



KU Leuven
Departement of Computer Science

MACHINE LEARNING

Who has my phone?

Group:

THIJS DIELTJENS
BRAM GELEN

Academiejaar 2014 – 2015

1 Abstract

In this assignment it is necessary to be able to detect which person is walking around with a phone. This detection will be based on previously labeled data, generated by people walking with their phone. The detection is done by extracting useful features for each of the given walks and then applying Machine Learning techniques on this data. Our solution provides a command line based user interface which offers a lot of options and flexibility. The results generated by this solution however do not offer a high accuracy.

2 Tackled Questions

The overall goal of this paper is to see if it is possible to determine who is walking with their phone machine by using learning techniques. To do so we use data generated by sensors of their phone and based on a set of similar data that has been gathered before. To achieve this goal answers need to be found to a number of questions. First it is necessary to understand which features of the acquired data are useful for the detection. Secondly based on previous studies it is shown that the data needs to be split up in windows to achieve better results [1]. Therefore a good window size and partitioning strategy is needed. Another question that arises is whether and how we need to preprocess the data and filter out the corrupt parts of it. Then a good classifier needs to be chosen to classify the data. Additionally we also need a correct way to assess the accuracy of the chosen classifier. Finally the implementation for the solution should be easy to use and understand.

3 Results

In this section we first briefly discuss the general flow of our implementation. Then we will explain the used methods and results for each question from before. We based our solutions on the findings in the previous literature study [2]. Our solution is implemented in Java using the Weka toolkit [3].

3.1 General Flow

The implementation that tries to achieve our goal follows the sequence showed in Figure 1. First both the training data and testing data are parsed from their corresponding CSV-files. Secondly these datasets are split up into different windows. Then a Feature Extractor will extract all useful features from these windows. Next a filter is used to filter out corrupt windows based on these features. Then the training data will train a classifier. This classifier will on its turn classify all the filtered testwindows. Finally all these windows then need to be recombined to one walk by voting for their classifications. This will in the end generate the result.

3.2 Windows

We used sliding windows with 50% overlap for feature extraction since this has been demonstrated succesfull in past works [1]. Our solution uses a sampling frequency of 50 Hz and each window represents 5.1 seconds. This way every window has at least 256 samples. A window of several seconds was used to sufficiently capture cycles in the walking [1]. We also tried to use windows of 2.55 seconds but this proved to give worse results.

3.3 Features

In total we extract 42 features for each window. We use a combination of time and frequency domain features that have given good results in previous papers [1][4]. All the features are computed for each axis except for the correlation which is determined by using every combination of two axes. The number in brackets indicates the number of features that are extracted.

- Mean (3), standard deviaton (3), median (3), first and third percentile (6)

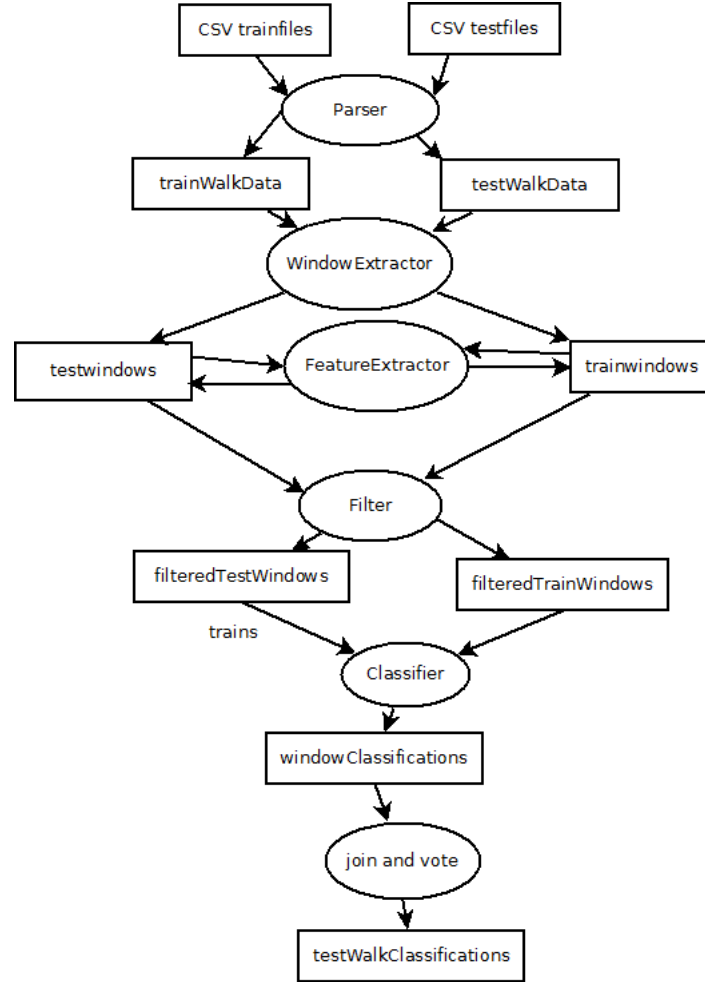


Figure 1: General work flow of solution

- minimum (3), maximum (3)
- Correlation between the axes (3)
- Magnitude of first 5 components of FFT analysis (15)
- Energy (3)

3.4 Filter

As mentioned in the assignment it is possible that certain windows will contain corrupt data. (e.g. generated when starting the sensors) Therefore we provide the possibility to create a filter to remove unwanted windows from both test and training set. This filter is based on a second classifier which aims to classify corrupt windows as unwanted. When the *manualfilter* command is given to the terminal, a screen will pop up showing a plot for the values of a window. The user can then either accept or reject this window. Then a next window will appear that the user has to manually classify. This process continues as long as the user wants. Based on the decisions the user made, a Decision Tree classifier will be trained. This classifier will then behave as a filter for all the remaining windows.

The created filter however did not seem to provide noticeable added accuracy for all classifiers. For the nearest neighbor classifier the accuracy did go up. The fact that it did not add accuracy to the other classifiers might however be caused because of the poor quality of the results before adding the filter (See 3.7).

3.5 Assessing Accuracy

To assess the accuracy of our solutions, we use leave-one-out cross-validation. (Results can be shown with the *crossvalidation* command). Cross-validation however requires independence between training and test set. This poses a problem if we were to use it directly on the windows because of the overlap. Therefore cross-validation is performed on entire walks.

An entire walk is classified by letting each of the classified windows vote for the classification they think is the best one. Each vote is weighted by the corresponding confidence.

If the *crossvalidation* command is given the system will print out for each known person as which person he was qualified and how many times.

3.6 Classification

The classifier mentioned in this section is the one used to classify windows as a person. Our solution has the ability to use every Weka classifier with every allowed option. We did however provide additional functionality for the following classifiers since these were used extensively in related work. In italics are the corresponding Weka classifiers if their name does not correspond

- Decision Trees (*J48*)
- Support Vector Machines (*SMO*)
- k Nearest Neighbors (*Ibk*)
- Naive Bayes
- Random Forest

3.7 Overall Result

All the used classifiers classify most, if not all test data as “Other”. This is not in se a bad thing, but we do not know why this happens. The cross-validation of the set of training walks gives the same result (it predicts a lot of “Other” classes). What follows is an overview of the different classifiers that were used in previous works. For these we provided a shortcut in the commandline. The arguments that are displayed in each subsection are the arguments that were given to our program. According to the accuracy the nearest neighbor classifier performs best with an accuracy of 73.6%. Therefore it is the default classifier.

3.7.1 Trees

Arguments : -d -cm -cv -c tree

Cross validation General accuracy of : 62.2%

other classified as : other(29) leander(3) wannes(2)
leander classified as : other(6) leander(1)
wannes classified as : other(8) leander(1) wannes(3)

Confusion matrix for windows	a	b	c	← classified as
	791	5	6	a = other
	3	120	0	b = leander
	6	2	64	c = wannes

	file	result	confidence
Classification	walk_59_unknown.csv	Wannes	0.45098039215686275
	walk_60_unknown.csv	Other	0.9905660377358491
	walk_61_unknown.csv	Other	0.620507925999485
	walk_62_unknown.csv	Other	0.70947389437955476
	walk_63_unknown.csv	Other	0.7036363636363636
	walk_64_unknown.csv	Wannes	0.8571428571428571
	walk_65_unknown.csv	Wannes	0.5257142857142857
	walk_66_unknown.csv	Other	0.8731768417931942
	walk_67_unknown.csv	Other	0.6641975308641975
	walk_68_unknown.csv	Other	0.9224150943396229

3.7.2 Naive bayes

Arguments : -d -cm -cv -c nbayes

Cross validation General accuracy of : 28.3%
other classified as : other(1) leander(29) wannes(4)
leander classified as : leander(7)
wannes classified as : other(1) leander(4) wannes(7)

	a	b	c	← classified as
Confusion matrix for windows	20	649	133	a = other
	0	118	5	b = leander
	1	10	61	c = wannes

	file	result	confidence
Classification	walk_59_unknown.csv	Leander	0.8361624408067347
	walk_60_unknown.csv	Leander	0.9998598348072463
	walk_61_unknown.csv	Leander	0.8903904326672154
	walk_62_unknown.csv	Wannes	0.690597901699413
	walk_63_unknown.csv	Leander	0.9643679601299486
	walk_64_unknown.csv	Leander	0.49048733077690815
	walk_65_unknown.csv	Leander	0.8266935179196736
	walk_66_unknown.csv	Leander	0.5389096239643194
	walk_67_unknown.csv	Wannes	0.999987704199726
	walk_68_unknown.csv	Leander	0.6953749184479269

3.7.3 Nearest neighbour

Arguments : -d -cm -cv -c knn

Cross validation General accuracy of : 73.6%
other classified as : other(34)
leander classified as : other(5) leander(2)
wannes classified as : other(9) wannes(3)

	a	b	c	← classified as
Confusion matrix for windows	802	0	0	a = other
	0	123	0	b = leander
	0	0	72	c = wannes

	file	result	confidence
Classification	walk_59_unknown.csv	Other	0.6653333333333336
	walk_60_unknown.csv	Other	0.9979999999999999
	walk_61_unknown.csv	Other	0.630315789473684
	walk_62_unknown.csv	Other	0.8554285714285713
	walk_63_unknown.csv	Leander	0.5443636363636363
	walk_64_unknown.csv	Other	0.6653333333333332
	walk_65_unknown.csv	Wannes	0.5702857142857142
	walk_66_unknown.csv	Other	0.7560606060606061
	walk_67_unknown.csv	Other	0.6653333333333332
	walk_68_unknown.csv	Other	0.9314666666666664

3.7.4 3-nearest neighbours

Arguments : -d -cm -cv -c knn -K 3

Cross validation General accuracy of : 66.0%
other classified as : other(34)
leander classified as : other(6) leander(1)
wannes classified as : other(12)

	a	b	c	← classified as
Confusion matrix for windows	795	6	1	a = other
	29	93	1	b = leander
	31	0	41	c = wannes

	file	result	confidence
Classification	walk_59_unknown.csv	Other	0.6662671748726209
	walk_60_unknown.csv	Other	0.9993319973279894
	walk_61_unknown.csv	Other	0.6487536476461695
	walk_62_unknown.csv	Other	0.8565702834239909
	walk_63_unknown.csv	Other	0.5451205441185402
	walk_64_unknown.csv	Other	0.6218659541304832
	walk_65_unknown.csv	Other	0.47590418933104306
	walk_66_unknown.csv	Other	0.7672010688042749
	walk_67_unknown.csv	Other	0.8883322199955467
	walk_68_unknown.csv	Other	0.8883322199955466

3.7.5 Support Vector Machines

Arguments : -d -cm -cv -c svm

Cross validation General accuracy of : 64.2%
other classified as : other(34)
leander classified as : other(7)
wannes classified as : other(12)

	a	b	c	← classified as
Confusion matrix for windows	802	0	0	a = other
	123	0	0	b = leander
	72	0	0	c = wannes

	file	result	confidence
Classification	walk_59_unknown.csv	Other	0.666666666666667
	walk_60_unknown.csv	Other	0.666666666666666
	walk_61_unknown.csv	Other	0.666666666666664
	walk_62_unknown.csv	Other	0.666666666666666
	walk_63_unknown.csv	Other	0.666666666666667
	walk_64_unknown.csv	Other	0.666666666666665
	walk_65_unknown.csv	Other	0.666666666666666
	walk_66_unknown.csv	Other	0.666666666666667
	walk_67_unknown.csv	other	0.666666666666666
	walk_68_unknown.csv	Other	0.666666666666665

3.7.6 Random Forest

Arguments : -d -cm -cv -c rforest

Cross validation General accuracy of : 64.2%

other classified as : other(34)

leander classified as : other(7)

wannes classified as : other(12)

	a	b	c	← classified as
Confusion matrix for windows	802	0	0	a = other
	7	116	0	b = leander
	5	0	67	c = wannes

	file	result	confidence
Classification	walk_59_unknown.csv	Other	0.7411764705882353
	walk_60_unknown.csv	Other	0.8333333333333334
	walk_61_unknown.csv	Other	0.6210526315789473
	walk_62_unknown.csv	Other	0.8
	walk_63_unknown.csv	Other	0.7
	walk_64_unknown.csv	Other	0.5533333333333332
	walk_65_unknown.csv	Other	0.35714285714285715
	walk_66_unknown.csv	Other	0.7424242424242424
	walk_67_unknown.csv	Other	0.7999999999999999
	walk_68_unknown.csv	Other	0.8933333333333334

3.8 Final program

To use the program a user has to run the executable jar-file *MLCode.jar*. To use this jar-file correctly we provide a read-me file (Appendix A). The program is very flexible because it allows a lot of options. It can use every Weka-classifier and on top of that is able to use all of its options. It is easy to change the training and testing folder. Details and crossvalidation for the classification can be printed. The confusionmatrix for the classification of the windows can be printed. The user also has the option to create a new filter or use a previous one.

4 Conclusion

By using the information we found in the papers of our previous lecture study, we were able to tackle most of the questions we posed. We found solutions for creating windows, extracting 42 features and filtering corrupt windows. We were able to make classifications for the windows and join them back together for each walk. A cross-validation method was implement to assess the accuracy. However it showed that the results of these classifications were not as good as expected. Most classifiers tend to classify all walks as *Other* so for those classifiers we expect the predictions for the unseen walks to be erroneous.

5 Time Spent

Table 1 gives an overview of the time spent on each subsection of the project for each student. Time was equally divided among us.

Section	Time spent	Summary
Literature Study and report	10h	Looking up papers and relevant information about the subject. Writing report
Further research	2h	Planning implementation. Learning Weka. Further reading.
Parsing CSV	1h20	Parsing CSV and transforming data into our own datatypes.
Dividing into windows	1h30	Creating 50% overlapping windows out of data.
Extracting Features	4h30	Extracting all the features we used.
Using Weka to classify	8h	Training classifier. Classify testdata.
Transforming results back to walks	2h	From weka data for a window to classification for a walk.
Cross-Validation	2h	Creating methods to use leave-one-out validation.
Creating working program	2h	Creating command line user interface
Analyzing results	4h	Analyzing results of final application
Final Report	6h30	Writing final report
Total	45h50	

Table 1: Time division

References

- [1] L. Bao and S. S. Intille, “Activity recognition from user-annotated acceleration data,” in *Pervasive computing*, pp. 1–17, Springer, 2004.
- [2] Dieltjens and Geelen, “Who has my phone?,” November 2014.
- [3] M. L. G. at the University of Waikato, “Weka 3: Data Mining Software in Java,” November 2014. <http://www.cs.waikato.ac.nz/ml/weka/>.
- [4] S. J. Preece, J. Y. Goulermas, L. P. Kenney, and D. Howard, “A comparison of feature extraction methods for the classification of dynamic activities from accelerometer data,” *Biomedical Engineering, IEEE Transactions on*, vol. 56, no. 3, pp. 871–879, 2009.

A Readme

This java program allows a user to classify unseen walking data using training data. Multiple classifiers can be used with their available options in Weka.

usage: java -jar MLCode.jar [options] [classifier] [classifier options]

Options:

-h --help	Prints this message.
-test	Path to the folder containing the test csv-files.
-train	Path to the folder containing the test csv-files.
-lo --listoptions	Lists the available options given the classifier.
-d --details	Prints the details of the classification.
-cm --confusionmatrix	Prints the confusion matrix of the windows of the training set.
-cv --crossvalidation	Performs cross validation on the dataset.
-mf --manualfilter	Presents a gui to extract valid windows manually.
-lf --lastfilter	Use last created manual filter to get valid windows.
-v --version	Prints the version of this build.

Classifier:

-c --classifier	The classifier to be used for the classification. Specify a classifier as one of the following: nbayes tree svm knn rforest or use the class name of a WEKA classifier.
-----------------	--

Classifier Options:

Additional options to be passed to the classifier can be specified here.

e.g. java -jar MLCode.jar -c knn -k 15

For a list of available options use -lo -c <classifier>.

Example Usage:

java -jar MLCode.jar -test testFolder -train trainFolder -d -cm -c tree