CS 189 Homework1

Xu Zhihao

July 1, 2019

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

Signature: Zhihao Xu

Contents

1 Python Configuration and Data Loading **2**

2 Data Partitioning **2**

3 Support Vector Machines: Coding **2**

4 Hyperparameter Tuning **4**

5 K-Fold Cross-Validation **5**

6 Kaggle **6**

7 Theory of Hard-Margin Support Vector Machines **7**

8 Appendix: Codes **9**

8.1 Data Partitioning 9
8.2 Support Vector Machines: Coding 10
8.3 Hyperparameter Tuning 14
8.4 K-Fold Cross-Validation 14
8.5 Prediction on the test data 15

1 Python Configuration and Data Loading

Configure python environment properly and successfully load the dataset.

2 Data Partitioning

Divide the dataset into training and validation set followed by the instruction.

3 Support Vector Machines: Coding

Train a linear support vector machine (SVM) with different number of training samples on all three datasets and plot the error rate versus the number of training sample.

(a) Train your model on **MNIST** dataset with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, 5,000, 10,000. The corresponding accuracy I get listed in following Table:

Accuracy of the Linear SVM Model on **MNIST** dataset

# of training sample	training accuracy	validation accuracy
100	1	0.7423
200	1	0.8088
500	1	0.8698
1000	1	0.8746
2000	1	0.8965
5000	1	0.9010
10000	1	0.9101

We can easily see that the validation accuracy increase with the increase of training sample.

(b) Train your model on **spam** dataset with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, **ALL**. The corresponding accuracy I get listed in following Table:

Accuracy of the Linear SVM Model on **spam** dataset

# of training sample	training accuracy	validation accuracy
100	0.840	0.753
200	0.850	0.813
500	0.856	0.814
1000	0.793	0.801
2000	0.7925	0.809
ALL(4138)	0. 800	0.812

(c) Train your model on **CIFAR-10** dataset with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, 5,000. The corresponding accuracy I get listed in following Table:

Accuracy of the Linear SVM Model on **CIFAR-10** dataset

# of training sample	training accuracy	validation accuracy
100	1	0.2154
200	1	0.2504
500	1	0.2714
1000	1	0.2738
2000	1	0.2910
5000	1	0.3030

I notice that there exist overfitting in the CIFAR-10 dataset. I tried to change the penalty term coefficient C, however it does not work. The training accuracy is still 1 and validation accuracy is relatively low.

4 Hyperparameter Tuning

Use different C value to train our linear SVM model. Here I listed all the C value I tried and the corresponding validation accuracy in following Table:

Accuracy of the Linear SVM Model on **MNIST** dataset with different C value

C value	validation accuracy	C value	validation accuracy
0.1	0.9124	0.8	0.9108
0.2	0.9087	0.9	0.9167
0.3	0.9075	1	0.9103
0.4	0.9104	2	0.9133
0.5	0.9119	3	0.9139
0.6	0.9157	4	0.9092
0.7	0.9149	5	0.9124

Here I notice that there is no strong difference in the validation an accuracy among different C values, the best C value here is C = 0.9

5 K-Fold Cross-Validation

Use different C value to train our linear SVM model and use 5-fold cross validation to get the validation accuracy. Here I listed all the C value I tried and the corresponding validation accuracy in following Table:

Accuracy of the Linear SVM Model on **spam** dataset with different C value

C value	validation accuracy	C value	validation accuracy
0.1	0.7939	0.8	0.8009
0.2	0.7968	0.9	0.8014
0.3	0.7993	1	0.8012
0.4	0.7993	2	0.8016
0.5	0.8003	3	0.8020
0.6	0.8007	4	0.8020
0.7	0.8005	5	0.8022

Pick the best C value with highest validation accuracy. Here I chose C=5.

6 Kaggle

My Kaggle username is Jack_xzh.
My Kaggle Score:

Dataset	Score
MNIST	0.91440
SPAM	0.94792
CIFAR-10	0.24540

Here I find one strange thing. When I apply other kernels like poly and gaussian (rbf), on validation set, the accuracy I can get is much higher than the linear kernel. However, when I submitted it on kaggle, the score is not as good as the linear one. Sometimes, even much lower. I guess it may caused by the split of training and testing data process. It may not general enough.

7 Theory of Hard-Margin Support Vector Machines

(a) In order to

$$\max_{\lambda_i \geq 0} \min_{w,\alpha} \|w\|_2 - \sum_{i=1}^{m} \lambda_i (y_i (X_i \cdot w + \alpha) - 1)$$

First we need to

$$\min_{w,\alpha} L_p = \|w\|_2 - \sum_{i=1}^{m} \lambda_i (y_i (X_i \cdot w + \alpha) - 1)$$

Take the first partial derivative of w and α

$$\frac{\partial L_p}{\partial w} = 2w - \sum_{i=1}^{m} \lambda_i y_i X_i = 0$$
$$\frac{\partial L_p}{\partial \alpha} = \sum_{i=1}^{m} \lambda_i y_i = 0$$

We can get $w = \frac{1}{2} \sum_{i=1}^{m} \lambda_i y_i X_i$ and $\sum_{i=1}^{m} \lambda_i y_i = 0$
Substitute it back:

$$\begin{aligned}
\|w\|_2 &= (\frac{1}{2} \sum_{i=1}^{m} \lambda_i y_i X_i)^T (\frac{1}{2} \sum_{i=1}^{m} \lambda_i y_i X_i) \\
&= \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j X_i X_j \\
L_p &= \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j X_i X_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j X_i X_j - \sum_{i=1}^{m} \alpha \lambda_i y_i + \sum_{i=1}^{m} \lambda_i \\
&= \sum_{i=1}^{m} \lambda_i - \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j X_i X_j
\end{aligned}$$

So, the Equation(3) can be rewritten as the dual optimization problem

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

(b) Use the result calculated in part (a), $w = \frac{1}{2} \sum_{i=1}^{m} \lambda_i y_i X_i$

$$w \cdot x + \alpha = (\frac{1}{2} \sum_{i=1}^{m} \lambda_i y_i X_i) x + \alpha,$$

Substitute the optimal value of λ^* and α^*,

$$w \cdot x + \alpha = \alpha^* + \frac{1}{2} (\sum_{i=1}^{m} \lambda_i^* y_i X_i) x,$$

So the decision rule in Equation (1) can be written as

$$r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2} (\sum_{i=1}^{m} \lambda_i^* y_i X_i) x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

(c)
- For all the non-support vector, the corresponding $\lambda_i = 0$. This point has no influence when we evaluate the Equation (4). So, the support vectors are the only training points needed to evaluate the decision rule.
- For all the non-support vector, when we fit the model, we still need it to help us find the support vectors and the decision boundary. When we fit the model firstly, we don't know whether the point is support vector or not. So, the non-support vectors still have some influence on the decision rule.

8 Appendix: Codes

All the codes and corresponding results are clearly listed in the appendix.

8.1 Data Partitioning

```
In [1]: import scipy.io
        import pandas as pd
        import numpy as np
        import random

        def processData(name,testing_size):
            path = 'data/' + name + ".mat"
            data = scipy.io.loadmat(path)

            data_X = data["training_data"]
            data_y = data["training_labels"]
            data_t = data["test_data"]

            if testing_size <= 1:
                testing_size = int(testing_size * data_X.shape[0])

            # random.seed(189)
            index = random.sample(range(data_X.shape[0]),data_X.shape[0]-testing_size

            data_X_train = data_X[index]
            data_X_validate = np.delete(data_X, index, axis=0)
            data_y_train = data_y[index]
            data_y_validate = np.delete(data_y, index, axis=0)

            Data = dict()
            Data["X_train"] = data_X_train
            Data["X_validate"] = data_X_validate
            Data["y_train"] = data_y_train
            Data["y_validate"] = data_y_validate
            Data["test"] = data_t
            return Data


In [2]: mnistData = processData("mnist_data",10000)

In [3]: spamData = processData("spam_data",0.2)
        print(spamData["X_train"].shape)

(4138, 32)


In [4]: cifar10Data = processData("cifar10_data",5000)
        print(cifar10Data["X_train"].shape)
```

```
(45000, 3072)
```

```
In [7]: from sklearn import svm
        from sklearn.metrics import accuracy_score

        def svmFit(data,training_sample,c=1,kernel="linear",gam="scale"):
            data_X = data["X_train"]
            data_y = data["y_train"]

            # random.seed(189)
            index = random.sample(range(data_X.shape[0]),training_sample)

            data_X_train = data_X[index]
            data_X_validate = data["X_validate"]
            data_y_train = data_y[index]
            data_y_validate = data["y_validate"]

            classifier=svm.SVC(C=c,kernel=kernel,max_iter=-1,gamma=gam)
            classifier.fit(data_X_train,data_y_train.ravel())

            y_validate = classifier.predict(data_X_validate)
            validate_accuracy = accuracy_score(y_validate,data_y_validate)

            y_train = classifier.predict(data_X_train)
            train_accuracy = accuracy_score(y_train,data_y_train)
            return train_accuracy,validate_accuracy
```

8.2 Support Vector Machines: Coding

Fit the model using given number of training sample and predict on the validation set.

```
In [9]: # mnistData
        t_error = []
        v_error = []
        training_sample = [100, 200, 500, 1000, 2000, 5000, 10000]
        for i in training_sample:
            ta,va = svmFit(mnistData,i,c=1)
            print(ta,va)
            t_error.append(1-ta)
            v_error.append(1-va)
```

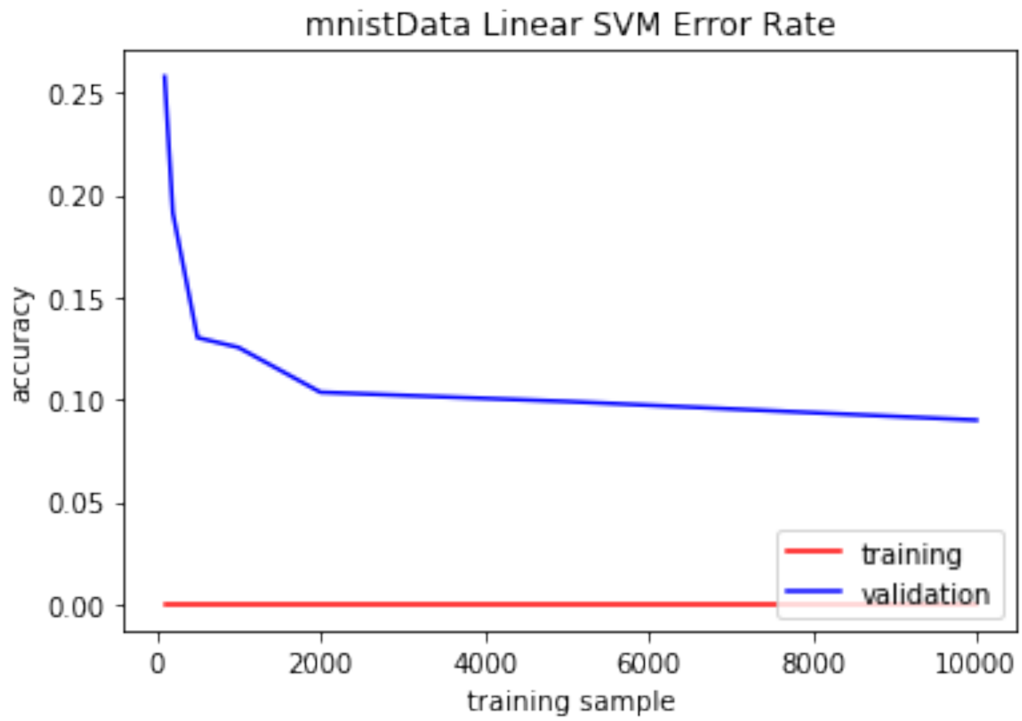
```
1.0 0.7423
1.0 0.8088
1.0 0.8698
1.0 0.8746
1.0 0.8965
1.0 0.901
1.0 0.9101
```

```
In [10]: # Optional: Fit the model by polynomial kernel
         svmFit(mnistData,10000,c=2,kernel="poly")[1]
```

```
Out[10]: 0.957
```

```
In [11]: import matplotlib.pyplot as plt
         %matplotlib inline
         plt.plot(training_sample,t_error,c="red",label="training")
         plt.plot(training_sample,v_error,c="blue",label="validation")
         plt.legend(loc="lower right")
         plt.xlabel("training sample")
         plt.ylabel("accuracy")
         plt.title("mnistData Linear SVM Error Rate")

         plt.show()
```




```
In [12]: # spamData
         t_error = []
         v_error = []

         training_sample = [100, 200, 500, 1000, 2000,4138]
         for i in training_sample:
             ta,va = svmFit(spamData,i,c=1,kernel="linear",gam="scale")
             print(ta,va)
             t_error.append(1-ta)
             v_error.append(1-va)
```

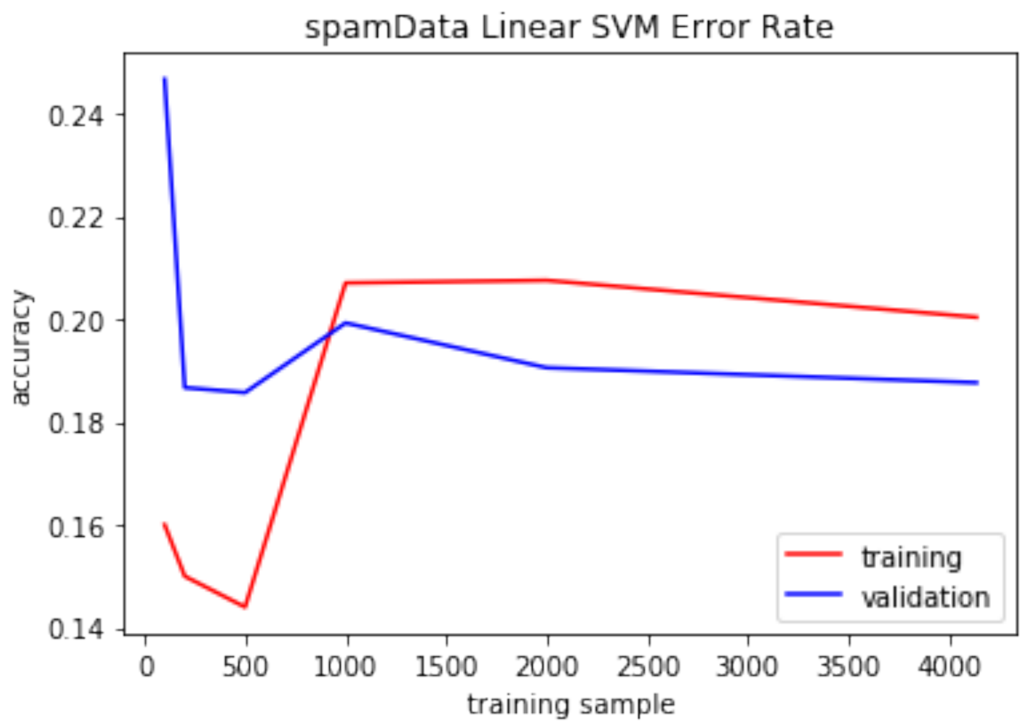
```
0.84 0.753384912959381
0.85 0.8133462282398453
0.856 0.8143133462282398
0.793 0.8007736943907157
0.7925 0.809477756286267
0.7996616723054616 0.8123791102514507
```

In [13]:
```
# Optional: Fit the model by kernel
svmFit(spamData,4138 ,c=20,kernel="rbf",gam="scale")[1]
```

Out[13]:
```
0.8355899419729207
```

In [14]:
```
plt.plot(training_sample,t_error,c="red",label="training")
plt.plot(training_sample,v_error,c="blue",label="validation")
plt.legend(loc="lower right")
plt.xlabel("training sample")
plt.ylabel("accuracy")
plt.title("spamData Linear SVM Error Rate")

plt.show()
```



In [15]:
```
# cifar10Data
t_error = []
v_error = []
```

```
training_sample = [100, 200, 500, 1000, 2000, 5000]
for i in training_sample:
    ta,va = svmFit(cifar10Data,i,c=1,kernel="linear")
    print(ta,va)
    t_error.append(1-ta)
    v_error.append(1-va)
```

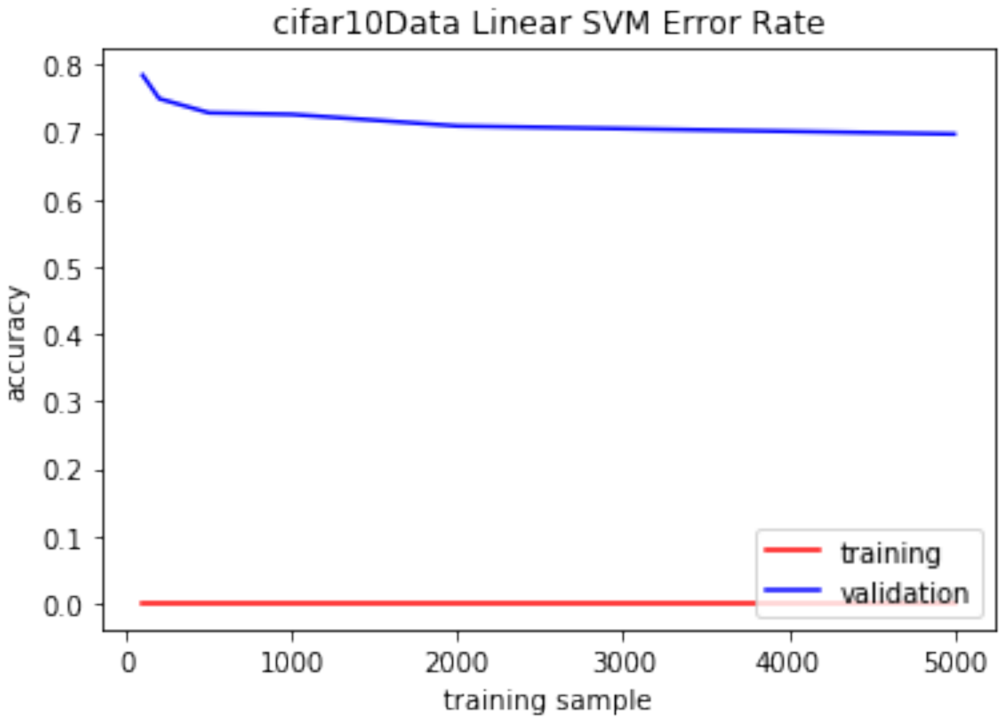
```
1.0 0.2154
1.0 0.2504
1.0 0.2714
1.0 0.2738
1.0 0.291
1.0 0.303
```

In [16]:
```
# Optional: Fit the model by kernel
svmFit(cifar10Data,5000,c=0.1,kernel="poly")[1]
```

Out[16]: 0.3778

In [17]:
```
plt.plot(training_sample,t_error,c="red",label="training")
plt.plot(training_sample,v_error,c="blue",label="validation")
plt.legend(loc="lower right")
plt.xlabel("training sample")
plt.ylabel("accuracy")
plt.title("cifar10Data Linear SVM Error Rate")

plt.show()
```



8.3 Hyperparameter Tuning

```
In [38]: v_accuracy = []

         listC = [x/10 for x in range(1,10)] + list(range(1,6))
         for i in listC:
             va = svmFit(mnistData,10000,c=i)[1]
             print(i,va)
             v_accuracy.append(va)
```

```
0.1 0.9124
0.2 0.9087
0.3 0.9075
0.4 0.9104
0.5 0.9119
0.6 0.9157
0.7 0.9149
0.8 0.9108
0.9 0.9167
1 0.9103
2 0.9113
3 0.9139
4 0.9092
5 0.9124
```

8.4 K-Fold Cross-Validation

```
In [39]: def KFoldCV(name,c,k=5,kernel="linear"):
             path = 'data/' + name + ".mat"
             data = scipy.io.loadmat(path)

             data_X = data["training_data"]
             data_y = data["training_labels"]

             random.seed(189)
             index = random.sample(range(data_X.shape[0]),data_X.shape[0])
             data_X = data_X[index]
             data_y = data_y[index]

             accu = []
             for i in range(k):
                 index = list(range( int(data_X.shape[0]*i/k) , int(data_X.shape[0]*(
                 data_X_validate = data_X[index]
                 data_X_train = np.delete(data_X, index, axis=0)
                 data_y_validate = data_y[index]
                 data_y_train = np.delete(data_y, index, axis=0)

                 classifier=svm.SVC(C=c,kernel=kernel,max_iter=-1,gamma="scale")
```

```
            classifier.fit(data_X_train,data_y_train.ravel())

            y_validate = classifier.predict(data_X_validate)
            validate_accuracy = accuracy_score(y_validate,data_y_validate)
            accu.append(validate_accuracy)

        return sum(accu)/len(accu)
```

```
In [40]: kfold_accuracy = []

         listC = [x/10 for x in range(1,10)] + list(range(1,6))
         for i in listC:
             ka = KFoldCV('spam_data',c=i,kernel="linear")
             print(i,ka)
             kfold_accuracy.append(ka)
```

```
0.1 0.7938898700230801
0.2 0.7967906633401546
0.3 0.799304235696465
0.4 0.799304235696465
0.5 0.8002709799194536
0.6 0.8006574533494053
0.7 0.8004642166344296
0.8 0.8008510638297872
0.9 0.8014309608574178
1 0.8012377241424421
2 0.8016243844550968
3 0.8020110447677515
4 0.8020112316504546
5 0.8022046552481334
```

8.5 Prediction on the test data

```
In [14]: # mnistData
         name = "mnist_data"
         path = 'data/' + name + ".mat"
         data = scipy.io.loadmat(path)

         data_X_train = data["training_data"]
         data_y_train = data["training_labels"]
         data_X_test = data["test_data"]

         # random.seed(189)
         index = random.sample(range(data_X_train.shape[0]),25000)

         data_X_train = data_X_train[index]
         data_y_train = data_y_train[index]
```

```
        classifier=svm.SVC(C=1,kernel="linear",max_iter=-1,gamma='scale')
        classifier.fit(data_X_train,data_y_train.ravel())

        y_test = classifier.predict(data_X_test)

In [15]: print(y_test)
         save = pd.DataFrame(y_test)
         save.index = range(1,len(save) + 1)
         save.to_csv("mnist_predict.csv")

[7 2 1 ... 4 5 6]


In [23]: # spamData
         name = "spam_data"
         path = 'data/' + name + ".mat"
         data = scipy.io.loadmat(path)

         data_X_train = data["training_data"]
         data_y_train = data["training_labels"]
         data_X_test = data["test_data"]

         classifier=svm.SVC(C=30,kernel="rbf",max_iter=-1,gamma='scale')
         classifier.fit(data_X_train,data_y_train.ravel())

         y_test = classifier.predict(data_X_test)

In [24]: print(y_test)
         save = pd.DataFrame(y_test)
         save.index = range(1,len(save) + 1)
         save.to_csv("spam_predict.csv")

[1 1 0 ... 0 0 0]


In [18]: # cifar10Data
         name = "cifar10_data"
         path = 'data/' + name + ".mat"
         data = scipy.io.loadmat(path)

         data_X_train = data["training_data"]
         data_y_train = data["training_labels"]
         data_X_test = data["test_data"]

         random.seed(189)
         index = random.sample(range(data_X_train.shape[0]),15000)

         data_X_train = data_X_train[index]
         data_y_train = data_y_train[index]
```

```
        print("Data done")
        classifier=svm.SVC(C=0.1,kernel="poly",max_iter=-1,gamma='scale')
        print("fitted")
        classifier.fit(data_X_train,data_y_train.ravel())
        print("start predicting")
        y_test = classifier.predict(data_X_test)
```

```
Data done
fitted
start predicting
```

```
In [19]: print(y_test)
         save = pd.DataFrame(y_test)
         save.index = range(1,len(save) + 1)
         save.to_csv("cifar10_predict.csv")
```

```
[3 9 0 ... 5 5 2]
```

```
In [ ]:
```