Appendix A: Technical documentation

The documentation contains a description of Jupyter Notebooks, Python package `vibrodiagnostics` with utility functions for machine learning, and data logger firmware. Pages of LaTeX documentation for Python were autogenerated by documentation tool **Sphinx**. Pages from C source code were created by **Doxygen**.

A.1 Jupyter notebooks

- **cluster-dbscan.ipynb** - The clustering feasibility of the DBSCAN algorithm in the MaFaulDa dataset. Experiments to find the best epsilon parameter for complete sets of time-domain and frequency-domain features. Epsilon is estimated by localizing the knee on the curve of nearest neighbours. The method turned out to be unreliable in separating different groups of fault types. The clustering produced too many or too few samples based on the choice of minimal samples.
- **eda-bearing-faults.ipynb** - Mark the bearing characteristic frequencies in the frequency spectrum in the axis of motion for chosen recordings from the MaFaulDa dataset, scroll compressors, water pumps, and electric motors. The bearing frequencies are calculated from coefficients or physical dimensions for the bearing designation.
- **eda-files.ipynb** - Time-domain vibration signal waveforms are charted for acceleration values and velocity after integration. Measurements are compared between fault types or measurement positions. Statistical tests for normality and stationarity of oscillations are conducted. Comparisons from all axes for peak finding MMS algorithm, cumulative frequency spectrum, and time-frequency spectrum. Orbitals of detected fundamental frequency are shown in the spatial graph. Options are to view signals for chosen files from the MaFaulDa dataset on the inner bearing, outer bearing, or for the Pumps

industrial dataset.

- **eda-ksb-cloud.ipynb** - Overall vibration rms velocities in hourly intervals for two identical water pumps are shown in ISO severity levels. Total running time is summed from periods when each pump has been operational. Frequency spectra are also compared but are proven to have too low resolution for any conclusive result in fault recognition.
- **eda-pumps.ipynb** - It visualizes amplitude histograms and frequency spectrum of every time series in the Pump dataset next to one another. Spectrograms of selected situations are depicted, such as rotation speed up or slow down. This notebook is instrumental in spotting patterns between different places and dates and checking for signal clipping.
- **eda-standing-fan.ipynb** - Estimate the rotational speed of the standing fan based on the audio recording and compare it to the estimate from vibrations. The accelerometer is attached to the back, side, and front of the shaft.
- **extraction-mafaulda.ipynb** - Feature extraction of a complete set of features from the entire MaFaulDa in time and frequency domain (TD and FD sets) to CSV files. Enable variable *EXTRACT* to proceed with a lengthy calculation that takes around ten minutes for each set. When the extraction flag is disabled, the notebook loads the already extracted files with attributes.
- **extraction-pumps.ipynb** - Feature extraction of a complete set of features from Pump dataset in time and frequency domain (TD and FD sets) to CSV files. Enable variable *EXTRACT* to proceed with calculation.
- **feature-summary.ipynb** - Count the occurrences of predictors in the best triplets over setup experimental conditions. Feature selection techniques produce bar charts with different orderings. The explained variance of complete feature sets is shown alongside the separability of clusters measured by silhouette score. This notebook generates CSV files of the best feature sets used in *knn-batch.ipynb*.
- **iit-src-paper.ipynb** - Run the experiments described in IIT.SRC 2024 paper in Appendix C. Set variables *USE_ONE_AXIS* and *MAFAULDA_ LABEL_METHOD* to one of the allowed options to generate all possible results. Running the experiments and not reading precomputing results requires en-

abling *GENERATE* variable. Additionally, the hyperparameters of the model can be changed to tune the accuracy.

- **knn-batch.ipynb** - Train k-NN classifiers with k = 5 on three members' best subsets of features found in *feature-summary.ipynb*. Evaluate a wide range of performance metrics. Plot relation of k-neighbours on error rate for subset picked by the rank product method.
- **knn-mafaulda.ipynb** - The results of the experiments are described in the Exploratory analysis of MaFaulDa and the evaluation chapter. The setup of experimental scenarios for batch and incremental learning is followed by scales of features and their correlations. Every combination of parameters put into k-NN is captured in graphs of accuracy. Feature selection methods are compared under experimental conditions. Incremental learning on subset combinations is carried out in the gradual evolution of relative fault severities.
- **knn-online.ipynb** - Simulate incremental learning process for the complete feature sets. The source domain for features is chosen in variable *DOMAIN*. The progressive valuation of k-NN accuracy compares situations with varied sliding windows and gaps between annotated observations. The performance at the end of the training is written in tables.
- **wavelets.ipynb** - Features computed from wavelet packet coefficients are chosen and sorted using feature selection scores. The effect of varying the depth of decomposition is viewed in stacked plots. The difference in scores between feature coefficients is plotted in bar charts.

A.2 Data analysis package

CHAPTER

ONE

VIBRODIAGNOSTICS PACKAGE

1.1 extraction module

`extraction.detrending_filter(`*dataframes*, *columns*`)`

Subtract average from each value in columns of multiple data frames

Parameters

- `dataframes` (*List[DataFrame]*) – list of data frames
- `columns` (*List[str]*) – attributes from which mean is removed

Returns
modified data frames with mean removed

Return type
List[*DataFrame*]

`extraction.downsample(`*x*, *k*, *fs_reduced*, *fs*`)`

Downsample the time series by a factor

Parameters

- `x` (*array*) – time series
- `k` (*int*) – factor for downsampling (when it is None, the factor is calculated as a ratio of sampling rates)
- `fs_reduced` (*int*) – desired sampling frequency
- `fs` (*int*) – original sampling frequency of the time series

Returns
downsampled time series

Return type
array

`extraction.energy(`*x*`)`

Calculate energy of the signal

Parameters
`x` (*array*) – input signal

Returns
energy of the signal

Return type
float

`extraction.envelope_signal(`*f*, *pxx*`)`

Approximate envelope of frequency spectrum with quadratic interpolation between peaks

Parameters

- `f` (*array*) – array of frequency bins
- `pxx` (*array*) – array of amplitudes at frequency bins

Returns
envelope of the amplitudes

Return type
array

`extraction.frequency_features_calc(`*df*, *col*, *fs*, *window*`)`

Extract complete set of features in frequency-domain (FD set)

Parameters

- `df` (*DataFrame*) – data frame with input time series in columns
- `col` (*str*) – column in data frame to use for extraction
- `fs` (*int*) – sampling frequency in Hz of the time series
- `window` (*int*) – length of FFT window

Returns
list of frequency-domain features in pairs of feature name with column prefix and its value

Return type
List[*Tuple*[str, *DataFrame*]]

`extraction.fs_list_files(`*root_path*`)`

List csv and tsv files in dataset in the directory

Parameters
`root_path` (*str*) – base path of dataset directory

Returns
list of filenames

Return type
List[str]

`extraction.harmonic_series_detection(`*f*, *pxx*, *fs*, *fft_window*`)`

Find all series of harmonic frequencies in the frequency spectrum according to paper: "Identification of harmonics and sidebands in a finite set of spectral components" (https://hal.science/hal-00845120v2/document)

Parameters

- `f` (*array*) – array of frequency bins
- `pxx` (*array*) – array of amplitudes at frequency bins
- `fs` (*int*) – sampling frequency in Hz
- `fft_window` (*int*) – length of FFT window for calculation of frequency bin resolution

Returns
list of harmonic frequency series with components described by their frequency and amplitude

Return type
List[*List*[*Tuple*[int, float]]]

`extraction.have_intersection(`*interval1*, *interval2*`)`

Check overlap of two numeric intervals

Parameters

- `interval1` (*Tuple[float, float]*) – numbers for bounds of the first interval
- `interval2` (*Tuple[float, float]*) – numbers for bounds of the second interval

Returns
overlap of intervals is present

Return type
bool

`extraction.list_files(`*dataset*`)`

List csv and tsv files in dataset within ZIP archive

Parameters
`dataset` (*ZipFile*) – dataset in zip archive

Returns
list of filenames

Return type
List[str]

`extraction.load_features(`*filename*, *axis*, *label_columns*`)`

Load features from csv file and aggregate values from chosen directions of movement

Parameters

- `filename` (*str*) – File path where extracted features are found
- `axis` (*List[str]*) – Elements to combine one feature from
- `label_columns` (*List[str]*) – Columns that are not features and are not processed just copied

Returns
data frame with columns of unique feature names

Return type
Tuple[*DataFrame*, *DataFrame*]

`extraction.load_files_split(`*dataset*, *func*, *parts=1*, *cores=4*`)`

Load files from the dataset in ZIP archive and process them with callback function

Parameters

- `dataset` (*ZipFile*) – dataset in zip archive
- `func` (*Callable*) – callback for file processing that takes dataset, filename, and number of parts to split file into as parameters
- `parts` (*int*) – number of partitions to split each file into
- `cores` (*int*) – number of cores to use in workload parallelization

Returns
data frame with extracted features and associated annotations for row

Return type
DataFrame

`extraction.mms_peak_finder(`*x*, *win_len=3*`)`

Robust non-parametric peak identification MMS algorithm according to description in paper: "Non-Parametric Local Maxima and Minima Finder with Filtering Techniques for Bioprocess" (https://doi.org/10.4236/jsip.2016.74018)

Parameters

- `x` (*array*) – time series

- `win_len` (*int*) – window length of points compared in peak finding

Returns
array of indexes where peaks are in the time series

Return type
array

`extraction.negentropy(`*x*`)`

Calculate negentropy of the signal

Parameters
`x` (*array*) – input signal

Returns
negentropy of the signal

Return type
float

`extraction.signal_to_noise(`*x*`)`

Calculate estimated signal to noise ratio (noisiness) of the signal. Formula is taken from: https://www.geeksforgeeks.org/signal-to-noise-ratio-formula/

Parameters
`x` (*array*) – input signal

Returns
SNR of the signal

Return type
float

`extraction.spectral_roll_off_frequency(`*f*, *pxx*, *percentage*`)`

Calculate roll-off frequency. Cumulative sum of energy in spectral bins below roll-off frequency is percentage of total energy

Parameters

- `f` (*array*) – array of frequency bins
- `pxx` (*array*) – array of amplitudes at frequency bins
- `percentage` (*float*) – ratio of total energy below the roll-off frequency

Returns
roll-off frequency in Hz

Return type
float

`extraction.spectral_transform(`*x*, *window*, *fs*`)`

Estimate frequency spectrum using Welch's method. Partition the signal with Hann window and 50% overlap.

Parameters

- `x` (*Series*) – input signal in time domain
- `window` (*int*) – length of Hann FFT window
- `fs` (*int*) – sampling frequency in Hz of the input signal

Returns
envelope of the amplitudes

Return type
Tuple[*array*, *array*]

`extraction.split_dataframe(`*dataframe*, *parts=None*`)`

Split to data frames to non overlapping parts

Parameters

- `dataframe` (*DataFrame*) – data frame to be split
- `parts` (*int*) – number of partitions to split data frame into

Returns
list of data frame parts

Return type
List[*DataFrame*]

`extraction.temporal_variation(`*x*, *window*`)`

Calculate temporal variations in successive frequency spectra. It is a inverse correlation of pairs formed from overlapping windows.

Parameters

- `x` (*Series*) – input signal in time domain
- `window` (*int*) – length of Hann FFT window

Returns
array of temporal variations

Return type
List[float]

`extraction.time_features_calc(`*df*, *col*, *fs*, *window*`)`

Extract complete set of features in time-domain (TD set)

Parameters

- `df` (*DataFrame*) – data frame with input time series in columns
- `col` (*str*) – column in data frame to use for extraction
- `fs` (*int*) – sampling frequency in Hz of the time series
- `window` (*int*) – length of FFT window (not used)

Returns
list of time-domain features in pairs of feature name with column prefix and its value

Return type
List[*Tuple*[str, *DataFrame*]]

`extraction.wavelet_features_calc(`*df*, *col*, *fs*, *window*`)`

Extract wavelet coefficients for Meyer wavelet and six levels deep. Each wavelet coefficient produces four features (energy, energy ratio, kurtosis, negentropy)

Parameters

- `df` (*DataFrame*) – data frame with input time series in columns
- `col` (*str*) – column in data frame to use for extraction
- `fs` (*int*) – sampling frequency in Hz of the time series
- `window` (*int*) – length of FFT window (not used)

Returns
list of wavelet-domain features in pairs of feature name with column prefix and its value

Return type
List[*Tuple*[str, *DataFrame*]]

1.2 mafaulda module

`mafaulda.BEARINGS = {'ball_diameter': 0.7145, 'balls': 8, 'bpfi_factor': 5.002, 'bpfo_factor': 2.998, 'bsf_factor': 1.871, 'ftf_factor': 0.375, 'pitch_diameter': 2.8519}`

Coefficients for bearing characteristic frequencies

`mafaulda.FAULTS = {'A': {'horizontal-misalignment': 'misalignment', 'imbalance': 'imbalance', 'normal': 'normal', 'underhang-ball_fault': 'ball fault', 'underhang-cage_fault': 'cage fault', 'underhang-outer_race': 'outer race fault', 'vertical-misalignment': 'misalignment'}, 'B': {'horizontal-misalignment': 'misalignment', 'imbalance': 'imbalance', 'normal': 'normal', 'overhang-ball_fault': 'ball fault', 'overhang-cage_fault': 'cage fault', 'overhang-outer_race': 'outer race fault', 'vertical-misalignment': 'misalignment'}}`

Annotation of fault types by bearing placement

`mafaulda.LABEL_COLUMNS = ['fault', 'severity', 'rpm']`

Metadata columns extracted from file path within dataset

`mafaulda.SAMPLING_RATE = 50000`

Sampling frequency in Hz of the sensors

`mafaulda.assign_labels(`*df*, *bearing*, *keep=False*`)`

Assign labels to fault types for bearing and optionally clean up data frame to contain only annotated rows with faults

Parameters

- `df` (*DataFrame*) – data frame after feature extraction with column "fault"
- `bearing` (*str*) – bearing to determine labels of fault types ("A" or "B")
- `keep` (*bool*) – do not remove metadata columns

Returns
annotated data frame

Return type
DataFrame

`mafaulda.bearing_frequencies(`*rpm*`)`

Calculate bearing characteristic frequencies for MaFaulDa machine simulator

Parameters
`rpm` (*int*) – Rotational speed of the machine

Returns
Bearing defect frequencies

Return type
Dict[str, float]

`mafaulda.clean_columns(`*df*`)`

Remove excessive columns with metadata and drop rows without label

Parameters
`df` (*DataFrame*) – data frame with excess labels

Returns
data frame that consists of columns with features and "label"

Return type
DataFrame

`mafaulda.csv_import(`*dataset*, *filename*`)`

Open a CSV file from MaFaulda zip archive

Parameters

- `dataset` (*`ZipFile`*) – ZIP archive of MaFaulDa dataset
- `filename` (*`str`*) – path to the file within dataset

Returns
data frame of the imported file

Return type
DataFrame

`mafaulda.features_by_domain(`*features_calc*, *dataset*, *filename*, *window=None*, *parts=1*, *multirow=False*`)`

Open a CSV file from MaFaulda zip archive

Parameters

- `features_calc` (*`Callable`*) – callback feature extraction function that has parameters for data frame, column to process in data frame, sampling frequency, window length of segment
- `dataset` (*`ZipFile`*) – ZIP archive of MaFaulDa dataset
- `filename` (*`str`*) – path to the file within dataset
- `window` (*`int`*) – length of window (usually for FFT)
- `parts` (*`int`*) – number of parts the input time series is split into
- `multirow` (*`bool`*) – extracted features are in rows, not in columns

Returns
row(s) of features extracted from the file

Return type
DataFrame

`mafaulda.get_classes(`*df*, *bearing*`)`

Create column "label" in data frame according to chosen bearing

Parameters

- `df` (*`DataFrame`*) – data frame after feature extraction with column "fault"
- `bearing` (*`str`*) – bearing to determine labels of fault types ("A" or "B")

Returns
data frame with "label" column

Return type
DataFrame

`mafaulda.label_severity(`*df*, *bearing*, *level*, *debug=False*, *keep=False*`)`

Relabel faults less than set relative severity level as "normal"

Parameters

- `df` (*`DataFrame`*) – data frame after feature extraction with column "fault"
- `bearing` (*`str`*) – bearing to determine labels of fault types ("A" or "B")
- `debug` (*`bool`*) – print relative severity levels
- `keep` (*`bool`*) – do not remove metadata columns
- `level` (*`float`*)

Returns
data frame with relabeled observations

Return type
DataFrame

`mafaulda.load_source(`*domain*, *row*, *train_size=0.8*`)`

Load features according to domain and split the observations into training and testing set

Parameters

- `domain` (*str*) – complete feature set ("TD", "FD")
- `row` (*dict*) – parameters for data filtering, e.g.: {"placement": "A", online: False}
- `train_size` (*float*) – ratio of training set to testing set

Returns
X_train, X_test, Y_train, Y_test

Return type
tuple

`mafaulda.lowpass_filter(`*data*, *cutoff=10000*, *fs=50000*, *order=5*`)`

Low-pass filter of n-th order the input signal at the cutoff frequency

Parameters

- `data` (*Series*) – input signal
- `cutoff` (*int*) – cutoff frequency
- `fs` (*int*) – sampling frequency in Hz of the input signal
- `order` (*int*) – steps of the filter

Returns
output signal after filtering

Return type
Series

`mafaulda.lowpass_filter_extract(`*dataframes*, *columns*`)`

Apply low-pass filter to columns in multiple data frames

Parameters

- `frames` (*data*) – list of input datafrmaes to which filter is applied to
- `columns` (*List[str]*) – columns that filter is applied to
- `dataframes` (*List[DataFrame]*)

Returns
list of data frames after filtering

Return type
List[*DataFrame*]

`mafaulda.mark_severity(`*df*, *bearing*, *debug=False*`)`

Calculate relative severity levels for data frame with original metadata columns

Parameters

- `df` (*DataFrame*) – data frame after feature extraction with column "fault" and "severity"
- `bearing` (*str*) – bearing to determine labels of fault types ("A" or "B")
- `debug` (*bool*) – print relative severity levels

Returns
data frame with columns for absolute and relative fault severity levels

Return type
DataFrame

`mafaulda.parse_filename(`*filename*`)`

Split path of file within dataset structure to label the time series

Parameters
`filename` (*str*) – path to file inside of zip archive

Returns
fault type, severity conditions, and file number

Return type
Tuple[str, str, str]

`mafaulda.rpm_calc(`*tachometer*`)`

Extract rotational speed in rpm units from tachometer pulse signal

Parameters
`tachometer` (*Series*) – tachometer signal

Returns
rotational speed in rpm

Return type
float

1.3 models module

`models.accuracies_to_table(`*domain*, *set*, *distribution*, *accuracy*`)`

Format accuracy to a row with metadata about hyperparamater and compute percentiles in the model accuracy distribution

Parameters

- `domain` (*str*) – source domain from which the features are extracted ("TD" or "FD")
- `set` (*str*) – Title for the feature set or selection method
- `distribution` (*DataFrame*) – accuracy distribution of the model
- `accuracy` (*DataFrame*) – accuracy of the model in training and testing set

Returns
formatted structure for row of accuracies and percentiles

Return type
dict

`models.all_features(`*X*, *Y*, *model_name='knn'*, *power_transform=False*, *k_neighbors=[1, 5, 9, 13, 17, 21, 25, 29, 33, 37]*, *kfolds=5*`)`

Evaluate complete feature sets in k-nearest neighbours classifier
with various k-value parameter

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*Series*) – column of labels

- `model_name` (*str*) – name of the machine learning model to evaluate. Options are: "knn", "lda", "bayes", "svm"
- `power_transform` (*bool*) – apply power transform of features in preprocessing instead of normalization
- `k_neighbors` (*list*) – number of neighbours for k-nearest neighbours model
- `kfolds` (*int*) – number of splits for k-fold cross-validation

Returns
training and testing accuracy for each value of k-neighbours

Return type
Dict[str, float]

`models.enumerate_models(`*X*, *Y*, *domain*, *k_neighbors=(3, 5, 11, 15)*, *num_of_features=(2, 3, 4, 5)*, *kfolds=5*, *power_transform=False*, *model='knn'*`)`

Grid search of parameters k-nearest neighbours classifier with feature subset combinations

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*DataFrame*) – column of labels
- `domain` (*str*) – source domain from which the features are extracted ("TD" or "FD")
- `k_neighbors` (*Tuple[int]*) – neighbours for k-nearest neighbours model to search in
- `num_of_features` (*Tuple[int]*) – number of features in the subset to search in
- `kfolds` (*int*) – number of splits for k-fold cross-validation
- `power_transform` (*bool*) – apply power transform of features in preprocessing instead of normalization
- `model` – name of the machine learning model to evaluate. Options are: "knn", "lda", "bayes", "svm"

Returns
training and testing accuracy for each hyperparameter and feature set combination

Return type
DataFrame

`models.feature_combinations(`*X*, *Y*, *k_neighbors*, *num_of_features*, *kfolds*, *domain*, *model*, *power_transform=False*`)`

Evaluate all combinations of feature subsets of given size out of complete sets

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*DataFrame*) – column of labels
- `k_neighbors` (*int*) – number of neighbours for k-nearest neighbours model
- `num_of_features` (*int*) – number of features in the subset
- `kfolds` (*int*) – number of splits for k-fold cross-validation
- `domain` (*str*) – source domain from which the features are extracted
- `model` (*str*) – name of the machine learning model to evaluate. Options are: "knn", "lda", "bayes", "svm"

- `power_transform` (*bool*) – apply power transform of features in preprocessing instead of normalization

Returns
training and testing accuracy for each combination of features

Return type
List[dict]

`models.feature_selection_accuracies(`*X*, *Y*, *domain*, *models_summary*, *k_neighbors*, *number_of_features*, *power_transform=False*`)`

Apply feature selection methods and evaluate accuracies for chosen number of neighbours and number of features

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*DataFrame*) – column of labels
- `domain` (*str*) – source domain from which the features are extracted ("TD" or "FD")
- `k_neighbors` (*int*) – neighbours for k-nearest neighbours model
- `number_of_features` (*int*) – number of features in the subset
- `power_transform` (*bool*) – apply power transform of features in preprocessing instead of normalization
- `models_summary` (*DataFrame*)

Params model_summary
accuracies from all feature subset combinations

Returns
accuracies and percentiles for all tested feature selection methods

Return type
List[*Dict*[str, int]]

`models.find_best_subset(`*X*, *Y*, *metric*, *members*, *kfolds=5*`)`

Find the best subset of features based on supplied feature selection metric name

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*Series*) – column of labels
- `metric` (*str*) – name of the bivariate similarity metric to compute for each feature and label
- `members` (*int*) – number of features in the subset
- `kfolds` (*int*) – number of splits for k-fold cross validation

Returns
list of the best features according to feature selection metric

Return type
List[str]

`models.kfold_accuracy(`*X*, *Y*, *k_neighbors*, *kfolds*, *model_name='knn'*, *power_transform=True*, *knn_metric='euclidean'*`)`

Evaluate classifier accuracy in k-fold validation on a data frame of features after oversampling to the majority class

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*Series*) – column of labels
- `k_neighbors` (*int*) – number of neighbours for k-nearest neighbours model
- `model_name` (*str*) – name of the machine learning model to evaluate. Options are: "knn", "lda", "bayes", "svm"
- `kfolds` (*int*) – number of splits for k-fold cross-validation
- `power_transform` (*bool*) – apply power transform of features in preprocessing instead of normalization
- `knn_metric` – distance metric name for k-nearest neighbours model

Returns
average accuracy of model over k-folds in training and testing sets

Return type
Dict[str, float]

`models.knn_online_learn(`*X*, *Y*, *window_len=1*, *learn_skip=0*, *clusters=False*, *n_neighbors=5*`)`

Progressive valuation of k-nearest neighbours classifier trained with increamental learning

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*DataFrame*) – column of labels
- `window_len` (*int*) – Length of the tumbling window
- `learn_skip` (*int*) – Gap of labeled observations in amount of samples
- `clusters` (*int*) – return data points instead of valuation
- `n_neighbors` (*int*) – number of neighbours for k-nearest neighbours model

Returns
performance of the model in progressive valuation over all generations

Return type
DataFrame

`models.model_boundaries(`*X*, *Y*, *n=5*, *model_name='knn'*, *knn_metric='euclidean'*`)`

Train k-nearest neighbours classifier to be used in determining its decision boundaries

Parameters

- `X` (*DataFrame*) – data frame of predictor features
- `Y` (*DataFrame*) – column of labels
- `n` (*int*) – number of neighbours for k-nearest neighbours model
- `model_name` (*str*) – name of the machine learning model to evaluate. Options are: "knn", "lda", "bayes", "svm"
- `knn_metric` (*str*) – distance metric name for k-nearest neighbours model

Returns
model fitted with training data of 80% from the original dataset

`models.transform_to_pca(`*X*, *n*`)`

Transform features to their principal components after normalization

Parameters

- `dataset` – data frame with columns only for predictors
- `n` (*int*) – number of principal components

- `X` (*DataFrame*)

Returns
data frames with rows of features replaced for principal components

Return type
DataFrame

1.4 pumps module

`pumps.LABEL_COLUMNS = ['date', 'device', 'position']`
Metadata columns extracted from file path within dataset

`pumps.SAMPLING_RATE = 26866`
Sampling frequency in Hz of the sensors

`pumps.assign_labels(df)`
Assign labels to fault types into "label" column based on measurement placement

Parameters
`df` (*DataFrame*) – data frame after feature extraction with column "fault"

Returns
annotated data frame

Return type
DataFrame

`pumps.beaglebone_measurement(filename, fs)`
Import csv file recorded on BeagleBone Black with accelerometer ADXL335

Parameters

- `filename` (*str*) – file name of the recording
- `fs` (*int*) – sampling frequency in Hz

Returns
data frame of the recording

Return type
Tuple[str, *DataFrame*]

`pumps.csv_import(dataset, filename)`
Open a CSV file from Pump zip archive

Parameters

- `dataset` (*ZipFile*) – ZIP archive of Pump industrial dataset
- `filename` (*str*) – path to the file within dataset

Returns
data frame of the imported file

Return type
DataFrame

`pumps.features_by_domain(features_ calc, dataset, filename, window=None, parts=None)`
Open a CSV file from Pump zip archive

Parameters

- `features_calc` (*Callable*) – callback feature extraction function that has parameters for data frame, column to process in data frame, sampling frequency, window length of segment

- `dataset` (*ZipFile*) – ZIP archive of Pump dataset
- `filename` (*str*) – path to the file within dataset
- `window` (*int*) – length of window (usually for FFT)
- `parts` (*int*) – number of parts the input time series is split into

Returns
row(s) of features extracted from the file

Return type
DataFrame

`pumps.features_by_domain_no_metadata(`*features_ calc*, *filename*, *window=None*, *parts=None*`)`

Parameters

- `features_calc` (*Callable*)
- `filename` (*str*)
- `window` (*int*)
- `parts` (*int*)

Return type
DataFrame

`pumps.get_classes(`*df*, *labels*, *keep=False*`)`

Assign labels to fault types into "label" column and optionally clean up data frame to contain only annotated rows with faults

Parameters

- `df` (*DataFrame*) – data frame after feature extraction with column "fault"
- `labels` (*Dict[str, dict]*) – Llbels to assign machine and measurement placement
- `keep` (*bool*) – do not remove metadata columns

Returns
annotated data frame

Return type
DataFrame

1.5 ranking module

`class ranking.ExperimentOutput(`*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

Bases: `Enum`

Types of experimental scenarios for feature selection techniques

`BEST_CORR = 7`

`BEST_F_STAT = 8`

`BEST_MI = 9`

`BEST_SET = 2`

`COUNTS = 1`

`PCA = 5`

`RANKS = 3`

`SCORES_RANGE = 4`

`SILHOUETTE = 6`

`ranking.batch_feature_ranking(`*X*, *Y*, *mode='rank'*`)`

Order features based on their importance

Parameters

- `X` (*DataFrame*) – data frame that contains features
- `Y` (*Series*) – labels for observations
- `mode` (*str*) – feature selection method, options: “corr”, “f_stat”, “mi”, “rank”

Returns
Sorted features with their scores

Return type
DataFrame

`ranking.best_columns(`*ranks*, *corr*, *n*`)`

Retain the best features that does not belong to correlated set

Parameters

- `ranks` (*DataFrame*) – features with their scores
- `corr` (*Set[Tuple[str, str]]*) – set of pairs with high correlations
- `n` (*int*) – number of best features to keep

Returns
list of best features

Return type
List[str]

`ranking.best_subset(`*ranks*, *corr*, *n*`)`

Retain the best features

Parameters

- `ranks` (*DataFrame*) – features with their scores
- `corr` (*Set[Tuple[str, str]]*) – set of pairs with high correlations
- `n` (*int*) – number of best features to keep

Returns
best features

Return type
DataFrame

`ranking.compute_correlations(`*X*, *corr_above*`)`

Find pairs of features correlated more than the threshold

Parameters

- `X` (*DataFrame*) – data frame that contains features
- `corr_above` (*float*) – correlation threshold level

Returns
pairs or correlated features

Return type
Set[*Tuple*[str, str]]

`ranking.online_feature_ranking(`*X*, *Y*, *mode='rank'*`)`

Sort features by gradual process

Parameters

- `X` (*DataFrame*) – data frame with sequence of events
- `Y` (*Series*) – labels for observations
- `mode` (*str*) – feature selection method, options: “corr”, “f_stat”, “mi”, “rank”

Returns
leaderboard of the features

Return type
DataFrame

`ranking.pca_explained_variances(`*X_train*, *pc*`)`

Explained variances of the principal components

Parameters

- `X_train` (*DataFrame*) – data points of the training set
- `pc` (*int*) – number of principal components

Returns
dictionary of principal components and explained variances

Return type
Dict[str, float]

`ranking.silhouette_scores(`*X_train*, *X_test*, *Y_train*, *Y_test*, *best_features*, *pc*`)`

Calculate silhouette score of data points after normalization of training and testing set with and without the principal components analysis

Parameters

- `X_train` (*DataFrame*) – data points of the training set
- `X_test` (*DataFrame*) – data points of the testing set
- `Y_train` (*DataFrame*) – labels for the training set
- `Y_test` (*DataFrame*) – labels for the testing set
- `best_features` (*List[str]*) – list of chosen feature names
- `pc` (*int*) – number of principal components

Returns
silhouette scores for data points

Return type
Dict[str, float]

1.6 selection module

`class selection.Correlation`

Bases: `Bivariate`

Online correlation to classes as dichotomous variables

`get()`

Return the current value of the statistic.

`update(`x, y`)`

Update and return the called instance.

`class selection.FisherScore`

Bases: `Bivariate`

Online F statistic

`get()`

Return the current value of the statistic.

`update(`x, y`)`

Update and return the called instance.

`class selection.MutualInformation`

Bases: `Bivariate`

Online Mutual information to binned labels

`get()`

Return the current value of the statistic.

`update(`x, y`)`

Update and return the called instance.

`selection.corr_classif(`X, y`)`

Calculate point-biserial correlations to features

Parameters

- `X` (*array*) – matrix of features
- `y` (*array*) – labels of observations

Returns
list of absolute value of correlations between classes and features

Return type
array

1.7 visualize module

`visualize.DOMAIN_TITLES = {'FD': 'Frequency domain', 'TD': 'Time domain', 'TD+FD': 'Time and Frequency domain'}`

Titles for signal source domain abbreviation

`visualize.boxplot_enumerate_models_accuracy(`*results*, *metric*, *plots_col*, *inplot_col*`)`

Boxplot of model accuracy distributions for various
number of features or neighbours

Parameters

- `results` (*DataFrame*) – model accuracy distributions
- `metric` (*str*) – column of values to show in values for accuracy
- `plots_col` (*str*) – constant parameter for subplot ("f" or "k")
- `inplot_col` (*str*) – comparison of different values for the parameter within subplot ("f" or "k")

`visualize.cross_cuts_3d_cluster(`*X_train*, *cluster*, *title*`)`

Scatter plot of clusters in 3D feature space shown in planar cross-sections through coordinate axes

Parameters

- `X_train` (*DataFrame*) – data frame of features
- `cluster` (*str*) – clusters that observations belong to
- `title` (*str*) – figure title

`visualize.evolution_of_severity_levels(`*df*`)`

Line chart of the amount of observations at relative severity levels

Parameters

`df` (*DataFrame*) – data frame with sorted "severity" level column

`visualize.loading_plot(`*loadings*, *feature_names*, *bottom*, *top*`)`

Loading plot of features

Parameters

- `loadings` (*list*) – Relation of features to coordinates that are created by two principal components
- `feature_names` (*List[str]*) – list of feature names corresponding to their loadings
- `bottom` (*float*) – lower limit of graph coordinates in x and y axes
- `top` (*float*) – upper limit of graph coordinates in x and y axes

`visualize.plot_all_knn(`*td_results*, *fd_results*`)`

Line chart of relationship of k-value to k-NN classifier accuracy

Parameters

- `td_results` (*Dict[str, float]*) – lists of k-values and accuracies for time-domain features in training and testing set
- `fd_results` (*Dict[str, float]*) – lists of k-values and accuracies for frquency-domain features in training and testing set

`visualize.plot_cumulative_explained_variance(`*td_variance*, *fd_variance*`)`

Line chart of relationship of number of principal components to total explained variance

Parameters

- `td_variance` (*array*) – Explained variances for time-domain features
- `fd_variance` (*array*) – Explained variances for frequency-domain features

`visualize.plot_label_occurences(`*y*`)`

Line chart of counters for classes in incremental learning

Parameters

`y` (*Series*) – sorted labels of observations

A.3 Data logger firmware

1 Topic Documentation

1.1 Accelerometer Data logger

Data flow control from the sensor to the memory card.

Topics

- Accelerometer

  Configuration of the accelerometer sensor.
- Memory card

  Filesystem operations of the SD card.
- Button

  Configuration of the recording button.
- LED

  Configuration of the indicator LED light.

Macros

- #define **MAX_FILENAME** 256

  Maximum length of the file name buffer.
- #define **MOUNT_POINT** "/sd"

  Directory in virtual file system (VFS) where microSD card gets mounted.
- #define **LOG_FOLDER** MOUNT_POINT"/"

  Path prefix of the directory where recordings are saved.
- #define **NO_WAIT** 10 / portTICK_PERIOD_MS

  Minimal possible delay in milliseconds for using a synchronization primitive.
- #define **SWITCH_DEBOUNCE** 2000 / portTICK_PERIOD_MS

  Period in milliseconds for which the button is disabled after the press to prevent the bouncing effect.
- #define **CARD_CLK_PIN** 14

  SD/MMC bus GPIO pin for CLK.
- #define **CARD_CMD_PIN** 15

  SD/MMC bus GPIO pin for CMD.
- #define **CARD_D0_PIN** 2

  SD/MMC bus GPIO pin for D0.
- #define **RECORD_SWITCH_PIN** 34

  GPIO pin for a button that starts and stops recording.
- #define **RECORD_LED_PIN** 32

  GPIO pin for the indicator LED.
- #define **SENSOR_MISO** 13

  GPIO pin for accelerometer SPI Master In Slave Out.
- #define **SENSOR_MOSI** 16

  GPIO pin for accelerometer SPI Master Out Slave In.
- #define **SENSOR_CLK** 4

  GPIO pin for accelerometer SPI Clock.
- #define **SENSOR_CS** 5

  GPIO pin for accelerometer SPI Chip Select.
- #define **SENSOR_INT1** 33

  GPIO pin for accelerometer interrupt pin.

- #define **SPI_BUS_FREQUENCY** SPI_MASTER_FREQ_8M

  SPI master bus frequency.
- #define **FIFO_LENGTH** 512

  Length of accelerometer FIFO buffer.
- #define **FIFO_WATERMARK** FIFO_LENGTH / 2

  Half length of accelerometer FIFO buffer.
- #define **NUM_OF_FIELDS** 4

  Number of columns per acceleration vector.
- #define **SENSOR_SPI_LENGTH** NUM_OF_FIELDS ∗ FIFO_LENGTH

  Length of buffer for SPI transaction.
- #define **QUEUE_LENGTH** 16

  Number of FIFO buffers that can be pushed to Queue before file write.

Functions

- void panic (int delay)

  Signal fatal error of system indiacted by blinking LED and halting execution.

1.1.1 Detailed Description

Data flow control from the sensor to the memory card.

1.1.2 Function Documentation

panic()

```
void panic (
            int delay )
```

Signal fatal error of system indiacted by blinking LED and halting execution.

Parameters

in	*delay*	Interval in milliseconds for LED blink

1.1.3 Accelerometer

Configuration of the accelerometer sensor.

Data Structures

- struct Acceleration

  Unprocessed data packet for storing samples from accelerometer.

Macros

- #define **SAMPLE_RATE** 9000

  Interval of the periodic timer in milliseconds that reads out circa half of accelerometer FIFO.
- #define **SPI_BUS** SPI3_HOST

  Hardware bus for accelerometer SPI interface.
- #define **AUTO_TURN_OFF_US** 60000000

  Duration of recoding in microseconds (60 s)
- #define **ACC_RESOLUTION** IIS3DWB_4g

  Resolution of the accelerometer in a unit of g.

Functions

- int sensor_enable (spi_device_handle_t ∗spi_dev, stmdev_ctx_t ∗dev)

  Configure SPI bus and set accelerometer to required parameters.
- void sensor_disable (spi_device_handle_t spi_dev)

  Remove accelerometer from the SPI bus and disable it.
- void sensor_events_enable (stmdev_ctx_t ∗dev)

  Enable accelerometer interrupts.
- void sensor_events_disable (stmdev_ctx_t ∗dev)

  Disable accelerometer interrupts.

1.1.3.1 Detailed Description

Configuration of the accelerometer sensor.

1.1.3.2 Function Documentation

sensor_disable()

```
void sensor_disable (
            spi_device_handle_t spi_dev )
```

Remove accelerometer from the SPI bus and disable it.

Parameters

in	*spi_dev*	SPI bus

sensor_enable()

```
int sensor_enable (
            spi_device_handle_t * spi_dev,
            stmdev_ctx_t * dev )
```

Configure SPI bus and set accelerometer to required parameters.

Parameters

in	*spi_dev*	SPI bus
in	*dev*	Accelerometer sensor

Returns

Status code of successful setup

sensor_events_disable()

```
void sensor_events_disable (
            stmdev_ctx_t * dev )
```

Disable accelerometer interrupts.

Parameters

in	*dev*	Accelerometer sensor

sensor_events_enable()

```
void sensor_events_enable (
            stmdev_ctx_t * dev )
```

Enable accelerometer interrupts.

Parameters

in	*dev*	Accelerometer sensor

1.1.4 Memory card

Filesystem operations of the SD card.

Functions

- sdmmc_card_t * storage_enable (const char *mount_point)

  Configure SD/MMC bus and mount SD memory card to FAT filesystem.
- void storage_disable (sdmmc_card_t *card, const char *mount_point)

  Disable and unmount SD memory card from FAT filesystem.
- void get_recording_filename (char *filename, const char *path)

  Get file name for new recording with sequentially higher unused number.

1.1.4.1 Detailed Description

Filesystem operations of the SD card.

1.1.4.2 Function Documentation

get_recording_filename()

```
void get_recording_filename (
            char * filename,
            const char * path )
```

Get file name for new recording with sequentially higher unused number.

Parameters

out	*filename*	Available file name for new file
in	*path*	Base path prefix for saving the recording

storage_disable()

```
void storage_disable (
            sdmmc_card_t * card,
            const char * mount_point )
```

Disable and unmount SD memory card from FAT filesystem.

Parameters

in	*card*	SD/MMC card information structure
in	*mount_point*	path where partition is registered

Returns

c

storage_enable()

```
sdmmc_card_t * storage_enable (
            const char * mount_point )
```

Configure SD/MMC bus and mount SD memory card to FAT filesystem.

Parameters

in	*mount_point*	path where the partition will be registered

Returns

SD/MMC card information structure

1.1.5 Button

Configuration of the recording button.

Functions

- void switch_enable (bool on, gpio_isr_t isr_handler)
  Configure GPIO input pin and interrupt handler for button press.
- void **switch_disable** (void)
  Remove interrupt handler for button press.

1.1.5.1 Detailed Description

Configuration of the recording button.

1.1.5.2 Function Documentation

switch_enable()

```
void switch_enable (
            bool on,
            gpio_isr_t isr_handler )
```

Configure GPIO input pin and interrupt handler for button press.

Parameters

in	*on*	decides whether the button is enabled or disabled
in	*isr_handler*	handler function for button press in interrupt context

1.1.6 LED

Configuration of the indicator LED light.

Functions

- void **led_enable** (void)
  Configure GPIO for LED to output mode.
- void led_light (bool on)
  Set the LED state.

1.1.6.1 Detailed Description

Configuration of the indicator LED light.

1.1.6.2 Function Documentation

led_light()

```
void led_light (
            bool on )
```

Set the LED state.

Parameters

in	*on*	turns LED light to be either on or off

1.2 Firmware Tasks

Main program of the firmware execution.

Functions

- void **push_trigger** (void ∗args)

  Task to start or stop recoding after siganl from button press.
- void **read_accelerometer** (void ∗args)

  Task to read FIFO buffer of the accelerometer and write it to Queue.
- void **write_card** (void ∗args)

  Task to write accelerations vectors from Queue to the memory card.
- void **app_main** (void)

  Entrypoint of firmware to setup hardware peripherals and run tasks.

Variables

- TaskHandle_t **trigger_task**

  Task handler for notification of button press.
- TaskHandle_t **sampler_task**

  Task handler for notification from sampling timer.
- QueueHandle_t **samples**

  Queue for sending samples from the sensor read task to the memory card write task.
- spi_device_handle_t **spi**

  SPI bus handle.
- stmdev_ctx_t **sensor**

  Accelerometer sensor device.
- sdmmc_card_t ∗ **card** = NULL

  SD memory card handle.

- FILE * **file** = NULL

  Currently opened file handle.
- SemaphoreHandle_t **file_mutex**

  Mutex to protect file handle.
- bool **is_recording** = false

  Flag for active recording in progress.
- int32_t **sensor_timestamp** = 0

  Last seen accelerometer timestamp.
- const esp_timer_create_args_t sampler_timer_conf
- esp_timer_handle_t **sampler_timer**

  Periodic timer to signal when to read FIFO buffer from accelerometer.
- const esp_timer_create_args_t stop_timer_conf
- esp_timer_handle_t **stop_timer**

  Timer to stop recording after fixed amount of time.

1.2.1 Detailed Description

Main program of the firmware execution.

1.2.2 Variable Documentation

sampler_timer_conf

```
const esp_timer_create_args_t sampler_timer_conf
```

Initial value:

```
= {
        .callback = &isr_sample
}
```

stop_timer_conf

```
const esp_timer_create_args_t stop_timer_conf
```

Initial value:

```
= {
    .callback = &stop_timer_run
}
```

2 Data Structure Documentation

2.1 Acceleration Struct Reference

Unprocessed data packet for storing samples from accelerometer.

```
#include <pinout.h>
```

Data Fields

- uint16_t **len**

  Number of samples in every array in the structure.
- int32_t **t** [FIFO_LENGTH]

  Array of timestamps relative to time when sensor was enabled.
- int32_t **x** [FIFO_LENGTH]

  Accelerometer samples for X axis.
- int32_t **y** [FIFO_LENGTH]

  Accelerometer samples for Y axis.
- int32_t **z** [FIFO_LENGTH]

  Accelerometer samples for Z axis.

2.1.1 Detailed Description

Unprocessed data packet for storing samples from accelerometer.

The documentation for this struct was generated from the following file:

- firmware/main/include/pinout.h

Appendix B: User Guide

B.1 Installation guide

The development platform was Acer Aspire A515-47 laptop with Linux distribution *Manjaro 23.1.4.* and kernel version 6.1. The installation process consists of getting the Python 3.11. packages for dataset analysis and development tools for building and flashing firmware for data logger. The paths are written relative to the root directory of the digital medium after unzipping.

The dependencies for running Jupyter notebooks can be installed by simply executing the following command ideally under a Python virtual environment:

```
$ pip install -r docs/requirements.txt
```

To run the Jupyter environment and view notebooks run:

```
$ jupyter lab
```

Building and uploading firmware of the data logger demands the following to be installed:

1. Install system prerequisites and download ESP-IDF:

   ```
   $ sudo pacman -S --needed gcc git make flex bison gperf
      python cmake ninja ccache dfu-util libusb
   $ mkdir -p ~/esp
   $ cd ~/esp
   $ git clone -b v5.2.1 --recursive https://github.com/
      espressif/esp-idf.git
   ```

2. Install the tools used by ESP-IDF which are compiler, debugger, etc:

   ```
   $ cd ~/esp/esp-idf
   $ ./install.sh esp32
   ```

3. Connect ESP32 microcontroller via USB and flash firmware onto the device:

   ```
   $ . ~/esp/esp-idf/esp-idf/export.sh
   $ cd firmware
   $ idf.py build
   $ idf.py -p /dev/ttyUSB0 flash
   ```

B.2 Data logger guide

To measure vibrations with a data logger follow these steps.

1. Insert empty microSD card to the data logger.
2. Attach the accelerometer sensor to the surface of measurement.
3. Press the button and watch the LED light up.
4. When the light turns off sooner before the minute mark, the buffer was dropped, and the final file can be incomplete.
5. Wait until the LED turns off, the recording is saved to file numbered sequentially from *1.tsv* upwards.
6. Move files from microSD card to a separate directory e.g. *abc* and run for conversion to tsv files:

   ```
   $ python bin2tsv.py abc 4
   ```

B.3 Documentation guide

Built documentation is located in directory *docs* of the digital medium. It can be rebuild using documentation tools *Doxygen* for firmware C source code and *Sphinx* for data analysis Python code. The steps of generating documentation from source codes are as follows.

1. Install necessary documentation tools:

   ```
   $ sudo pacman -S doxygen
   $ pip install sphinx sphinx-rtd-theme
      sphinx_autodoc_defaultargs
   ```

2. Rebuild the documentation using Doxygen and Sphinx:

   ```
   $ doxygen docs/firmware/doxygen.conf
   $ cd docs/vibrodiagnostics
   $ make html
   ```

3. View Doxygen docs in *docs/firmware/html/topics.html* and Sphinx docs in *docs/vibrodiagnostics/build/html/index.html*

Appendix C: IIT.SRC 2024 Paper

Fault Classification of Rotating Machinery using Limited Set of Features and k-Nearest Neighbors

Miroslav Hájek*

Faculty of Informatics and Information Technologies STU in Bratislava
xhajekm@stuba.sk

Abstract. The ability to detect and diagnose faulty parts of the machine is important for the industry at scale. Continuous monitoring of vibrations with accelerometers used as a diagnostic tool demands significant requrements for bandwidth and storage of samples. Standards already establish reliable attributes for the fault presence. The same cannot be said for purposes of root cause analysis. The goal is to examine the effect of reducing the number of features for fault prediction accuracy in the k-nearest neighbor model. We extract ten features in the time domain and eleven in frequency domain in the MaFaulDa dataset. We compare the accuracy of the whole feature set with three member subsets chosen by brute force, PCA, and feature selection methods: correlation, f-statistic, mutual information, and their rank product. We also qualitatively analyze features from the vibrations of water pumps. The rank product reaches accuracy 83.90% and 79.08%, for the time and frequency domain, respectively, which is by 1.82% and 5.13% less than the best three-member subset found by brute force.

Keywords: feature selection · machine fault diagnosis · vibrations · MaFaulDa · k-NN

1 Introduction

Rotary motion machinery is utilized throughout the industry as motors, pumps, and gearboxes. All equipment has a bounded productive lifespan because of wear and tear or improper operating conditions. Sooner or later, moving elements start to exhibit faults that can lead to failure if left untreated.

Early real-time fault detection can be achieved by monitoring machine vibration levels and signal frequency content [28]. Sensors for vibrations are piezoelectric or MEMS wideband accelerometers. Data acquisition procedures and best evaluation practices are standardized in ISO 20816 [8] and ISO 13373 [7].

Such a monitoring solution with the Internet of Things devices allows us to introduce predictive (PdM) or condition-based maintenance (CbM). This strategy prolongs the life of the equipment and reduces the cost of replacement parts and preventive maintenance [28].

* Master study programme in field: Informatics, Supervisor: Dr. Marcel Baláž, Institute of Computer Engineering and Applied Informatics, Faculty of Informatics and Information Technologies STU in Bratislava

A large number of machines in the plant and frequent sampling intervals pose a challenge with the sheer amount of samples gathered. Data reduction should happen close to the source to save on storage requirements, and bandwidth and promote retainment of important observations only. An additional difficulty is to tailor the fault classification model to individual machinery because each piece has natural imperfections and differences in its construction.

Technical diagnosis identifies faulty machinery parts that can be bearings, shafts, coupling, belts, or gear. The commonly found defects are imbalance, misalignment, looseness, eccentricity, deformation, crack, and rub [16, 22]. Most symptoms in vibrations appear synchronous to the rotational speed of the component under investigation as fundamental frequency of rotation or its harmonics [6]. Multiple faults can arise simultaneously or induce one another. For example, a severely damaged bearing introduces shaft imbalance.

2 Related work

Approaches taken in literature to reduce vibration samples and classify faults result in a similar framework of machine learning pipeline. The general sequence of steps consists of signal acquisition, feature extraction, dimensionality reduction, and pattern recognition or fault detection [25, 5].

Data retrieval from acceleration sensors is supplemented by a factory database of machinery mechanical parameters and operating conditions [10]. The signal processing methods extract low-level numerical features in the time domain, frequency domain, and time-frequency domain [19]. Time-frequency features depend on the transform method of which Morlet wavelet transform, discrete wavelet transform, or wavelet packet transform are the ones used besides short-time Fourier transform [14]. Then, statistical functions such as root mean square, centroid, kurtosis, energy, and entropy are applied to further compress the waveform.

Feature extraction of time and frequency domain attributes is utilized in online monitoring of band saw blade wear status from acoustic emission signal using SVM model [27] and in novelty detection of health status of centrifugal pump using autoencoder [18]. Wavelength length, average amplitude change, and zero crossing taken out of overview of time domain features for surface electromyography (sEMG) [15] are generic enough to be tried for machine diagnostic. Harmonic features originally designed for sound description [20] could be reused similarly. Many features found in the literature are already implemented in the Time Series Feature Extraction Library (TSFEL) which is a package for Python language [3].

Knowledge of rolling bearings geometry and rotational speed provides a way to calculate bearing defect frequencies for outer race (BPFO), inner race (BPFI), ball (BSF), and cage (FTF) [16, 28]. There are situations when bearing fault is not evidenced by defect frequency. Therefore, kurtosis is recommended as it is associated with impacts [5].

Normalization assigns the same weights to unbounded variables [26]. In three-dimensional motion, either axis aligned only with the principal direction of vibrations is retained, or the Euclidian norm is computed for each feature. The idea of feature vector magnitude comes from animal activity recognition based on collar tags independent of sensor orientation [11].

Extracted features lower the number of dimensions from the original signal. Further dimensionality reduction is carried out with various methods: PCA, t-SNE, ISOMAP, ICA, and AE [5]. Keeping a smaller number of attributes can result in better classification performance. However, the major disadvantage is the lack of meaning in acquired features.

Filtering feature selection methods decrease the feature set size so that the chosen predictors are related the strongest to a categorical predicted variable. The relationship is quantified with correlation, fisher score, or mutual information and among others [19]. The metrics are combined into ensembles using electoral methods or rank product [4].

Machinery Fault Database (MaFaulDa) [13] or Case Western Reserve University (CWRU) bearings dataset [12] are commonplace in the evaluation of machinery fault models. MaFaulDa contains a 1951 time series measured on two bearings simultaneously with three uniaxial accelerometers at a sampling rate of 50 kHz. Each recording lasts 5 seconds and represents one of 5 defects but at different loads, shaft shifts, and rotation speeds between 737 to 3686 RPM. The CWRU dataset has 161 scenarios at sample rates of 12 kHz or 48 kHz. The time series captures four different states of bearings on both sides of the motor.

Classification of faults in machinery diagnostics focuses primarily on bearings health monitoring. Review article [23] compares the strengths and weaknesses of the machine learning and deep learning models. Power spectral density (PSD) features from vibration signals of journal bearings in combination with k-NN and SVM reached classification accuracy 85.7% and 100%, respectively [17]. Ball bearing fault diagnostic of CWRU dataset indicates better accuracy of k-NN (98.8%) than SVM (96.2%) on frequency-domain features [9]. An approach based on statistical features utilizing a custom test rig compared three models for ball bearing faults: k-NN, SVM, and kernel linear discriminant analysis (KLDA). Results showed KLDA achieves the best average accuracy on PSD features (99.13%) followed by k-NN with PSD (95.64%) and SVM with EMD (80.49%) [1].

On the CWRU dataset, SVM performs better compared to state-of-the-art methods for random forest (RF) and softmax classifiers. The best accuracy of SVM is 61.05% on 5 time domain features (RF 59.07%, Softmax 48.92%) and 91.17% on 64 frequency domain (RF 86.14%, Softmax 86.05%) features [14]. Random forest applied after the similarity-based model has an accuracy of 96.43% on the MaFaulDa and 98.7% on the CWRU database [21].

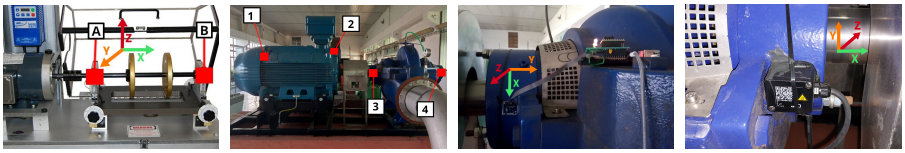
In the k-NN algorithm, the number of neighbors and similarity measure has to be chosen carefully. Labels in the dataset should be balanced in size when they are intended for the k-NN classifier. Otherwise, precision of the model drops 15% to over 40% depending on the imbalance ratio [24].

3 Methodology

The two principal questions regarding automated machinery fault diagnostics from vibrations are investigated in this study. How does reducing the number of features and different feature selection methods impact k-nearest neighbors multiclass classifier accuracy with varying k parameters? This first exploration is done by supervised learning with 5-fold cross-validation on a preprocessed MaFaulDa dataset. Then the challenges are presented of applying the same procedures to the monitoring solution of municipal water pumping station. Calibrated data from existing pump monitoring provided by the vendor on the site is taken as a reference.

Classification of the MaFaulDa dataset is carried out for extracted features in the time domain (TD) and frequency domain (FD) separately. The accuracy is determined for features in radial axis y and Euclidian norm of all directions (Fig. 1a) Experiments include different combinations of selected attributes according to predefined conditions:

1. whole feature set with k value as an odd number in the range from 1 to 37,
2. dimensionality reduction using principal components analysis (PCA) of the whole feature set to 3 principal components ($k = 5$),
3. every combination of 2, 3, 4, 5 member feature subsets with k value being sequentially: 3, 5, 11, 15 (results in $\sum_{j=2}^{5} \binom{n}{j}$ models for individual k value),
4. best feature subset picked using feature selection methods: mean of point-biserial correlation to every class as a dichotomous variable, ANOVA F-value, and mutual information ($k = 5$).



(a) MaFaulDa (b) Water pump (c) Accelerometer (d) KSB sensor

Fig. 1: Measurement positions and accelerometer orientations

Data preprocessing of MaFaulDa is necessary to create feature sets in TD and FD. This process has several phases.

1. **Signal filtering**: Subtraction of overall mean removes DC component. IIR Butterworth low pass filter of 5th order attenuates frequencies above 10 kHz.
2. **Feature extraction**: Machine behavior in one time series is time-invariant. Therefore, features are calculated out of the whole duration. Frequency domain features are produced using Welch's method for power spectrums from FFT with 2^{15} long Hann windows and 50% overlap (bin resolution of 1.53 Hz). The resulting features belong to domains making up respective feature sets:

(a) **10 features in time domain** (TD): peak-to-peak amplitude, zero-crossing rate, root mean square, skewness, kurtosis, shape factor, crest factor, impulse factor, clearance factor, average amplitude change,
(b) **11 features in frequency domain** (FD): spectral centroid, standard deviation, skewness, kurtosis, roll-on frequency of 5% total energy, roll-off frequency of 85% total energy, spectral flux as an average correlation of separate windows, noisiness as a signal to noise ratio, spectral negentropy [2], energy, entropy.

In summary 627 time domain and 1012 frequency domain feature subsets are evaluated by the same number of 5-fold cross-validated k-NN models for each value of k.

3. **Labeling**: Three target variables are constructed out of the inner bearing, both bearings and high severity defects on both bearings. The predicted variable has 6 classes: normal, misalignment (merged vertical and horizontal), imbalance, outer race fault, ball fault, and cage fault. In the "high severity" scenario, the normal class is assigned to observations where the weight or shaft shift applied is in lower half of all of the configurations for that fault numbered from least to most severe.
4. **Class balancing**: Random oversampling strategy is used on all classes except the majority one.

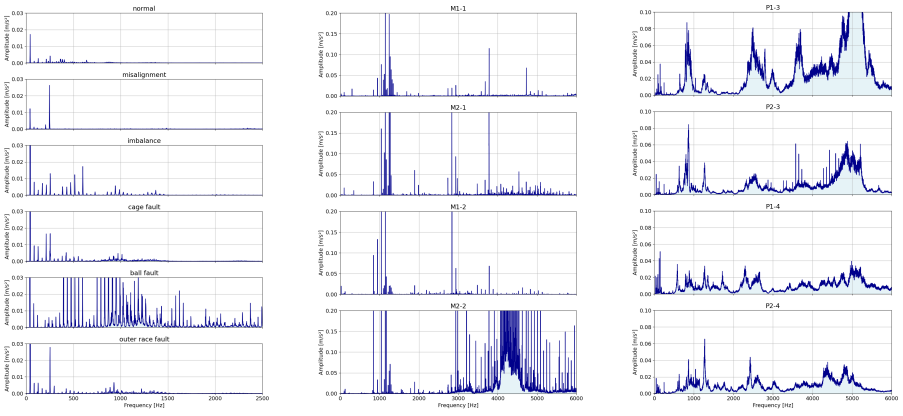
Vibrations from pumps in municipal pumping station for drinking water in Podunajské Biskupice reveal practical challenges for the deployment of machine learning models in the industry. Two pumps (P1, P2) of the same type KSB Omega 300-560, and two electric motors WEG W50 (M1, M2) were designated for measurements. Both machines have power 400 kW (ISO 20186 class III) and the main shaft rotates at 1493 RPM. For purposes of expert labeling, the bearing designations at numbered places are 6319-C3 (1), 6324-C3 (2), and 6317-2Z (3&4). Sensors were placed above bearings in numbered positions according to ISO 13373 (Fig. 1b) with axis orientation depicted in Fig. 1c. Each pump is equipped with KSB cloud monitoring at the place (3) (Fig. 1d) from which RMS velocity in mm/s is available for a year in hourly intervals.

Our accelerometer ST IIS3DWB mounted to the machine using thin double-sided tape has a wideband 5 kHz linear response sampled at 26.87 kHz. Microcontroller ESP32-POE-ISO stored triaxial 60-second long recordings onto an SD card. In total 48 time series were gathered, 6 in each place. These are split into 5-second intervals for 576 data points overall. Fault labels are missing, therefore we analyze rotational speeds and defective bearing frequencies in the frequency spectrum. The observations from machines are labeled based on source machinery to 4 classes: M1, M2, P1, P2; based on sensor position to 8 classes: M1-1, M1-2, P1-3, P1-4, etc.

4 Results

In qualitative exploratory data analysis, representative observations from both datasets are examined through their frequency spectra in the radial direction.

Patterns for categories of faults in MaFaulDa with shaft speed around 2500 RPM and in conditions of the worst severity are depicted in the wideband spectrum with a range up to 2.5 kHz (Fig. 2a). Clear distinctions in spectra are apparent for different labels. The rotational frequency shows up as a spike in each case and is the only one in a normal state. Misalignment gives rise in amplitude to 6th RPM harmonic, whereas imbalance excites even more harmonic frequencies.

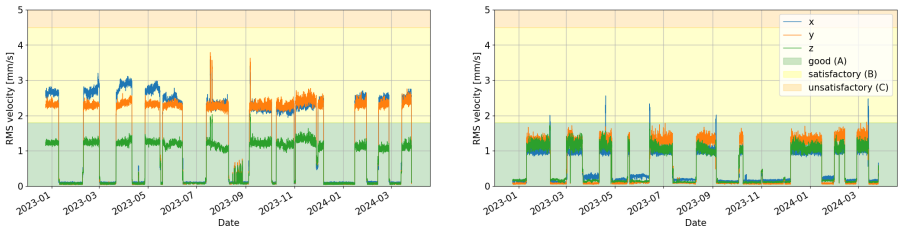


(a) MaFaulDa (b) Motors (c) Pumps

Fig. 2: Wideband power spectrum

The wideband spectrum of the electric motor is polluted by the wind coming out of the back fan blades (Fig 2b). Pump spectra are lookalike in the same respective positions (Fig 2c). P1 has significantly more energy than P2 in frequency bands around 2.5 kHz, 3.6 kHz, and 5 kHz and in low frequencies. This energy can indicate a fault in the early stages because amplitudes are not tall enough. The range of amplitudes measured is at most ± 50 m/s^2.

The long-term vendor's pump monitoring system confirms that the velocity of vibrations is still in normal conditions. Levels in radial axis (z) are almost identical for both pumps circa 1.6 mm/s (Fig. 3).



(a) P1-3 (b) P2-3

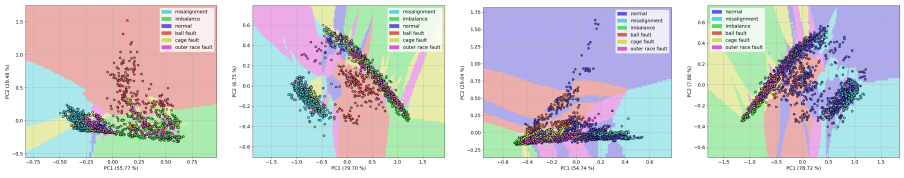
Fig. 3: Vibrations of water pumps and ISO severity levels

Table 1: Numbers of observations by label in MaFaulDa

Label	Bearing A	Both bearings	Severity
normal	49	49	1125
misalignment	498	498	198
imbalance	333	333	139
cage fault	188	376	181
ball fault	186	323	132
outer race fault	184	372	176
Σ	1438	1951	1951

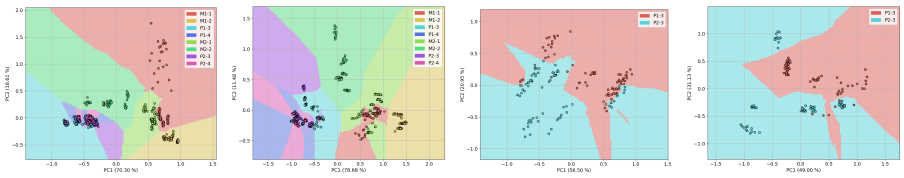
Scatter plots (Fig. 4, 5) visualize the spatial distribution of labels for entire feature sets after min-max scaler and PCA. Explained variance percentages are next to the component's name. Labels are assigned in three different ways producing observation counts according to Table 1. The imbalance ratio in the MaFaulDa dataset is 1016% for original classes and 852% for high-severity modification. Colored regions are decision boundaries of the k-NN classifier trained on 80% of the PCA transformed dataset.

Misalignment and ball fault groups are separated the best from the rest but the imbalance group is intertwined with bearing faults. Silhouette scores of the feature sets of the MaFaulDa after min-max scaling are 0.05 (TD) and 0.14 (FD). Sensor placements on pumps have silhouette scores of 0.20 (TD) and 0.21 (FD).



(a) Bearing A in TD (b) Bearing A in FD (c) Severity in TD (d) Severity in FD

Fig. 4: PCA scatter plots of MaFaulDa with 5-NN decision boundaries

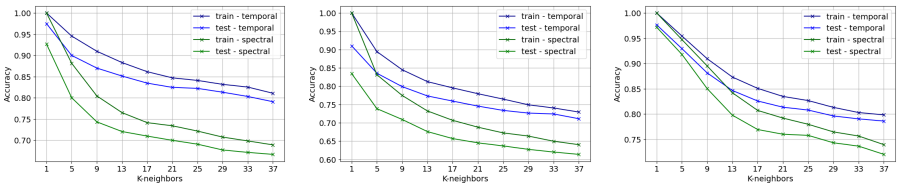


(a) Positions in TD (b) Positions in FD (c) P1-3/P2-3 in TD (d) P1-3/P2-3 in FD

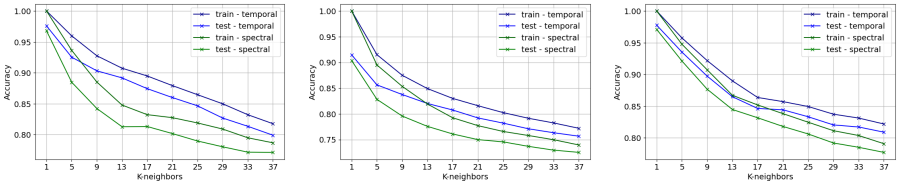
Fig. 5: PCA scatter plots of pumps with 5-NN decision boundaries

The increase in hyperparameter of k for whole feature sets results in less accuracy of k-NN (Fig. 6). The sharp decline of more than 10% in some cases is noticeable for k between 1 and 9. The features extracted in the time domain have overall better accuracy than those in the frequency domain. The attributes

combined from all sensor directions also reach better accuracies than just from a single axis.



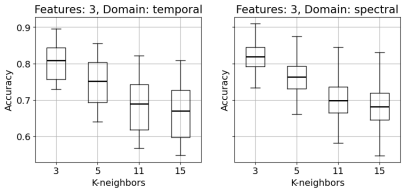
(a) One axis, Bearing A (b) One axis, Both bearings (c) One axis, High severity



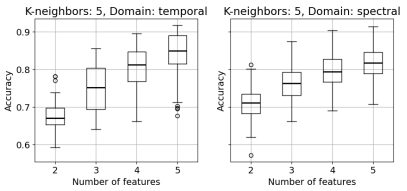
(d) All axis, Bearing A (e) All axis, Both bearings (f) All axis, High severity

Fig. 6: Accuracy on MaFaulDa feature sets TD and FD depending on k value

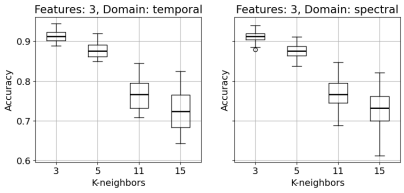
The tendency of decrease in accuracy with an increase in number of neighbors holds even in feature subsets (Fig. 7a, 7c). The k-NN trained with any 3 features from FD have accuracy distribution shifted higher than TD, contrary to whole feature sets. Picking a model with $k < 15$ at random always guarantees better than chance accuracy. Increasing the number of features increases the k-NN accuracies for both domains and scenarios at any k value (Fig. 7b, 7d).



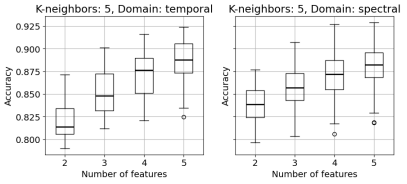
(a) Bearing A



(b) Bearing A



(c) High severity



(d) High severity

Fig. 7: Test accuracies of all combinations of feature subsets

Tables 2 and 3 compare the average 5-fold test accuracy of different feature selection methods for the k-NN model with k = 5. Three-axis features reach better accuracy the one-axis ones.

Table 2: Accuracy of k-NN in TD feature set and subsets of 3

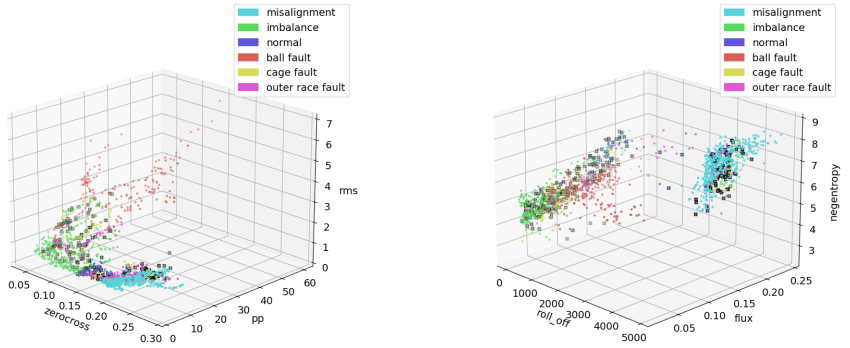
Axis	**Method**	**Bearing A**		**Both Bearings**		**High severity**	
		Train	**Test**	**Train**	**Test**	**Train**	**Test**
1	All features	*94.56*	89.99	*89.45*	83.53	*94.93*	91.60
	PCA 3 PC	*88.63*	80.09	*82.87*	73.66	*92.17*	87.22
	Best 3 features	*91.98*	85.37	*85.56*	78.48	*93.90*	89.64
	Rank product	*87.96*	79.99	*80.71*	71.42	*93.07*	88.50
	Correlation	*88.83*	80.99	*81.40*	72.83	*93.07*	88.50
	F-statistic	*87.53*	79.08	*80.71*	71.42	*93.07*	88.50
	Mutual information	*87.06*	78.75	*80.71*	71.42	*93.07*	88.50
3	All features	*95.98*	92.54	*91.52*	85.64	*95.37*	92.82
	PCA 3 PC	*91.17*	84.74	*85.41*	77.64	*93.53*	89.15
	Best 3 features	*91.91*	85.57	*86.04*	78.01	*93.75*	90.14
	Rank product	*90.09*	83.90	*83.90*	76.11	*93.62*	89.96
	Correlation	*91.44*	84.87	*83.90*	76.11	*93.66*	90.12
	F-statistic	*90.09*	83.90	*83.90*	76.11	*93.62*	89.96
	Mutual information	*91.44*	84.87	*83.90*	76.11	*93.62*	89.96

Table 3: Accuracy of k-NN in FD feature set and subsets of 3

Axis	**Method**	**Bearing A**		**Both Bearings**		**High severity**	
		Train	**Test**	**Train**	**Test**	**Train**	**Test**
1	All features	*88.14*	80.02	*83.12*	73.86	*94.63*	90.53
	PCA 3 PC	*83.85*	74.26	*77.94*	67.57	*92.02*	87.33
	Best 3 features	*88.64*	81.53	*82.43*	74.67	*93.70*	89.44
	Rank product	*86.85*	78.28	*82.24*	72.92	*92.93*	89.23
	Correlation	*80.53*	68.01	*73.47*	61.04	*92.93*	89.23
	F-statistic	*86.85*	78.28	*82.24*	72.92	*92.93*	89.23
	Mutual information	*87.88*	80.49	*78.55*	68.21	*92.93*	89.23
3	All features	*93.64*	88.49	*89.51*	82.83	*94.90*	91.49
	PCA 3 PC	*87.12*	79.18	*82.09*	74.33	*91.84*	86.96
	Best 3 features	*92.71*	87.48	*88.30*	81.96	*93.80*	90.72
	Rank product	*87.58*	79.08	*82.63*	73.90	*93.48*	89.99
	Correlation	*84.45*	74.87	*82.63*	73.90	*93.48*	89.99
	F-statistic	*90.51*	84.30	*82.43*	74.40	*93.48*	89.99
	Mutual information	*90.27*	83.03	*85.35*	78.88	*92.76*	88.74

Vibrations from just bearing A cannot reliably detect bearing faults in position B, located on the opposite end of the shaft. Average drops in accuracy to diagnose both bearings from the same position are 7.81% (TD) and 5.18% (FD) for 3-axis features. Rank product of feature selection methods does not always perform better than the individual ones but provides an acceptable middle-ground.

All extracted features for bearing A reach accuracies of 92.54% (TD) and 88.49% (FD). The best subset of the three features is worse by 6.97% (TD) and 2.92% (FD). Rank product regularly picks model with accuracy on upper quartile Q3 of k-NN accuracy distribution, but there is still room for improvement to the best subset. In high-severity situations, the classifier performance is better for TD with three-dimensional features: 92.82% (TD), 91.49% (FD). Features selected by rank product have accuracy larger than principal components with differences 1.28% (TD), 2.66% (FD).



(a) TD features (Accuracy = 83.90%) (b) FD features (Accuracy = 79.08%)

Fig. 8: Best 3 features for bearing A chosen by rank product

The most commonly chosen subset of features by rank product are in order of decreasing popularity in the time domain: zero-crossing rate, root mean square, peak-to-peak, and in the frequency domain: roll-off frequency, centroid, flux, or entropy. Figure 8 shows clustering abilities of selected feature space for bearing A. Scale of attributes values are inverse of mix-max scaler function.

Features expressing machinery vibration amplitude, chaos, or richer frequency content look to be better predictors of fault types. The potential design of new features should follow these observations.

5 Conclusions

We reviewed processes and standards for vibration monitoring of rotating machinery using accelerometers. The aim is to diagnose faults of the machinery parts with a machine learning model and fewer features to make results more comprehensible. The related work includes low-level feature extraction as an important preprocessing step to reduce information from vibration signals.

We built an embedded system with ESP32 MCU capable of recording vibrations from water pumps. Based on the data analysis of collected data, we discussed the problems with the implementation of a diagnostic model for machinery in the industry.

We developed the data pipeline with signal preprocessing and the k-nearest neighbors model for the MaFaulDa dataset. The bigger k values cause less accuracy for all feature spaces. Increasing the number of features taken out of the same feature set improves classification accuracy. The rank product balances the performance of other feature selection methods in the ensemble and reaches accuracies close to the upper quartile of the model distribution.

The best validation accuracies found for one bearing when using just three features picked by rank product are 83.90% (TD) and 79.08% (FD). It is less than the accuracy for the best subset of three features by 1.82% (TD), 5.13% (FD), and is less than for the whole feature set. Relabeling faults to be only those with high severity gives an accuracy of three features for both bearings 89.96% (TD) and 89.99% (FD). Testing out different base feature sets could bring improvements for so few features.

Acknowledgements

We thank Lukáš Doubravský (R-DAS, s.r.o.) for consultations on the methodology and assisting in sensor unit development. We appreciate domain experts in vibrodiagnostics prof. Stanislav Žiaran and Dr. Ondrej Chlebo (SjF STU) for checking our approches. We also thank Peter Csóka and Peter Kmeťko (Bratislavská vodárenská spoločnosť, a.s.) for enabling us access to water pumps.

References

[1] Muhammad Altaf et al. "A New Statistical Features Based Approach for Bearing Fault Diagnosis Using Vibration Signals". In: *Sensors* 22.5 (5 Jan. 2022), p. 2012. ISSN: 1424-8220. DOI: 10.3390/s22052012. URL: https://www.mdpi.com/1424-8220/22/5/2012.

[2] Moise Avoci. "Spectral Negentropy and Kurtogram Performance Comparison for Bearing Fault Diagnosis". In: Dubrovnik, Croatia: International Measurement Confederation (IMEKO), Oct. 20, 2020.

[3] Marília Barandas et al. "TSFEL: Time Series Feature Extraction Library". In: *SoftwareX* 11 (2020), p. 100456.

[4] Rainer Breitling et al. "Rank products: a simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments". In: *FEBS letters* 573.1-3 (Aug. 2004), pp. 83–92. ISSN: 0014-5793. DOI: 10.1016/j.febslet.2004.07.055.

[5] Lucas Costa Brito et al. "Fault Detection of Bearing: An Unsupervised Machine Learning Approach Exploiting Feature Extraction and Dimensionality Reduction". In: *Informatics* 8.4 (4 Dec. 2021), p. 85. ISSN: 2227-9709. DOI: 10.3390/informatics8040085. URL: https://www.mdpi.com/2227-9709/8/4/85.

[6] A. Davies. *Handbook of Condition Monitoring: Techniques and Methodology*. Dordrecht: Springer Netherlands, 2012. ISBN: 978-94-011-4924-2.

[7] *ISO 13373-1:2002 - Condition Monitoring and Diagnostics of Machines - Vibration Condition Monitoring - Part 1: General Procedures*. 2002. URL: https://www.iso.org/standard/21831.html.

[8] *ISO 20816-1:2016 - Mechanical Vibration - Measurement and Evaluation of Machine Vibration - Part 1: General Guidelines*. 2016. URL: https://www.iso.org/standard/63180.html.

[9] Mohd Atif Jamil, Md Asif Ali Khan, and Sidra Khanam. "Feature-Based Performance of SVM and KNN Classifiers for Diagnosis of Rolling Element Bearing Faults". In: *Vibroengineering PROCEDIA* 39 (2021), pp. 36–42. ISSN: 2345-0533. DOI: 10.21595/vp.2021.22307. URL: https://www.extrica.com/article/22307.

[10] Deokwoo Jung, Zhenjie Zhang, and Marianne Winslett. "Vibration Analysis for IoT Enabled Predictive Maintenance". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 2017 IEEE 33rd International Conference on Data Engineering (ICDE). Apr. 2017, pp. 1271–1282. DOI: 10.1109/ICDE.2017.170.

[11] Jacob W. Kamminga et al. "Robust Sensor-Orientation-Independent Feature Selection for Animal Activity Recognition on Collar Tags". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2.1 (Mar. 2018), 15:1–15:27. DOI: 10.1145/3191747. URL: https://dl.acm.org/doi/10.1145/3191747 (visited on 11/21/2023).

[12] K.A. Loparo. *Bearings vibration data set, Case Western Reserve University,* URL: https://engineering.case.edu/bearingdatacenter (visited on 03/29/2024).

[13] *MaFaulDa - Machinery Fault Database*. URL: http://www02.smt.ufrj.br/%20681%20~offshore/mfs/ (visited on 03/29/2024).

[14] Seetaram Maurya et al. "Condition-Based Monitoring in Variable Machine Running Conditions Using Low-Level Knowledge Transfer With DNN". In: *IEEE Transactions on Automation Science and Engineering* 18.4 (Oct. 2021), pp. 1983–1997. ISSN: 1558-3783. DOI: 10.1109/TASE.2020.3028151.

[15] Sidi Mohamed Sid'El Moctar, Imad Rida, and Sofiane Boudaoud. "Time-domain features for sEMG signal classification: A brief survey". en. In: (2023).

[16] Amiya Ranjan Mohanty. *Machinery Condition Monitoring: Principles and Practices*. CRC Press, 2015. 256 pp. ISBN: 978-1-4665-9305-3.

[17] Ashkan Moosavian et al. "An Appropriate Procedure for Detection of Journal-Bearing Fault using Power Spectral Density, K-Nearest Neighbor and Support Vector Machine". In: 5 (Sept. 2012), pp. 685–700. DOI: 10.21307/ijssis-2017-502.

[18] Alireza Mostafavi and Ali Sadighi. "A Novel Online Machine Learning Approach for Real-Time Condition Monitoring of Rotating Machines". In: *2021 9th RSI International Conference on Robotics and Mechatronics (ICRoM)*. Nov. 2021, pp. 267–273. DOI: 10.1109/ICRoM54204.2021.9663495.

[19] Asoke Kumar Nandi and Hosameldin Ahmed. *Condition Monitoring with Vibration Signals: Compressive Sampling and Learning Algorithms for Rotating Machines*. Hoboken, NJ, USA: Wiley-IEEE Press, 2019. ISBN: 978-1-119-54462-3.

[20] Geoffroy Peeters. "A Large Set of Audio Features for Sound Description". In: (2004).

[21] Felipe Ribeiro et al. "Rotating Machinery Fault Diagnosis Using Similarity-Based Models". In: *Anais de XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*. Sociedade Brasileira de Telecomunicações, 2017. DOI: 10.14209/sbrt.2017.133. URL: http://biblioteca.sbrt.org.br/articles/601.

[22] C. Scheffer and P. Girdhar. *Practical Machinery Vibration Analysis and Predictive Maintenance*. IDC Technologies, Elsevier, 2004. 252 pp. ISBN: 0-7506-6275-1.

[23] Hao Sheng et al. "Review of Artificial Intelligence-based Bearing Vibration Monitoring". In: *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan)*. Oct. 2020, pp. 58–67. DOI: 10.1109/PHM-Jinan48558.2020.00018.

[24] Zhan Shi. "Improving k-Nearest Neighbors Algorithm for Imbalanced Data Classification". In: *IOP Conference Series: Materials Science and Engineering* 719.1 (Jan. 2020), p. 012072. ISSN: 1757-8981, 1757-899X. DOI: 10.1088/1757-899X/719/1/012072. URL: https://iopscience.iop.org/article/10.1088/1757-899X/719/1/012072 (visited on 11/25/2023).

[25] Xiang Wang et al. "Bearing Fault Diagnosis Based on Statistical Locally Linear Embedding". In: 15 (July 2015), pp. 16225–47. DOI: 10.3390/s150716225.

[26] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning*. O'Reilly Media, 2018. ISBN: 978-1-4919-5324-2.

[27] Rongjin Zhuo et al. "Research on Online Intelligent Monitoring System of Band Saw Blade Wear Status Based on Multi-Feature Fusion of Acoustic Emission Signals". In: *The International Journal of Advanced Manufacturing Technology* 121.7 (Aug. 1, 2022), pp. 4533–4548. ISSN: 1433-3015. DOI: 10.1007/s00170-022-09515-3. URL: https://doi.org/10.1007/s00170-022-09515-3.

[28] Stanislav Žiaran. *Technická diagnostika*. 1. Bratislava: Vydavateľstvo STU, 2013. 332 pp. ISBN: 978-80-227-4051-7.

Appendix D: Work plan

D.1 Summer semester - DP1

Period	Work
1st week	Consultation with the supervisor on directions of the future work based on literature review during the previous semester.
2nd week	Outline the key sections of the analysis part in the thesis.
3rd week	Match supporting literature with analysis sections. Further investigation on the feature engineering methodology in CbM.
4th week	Summarize notes from condition monitoring articles and video recordings of tutorials and conferences.
5th week	Research transformation of a vibration signal to feature space using time-frequency, harmonic, and energy statistical metrics. Progress report meeting with the supervisor.
6th week	Find articles and take notes about unsupervised and semi-supervised techniques in streaming data for machinery diagnostics.
7th week	Narrow down a wide variety of applicable methods for signal decomposition.
8th week	Exploratory analysis on evaluation datasets. Progress report meeting with the supervisor on the topic of related work.
9th week	Organize detailed outline out of notes gathered during literature research.
10th week	Write up the problem analysis about condition monitoring and evaluation datasets.
11th week	Write up the analysis section about feature engineering.
12th week	Write up the analysis section about machine learning diagnostics and consult the final choice of methods in the analysis section.

D.2 Winter semester - DP2

Period	Work
1st week	Semester kickoff meeting to set goals, and experiments and discuss the status of collaborations with partners.
2nd week	Arrange collaboration with an alternative industry partner. Prepare a checklist for the technical inspection of machinery. Construct a device for exploratory measurements.
3rd week	Technical inspection of air conditioning units in the data center. Feature extraction step to calculate features from MaFaulDa.
4th week	Consultation about the plan for a machine to measure in the data center and a better device for measurement.
5th week	In feature selection using various metrics to determine sets of best features in the MaFaulDa.
6th week	Feature selection used in k-NN multiclass and binary classificator.
7th week	Explore incremental learning k-NN algorithm with MaFaulDa dataset. Consultation and status report.
8th week	Shorten analysis chapter about wavelets. Look at clustering detection incremental learning in MaFaulDa.
9th week	Refactor separate trials in feature selection to integrate them into the pipeline for k-NN validation. Consultation to discuss progress.
10th week	Include incremental learning in analysis. Experiment with gradual feature selection in incremental learning.
11th week	Consultation in preparation for machine inspections. Preliminary measurements of fan, compressors, and pump. Design the firmware for the new datalogger.
12th - 15th week	Write up design chapter: research questions and visualization export. Finish writing a chapter on design and implementation.

The semester for DP2 was split into 3 periods. Feature engineering and ML model evaluation was planned from 1st to 4th week, and technical inspections and plan of measurement in weeks 4th to 8th.

In reality, these tasks switched order because, between 1st to 5th week, we needed to acquire alternative partner and check their machiners for viability in our study. Only then since 4th until 11th week, the main focus was on experiments with the MaFaulDa.

Lastly, preliminary measurements were conducted, as was the plan. The requirements were presented for the new sensor device and its firmware.

D.3 Summer semester - DP3

Period	Work
1st week	Build and debug hardware of data logger with the consultant.
2nd week	Implement and test firmware on standing fan for logging signal to SD card.
3rd week	First compressor (20th February) and water pump (27th and 28th February) vibration measurements and their fault and frequency analysis.
4th week	Replace two time-domain features and evaluate on MaFaulDa and own dataset. Repeated measurement of compressors (5th March).
5th week	Perform PCA analysis of whole feature sets. Consultation about methods for own dataset evaluation.
6th week	Consultation with experts in SjF STU. Formulate research goals for scientific paper and create graphs of the model results. Experiment with the k-value and number of features to reach better accuracy. Repeated measurement of compressors (19th March).
7th week	Repeated measurement of water pumps (26th and 27th March). Request and download data about pumps from KSB Cloud.
8th week	Write up complete paper and submit to IIT.SRC student conference after feedback from supervisor.
9th week	Refactor Jupyter notebooks and apply different labeling strategies applied in paper to incremental learning.
10th - 13th week	Update design chapter and finish text according to figures generated by experiments.

The final semester aimed at gathering dataset from compressors and water pumps with the accelerometer data logger. The k-NN model was validated with different hyperparameters. The data collection finished sooner in 7th week whereas originally it was planned until 10th week because it became apparent that during a such short period, the change in machine vibrations would be negligible. The preparation for IIT.SRC conference was also not originally planned and took time away from finishing the thesis.

Appendix E: Digital medium

Registration number of thesis in AIS: FIIT-182905-102927

Contents of the digital medium:

Name of submitted archive directory: DP_MiroslavHajek.zip

Digital medium has more than 1 GiB and is available to the thesis supervisor.

The MaFaulDa dataset is publicly available at: `https://www02.smt.ufrj.br/~offshore/mfs/database/mafaulda/full.zip`

- **datasets** - raw vibration data from machinery, and reports that took long to calculate.
  - **FluidPump.zip** - custom recorded dataset of vibrations from air compressors and water pumps
  - *MAFAULDA.zip* - placeholder where the downloaded MaFaulDa dataset should be placed
  - **best-subset** - the subset of features chosen to three-member subsets most frequently
  - **features** - features extracted from datasets in multiple domains
  - **ksb-cloud** - data exported from KSB cloud monitoring of vibration rms velocity and frequency waveform for chosen dates.
  - **misc-fluid-pump** - other non-standard measurements from pumps such as during speed up, slow down, or noise.
  - **knn-accuracy-distribution** -
  - **knn-incremental-accuracy** - The incremental k-NN model accuracy after each sample for various lengths of tumbling window and gaps between labels
  - **standing-fan** - audio for an estimate of fan rotational speed and measurements done in firmware verification on the back, side, and front of the fan.

- **docs** - the documentation generated from source code comments using automated tool
  - **html-notebooks** - export of Jupyter notebooks for all basic configurations in HTML format
  - **doxygen** - documentation of *firmware*
  - **sphinx** - documentation of *vibrodiagnostics* package
- **firmware** - source code in *main* directory for accelerometer data logger firmware written in C language with ESP-IDF SDK. The script *bin2tsv.py* converts files saved to SD card in binary format to a tsv file.
- **notebooks** - Jupyter notebooks for data exploration, data analysis, and machine learning on provided datasets
- **vibrodiagnostics** - Python package of commonly used function in Jupyter notebooks for data processing