SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

Engineer thesis

Methods of classification of ski turns based on data from inertial sensors

Author: Monika Tota

Supervisor: dr inż. Agnieszka Szczęsna

Gliwice, January 2020

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 6 stycznia 2020

. . . .
(podpis)

. . . .
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Oświadczenie promotora

Oświadczam, że praca „Methods of classification of ski turns based on data from inertial sensors” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia 6 stycznia 2020

. . . .

(podpis promotora)

Contents

Chapter 1

Introduction

Nowadays, in every sports disciplines, details determine success or failure. To be on the top athletes must be in shape and work-out all the time. During trainings, modern technologies are often used to e.g. monitor progress, measure necessary body parameters and help to achieve goals. As an example it is worth to mention various mobile applications that accompany the athlete during training and record the travelled route, burned calories, body hydration.

When it comes to skiing, professionals use devices equipped with sensors that track the exact motion of a skier. Based on these data, it is possible to i.a. analyze from basic to advanced skiing techniques, manouvres like parallel turns and carving, various parameters such as parameters related with the risk of injury, or even create applications that visualize the movement of a skier. There are many reasons why to use such devices like to improve performance or just to have more fun from the activity.

Objective of the thesis

This project is devoted to recognition of ski turns. The aim is to create a program that based on data from inertial sensors, which are mounted to each ski, detects how many left and right turns occured while the skier was skiing. For this purpose there are a few methods of ski turns classification used, each of them is described and the results of using them are compared.

Scope of the thesis

- Sensors data processing
  1. Reading data from files
  2. Noise filtering
  3. Labelling samples
  4. Removing irrelevant samples
  5. Merging data into one dataset (training set)
- Creating classification model
  1. Testing different methods
  2. Choosing the best model
  3. Additional filtration
  4. Counting recognised ski turns
- Desktop aplication
  1. Designing appearance
  2. Creating windows application
  3. Preparing script which processes the data and predict ski turns using the best classification model

Description of chapters

Chapter Problem analysis constains descriptions of inertial sensors and the method of using signals from them to determine ski turns. It also intoduces to the machine learning field and explains briefly three machine learning algorithms. Next two chapters refer to the desktop application and describe requirements, used tools, and external specification. In Internal specification the whole process of data preparing is presented, as well as creating the machine learning model. This chapter also contains details of implementation of the desktop application. The final program is tested in Verification and validation.

Chapter 2

Problem analysis

2.1 Inertial sensors

The term inertial sensor refers to the combination of a three-axis gyroscope and a three-axis accelerometer. These sensors and sometimes also magnetometers are used combined in inertial mesaurement unit (IMU), which is an integrated sensor package that measures specific force, angular rate, and sometimes magnetic field, with respect to an inertial reference frame. Inertial sensors are used to determine motion and rotation of an object and to track the position and orientation. Gyroscopes and accelerometers are nowadays commonly used in many devices e.g in navigation systems on vehicles such as ships, aircraft, in modern smartphones to auto-rotate the screen when the phone is rotated, in virtual reality (VR) headsets to allows to translate movement into a virtual environment, in medical applications to analyze human limbs motion to improve patient's locomotion, in automotive vehicle stability control applications to compensate for driver error and to enhance vehicle handling [24].

For the past several years wearable inertial sensors have been becoming increasingly popular for body motion measurement since they can can be embedded in the body e.g chest, arms, legs, for monitoring the motion associated with human activities [25].

In these days, many inertial sensors are based on microelectromechanical system (MEMS) technology. Microsystems are cheaper to produce and thanks to the small sizes, they consume less power and are characterized by low weight. The integration of individual system components is also very important in one integrated circuit. This significantly reduces the number of external cables and connections that most affect the unreliability of electronic devices. What is more, the accuracy of MEMs has significantly increased over the years [22].

2.1.1 Gyroscope

A gyroscope is a device used for indicating the rate of rotation around a particular axis by measuring angular velocity. The gyroscope measurements are expressed in degrees per second ($°/s$) or in radians per second (rad/s) in International System of Units (SI) [2, 9, 10].

Conceptually gyroscope is a spinning wheel or disk on an axle. Once it is spinning fast, it creates a very large angular momentum, which is a vector and, therefore, has direction and magnitude. According to the law of conservation of angular momentum, when no external torque acts on an object, no change of angular momentum will occur, thus the direction of the angular momentum vector will not change unless a net torque is applied to the system [4, 6].

$$\vec{L} = I\vec{\omega} = const \tag{2.1}$$

where:
$\vec{L}$ - angular momentum
I - moment of inertia
$\vec{\omega}$ - angular velocity

In inertial navigation systems mostly three-axis gyroscopes are used since they can measure rotation of an object around three axes: x (roll), y (pitch), z (yaw) as shown in Fig. 2.1.

2.1.2 Acceletometer

An accelerometer is a device used for measuring linear acceleration along a particular axis. The accelerometer measurements are expressed in meters per unit time squared (m/s^2).

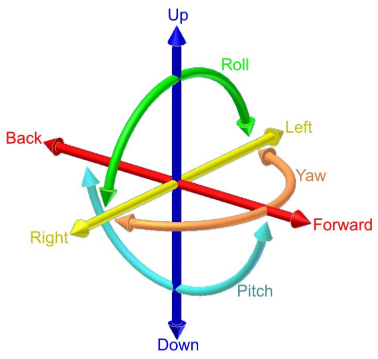


Figure 2.1: Six degrees of freedom of inertial sensor.
Source: `https://en.wikipedia.org/wiki/Six_degrees_of_freedom`

The mechanism of acceletometer operates on Newton's second law of motion, which states that if object is acted upon by an unbalanced force, then the object moves with acceleration that is proportional to the net force F, and is inversely proportional to the mass m (see equation 2.2). The MEMS accelerometers measure acceleration indirectly because they measure a force applied to one of the accelerometer's axes, e.g by detecting the displacement of the mass relative to fixed electrodes [1].

$$\vec{a} = \frac{\vec{F}}{m} \tag{2.2}$$

where:
$\vec{a}$ - acceleration
$\vec{F}$ - net force
m - object's mass

In inertial navigation systems mostly three-axis accelerometers are used since they can measure acceleration of an object along three axes: x (forward-back), y (right-left), z (up-down) as shown in Fig. 2.1.

2.2 Ski-turns detection

To recognize ski-turns, angular velocity around z-axis can be measured, which corresponds to twisting of a skier around a vertical axis, and then right-hand rule shall be used to establish direction of angular velocity. It is important to note that the sensors must be mounted to skis in a proper way, thus, in the correct coordinates system, the z-axis corresponds to vertical axis which is denoted as blue axis(yaw) shown in Fig. 2.1, the y-axis(pitch) is the yellow axis and the x-axis(roll) is the red axis. Examplary signals from gyroscope are presented in Fig. 2.2.

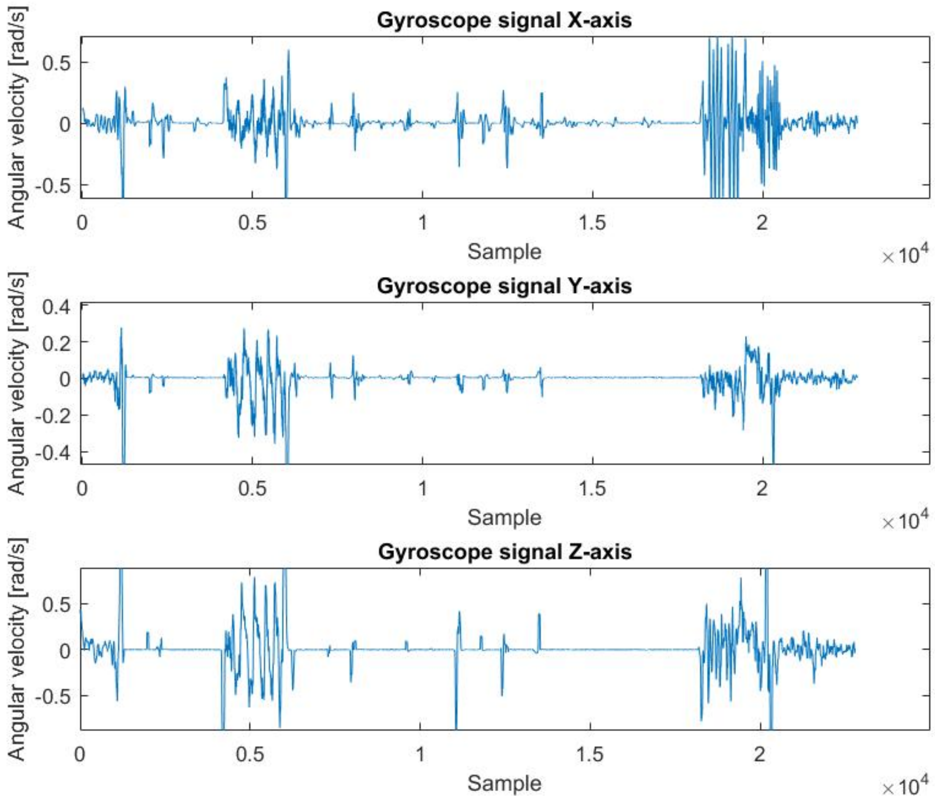
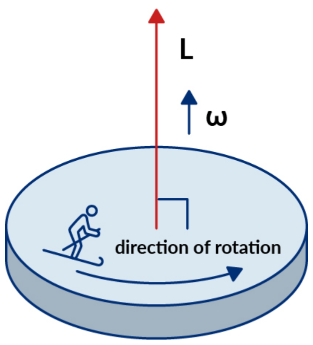
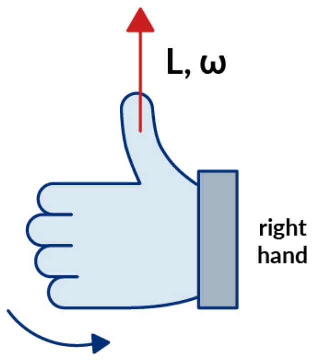


Figure 2.2: Gyroscope signals

Considering that angular velocity $\vec{\omega}$ is related to angular momentum $\vec{L}$ by equation 2.1, the direction of $\vec{\omega}$ is the same as the direction of $\vec{L}$. What is more, the direction of these vector quantities is perpendicular to the plane of circular motion. Using the right-hand rule, the direction is defined to be the direction in which the thumb of a right hand points while curling fingers in the direction of rotation [13, 6]. Hence, if a skier turns left as seen in Fig. 2.3a, the direction of angular velocity and angular momentum is positive as shown in Fig. 2.3b, and if a skier turns right as seen in Fig. 2.4a, the direction of angular velocity and angular momentum is negative as shown in Fig. 2.4b. As a consequence, positive values of angular velocity around z-axis corresponds to left turn and negative values around z-axis corresponds to right turn.

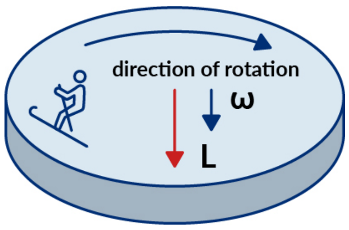


(a) A skier turns left

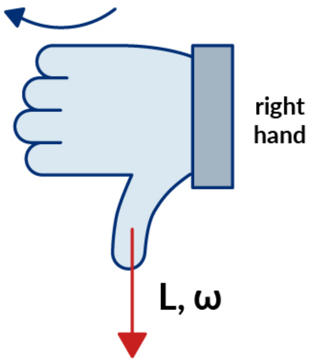


(b) The direction of angular velocity $\vec{\omega}$ and angular momentum $\vec{L}$ are positive

Figure 2.3: Left turn



(a) A skier turns right



(b) The direction of angular velocity $\vec{\omega}$ and angular momentum $\vec{L}$ are negative

Figure 2.4: Right turn

2.3 Machine Learning

Another way of classifying ski turns based on data from inertial sensors is to apply commonly used approach, which is machine learning.

Machine learning is a subset of artificial intelligence (AI). AI is an area of computer science which tries to build an intelligent robot in a way that works like a human being. It means that robot can learn, make mistakes and learn from mistakes to gain an experience and improve its performance.
Artificial intelligence is used in many fields including signal processing, image processing, speech recognition, self-driving cars. Machine learning is basically a set of tools, methods, algorithms and statistical techniques that allow machines to learn from data, and then to automate many useful tasks. This category includes deep learning that can solve problems by using neural networks [3, 23].

2.3.1 Core concepts and terms

Supervised Learning

Machine learning can be classified as *supervised learning* and *unsupervised learning*. In supervised learning, a data scientist or data analyst provides the algorithm with pairs of inputs and desired outputs, and the algorithm tries to recognize the pattern of how to produce the desired output based on given inputs. Those pairs of inputs and desired outputs are called *training data* because this set is used in the process of training the model to make predictions of the outputs. The training can be performed by many algorithms i.a. decision trees, k-nearest neighbors, support vector machines, which are discussed in more detail later in this chapter.

The goal of supervised learning is to build a model on the training data and then to use this model to make accurate predictions on unseen data, that is described in the same way as the training data. If the model predicts accurately on unseen example, it is said that *generalizes* well from the training set to the test set. The model should generalize as accurately as possible to avoid $overfitting$. It occurs when a model fits too closely to the particularities of the training set so that the model knows how to make predictions on the training set but is not able to generalize to new set [16].

Classification

The supervised machine learning can be applied to problems of *classification* (assigning observations to classes it means predefined list of possibilities) and *regression* (predicting quantitative values) [17]. Classification can be divided into *binary classification* in case of exactly two classes, and *multiclass classification*, when distinguishing between more than two classes [16].

Training data

In supervised classification, the training data must be labelled it means each observation must contain a label to which class is assigned to. In case of the training set in a form of a table, rows are known as samples or observations, and columns that are used to learn from, are described as features, attributes, inputs or predictor variables. One column that needs to be predicted is known e.g as response, output or correct answer [18].

Since format of the data has to be in proper manner i.e. feasible for the analysis, raw data first must be converted into a clean data set. This technique is known as *data preprocessing*. It includes handling inaccurate data, noisy data and inconsistent data that may bring misleading results [5].

Feature selection

Before builidng a machine learnining model it is important to understand the data and to see if the problem is solvable without machine learning, or if the data contain the desired information [16]. Some attributes might be more useful than others, and some of them might be irrelevant. Hence, the choice of features in the training set is a fundamental step and a highly problem-dependent task [15].

Hold-out

In order to assess the classifier's performance, the labeled data can be splitted into two parts: large *training set* and smaller *testing set*. The first part is used to train the model, and then the model is used to predict the responses of the second part. The accuracy is calculated by comparing classifier's responses and the original labels of the test set. If the classifier generalizes poorly, it may be a good idea to add or remove some features or select different type of a classifier. If the model does well, then it might be a potentially valuable model that can be deploy on data in the real world [20].

Cross-validation

Another way of evaluating generalization performance is $cross - validation$. This method is more stable and thorough than simple train and test split, since in cross-validation the data is split into a number of chunks and thus multiple models are trained. For instance, when data is partitioned into five parts called folds, folds 1-4 are used to train the model, and fold 5 is used to evaluate accuracy of particular split. This process is repeated five times for every fold and then the accuracy is computed as an average [16].

2.3.2 Training algorithms

k-Nearest Neighbors

The k-nearest neighbors (k-NN) algorithm is considered as one of the simplest machine learning algorithm. In a training phase, this method just stores all training data points. To make a prediction on a new dataset, the algorithm searchs for the closest data points in the training set - so called *nearest neighbors*. For this reason, the training process is typically very fast. However, all training data including features vectors and labels must be stored within a model, thus it can result in large model sizes [17]. Since finding the nearest data points in training dataset can be time consuming, the algorithm's classification speed is rather slow.

In the simplest form of the k-NN algorithm, only one nearest neighbor is considered, which is the data point in the training set that has the smallest distance to the new data point whose label shall be predicted. For instance, Fig.2.5 depicts data points from two classes: blue circles and red squares, where two new data points are added, shown as stars. The prediction is made based on the data points' label which are the closest. In the real world, instead of one nearest neighbor, arbitrary number k of neighbors is considered. In this case, the algorithm searchs for k nearest data points and chooses the most common class as the prediction.

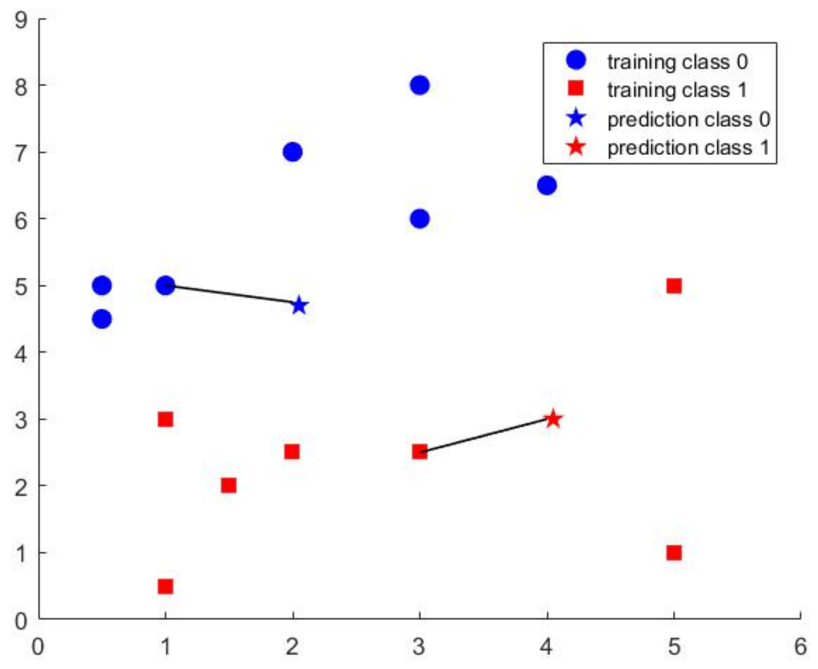


Figure 2.5: One nearest-neighbors model - prediction

Decision Trees

In machine learning, decision trees models are very useful since they are quite easy to explaing and visualize. Basically, they learn a structure of if/else questions that lead to a solution. Their big advantage is that they are pretty fast at computation, both while training and predicting rensponses on new data sets.

Learning a decision tree model means to choose such if/else questions, so called *tests*, that lead to a decision most quickly. Those questions comes with features in a form e.g. "Is feature i larger than value a?". To build a model, the method searches over all possible tests and chooses those that are the most informative. In this process, the hierarchical tree of decisions is created, with each node containing a question about one feature. This usually can lead to very deep and complex model, which generalizes poorly on a new dataset. In this case, it is crusial to restrict the depth of a decision tree or to remove nodes that bring little information, to avoid overfitting [16].

Support Vector Machines

Support Vector Machines (SVMs) are another algorithm used for classification. The main idea behind SVMs is to find an optimal hyperplane which separates data into different classes. The optimal hyperplane is defined as the linear decision boundary with widest margin i.e. distance between the classes. To construct such optimal hyperplane only a subset of the training data is taking into account, the so called *support vectors*, which determine the margin, as shown in Fig. 2.6 [19].

In real world application, the data is usually not lineraly separable, in this case the notion of $soft\ margin$ is introduced, that allow to learn on the training set with some error. Hence, the support vectors are no longer fired to lie on the margin [21]. In this solution, penalty parameter C has to be determined, that controls trade-off between margin and error. Increasing value C minimizes the number of errors on the data and it results in a stricter separation between classes and also worse generalization. Reducing value C allow for bigger mis-classification and it results in a wider margin but with points that lie within it [12]. What is more, large C can slow down the training process.

If a linear boundary is inappropriate, the training algorithm maps the input data into a higher dimension feature space. In this feature space an optimal separating hyperlane can be constructed with high generalization ability. The non-linear mapping is performed by kernel function e.g polynomial, which is a popular method; gaussian radial basis function. If polynomial kernel function is used, then the *degree* of the polynomial must be determined. For non-linear hyperplanes the *gamma* parameter is introduced, which defines how much influence any one training sample has on the resulting model. For smaller values of *gamma* the decision boundary is "smoother" and does not exactly capture the shape of the dataset. For larger values of *gamma* the model tries to perfectly fit the data and it might result in overfitting [17, 21].

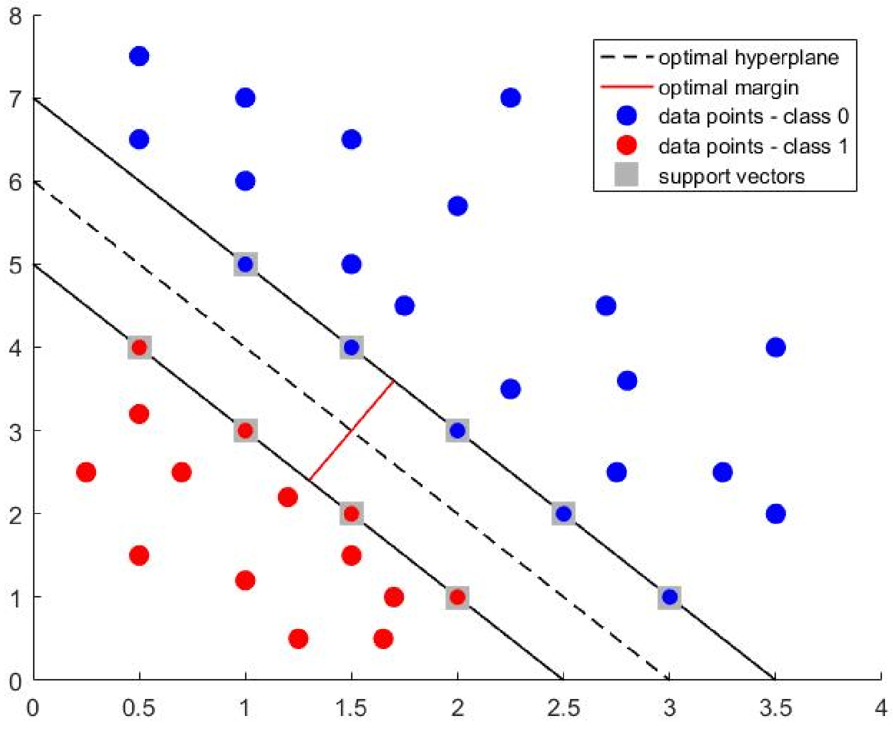


Figure 2.6: Example with two classes in two dimensional space. The data is separable and the hyperplane in this case is a line. The support vectors are marked with grey squares and define the margin of maximum separation of these two classes.

2.4 Review of systems using inertial sensors available on the market

Snowcookie

Snowcookie is a multi-sensor system designed to measure technique and style of skiing, which consists of two ski sensors, body sensor and the application. Each sensor contains IMU which allows to continuously monitor skis and body motion. The application provides advanced analysis based on sensors data such as number of turns, their max edge angles, technique of turning, turn types, the quality of the turns, skiers stance over the whole run and body mass distribution. Snowcookie monitors across six critical areas of performance and technique: stance, turn iQ, style, speed, stamina and engagement. These informations are helpful to point out ares that can be improved to become better at skiing [11].

Hawx Connected ski boot

Atomic's Hawx Connected ski boot is created to report skiing performance metrics via smartphone application. The boot is equipped with a nine-axis IMU which is used to measure pressure at key points and to track orientation of the boot to detect edging angles and turns. It can be paired with a smarphone via Bluetooth 4.0 (and later) and using *Atomic CTD* application the user receives a full performance analysis such as balance, edging, number of turns and G-force. Based on this parameters skiers can impove their balance, edging technique and pressure control. The application allows also to set goals and to compare obtained parameters against professional athletes' performance [7].

Zepp Tennis 2

Zepp Tennis 2 is a swing analyzer which allows to track practise and match statistics to get insights about performance. The key metrics which are tracked include: stroke type, ball speed, ball spin and sweet pot. The Zepp sensor constains dual accelerometers and 3-axis gyroscope and can be easily attached to any tennis racquet to capture data. The informations are wirelessly sent to a smarphone via Bluetooth. Zepp breaks down strokes by swing and spin type and provides separate reports for each and indicates areas of improvement. The Smart Rally Capture technology records every rally and generates clips of each stroke and automatically choose the best videos based on key performace metrics and the user can instantly analyze it [14].

Chapter 3

Requirements and tools

Windows desktop application

The created desktop application has some nonfunctional requirements that must be satisfy in order to run and use the application. The program provides few functionalites which are listed as functional requirements. The chapter constains also description of the behaviour of the application and used programming tools.

Nonfunctional requirements

The desktop application is created for Windows operating system. The program is set up to work on .NET Framework 4.5 and newer versions. To use the program MATLAB 2016a or newer version is required. Previous versions of MATLAB may not contain some necessary functions.

Functional requirements

- selecting .data file with the data from sensors
- displaying filtered gyroscope signals on a plot
- classifying ski-turns and indicating them on a plot
- displaying total number of detected right and left turns

Methodology of design

Figure 3.1 depicts simplified application architecture. First, desktop application is used to select sensors data file that is intended to process. Then the data from the file are sent to the service, which has a reference to MATLAB environment. By means of MATLAB scipts the service receives filtered signals and informations about classified turns. The results are sent to a client and displayed in the desktop application.

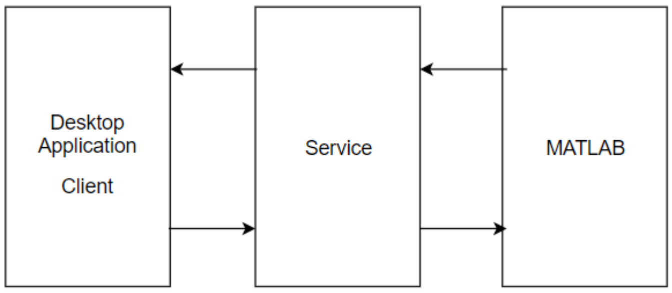


Figure 3.1: Scheme of the project

Description of tools

Both, desktop application and service are implemented in development environment Visual Studio with C# as a programming language. Desktop application is created using Windows Forms, which is a technology for the .NET Framework that allow to create client applications. Service is implemented using Windows Communication Foundation (WCF), which is a framework for building service-oriented applications.

Chapter 4

External specification

Windows desktop application

After launching the application, a user can see two buttons but is only able to to click "Select sensors data" button, as shown in Fig. 4.1. After choosing sensors data file, the file path is visible to the user as shown in Fig.4.2. Next the user is able to click "Detect ski-turns" button, and after usually few seconds (depending on the size of dataset) the results are visible on the screen.

Figure 4.3 depicts output i.e. filtered gyroscope signals from left and right ski, which are used to detect ski-turns, as well as classified ski-turns presented on the plot and total number of left and right turns. Since three signals are presented on one plot, it is possible to uncheck specific signals and analyze only some of them.

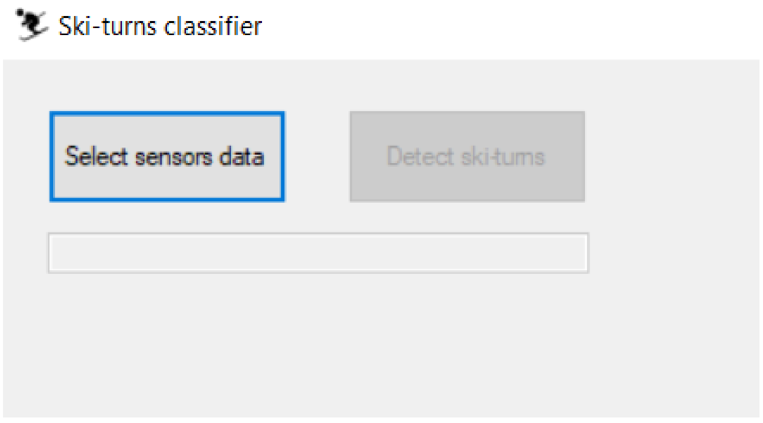


Figure 4.1: Main menu of the application

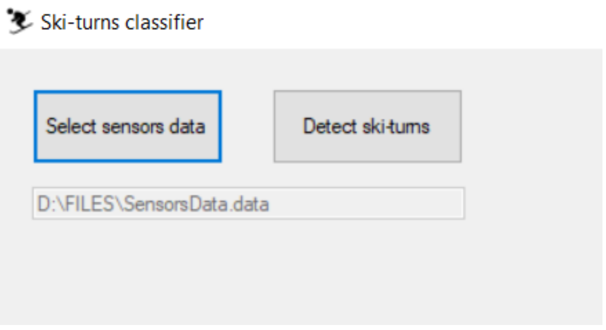
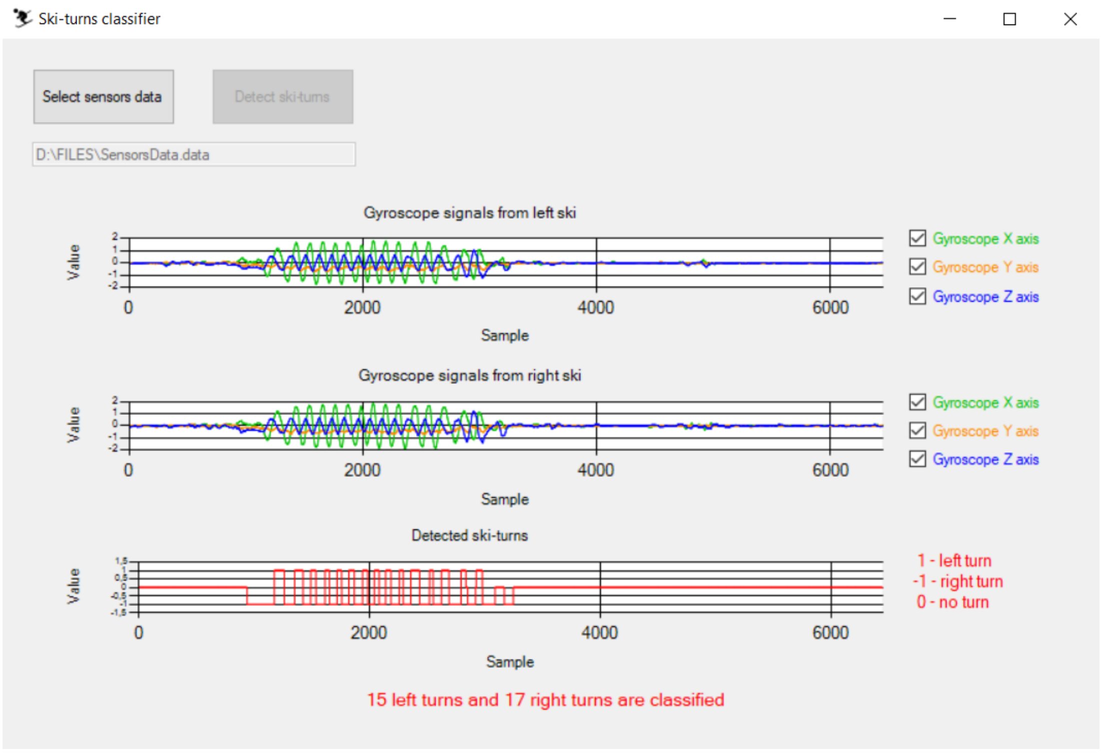


Figure 4.2: Main menu after selecting a file



15 left turns and 17 right turns are classified

Figure 4.3: Presented results of classification

Chapter 5

Internal specification

Since the aim of the project is to classify ski-turns, the main tasks which are described in this chapter are to create training set and then to create a classification model. Additionally, the desktop application is created in order to be able to easly select data and display classification results, instead of doing this in MATLAB environment.

5.1 Preparing a training set

In the project, five different datasets are used. Three of them are used to prepare a training set and train models. While training the training accuracy is calculated using cross-validation. Two remaining datasets are used to verify performance of the models and to choose the best one by evaluating calculated testing accuracy. The best classification model will be used in the final application to recognise ski-turns.

Each dataset represents one measurement while a skier was going down a slope. Sensors were mounted to each ski and to a chest. Collection of data contains i.a. signals from three-dimensional gyroscope, three-dimensional accelerometer separately from left ski, right ski and chest, and also timestamp, temperature, pressure. In this project only data from gyroscope and accelerometer from left and right ski are taking into account because other quantities do not bring any crucial informations in terms of classifying ski-turns.

The whole process of preparing a training set as well as a classification model is done in MATLAB environment.

Feature representation

Raw data from the gyroscope and accelerometer, which are used to measure angular velocity and linear acceleration, respectively, are in binary format. To obtain numerical format the parsing function is used, which reads the data and writes them in proper order to an array.

Next the dataset is build in terms of feature variables (see Fig. 5.1). Totally there are twelve predictor variables, since there are two skis and three-axis gyroscope and three-axis accelerometer on each ski.

```
%binParser function is used to obtain numerical form
data=binParser('SensorsData.data');

%Left ski
%Accelerometer
Left_Ski_Accelerometer_X_axis=data(:,12);
Left_Ski_Accelerometer_Y_axis=data(:,13);
Left_Ski_Accelerometer_Z_axis=data(:,14);
%Gyroscope
Left_Ski_Gyroscope_X_axis=data(:,15);
Left_Ski_Gyroscope_Y_axis=data(:,16);
Left_Ski_Gyroscope_Z_axis=data(:,17);

%Right ski
%Accelerometer
Right_Ski_Accelerometer_X_axis=data(:,21);
Right_Ski_Accelerometer_Y_axis=data(:,22);
Right_Ski_Accelerometer_Z_axis=data(:,23);
%Gyroscope
Right_Ski_Gyroscope_X_axis=data(:,24);
Right_Ski_Gyroscope_Y_axis=data(:,25);
Right_Ski_Gyroscope_Z_axis=data(:,26);
```

Figure 5.1: The feature variables

Data from all sensors was displayed on plot and analyzed. It turned out that sensor signals are very disturbed and impossible to analyze. As an example, Fig. 5.2 depict three signals from gyroscope from left ski, which represent one downhill skiing. To allow further analysis of sensors data, the next step is filtration.

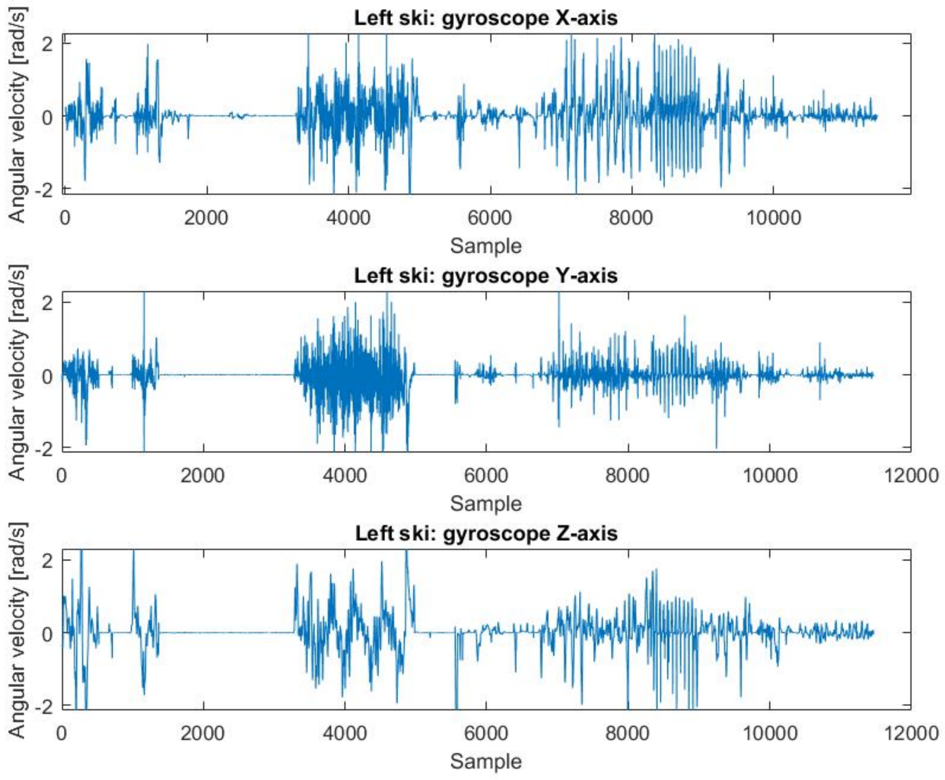


Figure 5.2: Data from gyroscope in numerical form presented on the plot

Signal filtering

To perform filtering, moving average filter is used, which is a simple Low Pass filter used for reducing random noise, which takes number of inputs and calculates the average to produce a single output. For this purpose, `movmean` Matlab function [8] is applied to all attributes. It was found that for 50-point mean values the signals are properly filtered, so that they still contain important informations but are more convenient to analyze and to interpret. Fig. 5.3 depicts results of applying moving average filter to gyroscope signals from left ski. The signals become much smoother and values can be read from the plots.

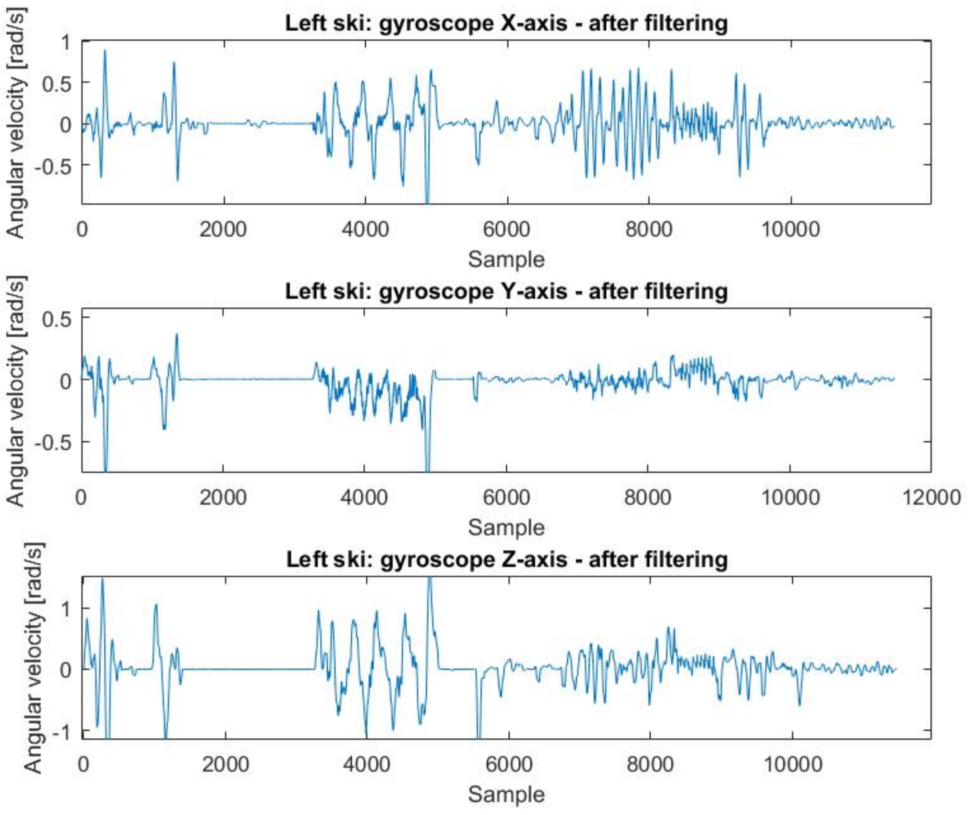
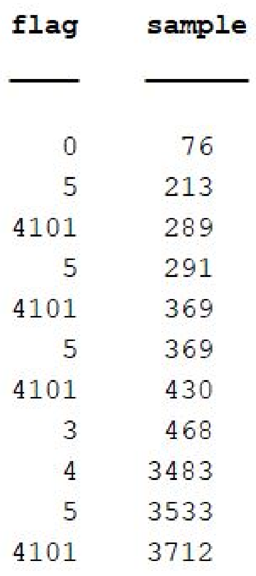


Figure 5.3: Filtered signals from gyroscope

Data labelling

Next step is to assign a label to each sample. In this project there are three possible labels: *right turn*, *left turn* and also *no turn*. *No turn* class is introduced since the datasets contains records that do not refer to ski-turns itself e.g while a skier might have been preparing, attaching the sensors or going straight down a slope.

During measurements, CSV file was created for each measurement, examplary file is presented in Fig. 5.4. The first column contains flags and the second columns constains samples numbers. Value 5 of flag indicates start of a ski-turn and value 4101 in the next line stands for end of this ski-turn. Another flags are irrelevant in this project. The CSV files are used to define which samples are related with turns and which are not. For instance, samples from 0 to 212 from Fig. 5.4 are recognized as *no turn* class, while samples from 213 to 289 are recognized as either *right turn* or *left turn* classes.



flag	sample
0	76
5	213
4101	289
5	291
4101	369
5	369
4101	430
3	468
4	3483
5	3533
4101	3712

Figure 5.4: Part of CSV file

The question then arizes: how to distinguish *right turn* and *left turn* classses? To answer this, method described in section 2.2 should be recalled. According to right-hand rule applied to a rotating object, in this case the rotaing ski when a skier left turn, the direction of angular velocity measured on the skis is positive, and in case of turning right, the direction of angular velocity of the rotating ski is negative. It is important to note, that to tell turns apart, only angular velocity around Z-axis (gyroscope around Z-axis) is crusial, since it corresponds to twisting of a skier and his skis around a vertical axis. Therefore, positive values from the gyroscope around Z-axis can be identified as *left turn* and negative values can be identified as *right turn*.

Fig. 5.5 depicts gyroscope signals from left ski with marked start-turn and end-turn points based on CSV file as well as with green line containing proper labels which indicate whether a sample is classified as *right turn*, *left turn* or *no turn*. Value 1 of represents *left turn* class and value -1 refers to *right turn* class. Samples between two turns, if occurs, and noises are classified as *no turn* class.

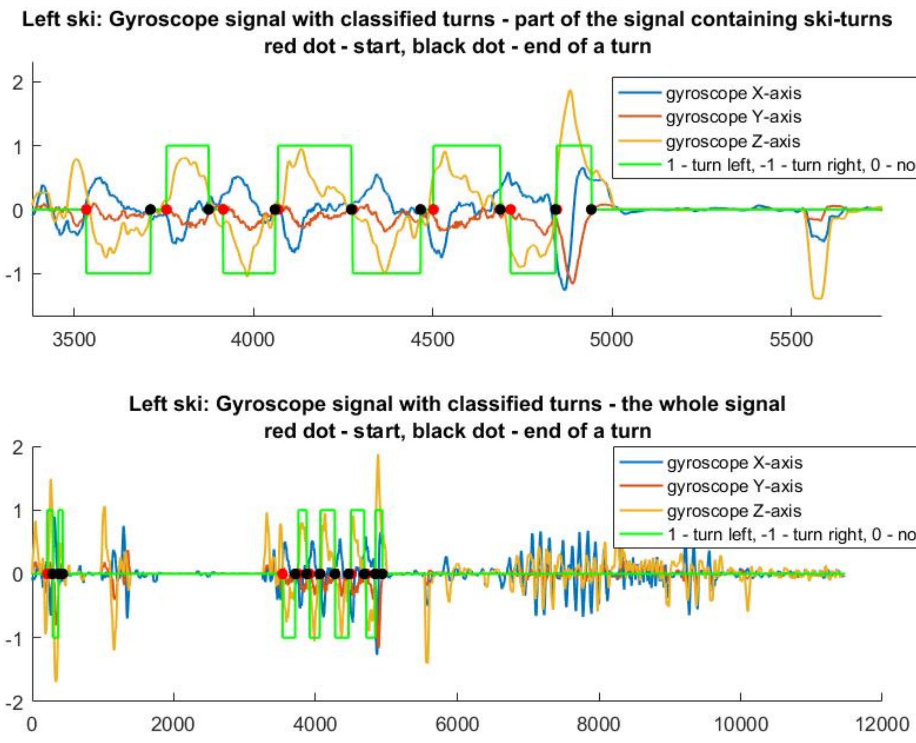


Figure 5.5: Gyroscope signals with classified ski-turns

Data cleaning

In the analyzed signals from gyroscope and accelerometer there are only a few ski turns and most of samples are classified as *no turn*. The *right turn* and *left turn* are the minority classes. The goal is to create a model to classify ski-turns, thus most of the records assigned as *no turn* should be removed from the training set to ensure balanced dataset and to be able calculate reliable accuracy.

Fig.5.6 shows results of removing most of samples referring to *no turn* class in signals from left ski from the gyroscope and the accelerometer. It can be seen that the size of a dataset has been reduced from about 12000 to only 2000 records.

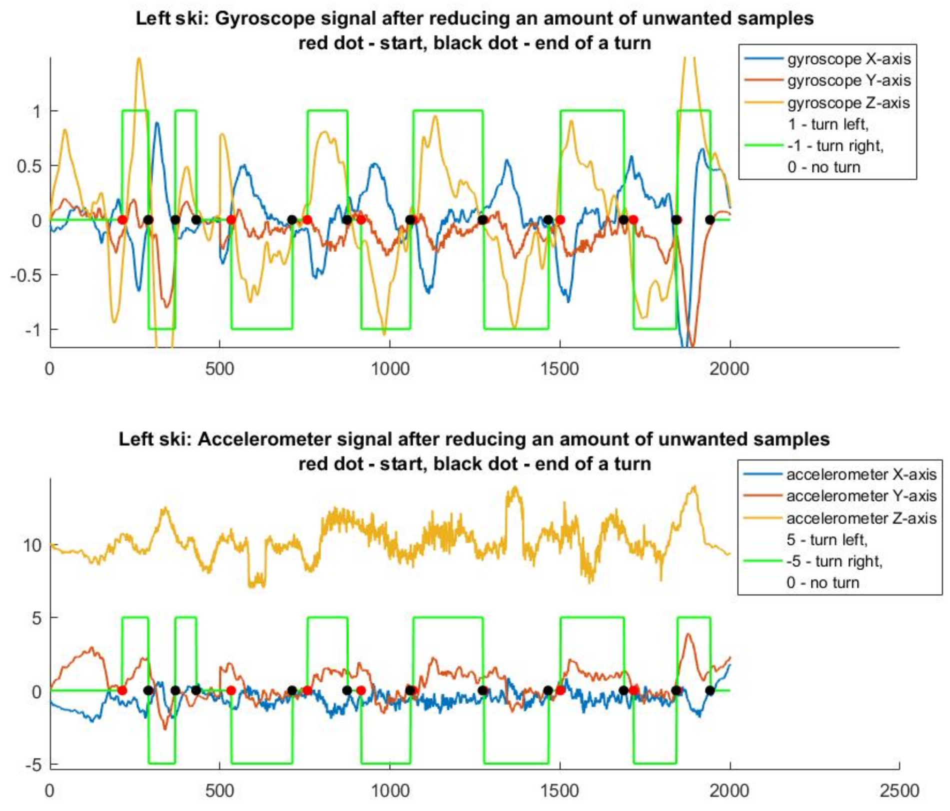


Figure 5.6: Signals with reduced amount of unwanted records

Merging datasets into one

All the previously described steps are repeated also for two remaining downhill skiing, which are also included in the training set. Next all three datasets are merged into the one training set, as shown in Fig.5.7, which will be used to train classification models. The training set consists of twelve features as a collection of data from two skies from three-axis gyroscope and three-axis accelerometer. At the beginning all attributes will be used to build classification models, but then it will be evaluated which features are irrelevant and should be removed to impove the perfomance of the model. All the samples are classified, it means that have a response variable with a label either *right turn*, *left turn* or *no turn*.

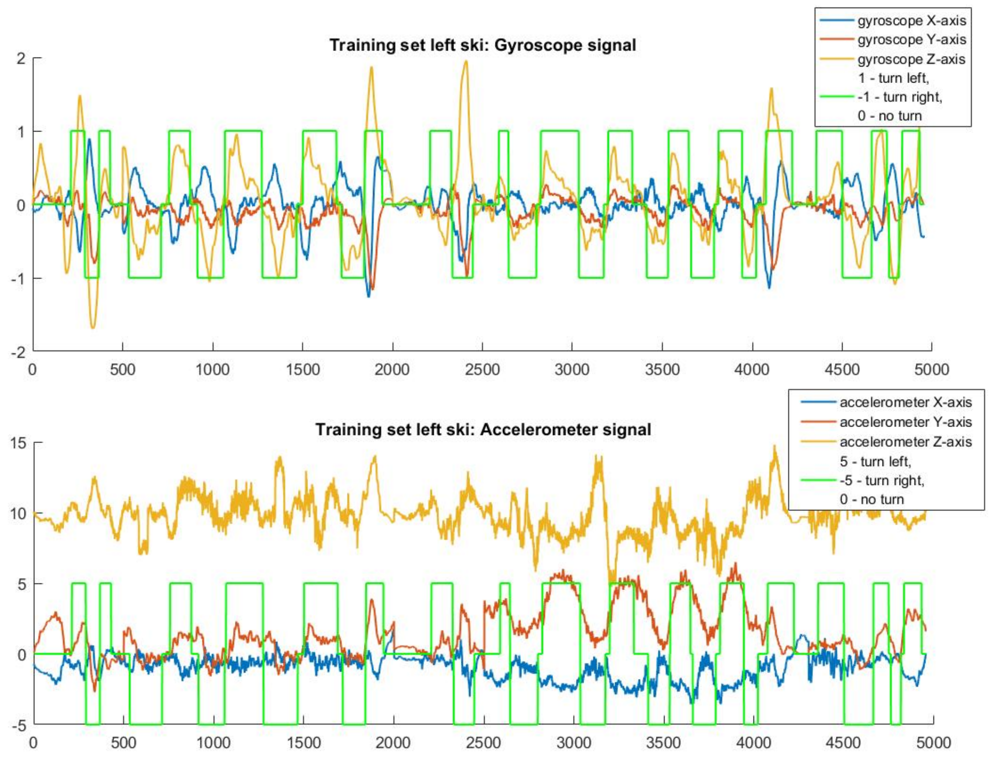


Figure 5.7: Training set: left ski

5.2 Creating machine learning model

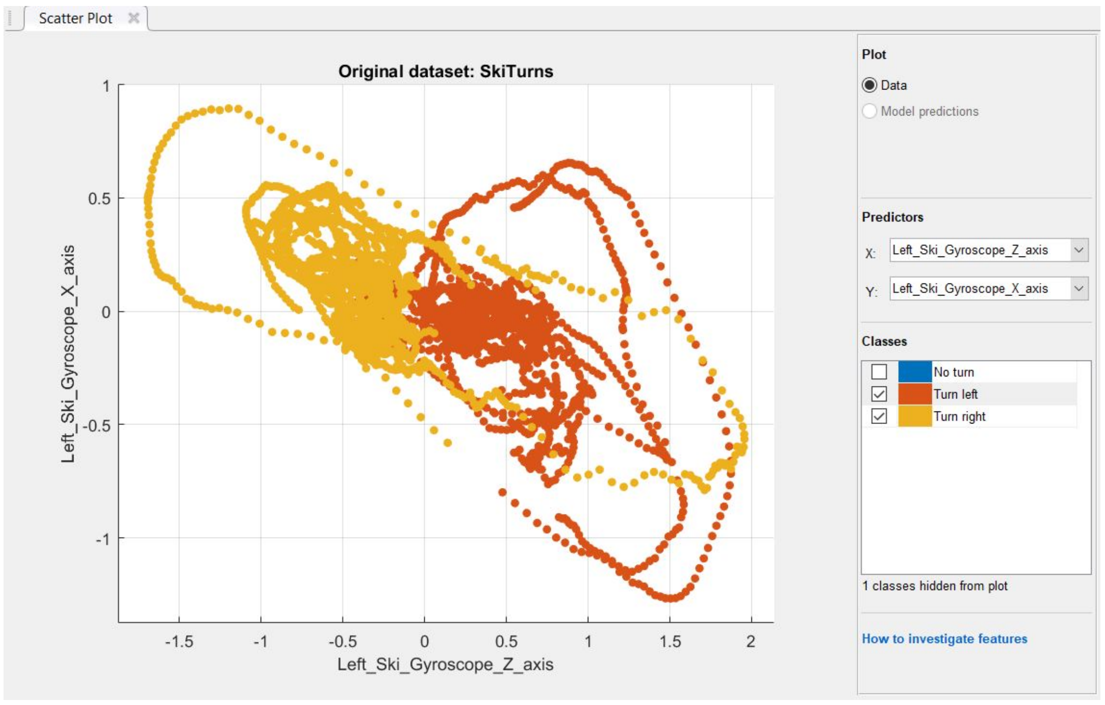
Having the training set, which is a dataset with labelled each record to one of three possible classes: *right turn*, *left turn* or *no turn*, next step is to train a classification model to make predictions.

Classification Learner app

To train a model to classify data using supervised machine learning, the Classification Learner app (Statistics and Machine Learning Toolbox^{TM}) in MATLAB environment is used. It supports many different classifiers and enables to explore the data, select features, specify validation techniques and visualize results using i.a. Scatter Plot and Confusion Matrix. This app also enables to generate MATLAB code to work with scripts, therefore this will be convenient to test a model on a new dataset, to modify the code to train a different model and to automate training and testing process to compare the generalisation of various models.

Scatter Plot

In the Classification Learner app the training data can be investigated using the Scatter Plot, which shows values obtained for two selected predictors - one plotted along the horizontal axis and second one around the vertical axis. The Scatter Plot is used to present the correlation of these two predictors.


Scatter Plot
Original dataset: SkiTurns
Left_Ski_Gyroscope_X_axis
Left_Ski_Gyroscope_Z_axis
1
0.5
0
-0.5
-1
-1.5
-1
-0.5
0
0.5
1
1.5
2
How to investigate features

Plot
● Data
○ Model predictions

Predictors
X: Left_Ski_Gyroscope_Z_axis
Y: Left_Ski_Gyroscope_X_axis

Classes

☐	🟦	No turn
☑	🟥	Turn left
☑	🟨	Turn right

1 classes hidden from plot

Figure 5.8: Scatter Plot: gyroscope Z-axis and gyroscope X-axis

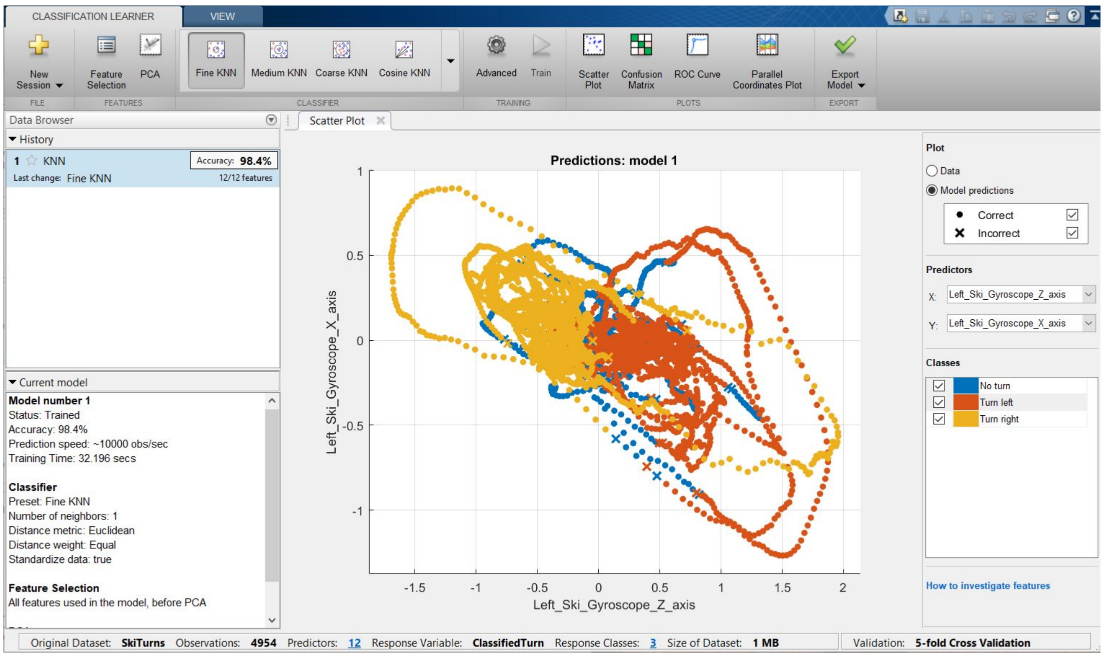
While creating a classification model, different pairs of predictors are analyzed using Scatter Plot and some correlations are observed. For instance, Fig. 5.8 depicts Scatter Plot that shows the values obtained for gyroscope Z-axis and gyroscope X-axis for *right turn* and *left turn* classes. It can be noticed, that both datasets are almost linearly separable with some error. Therefore, these features seem to be useful while predicting the responses. Similarly, for gyroscope Z-axis and gyroscope Y-axis the datasets are also almost linearly separable with some error.

In case of *no turn* class versus *right turn* and *left turn* classes, it turns out that the datasets are much closer to each other and overlap. Finally, from the Scatter Plots for gyroscope, it can be concluded that data from all three axis of the gyroscope are useful for predicting *right turn* and *left turn* classes. However, it might be quite diffucult for the classifier to distinguish *no turn* class versus *right turn* and *left turn* classes since they overlap.

The data from accelerometer are also analysed to notice some dependencies between the features. Generally, the datasets overlap and are not linearly separable. Therefore accelerometer data might decrease the accuracy of the classifier.

Training process in Classification Learner app

To train a machine learning model, first, a model type must be chosen on the Classification Learner tab, and then Train button clicked. Fig. 5.9 shows results of training the model using k-nearest neighbors algorithm with all featured used.



1 KNN Accuracy: 98.4%
Last change: Fine KNN 12/12 features

Model number 1
Status: Trained
Accuracy: 98.4%
Prediction speed: ~10000 obs/sec
Training Time: 32.196 secs

Classifier
Preset: Fine KNN
Number of neighbors: 1
Distance metric: Euclidean
Distance weight: Equal
Standardize data: true

Feature Selection
All features used in the model, before PCA

X: Left_Ski_Gyroscope_Z_axis
Y: Left_Ski_Gyroscope_X_axis

☑ 🟦	No turn
☑ 🟥	Turn left
☑ 🟨	Turn right

Original Dataset: **SkiTurns** Observations: **4954** Predictors: **12** Response Variable: **ClassifiedTurn** Response Classes: **3** Size of Dataset: **1 MB** Validation: **5-fold Cross Validation**

Figure 5.9: Trained model using Classification Learner app

The app informs the researcher about training accuracy of the model in percent (calculated using cross-validation), prediction speed and training time.

The results are also presented on the Scatter Plot, where an X indicates misclassified points. Under Plot, the reseacher can switch between the Data and Model predictions options for further analysis. Alternatively, to see only the incorrect points, the Correct check box can be cleared while plotting model predictions.

Classification Learner app allows to generate code from the app to recreate the same classifier and use it with new unseen data. The generated code can be modified to set different options and parameters, and what is more, the whole process of training and testing can be automated.

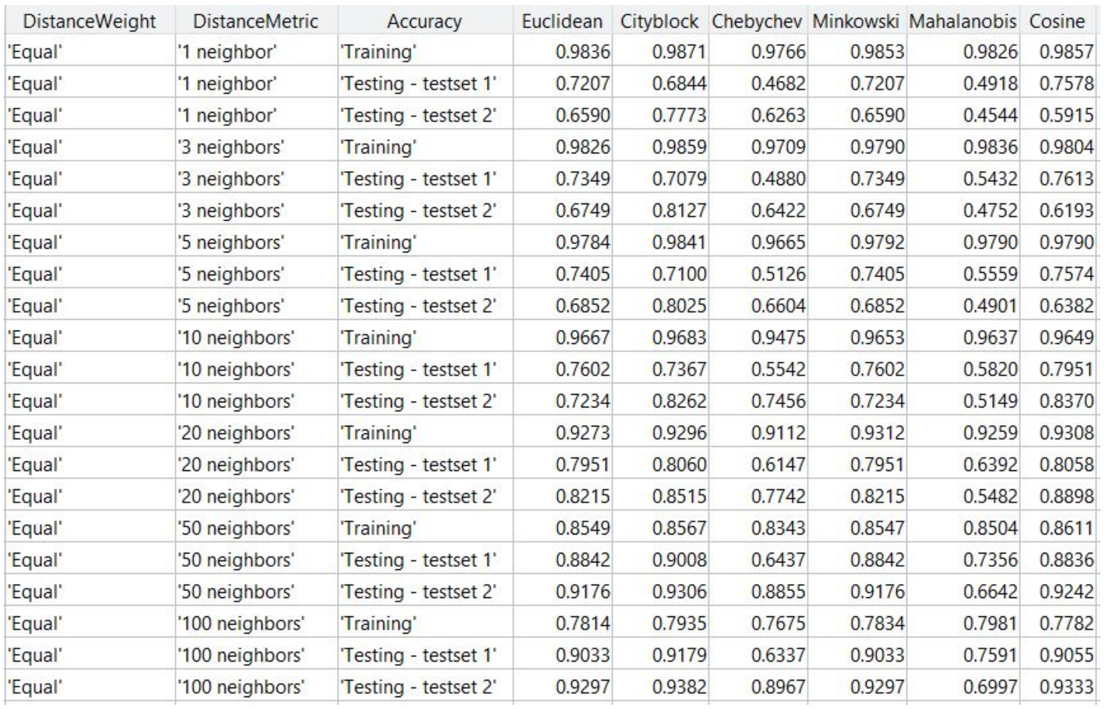
In this project three training algorithms are taking into consideration: k-nearest neighbors, decision trees and support vector machines. Each method is evaluated in terms of feature selection, training and testing accuracy using two remaining measurements, as well as prediction speed and training time. The aim is to create a model which performs well on unseen examples.

5.2.1 K-nearest neighbors method

Listing 1 presents generated function for k-nearest neighbors method, but modified in a way that the researcher can define necessary input parameters i.e. attributes (whether all variables are used i.e. accelerometer and gyroscope or only gyroscope), distance metrics, distance weights and numbers of neighbors. The function as an output returns trained model, which can be used on new example, and training accuracy.

All features used: gyroscope and accelerometer

At the very beginning, all attributes are used to train the models. Fig.5.10 presents table with accuracy of training and testing many models using k-nearest neighbors algorithm with distance weight define as 'equal' and several distance metrics and numbers of neighbors. The training accuracy is calculated while training each model using cross-validation. Each model is tested using two remaing test sets and for each the testing accuracy is calculated by comparing model predictions and desired predictions, by using additional script which creates the table. For both training and testing processes, the accuracy means percent of correctly predicted responses.



DistanceWeight	DistanceMetric	Accuracy	Euclidean	Cityblock	Chebychev	Minkowski	Mahalanobis	Cosine
'Equal'	'1 neighbor'	'Training'	0.9836	0.9871	0.9766	0.9853	0.9826	0.9857
'Equal'	'1 neighbor'	'Testing - testset 1'	0.7207	0.6844	0.4682	0.7207	0.4918	0.7578
'Equal'	'1 neighbor'	'Testing - testset 2'	0.6590	0.7773	0.6263	0.6590	0.4544	0.5915
'Equal'	'3 neighbors'	'Training'	0.9826	0.9859	0.9709	0.9790	0.9836	0.9804
'Equal'	'3 neighbors'	'Testing - testset 1'	0.7349	0.7079	0.4880	0.7349	0.5432	0.7613
'Equal'	'3 neighbors'	'Testing - testset 2'	0.6749	0.8127	0.6422	0.6749	0.4752	0.6193
'Equal'	'5 neighbors'	'Training'	0.9784	0.9841	0.9665	0.9792	0.9790	0.9790
'Equal'	'5 neighbors'	'Testing - testset 1'	0.7405	0.7100	0.5126	0.7405	0.5559	0.7574
'Equal'	'5 neighbors'	'Testing - testset 2'	0.6852	0.8025	0.6604	0.6852	0.4901	0.6382
'Equal'	'10 neighbors'	'Training'	0.9667	0.9683	0.9475	0.9653	0.9637	0.9649
'Equal'	'10 neighbors'	'Testing - testset 1'	0.7602	0.7367	0.5542	0.7602	0.5820	0.7951
'Equal'	'10 neighbors'	'Testing - testset 2'	0.7234	0.8262	0.7456	0.7234	0.5149	0.8370
'Equal'	'20 neighbors'	'Training'	0.9273	0.9296	0.9112	0.9312	0.9259	0.9308
'Equal'	'20 neighbors'	'Testing - testset 1'	0.7951	0.8060	0.6147	0.7951	0.6392	0.8058
'Equal'	'20 neighbors'	'Testing - testset 2'	0.8215	0.8515	0.7742	0.8215	0.5482	0.8898
'Equal'	'50 neighbors'	'Training'	0.8549	0.8567	0.8343	0.8547	0.8504	0.8611
'Equal'	'50 neighbors'	'Testing - testset 1'	0.8842	0.9008	0.6437	0.8842	0.7356	0.8836
'Equal'	'50 neighbors'	'Testing - testset 2'	0.9176	0.9306	0.8855	0.9176	0.6642	0.9242
'Equal'	'100 neighbors'	'Training'	0.7814	0.7935	0.7675	0.7834	0.7981	0.7782
'Equal'	'100 neighbors'	'Testing - testset 1'	0.9033	0.9179	0.6337	0.9033	0.7591	0.9055
'Equal'	'100 neighbors'	'Testing - testset 2'	0.9297	0.9382	0.8967	0.9297	0.6997	0.9333

Figure 5.10: Accuracy: all features used and distance weight define as 'equal'

For 1, 3, 5 and 10 neighbors the training accuracy is very high - above 90 percent but the testing accuracy is low - around 60-80 percent. Those models perform poorly. For 50 neighbors the testing accuracy in some cases obtains 88-93 percent, even though the training accuracy is lower - around 85 percent.

It is due to the fact that training data constains approximately equal number of samples from each of three classes, while the testing sets are the whole signals from sensors and constain mainly samples that refer to *no turn* class. Since it turns out that it is quite easy for the classifiers to predict *no turn* class, it happens that in some cases the testing accuracy is much higher than the training accuracy.

The highest testing accuracy around 92-94 percent obtains the model with distance weight defined as 'equal', 100 neighbors and distance metric defined as 'cityblock'. The model's predictions are visualised on a plot and compared to the correct predictions, as shown in Fig. 5.11. The green line indicates correctly predicted responses and the red line shows misclassifications. It can be observed that in both examples the real ski-turns are generally detected, but there are also some incorrectly predicted ski-turns when they do not really occur. Since there are many machine learning models created in the project, only some of them which get the highest accuracy are visualised on the plots.

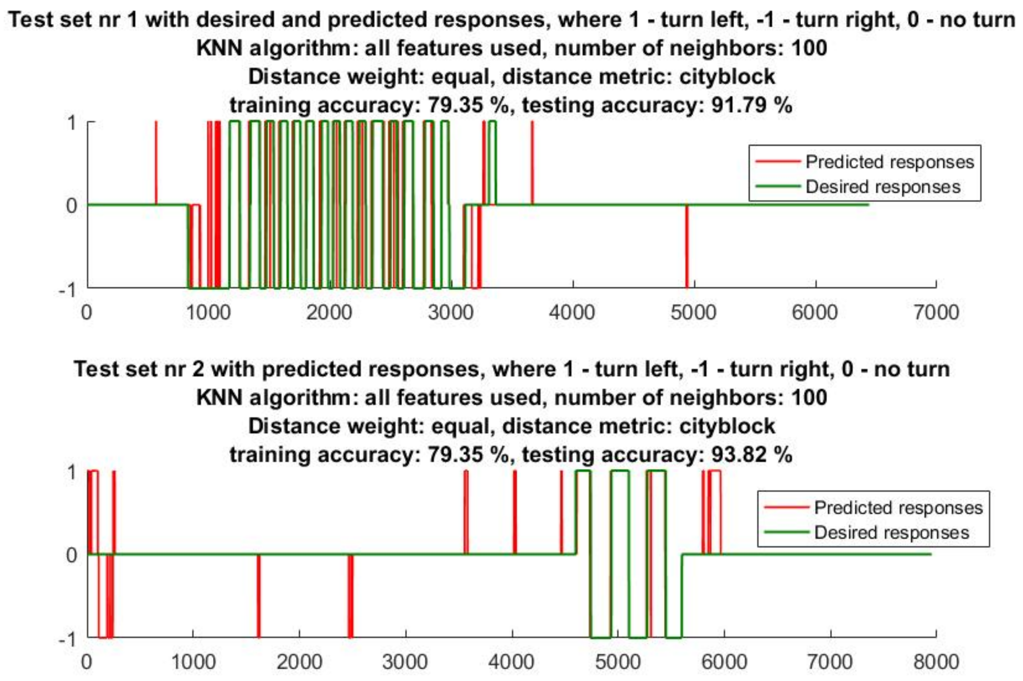
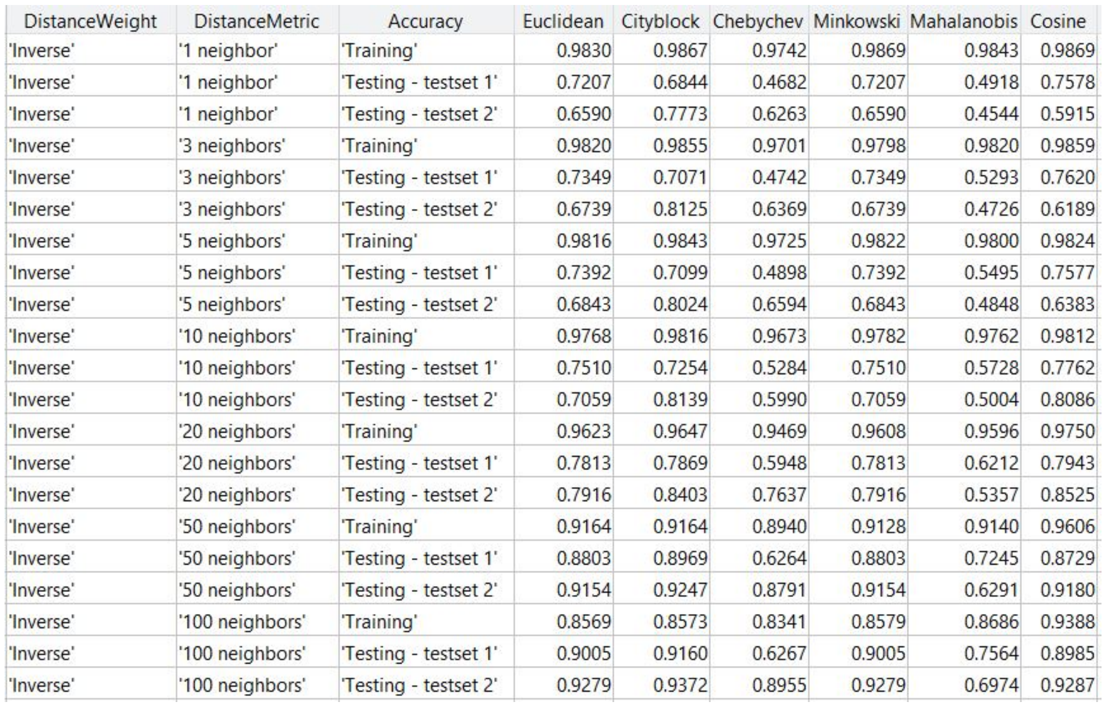


Figure 5.11: Test set nr 1 and nr 2 - all features used, distance weight: equal, distance metric: cityblock, number of neighbors: 100

When it comes to machine learning models with distance weight define as 'inverse' and several distance metrics and numbers of neighbors, the results of testing them using two training sets are presented in Fig.5.12. For small numbers of neighbors the training accuracy is very high, almost 98 percent in some cases, however, the testing accuracy which is more reliable is rather low and it implies that those models overfit. For 50 and 100 neighbors the testing accuracy is much higher and some model perform well. For instance, the model's testing accuracy with number of neighbors set to 100 and distance metric define as 'Cityblock' obtains around 92-94 percent, even though that the testing accuracy is a bit lower - 85.73 percent. There are six different distance metrics used and for most of them the accuracy is quite high. The lowest accuracy is for 'Chebychev' and 'Mahalanobis'.

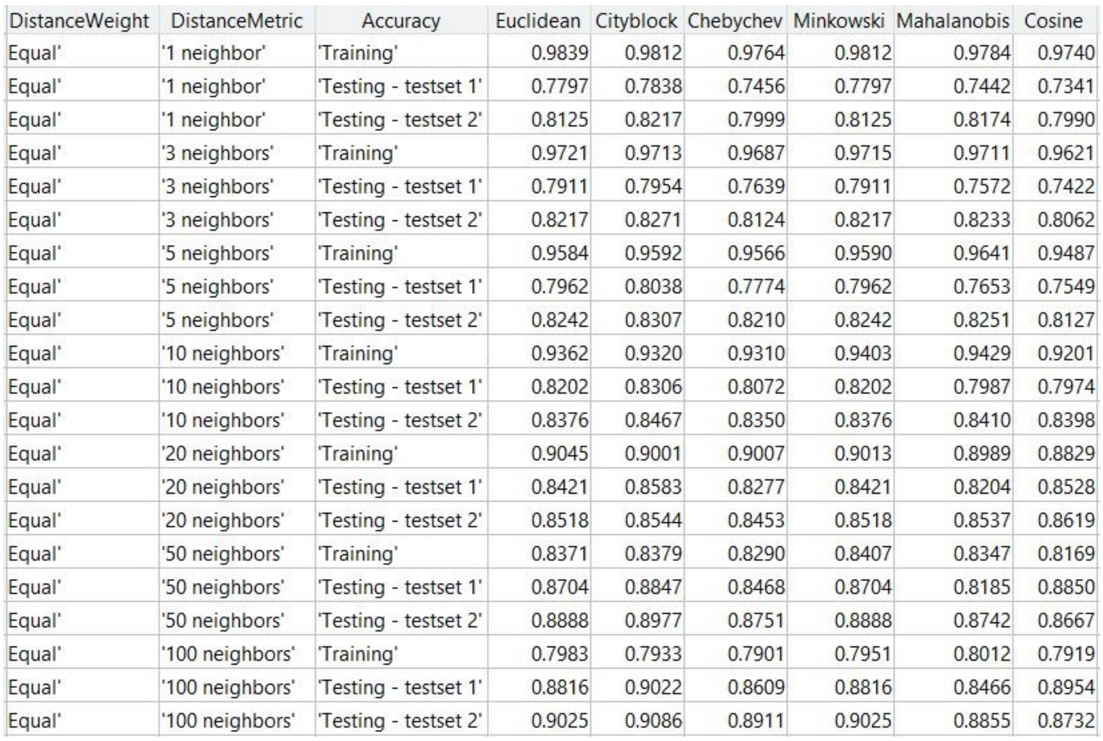


DistanceWeight	DistanceMetric	Accuracy	Euclidean	Cityblock	Chebychev	Minkowski	Mahalanobis	Cosine
'Inverse'	'1 neighbor'	'Training'	0.9830	0.9867	0.9742	0.9869	0.9843	0.9869
'Inverse'	'1 neighbor'	'Testing - testset 1'	0.7207	0.6844	0.4682	0.7207	0.4918	0.7578
'Inverse'	'1 neighbor'	'Testing - testset 2'	0.6590	0.7773	0.6263	0.6590	0.4544	0.5915
'Inverse'	'3 neighbors'	'Training'	0.9820	0.9855	0.9701	0.9798	0.9820	0.9859
'Inverse'	'3 neighbors'	'Testing - testset 1'	0.7349	0.7071	0.4742	0.7349	0.5293	0.7620
'Inverse'	'3 neighbors'	'Testing - testset 2'	0.6739	0.8125	0.6369	0.6739	0.4726	0.6189
'Inverse'	'5 neighbors'	'Training'	0.9816	0.9843	0.9725	0.9822	0.9800	0.9824
'Inverse'	'5 neighbors'	'Testing - testset 1'	0.7392	0.7099	0.4898	0.7392	0.5495	0.7577
'Inverse'	'5 neighbors'	'Testing - testset 2'	0.6843	0.8024	0.6594	0.6843	0.4848	0.6383
'Inverse'	'10 neighbors'	'Training'	0.9768	0.9816	0.9673	0.9782	0.9762	0.9812
'Inverse'	'10 neighbors'	'Testing - testset 1'	0.7510	0.7254	0.5284	0.7510	0.5728	0.7762
'Inverse'	'10 neighbors'	'Testing - testset 2'	0.7059	0.8139	0.5990	0.7059	0.5004	0.8086
'Inverse'	'20 neighbors'	'Training'	0.9623	0.9647	0.9469	0.9608	0.9596	0.9750
'Inverse'	'20 neighbors'	'Testing - testset 1'	0.7813	0.7869	0.5948	0.7813	0.6212	0.7943
'Inverse'	'20 neighbors'	'Testing - testset 2'	0.7916	0.8403	0.7637	0.7916	0.5357	0.8525
'Inverse'	'50 neighbors'	'Training'	0.9164	0.9164	0.8940	0.9128	0.9140	0.9606
'Inverse'	'50 neighbors'	'Testing - testset 1'	0.8803	0.8969	0.6264	0.8803	0.7245	0.8729
'Inverse'	'50 neighbors'	'Testing - testset 2'	0.9154	0.9247	0.8791	0.9154	0.6291	0.9180
'Inverse'	'100 neighbors'	'Training'	0.8569	0.8573	0.8341	0.8579	0.8686	0.9388
'Inverse'	'100 neighbors'	'Testing - testset 1'	0.9005	0.9160	0.6267	0.9005	0.7564	0.8985
'Inverse'	'100 neighbors'	'Testing - testset 2'	0.9279	0.9372	0.8955	0.9279	0.6974	0.9287

Figure 5.12: Accuracy: all features used and distance weight define as 'inverse'

Used features: only gyroscope

Since from the scatter plots it turned out that accelerometer features may decrease the performance of the model, it is good idea to train models with only gyroscope attributes selected and to observe if the accuracy increased. Thus all previously trained model are build once again with the same parameters but with removed accelerometer variables so only gyroscope features are taking into account. Fig.5.13 depicts table with accuracy of training models using k-nearest neighbors algorithm with only gyroscope data used, distance weight define as 'equal', several distance metrics and numbers of neighbors and testing them using two training sets. It turns out that after removing accelerometer features the testing accuracy decreased a few percent for 50 and 100 neighbors. For instance, previously analyzed model in Fig. 5.11, which obtained around 92-94 percent testing accuracy, now obtains around 90-91 percent. However, for smaller number of neighbors the testing accuracy has increased significantly but still those values are quite low.



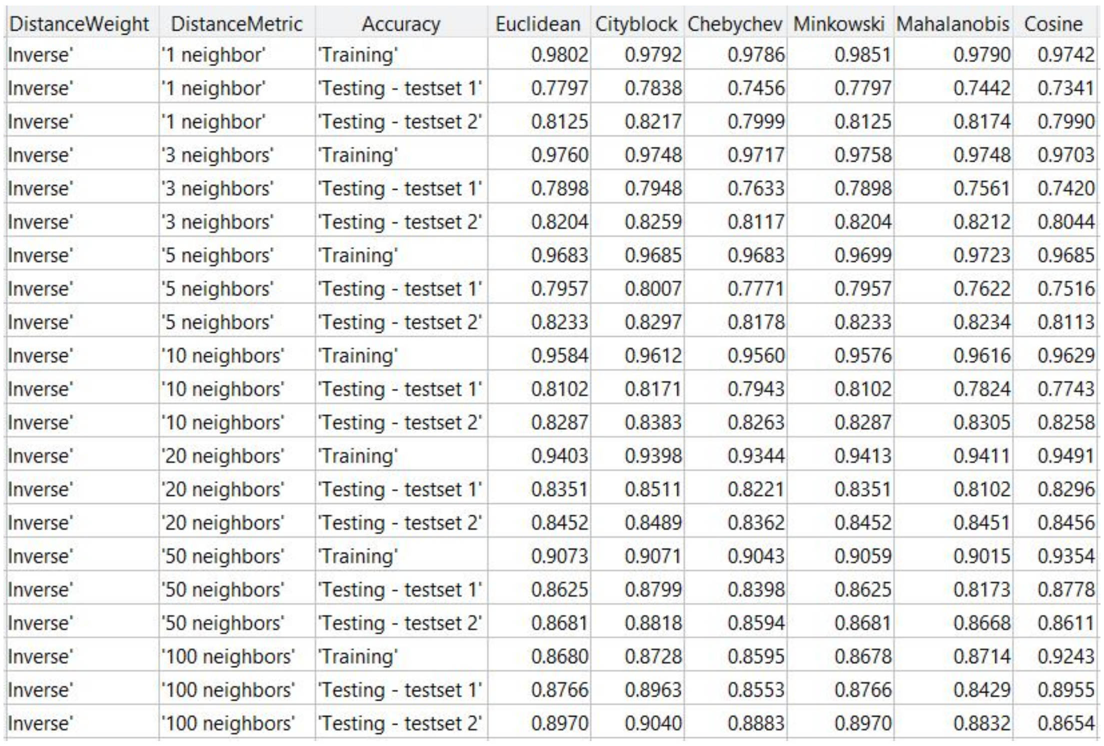
DistanceWeight	DistanceMetric	Accuracy	Euclidean	Cityblock	Chebychev	Minkowski	Mahalanobis	Cosine
Equal'	'1 neighbor'	'Training'	0.9839	0.9812	0.9764	0.9812	0.9784	0.9740
Equal'	'1 neighbor'	'Testing - testset 1'	0.7797	0.7838	0.7456	0.7797	0.7442	0.7341
Equal'	'1 neighbor'	'Testing - testset 2'	0.8125	0.8217	0.7999	0.8125	0.8174	0.7990
Equal'	'3 neighbors'	'Training'	0.9721	0.9713	0.9687	0.9715	0.9711	0.9621
Equal'	'3 neighbors'	'Testing - testset 1'	0.7911	0.7954	0.7639	0.7911	0.7572	0.7422
Equal'	'3 neighbors'	'Testing - testset 2'	0.8217	0.8271	0.8124	0.8217	0.8233	0.8062
Equal'	'5 neighbors'	'Training'	0.9584	0.9592	0.9566	0.9590	0.9641	0.9487
Equal'	'5 neighbors'	'Testing - testset 1'	0.7962	0.8038	0.7774	0.7962	0.7653	0.7549
Equal'	'5 neighbors'	'Testing - testset 2'	0.8242	0.8307	0.8210	0.8242	0.8251	0.8127
Equal'	'10 neighbors'	'Training'	0.9362	0.9320	0.9310	0.9403	0.9429	0.9201
Equal'	'10 neighbors'	'Testing - testset 1'	0.8202	0.8306	0.8072	0.8202	0.7987	0.7974
Equal'	'10 neighbors'	'Testing - testset 2'	0.8376	0.8467	0.8350	0.8376	0.8410	0.8398
Equal'	'20 neighbors'	'Training'	0.9045	0.9001	0.9007	0.9013	0.8989	0.8829
Equal'	'20 neighbors'	'Testing - testset 1'	0.8421	0.8583	0.8277	0.8421	0.8204	0.8528
Equal'	'20 neighbors'	'Testing - testset 2'	0.8518	0.8544	0.8453	0.8518	0.8537	0.8619
Equal'	'50 neighbors'	'Training'	0.8371	0.8379	0.8290	0.8407	0.8347	0.8169
Equal'	'50 neighbors'	'Testing - testset 1'	0.8704	0.8847	0.8468	0.8704	0.8185	0.8850
Equal'	'50 neighbors'	'Testing - testset 2'	0.8888	0.8977	0.8751	0.8888	0.8742	0.8667
Equal'	'100 neighbors'	'Training'	0.7983	0.7933	0.7901	0.7951	0.8012	0.7919
Equal'	'100 neighbors'	'Testing - testset 1'	0.8816	0.9022	0.8609	0.8816	0.8466	0.8954
Equal'	'100 neighbors'	'Testing - testset 2'	0.9025	0.9086	0.8911	0.9025	0.8855	0.8732

Figure 5.13: Only gyroscope used and distance weight define as 'equal'

For classification models with distance weight define as 'inverse' and several distance metrics and numbers of neighbors, the results of using two training sets are shown in Fig. 5.14. Similarly as in the previous case, after removing accelerometer features for 50 and 100 neighbors the accuracy decreased a few percent but for smaller number of neighbors the accuracy even increased but the values are still rather low.

Summary

Using k-nearest neighbors method to classify data from inertial sensors, it turns out that removing accelerometer features the testing accuracy increases or decreases depending on the numbers of neighbors. However, when it comes to the best trained model the removing of this variables causes decreasing of the testing accuracy. Increasing the number of neighbors causes increasing testing accuracy. The best scores gets the model with 100 neighbors. What is more, this value seems to be optimal since on average one ski turn lasts approximately 125 samples in the training set and around 85 sample in the testing sets. In terms of distance metric the best is 'cityblock' and when it comes to distance weight the most accurate is 'equal'.



DistanceWeight	DistanceMetric	Accuracy	Euclidean	Cityblock	Chebychev	Minkowski	Mahalanobis	Cosine
Inverse'	'1 neighbor'	'Training'	0.9802	0.9792	0.9786	0.9851	0.9790	0.9742
Inverse'	'1 neighbor'	'Testing - testset 1'	0.7797	0.7838	0.7456	0.7797	0.7442	0.7341
Inverse'	'1 neighbor'	'Testing - testset 2'	0.8125	0.8217	0.7999	0.8125	0.8174	0.7990
Inverse'	'3 neighbors'	'Training'	0.9760	0.9748	0.9717	0.9758	0.9748	0.9703
Inverse'	'3 neighbors'	'Testing - testset 1'	0.7898	0.7948	0.7633	0.7898	0.7561	0.7420
Inverse'	'3 neighbors'	'Testing - testset 2'	0.8204	0.8259	0.8117	0.8204	0.8212	0.8044
Inverse'	'5 neighbors'	'Training'	0.9683	0.9685	0.9683	0.9699	0.9723	0.9685
Inverse'	'5 neighbors'	'Testing - testset 1'	0.7957	0.8007	0.7771	0.7957	0.7622	0.7516
Inverse'	'5 neighbors'	'Testing - testset 2'	0.8233	0.8297	0.8178	0.8233	0.8234	0.8113
Inverse'	'10 neighbors'	'Training'	0.9584	0.9612	0.9560	0.9576	0.9616	0.9629
Inverse'	'10 neighbors'	'Testing - testset 1'	0.8102	0.8171	0.7943	0.8102	0.7824	0.7743
Inverse'	'10 neighbors'	'Testing - testset 2'	0.8287	0.8383	0.8263	0.8287	0.8305	0.8258
Inverse'	'20 neighbors'	'Training'	0.9403	0.9398	0.9344	0.9413	0.9411	0.9491
Inverse'	'20 neighbors'	'Testing - testset 1'	0.8351	0.8511	0.8221	0.8351	0.8102	0.8296
Inverse'	'20 neighbors'	'Testing - testset 2'	0.8452	0.8489	0.8362	0.8452	0.8451	0.8456
Inverse'	'50 neighbors'	'Training'	0.9073	0.9071	0.9043	0.9059	0.9015	0.9354
Inverse'	'50 neighbors'	'Testing - testset 1'	0.8625	0.8799	0.8398	0.8625	0.8173	0.8778
Inverse'	'50 neighbors'	'Testing - testset 2'	0.8681	0.8818	0.8594	0.8681	0.8668	0.8611
Inverse'	'100 neighbors'	'Training'	0.8680	0.8728	0.8595	0.8678	0.8714	0.9243
Inverse'	'100 neighbors'	'Testing - testset 1'	0.8766	0.8963	0.8553	0.8766	0.8429	0.8955
Inverse'	'100 neighbors'	'Testing - testset 2'	0.8970	0.9040	0.8883	0.8970	0.8832	0.8654

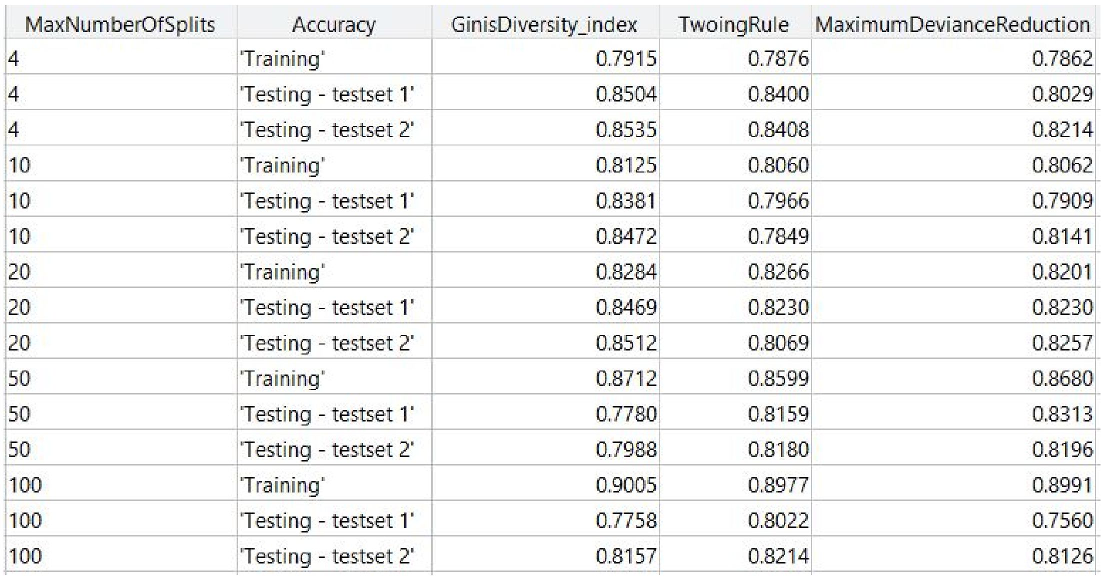
Figure 5.14: Only gyroscope used and distance weight define as 'inverse'

5.2.2 Decision Trees method

Decision Trees algorithm is tested using all attributes and then only gyroscope data, in both cases with several split criterions and maximum numbers of splits.

All features used: gyroscope and accelerometer

Fig.5.15 depicts table with accuracy of training models using decision tree algorithm with all features and testing them using two training sets. Colums indicates split criterions. For maximal 4, 10 and 20 of splits the training accuracy obtains around 79-82 percent and the testing accuracy about 80-84 percent for first test set and 78-85 percent for second test set. For maximal 50 and 100 of splits the training accuracy starts to rise up and the testing accuracy is much lower. The model with maximal number of splits set to 4 and split criterion defined as 'Gini's Diversity index' performs the best since in this case the testing accuracy obtains around 85 percent. For all the split criterions the accuracy have similar values, however, for 'Gini's Diversity index' the values are the highest.



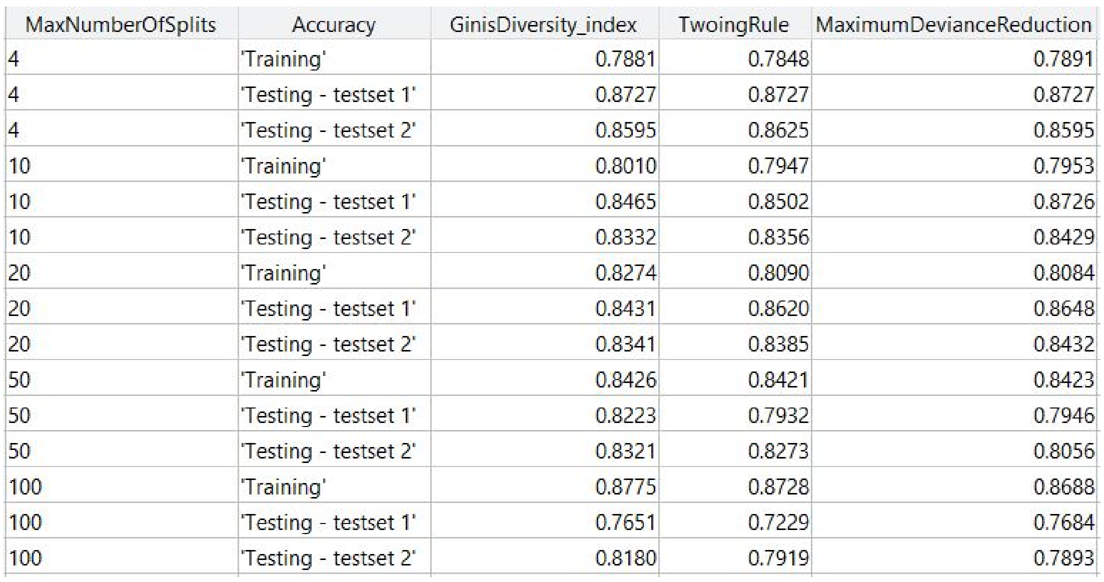
MaxNumberOfSplits	Accuracy	GinisDiversity_index	TwoingRule	MaximumDevianceReduction
4	'Training'	0.7915	0.7876	0.7862
4	'Testing - testset 1'	0.8504	0.8400	0.8029
4	'Testing - testset 2'	0.8535	0.8408	0.8214
10	'Training'	0.8125	0.8060	0.8062
10	'Testing - testset 1'	0.8381	0.7966	0.7909
10	'Testing - testset 2'	0.8472	0.7849	0.8141
20	'Training'	0.8284	0.8266	0.8201
20	'Testing - testset 1'	0.8469	0.8230	0.8230
20	'Testing - testset 2'	0.8512	0.8069	0.8257
50	'Training'	0.8712	0.8599	0.8680
50	'Testing - testset 1'	0.7780	0.8159	0.8313
50	'Testing - testset 2'	0.7988	0.8180	0.8196
100	'Training'	0.9005	0.8977	0.8991
100	'Testing - testset 1'	0.7758	0.8022	0.7560
100	'Testing - testset 2'	0.8157	0.8214	0.8126

Figure 5.15: All features used, maximum 4 splits and split criterion defined as 'Gini's Diversity index'

Used features: only gyroscope

Fig.5.16 depicts table with accuracy of training different models using decision tree algorithm with only gyroscope data and testing them using two training sets. Columns indicates split criterions.

When the accelerometer attributes are removed, the testing accuracy increased a few percent. The best model obtains 78.48 percent as training accuracy and around 87 percent as testing accuracy for maximal 4 of splits and 'Twoing Rule' as the split criterion. The model's predictions are visualised on the plot and compared to the correct predictions, as shown in Fig. 5.17. The green line indicates correctly predicted responses and the red line shows misclassifications. It can be observed that in both examples the real ski-turns are generally detected, but there are also many incorrectly predicted ski-turns when they do not really occur. Thus the model performs poorly.



MaxNumberOfSplits	Accuracy	GinisDiversity_index	TwoingRule	MaximumDevianceReduction
4	'Training'	0.7881	0.7848	0.7891
4	'Testing - testset 1'	0.8727	0.8727	0.8727
4	'Testing - testset 2'	0.8595	0.8625	0.8595
10	'Training'	0.8010	0.7947	0.7953
10	'Testing - testset 1'	0.8465	0.8502	0.8726
10	'Testing - testset 2'	0.8332	0.8356	0.8429
20	'Training'	0.8274	0.8090	0.8084
20	'Testing - testset 1'	0.8431	0.8620	0.8648
20	'Testing - testset 2'	0.8341	0.8385	0.8432
50	'Training'	0.8426	0.8421	0.8423
50	'Testing - testset 1'	0.8223	0.7932	0.7946
50	'Testing - testset 2'	0.8321	0.8273	0.8056
100	'Training'	0.8775	0.8728	0.8688
100	'Testing - testset 1'	0.7651	0.7229	0.7684
100	'Testing - testset 2'	0.8180	0.7919	0.7893

Figure 5.16: Only gyroscope data used, maximum 4 splits and split criterion defined as 'Twoing Rule'

Summary

Using decision tree model to classify data from inertial sensors, it turns out that removing accelerometer features causes increasing of testing accuracy of both test sets - reducing number of misclassified ski-turns. An important factor is maximum number of splits, the highest accuracy is for maximum 4 splits and when this number increases the testing accuracy descreases. When it comes to split criterions the differences are very small, however, the 'twoing rule' is the best choice since it guarantees the highest accuracy.

5.2.3 Support vector machines method

Support vector machines algorithm is tested using all attributes and then only gyroscope data, in both cases with several box constraint levels and kernel scales. In terms of kernel functions, four functions were used but only two of them 'Linear' and 'Quadratic' are mentioned in this project since two others: 'Cubic' and 'Gaussian' provide a little bit worse results.

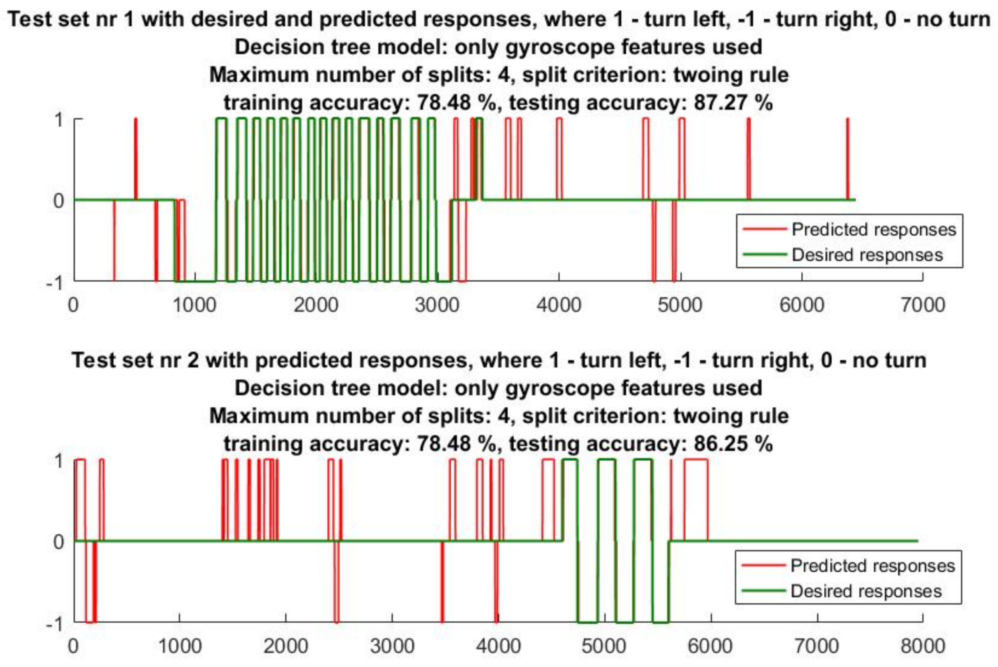
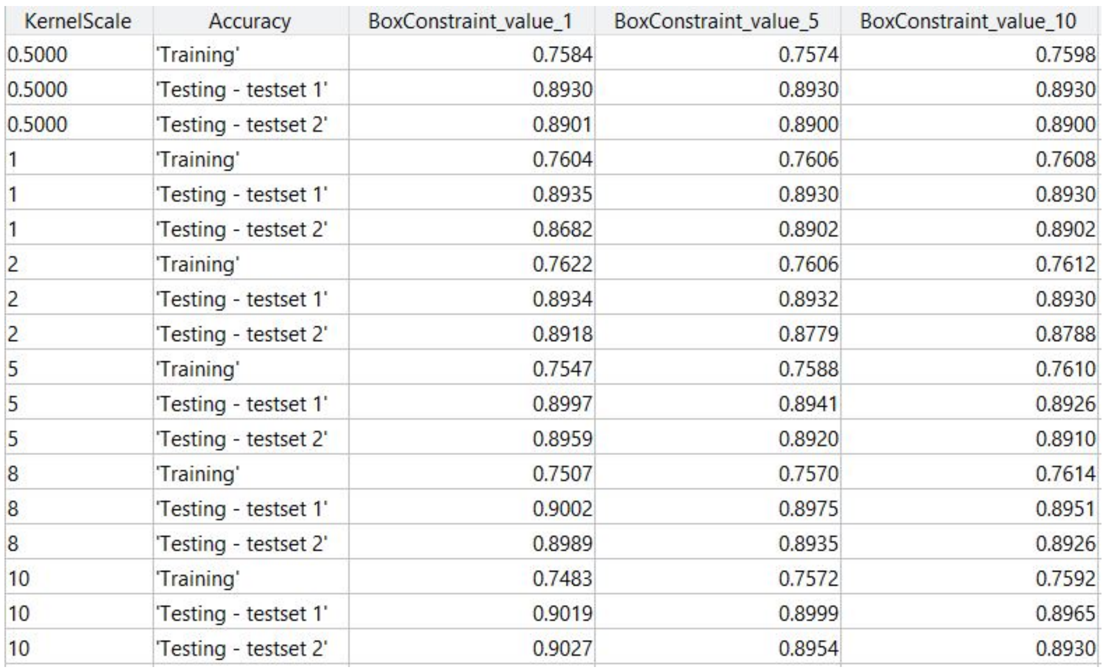


Figure 5.17: Test set nr 1 and nr 2 - only gyroscope data used, maximum 4 splits and split criterion defined as 'Twoing Rule'

All features used: gyroscope and accelerometer

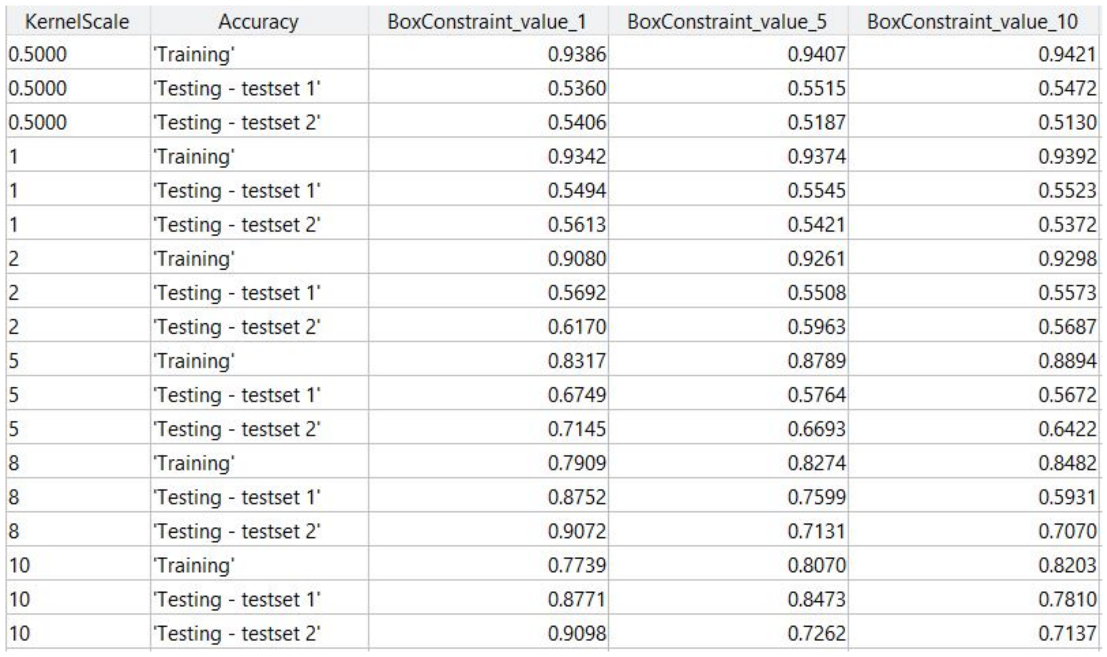
Fig.5.18 depicts table with accuracy of training classification models using support vector machines algorithm with all features used and linear kernel function, and then testing them using two training sets. In all cases the training accuracy obtains around 75-76 percent, and the testing accuracy about 89-90 percent for the first test set and about 88-90 percent for the second test set. The values are very similar and do not really depend on the box constaint levels and the kernel scales. The best model scores about 90 percent as testing accuracy and around 75 percent as training accuracy with kernel scale set to 10 and box constraint level set to 1. It happens that the testing accuracy is higher than the training accuracy because training accuracy is calculated based on training set which is balanced, while testing sets are the whole signals from sensors and constain mainly samples that refer to . Since it turns out that it is quite easy for the classifiers to predict *no turn* class, it happens that in some cases the testing accuracy is much higher than the training accuracy.



KernelScale	Accuracy	BoxConstraint_value_1	BoxConstraint_value_5	BoxConstraint_value_10
0.5000	'Training'	0.7584	0.7574	0.7598
0.5000	'Testing - testset 1'	0.8930	0.8930	0.8930
0.5000	'Testing - testset 2'	0.8901	0.8900	0.8900
1	'Training'	0.7604	0.7606	0.7608
1	'Testing - testset 1'	0.8935	0.8930	0.8930
1	'Testing - testset 2'	0.8682	0.8902	0.8902
2	'Training'	0.7622	0.7606	0.7612
2	'Testing - testset 1'	0.8934	0.8932	0.8930
2	'Testing - testset 2'	0.8918	0.8779	0.8788
5	'Training'	0.7547	0.7588	0.7610
5	'Testing - testset 1'	0.8997	0.8941	0.8926
5	'Testing - testset 2'	0.8959	0.8920	0.8910
8	'Training'	0.7507	0.7570	0.7614
8	'Testing - testset 1'	0.9002	0.8975	0.8951
8	'Testing - testset 2'	0.8989	0.8935	0.8926
10	'Training'	0.7483	0.7572	0.7592
10	'Testing - testset 1'	0.9019	0.8999	0.8965
10	'Testing - testset 2'	0.9027	0.8954	0.8930

Figure 5.18: All features used, linear kernel function

When it comes to machine learning models with quadratic kernel function and all features used, the results of training them and then testing using two test sets are presented in Fig. 5.19. For kernel scale 0.5, 1 and 2 the training accuracy is very high above 90 percent but the testing accuracy is very low around 50-60 percent. Those models overfit and therefore do not perfom well on unseen examples. For kernel scale 5, 8 and 10 the models perform much better, escpecially for 8 and 10 since the training accruacy obtains around 77-85 percent and the testing accuracy about 60-88 for the first test set and 71-91 percent for the second test set. However, the highest accuracy is obtained for box constaint value 1 and the lowest for box constaint with value 10. The best model with box constaint value 1 and kernel scale 10 has the training accuracy around 77 percent and testing accuracy about 88-91 percent.



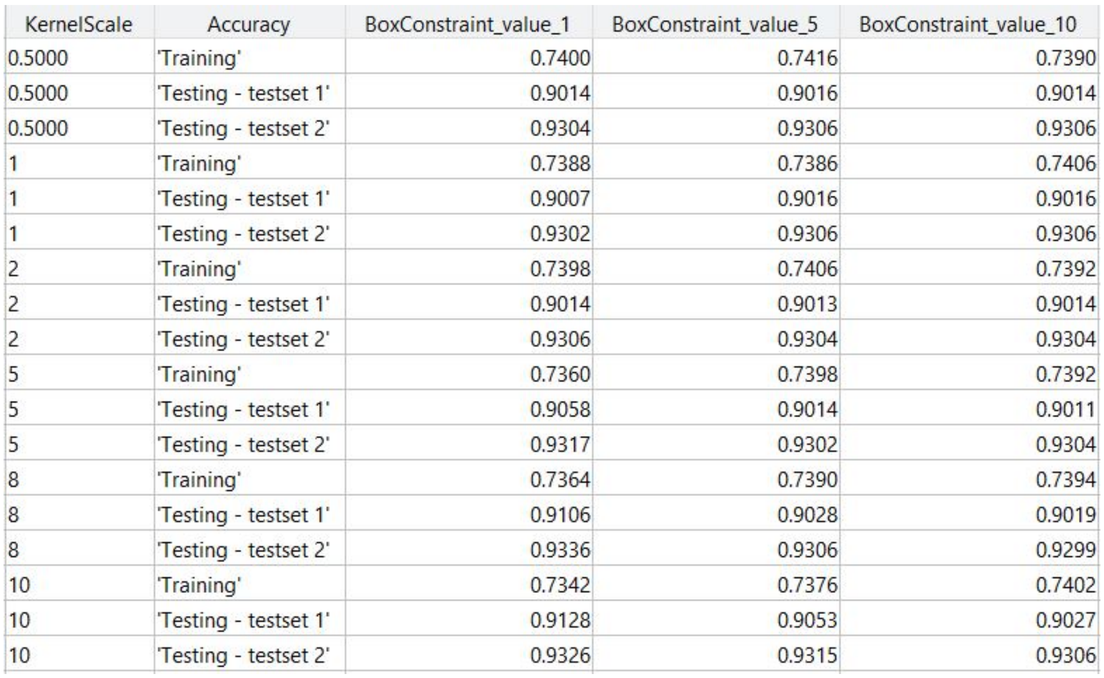
KernelScale	Accuracy	BoxConstraint_value_1	BoxConstraint_value_5	BoxConstraint_value_10
0.5000	'Training'	0.9386	0.9407	0.9421
0.5000	'Testing - testset 1'	0.5360	0.5515	0.5472
0.5000	'Testing - testset 2'	0.5406	0.5187	0.5130
1	'Training'	0.9342	0.9374	0.9392
1	'Testing - testset 1'	0.5494	0.5545	0.5523
1	'Testing - testset 2'	0.5613	0.5421	0.5372
2	'Training'	0.9080	0.9261	0.9298
2	'Testing - testset 1'	0.5692	0.5508	0.5573
2	'Testing - testset 2'	0.6170	0.5963	0.5687
5	'Training'	0.8317	0.8789	0.8894
5	'Testing - testset 1'	0.6749	0.5764	0.5672
5	'Testing - testset 2'	0.7145	0.6693	0.6422
8	'Training'	0.7909	0.8274	0.8482
8	'Testing - testset 1'	0.8752	0.7599	0.5931
8	'Testing - testset 2'	0.9072	0.7131	0.7070
10	'Training'	0.7739	0.8070	0.8203
10	'Testing - testset 1'	0.8771	0.8473	0.7810
10	'Testing - testset 2'	0.9098	0.7262	0.7137

Figure 5.19: All features used, quadratic kernel function

When it comes to different kernel functions, both linear and quadratic functions have high accuracy, but analyzing model on the plots it was found that model with quadratic kernel function fails to detect many ski-turns.

Used features: only gyroscope

Since accelerometer features might decrease the performance of the machine learning models in this project, the support vector machines algortihm is also tested using only gyroscope variables. The results of building and testing such models with several kernel scales and box constraints with linear kernel function are presented in Fig. 5.20. It can be observed that these model compared to models with accelerometer features included (see Fig. 5.18) obtains higher testing accuracy. For every kernel scale value the testing accuracy increased from around 86-90 percent to 90-93 percent. What is more, all values are high regarless of the kernel scales and the box constraints. The highest values obtains the classification model with kernel scale set to 10 and box constraint set to 1: training accuracy around 73 percent and 91.28 percent testing accuracy for the first test set and 93.26 percent



KernelScale	Accuracy	BoxConstraint_value_1	BoxConstraint_value_5	BoxConstraint_value_10
0.5000	'Training'	0.7400	0.7416	0.7390
0.5000	'Testing - testset 1'	0.9014	0.9016	0.9014
0.5000	'Testing - testset 2'	0.9304	0.9306	0.9306
1	'Training'	0.7388	0.7386	0.7406
1	'Testing - testset 1'	0.9007	0.9016	0.9016
1	'Testing - testset 2'	0.9302	0.9306	0.9306
2	'Training'	0.7398	0.7406	0.7392
2	'Testing - testset 1'	0.9014	0.9013	0.9014
2	'Testing - testset 2'	0.9306	0.9304	0.9304
5	'Training'	0.7360	0.7398	0.7392
5	'Testing - testset 1'	0.9058	0.9014	0.9011
5	'Testing - testset 2'	0.9317	0.9302	0.9304
8	'Training'	0.7364	0.7390	0.7394
8	'Testing - testset 1'	0.9106	0.9028	0.9019
8	'Testing - testset 2'	0.9336	0.9306	0.9299
10	'Training'	0.7342	0.7376	0.7402
10	'Testing - testset 1'	0.9128	0.9053	0.9027
10	'Testing - testset 2'	0.9326	0.9315	0.9306

Figure 5.20: Only gyroscope data used, linear kernel function

for the second test set. The model with such parameters but with all features previosly got around 90 percent testing accuracy, therefore removing accelerometer features caused increasing of the testing accuracy about 1-3 percent. This model seems to perform really well and thus the model's predictions are visualised on the plot and compared to the correct predictions, as shown in Fig. 5.21. The green line indicates correctly predicted responses and the red line shows misclassifications. It can be observed that in both examples the real ski-turns are generally detected with few incorrectly predicted ski-turns when they do not really occur. However, the model perform well.

For quadratic kernel function the results of training models using support vector machines algorithm with only gyroscope and several kernel scales and box constaint levels and then testing them using two training sets are presented in Fig. 5.22. It can be observed that for higher kernel scales the accuracy is much higher. Comparing accuracy of models with the same parameteres but with all features used i.e. acceleremeter and gyroscope (see Fig. 5.19) and only gyroscope (see Fig. 5.22), it turns out that the tested accuracy increased significantly when the accelerometer fatures are removed. The testing accuracy for test set nr 1 obtains 80-81 pernent for kernel scales 0.5, 1 and 2 (previously it was only 54-56 percent), and 82-88 percent for kernel scales 5, 8 and 10 (previously it was 51-62 percent). For the second test set the testing accuracy is 85-88 percent for kernel scales 0.5, 1 and 2 (previously it was 57-88 percent), and 90-93 percent for kernel scale 5, 8 and 10 (previously it was 64-91 percent).

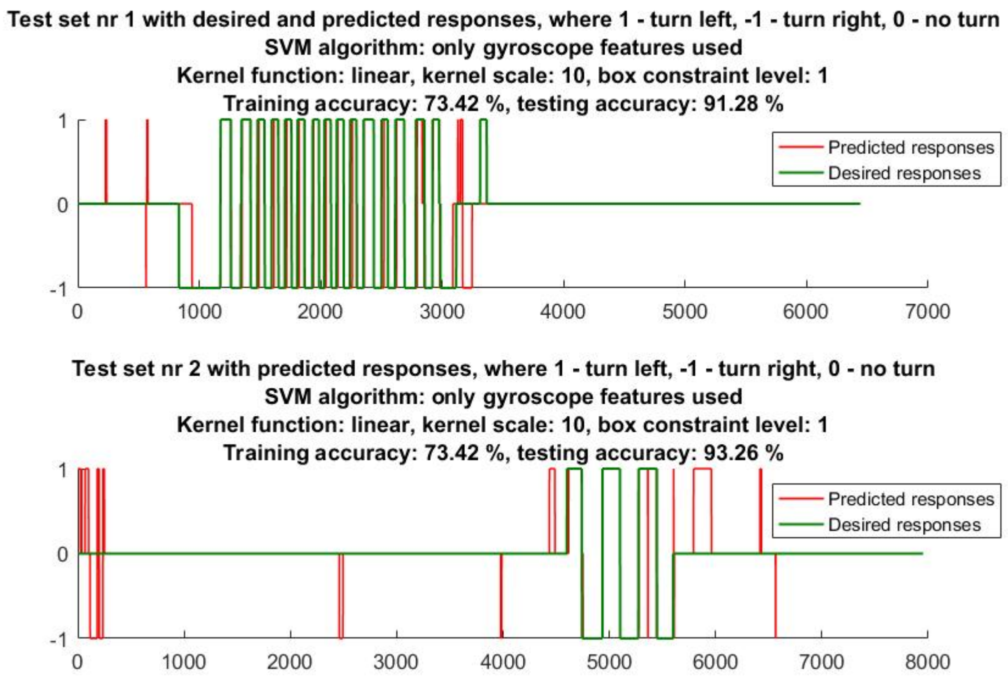
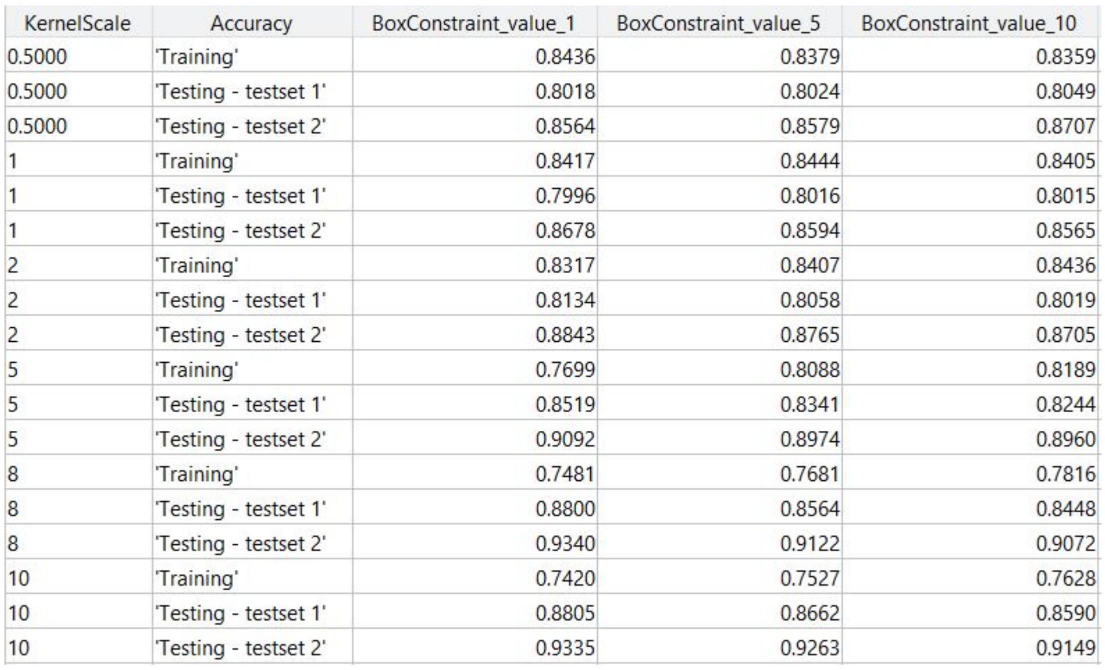


Figure 5.21: Test set nr 1 and nr 2 - only gyroscope data used, linear kernel function, kernel scale: 10, box constraint level: 1

Summary

Testing support vector machine algorithm with different parameters, it turns out that when the accelerometer features are removed, the testing accuracy increases and thus the model effectiveness impoves. When it comes to different kernel functions, the highest accuracy ensures linear function. Increasing kernel scale the testing accuracy increases, with the best scores for the value 10. The optimal box constraint level is 1, since for bigger values the accuracy is a little bit lower and it takes much more time to train a model.



KernelScale	Accuracy	BoxConstraint_value_1	BoxConstraint_value_5	BoxConstraint_value_10
0.5000	'Training'	0.8436	0.8379	0.8359
0.5000	'Testing - testset 1'	0.8018	0.8024	0.8049
0.5000	'Testing - testset 2'	0.8564	0.8579	0.8707
1	'Training'	0.8417	0.8444	0.8405
1	'Testing - testset 1'	0.7996	0.8016	0.8015
1	'Testing - testset 2'	0.8678	0.8594	0.8565
2	'Training'	0.8317	0.8407	0.8436
2	'Testing - testset 1'	0.8134	0.8058	0.8019
2	'Testing - testset 2'	0.8843	0.8765	0.8705
5	'Training'	0.7699	0.8088	0.8189
5	'Testing - testset 1'	0.8519	0.8341	0.8244
5	'Testing - testset 2'	0.9092	0.8974	0.8960
8	'Training'	0.7481	0.7681	0.7816
8	'Testing - testset 1'	0.8800	0.8564	0.8448
8	'Testing - testset 2'	0.9340	0.9122	0.9072
10	'Training'	0.7420	0.7527	0.7628
10	'Testing - testset 1'	0.8805	0.8662	0.8590
10	'Testing - testset 2'	0.9335	0.9263	0.9149

Figure 5.22: Only gyroscope data used, quadratic kernel function

5.2.4 Algorithms comparizon

From each method the best model is chosen and evaluated in terms of accuracy, training time, prediction speed and overall impression based on plots. As shown in table 5.1, k-nearest neighbors method ensures the highest both training and testing accuracy, however, in support vector machines method the testing accuracy is only around half percent lower. When it comes to prediction speed and training time, decision tree method has the best scores. SVM algorithm is almost six times faster in predicting responses than KNN algorithm, but the training time is two times greater.

The results of testing three algorithms on test set nr 1 are presented in Fig. 5.23. Green line indicates correctly predicted responses and red line misclassifications. It can be observed that for decision tree method there are many samples classified as either *right* or *left turn* instead of *no turn* class thus this method

Table 5.1: Methods comparison

	training algorithm		
	KNN	Decision Tree	SVM
used features	all	gyroscope	gyroscope
training accuracy [percent]	79.35	78.48	73.42
testing accuracy (test set nr 1) [percent]	91.79	87.27	91.28
testing accuracy (test set nr 2) [percent]	93.82	86.25	93.26
prediction speed [obs/sec]	10 000	140 000	59 000
training time [secs]	2.3123	1.38	4.196

performs worse than others. When it comes to SVM and KNN algorithms, the SVM methods predicts turns more precisely and there are less misclassifications. It turns out that in this problem the best machine learning algorithm is support vector machines method since ensures the best classifier's performance in terms of ski-turns detection. What is more, this algorithm is fast in predicting response and this is an important factor when it comes to integrating best-trained model into a production system and using it on un-seen data which can be a huge dataset.

5.2.5 Post-classification filtration and ski-turns counting

Generally, one ski-turn lasts at least dozens of samples thus when a classified ski-turn lasts only a few samples it is just misclassification and shall be filtered. Such incorrectly classified ski-turns can be noticed for instance in Fig. 5.21. Analyzing different datasets it can be assumed that the shortest ski-turns can last 30 samples. In the project additional script is applied which counts how many samples each ski-turns last, and when it is less than 30, then this ski-turn is recognized as misclassification and these records are reassigned to *no turn* class.

Next based on filtered predictions the total number of left and right turns are calculated for both test sets. These results are compared to desired numbers of turns, as shown in Fig. 5.24. In first case, the predicted number of turns is very similar to the reference values. In second case, there are a few over-recognized ski-turns.

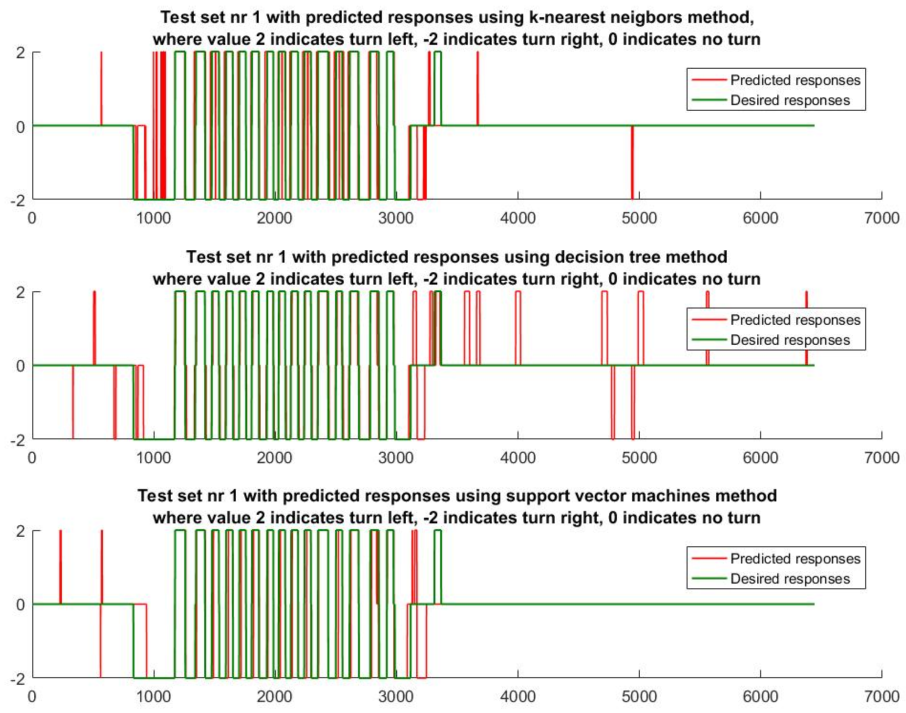


Figure 5.23: Test set nr 1 with predicted responses using three algorithms

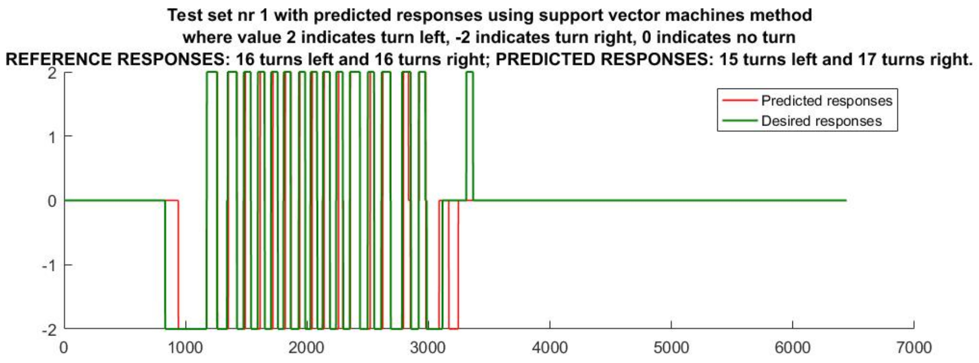
5.3 Program implementation

The program consists of three parts: windows desktop application, service and MATLAB scripts.

5.3.1 Windows desktop application

Windows desktop application constains main class named Form, which consists of following methods:

- SelectDataButton() - choosing file path to data sensors file
- DetectTurnsButton() - running a progress bar and revoking GetResponses() method in a second thread, which is presented in Fig. 5.25, and after the method is executed the DisplayResults () method is revoked



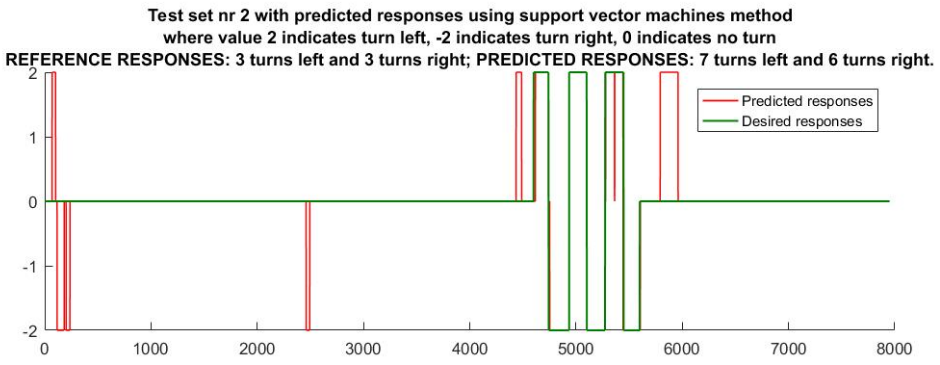


Figure 5.24: Test sets with counted ski-turns

- GetResponses() - reading data from selected file and sending data to service in packages of size 45000 bytes, and then after processing data by service getting results of classification
- DisplayResults () - displaying results of classification on plots

There are few more methods which are not mentionted here, since they only refer to proper displaying results on the chart.

5.3.2 Wcf service

Service consist of a few files, but the most important are

- IService.cs
- Service.svc

```
private void GetResponses()
{
    // create client to call service's methods
    ServiceClient client = new ServiceClient();

    // read data from selected file
    byte[] fileBytes = File.ReadAllBytes(
        @filePath);
    int arraySize = fileBytes.Length;

    // 45000 bytes are sent at a time
    int sizeOfPackage = 45000;
    int numberOfPackages = (int)(arraySize /
        sizeOfPackage) + 1;
    int sizeOfLastPackage = arraySize - (int)(
        arraySize / sizeOfPackage) *
        sizeOfPackage;

    FileStream readStream = new FileStream(
        @filePath, FileMode.Open);
    BinaryReader reader = new BinaryReader(
        readStream);

    // read and send each package to service
    for (int i = 1; i <= numberOfPackages; i++)
    {
        if (i < numberOfPackages)
        {
            byte[] package = reader.ReadBytes(
                sizeOfPackage);
            client.SendPackage(package,
                numberOfPackages);
        }
        else // for the last package the number
            of bytes is different
        {
            byte[] package = reader.ReadBytes(
                sizeOfLastPackage);
            client.SendPackage(package,
                numberOfPackages);
        }
    }
    results = client.DetectTurns();
    client.Close();
    reader.Close();
}
```

Figure 5.25: The GetResponses() method.

IService.sc

IService.cs file contains the interface IService and class OutputData. The interface IService (see Fig. 5.26) constains methods declarations that are exposed by the service, since they are marked with OperationContractAttribute attribute. The class OutputData has the DataContract attribute. This class defines type of the data that is serialized in order to be exchanged between service and client.

```
[ServiceContract]
public interface IService
{
    // Send data from client to service
    [OperationContract]
    void SendPackage(byte[] rawData, int
       numberOfpackages);

    // Use MATLAB scripts to process data and
       classify ski-turns
    [OperationContract]
    OutputData DetectTurns();
}
```

Figure 5.26: The IService interface.

Service.svc

Service.svc file contains main class Service , which inherits from IService interface. The class has two methods SendPackage and DetectTurns. The SendPackage method, which is presented in Fig. 5.27, is in charge of receiving data from client in packages (arrays of bytes) and writing them into temporary file located in the current project folder. The file is then used by DetectTurns method as an input parameter to MATLAB script. When the first package is received new instance of BinaryWriter (from System.IO namespace) class is opened, and when the last packages is already written to the file the writer is closed.

```
        public void SendPackage(byte[] rawData, int numberOfpackages)
        {
            countReceivedPackages++;

            if (countReceivedPackages == 1)
            {
                // Get directiories
                var currentDirectory = new DirectoryInfo(AppDomain.CurrentDomain.BaseDirectory);
                string projectDirectory = currentDirectory.Parent.FullName;
                newFilePath = projectDirectory + "\\matlab_scripts\\" + fileName;
                scriptDirectory = projectDirectory + "\\matlab_scripts";

                // Create temporary file with input data
                writeStream = new FileStream(newFilePath, FileMode.Create);
                writer = new BinaryWriter(writeStream);
            }

            // Write data to file
            writer.Write(rawData);

            // If this is last package, then close binary writer
            if (countReceivedPackages== numberOfpackages)
            {
                writer.Close();
                countReceivedPackages = 0;
            }
        }
```

Figure 5.27: The SendPackage method.

The DetectTurns method, which main part is presented in Fig. 5.28, allows to execute MATLAB script named classify_turns in order to get classification results and then saves results into instance of OutputData class instance in order to share the results with a client application.

```
        public OutputData DetectTurns()
        {
            // Change to the directory where the scripts
                are located
            string execute = "cd␣" + scriptDirectory;
            _matlab.Execute(@execute);

            // Define the output
            object output = null;

            // Call the MATLAB function classify_turns
            _matlab.Feval("classify_turns", 9, out
               output, newFilePath);

            // Define result as object
            object[] result = output as object[];

            // Prepare data to send to client
            OutputData classifiedTurns = new OutputData
               ();

        .......

            return classifiedTurns;
        }
```

Figure 5.28: The DetectTurns method.

5.3.3 MATLAB script

The MATLAB script, which is revoked by the service, as input parameter takes sensors data file. The data are processed using parsing function, filtered using moving average filter and represented as feature variables, all these steps are described in section 5.1.

Next support vector machines algorithm is used to predict ski-turns, since it turns out that this method performs the best in section 5.2.4. Details of implementation are presented in Fig. 5.29. The already trained model is loaded into MATLAB environment from the file and then the model is used to predict responses. The results are stored in numerical form, to be able to display results on plots.

```
%Load trained classification model
load('SVM_model');

%Classify turns
predictions = SVM_model.predictFcn(
    SkiTurns_dataset_to_classify);

%Write labels in numerical form to display results on
    the plot
classified_labels=zeros(size(predictions,1),1);
for i=1:size(predictions,1)
    if strcmp(predictions(i),'Right␣turn')
        classified_labels(i,1)=-1;
    elseif strcmp(predictions(i),'Left␣turn')
        classified_labels(i,1)=1;
    elseif strcmp(predictions(i),'No␣turn')
        classified_labels(i,1)=0;
    end
end
```

Figure 5.29: The fragment of classify_turns function.

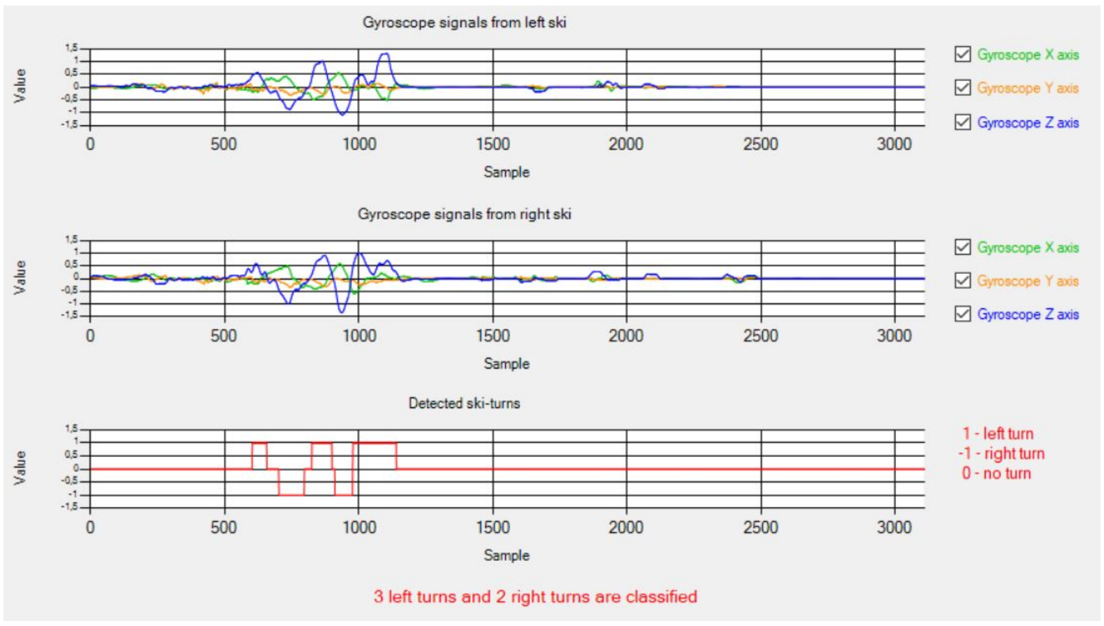
At the end additional filtration is applied to predicted responses and the total number of turns is calculated, as described in section 5.2.5. The function returns filtered data from gyroscope left and right ski - six variables, labels, and numbers of detected right and lef turns.

Chapter 6

Verification and validation

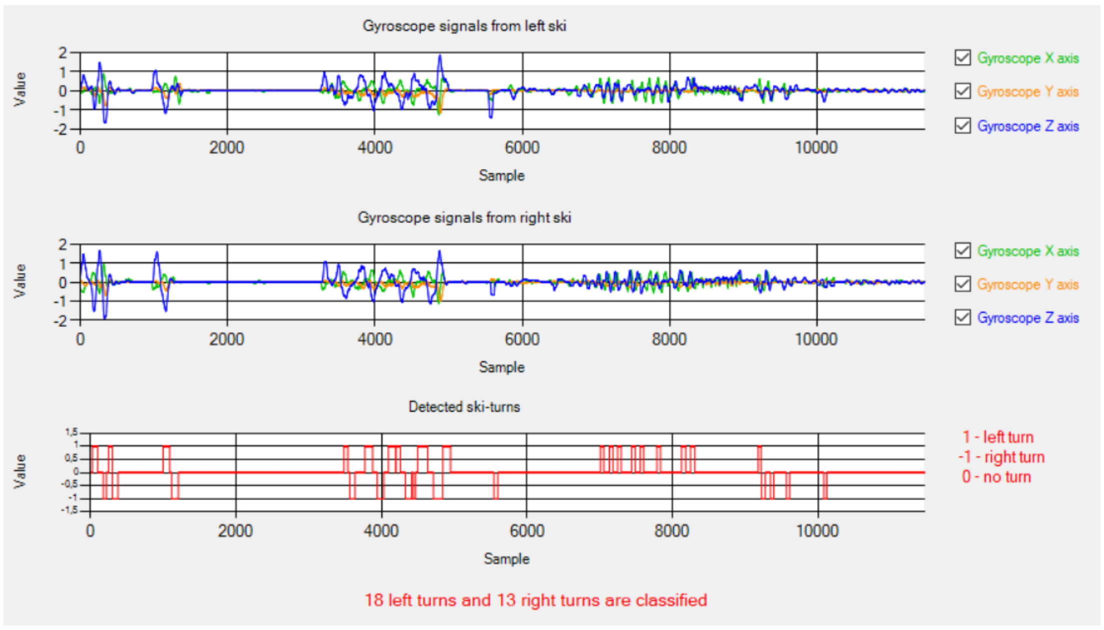
The most extensive part of the project was testing many different classification models and choosing the best one since it consumed the majority of time. It was necessary to analyze many tables with accuracy, select the ones with the highest values and observe how effective they are in predicting the answers on the charts. The relationships between the various parameters and results had to be studied. Some parameters caused that the time of training was significantly lengthened or that the accuracy was reduced. The values of accuracy were often very similar and the details were crucial. Moreover, high accuracy did not always mean an effective model. It happened that models with a little lower accuracy predicted better, which could be seen in the charts.

The final application was tested using two sensors data files, which were used to build the training set. Data were obtained from Snowcookie system, which was mentioned in chapter 2.4. The results generated by the program for the first dataset are presented in Fig. 6.1. There are 3 left turns and 2 right turns detected and these values are correct. Fig. 6.2 depicts the results generated by the program for the second dataset. This dataset contains many disturbances in the range 7000-10000 samples, and these noises are classified as turns which is not correct. It turns out that the application can detect ski-turns which really occured but it not so accurate when it comes to signal disturbances.



3 left turns and 2 right turns are classified

Figure 6.1: Application testing - first test set



18 left turns and 13 right turns are classified

Figure 6.2: Application testing - second test set

Chapter 7

Conclusions

The project itself it was just the introduction to machine learning study. To train the models, a ready-made tool was used that easily illustrates how the process works and then allows to generate code to automate training the model and use it on a new dataset. For this purpose it could also be used other tools, programming languages and libraries that are more supported by software developers and desribed in many book and tutorials. For training models three commonly used machine learning algorithms were used. Some of these methods have been developed for decades and are quite complicated but in the real world already implemented classifiers are used. However, in further development it is possible to come up with a new classification algorithm or generally method of ski-turns recognition based on data from inertial sensors.

When it comes to the best classification model, its accuracy obtains around 91-93 percent, as was presented in table 5.1. The model is able to correctly predict ski-turns in most cases and the accuracy is high. However, probably much better results could be achieved since the model had problems with disturbances as some of them were incorrectly recognised as ski-turns, as it was described in chapter 6.

Additionally, for the purpose of this project desktop application was created and in further development web and mobile applications can be created in order to allow more users to use the app.

Bibliography

[1] Accelerometer and gyroscopes sensors. https://www.edn.com/design/analog/4429624/Accelerometer-and-Gyroscopes-Sensors--Operation--Sensing--and-Applications-. [access date: 2019-10-13].

[2] Accelerometer vs. gyroscope: What's the difference? https://www.livescience.com/40103-accelerometer-vs-gyroscope.html. [access date: 2019-12-07].

[3] Artifical intelligence vs machine learning vs deep learning. https://medium.com/@mapendomobile/a-i-vs-machine-learning-vs-deep-learning-61428a98e347. [access date: 2019-10-13].

[4] Conservation of angular momentum. https://courses.lumenlearning.com/boundless-physics/chapter/conservation-of-angular-momentum/. [access date: 2019-10-13].

[5] Data preparation, preprocessing and wrangling in deep learning. https://www.xenonstack.com/blog/data-preparation/. [access date: 2019-10-13].

[6] Gyroscopic effects: Vector aspects of angular momentum. https://opentextbc.ca/physicstestbook2/chapter/gyroscopic-effects-vector-aspects-of-angular-momentum/. [access date: 2019-10-13].

[7] Hawx connected ski boot. https://www.nordicsemi.com/News/2019/03/Atomic-ski-boot-integrates-multiple-force-and-motion-sensors. [access date: 2019-12-11].

[8] Moving mean - matlab function. `https://www.mathworks.com/help/matlab/ref/movmean.html`. [access date: 2019-05-10].

[9] Rotation angle and angular velocity. `https://courses.lumenlearning.com/physics/chapter/6-1-rotation-angle-and-angular-velocity/`. [access date: 2019-12-07].

[10] Si units, symbols and abbreviations. `https://www.electronics-notes.com/articles/basic_concepts/si-system-international/si-units-symbols.php`. [access date: 2019-12-07].

[11] Snowcookie. `https://snowcookiesports.com/`. [access date: 2019-12-11].

[12] Support vector machines for binary classification. `https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html`. [access date: 2019-10-13].

[13] Vector nature of rotational kinematics. `https://courses.lumenlearning.com/boundless-physics/chapter/vector-nature-of-rotational-kinematics/`. [access date: 2019-10-13].

[14] Zepp tennis 2. `https://www.zepp.com/en-us/tennis/`. [access date: 2019-12-11].

[15] Angelo Maria Sabatini Andrea Mannini. Machine learning methods for classifying human physical activity from on-body accelerometers. *Sensors*, 19(10):1154–1175, 2010.

[16] Sarah Guido Andreas C. Müller. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2016.

[17] David Freeman Clarence Chio. *Machine Learning and Security: Protecting Systems with Data and Algorithms*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2018.

[18] Darren Cook. *Practical Machine Learning with H2O*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2016.

[19] Vladimir Vapnik Corinna Cortes. Support-vector networks. *Machine Learning*, (20):273–297, 1995.

[20] Adrian Kaehler Gary Bradski. *Learning OpenCV 3.* O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2016.

[21] Steve Gunn. Support vector machines for classification and regression. 1998.

[22] Marcin Karbowniczek. Układy mems. *Elektronika Praktyczna*, 26(2):54–56, 2010.

[23] Matthew Kirk. *Thoughtful Machine Learning with Python: A Test-Driven Approach.* O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2017.

[24] Jeroen D. Hol Manon Kok and Thomas B. Schon. Using inertial sensors for position and orientation estimation. *Foundations and Trends in Signal Processing*, 11(1-2):1–153, 2017.

[25] Edward Sazonov. *Wearable Sensors Fundamentals, Implementation and Applications.* Academic Press, 2014.

Appendices

List of abbreviations and symbols

IMU	Inertial measurement unit
VR	Virtual reality
MEMS	Microelectromechanical systems
SI	International System of Units
AI	Artificial intelligence
k-NN	k-Nearest Neighbors algorithm
SVMs	Support Vector Machines algorithm
C	penalty parameter
$gamma$	parameter for non linear hyperplanes

Listings

Listing 1: Training function - k-nearest neighbors method

```
function [trainedClassifier, validationAccuracy] = trainClassifier_KNN(trainingData,distance_metric,distance_weight,number_neighbors,used_features)

% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
inputTable = trainingData;
predictorNames = {'Left_Ski_Accelerometer_X_axis', 'Left_Ski_Accelerometer_Y_axis', 'Left_Ski_Accelerometer_Z_axis', 'Left_Ski_Gyroscope_X_axis', 'Left_Ski_Gyroscope_Y_axis', 'Left_Ski_Gyroscope_Z_axis', 'Right_Ski_Accelerometer_X_axis', 'Right_Ski_Accelerometer_Y_axis', 'Right_Ski_Accelerometer_Z_axis', 'Right_Ski_Gyroscope_X_axis', 'Right_Ski_Gyroscope_Y_axis', 'Right_Ski_Gyroscope_Z_axis'};
predictors = inputTable(:, predictorNames);
response = inputTable.ClassifiedTurn;
```

```
isCategoricalPredictor = [false, false, false, false,
    false, false, false, false, false, false, false,
    false];

% Data transformation: Select subset of the features
% This code selects the same subset of features as were
    used in the app.
includedPredictorNames = predictors.Properties.
    VariableNames(used_features);
predictors = predictors(:,includedPredictorNames);
isCategoricalPredictor = isCategoricalPredictor(
    used_features);

% Train a classifier
% This code specifies all the classifier options and
    trains the classifier.
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', distance_metric, ...
    'Exponent', [], ...
    'NumNeighbors', number_neighbors, ...
    'DistanceWeight', distance_weight, ...
    'Standardize', true, ...
    'ClassNames', {'No␣turn'; 'Left␣turn'; 'Right␣turn'
       });

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
featureSelectionFcn = @(x) x(:,includedPredictorNames);
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x) knnPredictFcn(
    featureSelectionFcn(predictorExtractionFcn(x)));
```

```

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'Left_Ski_Accelerometer_X_axis', 'Left_Ski_Accelerometer_Y_axis', 'Left_Ski_Accelerometer_Z_axis', 'Left_Ski_Gyroscope_X_axis', 'Left_Ski_Gyroscope_Y_axis', 'Left_Ski_Gyroscope_Z_axis', 'Right_Ski_Accelerometer_X_axis', 'Right_Ski_Accelerometer_Y_axis', 'Right_Ski_Accelerometer_Z_axis', 'Right_Ski_Gyroscope_X_axis', 'Right_Ski_Gyroscope_Y_axis', 'Right_Ski_Gyroscope_Z_axis'};
trainedClassifier.ClassificationKNN = classificationKNN;
trainedClassifier.About = 'This␣struct␣is␣a␣trained␣classifier␣exported␣from␣Classification␣Learner␣R2016a.';
trainedClassifier.HowToPredict = sprintf('To␣make␣predictions␣on␣a␣new␣table,␣T,␣use:␣\n␣␣yfit␣=␣c.predictFcn(T)␣\nreplacing␣''c''␣with␣the␣name␣of␣the␣variable␣that␣is␣this␣struct,␣e.g.␣''trainedClassifier''.␣\n␣\nThe␣table,␣T,␣must␣contain␣the␣variables␣returned␣by:␣\n␣␣c.RequiredVariables␣\nVariable␣formats␣(e.g.␣matrix/vector,␣datatype)␣must␣match␣the␣original␣training␣data.␣\nAdditional␣variables␣are␣ignored.␣\n␣\nFor␣more␣information,␣see␣<a␣href="matlab:helpview(fullfile(docroot,␣''stats'',␣''stats.map''),␣''appclassification_exportmodeltoworkspace'')">How␣to␣predict␣using␣an␣exported␣model</a>.');

```

```
% Extract predictors and response
% This code processes the data into the right shape for
    training the
% classifier.
inputTable = trainingData;
predictorNames = {'Left_Ski_Accelerometer_X_axis', '
    Left_Ski_Accelerometer_Y_axis', '
    Left_Ski_Accelerometer_Z_axis', '
    Left_Ski_Gyroscope_X_axis', '
    Left_Ski_Gyroscope_Y_axis', '
    Left_Ski_Gyroscope_Z_axis', '
    Right_Ski_Accelerometer_X_axis', '
    Right_Ski_Accelerometer_Y_axis', '
    Right_Ski_Accelerometer_Z_axis', '
    Right_Ski_Gyroscope_X_axis', '
    Right_Ski_Gyroscope_Y_axis', '
    Right_Ski_Gyroscope_Z_axis'};
predictors = inputTable(:, predictorNames);
response = inputTable.ClassifiedTurn;
isCategoricalPredictor = [false, false, false, false,
    false, false, false, false, false, false, false,
    false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.
    ClassificationKNN, 'KFold', 5);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, '
    LossFun', 'ClassifError');
```

Contents of attached CD

The thesis is accompanied by a CD containing:

- thesis (LaTeX source files and final `pdf` file),
- source code of the desktop application,
- test data (sensors data files).

List of Figures

List of Tables