**Problem description**

The problem at hand involves the condition monitoring of railway switches and identifying local discontinuities in the rail through processing the data from an acceleration sensor installed on the railway switch and classifying the results to normal or abnormal events in the switch operation. These events are typically caused by local discontinuities in the rail, such as surface defects, misalignments, or mechanical faults. When a train passes over such anomalies, abrupt changes in acceleration are generated, which can be captured by wayside acceleration sensors mounted near the tracks. The data collected this way include several statistical features (e.g., mean, variance, skewness) and frequency features (e.g., spectral energy, dominant frequency) as well as a label indicating the presence (or not) of a transient event.

Figuring out and classifying these events with machine learning can really help build smarter train monitoring systems. Such systems make maintenance easier, cut down delays and shutdown times as well as optimize the use of maintenance resources.

**Method of solution**

A supervised classification approach is being used in this project to detect abnormal transient events on railway switches from a series of features recorded from the acceleration sensor recorded data. More specifically, we are planning to perform binary classification with all the different labels for the various transient effects grouped under single “abnormal” label. The project involves the preprocessing, transformation, and classification of the sensor entries and using a Support Vector Classifier. The classifier is trained on a training sub-set and can then be used to make predictions about the presence of an abnormal event.

More importantly we explore different feature selection techniques to select the most important input features in the dataset that contribute the most in the correct classification of the target label. By optimally selecting only the most important features, rather than classifying based on all available ones, we try to:

* Reduce the chance of overfitting
* Maintain high model accuracy with increased generalization
* Potentially increase the speed of training and save on computational expenses (for big datasets with many available features)
* Make the model easier to understand as users might have difficulties interpreting a model with a big number of features

**Steps to Implement Event Detection**

1. **Preprocessing**
   1. **Data Loading**

The data from the 3 files are loaded into a single pandas data frame for ease of pre-processing and slicing.

* 1. **Data pre-processing**

In the preprocessing step we can:

* drop unused columns (start\_time, axle, cluster, tsne\_1, tsne\_2) using the .drop() method on the data frame.
* Set the binary labels, 0 for the normal entries and 1 for all other ones
  1. **Train-Test sub-set splitting.**

We perform a stratified split the dataset to train and test subsets using the train\_test\_split() function from scikit-learn. We define a 80-20 ratio and a constant random\_state (=42) so that we can reproduce the same split.

* 1. **Normalizing.**

For normalizing we take two different approaches:

* First, we normalize the train and test sub-sets using the StandardScaler from sklearn.preprocessing. In order not to have leakage towards the test set, we train the scaler on the train subset and apply it also on the test.
* Specifically for the Cross-Validation run, we will not be using the scaled train and test sets but we apply the scaler in a pipeline just before the classifier.

1. **Cross-validation.**

We perform a 5-fold StratifiedKFold cross-validation using the cross\_val\_score that splits the dataset into 5 folds maintaining in each fold the class distribution of the original dataset. The scaler is applied through a pipeline which ensures that the scaler is applied only on the training subset of each fold. We compute the classification accuracy of each fold and the mean accuracy across the 5 folds, getting the following results.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Comparison with a simple 80/20 split.**

We can now compare the results of the K-Fold cross-validation with the results of a single classification using the train and test subsets created in step 1c. We get an accuracy value of 0.9333, which is the lowest value we got in any fold accuracy during the cross-validation.

This could be simply a matter of change for the particular split. Indeed if we change the random state of the data split to random\_state=0 we get an accuracy of 0.9667, equal to the mean accuracy of the K-Fold cross-validation. There should be a particular split that would lead to an accuracy of 1.000, the maximum accuracy observed in any fold. What we can conclude from this is that K-Fold validation benefit, is not the increased accuracy per se, but increasing the chance of getting an average one, avoiding lucky/unlucky splits that could lead to low/high accuracies, thus adding robustness to the process.

1. **Feature Selection**

We apply 4 different selection feature algorithms.

* 1. **Pearson correlation**

Pearson correlation feature selection is a filter-based technique that can be utilized to identify the most impactful features by calculating the correlation of the linear relationship between each feature and the target variable. Features with high absolute correlations, higher than a threshold value, are selected. This technique is simple, fast, and effective for detecting linear dependencies, but it may miss nonlinear relationships. This could be potentially addressed through feature engineering by creating new features from the powers or other functions of the original features.

In this project we set a threshold of 0.5, which leads to the selection of the following features:

*Selected features : ['max', 'min', 'range', 'kurtosis', 'crest\_factor']*

Training a Support Vector Classifier model with the subset of the original pre-processed dataset containing only these features leads to an accuracy of **0.9667** which is slightly higher than the original, simple training with all features (0.9333).

* 1. **Recursive Feature Elimination (RFE)**

Recursive Feature Elimination (RFE) is a wrapper technique for feature selection that involves repeatedly training a model and eliminating the least important features at each iteration, based on the model’s assessment of feature importance. In this project the method is implemented using a Support Vector Classifier (linear kernel) as the estimator model.

The RFE object is initialized and trained on the scaled train features and the corresponding train targets. From the trained object we can the extract the selected features to a list and slice the original pre-processed train and test features sub-sets with them. The selected features are:

*Selected features: ['crest\_factor', 'zero\_crossings', 'dominant\_freq', 'spectral\_centroid', 'spectral\_bandwidth']*

Similarly to the case of Pearson Correlation we can train a Support Vector Classifier with only these features and again get an accuracy value of **0.9667**.

* 1. **LASSO**

LASSO (Least Absolute Shrinkage and Selection Operator) is an example of an embedded method for feature selection. It performs feature selection as part of the model training process. Its working principle involves adding a penalty to the model that encourages the coefficients of less important features dent to reach zero and so they are effectively removed from the model. The main advantage of the method is that it automatically selects features in parallel to the training (doesn’t add an additional step to the process) which is important especially in high-dimensional datasets. On the other hand, LASSO may discard useful features in cases of highly correlated features. Contrary to the previous cases, with LASSO we use Logistic Regression with L1 regularization instead of SVC for feature selection. This is because Logistic Regression naturally performs sparsity and drives non-important features to zero.

After fitting the LASSO model we can extract the support of each feature and selected only those features with non-zero support. The selected features are:

*Selected features: ['crest\_factor', 'zero\_crossings', 'dominant\_freq', 'spectral\_centroid']*

Again, we can now slice the pre-processed train and test feature data based on this list of selected features and train an SVC model with those. The resulting accuracy is **0.9333**.

* 1. **Greedy feature elimination method using a Decision Tree**

Greedy feature elimination using a Decision Tree classifier is a wrapper-based feature selection method. It is an iterative method removing the least important feature based on the feature importances scores provided by a trained Decision Tree. In each iteration the model is retrained with the remaining features. In our implementation we start with all 16 features and proceed recalculating the accuracy after removing the least important feature until we are left with only a single feature. Plotting the accuracies for all iterations we get:

A graph with blue lines

AI-generated content may be incorrect.

We see that even with only the two or three most important features we can get a test accuracy of 1.0000. We can then see the two most important features as:

A screenshot of a computer

AI-generated content may be incorrect.

**Your observations and reflections**

Examining the results we get from all the above steps we can make some observations:

* K-Fold cross validation can be very easily implemented utilizing the available functions and modules in sklearn. Cross-validation can provide robustness to our analysis.
* In the table below we show the features selected by the different techniques

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Pearson** | **RFE** | **LASSO** | **Greedy** |
| **max** | **X** |  |  |  |
| **min** | **X** |  |  |  |
| **range** | **X** |  |  |  |
| **kurtosis** | **X** |  |  | **X** |
| **crest\_factor** | **X** | **X** | **X** |  |
| **zero\_crossings** |  | **X** | **X** |  |
| **dominant\_freq** |  | **X** | **X** |  |
| **spectral\_centroid** |  | **X** | **X** |  |
| **spectral\_bandwidth** |  | **X** |  |  |
| **skewness** |  |  |  | **X** |
| **std** |  |  |  |  |
| **rms** |  |  |  |  |
| **variance** |  |  |  |  |
| **spectral\_energy** |  |  |  |  |
| **spectral\_flatness** |  |  |  |  |
| **mean** |  |  |  |  |

We note the following:

* + No feature is selected by all 4 techniques
  + The is a “common agreement" among the different techniques that certain features (std, rms, variance, spectral\_energy, spectral\_flatness, mean) are not that impactful in the classification, so these can be ignored with some certainty.
  + There is a significant overlap between RFE and LASSO while Pearson Correlation give a significantly different set of features.

**Github Link**

[**https://github.com/GeSav/D7015B/tree/main/Assignment\_3**](https://github.com/GeSav/D7015B/tree/main/Assignment_3)