SCHOOL OF MECHATRONIC SYSTEMS ENGINEERING

SIMON FRASER UNIVERSITY

MSE 491: Introduction to Machine Learning

Final Project:

Design and Implementation of Audio Recognition ML Algorithms

April 10, 2021

Group number: 19

Tim Saxon	301 339 761	50% Contribution
Kyle Christie	301 357 617	50% Contribution

Introduction

The goal of this project was to develop machine learning models catered to audio classification. UrbanSound8K, a public dataset of common and distinctive sounds found in urban environments, was used to train and test various machine learning models. Audio classification can have very useful applications in mechatronic systems, especially autonomous systems like self-driving cars and robots that must analyze their surroundings with simple sensors. These systems could also be used to identify when and where an event took place by cross referencing audio recordings and identifying the similar patterns across them. Using audio classification in unison with other classification techniques like image classification, autonomous systems can be made to function with extreme safety and precision, without the need for constant human intervention.

Methods

UrbanSound8K is a publicly available dataset of 8732 samples belonging to ten general classes, which correspond to ten common noises found in an urban environment. These classes can be described and shown in table 1 and figure 1.

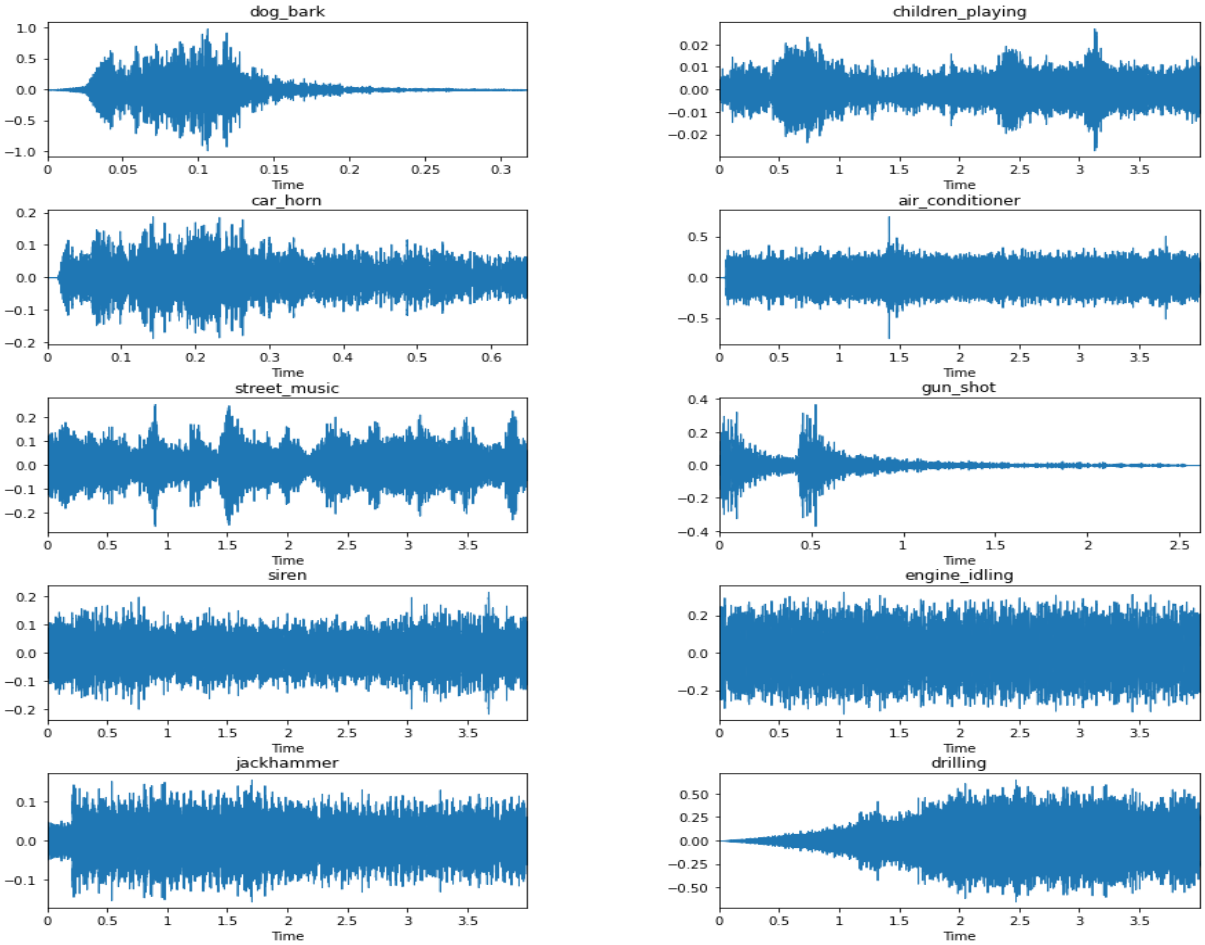

Figure 1. Example waveforms of each class

Table 1: Data Classes

Label	Description
0	Dog barking
1	Car horn
2	Street music
3	Siren
4	Jackhammer
5	Children playing
6	Air conditioner
7	Gun shot
8	Engine idling
9	Drilling

The time-domain waveforms (fig. 1) are not always a perfect estimate of any class. The recordings both in the dataset and in a real world application will often vary in both amplitude and length, i.e., volume and time recorded. The distance from the noise being recorded, type of device used for recording and/or any audio pre-processing on the device and many more variables can also affect the time-domain samples.

Analysing the audio signals in the frequency-domain gives a much greater understanding of each sample and class. A feature known as Mel Frequency Cepstral Coefficients (MFCCs) can be extracted and used as the defining characteristic between classes. The Mel scale relates a signal's actual frequency to that perceived by the human ear. The perceived frequency of a signal can be calculated from [1]:

$$Mel(f) = 2595log(1 + \frac{f}{700}) \quad (1)$$

where f is the actual frequency. The general procedure for determining an audio signals MFCCs is as follows: split the system into short windows and take each window's Fourier transform; map the transform onto the Mel scale with overlapping triangular windows, that essentially quantize the data; take the log of the powers at each Mel frequency, apply the discrete cosine transform of the logs; yielding the MFCCs as the amplitudes of the resulting power spectrum. The python library Librosa is used for audio analysis, and provides a useful method of quickly preprocessing the data and extracting features [2].

Applying the process above [3] divides each sample into 40 MFCCs, essentially splitting the energies over each sample's frequency response into 40 categories across the audible frequency spectrum. Each MFCC is a numerical value describing how much can be heard in its specific frequency range.

Results and Discussion

With the features extracted from each sample as MFCCs, the data was available for use to train various machine learning models. To evaluate the efficiency of each model, a performance factor was calculated as follows:

$$P_f = \frac{test\ score}{run\ time} \quad (2)$$

The run time used in the performance factor calculation was evaluated from the start of the model creation to after both train and test scores were calculated.

Part 1: Neural Networks

1.1 Simple Feed Forward NN

The feed forward neural networks (NN) used keras.Sequential() [4], with the number of hidden layers and their properties varied. All the feed forward models analysed contained an input layer of the same width as the input feature vector (40) as well as an output layer with the same shape as the label vector (10). The resulting performance for the multiple feed forward NNs can be found in table 2.

Table 2: Results of Simple Feed Forward NNs

Epochs	Hidden Layers	Nodes Per Hidden Layers	Training Score	Testing Score	Run Time	Performance Factor
50	2	128, 256	85.73 %	80.37 %	19.85 s	4.04
100	2	128, 256	89.13 %	84.09 %	47.38 s	1.77
1000	2	128, 256	94.04 %	86.78 %	527.18 s	0.16
50	2	256, 256	90.39%	86.15 %	22.79 s	3.78
100	2	256, 256	92.20 %	86.20 %	30.98 s	2.78
1000	2	256, 256	97.57 %	90.15 %	439.85 s	0.20
50	3	128,256,256	81.07 %	77.73 %	20.19 s	3.85
100	3	128,256,256	85.97%	81.51%	35.01 s	2.33
1000	3	128,256,256	90.31%	85.00%	347.86 s	0.24
100	3	256, 256, 256	88.86 %	84.54 %	38.22 s	2.21

Increasing the number of layers and/or number of nodes per layer beyond those shown did not provide any more accurate results, and instead began to reduce the testing accuracies of the models. The most accurate model used 100 epochs and a moderate number of nodes per hidden layer, with a testing accuracy of over 90%; however, the model with the overall best performance factor was the simplest model, which although less accurate took far less computational time.

1.2 Multi-Layer Perceptron NN

A multi-layer perceptron NN is very similar to the forward feed NNs examined above, but with added backpropagation. This allows the model to feed information forwards and backwards, and update the weights of each node much more efficiently, ultimately increasing the efficiency. In the Scikit-Learn documentation for the MLPClassifier(), it is implied that there are 2 additional layers aside from the hidden layers defined, which are most likely the input and output layers [5]. Table 3 shows the results of a variety of scenarios of applying the multilayer-perceptron NNs.

Table 3: Results of Multi-Layer Perceptron NNs

Hidden Layers	Nodes Per Layers	Training Score	Testing Score	Run Time	P_f
2	128, 256	99.60 %	86.83 %	23.50 s	3.69
2	256,256	99.64 %	89.47 %	26.44 s	3.38
3	128, 256, 10	99.37 %	88.78 %	43.51 s	2.05
3	128, 256, 256	99.11 %	89.70 %	33.58 s	2.67
3	40, 256, 256	98.27 %	86.72 %	30.34 s	2.86
3	256, 256, 10	0.00 %	0.00 %	15.69 s	Failed
3	256, 256, 256	99.70 %	91.36 %	34.64 s	2.64
3	128, 256, 512	99.71 %	90.67 %	121.15 s	0.748
4	256, 256, 256, 256	98.74 %	90.04 %	52.87 s	1.70
4	128, 128, 256, 256	97.72 %	88.61 %	35.04 s	2.53
4	128, 256, 256, 256	99.13 %	90.10 %	47.94 s	1.88

Similarly to the feed forward NNs, the most accurate model was one with a middling amount of nodes per layer. This model also had three hidden layers, and resulted in an accuracy of over 91%. The model with the highest performance factor was again the simplest NN, with almost 87% accuracy and a low run time. Figure 2 gives the confusion matrices for the best models of each of feed forward NN and MLP NN.

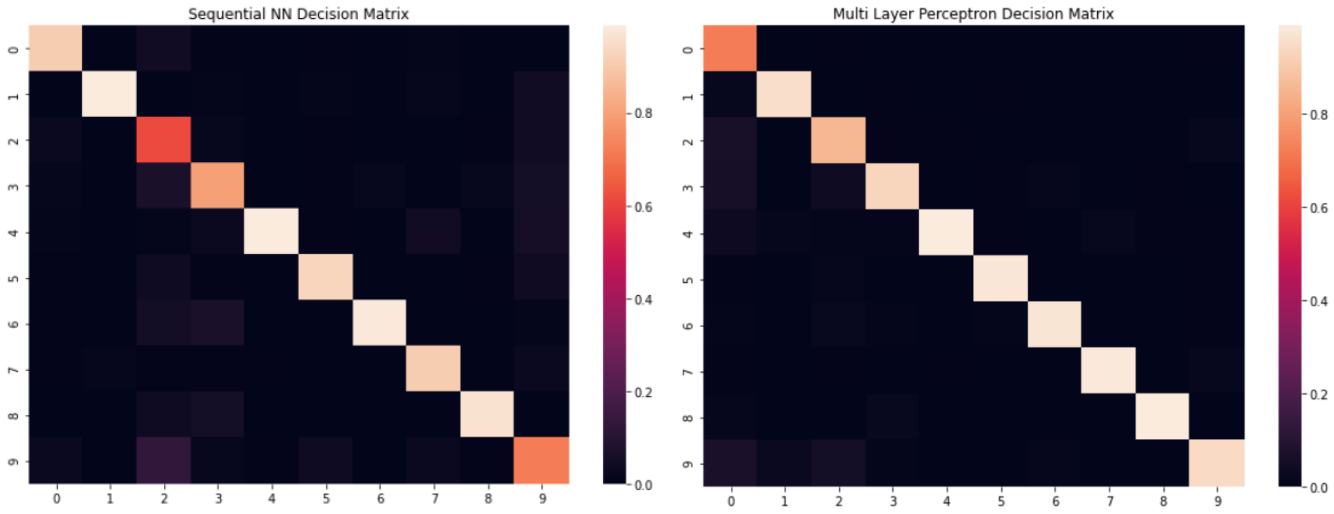

Figure 2: Neural network confusion matrices from best results

Part 2: Support Vector Machines

The support vector machine (SVM) algorithms do not accept the one-hot encoded target vector used on the other models; however, SVM does accept the string target vector (simply not encoded, thus only containing the related label instead of an array of all the labels as 0's and 1's). This allowed for evaluating the efficacy of SVM in a multi-class scenario. Three methods of SVM were used: SVC [6], NuSVC [7] and LinearSVC [8]. SVC is a standard SVM procedure, NuSVC is similar to SVC but uses a parameter to limit the number of support vectors, and LinearSVC is similar to SVC but using a different library.

Table 4: Results of Various SVM Models

Algorithm	Settings	Train Score	Test Score	Run Time	P_f
SVC	Default	90.71 %	87.58 %	3.52 s	24.88
NuSVC	Default	76.96 %	74.53 %	5.93 s	12.56
LinearSVC	Max_iter=5*default	58.87 %	57.36 %	4.33 s	Failed to converge

LinearSVC failed to converge, but SVC and NuSVC were both successful. SVC was superior in every category; it weighed in with the highest test score of 87.58% with the quickest run time of 3.52 seconds. The results of the SVM classification are shown graphically for the best and worst cases in figure 3.

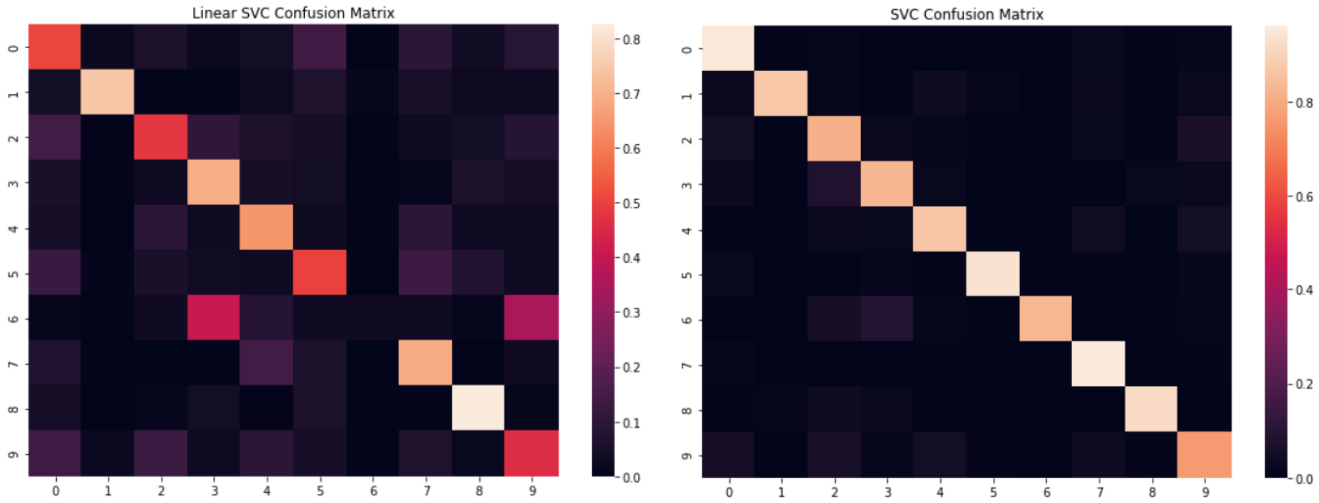

Figure 3: Confusion matrices for worst and best SVM classifiers

Part 3: Decision Trees

Decision tree classifiers do accept one-hot encoded labels, and could be evaluated with the same methods as the neural networks. Two types of tree classifiers, DecisionTreeClassifier [9] and ExtraTreesClassifier [10] were tested, and both showed little to no difference in performance when adjusting the algorithm parameters. Figure 4 plots the confusion matrices for decision tree and extra trees classifiers and table 5 contains the results.

Table 5: Results of Decision Tree Models

Algorithm	Settings	Train Score	Test Score	Run Time	P_f
Decision Tree	Auto	100 %	70.46 %	0.44 s	160
Extra Trees	Auto	100 %	65.20 %	0.04 s	1630

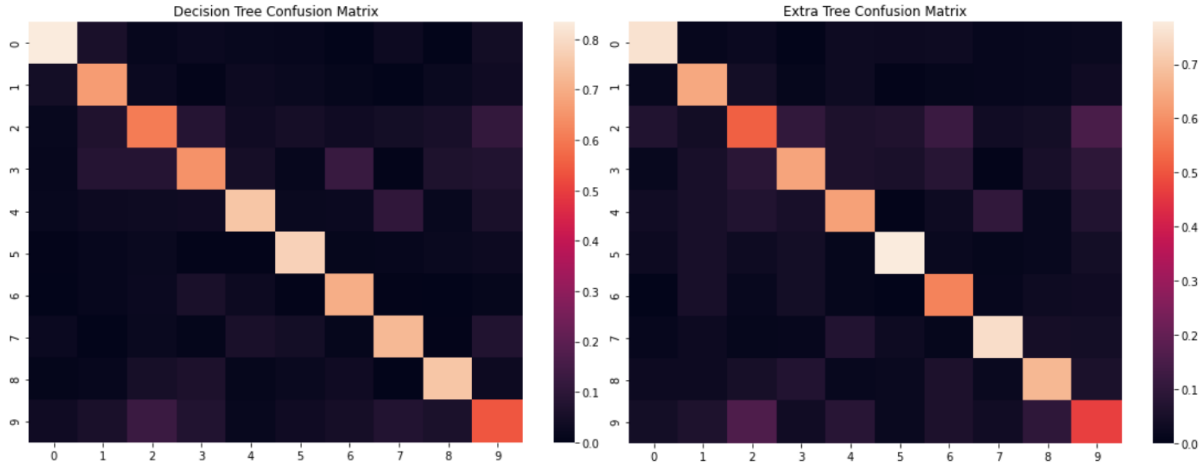

Figure 4. Confusion matrices for the decision tree classifiers

While the decision trees ran extremely fast, resulting in extremely high performance factors, the testing accuracies obtained were not good enough to justify the use of the algorithms for the analysis of new data. However, the training scores obtained were the best out of any models tested, with perfect scores at 100% accuracy. In cases where input data is identical to the data on which the model was trained, decision trees would be an exceptional solution.

Part 4: KNeighbors

The final multi-class, multilabel (one-hot) classifier tested was KNeighbors [11]. The model was generated with all points in each neighborhood weighed equally. The largest change in results was observed when varying the number of neighbors. The results were as follows in table :

Table 6: Results of KNeighbors Models

# Neighbors	Train Score	Test Score	Run Time	P_f
1	100 %	94.39 %	2.05 s	46.04
2	93.60 %	87.35 %	3.63 s	24.14
3	95.85 %	90.96 %	4.00 s	22.74
4	90.63 %	85.35 %	4.41 s	20.55
Default = 5	92.20 %	87.69 %	4.47 s	19.62
10	77.98 %	75.44 %	4.83 s	15.62

Setting the number neighbors to one provided incredible results, with a perfect training score of 100% and the highest test score achieved out of any of the tested classifiers at 94.39%. Not only were the results impeccable, but the time it took to achieve them was remarkably low at just over 2 seconds. The confusion matrix for this optimal trial is shown in figure 5.

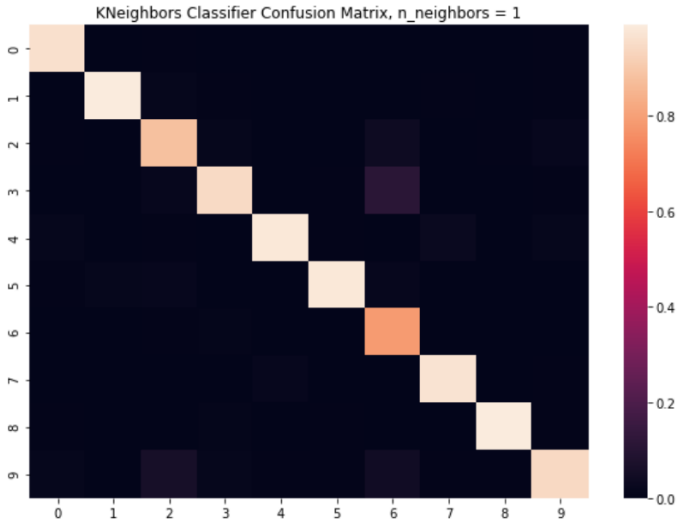

Figure 5. Confusion matrix for KNeighbors classifier with one neighbor

Final Comparison

Of all the classification methods tested, each method had their own pros and cons. To more easily gauge the performance of each classifier's most accurate cases, the data of the best model of each algorithm is summarized below (table 7):

Table 7: Results of the Best Models from Each Approach

Classifier	Train Score	Test Score	Run Time	P_f
Sequential NN	97.57 %	90.15 %	439.85 s	0.20
MLP NN	99.70 %	91.36 %	34.64 s	2.64
SVM	90.71 %	87.58 %	3.52 s	24.88
Decision Tree	100 %	70.46 %	0.44 s	160
KNeighbors	100 %	94.39 %	2.05 s	46.04

It can be determined from the table above that KNeighbors was easily the best classifier when analysing the MFCC features from each audio sample. Both sequential and multi-layer perceptron neural network models were a close second in terms of accuracy; however, they lacked computational speed, being the slowest algorithms of any tested.

Conclusion

Identification of common city sounds was achieved to varying degrees of success using a multitude of machine learning algorithms. Models falling under the umbrellas of feed forward NNs, multi-layer perceptron NNs, support vector machines, decision trees and K neighbours classifiers were included. The most accurate model for this audio classification application was K neighbours, with 94.39% accuracy on the test data, and a quick run time as well. This occurred when K was set to one. Both forms of NN exhibited similar test accuracy of approximately 90%, but the feed forward NN did so with an extremely long run time. SVM was the middling algorithm, with a still very good 87.58% test accuracy, and less than 4 seconds run time. Decision trees were the fastest method by far; however, they were also the least accurate with just over 70% test accuracy. As such, all models were effectively validated in their suitability for analysis of the urban audio signals, but K neighbours is by far the preferred approach due to its unparalleled accuracy complemented with fantastic speed.

References

[1] N. Laskaris, *How to apply machine learning and deep learning methods to audio analysis*, Towards Data Science, Nov. 18, 2019. Accessed on April 21, 2021. [Online]. Available: https://towardsdatascience.com/how-to-apply-machine-learning-and-deep-learning-methods-to-audio-analysis-615e286fcbbc

[2] *librosa*, Librosa. Accessed on: April 22, 2021. [Online]. Available: https://librosa.org/

[3] *demo/audio-urbansound8k/Experiment (378a5a2cc2814b569de1f43c74e050fe*, Comet. Accessed on: April 20, 2021. [Online]. Available: https://www.comet.ml/demo/audio-urbansound8k/378a5a2cc2814b569de1f43c74e050fe?experiment-tab=code

[4] *The Sequential Class*, Keras. Accessed on: April 22, 2021. [Online]. Available: https://keras.io/api/models/sequential/

[5] *sklearn.neural_network/MLPClassifier*, Scikit-learn. Accessed on: April 22, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[6] *sklearn.svm.SVC*, Scikit-learn. Accessed on: April 22, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

[7] *sklearn.svm.NuSVC*, Scikit-learn. Accessed on: April 22, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html#sklearn.svm.NuSVC

[8] *sklearn.svm.LinearSVC*, Scikit-learn. Accessed on: April 22, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

[9] *sklearn.tree.DecisionTreeClassifier*, Scikit-learn. Accessed on: April 22, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[10] *sklearn.ensemble.ExtraTreesClassifier*, Scikit-learn. Accessed on: April 22, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

[11] *sklearn.neighbors.KNeighborsClassifier*, Scikit-learn. Accessed on: April 22, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html