10000:]


    svm_model = SVC(kernel='linear')
    size_arr = [100, 200, 500, 1000, 2000, 5000, 10000]
    nsamples, n1, n2, n3 = mnist_validation.shape
    mnist_validation = mnist_validation.reshape(nsamples, n1*n2*n3)
    val_acc_arr = []
    train_acc_arr = []
    for i in range(len(size_arr)):
        mnist_train_section = mnist_train[:size_arr[i]-1]
        mnist_train_label_section = mnist_train_label[:size_arr[i]-1]

        # Since the fit function can only accept 2 dimension dataset, so here we use reshape to make it 2D
        nsamples, n1, n2, n3 = mnist_train_section.shape
        mnist_train_section = mnist_train_section.reshape(nsamples, n1*n2*n3)

        svm_model.fit(mnist_train_section, mnist_train_label_section)

        mnist_predict = svm_model.predict(mnist_train_section)

        mnist_accuracy = metrics.accuracy_score(y_true=mnist_train_label_section, y_pred=mnist_predict)

        train_acc_arr.append(mnist_accuracy)

        mnist_predict = svm_model.predict(mnist_validation)

        mnist_accuracy = metrics.accuracy_score(y_true=mnist_validation_label, y_pred=mnist_predict)
        # print("accu = ")
        # print(mnist_accuracy)
        val_acc_arr.append(mnist_accuracy)

    # plt.plot(size_arr, train_acc_arr, label='mnist training accuracy', marker
```

```
="."）
    # plt.plot(size_arr, val_acc_arr, label='mnist validation accuracy', marker="x")
    # plt.xlabel('number of training examples')
    # plt.ylabel('accuracy')
    # plt.title("accuracy on the training and validation sets versus the number of training examples")
    # plt.grid(True)
    # plt.legend()
    # plt.show()



    # For the spam dataset, write code that sets aside 20\% of the training data as a validation set.
    spam_data = np.load("../data/spam-data.npz")
    spam_train = spam_data["training_data"]
    spam_train_label = spam_data["training_labels"]

    state = np.random.get_state()
    np.random.shuffle(spam_train)
    np.random.set_state(state)
    np.random.shuffle(spam_train_label)

    len_spam_train = int(len(spam_train)*0.8)
    len_spam_validation = len(spam_train) - len_spam_train

    spam_validation = spam_train[:len_spam_validation - 1]
    spam_validation_label = spam_train_label[:len_spam_validation-1]

    spam_train = spam_train[len_spam_validation:]
    spam_train_label = spam_train_label[len_spam_validation:]

    size_arr = [100, 200, 500, 1000, 2000, len_spam_train]

    spam_train_acc_array = []
    spam_validation_acc_array = []

    for i in range(len(size_arr)):
        spam_train_section = spam_train[:size_arr[i] - 1]
        spam_train_label_section = spam_train_label[:size_arr[i] - 1]
        svm_model.fit(spam_train_section, spam_train_label_section)

        spam_train_pred = svm_model.predict(spam_train_section)
        spam_train_acc = metrics.accuracy_score(y_true=spam_train_label_section, y_pred=spam_train_pred)
        spam_train_acc_array.append(spam_train_acc)

        spam_val_pred = svm_model.predict(spam_validation)
        spam_val_acc = metrics.accuracy_score(y_true=spam_validation_label, y_pred = spam_val_pred)
        spam_validation_acc_array.append(spam_val_acc)

    # plt.plot(size_arr, spam_train_acc_array, label='spam training accuracy', marker=".")
    # plt.plot(size_arr, spam_validation_acc_array, label='spam validation accuracy', marker="x")
    # plt.xlabel('number of training examples')
    # plt.ylabel('accuracy')
    # plt.title("accuracy on the training and validation sets versus the number of training examples")
```

```
    # plt.grid(True)
    # plt.legend()
    # plt.show()

    # For the CIFAR-10 dataset, write code that sets aside 5,000 training images as a validation set.
    cifar_data = np.load("../data/cifar10-data.npz")
    cifar_train = cifar_data["training_data"]
    cifar_train_label = cifar_data["training_labels"]

    state = np.random.get_state()
    np.random.shuffle(cifar_train)
    np.random.set_state(state)
    np.random.shuffle(cifar_train_label)

    cifar_validation = cifar_train[:4999]
    cifar_validation_label = cifar_train_label[:4999]

    cifar_train = cifar_train[5000:]
    cifar_train_label = cifar_train_label[5000:]

    size_arr = [100, 200, 500, 1000, 2000, 5000]
    cifar_train_acc_array = []
    cifar_validation_acc_array = []
    for i in range(len(size_arr)):
        cifar_train_section = cifar_train[:size_arr[i]-1]
        cifar_train_label_section = cifar_train_label[:size_arr[i]-1]

        svm_model.fit(cifar_train_section, cifar_train_label_section)

        cifar_train_pred = svm_model.predict(cifar_train_section)
        cifar_train_acc = metrics.accuracy_score(y_true=cifar_train_label_section, y_pred = cifar_train_pred)
        cifar_train_acc_array.append(cifar_train_acc)

        cifar_validation_pred = svm_model.predict(cifar_validation)
        cifar_validation_acc = metrics.accuracy_score(y_true = cifar_validation_label, y_pred=cifar_validation_pred)
        cifar_validation_acc_array.append(cifar_validation_acc)

    plt.plot(size_arr, cifar_train_acc_array, label='cifar10 training accuracy', marker=".")
    plt.plot(size_arr, cifar_validation_acc_array, label='cifar10 validation accuracy', marker="x")
    plt.xlabel('number of training examples')
    plt.ylabel('accuracy')
    plt.title("accuracy on the training and validation sets versus the number of training examples")
    plt.grid(True)
    plt.legend()
    plt.show()
```

- Code for Hyper-parameter Tuning

```
import numpy as np
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt

if __name__ == "__main__":
    # For the MNIST dataset, write code that sets aside 10,000 training images
        as a validation set.
    mnist_data = np.load("../data/mnist-data.npz")
    mnist_train = mnist_data["training_data"]
    mnist_train_label = mnist_data["training_labels"]

    mnist_state = np.random.get_state()
    np.random.shuffle(mnist_train)
    np.random.set_state(mnist_state)
    np.random.shuffle(mnist_train_label)

    mnist_validation = mnist_train[:9999]
    mnist_validation_label = mnist_train_label[:9999]

    mnist_train = mnist_train[10000:]
    mnist_train_label = mnist_train_label[10000:]

    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
        ]

    mnist_train = mnist_train[:10000]
    mnist_train_label = mnist_train_label[:10000]

    nsamples, n1, n2, n3 = mnist_validation.shape
    mnist_validation = mnist_validation.reshape(nsamples, n1*n2*n3)

    nsamples, n1, n2, n3 = mnist_train.shape
    mnist_train = mnist_train.reshape(nsamples, n1*n2*n3)

    val_acc_arr = []
    for i in range(len(c_arr)):
        svm_model = SVC(kernel="linear", C=c_arr[i])
        svm_model.fit(mnist_train, mnist_train_label)
        mnist_pred = svm_model.predict(mnist_validation)
        accuracy = metrics.accuracy_score(y_true=mnist_validation_label, y_pred
            =mnist_pred)
        val_acc_arr.append(accuracy)

    print(val_acc_arr)
    plt.plot(c_arr, val_acc_arr, label='mnist training accuracy', marker="x")
    plt.xlabel('Value of C')
    plt.ylabel('accuracy')
    plt.title("accuracy on the validation sets versus the value of C in SVM
        model")
    plt.grid(True)
    plt.legend()
    plt.show()
```

- Code for K-Fold Cross Validation

```python
import numpy as np
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt

if __name__ == "__main__":
    spam_data = np.load("../data/spam-data.npz")
    spam_train = spam_data["training_data"]
    spam_train_label = spam_data["training_labels"]

    state = np.random.get_state()
    np.random.shuffle(spam_train)
    np.random.set_state(state)
    np.random.shuffle(spam_train_label)

    len_spam_train = len(spam_train)
    len_partition = int(len_spam_train / 5)
    c_array = [1, 2, 3, 4, 5, 6, 7, 8]
    acc_arr = []
    for j in range(len(c_array)):
        svm_model = SVC(kernel="linear", C=c_array[j])
        sub_acc_arr = []
        for i in range(5):
            training_set = spam_train[len_partition*(i):len_partition*(i+1)-1]
            training_set_label = spam_train_label[len_partition*(i):len_partition*(i+1)-1]
            j = (i+1)%5
            validation_set = spam_train[len_partition*(j):len_partition*(j+1)-1]
            validation_set_label = spam_train_label[len_partition*(j):len_partition*(j+1)-1]

            svm_model.fit(training_set, training_set_label)
            pred = svm_model.predict(validation_set)
            acc = metrics.accuracy_score(y_true=validation_set_label, y_pred=pred)
            sub_acc_arr.append(acc)
        num_arr = np.array(sub_acc_arr)
        avg_val = np.mean(num_arr)
        acc_arr.append(avg_val)

    print(acc_arr)


    plt.plot(c_array, acc_arr, label='spam validation accuracy', marker=".")
    plt.xlabel('number of training examples')
    plt.ylabel('accuracy')
    plt.title("accuracy on the training and validation sets versus the number of training examples")
    plt.grid(True)
    plt.legend()
    plt.show()
    \label{k_fold_py}
    \caption{Code for K-Fold Validation}

```

- Code for Kaggle MNIST

```
import numpy as np
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

# 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')

if __name__ == "__main__":
    # For the MNIST dataset, write code that sets aside 10,000 training images
                                             as a validation set.
    mnist_data = np.load("../data/mnist-data.npz")
    mnist_train = mnist_data["training_data"]
    mnist_train_label = mnist_data["training_labels"]
    mnist_test = mnist_data["test_data"]

    mnist_state = np.random.get_state()
    np.random.shuffle(mnist_train)
    np.random.set_state(mnist_state)
    np.random.shuffle(mnist_train_label)

    mnist_validation = mnist_train[:9999]
    mnist_validation_label = mnist_train_label[:9999]

    mnist_train = mnist_train[10000:]
    mnist_train_label = mnist_train_label[10000:]

    # c_arr = [1.05e-2]

    mnist_train = mnist_train[:15000]
    mnist_train_label = mnist_train_label[:15000]

    nsamples, n1, n2, n3 = mnist_validation.shape
    mnist_validation = mnist_validation.reshape(nsamples, n1*n2*n3)

    nsamples, n1, n2, n3 = mnist_train.shape
    mnist_train = mnist_train.reshape(nsamples, n1*n2*n3)

    nsamples, n1, n2, n3 = mnist_test.shape
    mnist_test = mnist_test.reshape(nsamples, n1*n2*n3)

    # svm_model = SVC(kernel="linear", C=c_arr[0]) 93.7%
    params = {"C": [0.1, 1, 10, 100, 1000]}

    folds = KFold(n_splits = 5, shuffle = True, random_state = 4)

    # instantiating a model with cost=1
    model = SVC()

    # set up grid search scheme
    # note that we are still using the 5 fold CV scheme we set up earlier
    svm_model = GridSearchCV(estimator = model, param_grid = params,
                             scoring= 'accuracy',
```

```
                             cv = folds,
                             verbose = 1,
                         return_train_score=True)
    print("here")
    # svm_model = SVC(C=10, gamma=0.001, kernel="rbf")
    svm_model.fit(mnist_train, mnist_train_label)
    print("after")
    pred = svm_model.predict(mnist_validation)
    acc = metrics.accuracy_score(y_true=mnist_validation_label, y_pred=pred)
    print("mnist accuracy = ", str(acc))
    pred = svm_model.predict(mnist_test)
    results_to_csv(pred)

```

- Code for Kaggle SPAM

```
import numpy as np
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

# 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('spam_submission.csv', index_label='Id')

if __name__ == "__main__":
    spam_data = np.load("../data/spam-data.npz")
    spam_train = spam_data["training_data"]
    spam_train_label = spam_data["training_labels"]
    spam_test = spam_data["test_data"]
    print(spam_test.shape)

    # state = np.random.get_state()
    # np.random.shuffle(spam_train)
    # np.random.set_state(state)
    # np.random.shuffle(spam_train_label)

    # len_spam_train = len(spam_train)
    # len_partition = int(len_spam_train / 5)
    state = np.random.get_state()
    np.random.shuffle(spam_train)
    np.random.set_state(state)
    np.random.shuffle(spam_train_label)

    len_spam_train = int(len(spam_train)*0.8)
    len_spam_validation = len(spam_train) - len_spam_train

    spam_validation = spam_train[:len_spam_validation - 1]
    spam_validation_label = spam_train_label[:len_spam_validation-1]

    spam_train = spam_train[len_spam_validation:]
    spam_train_label = spam_train_label[len_spam_validation:]
    params = {"C": [0.1, 1, 10, 100, 1000]}

    folds = KFold(n_splits = 5, shuffle = True, random_state = 4)

    # instantiating a model with cost=1
    model = SVC()

    # set up grid search scheme
    # note that we are still using the 5 fold CV scheme we set up earlier
    svm_model = GridSearchCV(estimator = model, param_grid = params,
                             scoring= 'accuracy',
                             cv = folds,
                             verbose = 1,
                         return_train_score=True)
    # svm_model = SVC(kernel="linear", C=c)
    svm_model.fit(spam_train, spam_train_label)
    pred = svm_model.predict(spam_validation)
    acc = metrics.accuracy_score(y_true=spam_validation_label, y_pred=pred)
```

```
    print(acc)
    pred = svm_model.predict(spam_test)
    results_to_csv(pred)
```

- Code for Kaggle CIFAR-10

```python
import numpy as np
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

# 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('cifar_submission.csv', index_label='Id')


if __name__ == "__main__":
    cifar10_data = np.load("../data/cifar10-data.npz")
    cifar10_train = cifar10_data["training_data"]
    cifar10_train_label = cifar10_data["training_labels"]
    cifar10_test = cifar10_data["test_data"]
    print(cifar10_test.shape)

    # len_cifar10_train = len(cifar10_train)
    # len_partition = int(len_cifar10_train / 5)
    state = np.random.get_state()
    np.random.shuffle(cifar10_train)
    np.random.set_state(state)
    np.random.shuffle(cifar10_train_label)

    # len_cifar10_train = int(len(cifar10_train)*0.8)
    # len_cifar10_validation = len(cifar10_train) - len_cifar10_train

    cifar10_validation = cifar10_train[:4999]
    cifar10_validation_label = cifar10_train_label[:4999]

    cifar10_train = cifar10_train[5000:10000]
    cifar10_train_label = cifar10_train_label[5000:10000]
    params = {"C": [0.1, 1, 10, 100, 1000]}

    folds = KFold(n_splits = 5, shuffle = True, random_state = 4)

    # instantiating a model with cost=1
    model = SVC()

    # set up grid search scheme
    # note that we are still using the 5 fold CV scheme we set up earlier
    svm_model = GridSearchCV(estimator = model, param_grid = params,
                             scoring= 'accuracy',
                             cv = folds,
                             verbose = 1,
                         return_train_score=True)
    # svm_model = SVC(kernel="linear", C=c)
    print("here")
    svm_model.fit(cifar10_train, cifar10_train_label)
    print("after fit")
    # pred = svm_model.predict(cifar10_validation)
    # print("after")
    # acc = metrics.accuracy_score(y_true=cifar10_validation_label, y_pred=pred
        )
```

```
    # print(acc)
    pred = svm_model.predict(cifar10_test)
    results_to_csv(pred)


```