UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Enginyeria de Sistemes TIC

Predictive Maintenance

Network solutions for IoT

Albert Marquillas

September 6, 2025

Contents

1 Objectives

The purpose of this practical session is to explore predictive maintenance using machine learning techniques, specifically through the analysis of sensor data from industrial machinery. Students will apply exploratory data analysis (EDA), correlation analysis, and build predictive models using K-Nearest Neighbors (KNN) and Support Vector Machines (SVM).

2 Dataset description

The dataset used in this activity is available on Kaggle:

- **Title:** Machine Predictive Maintenance Classification
- **Source:** `https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification`

It contains measurements from a machine and the corresponding label indicating whether a failure has occurred. Each row represents a record of machine status and sensor readings.

- `Air temperature [K]`
- `Process temperature [K]`
- `Rotational speed [rpm]`
- `Torque [Nm]`
- `Tool wear [min]`
- `Machine failure` (Target: 0 = No failure, 1 = Failure)

3 Introduction to Machine Learning Models

3.1 K-Nearest Neighbors (KNN)

KNN is a simple, non-parametric algorithm used for classification and regression. In classification, a data point is assigned the class most common among its k nearest neighbors in the feature space. The distance (typically Euclidean) between data points is used to determine which are "closest". It is easy to implement and understand but can be sensitive to the choice of k and the scale of the features.

3.2 Support Vector Machines (SVM)

SVM is a supervised machine learning algorithm that finds the optimal hyperplane that best separates data points of different classes. In cases where data is not linearly separable, the SVM can use a kernel function (such as the RBF kernel) to map data into a higher-dimensional space. SVMs are powerful and often effective in high-dimensional spaces, though they may require careful parameter tuning.

4 Activity Structure

4.1 Initial Data Exploration (EDA)

- Load the dataset using Pandas.
- Display the first rows and summary statistics.
- Plot the class distribution of the target variable.
- Discuss whether the dataset is balanced or imbalanced.

Example code snippet:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('predictive_maintenance.csv')
print(df.head())
df['Machine␣failure'].value_counts().plot(kind='bar')
plt.title('Failure␣Distribution')
plt.show()
```

4.2 Correlation Analysis

- Compute the correlation matrix between variables.
- Visualize it using a heatmap.
- Identify which features are most correlated with machine failure.

Example code snippet:

```
import seaborn as sns

corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

4.3 Data Preparation

- Separate features and target.
- Split the data into training and testing sets.
- Apply standard scaling to normalize the data.

4.4 Classification with KNN

- Train a KNN classifier using $k = 5$.
- Evaluate model performance on the test set.
- Show confusion matrix and accuracy score.

Example code snippet:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

X = df.drop(['Machine failure'], axis=1)
y = df['Machine failure']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    ↪ random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)

print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

4.5 Classification with SVM

- Train an SVM classifier using the RBF kernel.
- Compare results with KNN.

Example code snippet:

```
from sklearn.svm import SVC

svm = SVC(kernel='rbf')
svm.fit(X_train_scaled, y_train)
y_pred_svm = svm.predict(X_test_scaled)

print(accuracy_score(y_test, y_pred_svm))
print(confusion_matrix(y_test, y_pred_svm))
```

5 Optional extensions

- Try different values of k for KNN.
- Use GridSearchCV to optimize hyperparameters.
- Apply SMOTE to balance the dataset.
- Try other classifiers like Logistic Regression or Random Forest.

6 Conclusion and questions

- Which model performed better? Why?
- What are the limitations of this dataset?
- What kind of preprocessing or feature engineering could improve the models?