University of Bucharest

Faculty of Mathematics and Information Technology

Artificial Intelligence

Prediction based on accelerometer signals

Practical Machine Learning – Project 1

Student: Radu Madalin-Cristian

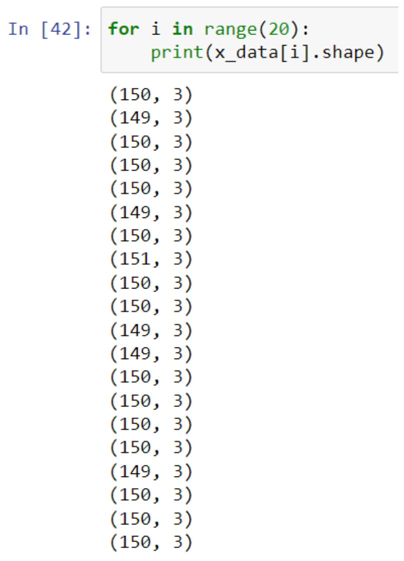
1) Introduction

We are asked to differentiate between 20 mobile device users. Each user has 450 signal recordings taken from their device's accelerometer. Each recording is 1.5 seconds in length taken at 100 Hz frequency, meaning that the final signal will have 150 values. Every entry in the signal has a value for the 3 axes: x, y, z.

Each training sample is assigned to 1 of 20 classes. The training set consists of 9000 labeled samples. The test set consists of another 5000 samples. The test labels are not provided with the data.

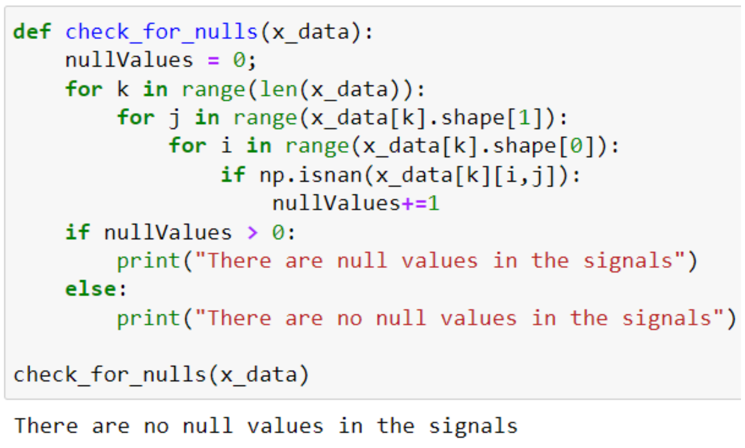
2) Exploratory data analysis and pre-processing

First, we want to make sure the signals are complete and consistent. Let's see the dimensions of the first signals and check for null values:



```
In [42]: for i in range(20):
             print(x_data[i].shape)

(150, 3)
(149, 3)
(150, 3)
(150, 3)
(150, 3)
(149, 3)
(150, 3)
(151, 3)
(150, 3)
(150, 3)
(149, 3)
(149, 3)
(150, 3)
(150, 3)
(150, 3)
(150, 3)
(149, 3)
(150, 3)
(150, 3)
(150, 3)
```



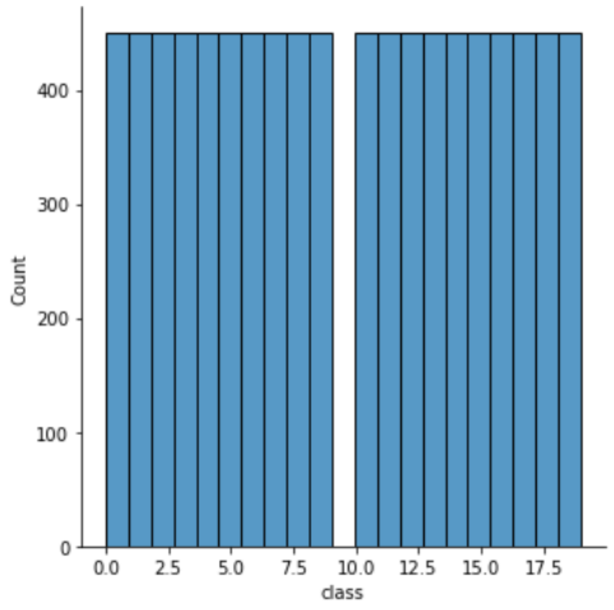
```
def check_for_nulls(x_data):
    nullValues = 0;
    for k in range(len(x_data)):
        for j in range(x_data[k].shape[1]):
            for i in range(x_data[k].shape[0]):
                if np.isnan(x_data[k][i,j]):
                    nullValues+=1
    if nullValues > 0:
        print("There are null values in the signals")
    else:
        print("There are no null values in the signals")

check_for_nulls(x_data)

There are no null values in the signals
```

As we can see, the signals do not have entries with null values, but a lot of them have less or more than 150 values. To make the entire set consistent, I trimmed the samples with more than 150 values and tried 2 approaches for the ones with less than 150 values. First, I filled the signals with the mean on each axis, and then I filled them with the last entry in the signal. The latter approach proved to be slightly better, so I kept it.

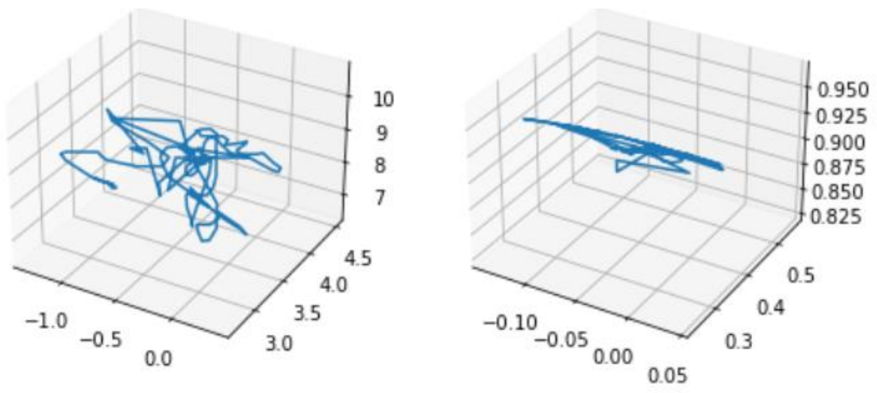
The target classes are integers from 1 to 20. To properly train the neural networks, I encoded these classes by shifting them to the left by 1, so that the values start from 0. Next, let's look at the distribution of the labels:

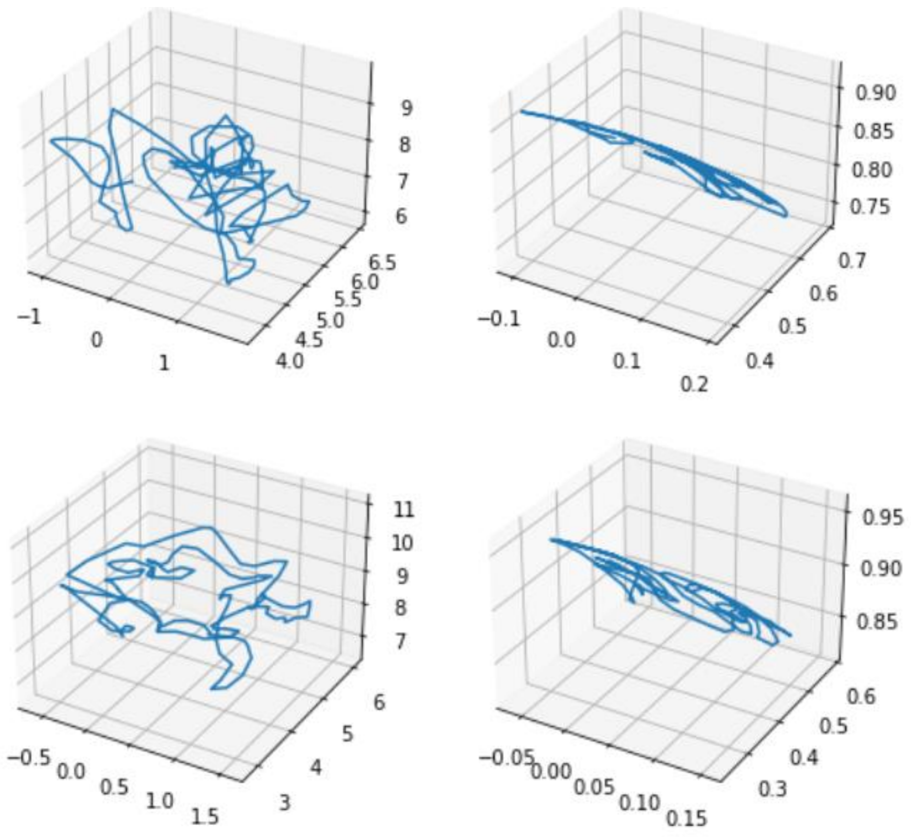


```
Number of occurences for each target class:
6      450
12     450
3      450
14     450
19     450
4      450
13     450
1      450
16     450
17     450
15     450
7      450
8      450
9      450
10     450
11     450
2      450
0      450
5      450
18     450
Name: class, dtype: int64
```

It looks like the classes are completely evenly distributed, which should come as no surprise, since they were created by taking 450 signals from 20 people.

Now, let's take a look at a few of the signals, put side by side with their normalized counterpart. For that, I plotted the first signal for classes 1, 2, and 3, in order:





It looks like there is no discernable pattern to be seen in the original signals. They also don't seem to suffer that much from noise. The normalized signals, though, show a slight difference in skewness and deviation. That being said, when I compared the difference in accuracy between the models I trained with the original signals and the ones I trained with normalized signals, I couldn't produce a noticeable difference, so I kept the original signals for future trainings.

3) Building and training the models

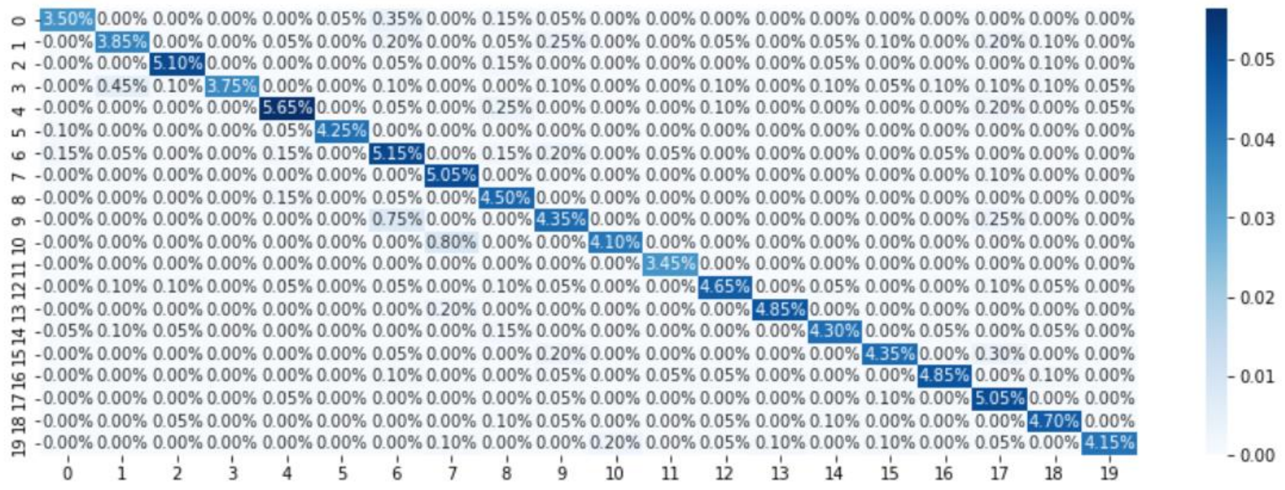
Since the input data is only numerical, I chose to implement the models using Support Vector Machine, Neural Network and Convolutional Neural Network.

For the first model, I used a neural network. I started by making a net with 3 hidden layers, the first having 16 neurons and the next having exponentially more neurons. Then I added a dropout layer and a classification layer. This model reached 85% accuracy in 300 epochs, with a validation accuracy of 83%. The test accuracy turned out to be a little over 51%, so the model needed some improvement. I used KFold to do 5-fold cross-validation and keras_tuners RandomSearch to look for the best hyperparameters. These are the ones I chose for tuning:

```
Search space summary
Default search space size: 5
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 25, 'step': 1, 'sampling': None}
units0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
initializer (Choice)
{'default': 'uniform', 'conditions': [], 'values': ['uniform', 'normal'], 'ordered': False}
activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'tanh'], 'ordered': False}
```

“num_layers” denotes the number of dense layers in the network. For each of these layers, RandomSearch also looks for the best option for kernel initializer and activation method. The final network has uniform initializer and rectify linear activation for all dense layers. The number of neurons in each layer is: 192, 352, 64, and 288. This model reached 91% accuracy in 300 epochs on the training set and 89% on the validation set:

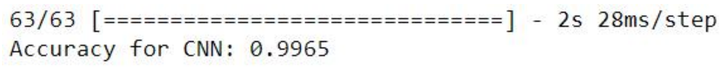
```
63/63 [==============================] - 0s 955us/step
0.896
```



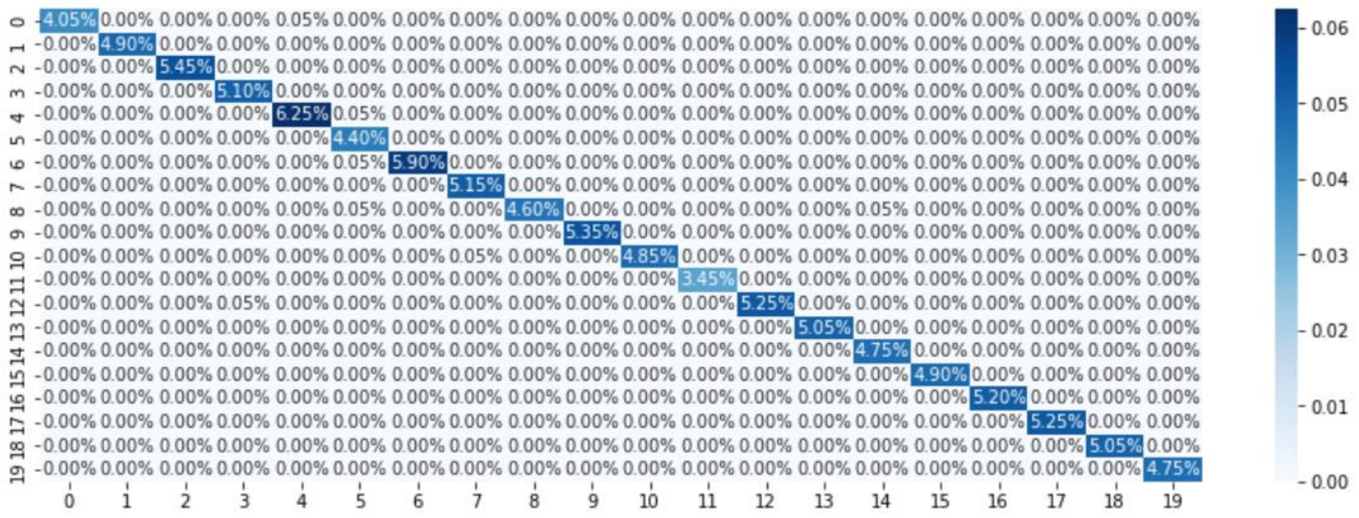
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	3.50%	0.00%	0.00%	0.00%	0.00%	0.05%	0.35%	0.00%	0.15%	0.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	0.00%	3.85%	0.00%	0.00%	0.05%	0.00%	0.20%	0.00%	0.05%	0.25%	0.00%	0.00%	0.05%	0.00%	0.05%	0.10%	0.00%	0.20%	0.10%	0.00%
2	0.00%	0.00%	5.10%	0.00%	0.00%	0.00%	0.05%	0.00%	0.15%	0.00%	0.00%	0.00%	0.00%	0.00%	0.05%	0.00%	0.00%	0.00%	0.10%	0.00%
3	0.00%	0.45%	0.10%	3.75%	0.00%	0.00%	0.10%	0.00%	0.00%	0.10%	0.00%	0.00%	0.10%	0.00%	0.10%	0.05%	0.10%	0.10%	0.10%	0.05%
4	0.00%	0.00%	0.00%	0.00%	5.65%	0.00%	0.05%	0.00%	0.25%	0.00%	0.00%	0.00%	0.10%	0.00%	0.00%	0.00%	0.00%	0.20%	0.00%	0.05%
5	0.10%	0.00%	0.00%	0.00%	0.05%	4.25%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
6	0.15%	0.05%	0.00%	0.00%	0.15%	0.00%	5.15%	0.00%	0.15%	0.20%	0.00%	0.05%	0.00%	0.00%	0.00%	0.00%	0.05%	0.00%	0.00%	0.00%
7	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.10%	0.00%	0.00%
8	0.00%	0.00%	0.00%	0.00%	0.15%	0.00%	0.05%	0.00%	4.50%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
9	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.75%	0.00%	0.00%	4.35%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.25%	0.00%	0.00%
10	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.80%	0.00%	0.00%	4.10%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
11	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	3.45%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
12	0.00%	0.10%	0.10%	0.00%	0.05%	0.00%	0.05%	0.00%	0.10%	0.05%	0.00%	0.00%	4.65%	0.00%	0.05%	0.00%	0.00%	0.10%	0.05%	0.00%
13	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.20%	0.00%	0.00%	0.00%	0.00%	0.00%	4.85%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
14	0.05%	0.10%	0.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.15%	0.00%	0.00%	0.00%	0.00%	0.00%	4.30%	0.00%	0.05%	0.00%	0.05%	0.00%
15	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.05%	0.00%	0.00%	0.20%	0.00%	0.00%	0.00%	0.00%	0.00%	4.35%	0.00%	0.30%	0.00%	0.00%
16	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.10%	0.00%	0.00%	0.05%	0.00%	0.05%	0.05%	0.00%	0.00%	0.00%	4.85%	0.00%	0.10%	0.00%
17	0.00%	0.00%	0.00%	0.00%	0.05%	0.00%	0.00%	0.00%	0.00%	0.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.10%	0.00%	5.05%	0.00%	0.00%
18	0.00%	0.00%	0.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.10%	0.05%	0.00%	0.00%	0.05%	0.00%	0.10%	0.00%	0.00%	0.00%	4.70%	0.00%
19	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.10%	0.00%	0.00%	0.20%	0.00%	0.05%	0.10%	0.00%	0.10%	0.00%	0.05%	0.00%	4.15%

The accuracy for the test set, however, was 55%. This was still too low, so I tried a different model.

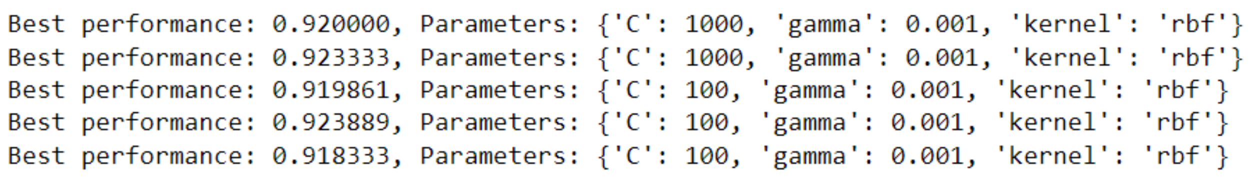
For my next approach, I used a convolutional neural network. The concept is similar to the regular neural network, except this model also has at least one convolution layer. The process for creating this model was pretty similar to the previous one. I started from a base network with 2 layers which gave an accuracy of 58% on the test set. After tuning, I obtained a network with 4 convolution layers, which have 96, 72, 48, and 112 filters and a kernel size of (3,3) for the first two layers and (5,5) for the next two layers. The activation for all these layers is rectify linear. After these four layers I added a pooling layer, a dropout layer, and a layer to flatten the data. Then, I put a dense layer with 256 neurons, then another dropout layer and finally, a dense layer with 20 neurons as the decision layer. The accuracy for this model was 99% for validation and almost 70% for test.



```
63/63 [==============================] - 2s 28ms/step
Accuracy for CNN: 0.9965
```



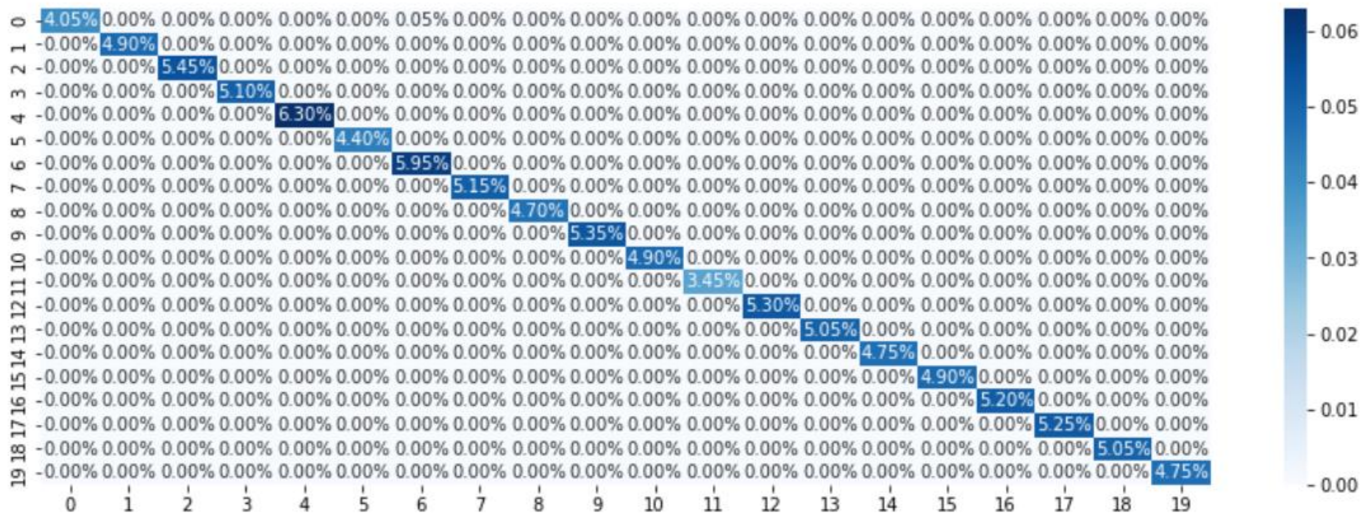
Finally, I tried to improve my test accuracy by implementing SVM. First, I flattened the matrices because SVM only accepts 1d arrays as input. Then I built the SVM model with the default parameters. This gave me a validation accuracy of 90%. I tuned the hyperparameters using grid search and obtained these values on each fold of the 5-fold cross-validation:



```
Best performance: 0.920000, Parameters: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
Best performance: 0.923333, Parameters: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
Best performance: 0.919861, Parameters: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
Best performance: 0.923889, Parameters: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
Best performance: 0.918333, Parameters: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
```

It looks like the best parameters are C=100, gamma=0.001, and kernel=rbf. This model produced the best results for test and validation: 85% and 99%:

```
Validation accuracy for SVM: 0.9995
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	4.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	0.00%	4.90%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2	0.00%	0.00%	5.45%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
3	0.00%	0.00%	0.00%	5.10%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4	0.00%	0.00%	0.00%	0.00%	6.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
5	0.00%	0.00%	0.00%	0.00%	0.00%	4.40%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
6	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
7	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.15%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
8	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	4.70%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
9	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.35%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
10	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	4.90%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
11	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	3.45%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
12	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
13	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
14	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	4.75%	0.00%	0.00%	0.00%	0.00%	0.00%
15	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	4.90%	0.00%	0.00%	0.00%	0.00%
16	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.20%	0.00%	0.00%	0.00%
17	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.25%	0.00%	0.00%
18	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.05%	0.00%
19	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	4.75%

4) Bibliography

- https://www.kaggle.com/competitions/pml-2022-smart/