Lab 1: Activity Recognition

Learning outcome: Install the Anaconda environment and perform activity recognition based on the HAPT dataset.

# Introduction

## Lab overview

## Welcome to the first laboratory of this course. In this lab, we will build machine learning models with Anaconda and Python. We will use three different machine learning methods: Linear Support Vector Classifier, Support Vector Classifier, and Decision Tree. With the methods, we will implement activity recognition models which take accelerometer data as input and predict the type of activity based on the accelerometer data.

# Requirements

## Software requirements

You will need a PC running Windows 10 and an internet connection.

**The HAPT Dataset**

We use a dataset of 3-axial accelerometer signals from an academic experiment on the UC Irvine Machine Learning Repository. Download the dataset by accessing the following link: [Dataset Link](https://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions). Move the archive file to the lab1 folder and extract the file.

# Getting Set-up

Anaconda is a distribution of Python language for data science and machine learning. With Anaconda, you can easily install open-source machine learning packages. First, we will introduce how to install Anaconda on your machine.

## Installation:

1. Visit the official Anaconda webpage: *anaconda.com*

2. On products, click Anaconda Distribution menu.

3. You can download the Anaconda Installer by clicking this button.

4. Open the Anaconda installer and install using default options.

5. Now, the installation is finished.

## How to use the Anaconda Prompt:

1. Open Anaconda Prompt

2. On Anaconda Prompt, you can execute *conda* commands.

3. For example, you can create an environment by typing:

conda create -n ml\_lab python=3.8

4. Then you can activate the new environment by typing:

conda activate ml\_lab

5. Now, you can see that the current environment name changed as [ml\_lab].

6. For the new environment, you can install conda packages using

conda install

7. Then you can deactivate the environment by typing:

conda deactivate

## How to open Jupyter Notebook:

1. Unzip the lab1.zip archive file with File Explorer

2. Open Anaconda Prompt

3. Activate the conda environment by typing:

conda activate ml\_lab

4. Navigate to the folder where lab1.ipynb is located.

cd “path to the folder”

5. Open lab1.ipynb by typing:

jupyter notebook lab1.ipynb

# Application Code

In this lab exercise you will learn how to utilize Anaconda and Python to work with datasets and use machine learning. Second, you will learn how to utilize machine learning API to extract features from the data and implement machine learning models. Third, you will learn how to implement activity recognition models with accelerometer data.

In Jupyter Notebook, you can execute a code block by pressing [SHIFT] + [ENTER]. The first code block imports necessary python packages. Execute the first block.

|  |
| --- |
| **from** \_\_future\_\_ **import** division  **import** pandas **as** pd  **import** numpy **as** np  **import** matplotlib.pyplot **as** plt  %matplotlib inline  **import** random  random.seed(7)  # display pandas results to 3 decimal points, not in scientific notation  pd.set\_option('display.float\_format', **lambda** x: '%.3f' % x) |

The second code block loads the information about the dataset such as feature names and activity names. Execute the second block. Then you can check the number of features, the number of activities, and the list of activities.

|  |
| --- |
| **with** open('./HAPT Data Set/features.txt') **as** f:  features = f.read().split()  print('There are {} features.'.format(len(features)))    **with** open('./HAPT Data Set/activity\_labels.txt') **as** f:  activity\_labels = f.readlines()  activity\_df = [x.split() **for** x **in** activity\_labels]  **print**('There are {} activities.'.format(len(activity\_df)))  pd.DataFrame(activity\_df, columns = ['Activity\_id', 'Activity\_label']) |

The next three code blocks load the train and test datasets to be used for machine learning.

|  |
| --- |
| X\_train = pd.read\_table('./HAPT Data Set/Train/X\_train.txt', header=None, sep=" ")  X\_train.columns = features  X\_train.iloc[:10, :10].head() |

|  |
| --- |
| y\_train = pd.read\_table('./HAPT Data Set/Train/y\_train.txt', header = None, sep = " ", names = ['Activity\_id'])  y\_train |

|  |
| --- |
| X\_test = pd.read\_table('./HAPT Data Set/Test/X\_test.txt', header = None, sep = " ")  X\_test.columns = features  y\_test = pd.read\_table('./HAPT Data Set/Test/y\_test.txt', header = None, sep = " ", names = ['Activity\_id']) |

### Activity Recognition with Linear SVC (Full Features)

We've prepared the dataset for training and testing, so we can start building a machine learning model for activity recognition. In this lab, we are going to use a machine learning Python package, scikit-learn.

Since the test set must not be used until the very last step of testing, we are going to split the training set into training and validation sets. Also, we are going to use 5-fold cross-validation, which means that we are going to partition the training set into 5 sub-datasets and use 4 sub-datasets for training and 1 sub-dataset for validation. Let's start with Linear Support Vector Classification.

First, we declare a set of hyper-parameter values we are going to try. For Linear SVC, we need to specify the regularization parameter C, which indicates the amount of regularization. Since we don't know what is the best value for C yet, we are going to try a set of values and pick the best one based on the results. Next, we declare the classifier. This *random\_state* argument is used for shuffling the data.

Then, we train and validate the classifier. Note that we set the cross validation parameter as 5 because we use 5-fold cross-validation. Execute the code block.

|  |
| --- |
| **from** sklearn.svm **import** LinearSVC  **from** sklearn.model\_selection **import** validation\_curve  # Declare the hyper-parameter  # - C: The amount of regularization  C\_params = np.logspace(-6, 3, 10)  # Declare the classfier  clf\_svc = LinearSVC(random\_state = 7)  # Compute training and test scores for varying parameter values  train\_scores, val\_scores = validation\_curve(  clf\_svc, X\_train.values, y\_train.values.flatten(),  param\_name = "C", param\_range = C\_params,  cv = 5, scoring = "accuracy", n\_jobs = -1) |

The execution is now finished. The training and validation scores are saved in *train\_scores* and *val\_scores*. We are going to plot the training and validation accuracy with different parameter values. Execute the code block. The figure shows that the best parameter value is around 0.1 because the model obtains the best accuracy with the value.

Shape

Description automatically generated with low confidence

The mean validation score for the best parameter is 0.9331.

### Activity Recognition with Linear SVC (40 Features)

Until now, we used all the features that the dataset provides. But we are going to reduce the number of features and see how the accuracy changes with a smaller number of features. This is the list of the features that the dataset provides.

|  |
| --- |
| [  'tBodyAcc-Mean-1',  'tBodyAcc-Mean-2',  'tBodyAcc-Mean-3',  'tBodyAcc-STD-1',  'tBodyAcc-STD-2',  'tBodyAcc-STD-3',  …  ] |

We are going to use only the first 40 features from the dataset. We declare new train and test datasets to have 40 features. Then we train and validate the classifier with the new dataset. Execute the code block.

|  |
| --- |
| X\_train\_new = X\_train.iloc[:,0:40]  X\_test\_new = X\_test.iloc[:,0:40]  # Compute training and test scores for varying parameter values  train\_scores, val\_scores = validation\_curve(  clf\_svc, X\_train\_new.values, y\_train.values.flatten(),  param\_name = "C", param\_range = C\_params,  cv = 5, scoring = "accuracy", n\_jobs = -1)  plot\_accuracy(train\_scores, val\_scores, C\_params) |

In this figure, you can see the training and validation accuracy drop with the smaller number of features.

Chart, line chart

Description automatically generated

### Activity Recognition with Linear SVC (Feature Selector)

Rather than using the first 40 features, we are going to use FutureSelection API from scikit-learn which helps choose the best K features from the dataset.

We first declare the feature selector which picks only the 20 best features from the train dataset. Based on the feature selector, we define new train and test sets.

|  |
| --- |
| **from** sklearn.feature\_selection **import** SelectKBest  feature\_selector = SelectKBest(k=20).fit(X\_train.values, y\_train.values.flatten())  X\_train\_new = feature\_selector.transform(X\_train.values)  X\_test\_new = feature\_selector.transform(X\_test.values)  # We use .values because X\_train is a panda dataset  # The output of .transform is an array, therefore we don't need to use .values anymore in the validation\_curve |

We train and validate the classifier with the 20 best features. Now you can see that this new set of features leads to the better training and validation accuracy.

|  |
| --- |
| # Compute training and test scores for varying parameter values  train\_scores, val\_scores = validation\_curve(  clf\_svc, X\_train\_new, y\_train.values.flatten(),  param\_name = "C", param\_range = C\_params,  cv = 5, scoring = "accuracy", n\_jobs = -1)  plot\_accuracy(train\_scores, val\_scores, C\_params) |

Chart, line chart

Description automatically generated

### Activity Recognition with General SVC

Although the performance of Linear SVC is already good, we are going to try other machine learning algorithms to compare with each other.

Let's try the general Support Vector Classifier with GridSearchCV, which helps find the best parameter values. We first declare a set of parameters we are going to try. For SVC, we can choose which type of kernel function to use. For example, we can use radial basis kernel (rbf) or polynomial kernel (poly). And similar to Linear SVC, we can specify the regularization parameter C. For the polynomial kernel, we can additionally specify its degree and coefficient. Then, we declare SVC and GridSearchCV. Execute the code blocks.

|  |
| --- |
| # import the functions SVC and GridSearchCV  **from** sklearn.svm **import** SVC  **from** sklearn.model\_selection **import** GridSearchCV  # Declare the parameters  # - Pay attention to the number of paramters you declare.  # - More points in the GridSearch will cost you longer training time.  # List of parameters  # - kernel: Type of the kernel function  # {‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’}  # - C: The amount of regularization  param\_C = np.logspace(-6, 3, 5)  parameters = [{'kernel': ['rbf'], 'C': param\_C},  {'kernel': ['poly'], 'C': param\_C, 'degree': [2,3,4]}]  # Declare the classifier (estimator) to be used in GridSearchCV  clf\_svc = SVC(random\_state = 7)  # Declare the classifier using GridSearchCV  clf\_GSCV = GridSearchCV(estimator = clf\_svc, param\_grid = parameters, cv = 5, n\_jobs = -1)  # Fit the classifier and find the best parameters using GridSearchCV  clf\_GSCV.fit(X\_train.values, y\_train.values.flatten()) |

Let's check the best parameter values. The best set of parameters uses the rbf kernel and uses 1000 for C.

Activity Recognition with Decision Tree

We've learned how to train, validate, and test machine learning models. Let's try the last machine learning model, decision tree. We import and declare the classifier. We train and validate the decision tree classifier with the *validation\_curve* function.

|  |
| --- |
| # Now try other classifiers, for example Decision Trees, Nearest Neighbors, or Ensemble methods. Can you get better results?  # After identifying the best model with the optimal set of parameters, test your model using the test set.  # What's your test accuracy? Are you satisfied? How complex is your model to be deployed on a microcontroller?  # A possible solution:  # Import the classifier  **from** sklearn.tree **import** DecisionTreeClassifier  # Declare the classifier  classifier = DecisionTreeClassifier(random\_state=7)  # Declare the parameter  # - min\_samples\_split: The minimum number of samples to split an internal node  min\_samp\_split = np.arange(2,10)  # Let's use validation curve  train\_scores, val\_scores = validation\_curve(  classifier, X\_train.values, y\_train.values.flatten(),  param\_name = "min\_samples\_split", param\_range = min\_samp\_split,  cv = 5, scoring = "accuracy", n\_jobs = -1)  plot\_accuracy(train\_scores, val\_scores, min\_samp\_split) |

You can see the decision tree model obtains the lower accuracy than SVC. Based on these results, Linear SVC is the best machine learning model for activity recognition among the three machine learning models we've covered today.

A picture containing graphical user interface

Description automatically generated