A4 REFLECTION + MODEL EVALUATION

CSE 599 Prototyping Interactive Systems | Lecture 17 | May 28

Jon Froehlich • Jasper O'Leary (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON

REMAINING TWO WEEKS

YOUR VOTES

In the next two weeks, we should: *

50%	☐	Have at least one overview lecture on applying some of what we've been learning to images (i.e., basic computer vision)
60%	☐	Have at least one lecture going over event segmentation and real-time ML (as we were originally going to do for A4)
50%	☐	Have at least one lecture going over feature selection and parameter tuning
10%	☐	Return to Physical Computing and learn about more of the electronics in your kit (e.g., the piezo buzzer) plus possibly concepts like using transistors to power motors, etc.
70%	☐	Have a lecture on laser cutting (Jasper needs to setup the Glowforge for this but they also have laser cutters on the on-campus makerspaces)
50%	☐	I would prefer that we spend most of the classtime working on our projects and getting feedback
	☐	Other:

A4: Signal Processing + Machine Learning 2: Model-Based Gesture Recognizer

Published | Edit

Overview

This assignment directly builds on A3: you will build a *feature-based* (or *model-based*) recognizer using a support-vector machine (SVM) (recommended) or an alternative supervised learning approach of your choosing (*e.g.*, an HMM). You will also compare your results to the shape-matching algorithm from A3. At a minimum, you must test on my gesture set (11 gestures) and your gesture set (11 gestures). You need to upload your gesture set to this Google Drive folder as part of the assignment.

To get started, download and open this Jupyter notebook: SVMGestureRecognizer.ipynb. Like with A3, I've provided you with some parsing code, data structures, and some initial visualizations. In addition, I've also built the full training+test pipeline for training and testing the SVM using k-fold cross validation. Note: for the file handling code to work, you must have a folder called 'GestureLogs' in the same dir as this Notebook. Within 'GestureLogs', place folders of your gesture sets (at the very least, mine and yours but you could also download others from the Google Drive)

You need to decide how to incorporate your A3 shape-matching approach into this notebook (*e.g.*, you could make a copy of your A3 assignment notebook and then copy over relevant SVM code or you could copy over your A3 code into this notebook).

Learning Goals

- Reinforce signal processing skills from A2 and complement them frequency signal analysis
- Introduce and learn feature extraction and feature classification concepts
- Introduce and learn notions of training + test and reinforce use of k-fold cross validation

Parts

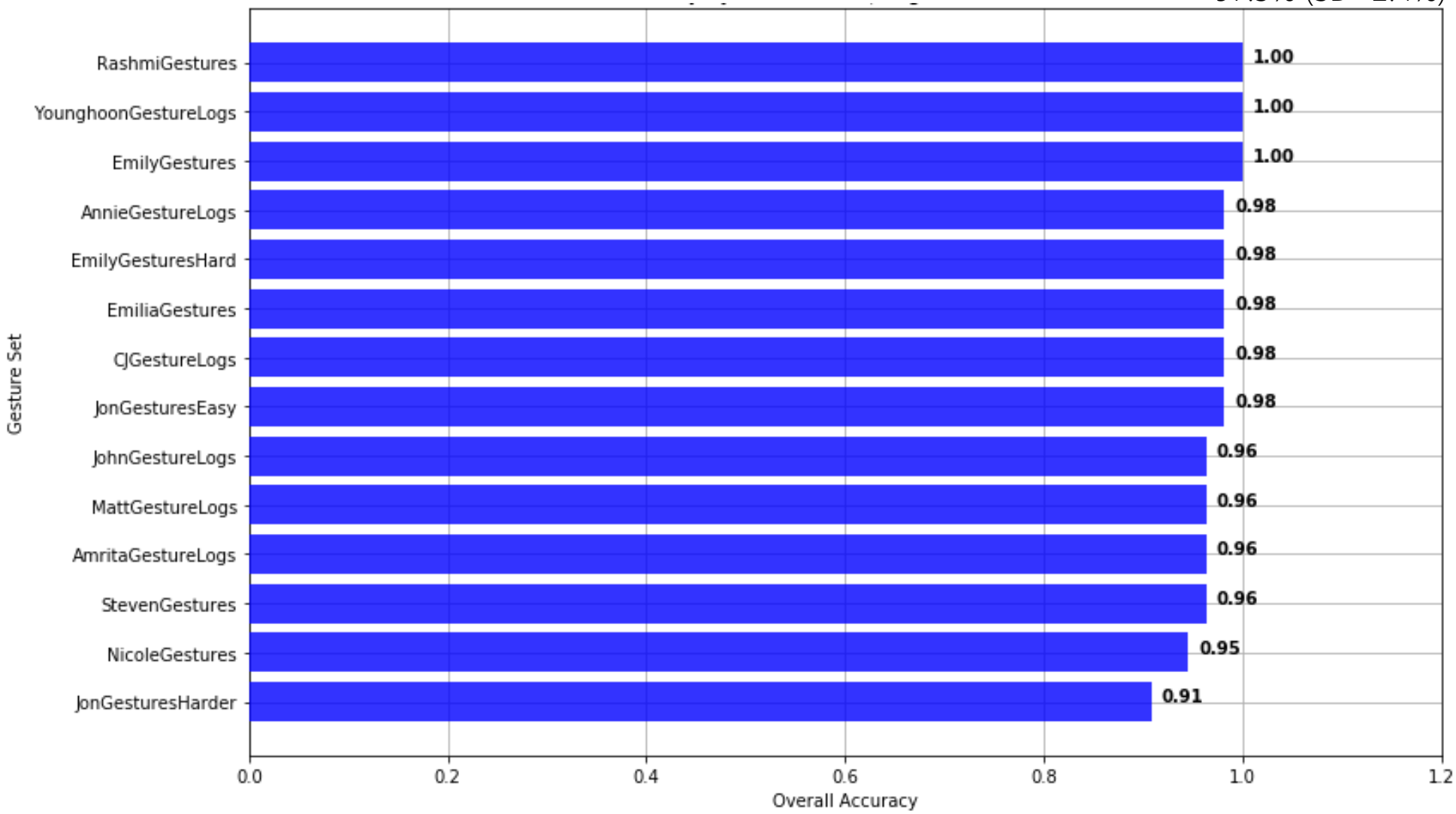
A4 SHAREOUTS

A4 REFLECTION

MY SVM CLASSIFIER: 97.3% ACROSS ALL GESTURE SETS

SVM.SVC(kernel='linear', C=0.1)

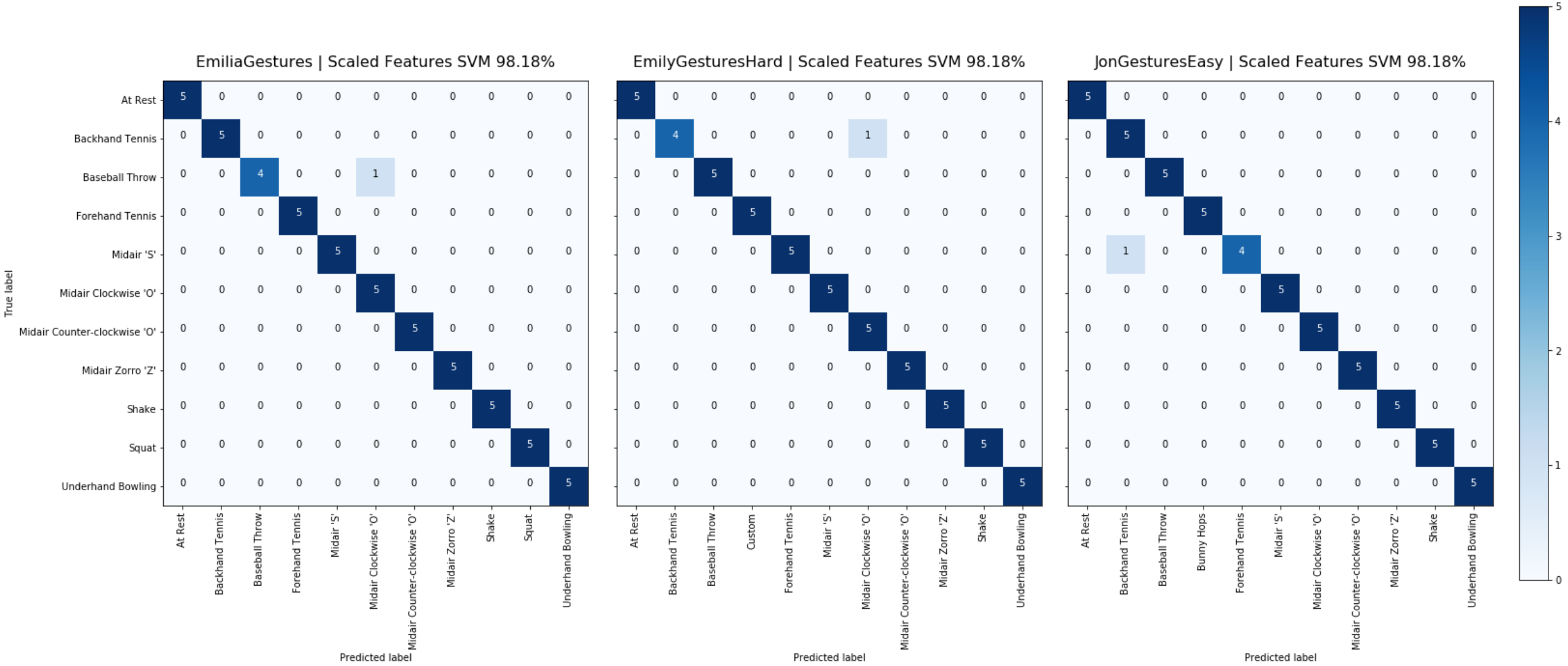
97.3% (SD=2.4%)



A4 REFLECTION

SVM CLASSIFIER MISTAKES

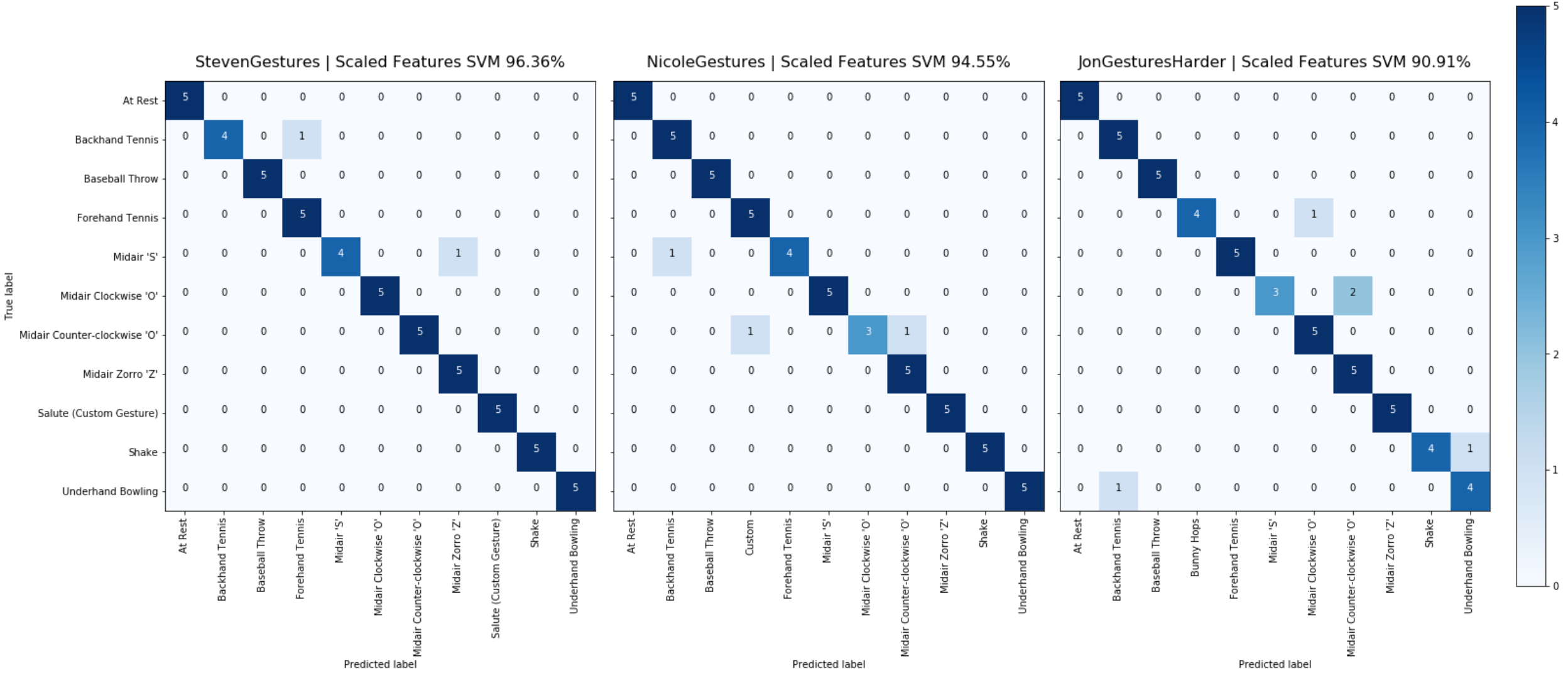
SVM.SVC(kernel='linear', C=0.1)



A4 REFLECTION

SVM CLASSIFIER MISTAKES

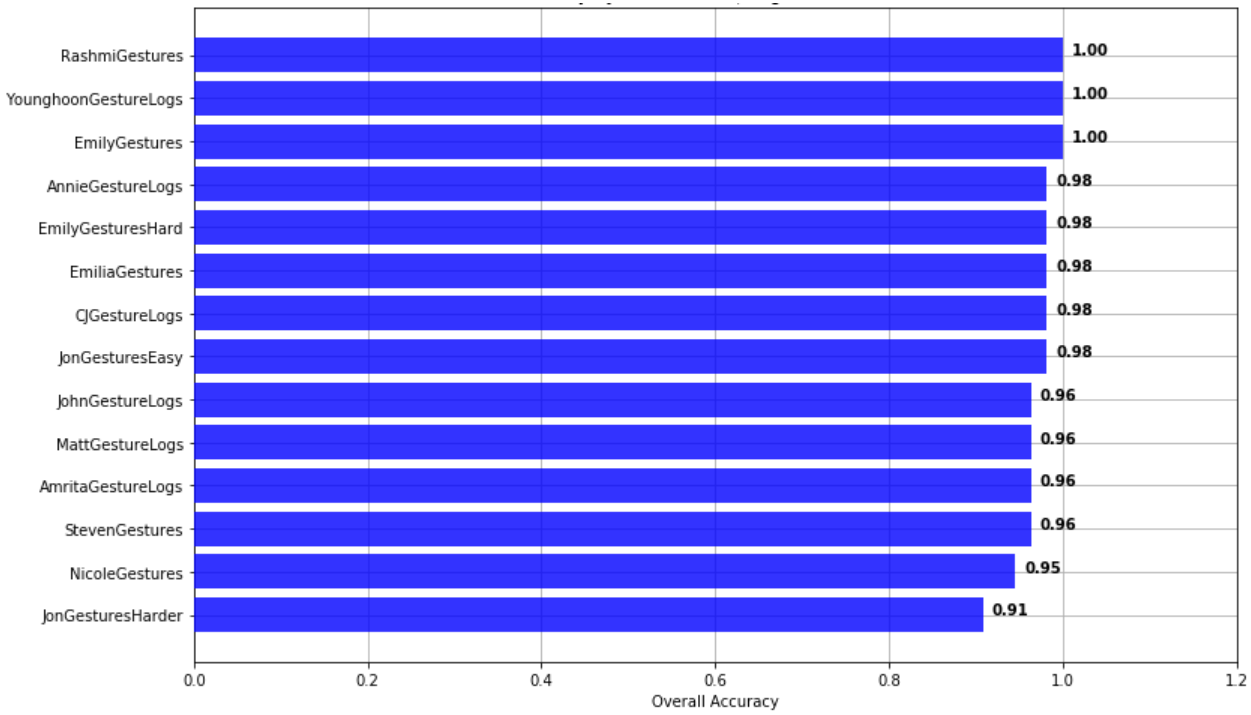
SVM.SVC(kernel='linear', C=0.1)



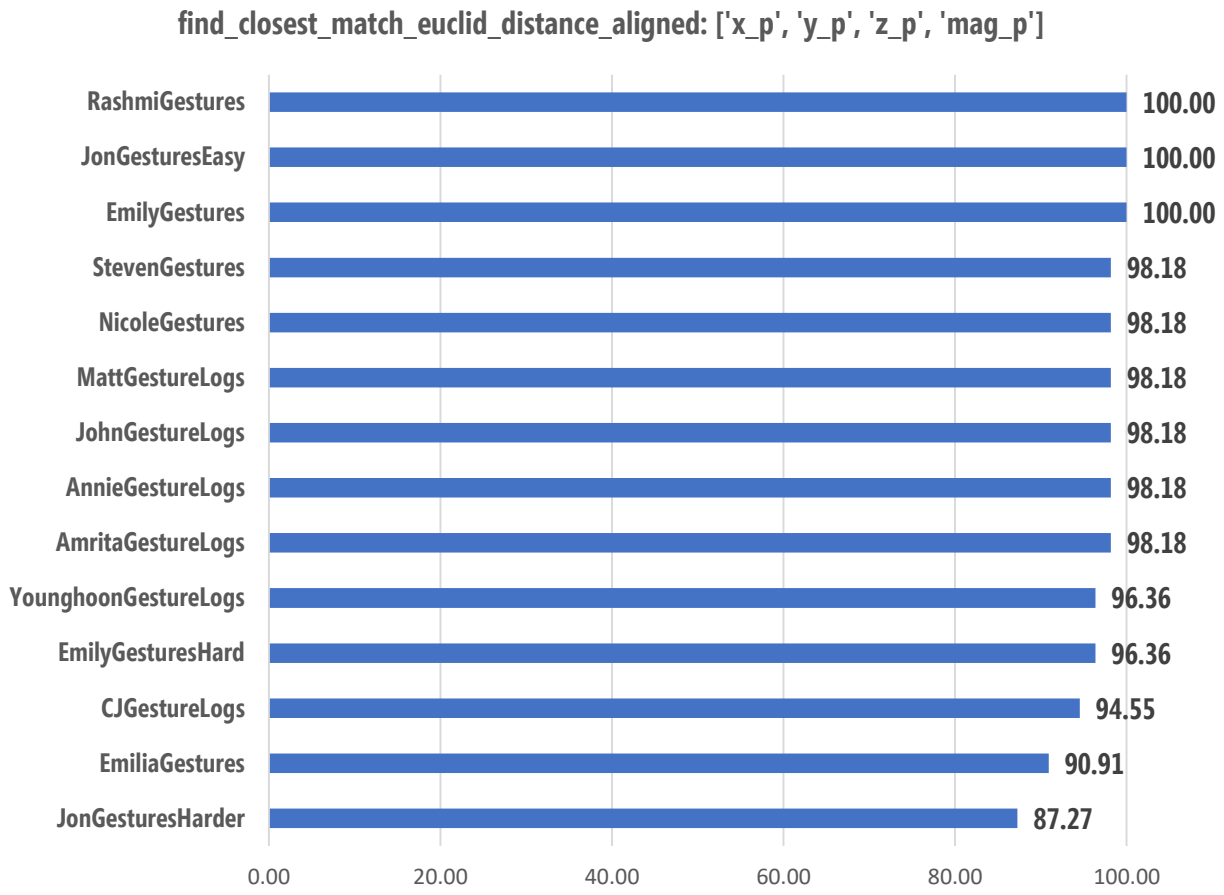
A4 REFLECTION

MY SVM VS. SHAPE-MATCHING

SVM.SVC(kernel='linear', C=0.1) | 97.3% (SD=2.4%)



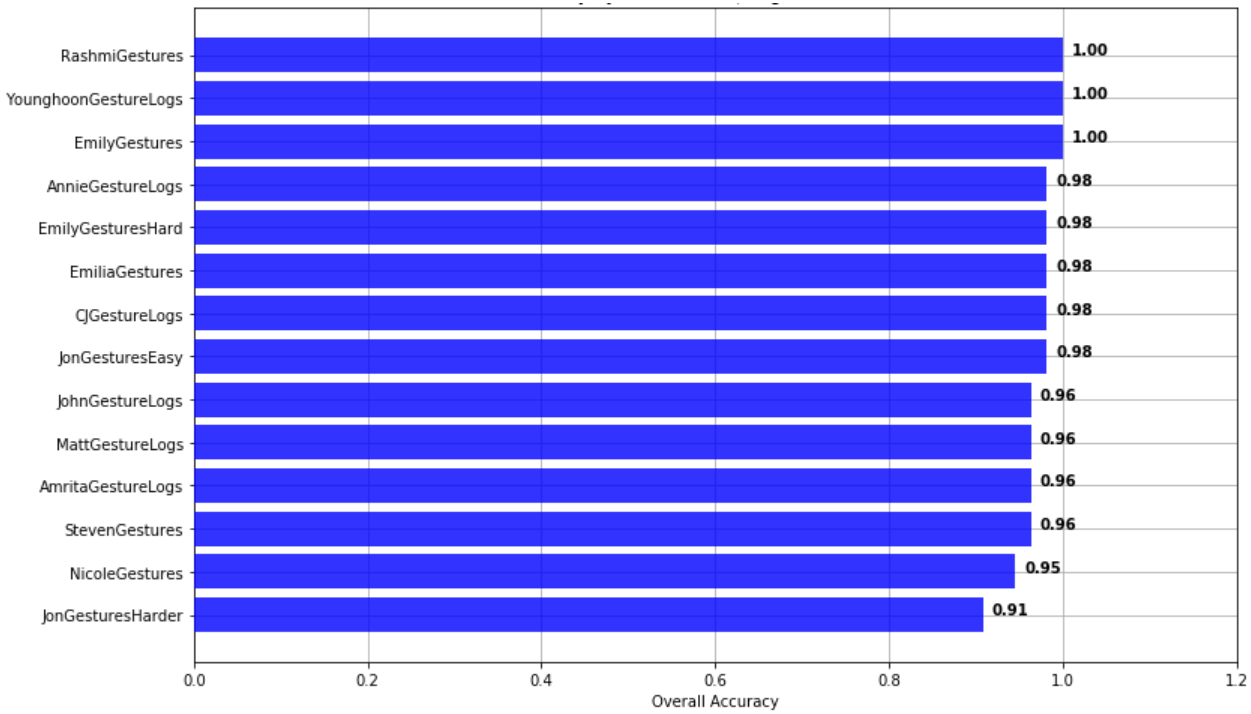
Shape Matching | 96.8% (SD=3.7%)



A4 REFLECTION

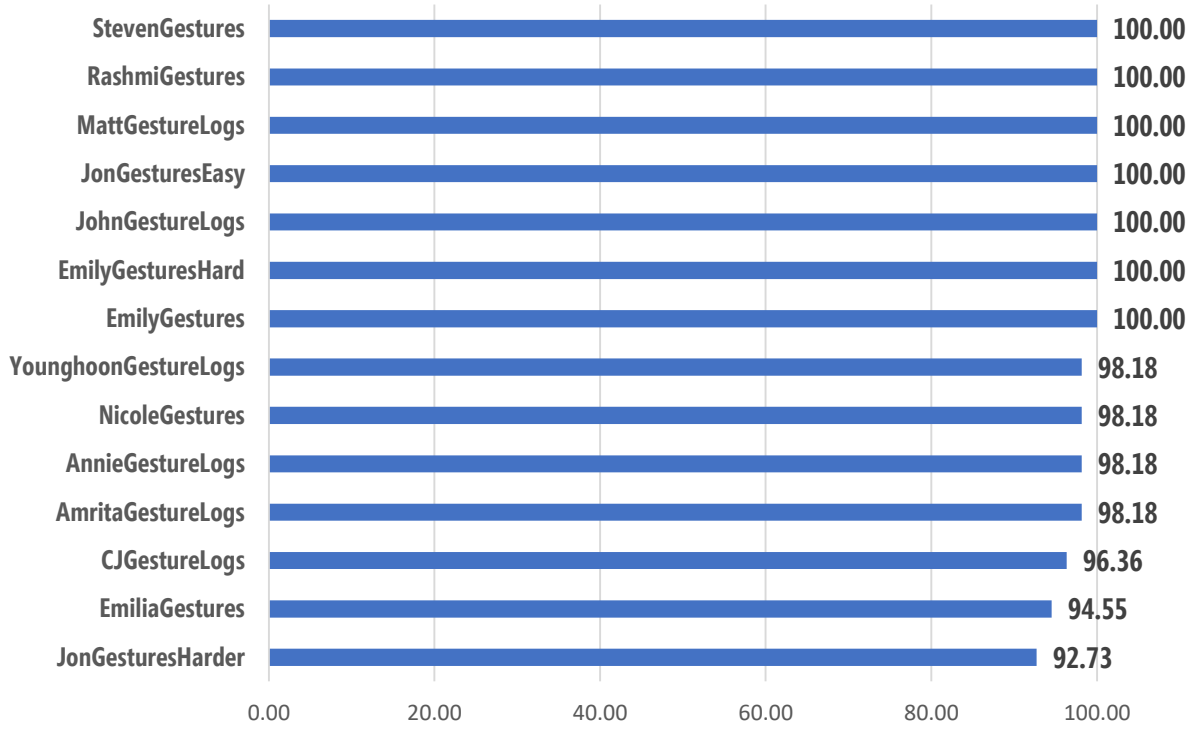
MY SVM VS. SHAPE-MATCHING

SVM.SVC(kernel='linear', C=0.1) | 97.3% (SD=2.4%)



Shape Matching | 98.3% (SD=2.3%)

find_closest_match_dtw: ['x_p', 'y_p', 'z_p', 'mag_p']



A4 REFLECTION

PRIOR WORK

Gesture Recognition with a 3-D Accelerometer

Jiahui Wu[1], Gang Pan[1], Daqing Zhang[2], Guande Qi[1], and Shijian Li[1]

[1] Department of Computer Science
Zhejiang University, Hangzhou, 310027, China
{cat_ng,gpan,shijianli}@zju.edu.cn
[2] Handicom Lab, Institut TELECOM SudParis, France
daqing.zhang@it-sudparis.eu

Abstract. Gesture-based interaction, as a natural way for human-computer interaction, has a wide range of applications in ubiquitous computing environment. This paper presents an acceleration-based gesture recognition approach, called *FDSVM (*Frame-based Descriptor and multi-class SVM), which needs only a wearable 3-dimensional accelerometer. With FDSVM, firstly, the acceleration data of a gesture is collected and represented by a frame-based descriptor, to extract the discriminative information. Then a SVM-based multi-class gesture classifier is built for recognition in the nonlinear gesture feature space. Extensive experimental results on a data set with 3360 gesture samples of 12 gestures over weeks demonstrate that the proposed FDSVM approach significantly outperforms other four methods: DTW, Naïve Bayes, C4.5 and HMM. In the user-dependent case, FDSVM achieves the recognition rate of 99.38% for the 4 direction gestures and 95.21% for all the 12 gestures. In the user-independent case, it obtains the recognition rate of 98.93% for 4 gestures and 89.29% for 12 gestures. Compared to other accelerometer-based gesture recognition approaches reported in literature FDSVM gives the best resulrs for both user-dependent and user-independent cases.

1 Introduction

As computation is getting to play an important role in enhancing the quality of life, more and more research has been directed towards natural human-computer interaction. In a smart environment, people usually hope to use the most natural and convenient ways to express their intentions and interact with the environment. Button pressing, often used in the remote control panel, provides the most traditional means of giving commands to household appliances. Such kind of operation, however, is not natural and sometimes even inconvenient, especially for elders or visually disabled people who are not able to distinguish the buttons on the device. In this regard, gesture-based interaction offers an alternative way in a smart environment.

Most of previous work on gesture recognition has been based on computer vision techniques [12]. However, the performance of such vision-based approaches depends strongly on the lighting condition and camera facing angles, which greatly restricts its applications in the smart environments. Suppose you are enjoying movies in your home theater with all the lights off. If you intend to change the volume of TV with gesture, it turns out to be rather difficult to accurately recognize the gesture under poor lighting condition using a camera based system. In addition, it is also uncomfortable

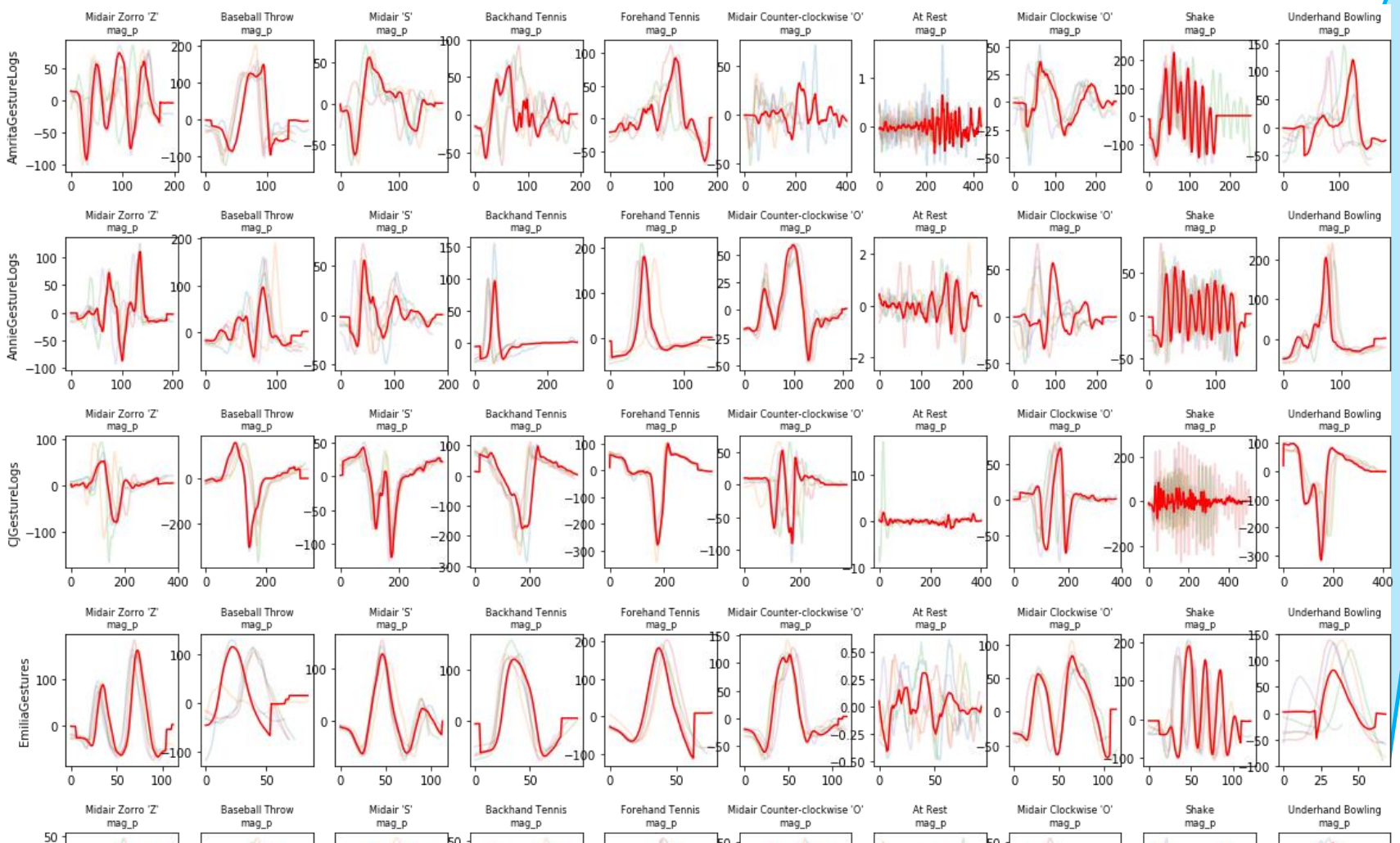
D. Zhang et al. (Eds.): UIC 2009, LNCS 5585, pp. 25–38, 2009.

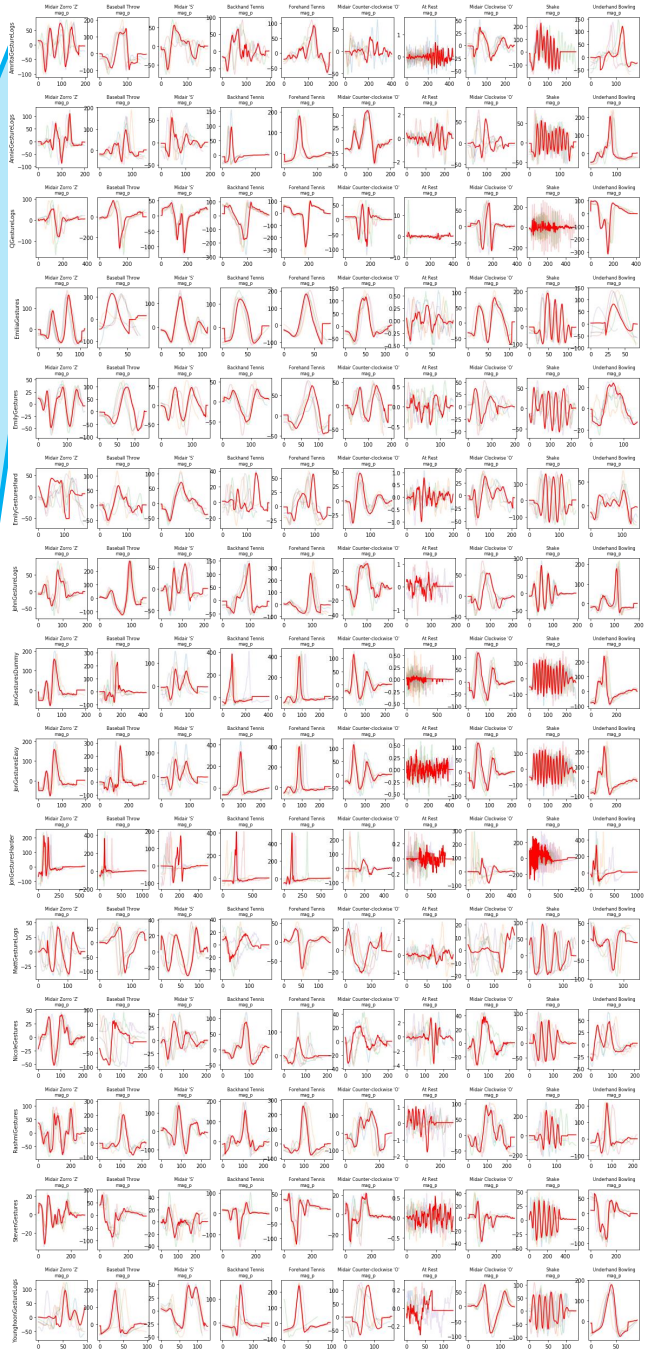
There are usually two ways to evaluate the performance of the gesture recognition algorithms: *user-dependent* and *user-independent*. Previous work focuses more on user-dependent gesture recognition [1,6,18], in which each user is required to perform a couple of gestures as training/template samples before using the system. One of the solutions to this problem is to reduce the user efforts in training by adding artificial noise to the original gesture data to augment the training data [5]. The user-independent gesture recognition does not require a user to enroll any gesture samples for the system before using it, where the model for classification has been well-trained before and the algorithm does not depend on users. The user-independent gesture recognition is more difficult than the user-dependent one since there is much more variation for each same gesture.

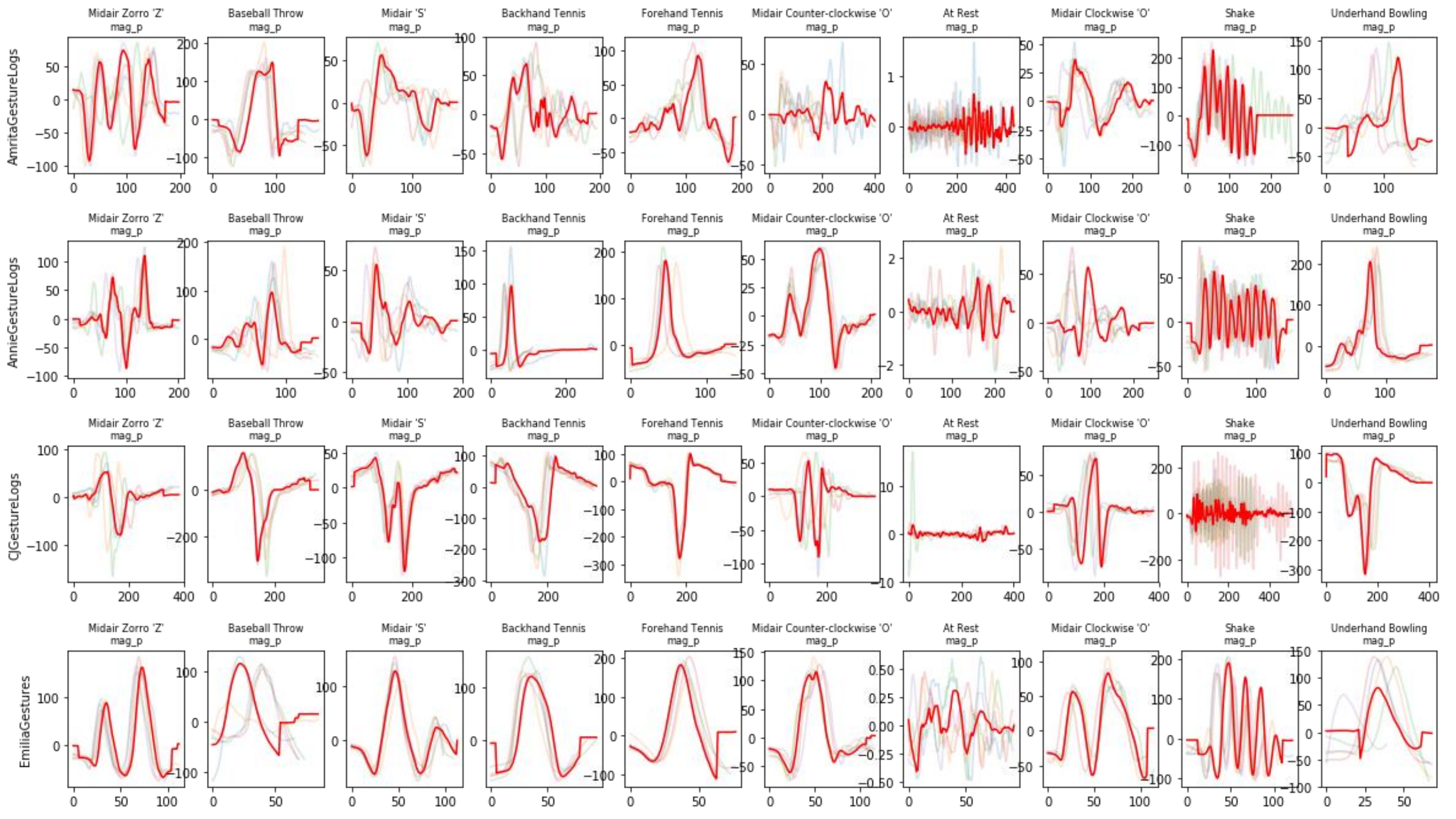
Source: Wu *et al.*, Gesture Recognition with a 3-D Accelerometer, UIC'09 (link)

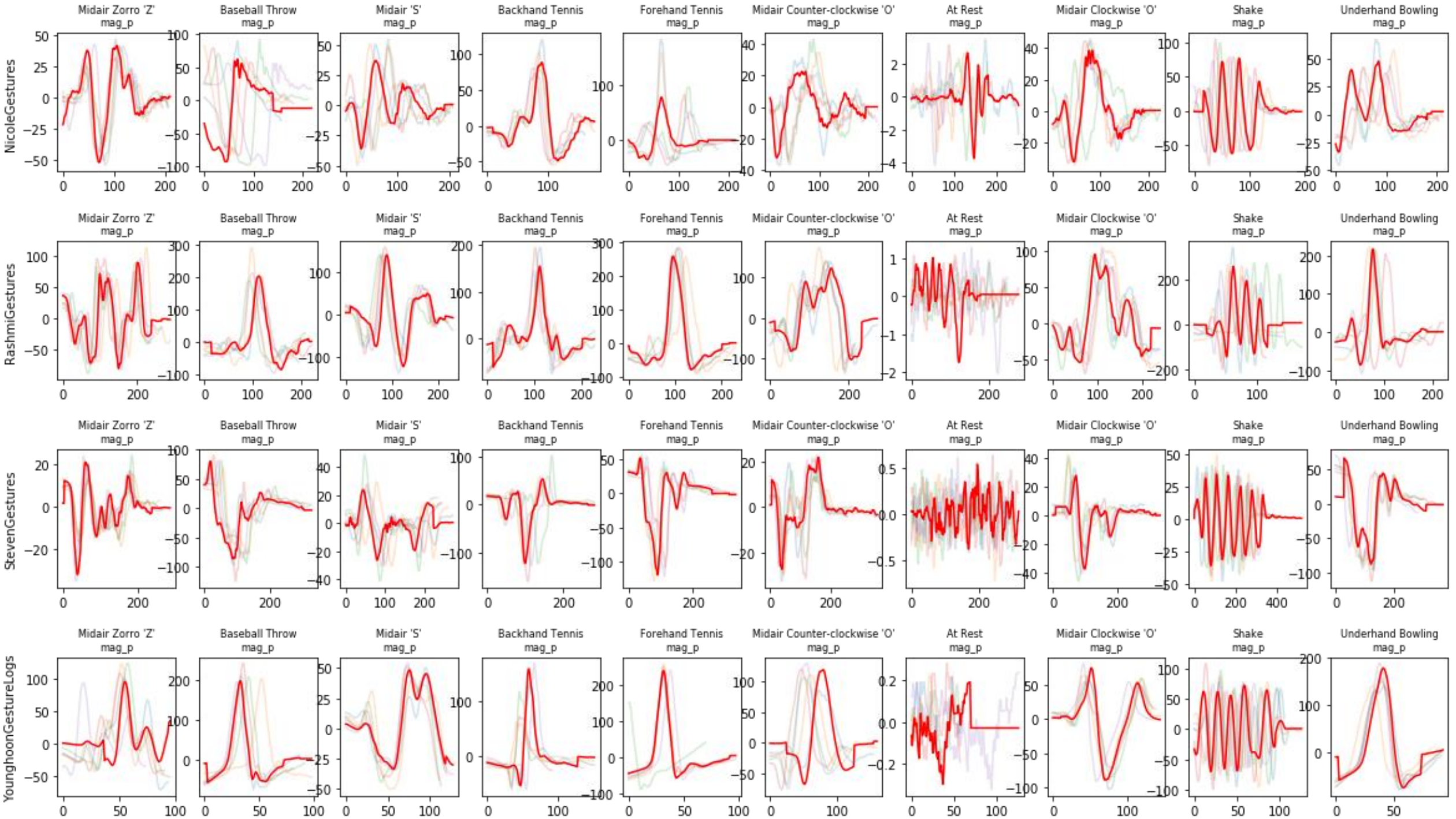
A4 REFLECTION

EXAMINING SIMILARITY ACROSS GESTURE SETS






AmritaGestureLogs
AnnieGestureLogs
CJGestureLogs
EmiliaGestures
Midair Zorro 'Z'
Baseball Throw
Midair 'S'
Backhand Tennis
Forehand Tennis
Midair Counter-clockwise 'O'
At Rest
Midair Clockwise 'O'
Shake
Underhand Bowling
mag_p

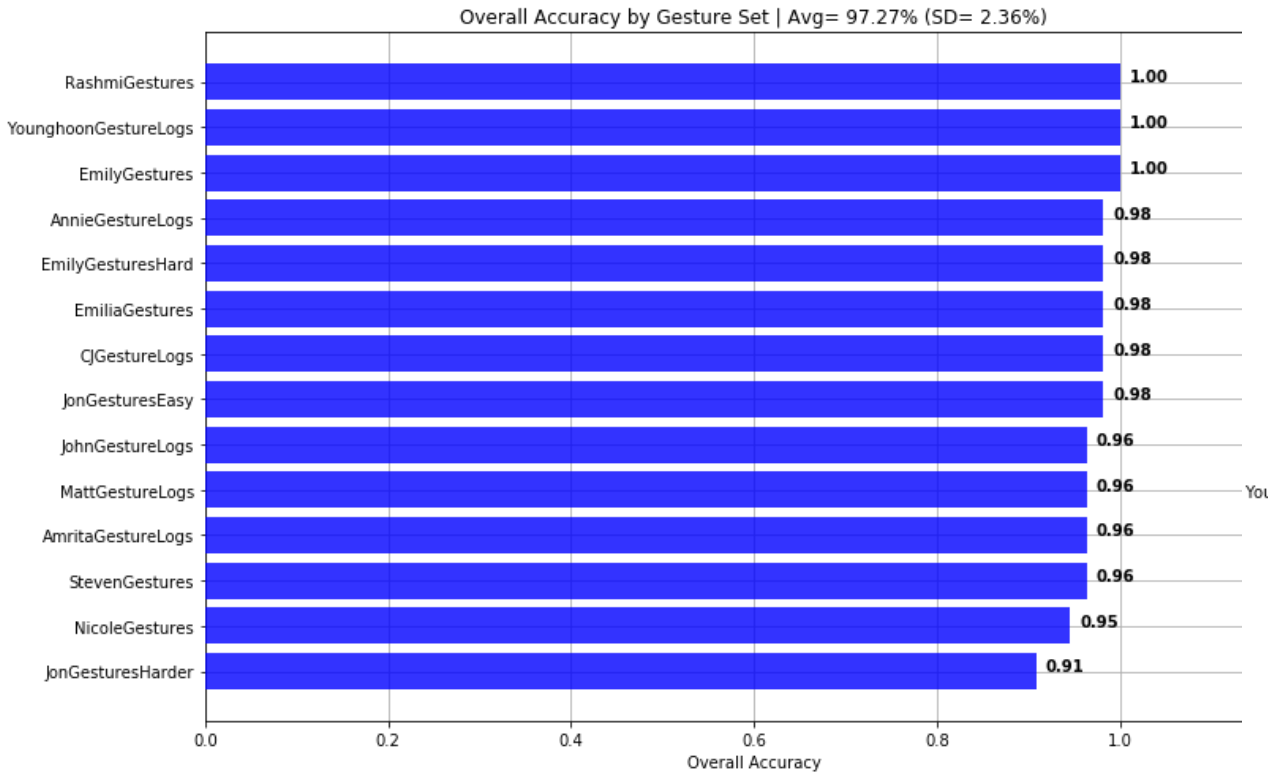

NicoleGestures
RashmiGestures
StevenGestures
YounghoonGestureLogs
Midair Zorro 'Z'
Baseball Throw
Midair 'S'
Backhand Tennis
Forehand Tennis
Midair Counter-clockwise 'O'
At Rest
Midair Clockwise 'O'
Shake
Underhand Bowling
mag_p

A4 REFLECTION

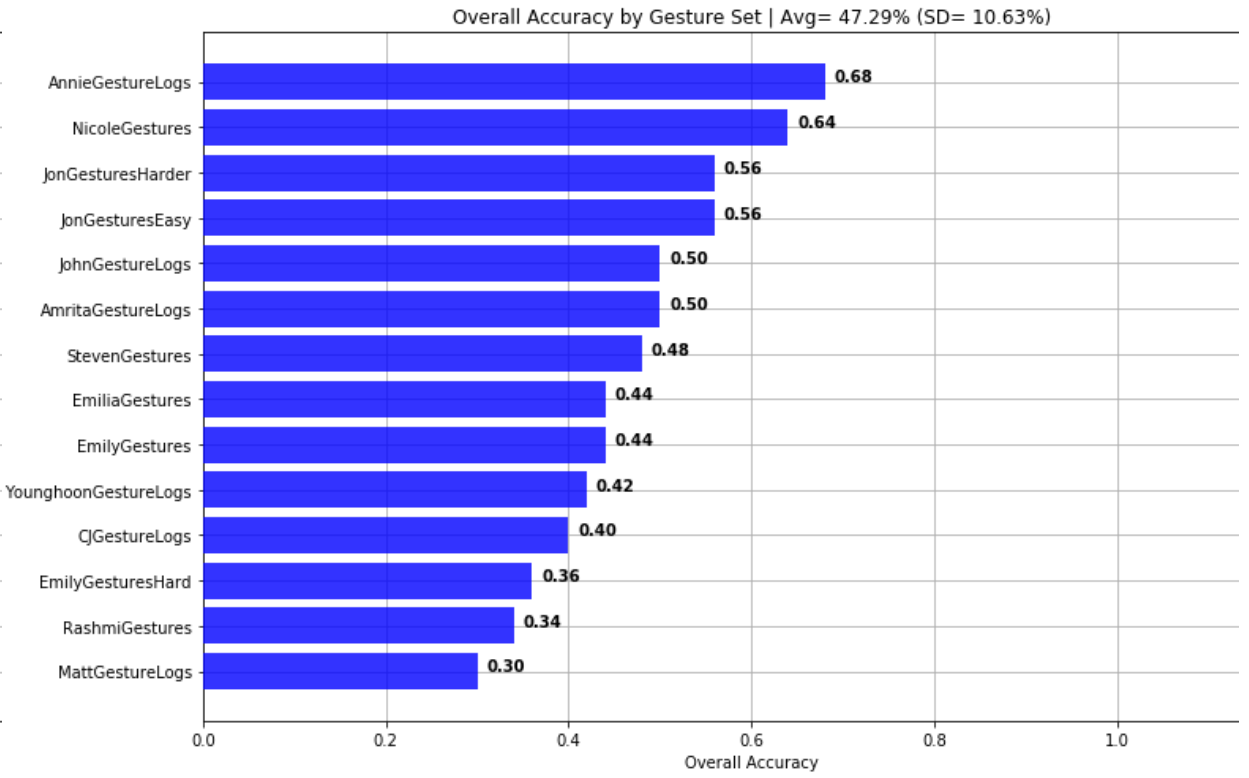
SVM WITHIN VS. CROSS-USER TRAIN/TEST

SVM.SVC(kernel='linear', C=0.01)

Train/Test **Within User** Using K-Fold: **97.3%**



Train/Test **Across Users**: **47.3%**



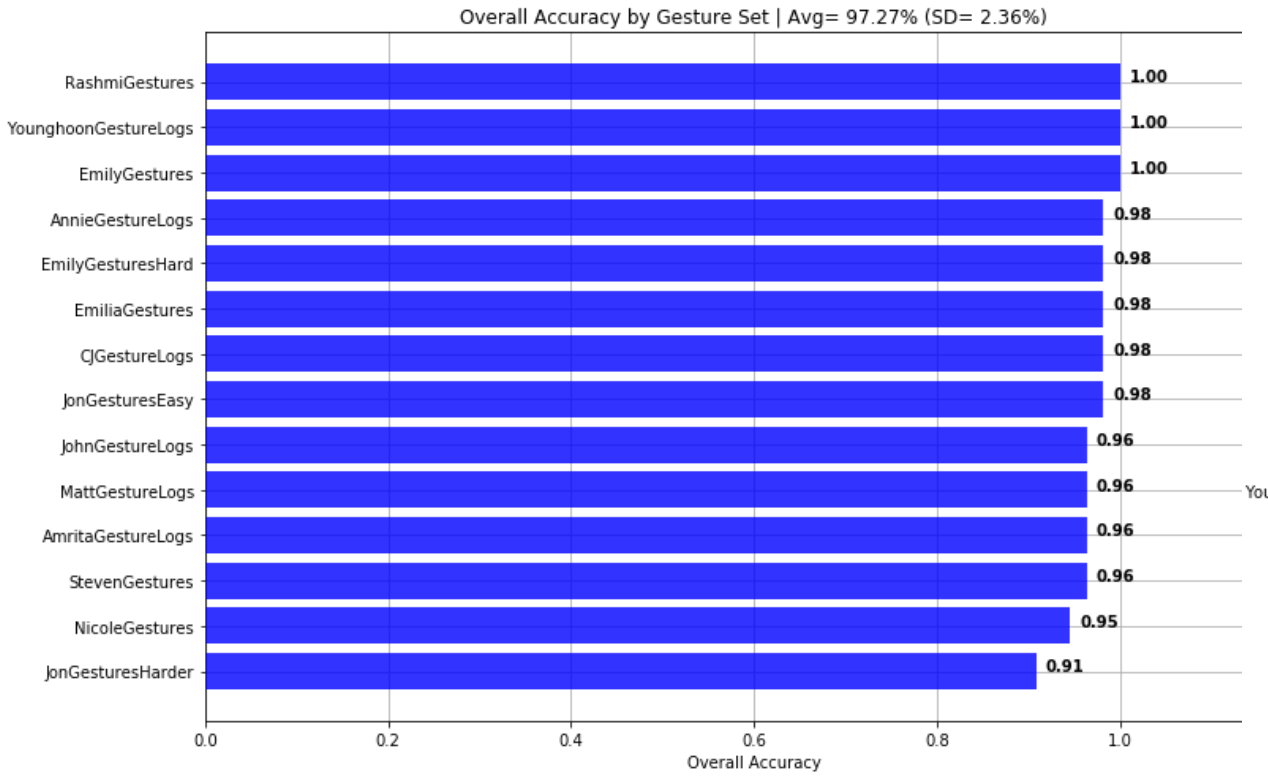
A4 REFLECTION

SVM WITHIN VS. CROSS-USER TRAIN/TE

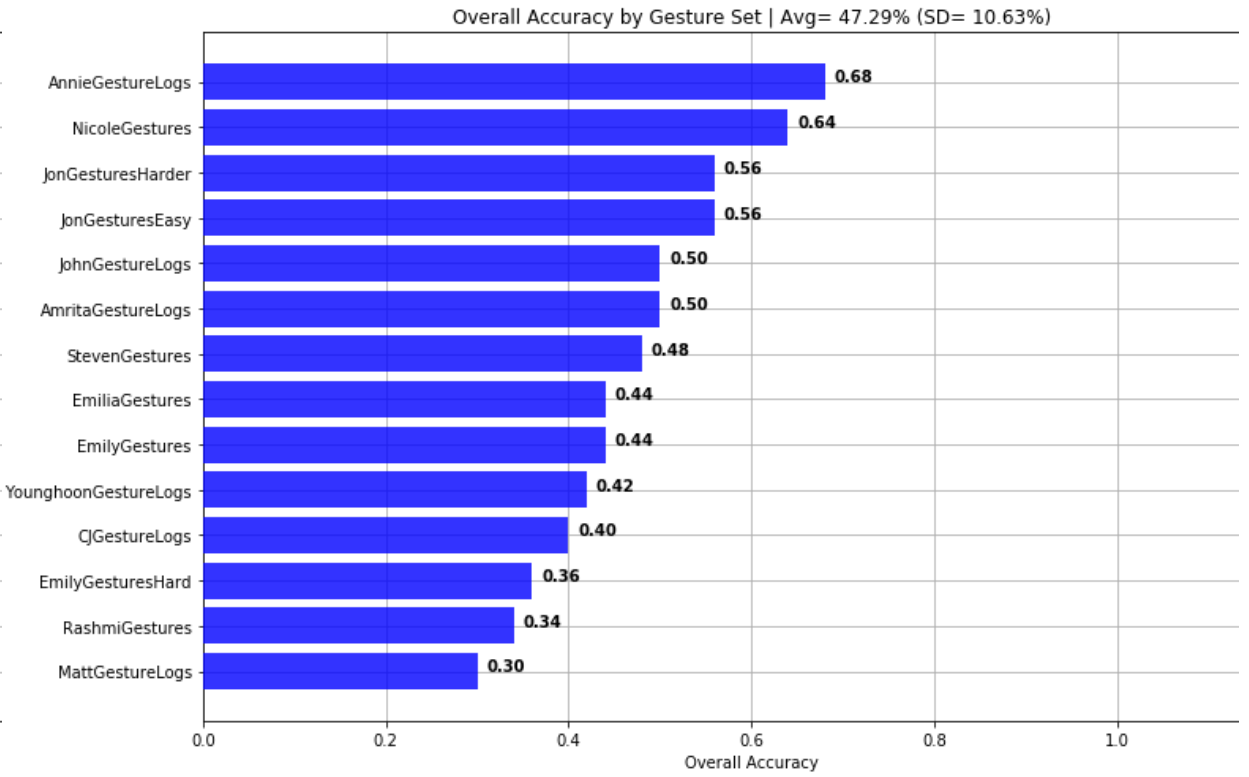
SVM.SVC(kernel='linear', C=0.01)

For this, I **trained on N-1 gesture sets** and tested on the remaining gesture set

Train/Test **Within User** Using K-Fold: **97.3%**



Train/Test **Across Users**: **47.3%**



A4 REFLECTION

ACTIVITY: BRAINSTORM

How might we **improve** cross-user gesture classification?

A4 REFLECTION

PRIOR WORK

Gesture Recognition with a 3-D Accelerometer

Jiahui Wu[1], Gang Pan[1], Daqing Zhang[2], Guande Qi[1], and Shijian Li[1]

[1] Department of Computer Science
Zhejiang University, Hangzhou, 310027, China
{cat_ng,gpan,shijianli}@zju.edu.cn
[2] Handicom Lab, Institut TELECOM SudParis, France
daqing.zhang@it-sudparis.eu

Abstract. Gesture-based interaction, as a natural way for human-computer interaction, has a wide range of applications in ubiquitous computing environment. This paper presents an acceleration-based gesture recognition approach, called *FDSVM* (Frame-based Descriptor and multi-class SVM), which needs only a wearable 3-dimensional accelerometer. With FDSVM, firstly, the acceleration data of a gesture is collected and represented by a frame-based descriptor, to extract the discriminative information. Then a SVM-based multi-class gesture classifier is built for recognition in the nonlinear gesture feature space. Extensive experimental results on a data set with 3360 gesture samples of 12 gestures over weeks demonstrate that the proposed FDSVM approach significantly outperforms other four methods: DTW, Naïve Bayes, C4.5 and HMM. In the user-dependent case, FDSVM achieves the recognition rate of 99.38% for the 4 direction gestures and 95.21% for all the 12 gestures. In the user-independent case, it obtains the recognition rate of 98.93% for 4 gestures and 89.29% for 12 gestures. Compared to other accelerometer-based gesture recognition approaches reported in literature FDSVM gives the best resulrs for both user-dependent and user-independent cases.

1 Introduction

As computation is getting to play an important role in enhancing the quality of life, more and more research has been directed towards natural human-computer interaction. In a smart environment, people usually hope to use the most natural and convenient ways to express their intentions and interact with the environment. Button pressing, often used in the remote control panel, provides the most traditional means of giving commands to household appliances. Such kind of operation, however, is not natural and sometimes even inconvenient, especially for elders or visually disabled people who are not able to distinguish the buttons on the device. In this regard, gesture-based interaction offers an alternative way in a smart environment.

Most of previous work on gesture recognition has been based on computer vision techniques [12]. However, the performance of such vision-based approaches depends strongly on the lighting condition and camera facing angles, which greatly restricts its applications in the smart environments. Suppose you are enjoying movies in your home theater with all the lights off. If you intend to change the volume of TV with gesture, it turns out to be rather difficult to accurately recognize the gesture under poor lighting condition using a camera based system. In addition, it is also uncomfortable

D. Zhang et al. (Eds.): UIC 2009, LNCS 5585, pp. 25–38, 2009.

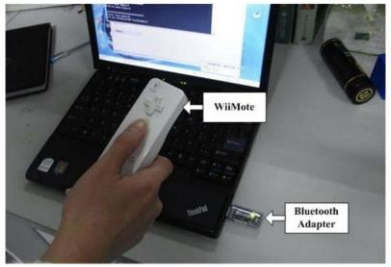


Fig. 3. Acquisition devices of gesture acceleration data

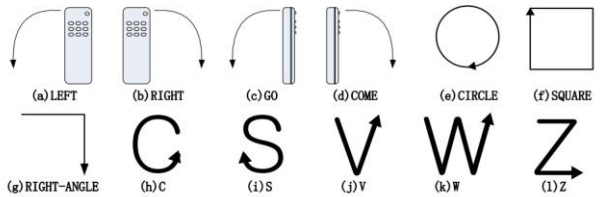


Fig. 4. Twelve gestures in the data set. (a) – (d) describe the gestures to swing the remote to left, right, forward or backward centered at the bottom point of the remote. (e) – (l) describe the gestures to draw a shape or letter.

Ten people participated in the data collection, including two female and eight male students aged between 21 and 25. Each student was asked to perform each gesture for 28 times over two weeks, namely 336 gesture samples in total. Every 12 different gestures performed sequentially was regarded as one set of gesture data. In order to consider variability of user performance, a participant was not allowed to perform more than two sets each time and not more than twice per day. Some acceleration data samples of the same gesture from a single participant are depicted in Fig.5, which obviously demonstrate the large variation of a gesture.

Table 1. Twelve gestures are divided into 4 groups for extensive evaluation

No.	Group Name	Included Gestures
1	Direction	LEFT, RIGHT, GO, COME
2	Direction+Shape	LEFT, RIGHT, GO, COME, CIRCLE, SQUARE, RIGHT-ANGLE
3	One-stroke Letter	C, S, V, W, Z, CIRCLE(O), RIGHT-ANGLE(L)
4	All	LEFT, RIGHT, GO, COME, CIRCLE, SQUARE, RIGHT-ANGLE, C, S, V, W, Z

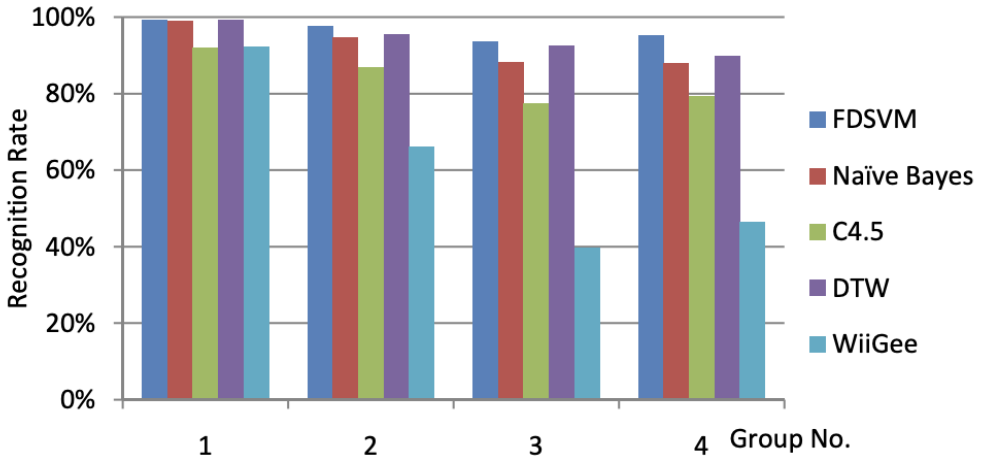


Fig. 8. The experimental results for the user-dependent case

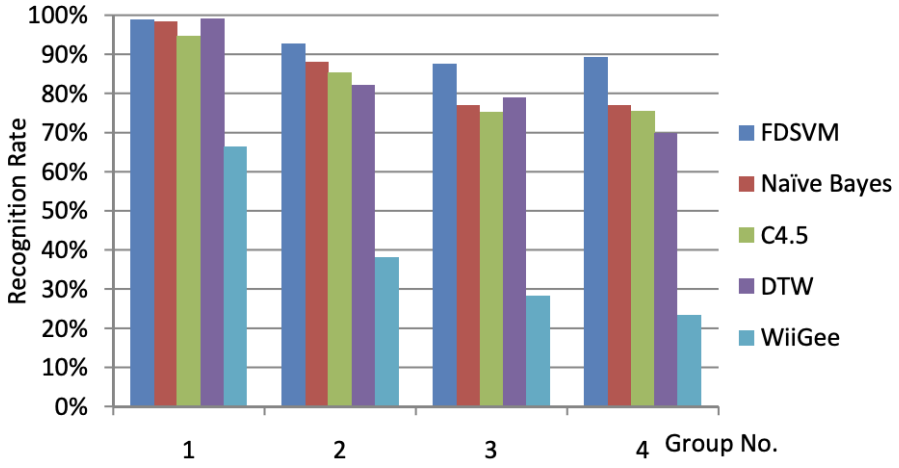


Fig. 9. The experimental result for the user-independent case

Source: Wu *et al.*, Gesture Recognition with a 3-D Accelerometer, UIC'09 (link)

A4 REFLECTION

PRIOR WORK: WHY DO THEY DO BETTER?

Controlled data collection

Same controller with same orientation

Improved algorithm with sliding window SVM

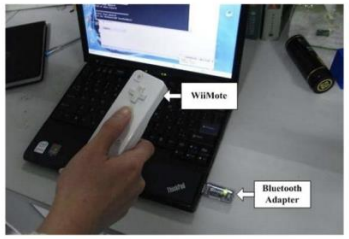


Fig. 3. Acquisition devices of gesture acceleration data

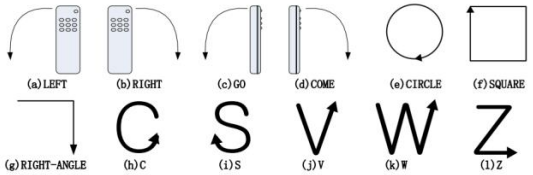


Fig. 4. Twelve gestures in the data set. (a) – (d) describe the gestures to swing the remote to left, right, forward or backward centered at the bottom point of the remote. (e) – (l) describe the gestures to draw a shape or letter.

Ten people participated in the data collection, including two female and eight male students aged between 21 and 25. Each student was asked to perform each gesture for 28 times over two weeks, namely 336 gesture samples in total. Every 12 different gestures performed sequentially was regarded as one set of gesture data. In order to consider variability of user performance, a participant was not allowed to perform more than two sets each time and not more than twice per day. Some acceleration data samples of the same gesture from a single participant are depicted in Fig.5, which obviously demonstrate the large variation of a gesture.

Table 1. Twelve gestures are divided into 4 groups for extensive evaluation

No.	Group Name	Included Gestures
1	Direction	LEFT, RIGHT, GO, COME
2	Direction+Shape	LEFT, RIGHT, GO, COME, CIRCLE, SQUARE, RIGHT-ANGLE
3	One-stroke Letter	C, S, V, W, Z, CIRCLE(O), RIGHT-ANGLE(L)
4	All	LEFT, RIGHT, GO, COME, CIRCLE, SQUARE, RIGHT-ANGLE, C, S, V, W, Z

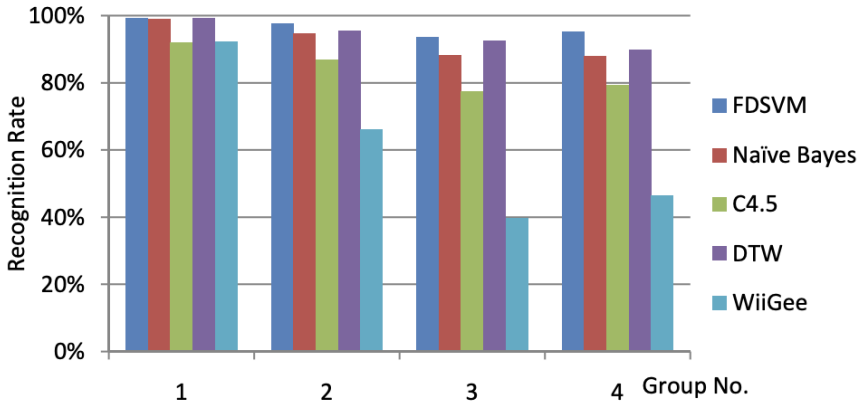


Fig. 8. The experimental results for the user-dependent case

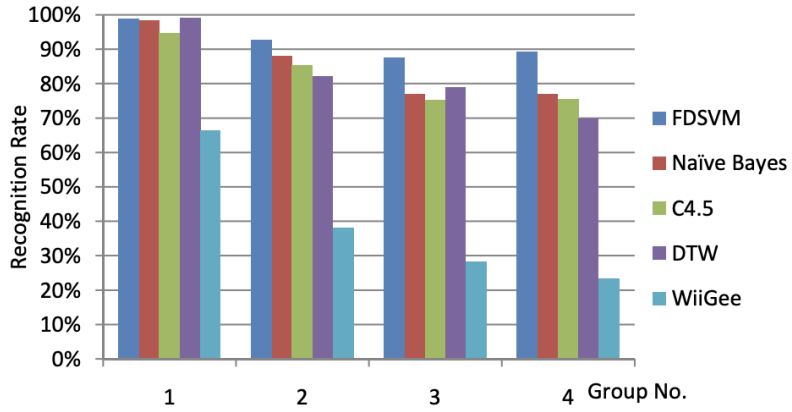


Fig. 9. The experimental result for the user-independent case

Source: Wu *et al.*, Gesture Recognition with a 3-D Accelerometer, UIC'09 (link)

We've been using custom code for **k-fold cross validation** but we could also use **scikit-learn** for this!

MODEL EVALUATION

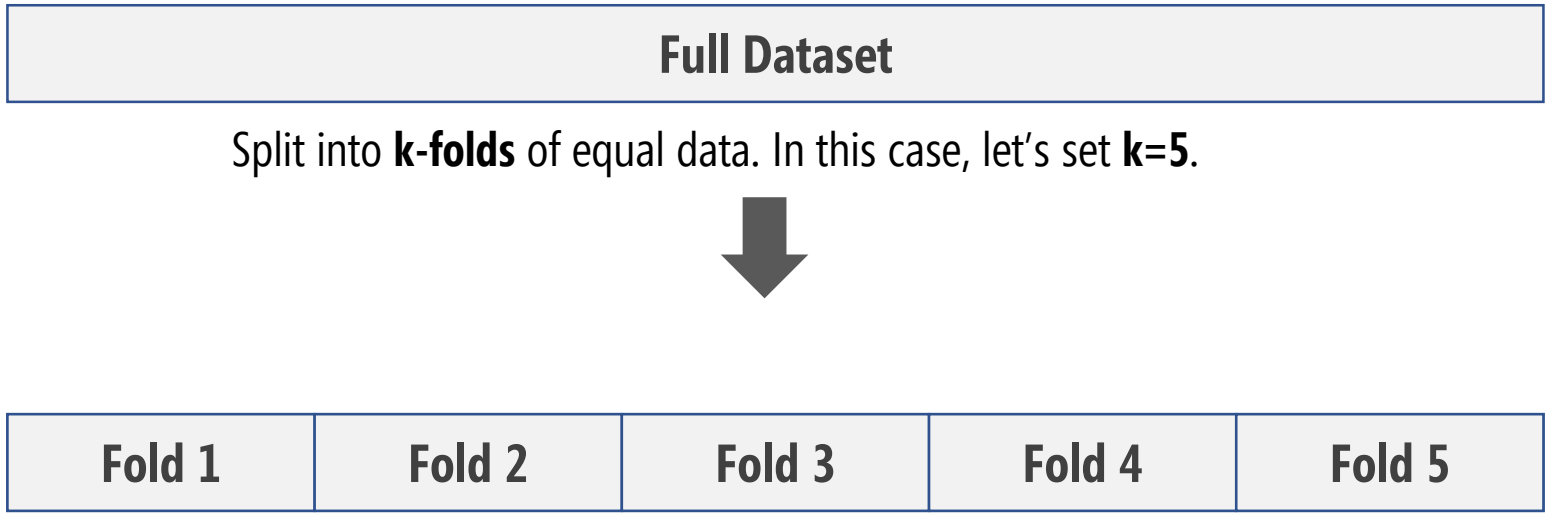
K-FOLD CROSS-VALIDATION

Full Dataset

Split into **k-folds** of equal data. In this case, let's set **k=5**.

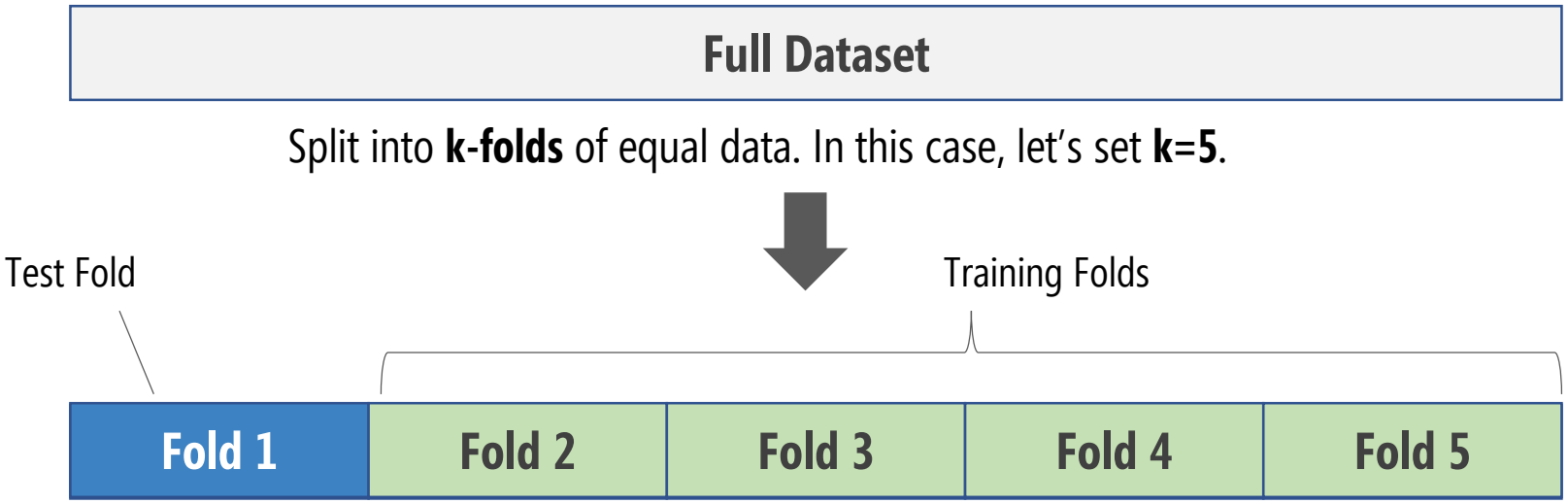
MODEL EVALUATION

K-FOLD CROSS-VALIDATION



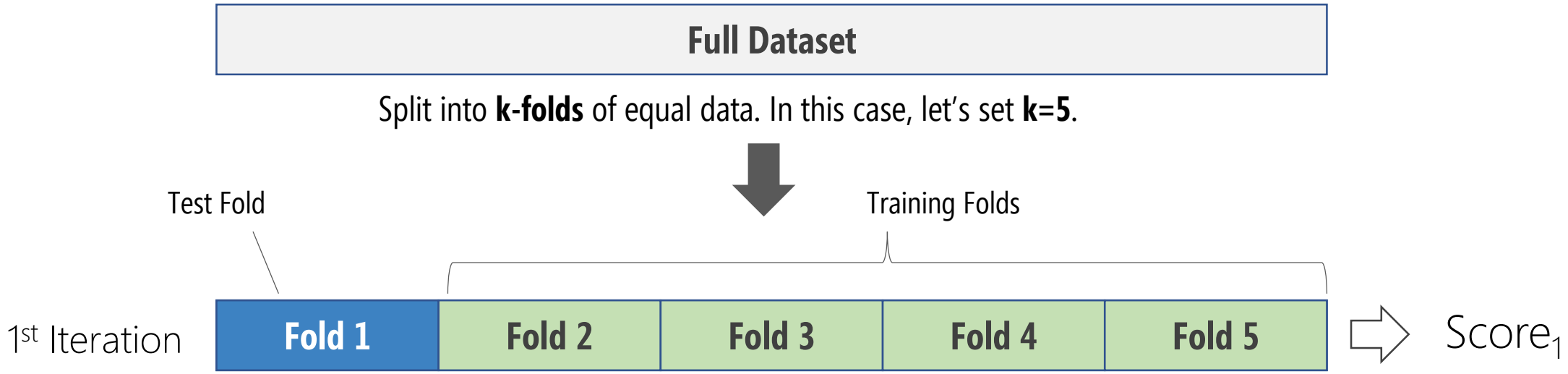
MODEL EVALUATION

K-FOLD CROSS-VALIDATION



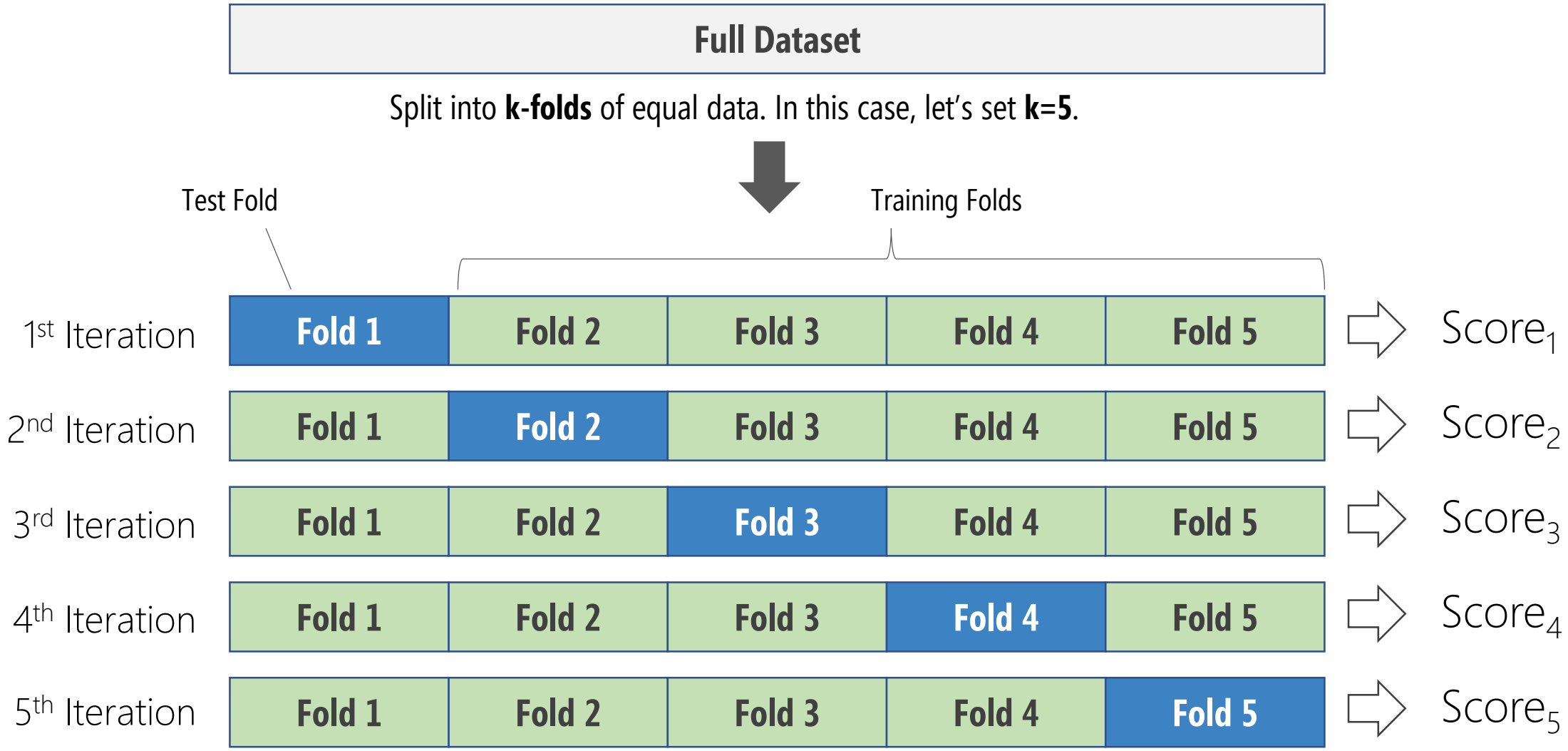
MODEL EVALUATION

K-FOLD CROSS-VALIDATION



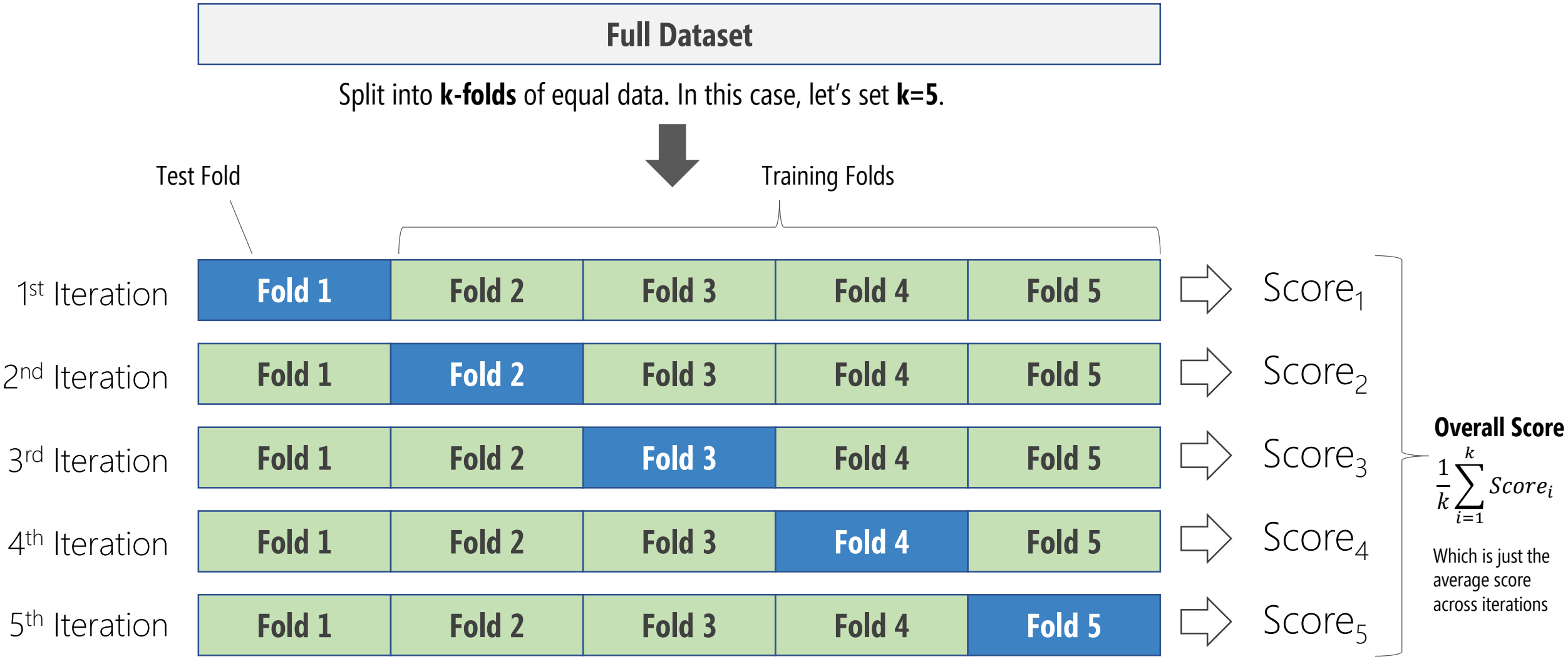
MODEL EVALUATION

K-FOLD CROSS-VALIDATION



MODEL EVALUATION

K-FOLD CROSS-VALIDATION



MODEL EVALUATION

EXAMPLE K-FOLD: DATASET

Full Dataset (sans class labels)

In scikit-learn, the full dataset is referred to as **X**

	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag.mean	x.mean	y.mean	z.mean	mag_p.median	...	r
0	349	-0.003680	0.002231	-0.010517	0.001557	941.461182	487.939828	523.269341	611.925501	-0.002665	...	
1	329	0.006343	0.002568	0.005237	0.003226	941.442396	487.802432	523.452888	611.848024	0.009418	...	
2	266	0.007504	-0.005117	0.001735	0.014058	941.570441	487.887218	523.593985	611.845865	0.009391	...	
3	411	-0.001546	0.002486	0.001883	-0.005967	941.647837	487.922141	523.739659	611.822384	0.003814	...	
4	443	0.005105	-0.000706	0.005241	0.003945	941.695043	487.954853	523.819413	611.801354	0.013190	...	
5	224	1.781186	2.100282	0.081154	0.468409	978.420645	542.017857	508.205357	625.700893	-35.142362	...	
6	219	1.482731	1.722283	0.379335	0.206086	974.979163	535.164384	517.251142	621.908676	-17.023540	...	
7	234	1.341726	1.639692	0.260479	0.141812	974.949140	535.525641	521.311966	617.256410	-26.166495	...	
8	203	1.877071	2.107988	0.644584	0.149121	983.168663	545.261084	523.783251	618.669951	-29.787831	...	
9	254	1.146548	1.376192	0.228604	0.156053	976.387843	537.562992	521.389764	618.937008	-18.353688	...	
10	280	0.771997	1.154652	0.511057	-0.457802	971.279581	517.967857	567.910714	586.092857	-10.356645	...	
11	306	0.563156	0.672193	0.599782	-0.508517	980.382968	525.993464	580.692810	580.218954	-11.435248	...	
12	333	0.550743	0.633333	0.348090	-0.277344	994.310399	548.222222	584.675676	577.948949	-14.705581	...	
13	331	0.608548	0.629733	0.616082	-0.392998	989.807322	544.619335	573.797583	585.661631	-16.626730	...	
14	282	1.331867	1.640106	1.068490	-0.766823	1008.424162	567.460993	584.088652	580.439716	-23.186873	...	
15	357	0.197062	0.117094	0.133367	0.041783	949.635242	516.316527	505.490196	613.871148	-4.187680	...	
16	326	0.201396	0.126150	0.124161	0.055284	955.349994	510.987730	523.539877	612.647239	-3.785490	...	
17	376	0.139551	0.108374	0.110067	-0.008895	959.576622	513.441489	530.388298	611.348404	-3.820511	...	

Class Labels (aka ground truth labels)

In scikit-learn, the full dataset is referred to as **y**

	gesture
0	At Rest
1	At Rest
2	At Rest
3	At Rest
4	At Rest
5	Backhand Tennis
6	Backhand Tennis
7	Backhand Tennis
8	Backhand Tennis
9	Backhand Tennis
10	Baseball Throw
11	Baseball Throw
12	Baseball Throw
13	Baseball Throw
14	Baseball Throw
15	Bunny Hops
16	Bunny Hops
17	Bunny Hops

MODEL EVALUATION

SCIKIT-LEARN KFOLD CLASS

class sklearn.model_selection.KFold(*n_splits='warn', shuffle=False, random_state=None*)

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the User Guide.

Parameters:

n_splits : *int, default=3*

Number of folds. Must be at least 2.

Changed in version 0.20: `n_splits` default value will change from 3 to 5 in v0.22.

shuffle : *boolean, optional*

Whether to shuffle the data before splitting into batches.

random_state : *int, RandomState instance or None, optional, default=None*

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`. Used when `shuffle` == True.

Source: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

MODEL EVALUATION

KFOLD EXAMPLE

```
from sklearn.model_selection import KFold

kf_cross_validator = KFold(n_splits=5, shuffle=True)
print_folds(kf_cross_validator, X, y)
```

```
def print_folds(cross_validator, X, y):
    # Print out the k-fold splits.
    fold_cnt = 0
    for train_index, test_index in cross_validator.split(X, y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        print("TEST FOLD {}".format(fold_cnt))
        for i in test_index:
            print("\t{} {}".format(y[i], trial_indices[i]))
        fold_cnt += 1
```

MODEL EVALUATION

KFOLD EXAMPLE

```
from sklearn.model_selection import KFold

kf_cross_validator = KFold(n_splits=5, shuffle=True)
print_folds(kf_cross_validator, X, y)
```

```
TEST FOLD 0
        At Rest 2
        Backhand Tennis 1
        Bunny Hops 2
        Forehand Tennis 1
        Forehand Tennis 3
        Midair 'S' 1
        Midair Clockwise 'O' 1
        Midair Zorro 'Z' 1
        Shake 3
        Underhand Bowling 2
        Underhand Bowling 4
TEST FOLD 1
        At Rest 0
        At Rest 3
        At Rest 4
        Backhand Tennis 0
        Bunny Hops 4
        Forehand Tennis 4
        Midair Clockwise 'O' 2
        Midair Zorro 'Z' 0
        Midair Zorro 'Z' 2
        Underhand Bowling 1
        Underhand Bowling 3
TEST FOLD 2
        Baseball Throw 0
        Bunny Hops 3
        Forehand Tennis 0
        Forehand Tennis 2
        Midair 'S' 0
        Midair 'S' 3
        Midair 'S' 4
        Midair Clockwise 'O' 3
        Midair Clockwise 'O' 4
        Midair Counter-clockwise 'O'
        Shake 4
TEST FOLD 3
        At Rest 1
        Backhand Tennis 2
        Baseball Throw 1
        Baseball Throw 2
        Baseball Throw 3
        Bunny Hops 0
        Midair Zorro 'Z' 3
        Midair Zorro 'Z' 4
        Shake 0
        Shake 1
        Underhand Bowling 0
TEST FOLD 4
        Backhand Tennis 3
        Backhand Tennis 4
        Baseball Throw 4
        Bunny Hops 1
        Midair 'S' 2
        Midair Clockwise 'O' 0
        Midair Counter-clockwise 'O' 0
        Midair Counter-clockwise 'O' 1
        Midair Counter-clockwise 'O' 3
        Midair Counter-clockwise 'O' 4
        Shake 2
```

Question:
What do you observe about the fold splits?

MODEL EVALUATION

KFOLD EXAMPLE

```
from sklearn.model_selection import KFold

kf_cross_validator = KFold(n_splits=5, shuffle=True)
print_folds(kf_cross_validator, X, y)
```

```
TEST FOLD 0
        At Rest 2
        Backhand Tennis 1
        Bunny Hops 2
        Forehand Tennis 1
        Forehand Tennis 3
        Midair 'S' 1
        Midair Clockwise 'O' 1
        Midair Zorro 'Z' 1
        Shake 3
        Underhand Bowling 2
        Underhand Bowling 4
TEST FOLD 1
        At Rest 0
        At Rest 3
        At Rest 4
        Backhand Tennis 0
        Bunny Hops 4
        Forehand Tennis 4
        Midair Clockwise 'O' 2
        Midair Zorro 'Z' 0
        Midair Zorro 'Z' 2
        Underhand Bowling 1
        Underhand Bowling 3
TEST FOLD 2
        Baseball Throw 0
        Bunny Hops 3
        Forehand Tennis 0
        Forehand Tennis 2
        Midair 'S' 0
        Midair 'S' 3
        Midair 'S' 4
        Midair Clockwise 'O' 3
        Midair Clockwise 'O' 4
        Midair Counter-clockwise 'O'
        Shake 4
TEST FOLD 3
        At Rest 1
        Backhand Tennis 2
        Baseball Throw 1
        Baseball Throw 2
        Baseball Throw 3
        Bunny Hops 0
        Midair Zorro 'Z' 3
        Midair Zorro 'Z' 4
        Shake 0
        Shake 1
        Underhand Bowling 0
TEST FOLD 4
        Backhand Tennis 3
        Backhand Tennis 4
        Baseball Throw 4
        Bunny Hops 1
        Midair 'S' 2
        Midair Clockwise 'O' 0
        Midair Counter-clockwise 'O' 0
        Midair Counter-clockwise 'O' 1
        Midair Counter-clockwise 'O' 3
        Midair Counter-clockwise 'O' 4
        Shake 2
```

An **uneven distribution** of class samples in the folds!

MODEL EVALUATION

STRATIFIED K-FOLD CROSS-VALIDATION

A **Stratified K-Fold** ensures that each fold has the same proportion of each class.

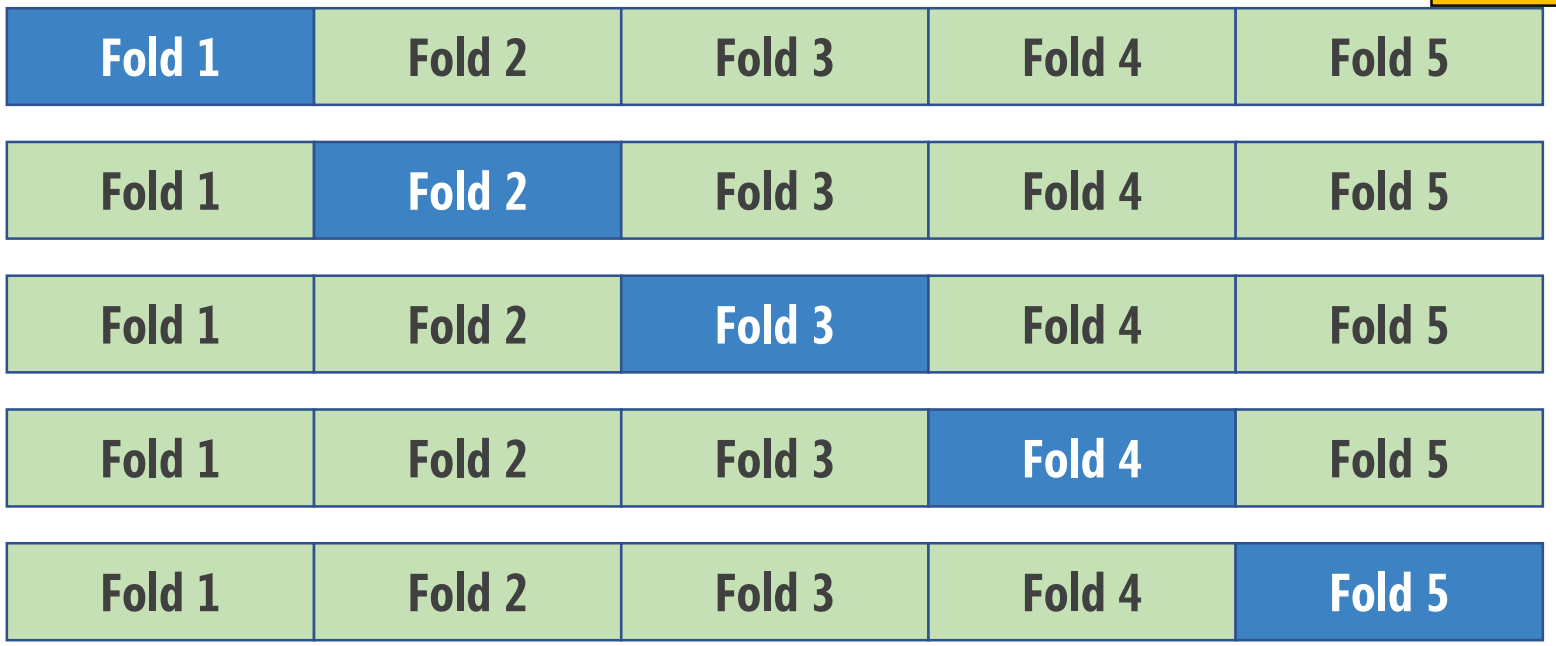
For example, we have **5 samples** of each gesture in our gesture set, so a stratified k-fold would include 1 sample of each in each fold

Full Dataset

Split into **k-folds** of equal data. In this case, let's set **k=5**.

Test Fold

Training Folds



Overall Score

$$\frac{1}{k}\sum_{i=1}^{k} Score_i$$

Which is just the average score across iterations

MODEL EVALUATION

SCIKIT-LEARN STRATIFIEDKFOLD CLASS

class sklearn.model_selection.StratifiedKFold(*n_splits='warn', shuffle=False, random_state=None*)

Stratified K-Folds cross-validator

Provides train/test indices to split data in train/test sets.

This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

Read more in the User Guide.

Parameters:

n_splits : *int, default=3*

Number of folds. Must be at least 2.

Changed in version 0.20: `n_splits` default value will change from 3 to 5 in v0.22.

shuffle : *boolean, optional*

Whether to shuffle each class's samples before splitting into batches.

random_state : *int, RandomState instance or None, optional, default=None*

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`. Used when `shuffle` == True.

Source: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

MODEL EVALUATION

STRATIFIED KFOLD

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 2
        Backhand Tennis 2
        Baseball Throw 0
        Bunny Hops 0
        Forehand Tennis 1
        Midair 'S' 3
        Midair Clockwise 'O' 4
        Midair Counter-clockwise 'O' 1
        Midair Zorro 'Z' 0
        Shake 3
        Underhand Bowling 1
TEST FOLD 1
        At Rest 3
        Backhand Tennis 1
        Baseball Throw 1
        Bunny Hops 2
        Forehand Tennis 4
        Midair 'S' 4
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 2
        Midair Zorro 'Z' 3
        Shake 1
        Underhand Bowling 2
TEST FOLD 2
        At Rest 1
        Backhand Tennis 0
        Baseball Throw 2
        Bunny Hops 3
        Forehand Tennis 3
        Midair 'S' 0
        Midair Clockwise 'O' 0
        Midair Counter-clockwise 'O' 4
        Midair Zorro 'Z' 2
        Shake 4
        Underhand Bowling 3
TEST FOLD 3
        At Rest 0
        Backhand Tennis 4
        Baseball Throw 4
        Bunny Hops 4
        Forehand Tennis 2
        Midair 'S' 2
        Midair Clockwise 'O' 1
        Midair Counter-clockwise 'O' 0
        Midair Zorro 'Z' 1
        Shake 0
        Underhand Bowling 4
TEST FOLD 4
        At Rest 4
        Backhand Tennis 3
        Baseball Throw 3
        Bunny Hops 1
        Forehand Tennis 0
        Midair 'S' 1
        Midair Clockwise 'O' 2
        Midair Counter-clockwise 'O' 3
        Midair Zorro 'Z' 4
        Shake 2
        Underhand Bowling 0
```

MODEL EVALUATION

STRATIFIED KFOLD

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=False)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 0
        Backhand Tennis 0
        Baseball Throw 0
        Bunny Hops 0
        Forehand Tennis 0
        Midair 'S' 0
        Midair Clockwise 'O' 0
        Midair Counter-clockwise 'O' 0
        Midair Zorro 'Z' 0
        Shake 0
        Underhand Bowling 0
TEST FOLD 1
        At Rest 1
        Backhand Tennis 1
        Baseball Throw 1
        Bunny Hops 1
        Forehand Tennis 1
        Midair 'S' 1
        Midair Clockwise 'O' 1
        Midair Counter-clockwise 'O' 1
        Midair Zorro 'Z' 1
        Shake 1
        Underhand Bowling 1
```

```
TEST FOLD 2
        At Rest 2
        Backhand Tennis 2
        Baseball Throw 2
        Bunny Hops 2
        Forehand Tennis 2
        Midair 'S' 2
        Midair Clockwise 'O' 2
        Midair Counter-clockwise 'O' 2
        Midair Zorro 'Z' 2
        Shake 2
        Underhand Bowling 2
TEST FOLD 3
        At Rest 3
        Backhand Tennis 3
        Baseball Throw 3
        Bunny Hops 3
        Forehand Tennis 3
        Midair 'S' 3
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 3
        Midair Zorro 'Z' 3
        Shake 3
        Underhand Bowling 3
```

```
TEST FOLD 4
        At Rest 4
        Backhand Tennis 4
        Baseball Throw 4
        Bunny Hops 4
        Forehand Tennis 4
        Midair 'S' 4
        Midair Clockwise 'O' 4
        Midair Counter-clockwise 'O' 4
        Midair Zorro 'Z' 4
        Shake 4
        Underhand Bowling 4
```

Uh, oh! The same split every time and it's **not random**!

MODEL EVALUATION

WITH SHUFFLE, SPLIT IS DIFFERENT ON EACH EXECUTION

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 2
        Backhand Tennis 4
        Baseball Throw 0
        Bunny Hops 1
        Forehand Tennis 1
        Midair 'S' 1
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 4
        Midair Zorro 'Z' 4
        Shake 1
        Underhand Bowling 0
TEST FOLD 1
        At Rest 4
        Backhand Tennis 1
        Baseball Throw 4
        Bunny Hops 4
        Forehand Tennis 4
        Midair 'S' 0
        Midair Clockwise 'O' 4
        Midair Counter-clockwise 'O' 2
                ...
```

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 3
        Backhand Tennis 2
        Baseball Throw 3
        Bunny Hops 4
        Forehand Tennis 3
        Midair 'S' 3
        Midair Clockwise 'O' 4
        Midair Counter-clockwise 'O' 4
        Midair Zorro 'Z' 4
        Shake 2
        Underhand Bowling 4
TEST FOLD 1
        At Rest 0
        Backhand Tennis 0
        Baseball Throw 0
        Bunny Hops 2
        Forehand Tennis 4
        Midair 'S' 2
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 3
                ...
```

```
from sklearn.model_selection impo

skf = StratifiedKFold(n_splits=5,
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 3
        Backhand Tennis 0
        Baseball Throw 0
        Bunny Hops 2
        Forehand Tennis 4
        Midair 'S' 1
        Midair Clockwise 'O' 2
        Midair Counter-clockwis
        Midair Zorro 'Z' 0
        Shake 0
        Underhand Bowling 1
TEST FOLD 1
        At Rest 1
        Backhand Tennis 1
        Baseball Throw 3
        Bunny Hops 1
        Forehand Tennis 2
        Midair 'S' 4
        Midair Clockwise 'O' 1
        Midair Counter-clockwis
        Midair Zorro 'Z' 3
                ...
```

CONTROL RANDOM SPLITS ACROSS EXECUTIONS

Often it's useful to control random fold splits across code executions so you can compare parameters across experiments

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True,
                      random_state=99)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 2
        Backhand Tennis 4
        Baseball Throw 0
        Bunny Hops 2
        Forehand Tennis 4
        Midair 'S' 1
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 0
        Midair Zorro 'Z' 2
        Shake 2
        Underhand Bowling 4
TEST FOLD 1
        At Rest 0
        Backhand Tennis 3
        Baseball Throw 3
        Bunny Hops 3
        Forehand Tennis 1
        Midair 'S' 4
        Midair Clockwise 'O' 2
        Midair Counter-clockwise 'O' 1
        Midair Zorro 'Z' 3

                ...
```

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True,
                      random_state=99)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 2
        Backhand Tennis 4
        Baseball Throw 0
        Bunny Hops 2
        Forehand Tennis 4
        Midair 'S' 1
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 0
        Midair Zorro 'Z' 2
        Shake 2
        Underhand Bowling 4
TEST FOLD 1
        At Rest 0
        Backhand Tennis 3
        Baseball Throw 3
        Bunny Hops 3
        Forehand Tennis 1
        Midair 'S' 4
        Midair Clockwise 'O' 2
        Midair Counter-clockwise 'O' 1
        Midair Zorro 'Z' 3

                ...
```

If **int**, **random_state is the seed** used by the random number generator

MODEL EVALUATION

CONTROL RANDOM SPLITS ACROSS EXECUTIONS

Often it's useful to control random fold splits across code executions so you can compare parameters across experiments

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True,
                      random_state=3)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 3
        Backhand Tennis 2
        Baseball Throw 0
        Bunny Hops 3
        Forehand Tennis 3
        Midair 'S' 3
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 4
        Midair Zorro 'Z' 3
        Shake 4
        Underhand Bowling 2
TEST FOLD 1
        At Rest 4
        Backhand Tennis 1
        Baseball Throw 4
        Bunny Hops 0
        Forehand Tennis 2
        Midair 'S' 4
        Midair Clockwise 'O' 1
        Midair Counter-clockwise 'O' 0
        Midair Zorro 'Z' 0
                ...
```

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True,
                      random_state=3)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 3
        Backhand Tennis 2
        Baseball Throw 0
        Bunny Hops 3
        Forehand Tennis 3
        Midair 'S' 3
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O' 4
        Midair Zorro 'Z' 3
        Shake 4
        Underhand Bowling 2
TEST FOLD 1
        At Rest 4
        Backhand Tennis 1
        Baseball Throw 4
        Bunny Hops 0
        Forehand Tennis 2
        Midair 'S' 4
        Midair Clockwise 'O' 1
        Midair Counter-clockwise 'O' 0
        Midair Zorro 'Z' 0
                ...
```

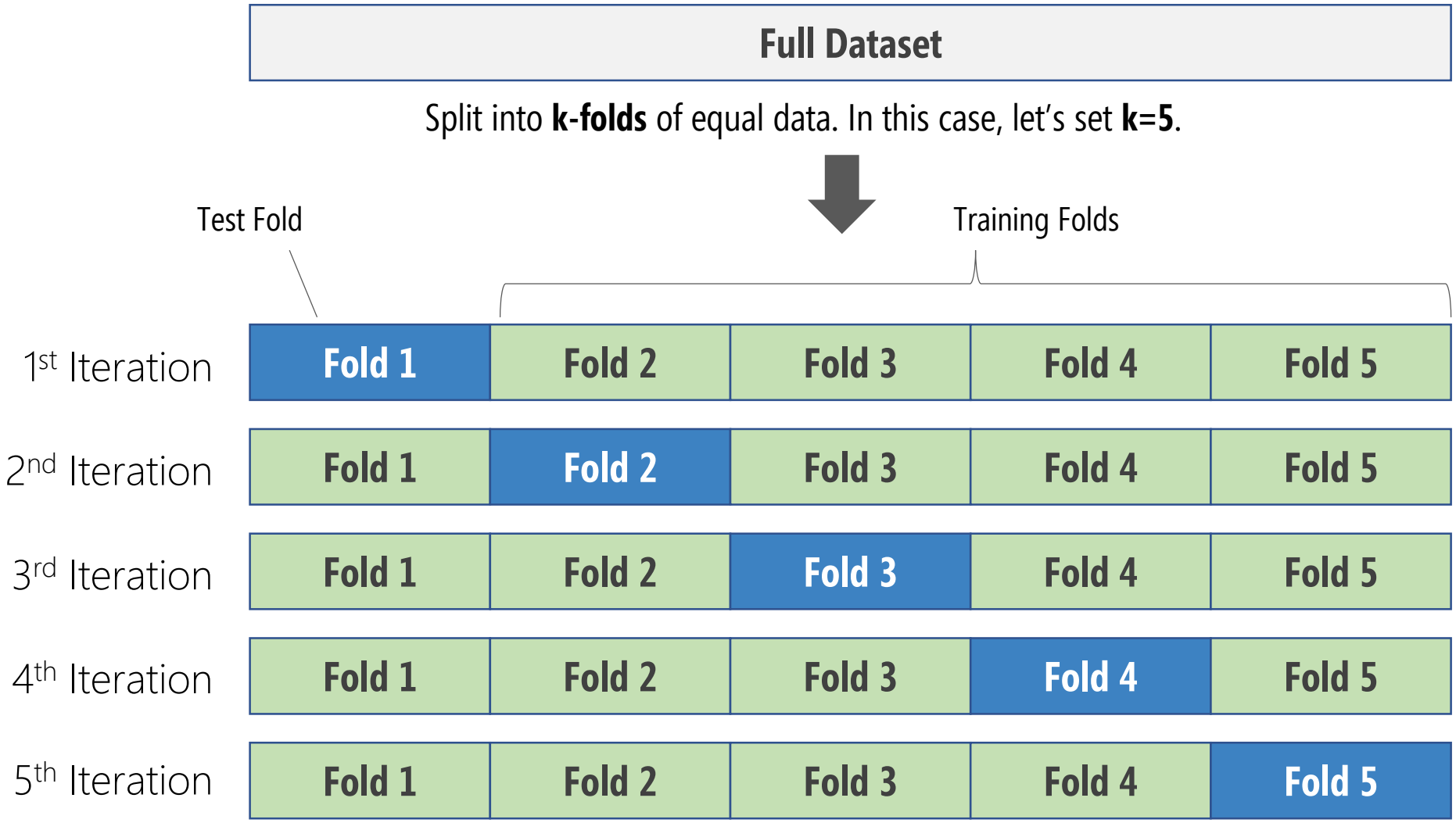
```
from sklearn.model_selection import St

skf = StratifiedKFold(n_splits=5, shuf
                      random_state=3)
print_folds(skf, X, y)
```

```
TEST FOLD 0
        At Rest 3
        Backhand Tennis 2
        Baseball Throw 0
        Bunny Hops 3
        Forehand Tennis 3
        Midair 'S' 3
        Midair Clockwise 'O' 3
        Midair Counter-clockwise 'O
        Midair Zorro 'Z' 3
        Shake 4
        Underhand Bowling 2
TEST FOLD 1
        At Rest 4
        Backhand Tennis 1
        Baseball Throw 4
        Bunny Hops 0
        Forehand Tennis 2
        Midair 'S' 4
        Midair Clockwise 'O' 1
        Midair Counter-clockwise 'O
        Midair Zorro 'Z' 0
                ...
```

But when we are running feature selection and/or parameter tuning experiments, we need to use a slightly different approach or we **risk overfitting our data**.

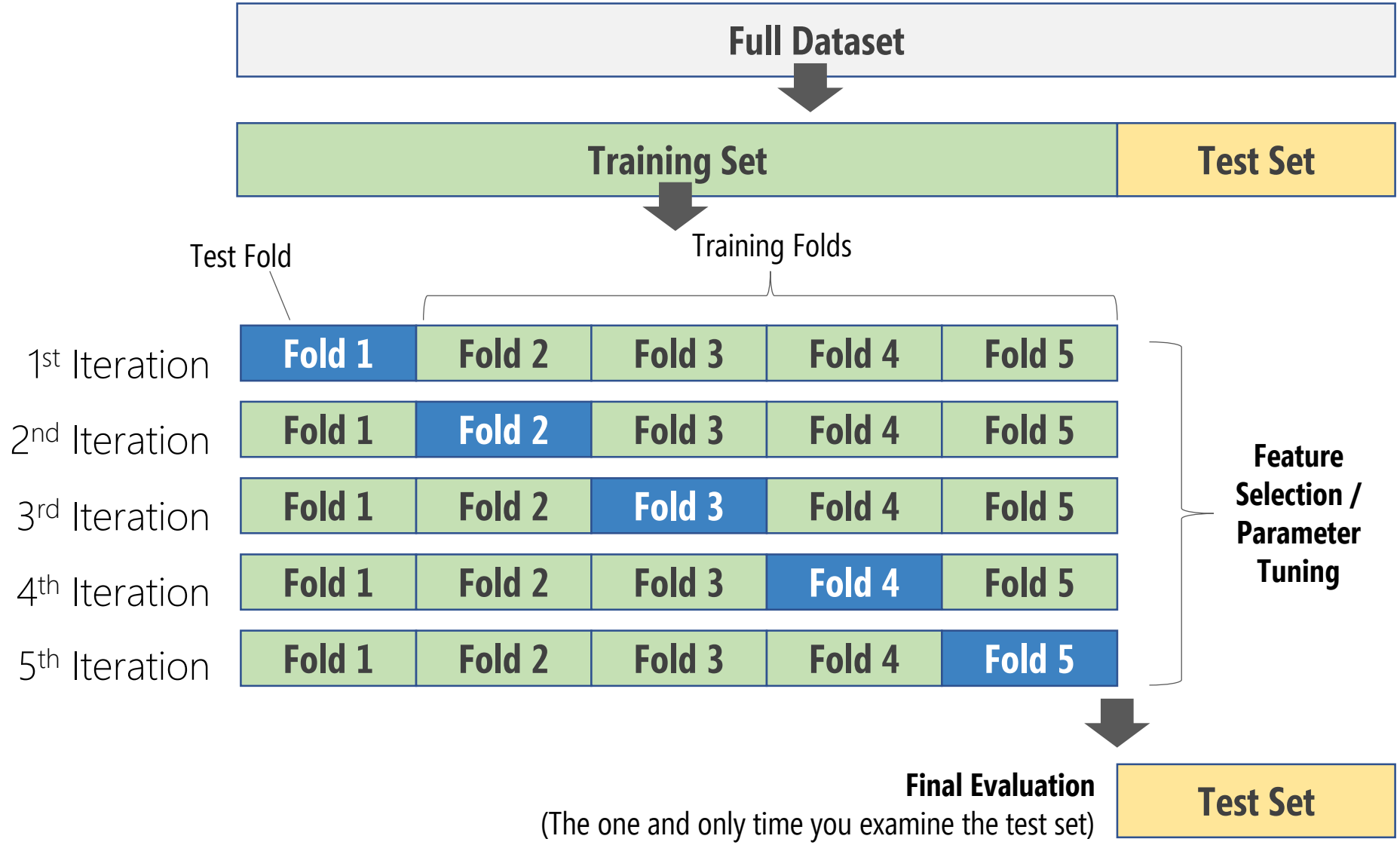
MODEL EVALUATION

TRADITIONAL K-FOLD CROSS VALIDATION APPROACH



MODEL EVALUATION

WITH FEATURE SELECTION/PARAMETER TUNING



Based on: https://scikit-learn.org/stable/modules/cross_validation.html