Laboratory report, Machine Learning and Data Acquisition (November 2021)

Mattias Ahle, *Kristianstad University*

***Abstract*—In this laboratory work, the path from a first data collection, through various data analysis and pre-processing procedures, via different machine learning algorithms, a final, well performing neural network was found. The task was to predict different movements of a person, collecting information from a mobile phone in the person's pocket. The data was recorded with an application in the phone, then analysed, trained, and predicted in a computer. Python was the language of choice throughout the process where the K-nearest neighbour, the Softmax Regression, and the Recurrent Neural Network machine learning algorithms were explored. The main goal of this work was not to land in a perfect solution to the problems presented, but rather to be aware of every step taken in the process – to understand the steps taken from analysing, transforming, and discarding data to tweaking hyperparameters for the machine learning models. Being able to handle these intellectual and computational processes in a lean manner with the help of already existing Python libraries proved essential.**

I. Introduction

THIS laboratory work gradually ascended from the first data recordings to the final neural network prediction. Data on a person both being stationary and moving was collected. The objective of the laboratory work was to investigate how different machine learning algorithms, or models, would perform on this data, i.e., how well the models could predict unseen data, both in its raw form and after various pre-processing procedures.

Data recordings was to be performed with a mobile phone using an application able to record the phone's sensors' outputs, with selectable sample frequencies. For non-movement data, the person should stand, sit, and lie down, recording data from the accelerometer and magnetometer. For movement data, the person should run, walk, jump, and do push-ups, recording data from the accelerometer, magnetometer, and gyroscope.

Machine learning models to be explored in this work was K-Nearest Neighbour (KNN) [1], Logistic Regression using the Softmax function [2], also known as Softmax, or Multinomial Regression [3] and Recurrent Neural Networks (RNN) [4].

Performance of the different models were to be evaluated, mainly with regards to accuracy, but also with computing complexity in mind.

The data was to be evaluated with several aspects in mind to determine relevant features and hyperparameters for classification: numeric and graphical inspections, correlation and autocorrelation, mathematical and statistical transformations, and by running the data through machine learning models to determine which features contributed the most to a good model. The data could be put through a model in its raw form or pre-processed in some way. The pre-processing could consist of manual or statistical removal of outliers and corrupted data, removal of features deemed not relevant, statistical processing, scaling, etc.

II. Methodology

The collection of all data used in this work was done with a mobile phone [5], running the Android application Sensor Record [6]. To determine suitable sampling frequencies for all sensors, recordings were performed at $100Hz$ (a relatively high rate). To investigate if the sensors could handle the sample rate, the raw data was inspected for duplicate, neighbouring values. The recordings are stored in csv-files in the phone by Sensor Record. Those recordings were manually copied to a computer for further analysis.

The computer used was a laptop [7] running Windows 10. To analyse the data and run the machine learning algorithms Python 3.7.10 was used. The environment (see TABLE 1 for specifications) was created in Anaconda Navigator (anaconda3). Jupyter Notebook 6.4.3 and Visual Studio Code 1.61.2 were used as IDEs.

TABLE 1
PYTHON ENVIRONMENT SPECIFICATIONS

Package	Version
Pandas	1.2.4
Numpy	1.20.2
Matplotlib	3.1.2
Scikit-learn	0.24.2
Seaborn	0.11.1
Keras	2.3.1
Tensorflow	2.1.0

All recordings were performed with the mobile phone in the persons left, front pocket, with the display facing outwards and up-side-down in relation to the person standing up. For non-movement recordings, at least five seconds of data was recorded for each instance. For movement recordings, at least 10 seconds of data was recorded.

Date of submission: 2021-11-09. This laboratory work is a part of the course Machine Learning and Data Acquisition, course code: DT374B, at Kristianstad University, Sweden.

The author of this laboratory report, Mattias Ahle is a student at the Bachelor Programme in Computer Science with specialization in Internet of Things, Kristianstad University, (e-mail: mattias.ahle0081@stud.hkr.se).

Several recordings were made for each class, where one of the recordings were set aside for testing purposes. For the movement recordings, two extra recordings were made with all movements conducted consecutively in one recording, yielding an extra test set designed to resemble a real-world case. The first of these recordings were performed with the same trousers as the training sets – a pair of loose-fit shorts, and the second recording was performed with another set of trousers with tighter pockets.

Corrupted data at the beginning and end of each recording was manually removed.

For movement data, scaling [8] [9] was applied to see if the predictions could be improved.

For all raw data, the angle, θ, between the vectors v and a selected axis of the vector, v_{axis}, (z-axis for the accelerometer) (1) together with the magnitude, $\|v\|$, of the vectors (2) were calculated. The same mathematical operations were performed on the magnetometer vectors with the minor difference that the angle was calculated against the y-axis instead.

$$\theta = \cos^{-1}\left(\frac{v_{axis}}{\|v\|}\right) \tag{1}$$

$$\|v\| = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{2}$$

All raw and as above transformed data was inspected with a Seaborn pairplot [10], in attempts to narrow in on the best features for classification. The more spread between the features' data points, the more likely they are good candidates for a classification.

With the same principle in mind – good spread of data – as in the case with the raw and transformed data, the relationship of each data point, x_i, with regards to its n neighbors over time, were of high interest. Upon exploring the movement data, the sample standard deviation (3) surfaced as a possible contender for best feature, as it, given a relevant sample size, n, will represent a spread in a sample of data over time. The bumpier the movement – the larger the standard deviation.

$$s_x = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}} \tag{3}$$

A suitable value for n, covering at least one entire movement for all classes, was decided by the movement with the lowest frequency. Manual inspection of the graphs of the different features was first conducted to find the sensor with the greatest spread of values. The standard deviation values for those features were then inspected with the goal of further narrowing down the number of features.

On non-movement data, a KNN classifier [11] was explored. Using GridSearchCV [12] from Scikit-learn's model selection library, the hyperparameter k could be tested in a practical manner. A range of $1-60$ was used in the training of the KNN classifier. GridSearchCV was also used for cross validation [13].

On movement data, KNN, Softmax Regression [14] and RNN were explored. GridSearchCV was used in same manner as for non-movement data training for KNN and Softmax Regression.

RNN was implemented using the Keras model Sequential [15]. The time variable, i.e., the chunk size of each input instance to the model, was altered and tested in powers of two. Other hyperparameters tweaked were the following: type of neuron (LSTM and GRU [16]), number of neurons per layer, number of layers, and dropout [17]. When fitting [18] the training data to the model, two more parameters were tweaked: cross validation chunk size and if the data should be shuffled or not. Testing different hyperparameters for the RNN classifiers was done manually. The Softmax function was used as output layer activation function, training optimizer was set to Adam [19], and as loss function categorical cross entropy [20] was used.

At an early stage, the magnetometer data was omitted from the movement data collection. The magnetometer only holds information of the direction of the magnetic north in relation to the sensor position. This information was deemed not relevant in the quest for classifying movement.

III. Results

The sensors in the mobile phone had a capacity of a sampling frequency of $100Hz$.

A first test with KNN on non-movement data, using both accelerometer and magnetometer data, gave 100% accuracy for $k = 1-60$. Narrowing in on single features, e.g., x-axis data for the accelerometer, also gave 100% accuracy. This held true for all features when used as standalone features.

Movement data was also tested with a KNN classifier. The movements were a bit harder for a KNN to classify. Utilizing all six remaining features after dropping the magnetometer gave an accuracy of 88% on the test set.

The movement data was scaled to see if the KNN predictions could improve. The scaling resulted in no improvement in most cases, but major deterioration in other. All data showed similar results. In TABLE 2 a representative outtake of the tests is shown.

TABLE 2
DATA SCALING IMPACT

Scaler	Training set accuracy	Test set accuracy
No scaler	97%	88%
StandardScaler	98%	88%
MinMaxScaler	99%	46%

The correlation between the features were calculated. The result is shown in TABLE 3.

TABLE 3
MOVEMENT FEATURE CORRELATION

Feature	ax	ay	az	gx	gy	gz
ax	1.00	-	-	-	-	-
ay	0.11	1.00	-	-	-	-
az	-0.14	-0.22	1.00	-	-	-
gx	-0.06	0.00	0.03	1.00	-	-
gy	0.14	0.07	-0.18	0.40	1.00	-
gz	-0.02	-0.08	-0.13	0.23	0.26	1.00

ax, ay, az = accelerometer values, gx, gy, gz = gyroscope values

To see how the errors were distributed between the four classes, a confusion matrix [21] was created. Fig. 1 shows that the run class was the hardest class for the KNN classifier to classify, being confused a lot with jump and quite a lot with walk.

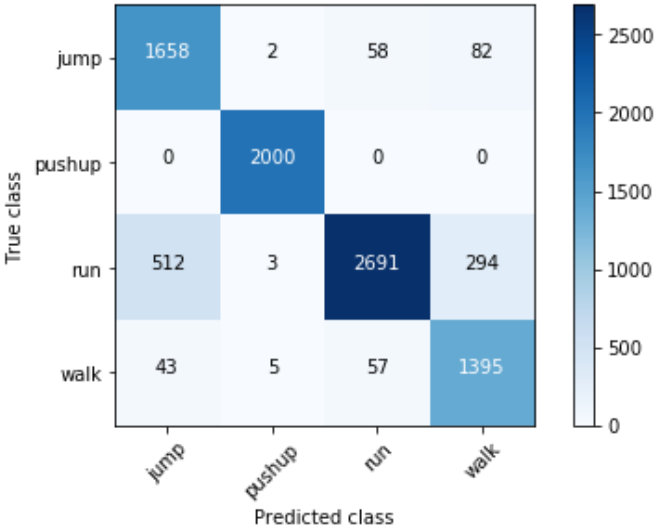

Fig. 1. Confusion matrix showing predictions on the movement test set using a trained KNN classifier.

The standard deviation approach was now investigated. Push-ups was observed having the lowest frequency of all movements: around two seconds. The sample size, n, for calculating standard deviation, was selected to $100Hz \cdot 2s = 200$ data points.

TABLE 4
STANDARD DEVIATIONS FOR THE ACCELEROMETER

Movement	x	y	z
Jump	3	11	9
Push-up	1	1	1
Run	10	19	18
Walk	7	7	7
Standard deviation	4.0	7.5	7.0

TABLE 4 shows the standard deviation values for each feature and class for the accelerometer values. It also shows the standard deviation within the standard deviations of each feature.

The confusion matrix of the same data as in Fig. 1 can be seen in Fig. 3. The difference is that the data in Fig. 3 is pre-processed by calculating the standard deviation.

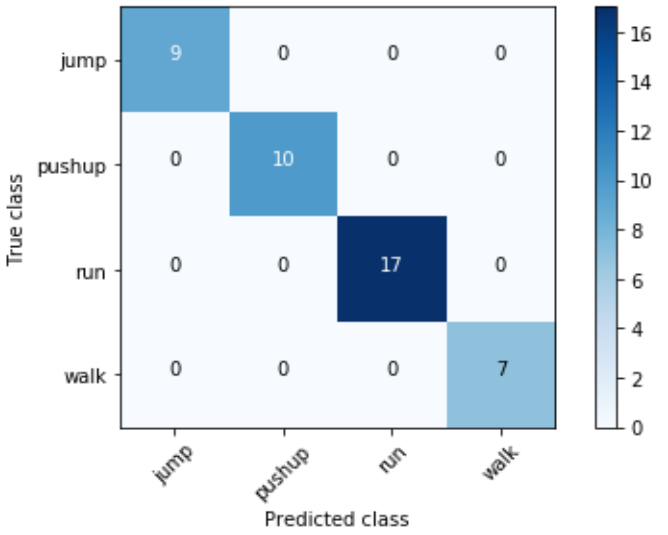

Fig. 3. Confusion matrix showing predictions on the standard deviation test set for movement using a trained KNN classifier.

The same classifier used in Fig. 3 was also tested on the consecutive data. The result is shown in Fig. 2. A Softmax Regression classifier was also tested and performed similarly to the KNN classifier.

In Fig. 2 and Fig. 4 the first half of the plot represents loose fit trousers, i.e., what the classifier is trained on, and the second half represents tight fit trousers. The true classes are written along the accelerometer y-axis standard deviation plot (stdev). The colours represent the predicted classes. The stdev-plot is plotted as a reference to the data used to predict the classes.

Fig. 4 shows the result of an RNN classifier on the same data as above. The hyperparameters mentioned in the Methodology chapter were tweaked to obtain a satisfying result. The setup yielding the result in Fig. 4 can be seen in TABLE 5, where the dropout was set to 0.1, a validation set of 10% was used, and the input values were shuffled.

TABLE 5
NEURAL NETWORK SETUP

Layer	# Neurons	# Params
LSTM	256	269,312
Dropout	256	0
Dense	4	1,028

The model converged rather fast during training, i.e., this setup did not create a computationally heavy situation. The convergence can be seen in Fig. 5.

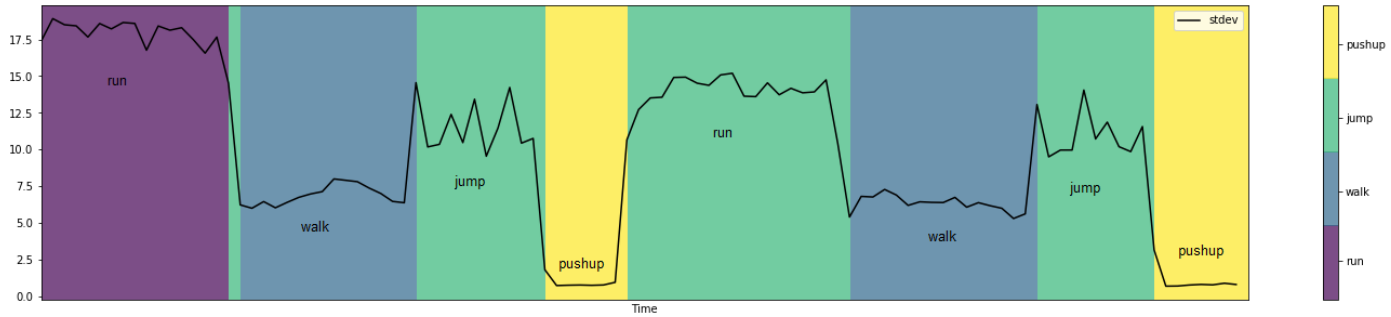

Fig. 2. KNN predictions on consecutive movements.

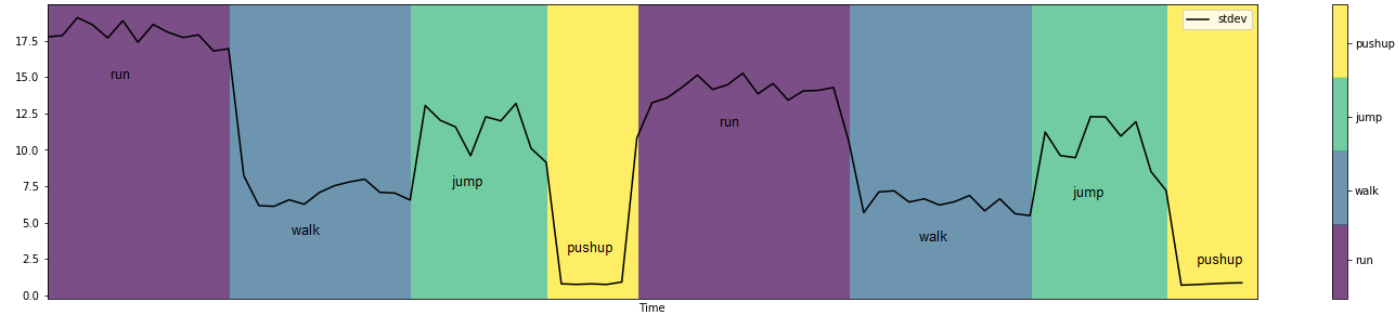

Fig. 4. RNN predictions on consecutive movements.

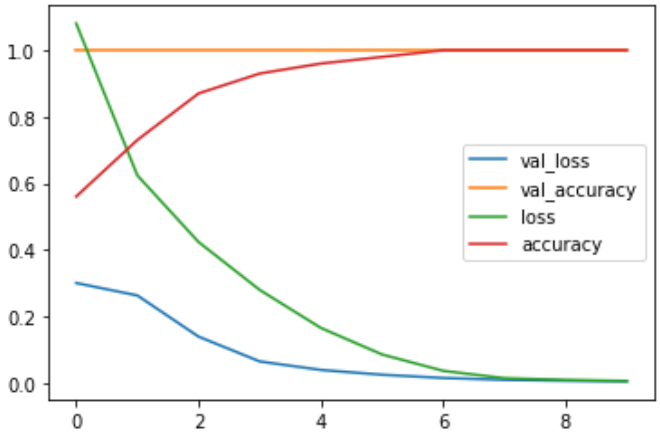

Fig. 5. Training history of the final RNN model. Epochs on the x-axis.

IV. Discussion

As all recordings were performed with a sample rate of $100Hz$, no doors were closed regarding future data utilization.

The first run with raw non-movement features on a KNN classifier resulted in 100% accuracy. Upon a pairplot inspection of the data, the data was well separated in between classes, thus making it easy for the KNN classifier. An example of well separated, but not perfect, data can be seen in Fig. 6.

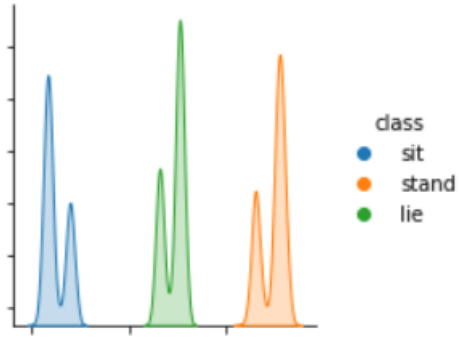

Fig. 6. The data of three classes well spread within one feature.

In the quest for keeping things simple, the features of the transformed data – angle and magnitude – could be omitted as they did not add any value to the classification problem, as the classifier had no problems classifying using only one feature's raw data. The path of least resistance is always favourable.

Upon achieving 100% accuracy for non-movement data, no further analysis of this data was necessary.

When attempting the same approach on movement data, the accuracy started to drop for the KNN algorithm. Inspecting the pairplot revealed the reason behind this phenomenon: The classes' data was not at all as well separated as in the case of the non-movement data. A representative outtake from the pairplot can be seen in Fig. 7.

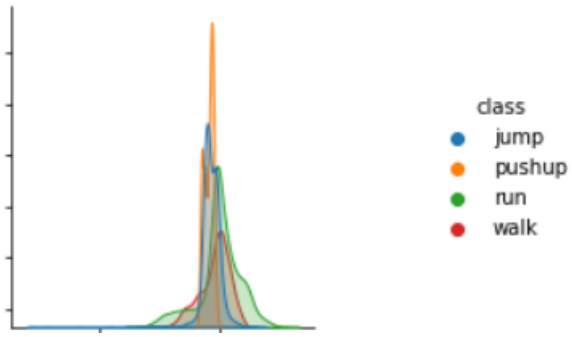

Fig. 7. The data of four classes not well spread within one feature.

Adding scaling to the problem did not contribute to any improvement and in some cases significantly worsened the results. An assumption can be made that the real-world scale of the sensor output in most cases has no relevance, but in some cases, it is of importance. Either way, scaling could not contribute to any improvements and was omitted for the remainder of this work.

As seen in TABLE 3, the correlation in most cases between the features are low to very low. This indicates that all features provide their own unique information to a classifying problem, thus are desirable to keep, if used in their raw form.

The KNN classifier seemed to have met its limits on predicting movement using both the accelerometer and the gyroscope raw values. The confusion matrix in Fig. 1 reveals some interesting facts: The biggest confusion, i.e., most incorrect classification, can be seen in the run class, and it is mostly confused with the jump class, but also a significant amount with the walk class. Hardly any confusion can be seen with the pushup class. In real life, for a human being, differentiating between a person running or a person jumping is not hard, but if you think about the forces created by those movements on a phone in the persons pocket, those movements have more in common than first meets the eye. Confusing run with walk is an easy mistake to make, even for a human.

It seemed like KNN could differentiate between the different movements regarding their influence on the mobile sensors in space, but not in time. Time needed to be added to the equation. Was this possible for a KNN classifier which only looks in space for information?

Statistical exploration revealed an answer to the missing time variable. Standard deviation was now emerging as a strong contender. The values seemed to be well spread-out. It has already been stated that well spread-out values are important in a classification context. TABLE 4 indicates that we might even be well off with only one feature. If we look at the feature with the greatest internal standard deviation, y, we see the values 1, 7, 11, and 19 each representing a class. In comparing Fig. 1 with Fig. 3, the improvement is obvious. Not only is the accuracy much better, the time for training the model has massively decreased due to the simplification of the classification problem.

Looking at the result in Fig. 2, a problem can be seen. The KNN classifier had no problem predicting data recorded with the same trousers as the training data but confused tight-trousers-running with loose-trousers-jumping. The standard deviation plot reveals the reason. Running with tighter trousers impacts the phone in the pocket in a similar way to jumping. A question arose: Could another classifier handle this problem?

The solution to the miss-classification of running with tight trousers lied in a neural network. As seen in Fig. 4, all classes where correctly classified. Not surprisingly, the size of each input had to be at least 200 data points, just as in the cases earlier described. A guess to why the neural network succeeded where a KNN or Softmax Regression failed, is that the RNN had more parameters to optimize, and the underlaying learning mechanisms where much more advanced than the other models.

V. Conclusion

As both simpler and more complex classification problems have been explored in this lab, it has been made clear that not overdoing things is good guidance, as always, in problem solving situations. If the problem is simple to start with, maybe not an advanced neural network is the best solution, even if it may work. Maybe a couple of if-else statements would suffice.

Some light has also been shined on another insight during this work. Under the surface of a seemingly complex problem, a simple solution might be hidden. In the process of breaking down a problem, one's mind should always be open to the simple solutions. Sometimes such a solution might appear, other times not, but when they appear, they often seem obvious. Sometimes too obvious, and the risk of discarding them due to their simple nature is always present.

The problem of obtaining an optimal setup for a machine learning model, including the most optimal features and hyperparameters, grows exponentially for each added variable. Even for a reasonable sized problem, the computational complexity quickly grows very large if not some level of human know-how is introduced into the process. Therefore, it is necessary to have some feature, or general problem, knowledge. It is of great help in the process of narrowing in on the most optimal solution without having to indulge in time consuming, maybe unrealistic, computational endeavours.

References

[1] "k-nearest neighbors algorithm," Wikipedia, 24 10 2021. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. [Accessed 03 11 2021].

[2] "Softmax function," Wikipedia, 26 10 2021. [Online]. Available: https://en.wikipedia.org/wiki/Softmax_function. [Accessed 03 11 2021].

[3] S. Raschka, "What is Softmax Regression and How is it Related to Logistic Regression?," KDnuggets, 07 2016. [Online]. Available: https://www.kdnuggets.com/2016/07/softmax-regression-related-logistic-regression.html. [Accessed 03 11 2021].

[4] "Recurrent neural network," Wikipedia, 13 10 2021. [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network. [Accessed 03 11 2021].

[5] "Mi 9 SE Specifications," Xiaomi, [Online]. Available: https://www.mi.com/global/mi-9-se/specs/. [Accessed 03 11 2021].

[6] M. Golpashin, "Sensor Record," Google Play, 21 08 2018. [Online]. Available: https://play.google.com/store/apps/details?id=de.martingolpashin.sensor_record&hl=en_US&gl=US. [Accessed 03 11 2021].

[7] "Dell Vostro 3583 Setup and specifications guide," Dell, [Online]. Available: https://www.dell.com/support/manuals/sv-se/vostro-15-3583-laptop/vostro_3583_setupandspec/system-specifications?guid=guid-ab67ed37-0818-4592-a25c-f04b3a73c18d. [Accessed 03 11 2021].

[8] "sklearn.preprocessing.StandardScaler," Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. [Accessed 04 11 2021].

[9] "sklearn.preprocessing.MinMaxScaler," Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. [Accessed 04 11 2021].

[10] M. Waskom, "seaborn.pairplot," Seaborn, [Online]. Available: https://seaborn.pydata.org/generated/seaborn.pairplot.html. [Accessed 04 11 2021].

[11] "sklearn.neighbors.KNeighborsClassifier," Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed 04 11 2021].

[12] "sklearn.model_selection.GridSearchCV," Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accessed 03 11 2021].

[13] "Cross-validation (statistics)," Wikipedia, 28 10 2021. [Online]. Available: https://en.wikipedia.org/wiki/Cross-validation_(statistics). [Accessed 03 11 2021].

[14] "sklearn.linear_model.LogisticRegression," Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Accessed 04 11 2021].

[15] "The Sequential class," Keras, [Online]. Available: https://keras.io/api/models/sequential/. [Accessed 04 11 2021].

[16] M. Phi, "Illustrated Guide to LSTM's and GRU's: A step by step explanation," towards data science, 24 09 2018. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Accessed 03 11 2021].

[17] A. G, "A review of Dropout as applied to RNNs," 3 6 2018. [Online]. Available: https://adriangcoder.medium.com/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b. [Accessed 03 11 2021].

[18] J. Allaire and F. Chollet, "Train a Keras model," RStudio, Google, [Online]. Available: https://keras.rstudio.com/reference/fit.html. [Accessed 04 11 2021].

[19] "Adam," Keras, [Online]. Available: https://keras.io/api/optimizers/adam/. [Accessed 09 11 2021].

[20] "Probabilistic losses, CategoricalCrossentropy class," Keras, [Online]. Available: https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class. [Accessed 09 11 2021].

[21] "Confusion matrix," Wikipedia, 4 07 2021. [Online]. Available: https://en.wikipedia.org/wiki/Confusion_matrix. [Accessed 04 11 2021].