Practical Machine Learning

Smartphone User Identification Competition

Tomeh Lara 507

This documentation describes a smartphone user identification project in which multiple classification models have been trained on a data set containing 3D accelerometer recordings. The aim is to distinguish between 20 different mobile device users. There are approximately 450 accelerometer signal recordings (examples) accessible for training for each user. While the user taps on the smartphone's screen, the signals are collected for 1.5 seconds. Because the accelerometer signal is collected at 100 Hz, it has around 150 values. The values are recorded on three axes: x, y, and z at a given moment. Initially, the libraries required for the completion of the task were imported, including certain libraries from Sklearn, Numpy, Pandas, and Matplotlib, as well as the libraries containing the models used in the implementation. In order to read the training and testing files, the Glob module was used to finds all paths names, which in our case is csv. As for reading the labels, it was done through the read_csv function provided by Pandas. Each csv file has around 150 rows of three columns. Some of the files are lower than others. They have been turned all into 150 rows of three columns, then flattened to 450 columns and one row. The data was partitioned using train_test_split utility, where the data was divided into training data and validation data, and the validation data is the last 2000 sample, as the shuffle parameter has a value of false, and therefore the data has not changed its default order.

As for the used models, most of them were machine learning models. The first model is SVC Support Vector Classifier. Creating the model was done by making a pipeline in addition to using standard scaler for data preprocessing and PCA which stands for Principal Component Analysis strategy for reducing dimensionality. It enables to compress a data set into a smaller data set with fewer features while retaining as much of the original data as feasible, it enhances the performance of the classification and regression methods by preventing overfitting and decreasing noise. It contributes to the model's training time being reduced, the number of PCA components was given the following values (30, and 50) and to make matters clearer, these values have been added to all models. Additionally, the parameters of the model were obtained through the get params function. Some parameters were selected and each one was given different values such as the PCA number of components and the value of gamma 0.01, 0.02, 0.03 and scale, and degree value as 2, 3, 4, and 5, in addition to C value as 1.0 and 2.0. After that, using grid search, which performs a comprehensive search for the previously defined parameter values and chooses the best ones, this is called hyperparameters tuning. It is worth mentioning that grid search has a parameter which is cv and this parameter takes values of type integer, and its default value is None, which is the default 5-fold cross-validation, which was used in the models. Following that, the best score 0.9059 of the model was printed and the best parameters also using best_params function and the output was PCA number of components is 30, C value is 2, degree is 2, gamma is 0.01, then the model was fitted and the confusion matrix was shown, which indicates the difference between the actual labels and the ones predicted by the model. Finally, the model score on validation data was then computed, it reached 0.9135 which is the best result of all models.

The second model is K-Neighbors Classifier that selects the k closest neighbors of a sample to be categorized, this model was implemented in a very similar way to the previous one, and a number of different values were passed to the parameters to be passed to Grid Search, such as PCA number of components whose values have been previously indicated, number of neighbors which its default value is five, and two other values, six and seven have been added, also P value which its default value is two and another value of one has been added, and the type of algorithm, whether it is ball tree, kd tree, brute-force search, or auto which selects the best algorithm based on the variables supplied to the fit method, and also the weights parameter which is the weight function used in prediction either uniform or distance. The grid search algorithm was applied for hyper parameter tuning, then it gives the best parameters as following auto algorithm, five number of neighbors, P value is 2, weights type is distance, and PCA number of components is 30, then fitting for the model, and plotting also for confusion matrix, and the score model calculation on validation data, which is 0.76.

The third model is Random Forest model, in the same way as the previous two models, parameters such as maximum depth, if its value is none, which is the default value, then the nodes will expand until all the leaves of the trees are burned, or it can be given an integer value, as in the current example is 10, 20, the second parameter number of estimators as 50, 100, and 200, and the last parameter is criterion the model got gini from Gini impurity, and entropy. At the end, hyperparameter tuning, which output was 50 number of PCA component, gini criterion, maximum depth of 2, and number of estimators is 200. model fitting, confusion matrix plotting, and model score calculation on validation data was made, which scored a value of 0.878.

The fourth model is XGBoost which stands for Extreme Gradient Boosting, it is very accurate model. It employs learning methods, which mix multiple machine learning algorithms to produce a more accurate model. After some parameters were added, like learning rate which is by default 0.3, and 0.1, 0.2, 0.4 were added, and the number of estimators like 50, 70, 100 and 200, in addition to the max depth 5, 6, 7 and 9. The randomized search CV algorithm from SKlearn library on hyper parameters was used. This approach does not test all parameter values, but rather a set number of parameters randomly defined by the number of iterations, in quite the opposite of the grid search CV algorithm. Finally, the model score achieved 0.8715.

Last but not least, Bagging Classifier model provided by SKlearn library was implemented, like the first three models, hyperparameters tuning used by grid search algorithm. The hyperparameters given were the number of estimators which is 10 by default and 20, 30, 40, and 50 was added. The grid search gave 50 number of estimator and 30 number of PCA component as the best parameters. Finally, the model scored 0.812 for the validation dataset.

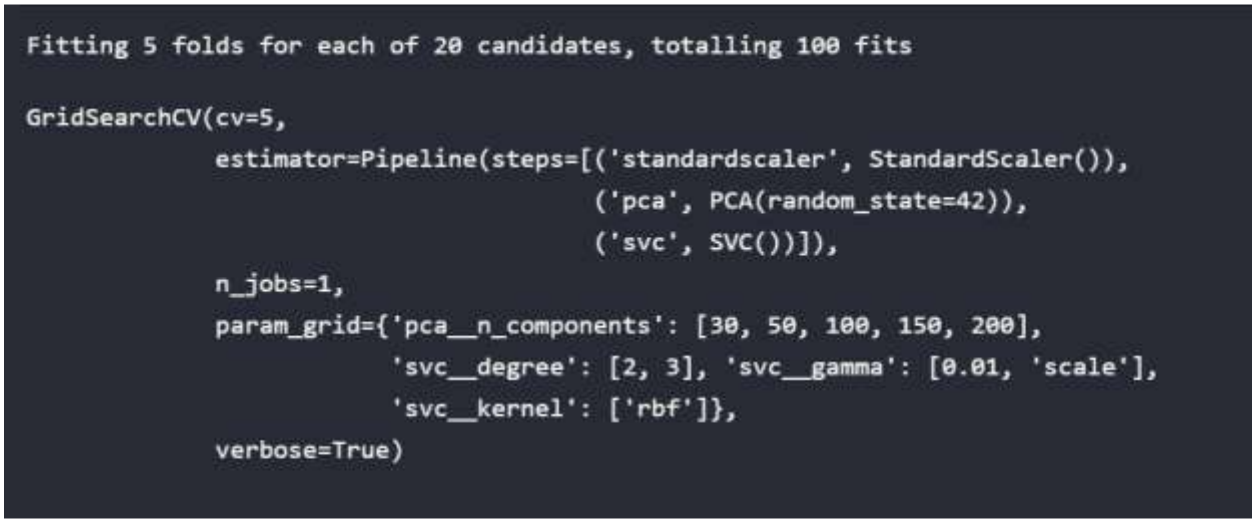
After completing the training of all the previously mentioned models, the models were tested on the testing data. The testing data was read as previously the training data was read, and in the same way of preprocessing. After reading each file, the files were converted into an array. Then the names of the files are read and stored in a data frame until each one is given a value that is stored next to the file name, and then the labels are expected for each file and from each model separately and these expectations (predictions) are added to the previously mentioned data frame under the column named class, and then stored the data frame is in the form of a csv file on the computer.

In conclusion, the best model in performance is support vector classification, in the second level the random forest and XGBoost were, in contrary, K-Neighbors is the is the lowest in terms of performance and in general, machine learning models take a long time to implement and need high-performance computers and rather large memories.

Accuracy score on training set and validation set

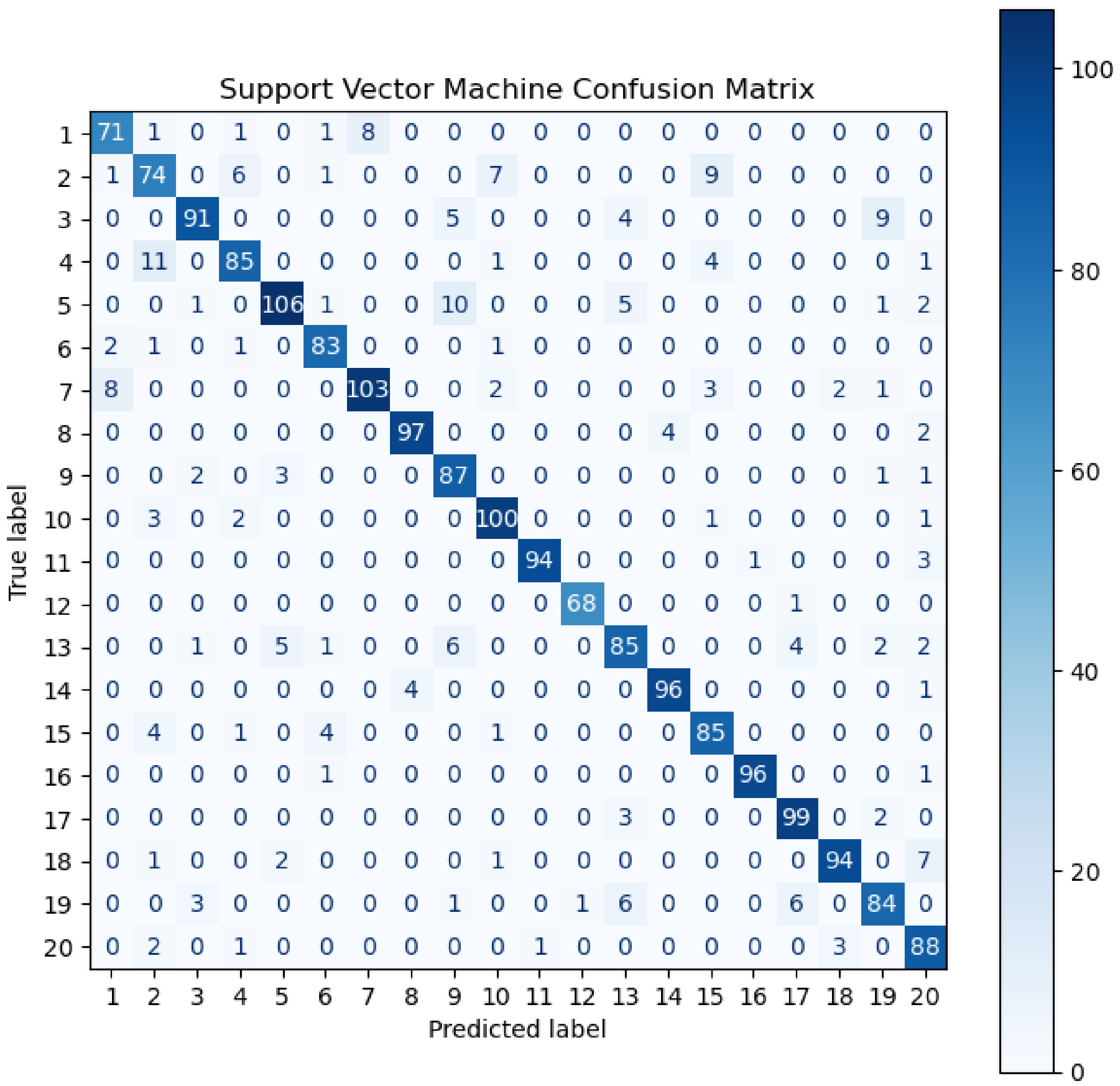
Classifier	CV accuracy score on Training Set	Accuracy on Validation Set
SVC	0.9059	0.9135
K-Neighbors	0.7399	0.76
Random Forest	0.8747	0.878
XGBoost	0.8627	0.8715
Bagging Classifier	0.813	0.812

5-fold cross-validation on the training set

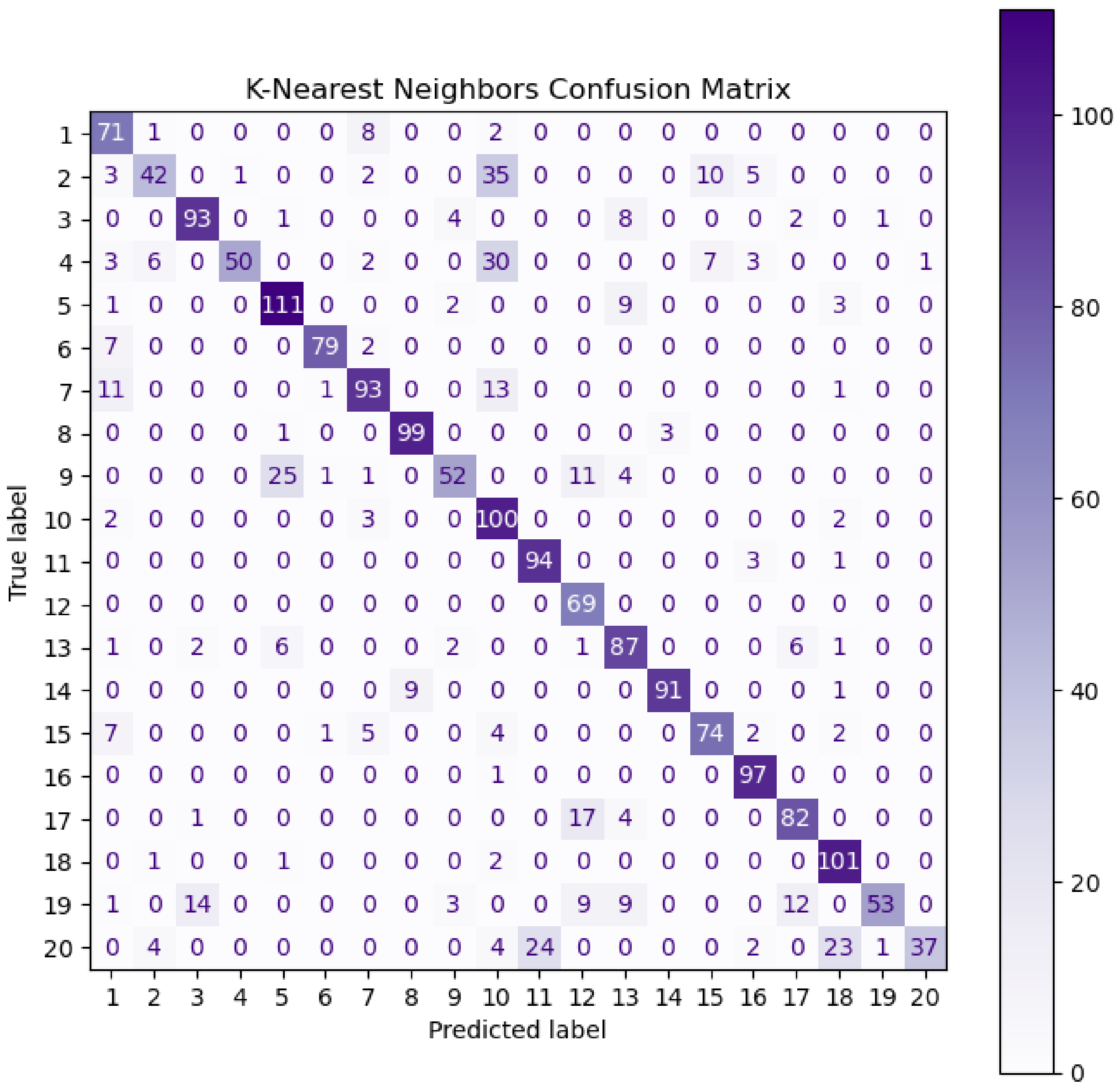


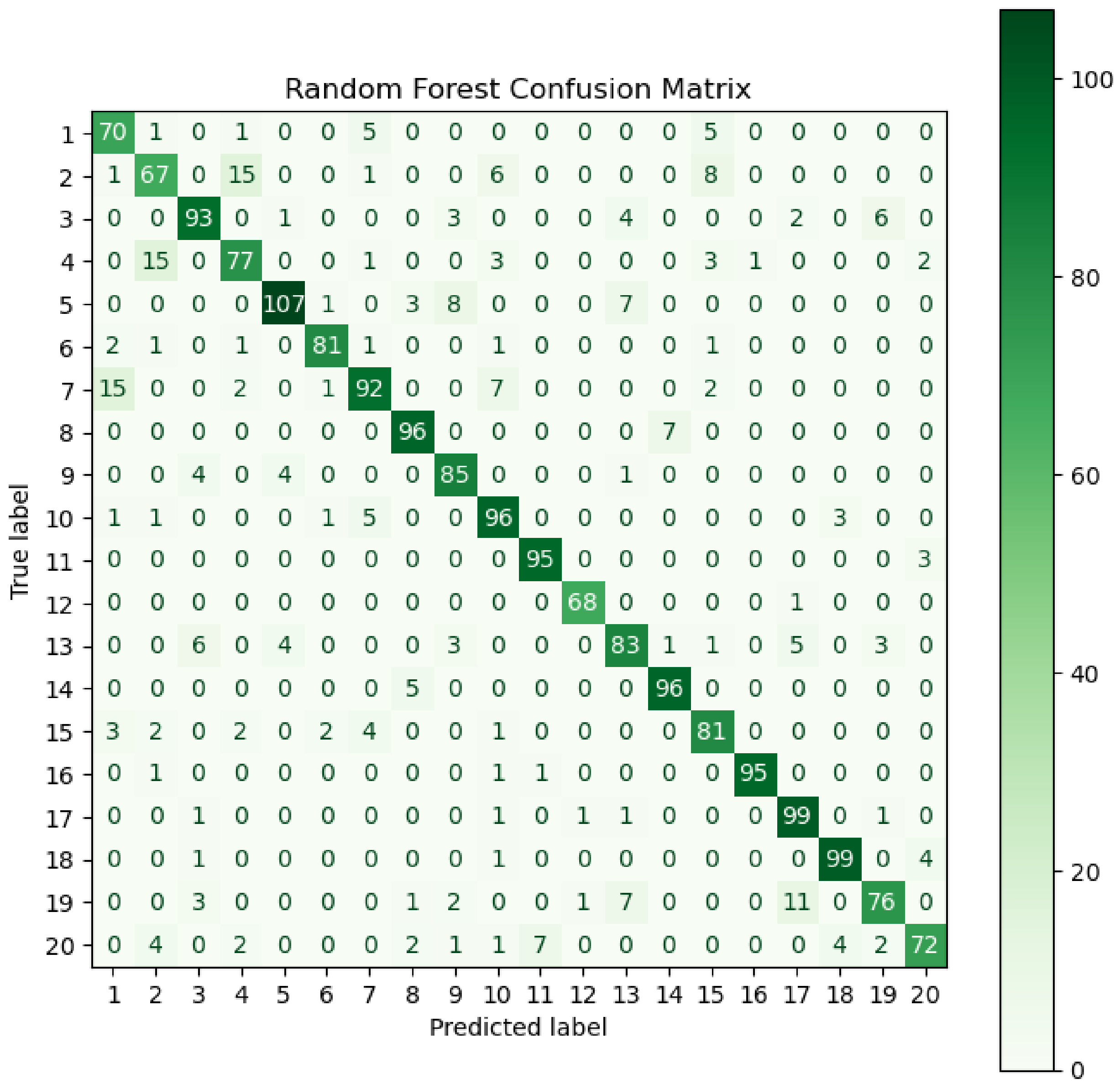
```
Fitting 5 folds for each of 20 candidates, totalling 100 fits

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('standardscaler', StandardScaler()),
                                       ('pca', PCA(random_state=42)),
                                       ('svc', SVC())]),
             n_jobs=1,
             param_grid={'pca__n_components': [30, 50, 100, 150, 200],
                         'svc__degree': [2, 3], 'svc__gamma': [0.01, 'scale'],
                         'svc__kernel': ['rbf']},
             verbose=True)
```

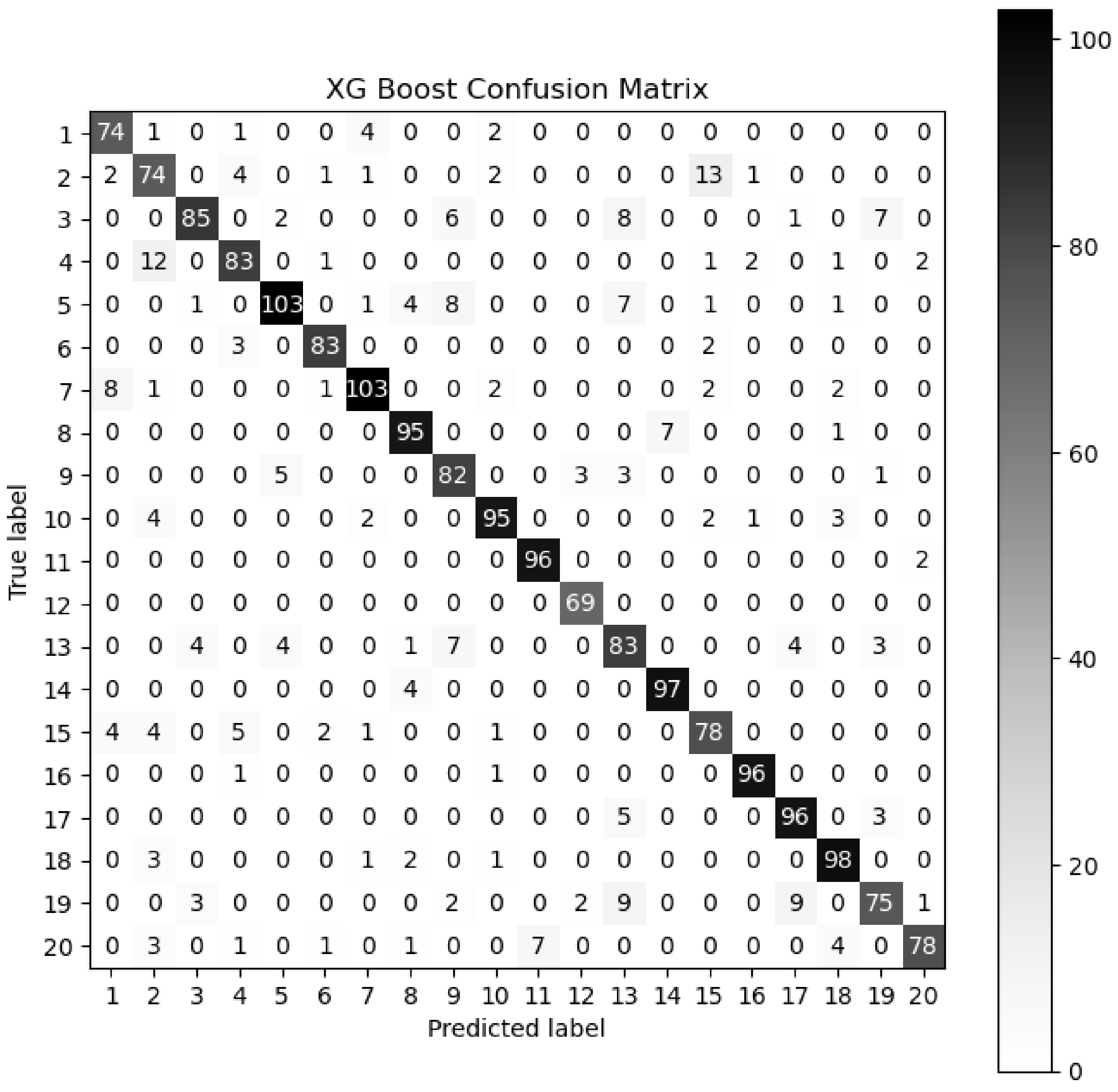


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	71	1	0	1	0	1	8	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	74	0	6	0	1	0	0	0	7	0	0	0	0	9	0	0	0	0	0
3	0	0	91	0	0	0	0	0	5	0	0	0	4	0	0	0	0	0	9	0
4	0	11	0	85	0	0	0	0	0	1	0	0	0	0	4	0	0	0	0	1
5	0	0	1	0	106	1	0	0	10	0	0	0	5	0	0	0	0	0	1	2
6	2	1	0	1	0	83	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	8	0	0	0	0	0	103	0	0	2	0	0	0	0	3	0	0	2	1	0
8	0	0	0	0	0	0	0	97	0	0	0	0	0	4	0	0	0	0	0	2
9	0	0	2	0	3	0	0	0	87	0	0	0	0	0	0	0	0	0	1	1
10	0	3	0	2	0	0	0	0	0	100	0	0	0	0	1	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	94	0	0	0	0	1	0	0	0	3
12	0	0	0	0	0	0	0	0	0	0	0	68	0	0	0	0	1	0	0	0
13	0	0	1	0	5	1	0	0	6	0	0	0	85	0	0	0	4	0	2	2
14	0	0	0	0	0	0	0	4	0	0	0	0	0	96	0	0	0	0	0	1
15	0	4	0	1	0	4	0	0	0	1	0	0	0	0	85	0	0	0	0	0
16	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	96	0	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	99	0	2	0
18	0	1	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	94	0	7
19	0	0	3	0	0	0	0	0	1	0	0	1	6	0	0	0	6	0	84	0
20	0	2	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	3	0	88

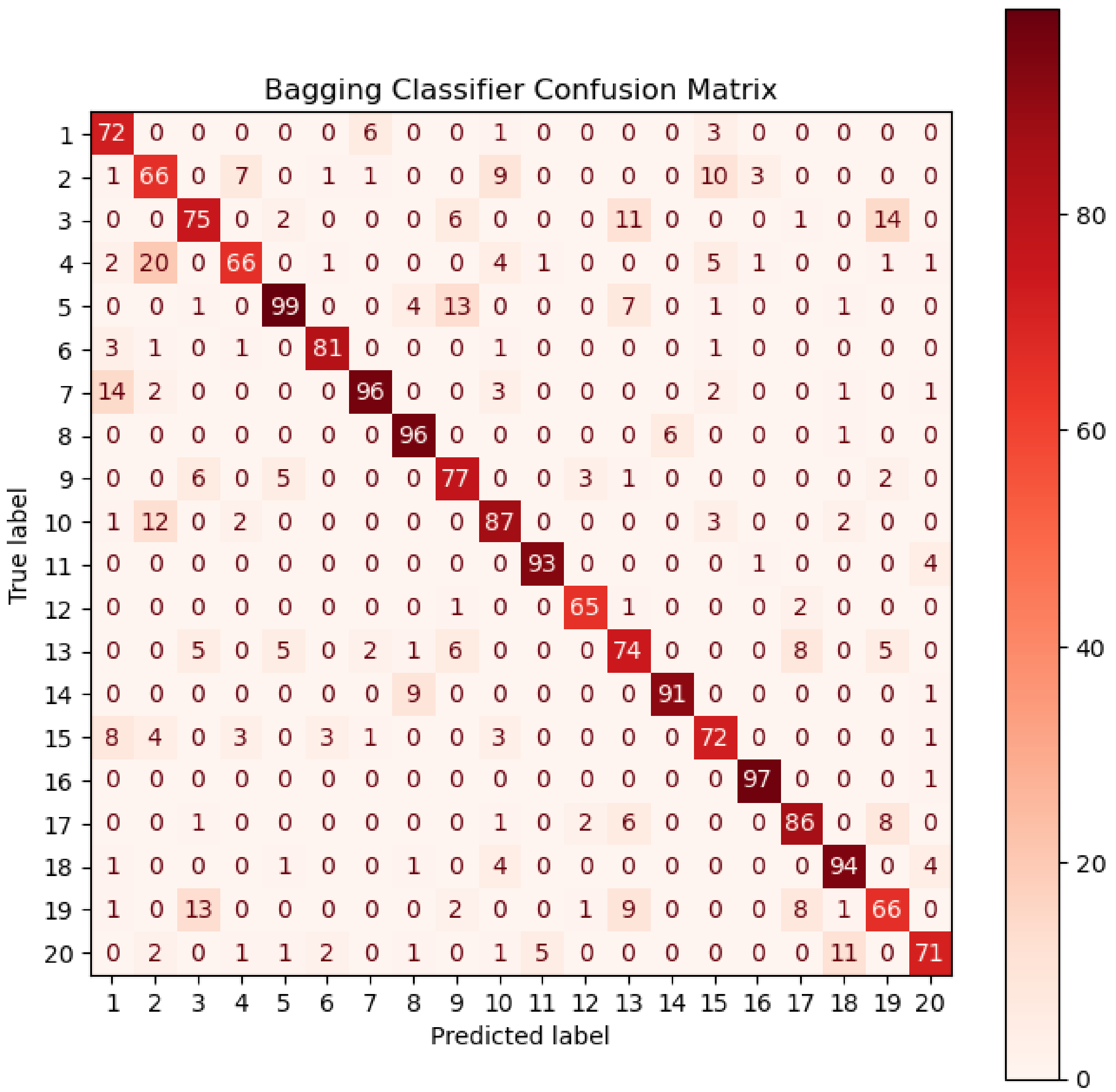

K-Nearest Neighbors Confusion Matrix
True label
Predicted label
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1: 71 1 0 0 0 0 8 0 0 2 0 0 0 0 0 0 0 0 0 0
2: 3 42 0 1 0 0 2 0 0 35 0 0 0 0 10 5 0 0 0 0
3: 0 0 93 0 1 0 0 0 4 0 0 0 8 0 0 0 2 0 1 0
4: 3 6 0 50 0 0 2 0 0 30 0 0 0 0 7 3 0 0 0 1
5: 1 0 0 0 111 0 0 0 2 0 0 0 9 0 0 0 0 3 0 0
6: 7 0 0 0 0 79 2 0 0 0 0 0 0 0 0 0 0 0 0 0
7: 11 0 0 0 0 1 93 0 0 13 0 0 0 0 0 0 0 1 0 0
8: 0 0 0 0 1 0 0 99 0 0 0 0 0 3 0 0 0 0 0 0
9: 0 0 0 0 25 1 1 0 52 0 0 11 4 0 0 0 0 0 0 0
10: 2 0 0 0 0 0 3 0 0 100 0 0 0 0 0 0 0 2 0 0
11: 0 0 0 0 0 0 0 0 0 0 94 0 0 0 0 3 0 1 0 0
12: 0 0 0 0 0 0 0 0 0 0 0 69 0 0 0 0 0 0 0 0
13: 1 0 2 0 6 0 0 0 2 0 0 1 87 0 0 0 6 1 0 0
14: 0 0 0 0 0 0 0 9 0 0 0 0 0 91 0 0 0 1 0 0
15: 7 0 0 0 0 1 5 0 0 4 0 0 0 0 74 2 0 2 0 0
16: 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 97 0 0 0 0
17: 0 0 1 0 0 0 0 0 0 0 0 17 4 0 0 0 82 0 0 0
18: 0 1 0 0 1 0 0 0 0 2 0 0 0 0 0 0 0 101 0 0
19: 1 0 14 0 0 0 0 0 3 0 0 9 9 0 0 0 12 0 53 0
20: 0 4 0 0 0 0 0 0 0 4 24 0 0 0 0 2 0 23 1 37
0
20
40
60
80
100



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	70	1	0	1	0	0	5	0	0	0	0	0	0	0	5	0	0	0	0	0
2	1	67	0	15	0	0	1	0	0	6	0	0	0	0	8	0	0	0	0	0
3	0	0	93	0	1	0	0	0	3	0	0	0	4	0	0	0	2	0	6	0
4	0	15	0	77	0	0	1	0	0	3	0	0	0	0	3	1	0	0	0	2
5	0	0	0	0	107	1	0	3	8	0	0	0	7	0	0	0	0	0	0	0
6	2	1	0	1	0	81	1	0	0	1	0	0	0	0	1	0	0	0	0	0
7	15	0	0	2	0	1	92	0	0	7	0	0	0	0	2	0	0	0	0	0
8	0	0	0	0	0	0	0	96	0	0	0	0	0	7	0	0	0	0	0	0
9	0	0	4	0	4	0	0	0	85	0	0	0	1	0	0	0	0	0	0	0
10	1	1	0	0	0	1	5	0	0	96	0	0	0	0	0	0	0	3	0	0
11	0	0	0	0	0	0	0	0	0	0	95	0	0	0	0	0	0	0	0	3
12	0	0	0	0	0	0	0	0	0	0	0	68	0	0	0	0	1	0	0	0
13	0	0	6	0	4	0	0	0	3	0	0	0	83	1	1	0	5	0	3	0
14	0	0	0	0	0	0	0	5	0	0	0	0	0	96	0	0	0	0	0	0
15	3	2	0	2	0	2	4	0	0	1	0	0	0	0	81	0	0	0	0	0
16	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	95	0	0	0	0
17	0	0	1	0	0	0	0	0	0	1	0	1	1	0	0	0	99	0	1	0
18	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	99	0	4
19	0	0	3	0	0	0	0	1	2	0	0	1	7	0	0	0	11	0	76	0
20	0	4	0	2	0	0	0	2	1	1	7	0	0	0	0	0	0	4	2	72



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	74	1	0	1	0	0	4	0	0	2	0	0	0	0	0	0	0	0	0	0
2	2	74	0	4	0	1	1	0	0	2	0	0	0	0	13	1	0	0	0	0
3	0	0	85	0	2	0	0	0	6	0	0	0	8	0	0	0	1	0	7	0
4	0	12	0	83	0	1	0	0	0	0	0	0	0	0	1	2	0	1	0	2
5	0	0	1	0	103	0	1	4	8	0	0	0	7	0	1	0	0	1	0	0
6	0	0	0	3	0	83	0	0	0	0	0	0	0	0	2	0	0	0	0	0
7	8	1	0	0	0	1	103	0	0	2	0	0	0	0	2	0	0	2	0	0
8	0	0	0	0	0	0	0	95	0	0	0	0	0	7	0	0	0	1	0	0
9	0	0	0	0	5	0	0	0	82	0	0	3	3	0	0	0	0	0	1	0
10	0	4	0	0	0	0	2	0	0	95	0	0	0	0	2	1	0	3	0	0
11	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	2
12	0	0	0	0	0	0	0	0	0	0	0	69	0	0	0	0	0	0	0	0
13	0	0	4	0	4	0	0	1	7	0	0	0	83	0	0	0	4	0	3	0
14	0	0	0	0	0	0	0	4	0	0	0	0	0	97	0	0	0	0	0	0
15	4	4	0	5	0	2	1	0	0	1	0	0	0	0	78	0	0	0	0	0
16	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	96	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	96	0	3	0
18	0	3	0	0	0	0	1	2	0	1	0	0	0	0	0	0	0	98	0	0
19	0	0	3	0	0	0	0	0	2	0	0	2	9	0	0	0	9	0	75	1
20	0	3	0	1	0	1	0	1	0	0	7	0	0	0	0	0	0	4	0	78



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	72	0	0	0	0	0	6	0	0	1	0	0	0	0	3	0	0	0	0	0
2	1	66	0	7	0	1	1	0	0	9	0	0	0	0	10	3	0	0	0	0
3	0	0	75	0	2	0	0	0	6	0	0	0	11	0	0	0	1	0	14	0
4	2	20	0	66	0	1	0	0	0	4	1	0	0	0	5	1	0	0	1	1
5	0	0	1	0	99	0	0	4	13	0	0	0	7	0	1	0	0	1	0	0
6	3	1	0	1	0	81	0	0	0	1	0	0	0	0	1	0	0	0	0	0
7	14	2	0	0	0	0	96	0	0	3	0	0	0	0	2	0	0	1	0	1
8	0	0	0	0	0	0	0	96	0	0	0	0	0	6	0	0	0	1	0	0
9	0	0	6	0	5	0	0	0	77	0	0	3	1	0	0	0	0	0	2	0
10	1	12	0	2	0	0	0	0	0	87	0	0	0	0	3	0	0	2	0	0
11	0	0	0	0	0	0	0	0	0	0	93	0	0	0	0	1	0	0	0	4
12	0	0	0	0	0	0	0	0	1	0	0	65	1	0	0	0	2	0	0	0
13	0	0	5	0	5	0	2	1	6	0	0	0	74	0	0	0	8	0	5	0
14	0	0	0	0	0	0	0	9	0	0	0	0	0	91	0	0	0	0	0	1
15	8	4	0	3	0	3	1	0	0	3	0	0	0	0	72	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	1
17	0	0	1	0	0	0	0	0	0	1	0	2	6	0	0	0	86	0	8	0
18	1	0	0	0	1	0	0	1	0	4	0	0	0	0	0	0	0	94	0	4
19	1	0	13	0	0	0	0	0	2	0	0	1	9	0	0	0	8	1	66	0
20	0	2	0	1	1	2	0	1	0	1	5	0	0	0	0	0	0	11	0	71