ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

SUPPORT VECTOR MACHINES

Panagiotis Karvounaris
AEM 10193

December 24, 2023

A report presented for the course of Neural Networks: Deep learning
Aristotle University of Thessaloniki Faculty of Electrical and Computer Engineering

Contents

1 Introduction

The goal of this project is to use the Support Vector Machines (SVM), from sklearn python library, to classify correctly the pictures of CIFAR-10 dataset. The dataset is consisted of pictures 32x32 pixels and coloured. We will use different SVMs and options for training in order to achieve the highest test accuracy. We start training a variety of parameters with different values for all the possible SVMs with the use of 5000 samples. This decision was made due to time limitation for these many SVMs' training. After this run, we will save the best results and run again for 10000 samples. We save the results of this run too. Then, we do the 10000 samples run again but we are using PCA to preprocess the training and test data. At the end, we get the parameters that gave us the best results from the above and use this train cases to train the SVM to 50000 samples (the whole dataset).

2 Code Description

In order to test multiple SVMs and a great variety of parameters, we used COTS SVMs from sklearn library. The code analysis is explained below.

2.1 Load and Preprocess the CIFAR-10 dataset

We use the 'tensorflow.keras.datasets' library to load the CIFAR-10 dataset. First of all, we load the data for training and testing, then we reshape the image data and scales the pixel values dividing by 255. We reshape the original image data from a 4D array to a 2D array. It uses information about the number of samples, image height, image width, and the number of channels. So, each image has a total of 32x32x3 = 3072 features. Furthermore, we shuffle the data in order to have better results in the training for 5000 and 10000 samples.

```
import tensorflow as tf
from sklearn.utils import shuffle

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

x_train = x_train.reshape((-1, 3072)) / 255.0
x_test = x_test.reshape((-1, 3072)) / 255.0

y_train = y_train.ravel()
y_test = y_test.ravel()

x_train, y_train = shuffle(x_train, y_train, random_state=0)
```

2.2 PCA

For most of the training instances we use Primary Component Analysis (PCA), in order to speed up the training process. PCA is a linear dimensionality reduction technique and we use to to reduce the every image dimension from 3072 to about 100 dimensions, while keeping at least 90 percent of the initial information of the image. The speed of the training is improving, thus we can use more training samples, so we end up training a bigger potion of the dataset at the same time, that increases the performance of the SVM.

```
from sklearn.decomposition import PCA
pca = PCA(0.9).fit(x_train)
pca_train_data = pca.transform(x_train)
pca_test_data = pca.transform(x_test)
```

2.3 Options For Gamma Parameters and RBF Kernel SVC

if gamma is equal to 'scale', which is the default option, then it uses $1/(n\_features * X.var())$ as value of gamma.

if gamma is equal ‘auto’, gamma is uses $1/n\_features$
if decision function shape = ”ovo”, whether to return a one-vs-rest (‘ovr’) decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one (‘ovo’) decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, note that internally, one-vs-one (‘ovo’) is always used as a multi-class strategy to train models

2.4 Save Data to File

In order to keep track of the results and use them afterwards to reach conclusions, we save the results of the the training in a .txt file. We could use grid search in order to obtain the best combination of parameters, but we chose to run all the the SVM models and find the best combination manually. We also print the results in terminal for progress notification reasons.

```
def print_and_log(text, log_file):
    print(text)
    log_file.write(text + '\n')

with open("svm_log.txt", "w") as log_file:
    print_and_log('data to be saved', log_file)
```

2.5 Training SVM Models

1) Linear Kernel, 6 values for C

$$C = \big[0.1, 1, 10, 50, 100, 1000\big]$$

2) Polynomial Kernel, 2 values for degree, 6 values for C and 7 values for gamma

$$degree = \big[2, 3\big]$$

$$C = \big[0.1, 1, 10, 50, 100, 1000\big]$$

$$gamma = \big['scale', 'auto', 0.1, 0.5, 1, 10, 20\big]$$

3) Sigmoid Kernel, 6 values for C and 7 values for gamma

$$C = \big[0.1, 1, 10, 50, 100, 1000\big]$$

$$gamma = \big['scale', 'auto', 0.1, 0.5, 1, 10, 20\big]$$

4) Radial Basis Function (RBF) Kernel, 6 values for C and 7 values for gamma

$$C = \big[0.1, 1, 10, 50, 100, 1000\big]$$

$$gamma = \big['scale', 'auto', 0.1, 0.5, 1, 10, 20\big]$$

2.6 No PCA, 5000 Samples

```
for C in [0.1, 1, 10, 50, 100, 1000]:
    start_time = time()
    linearSVM = SVC(kernel='linear', C=C)
    linearSVM.fit(x_train[:5000], y_train[:5000])
    output = "Linear Kernel, C = " + str(C) + "\n"
    output += "Number of support vectors = " + str(linearSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(linearSVM.score(x_train[:5000], y_train[:5000])) + '
    output += "Test accuracy = " + str(linearSVM.score(x_test, y_test)) + "\n"
    print_and_log(output, log_file)

for degree in [2, 3]:
    for C in [0.1, 1, 10, 50, 100, 1000]:
        for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
            start_time = time()
            polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma)
            polySVM.fit(x_train[:5000], y_train[:5000])
            output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", (
            output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
            output += "Elapsed time: " + str(time() - start_time) + "\n"
            output += "Train accuracy = " + str(polySVM.score(x_train[:5000], y_train[:5000]
            output += "Test accuracy = " + str(polySVM.score(x_test, y_test)) + "\n"
            print_and_log(output, log_file)

for C in [0.1, 1, 10, 50, 100, 1000]:
    for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
        start_time = time()
        sigmoidSVM = SVC(kernel='sigmoid', C=C, gamma=gamma)
        sigmoidSVM.fit(x_train[:5000], y_train[:5000])
        output = "Sigmoid Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
        output += "Number of support vectors = " + str(sigmoidSVM.n_support_) + "\n"
        output += "Elapsed time: " + str(time() - start_time) + "\n"
        output += "Train accuracy = " + str(sigmoidSVM.score(x_train[:5000], y_train[:5000])
        output += "Test accuracy = " + str(sigmoidSVM.score(x_test, y_test)) + "\n"
        print_and_log(output, log_file)

for C in [0.1, 1, 10, 50, 100, 1000]:
    for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
        start_time = time()
        rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma)
        rbfSVM.fit(x_train[:5000], y_train[:5000])
        output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
        output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
        output += "Elapsed time: " + str(time() - start_time) + "\n"
```

```
        output += "Train accuracy = " + str(rbfSVM.score(x_train[:5000], y_train[:5000])) +
        output += "Test accuracy = " + str(rbfSVM.score(x_test, y_test)) + "\n"
        print_and_log(output, log_file)
```

Best results are below:

- RBF Kernel, C = 10, Gamma = scale
Number of support vectors = [466 473 486 527 470 543 480 483 433 501]
Elapsed time: 48.928696155548096
Train accuracy = 0.997
Test accuracy = 0.4462
- RBF Kernel, C = 50, Gamma = scale
Number of support vectors = [470 473 486 527 470 542 482 484 429 501]
Elapsed time: 48.48292779922485
Train accuracy = 1.0
Test accuracy = 0.4414
- RBF Kernel, C = 100, Gamma = scale
Number of support vectors = [470 473 487 527 470 542 482 484 429 501] //
Elapsed time: 47.37176752090454
Train accuracy = 1.0
Test accuracy = 0.4414
- RBF Kernel, C = 1000, Gamma = scale
Number of support vectors = [470 473 487 527 470 542 482 484 429 501]
Elapsed time: 45.42233157157898
Train accuracy = 1.0
Test accuracy = 0.4414

2.7 No PCA, 10000 Samples

```
for C in [0.1, 1, 10, 50, 100, 1000]:
    start_time = time()
    linearSVM = SVC(kernel='linear', C=C)
    linearSVM.fit(x_train[:10000], y_train[:10000])
    output = "Linear Kernel, C = " + str(C) + "\n"
    output += "Number of support vectors = " + str(linearSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(linearSVM.score(x_train[:10000], y_train[:10000]))
    output += "Test accuracy = " + str(linearSVM.score(x_test, y_test)) + "\n"
    print_and_log(output, log_file)

for degree in [2, 3]:
    for C in [0.1, 1, 10, 50, 100, 1000]:
        for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
            start_time = time()
            polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma)
            polySVM.fit(x_train[:10000], y_train[:10000])
            output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ",
```

```
            output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
            output += "Elapsed time: " + str(time() - start_time) + "\n"
            output += "Train accuracy = " + str(polySVM.score(x_train[:10000], y_train[:1000
            output += "Test accuracy = " + str(polySVM.score(x_test, y_test)) + "\n"
            print_and_log(output, log_file)

for C in [0.1, 1, 10, 50, 100, 1000]:
    for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
        start_time = time()
        sigmoidSVM = SVC(kernel='sigmoid', C=C, gamma=gamma)
        sigmoidSVM.fit(x_train[:10000], y_train[:10000])
        output = "Sigmoid Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
        output += "Number of support vectors = " + str(sigmoidSVM.n_support_) + "\n"
        output += "Elapsed time: " + str(time() - start_time) + "\n"
        output += "Train accuracy = " + str(sigmoidSVM.score(x_train[:10000], y_train[:10000
        output += "Test accuracy = " + str(sigmoidSVM.score(x_test, y_test)) + "\n"
        print_and_log(output, log_file)

for C in [0.1, 1, 10, 50, 100, 1000]:
    for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
        start_time = time()
        rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma)
        rbfSVM.fit(x_train[:10000], y_train[:10000])
        output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
        output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
        output += "Elapsed time: " + str(time() - start_time) + "\n"
        output += "Train accuracy = " + str(rbfSVM.score(x_train[:10000], y_train[:10000]))
        output += "Test accuracy = " + str(rbfSVM.score(x_test, y_test)) + "\n"
        print_and_log(output, log_file)
```

Best results are below:

- Polynomial Kernel, Degree = 3, C = 1000, Gamma = auto

Number of support vectors = [815 871 979 997 939 955 890 857 756 908]
Elapsed time: 88.35298943519592
Train accuracy = 0.8619
Test accuracy = 0.4447

- RBF Kernel, C = 1, Gamma = scale

Number of support vectors = [898 940 1007 1022 955 990 895 931 858 998]
Elapsed time: 98.53359794616699
Train accuracy = 0.7101
Test accuracy = 0.4721

- RBF Kernel, C = 10, Gamma = scale

Number of support vectors = [938 953 1005 1022 954 1002 924 939 874 1014]
Elapsed time: 123.72975516319275
Train accuracy = 0.9923
Test accuracy = 0.4782

- RBF Kernel, C = 100, Gamma = scale

Number of support vectors = [940 952 1005 1022 957 1000 926 940 879 1016]
Elapsed time: 123.63887023925781
Train accuracy = 1.0
Test accuracy = 0.4728

2.8 No PCA, 10000 Samples

```
pca = PCA(0.9).fit(x_train)
pca_train_data = pca.transform(x_train)
pca_test_data = pca.transform(x_test)

for C in [0.1, 1, 10, 50, 100, 1000]:
    start_time = time()
    linearSVM = SVC(kernel='linear', C=C)
    linearSVM.fit(pca_train_data[:10000], y_train[:10000])
    output = "Linear Kernel, C = " + str(C) + "\n"
    output += "Number of support vectors = " + str(linearSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(linearSVM.score(pca_train_data[:5000], y_train[:5000
    output += "Test accuracy = " + str(linearSVM.score(pca_test_data, y_test)) + "\n"
    print_and_log(output, log_file)

for degree in [2, 3]:
    for C in [0.1, 1, 10, 50, 100, 1000]:
        for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
            start_time = time()
            polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma)
            polySVM.fit(pca_train_data[:10000], y_train[:10000])
            output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", G
            output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
            output += "Elapsed time: " + str(time() - start_time) + "\n"
            output += "Train accuracy = " + str(polySVM.score(pca_train_data[:10000], y_trai
            output += "Test accuracy = " + str(polySVM.score(pca_test_data, y_test)) + "\n"
            print_and_log(output, log_file)

for C in [0.1, 1, 10, 50, 100, 1000]:
    for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
        start_time = time()
        sigmoidSVM = SVC(kernel='sigmoid', C=C, gamma=gamma)
        sigmoidSVM.fit(pca_train_data[:10000], y_train[:10000])
        output = "Sigmoid Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
        output += "Number of support vectors = " + str(sigmoidSVM.n_support_) + "\n"
        output += "Elapsed time: " + str(time() - start_time) + "\n"
        output += "Train accuracy = " + str(sigmoidSVM.score(pca_train_data[:10000], y_train
        output += "Test accuracy = " + str(sigmoidSVM.score(pca_test_data, y_test)) + "\n"
```

```
        print_and_log(output, log_file)

for C in [0.1, 1, 10, 50, 100, 1000]:
    for gamma in ['scale', 'auto', 0.1, 0.5, 1, 10, 20]:
        start_time = time()
        rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma)
        rbfSVM.fit(pca_train_data[:10000], y_train[:10000])
        output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
        output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
        output += "Elapsed time: " + str(time() - start_time) + "\n"
        output += "Train accuracy = " + str(rbfSVM.score(pca_train_data[:10000], y_train[:1
        output += "Test accuracy = " + str(rbfSVM.score(pca_test_data, y_test)) + "\n"
        print_and_log(output, log_file)
```

Best results are below:

- Polynomial Kernel, Degree = 3, C = 1000, Gamma = auto
Number of support vectors = [815 871 979 997 939 955 890 857 756 908]
Elapsed time: 88.35298943519592
Train accuracy = 0.8619
Test accuracy = 0.4447
- RBF Kernel, C = 1, Gamma = scale
Number of support vectors = [898 940 1007 1022 955 990 895 931 858 998]
Elapsed time: 98.53359794616699
Train accuracy = 0.7101
Test accuracy = 0.4721
- RBF Kernel, C = 10, Gamma = scale
Number of support vectors = [938 953 1005 1022 954 1002 924 939 874 1014]
Elapsed time: 123.72975516319275
Train accuracy = 0.9923
Test accuracy = 0.4782
- RBF Kernel, C = 100, Gamma = scale
Number of support vectors = [940 952 1005 1022 957 1000 926 940 879 1016]
Elapsed time: 123.63887023925781
Train accuracy = 1.0
Test accuracy = 0.4728

2.9 PCA, 5000 Samples, Best Cases

```
pca = PCA(0.9).fit(x_train)
pca_train_data = pca.transform(x_train)
pca_test_data = pca.transform(x_test)

C = 0.1
start_time = time()
linearSVM = SVC(kernel='linear', C=C)
linearSVM.fit(pca_train_data[:5000], y_train[:5000])
```

```
output = "Linear Kernel, C = " + str(C) + "\n"
output += "Number of support vectors = " + str(linearSVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(linearSVM.score(pca_train_data[:5000], y_train[:5000]))
output += "Test accuracy = " + str(linearSVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

C = 10
gamma = 'scale'
degree = 2
coef0 = 0.5
start_time = time()
polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma, coef0 = coef0)
polySVM.fit(pca_train_data[:5000], y_train[:5000])
output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", Gamma = " + s
output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(polySVM.score(pca_train_data[:5000], y_train[:5000])) +
output += "Test accuracy = " + str(polySVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

C = 10
gamma = 0.5
degree = 2
coef0 = 0.5
start_time = time()
polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma, coef0 = coef0, probability = T
polySVM.fit(pca_train_data[:5000], y_train[:5000])
output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", Gamma = " + s
output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(polySVM.score(pca_train_data[:5000], y_train[:5000])) +
output += "Test accuracy = " + str(polySVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

gamma = 'scale'
for C in [1, 10, 100, 1000]:
    start_time = time()
    rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma)
    rbfSVM.fit(pca_train_data[:5000], y_train[:5000])
    output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
    output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(rbfSVM.score(pca_train_data[:5000], y_train[:5000]))
    output += "Test accuracy = " + str(rbfSVM.score(pca_test_data, y_test)) + "\n"
    print_and_log(output, log_file)
```

```
gamma = 'scale'
for C in [1, 10, 100, 1000]:
    start_time = time()
    rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma, decision_function_shape = "ovo")
    rbfSVM.fit(pca_train_data[:5000], y_train[:5000])
    output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "decision_function_sh
    output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(rbfSVM.score(pca_train_data[:5000], y_train[:5000]))
    output += "Test accuracy = " + str(rbfSVM.score(pca_test_data, y_test)) + "\n"
    print_and_log(output, log_file)
```

Best results are below:

- Linear Kernel, C = 0.1

Number of support vectors = [446 436 477 516 463 523 441 438 397 452]
Elapsed time: 2.4526445865631104
Train accuracy = 0.4708
Test accuracy = 0.3621

- Polynomial Kernel, Degree = 2, C = 10, Gamma = scale, coef0 = 0.5

Number of support vectors = [406 430 480 510 463 503 460 446 385 461]
Elapsed time: 1.7975034713745117
Train accuracy = 0.9206
Test accuracy = 0.4199

- RBF Kernel, C = 1, Gamma = scale

Number of support vectors = [453 469 486 526 467 535 465 477 424 488]
Elapsed time: 1.495023488998413
Train accuracy = 0.6632
Test accuracy = 0.4359

- RBF Kernel, C = 10, Gamma = scale

Number of support vectors = [454 468 485 525 469 533 472 470 421 493]
Elapsed time: 2.0700836181640625
Train accuracy = 0.972
Test accuracy = 0.4446

- RBF Kernel, C = 100, Gamma = scale

Number of support vectors = [453 466 485 525 469 529 477 470 419 490]
Elapsed time: 2.179264545440674
Train accuracy = 1.0
Test accuracy = 0.4275

- RBF Kernel, C = 1000, Gamma = scale

Number of support vectors = [453 466 485 525 470 529 477 470 419 490]
Elapsed time: 2.17147159576416
Train accuracy = 1.0
Test accuracy = 0.4276

- RBF Kernel, C = 1, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [453 469 486 526 467 535 465 477 424 488]

Elapsed time: 1.5112762451171875
Train accuracy = 0.6632
Test accuracy = 0.4359
- RBF Kernel, C = 10, Gamma = scale, decision_function_shape = "ovo"
Number of support vectors = [454 468 485 525 469 533 472 470 421 493]
Elapsed time: 2.059635639190674
Train accuracy = 0.972
Test accuracy = 0.4446
- RBF Kernel, C = 100, Gamma = scale, decision_function_shape = "ovo"
Number of support vectors = [453 466 485 525 469 529 477 470 419 490]
Elapsed time: 2.1751701831817627
Train accuracy = 1.0
Test accuracy = 0.4275
- RBF Kernel, C = 1000, Gamma = scale, decision_function_shape = "ovo"
Number of support vectors = [453 466 485 525 470 529 477 470 419 490]
Elapsed time: 2.183816909790039
Train accuracy = 1.0
Test accuracy = 0.4276

2.10 PCA, 10000 Samples, Best Cases

```
pca = PCA(0.9).fit(x_train)
pca_train_data = pca.transform(x_train)
pca_test_data = pca.transform(x_test)

C = 0.1
start_time = time()
linearSVM = SVC(kernel='linear', C=C)
linearSVM.fit(pca_train_data[:10000], y_train[:10000])
output = "Linear Kernel, C = " + str(C) + "\n"
output += "Number of support vectors = " + str(linearSVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(linearSVM.score(pca_train_data[:10000], y_train[:10000])
output += "Test accuracy = " + str(linearSVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

C = 10
gamma = 'scale'
degree = 2
coef0 = 0.5
start_time = time()
polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma, coef0 = coef0)
polySVM.fit(pca_train_data[:10000], y_train[:10000])
output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", Gamma = " + s
output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
```

```
output += "Train accuracy = " + str(polySVM.score(pca_train_data[:10000], y_train[:10000]))
output += "Test accuracy = " + str(polySVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

C = 10
gamma = 0.5
degree = 2
coef0 = 0.5
start_time = time()
polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma, coef0 = coef0, probability = T
polySVM.fit(pca_train_data[:10000], y_train[:10000])
output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", Gamma = " + s
output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(polySVM.score(pca_train_data[:10000], y_train[:10000]))
output += "Test accuracy = " + str(polySVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

gamma = 'scale'
for C in [1, 10, 100, 1000]:
    start_time = time()
    rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma)
    rbfSVM.fit(pca_train_data[:10000], y_train[:10000])
    output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
    output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(rbfSVM.score(pca_train_data[:10000], y_train[:10000]
    output += "Test accuracy = " + str(rbfSVM.score(pca_test_data, y_test)) + "\n"
    print_and_log(output, log_file)

gamma = 'scale'
for C in [1, 10, 100, 1000]:
    start_time = time()
    rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma, decision_function_shape = "ovo")
    rbfSVM.fit(pca_train_data[:10000], y_train[:10000])
    output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "decision_function_sh
    output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(rbfSVM.score(pca_train_data[:10000], y_train[:10000]
    output += "Test accuracy = " + str(rbfSVM.score(pca_test_data, y_test)) + "\n"
    print_and_log(output, log_file)
```

Best results are below:

- Linear Kernel, C = 0.1

Number of support vectors = [894 882 991 1009 948 973 839 856 827 899]
Elapsed time: 8.904108762741089

Train accuracy = 0.4403
Test accuracy = 0.3838

- Polynomial Kernel, Degree = 2, C = 10, Gamma = scale, coef0 = 0.5

Number of support vectors = [789 837 976 983 933 930 858 849 768 890]
Elapsed time: 7.057990789413452
Train accuracy = 0.8739
Test accuracy = 0.4577

- RBF Kernel, C = 1, Gamma = scale

Number of support vectors = [881 926 1003 1023 956 976 879 909 840 971]
Elapsed time: 5.395939111709595
Train accuracy = 0.6554
Test accuracy = 0.4708

- RBF Kernel, C = 10, Gamma = scale

Number of support vectors = [905 922 1004 1017 952 992 901 915 840 982]
Elapsed time: 7.345630645751953
Train accuracy = 0.9614
Test accuracy = 0.4709

- RBF Kernel, C = 100, Gamma = scale

Number of support vectors = [920 919 1000 1015 953 982 912 924 841 981]
Elapsed time: 8.281328201293945
Train accuracy = 0.9997
Test accuracy = 0.4556

- RBF Kernel, C = 1000, Gamma = scale

Number of support vectors = [920 919 999 1016 948 982 910 924 840 981]
Elapsed time: 8.17867136001587
Train accuracy = 1.0
Test accuracy = 0.4557

- RBF Kernel, C = 1, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [881 926 1003 1023 956 976 879 909 840 971]
Elapsed time: 5.391200542449951
Train accuracy = 0.6554
Test accuracy = 0.4708

- RBF Kernel, C = 10, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [905 922 1004 1017 952 992 901 915 840 982]
Elapsed time: 7.318029880523682
Train accuracy = 0.9614
Test accuracy = 0.4709

- RBF Kernel, C = 100, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [920 919 1000 1015 953 982 912 924 841 981]
Elapsed time: 8.248036623001099
Train accuracy = 0.9997
Test accuracy = 0.4556

- RBF Kernel, C = 1000, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [920 919 999 1016 948 982 910 924 840 981]
Elapsed time: 8.183026313781738
Train accuracy = 1.0

Test accuracy = 0.4557

2.11 PCA, 50000 Samples, Best Cases

```
C = 0.1
start_time = time()
linearSVM = SVC(kernel='linear', C=C)
linearSVM.fit(pca_train_data[:50000], y_train[:50000])
output = "Linear Kernel, C = " + str(C) + "\n"
output += "Number of support vectors = " + str(linearSVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(linearSVM.score(pca_train_data[:50000], y_train[:50000])
output += "Test accuracy = " + str(linearSVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

C = 10
gamma = 'scale'
degree = 2
coef0 = 0.5
start_time = time()
polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma, coef0 = coef0)
polySVM.fit(pca_train_data[:50000], y_train[:50000])
output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", Gamma = " + s
output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(polySVM.score(pca_train_data[:50000], y_train[:50000]))
output += "Test accuracy = " + str(polySVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

C = 10
gamma = 0.5
degree = 2
coef0 = 0.5
start_time = time()
polySVM = SVC(kernel='poly', C=C, degree=degree, gamma=gamma, coef0 = coef0, probability = T
polySVM.fit(pca_train_data[:50000], y_train[:50000])
output = "Polynomial Kernel, Degree = " + str(degree) + ", C = " + str(C) + ", Gamma = " + s
output += "Number of support vectors = " + str(polySVM.n_support_) + "\n"
output += "Elapsed time: " + str(time() - start_time) + "\n"
output += "Train accuracy = " + str(polySVM.score(pca_train_data[:50000], y_train[:50000]))
output += "Test accuracy = " + str(polySVM.score(pca_test_data, y_test)) + "\n"
print_and_log(output, log_file)

gamma = 'scale'
for C in [1, 10, 100, 1000]:
    start_time = time()
```

```
    rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma)
    rbfSVM.fit(pca_train_data[:50000], y_train[:50000])
    output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + "\n"
    output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(rbfSVM.score(pca_train_data[:50000], y_train[:50000]
    output += "Test accuracy = " + str(rbfSVM.score(pca_test_data, y_test)) + "\n"
    print_and_log(output, log_file)

gamma = 'scale'
for C in [1, 10, 100, 1000]:
    start_time = time()
    rbfSVM = SVC(kernel='rbf', C=C, gamma=gamma, decision_function_shape = "ovo")
    rbfSVM.fit(pca_train_data[:50000], y_train[:50000])
    output = "RBF Kernel, C = " + str(C) + ", Gamma = " + str(gamma) + ", decision_function_
    output += "Number of support vectors = " + str(rbfSVM.n_support_) + "\n"
    output += "Elapsed time: " + str(time() - start_time) + "\n"
    output += "Train accuracy = " + str(rbfSVM.score(pca_train_data[:50000], y_train[:50000]
    output += "Test accuracy = " + str(rbfSVM.score(pca_test_data, y_test)) + "\n"
    print_and_log(output, log_file)
```

Best results are below:

- Linear Kernel, C = 0.1

Number of support vectors = [4497 4277 4925 4949 4892 4794 4301 4379 4164 4321]
Elapsed time: 366.7933552265167
Train accuracy = 0.4206
Test accuracy = 0.4075

- Polynomial Kernel, Degree = 2, C = 10, Gamma = scale, coef0 = 0.5

Number of support vectors = [3630 3496 4590 4598 4564 4370 3997 3655 3382 3816]
Elapsed time: 290.8063061237335
Train accuracy = 0.7454
Test accuracy = 0.5421

- RBF Kernel, C = 1, Gamma = scale

Number of support vectors = [4065 4177 4791 4896 4728 4662 4243 4127 3738 4305]
Elapsed time: 234.4116370677948
Train accuracy = 0.65468
Test accuracy = 0.5402

- RBF Kernel, C = 10, Gamma = scale

Number of support vectors = [4094 4072 4743 4896 4626 4711 4265 4070 3739 4353]
Elapsed time: 283.5378005504608
Train accuracy = 0.9216
Test accuracy = 0.5624

- RBF Kernel, C = 100, Gamma = scale

Number of support vectors = [4070 4055 4735 4836 4651 4664 4343 4065 3810 4376]
Elapsed time: 378.86197662353516
Train accuracy = 0.99672
Test accuracy = 0.5442

- RBF Kernel, C = 1000, Gamma = scale

Number of support vectors = [4038 4057 4709 4835 4636 4647 4333 4063 3804 4376]
Elapsed time: 381.44525718688965
Train accuracy = 1.0
Test accuracy = 0.5386

- RBF Kernel, C = 1, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [4065 4177 4791 4896 4728 4662 4243 4127 3738 4305]
Elapsed time: 185.95598721504211
Train accuracy = 0.65468
Test accuracy = 0.5402

- RBF Kernel, C = 10, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [4094 4072 4743 4896 4626 4711 4265 4070 3739 4353]
Elapsed time: 227.3112096786499
Train accuracy = 0.9216
Test accuracy = 0.5624

- RBF Kernel, C = 100, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [4070 4055 4735 4836 4651 4664 4343 4065 3810 4376]
Elapsed time: 302.2709484100342
Train accuracy = 0.99672
Test accuracy = 0.5442

- RBF Kernel, C = 1000, Gamma = scale, decision_function_shape = "ovo"

Number of support vectors = [4038 4057 4709 4835 4636 4647 4333 4063 3804 4376]
Elapsed time: 305.47143054008484
Train accuracy = 1.0
Test accuracy = 0.5386

3 Issues

The first issue that was encountered was the size of the whole dataset. The 50.000 training data and the 10.000 test data could not be processed in order to take the full capability of every model. That's why at the start we chose to use only 5.000 samples for training and testing, that gave us a sense of the best options for the parameters of the model. But the best way to solve this issue was the use of PCA before training. This huge reduction of dimensions speed up the training process, so we can run models from 5.000 samples up to 50.000 samples in decent time and get the best result possible.

4 Conclusion

For the above results we can conclude that:

- CIFAR-10 dataset is a difficult dataset for classification and 3072 features needs too much computational power for 50.000 samples.
- PCA is a must methodology for preprocessing. Improves speed and accuracy.
- Sigmoid Kernel is not generally a good fit for SVM models.
- Polynomial and RBF Kernel have great results combined with PCA. The best result of every model was from a RBF Kernel SVC:

  RBF Kernel, C = 10, Gamma = scale
  Train accuracy = 0.9216
  Test accuracy = 0.5624

  RBF Kernel, C = 10, Gamma = scale, decision_function_shape = "ovo"
  Train accuracy = 0.9216
  Test accuracy = 0.5624

  Of course, we can see a pretty clear overfiting situation, but we know that for this dataset 56% test accuracy is really good, so overfiting is a logic result.