Master's degree in Computer Engineering for Robotics and Smart Industry - a.y. 2022/2023
Machine Learning and Artificial Intelligence

Smoke detector: implementation and comparison of machine learning methods for binary classification

Lorenzo Busellato[1] and Daniele Nicoletti[2]

[1]VR472249 - lorenzo.busellato_02@studenti.univr.it
[2]VR472590 - daniele.nicoletti@studenti.univr.it

Abstract

Accurate and reliable smoke and fire detection remains an open problem in machine learning. State of the art deep learning approaches perform better than classical machine learning approaches but suffer from a lack of extensive and publicly available datasets.

In this project, a comparison of classical machine learning binary classifier models, both supervised (Naive Bayes, K-Nearest Neighbors, Support Vector Machines and Multi-Layer Perceptron) and unsupervised (K-Means clustering), is presented. The models are trained and tested on a publicly available dataset of readings from a net of IOT sensors.

Two dimensionality reduction methods, Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), are implemented and compared. Each model undergoes a procedure of hyperparameter tuning, producing a ”best” model. Figures for accuracy, precision, recall, FAR (False Alarm Rate) and F1-score as well as the confusion matrix are provided for the best models, which are then compared.

It is found that, among the tested feature selection-model-hyperparameter combinations, K-Nearest Neighbors using PCA for dimensionality reduction, using the Euclidean distance metric with $K = 7$ and weights assigned by distance achieves the best performance in terms of F1-score and FAR.

Contents

LIST OF FIGURES

LIST OF TABLES

Master's degree in Computer Engineering for Robotics and Smart Industry - a.y. 2022/2023
Machine Learning and Artificial Intelligence

Smoke detector: implementation and comparison of machine learning methods for binary classification

I. MOTIVATION AND RATIONALE

Smoke detectors are an essential fire safety measure that has helped sharply decrease fire-related casualties by 65% in Europe over the last three decades [1]. An open problem with smoke detectors is their reliability. For instance, in the year 2020 in England, the majority of false alarms was 'due to apparatus' [2], i.e. due to equipment error. False alarms cause a waste of resources of the rescue services, as well as being a nuisance. Furthermore, due to the increasing number of installed smoke detectors, the number of reported false alarms is increasing. Therefore methods that more efficiently detect fires, minimising the number of false alarms, are needed.

II. STATE OF THE ART

State of the art techniques for smoke and fire detection, according to [3], are evolving along three main research directions:

- Smoke classification;
- Smoke segmentation;
- Smoke localisation by estimating the bounding boxes in images and videos.

Most of the methods in the smoke and fire detection field make use of images and/or videos, and these types of data are often combined with other environmental ones to improve the accuracy of the detection. The methods referred in [3] are considering only the **outdoor** smoke detection scenario and for each of the previous categories are reported methodologies using machine learning and image processing as well as methods implementing deep learning techniques. In general, with the growing application of deep learning techniques (eg. Convolutional Neural Networks) it is possible to achieve better results w.r.t traditional machine learning and image processing techniques by improving performances and reducing FAR (False Alarm Rate), in each of the three branches. However, not many publicly available datasets are present for this research field, so deep learning models are penalized due to the scarcity of available data.

III. OBJECTIVES

The main objective of this project is the implementation, test and most importantly the comparison of several machine learning techniques for binary classification, going from supervised learning (Naive Bayes, K-Nearest Neighbors, and Support Vector Machines) and unsupervised learning (K-Means clustering) approaches to an approach using neural networks.

A secondary objective is the implementation of two different feature dimensionality reduction methods, namely Principal Component Analysis and Linear Discriminant Analysis, to highlight the relationship between feature dimensionality reduction method and model performance.

Finally, the last objective of this project is to determine which combination of feature dimensionality reduction, model and model parameters produces the best classifier in the context of smoke detection.

IV. METHODOLOGY

A. Dataset

The dataset picked for this project is the Smoke Detection Dataset[4]. The dataset is a collection of more than 60.000 readings produced from a network of IOT devices in a number of different scenarios:

- Normal indoor
- Normal outdoor
- Indoor wood fire
- Indoor gas fire
- Outdoor wood, coal, gas grill
- Outdoor with high umidity
- etc.

The sensor readings constitute the dataset features, in particular:

- Air temperature (Temperature)
- Air humidity (Humidity)
- Total Volatile Organic Compounds (TVOC), measured in ppb
- Carbon dioxide equivalent concentration (eCO2), measured in ppm
- Raw molecular hydrogen concentration (Raw H2)
- Raw ethanol concentration (Raw Ethanol)
- Air pressure (Pressure), measured in hPa.
- Densities of particles, sizes $< 1.0\mu m$ (PM1.0) and $< 2.5\mu m$ (PM2.5), measured in $\mu g/m^3$
- Number of particles, sizes $< 0.5\mu m$ (NC0.5), $< 1.0\mu m$ (NC1.0) and $< 2.5\mu m$ (PM2.5)
- Sample counter (CNT)
- UTC timestamp (UTC)
- Fire alarm, binary value (0 for no alarm, 1 for alarm) that constitutes the ground truth

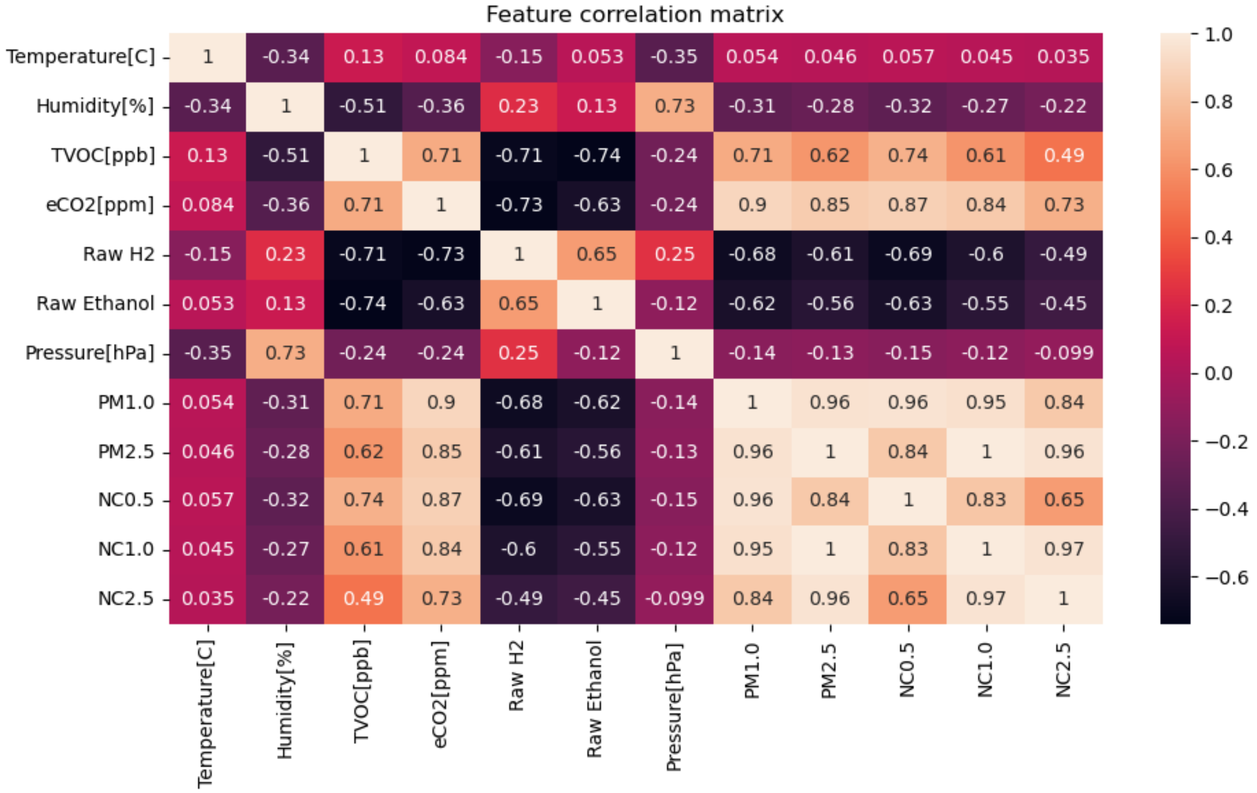

Fig. 1. Feature correlation matrix

Of the available features, the sample counter CNT and the UTC timestamp will be ignored. This is because they carry little to no meaningful information about the task. UTC would be more meaningful if the readings were evenly distributed across the year, but in this case they were all taken within the same month.

A problem with the ”raw” dataset is that classes are not balanced, since there are about 45.000 samples for class 1 and about 18.000 for class 0. To resolve the situation, undersampling was used, i.e. from the samples relative to class 1 only a number equal to the samples of class 0 was picked. Since the resulting dataset still contains a good amount of samples, around 18.000 for each class, this was not judged too problematic. A better option would have been to oversample the lacking class, but this of course was not possible in this case.

Before they could be used, the features also needed to be scaled in order to have their values be in the same range. This was done with min-max scaling, which brought each feature’s range to $[0, 1]$.

Another consideration is about the statistical correlation between features. By constructing the correlation matrix (figure 1), which makes apparent the high statistical correlation between the various features regarding particle number (NCx) and density (PMx). This is not an issue since, before using them, the features will undergo a process of dimensionality reduction, which has the secondary effect of removing the components of the data that are strongly correlated to each other.

The dataset was split into a training set, made up of 80% of the samples, and a test set, made up of the remaining 20% of samples.

B. *Principal Component Analysis*

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of a dataset. It does so by mapping linearly the data into a new coordinate system, maintaining most of the variation of the original dataset.

The algorithm is as follows:

1) Subtract from each feature vector x^k, $k = 1, \dots, n$ the mean:
$$\mu_k = \frac{1}{n}\sum_{k=1}^{n} x^k \tag{1}$$
thus obtaining the centered data x_c^k.
2) Compute the covariance matrix C of the centered data.
3) Compute the eigenvalues and eigenvectors of C.
4) Sort the eigenvalues from largest to smallest.
5) Construct a transformation matrix T whose columns are the first N eigenvectors corresponding to the largest N eigenvalues.
6) Compute the projected data ω^k by multiplying each x_c^k by T.

The choice of N is related to the amount of variance individually carried by the eigenvectors. Typically N is such that at least 80% of the variance is kept.

C. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method for dimensionality reduction that aims at reducing the number of features while preserving as much as the discrimination between classes as possible.

Given a set of n samples $x_1, \dots, x_n$, of which n_1 are classified as ω_1 and n_2 are classified as ω_2, the aim is to find a projection of the features on a straight line whose direction better separates the classes. The separation is measured as the distance between the means of the samples belonging to the two classes. Once the function is found, the samples are projected as scalars on a line, i.e. $y = w^T x$. Such a projection is called Fisher's transform, and is defined as the linear function that maximizes the criterion function:

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2} \tag{2}$$

where $\tilde{\mu}_i$ is the projected mean values of feature i:

$$\tilde{\mu}_i = \frac{1}{n} \sum_{y \in \omega_i} y = \frac{1}{n} \sum_{x \in \omega_i} w^T x = w^T \mu_i \tag{3}$$

and $\tilde{s}_i^2$ are the scatters of feature i, which are equivalent to the variance:

$$\tilde{s}_i^2 = \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2 \tag{4}$$

Since:

$$\begin{aligned} \tilde{s}_i^2 &= \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2 = \sum_{x \in \omega_i} (w^T x - w^T \mu_i)^2 \qquad (5) \\ &= \sum_{x \in \omega_i} w^T (x - \mu_i)(x - \mu_i)^T w = w^T S_i w \\ &\implies \tilde{s}_1^2 + s_2^2 = w^T (S_1 + S_2) w = w^T S_w w \end{aligned}$$

where S_w is called within-class scatter matrix, and:

$$\begin{aligned} (\tilde{\mu}_1 - \tilde{\mu}_2) &= (w^T \mu_1 - w^T \mu_2)^2 \qquad (6) \\ &= w^T (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w \\ &= w^T S_w w \qquad (7) \end{aligned}$$

where S_w is called between-class scatter matrix, Fisher's transform is therefore the function w^* such that:

$$w^* = \underset{w}{argmax} \frac{w^T S_B w}{w^T S_w w} \tag{8}$$

which is finally used to project the features onto a reduced space of dimension 1 (in the case of binary classification).

D. Naive Bayes

Naive Bayes refers to a set of classifiers based on Bayes' theorem on conditional probability.

Given a feature vector x, Bayes' theorem states that its conditional probability on the class variable y_k is:

$$p(y_k \mid x) = \frac{p(y_k) p(x \mid y_k)}{p(x)} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}} \tag{9}$$

Since the evidence depends only on the features, the denominator is constant and acts only as a scaling factor. Under the "naive" assumption, given the class variable all features in x are mutually independent, so the posterior probability can be expressed as:

$$p(y_k \mid x_1, \dots, x_n) = \frac{1}{p(x)} p(y_k) \prod_{i=1}^{n} p(x_i \mid y_k) \tag{10}$$

where $p(x) = \sum_k p(y_k) p(x \mid y_k)$.

Naive Bayes classifiers combine the above model for the probability distribution of x with a decision rule, typically the maximum a posteriori rule:

$$\hat{y} = \underset{k \in 1, \dots, K}{argmax} \, p(y_k) \prod_{i=1}^{n} p(x_i \mid y_k) \tag{11}$$

The rule assigns to the vector x the class variable $\hat{y}$ that maximizes the posterior probability.

To apply the classifier, the likelihood $p(x \mid y_k)$ must be defined. This can be done in a number of ways, but in this project it is assumed that each feature is normally distributed.

Therefore the likelihood for the feature vector x is modelled as a Gaussian distribution:

$$p(x \mid y_k) = \frac{1}{\sqrt{2\pi} \sigma_k} e^{-\frac{(x - \mu_k)^2}{2\sigma_k^2}} \tag{12}$$

where μ_k and σ_k^2 are the mean and variance of the distribution.

Classification is finally obtained by computing the posteriors for each class for a given new feature vector x, which will be assigned to the class which produces the highest posterior probability.

E. K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a supervised learning method used for classification.

Given a set of training feature vectors, a positive integer K and a test feature vector to be classified, the algorithm works by first selecting the k training feature vectors closest to the test feature vector according to some distance metric. The test feature vector is then assigned to the class most common among the k closest training feature vectors.

The value of k can be determined by plugging into the algorithm different values and comparing the resulting performance, which is the method used in this project. A common heuristic is to choose k equal to the square root of the number of samples.

In this project the Minkowski metric is used. Given two vectors $x = (x_1, \dots, x_i, \dots, x_n)$ and $y = (y_1, \dots, y_i, \dots, y_n)$, the Minkowski metric is defined as:

$$D(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{13}$$

with $p = 1, 2, 3$.

In the case of $p = 1$, the metric is equal to the Manhattan (or Cityblock) distance:

$$D(x,y) = \sum_{i=1}^{n} |x_i - y_i| \tag{14}$$

In the case of $p = 2$, the metric is equal to the $\mathbb{R}^2$ Euclidean distance:

$$D(x,y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{15}$$

Another thing to consider is the weight to assign to each closest neighbor's vote. Weights can be assigned uniformly, with every neighbor's vote being equally weighted, or based on distance, with the closest neighbors' votes being weighted higher than the farthest neighbors'. In this project, both approaches will be tested.

F. Support Vector Machines

Support Vector Machines (SVMs) are supervised learning models used in classification.

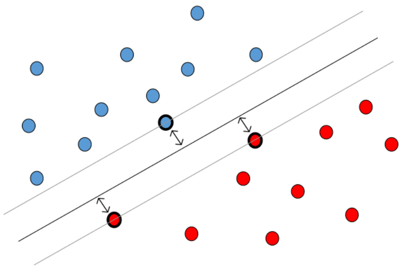

Fig. 2. SVM in the linear case (image source [5])

In binary classification of linearly separable data, there exists at least an hyperplane of dimension $(p-1)$, where p is the dimension of the data, that separates the classes.

The "best" hyperplane is found by maximizing the margin, which is the sum of the distances from the hyperplane of the feature vectors closest to it. The closest feature vectors are called support vectors.

The hyperplane is found by solving the following quadratic optimization problem:

$$L_D(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{16}$$

with:

$$\alpha_i \geq 0, \forall i = 1, \dots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{17}$$

The tuple (x_i, y_i) indicates the feature-label pair associated to the i-th support vector, while the α terms are the dual variables of the optimization problem.

Classification of a new vector x is obtained by computing:

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i \implies f(x) = sign(w \cdot x + b) \tag{18}$$

where b is computed from:

$$\alpha_i (y_i (w \cdot x_i + b) - 1) = 0 \tag{19}$$

In the non-linear case, a simple approach is to introduce slack variables to the optimization problem in order to allow vectors to cross the margin and therefore be misclassified. This introduces a parameter C in the model which represents the sensitivity of the classifier to misclassification and its generalization performance.

Another strategy is to find a mapping φ that maps the feature space to an higher-dimensional space, in which linear separation may be possible or easier to compute. However if the feature vectors already lie on an high-dimensional space, actually computing the mapping is computationally expensive. By observing that in the optimization problem the feature vectors appear only as a dot product of each other, by finding a function:

$$K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j) \tag{20}$$

it is possible to train the model without explicitly computing φ. Such a function is called kernel, and the strategy is therefore called kernel trick. The kernel functions used in this project are the following:

$$\begin{aligned} \text{Linear} \quad & K(x,z) = \langle x, z \rangle && (21) \\ \text{Polynomial} \quad & K(x,z) = (\langle x, z \rangle + 1)^p && (22) \\ \text{Radial Basis Function} \quad & K(x,z) = e^{-\frac{|x-z|^2}{2\sigma^2}} && (23) \\ \text{Sigmoid} \quad & K(x,z) = tanh(a\langle x, z \rangle + b) && (24) \end{aligned}$$

G. K-Means clustering

K-Means clustering is an unsupervised learning method used for classification.

Given a set of samples $x_1, \dots, x_n$, the idea is to partition the set into k subsets, called clusters, in such a way that each cluster's variance is minimized. The clusters are centered around the mean, or centroid, of the samples they contain.

Given a set of initial centroids $m_1^{(1)}, \dots, m_k^{(1)}$, the algorithm iterates between two steps:

1) Assignment: each sample is assigned to the cluster $S_i^{(t)}$ with the nearest mean, i.e. with the smallest squared Euclidean distance:

$$S_i^{(t)} = \left\{ x_p \mid \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2, \forall j, 1 \leq j \leq k \right\} \tag{25}$$

2) Update: the centroids of the clusters are recomputed:

$$m_i^{(t+1)} = \frac{1}{\#\{S_i^{(t)}\}} \sum_{x_j \in S_i^{(t)}} x_j \tag{26}$$

The algorithm converges when there are no new assignments made in the first step.

Classification is done simply by assigning to an unknown sample the class corresponding to the cluster it belongs to.

H. Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a feedforward neural network model consisting in a series of layers of nodes, each fully connected to the following one. The nodes of the layers are perceptrons with nonlinear activating functions.

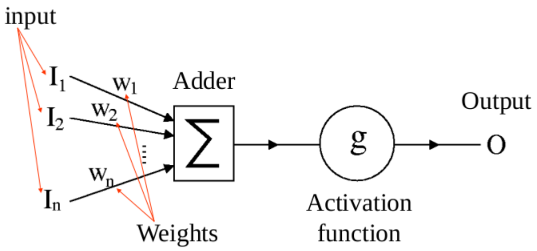

Fig. 3. Perceptron model (image source [5])

The output of the neuron is given by the activation function g evaluated with the weighted sum of the inputs to the perceptron:

$$O = g\left(\sum_{i=1}^{n} w_i I_i\right) \tag{27}$$

The activation functions tested in this project are:

$$\text{Logistic} \quad g(x) = \frac{1}{1+e^{-x}} \tag{28}$$

$$\text{Hyperbolic tangent} \quad g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{29}$$

$$\text{Rectified Linear Unit (ReLU)} \quad g(x) = \begin{cases} x & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

The network is structured in layers. The first layer, which has as many neurons as the dimension of the feature space, is the input layer. The last layer, which has an output neuron for each class, is the output layer. Between them there may be one or more hidden layers. Each layer is fully connected to the following one.

The number of hidden layers is determined by the problem. If the data is linearly separable, a network with no hidden layers can represent the decision boundary. Adding hidden layers improves the network's ability to approximate the decision boundary for non-linearly separable data. Theoretically, only two hidden layers are needed for the network to be able to approximate any decision boundary. To determine the number of neurons in each hidden layer, a number of rule-of-thumb methods can used[6]. In this project, in order to reduce training and testing times, a small set of hidden layer number and sizes will be tested.

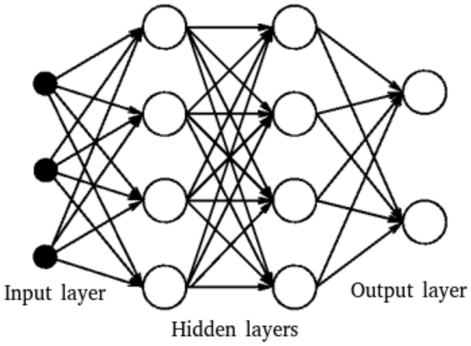

Fig. 4. MLP network structure (image source [5])

The training of the network consists in adjusting the weights of the network in order to improve the resulting accuracy, or in other words it is a process of minimization of some error function, called loss function, with respect to the network's weights. The minimization of the loss function involves the computation of its gradient, which is computationally expensive. A procedure called backpropagation is introduced in order to optimize this computation. Backpropagation works by exploiting the chain rule of derivation, computing the gradient one layer at a time starting from the output layer and then iterating backwards, avoiding redundant calculations.

In general terms, the update to the weight of the i-th node in the j-th layer is given by:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{31}$$

where E is the loss function and η is an hyperparameter called learning rate, which influences the strength of the correction applied to the weight.

V. Implementation Details

This project's code is available on GitHub[7]. Instructions for how to run the code and interpret the results are included in the repository's README. All the previously described methods for feature selection and for classification are implemented in Python using scikit-learn[8].

VI. Experiments and Results

A. Hyperparameter tuning

All of the classifier models previously discussed, except Naive Bayes and K-Means clustering, have a number of hyperparameters that need to be chosen.

To perform hyperparameter tuning, a grid search approach is used. A list of possible parameter values is constructed for each model, which is then trained and tested with every possible combination of the parameters in the list.

To improve the generalization ability of the models, i.e. to avoid overfitting, a cross-validation procedure is used. The training data is split into 5 folds, then each of the folds is used as test data, while the remaining 4 are used as training

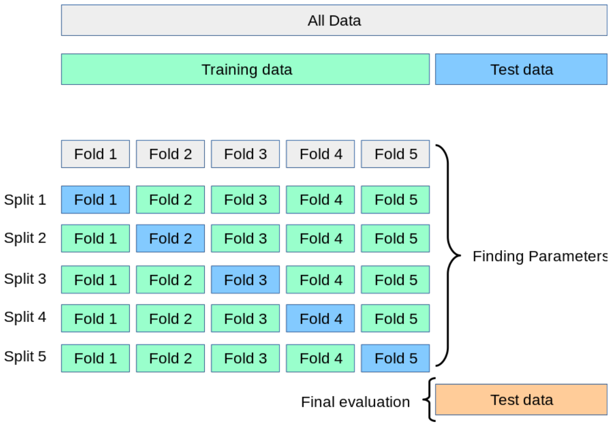

Fig. 5. Dataset splits for cross-validation (image source [9])

data. The model is therefore trained and tested 5 times on 5 ”different” training sets for each parameter combination. Each split is scored, and the global score for the parameter combination is obtained by averaging the scores of each split.

The scoring metric used for cross-validation is the F1-score (described in section VI-B).

The best parameter combination therefore is the one that maximizes the average score in the cross-validation procedure. Note that the combination is not the global optimum, but only the optimum among the possible combinations of parameters supplied to the procedure.

B. Evaluation metrics

A first visual representation of model performance is the confusion matrix. Given the number of true positives TP, the number of false positives FP, the number of true negatives TN and the number of false negatives FN, the confusion matrix is defined as:

$$C = \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \tag{32}$$

The computed metrics for the models are accuracy, F1-score and False Alarm Rate (FAR).

Accuracy is the ratio of correct predictions over total number of predictions made:

$$a = \frac{TP + TN}{TP + FP + TN + FN} \tag{33}$$

The F1-score is the harmonic mean of precision and recall:

$$F1 = 2\frac{P \cdot R}{P + R} \tag{34}$$

Precision is the ratio between correct positive predictions and total number of positive predictions:

$$P = \frac{TP}{TP + FP} \tag{35}$$

Recall is the ratio between correct positive predictions and total number of predictions that should have been predicted as positive:

$$R = \frac{TP}{TP + FN} \tag{36}$$

The False Alarm Rate (FAR) is the ratio of false positives and total number of predictions that should have been predicted as positive:

$$FAR = \frac{FP}{FP + TN} \tag{37}$$

Comparison between models will be carried out in terms of accuracy and F1-score, which should have higher values for the best performing model, and FAR, which should have lower values for the best performing model.

C. Naive Bayes

By comparing the confusion matrices for the PCA features case (figure 6) and the LDA features case (figure 7), it is immediately apparent that the model using LDA features performs far better than the one using PCA features. In particular, the model with PCA features has very low recall for class 0 (no alarm), and this results in a high amount of detected false positives, thus greatly increasing the FAR.

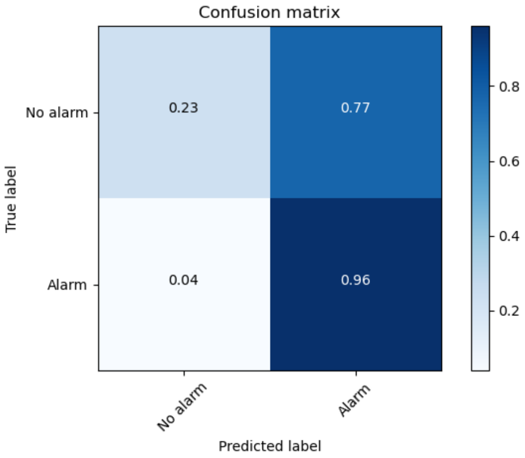

Fig. 6. Naive Bayes with PCA features - normalized confusion matrix

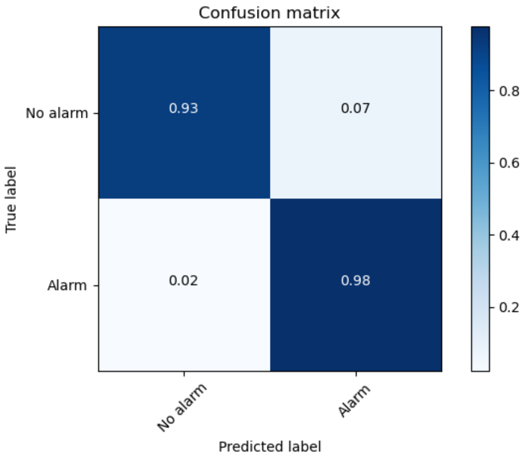

Fig. 7. Naive Bayes with LDA features - normalized confusion matrix

TABLE I
NAIVE BAYES WITH PCA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.85	0.23	0.36
1	0.55	0.96	0.70

accuracy	0.59
FAR	0.769

TABLE II
NAIVE BAYES WITH LDA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.98	0.93	0.95
1	0.93	0.98	0.95

accuracy	0.95
FAR	0.073

The best performing Naive Bayes model therefore is the one using LDA features, achieving an accuracy of about 95% and a FAR of about 7%.

The difference in performance could be due to the shape of the distributions of the PCA features in the reduced space, which is probably not close enough to a Gaussian distribution. Therefore the model incorrectly estimates the likelihood probability, affecting accuracy. This idea is indirectly confirmed by the model using LDA features: since a consequence of using Fisher's transform is that the resulting feature is normally distributed, the Naive Bayes classifier correctly estimates the likelihood probability, thus improving accuracy.

D. K-Nearest Neighbors

The parameter ranges used during hyperparameter tuning are:

- K: 50 values evenly spaced in the range 1-300
- distance metric: Cityblock, Euclidean, Minkowski
- sample weights: uniform and based on distance

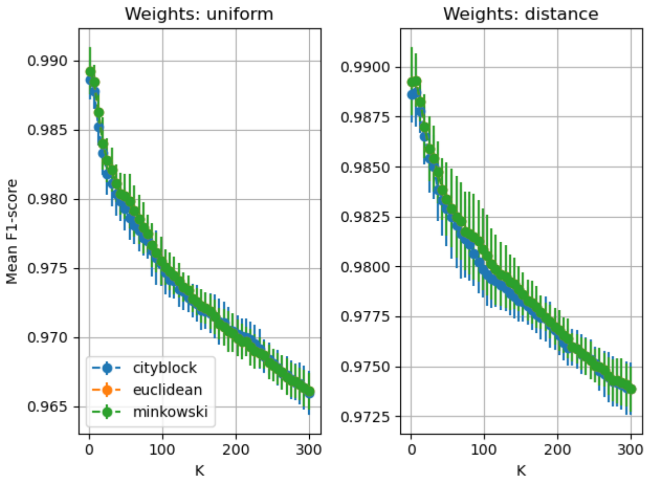

Fig. 8. KNN with PCA features - hyperparameter tuning

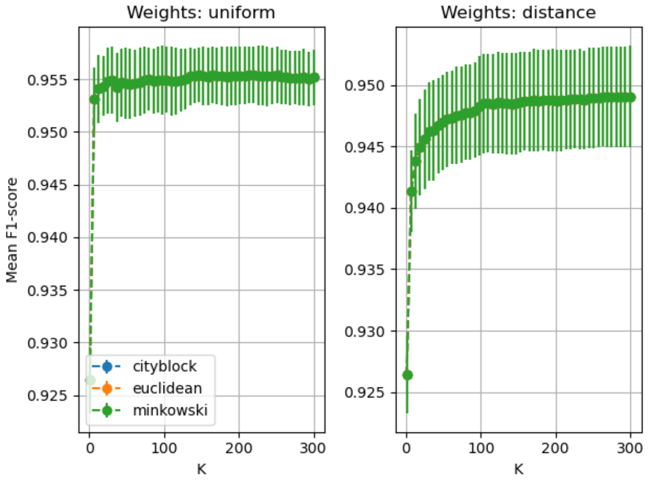

Fig. 9. KNN with LDA features - hyperparameter tuning

The hyperparameter tuning procedure shows that, independently from the used feature selection method, the distance metric used doesn't have a great influence on model performance. Cityblock seems to perform slightly worse than Euclidean or Minkowski, which perform exactly the same.

TABLE III
KNN WITH PCA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	1.00	0.97	0.98
1	0.97	1.00	0.98

accuracy	0.98
FAR	0.034

TABLE IV
KNN WITH LDA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.99	0.91	0.95
1	0.92	0.99	0.95

accuracy	0.95
FAR	0.087

The best model with PCA features is with $K = 7$, weights by distance and Euclidean metric, while with LDA features the best model is with $K = 220$, uniform weights and Cityblock metric. The best performing model of the two is the one with PCA features, which achieves an higher accuracy and a FAR 2.5 times lower than the one with LDA features

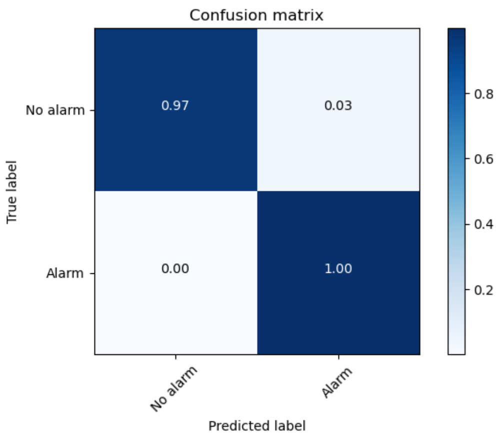

Fig. 10. KNN with PCA features - normalized confusion matrix

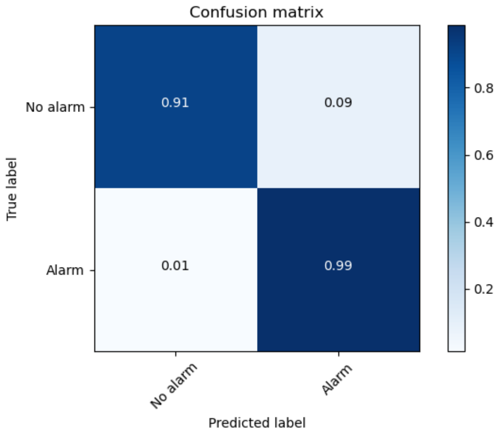

Fig. 11. KNN with LDA features - normalized confusion matrix

E. Support Vector Machines

The parameter ranges used during hyperparameter tuning are:

- kernel: linear, polynomial, rbf, sigmoid
- regularization parameter C: 15 values evenly spaced in the range 0.001-1

Hyperparameter tuning shows that the best performing kernels are rbf and linear. In the PCA case, rbf performs far better than every other kernel, while in the LDA case they perform essentially the same. In both cases it seems that performance increases when the regularization parameter C increases, until a plateau.

The best model with PCA features is with $C = 1$ and with the rbf kernel, while the best model with LDA features is with $C = 0.144$ and the linear kernel. The best performing model is the one using LDA features, which achieves a slightly higher accuracy and a lower FAR than the one using PCA features.

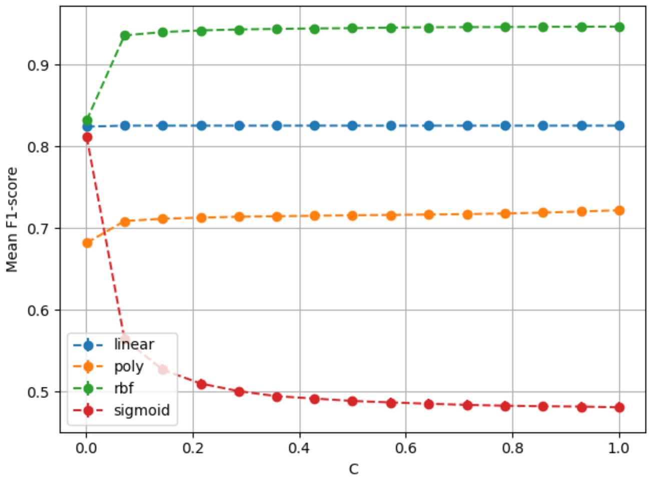

Fig. 12. SVM with PCA features - hyperparameter tuning

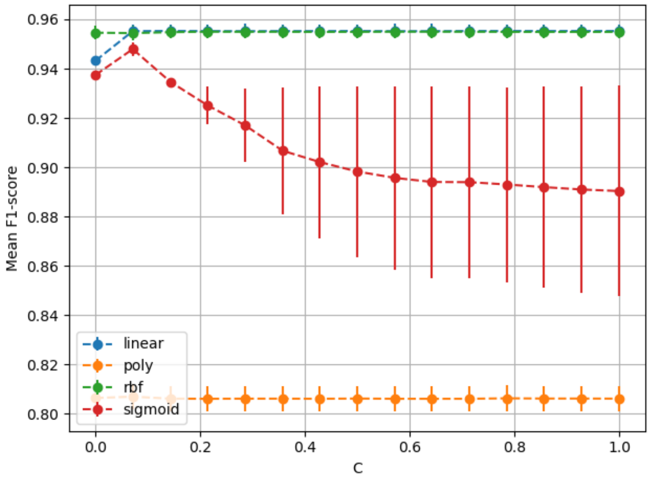

Fig. 13. SVM with LDA features - hyperparameter tuning

TABLE V
SVM WITH PCA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	1.00	0.89	0.94
1	0.90	1.00	0.94

accuracy	0.94
FAR	0.113

TABLE VI
SVM WITH LDA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.98	0.91	0.95
1	0.92	0.99	0.95

accuracy	0.95
FAR	0.086

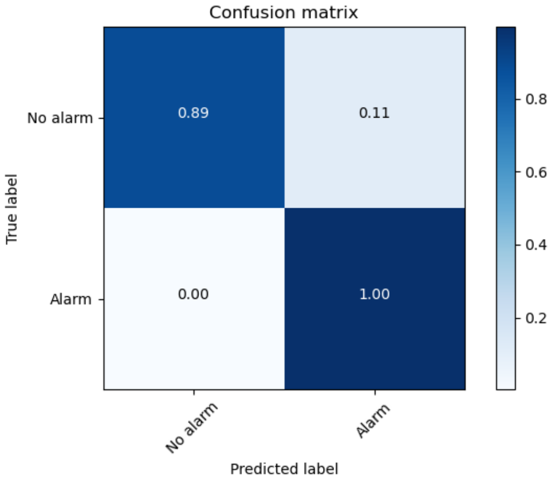

Fig. 14. SVM with PCA features - normalized confusion matrix

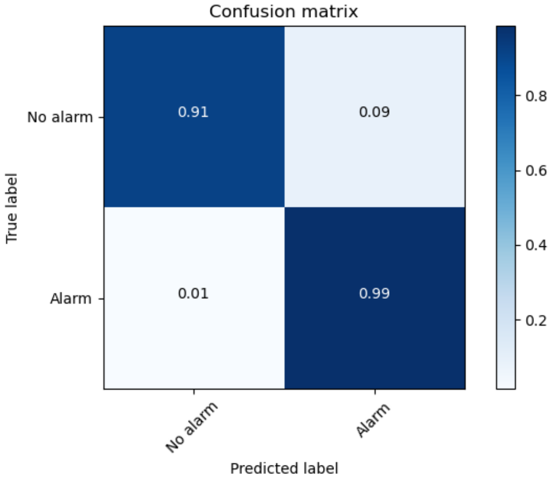

Fig. 15. SVM with LDA features - normalized confusion matrix

F. K-Means clustering

TABLE VII
K-MEANS WITH PCA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.50	0.97	0.66
1	0.00	0.00	0.00

accuracy	0.49
FAR	0.025

TABLE VIII
K-MEANS WITH LDA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.99	0.87	0.93
1	0.88	0.99	0.93

accuracy	0.93
FAR	0.129

The best model using LDA features achieves an accuracy of 0.93 and a FAR of 0.129. The best model using PCA features performs far worse, because while it achieves a very low FAR of 0.025, it also has a very low accuracy of 0.49. The reason behind this difference in performance is apparent by comparing the result of clusterization to the ground truth (figures 16 and 17).

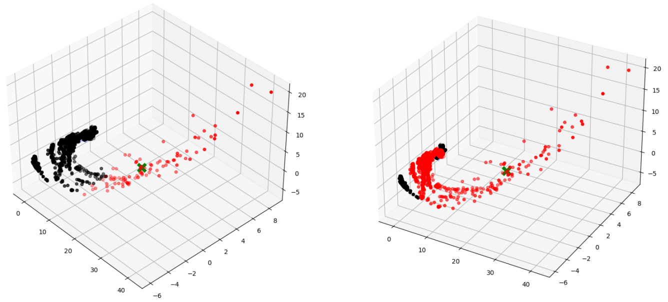

Fig. 16. K-Means with PCA features - clusters (left) vs ground truth (right)

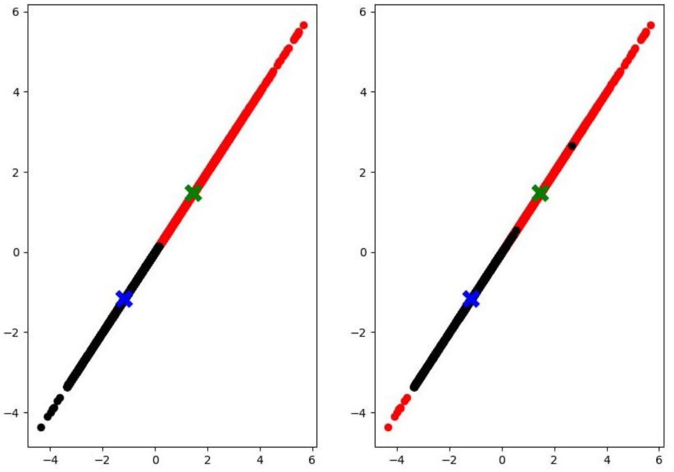

Fig. 17. K-Means with LDA features - clusters (left) vs ground truth (right)

In the PCA case the algorithm fails, because it is not possible to find two spherical clusters with similar variance, since the data points belonging to class 0 are grouped in two groups on either side of the big cluster of data points belonging to class 1. In the LDA case this is not the case, as could be expected since Fisher's transform main idea is to project the feature space into a reduced space in which class separation is the highest possible.

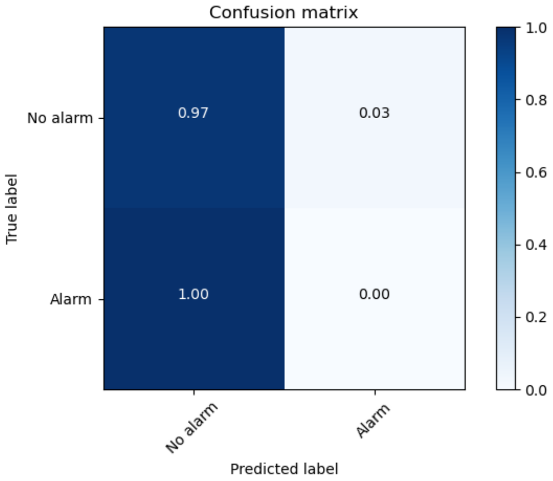

Fig. 18. K-Means with PCA features - normalized confusion matrix

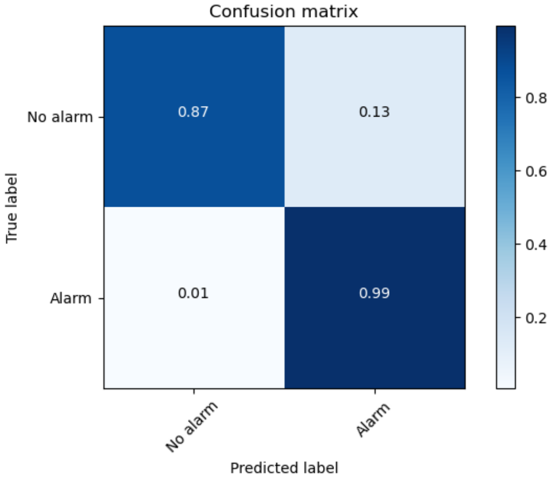

Fig. 19. K-Means with LDA features - normalized confusion matrix

G. Multi-Layer Perceptron

The parameter ranges used during hyperparameter tuning are:

- activation function: logistic, tanh and ReLU
- learning rate η: 0.001-0.01-0.1-1-10
- hidden layers: none, one with 2 nodes, one with 4 nodes, 2 with 2 nodes each, 2 with 4 nodes each

TABLE IX
MLP WITH PCA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.99	0.91	0.94
1	0.91	0.99	0.95

accuracy	0.95
FAR	0.094

TABLE X
MLP WITH LDA FEATURES - CLASSIFICATION REPORT

class	precision	recall	F1-score
0	0.99	0.91	0.95
1	0.92	0.99	0.95

accuracy	0.95
FAR	0.089

The best performing model with PCA features is with $\eta = 0.01$, the ReLU activation function and with two hidden layers with 4 nodes each. The best performing model with LDA features is with $\eta = 0.01$, the hyperbolic tangent activation function and with a single hidden layer with 4 nodes. The models both achieve high accuracy and low FAR, with the FAR of the model using LDA features being slightly lower than the one using PCA features.

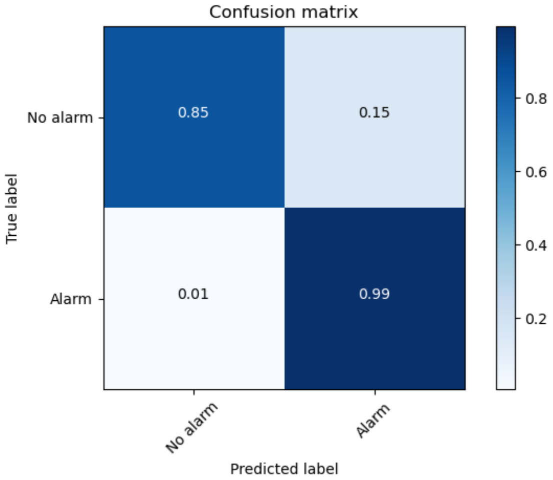

Fig. 20. MLP with PCA features - normalized confusion matrix

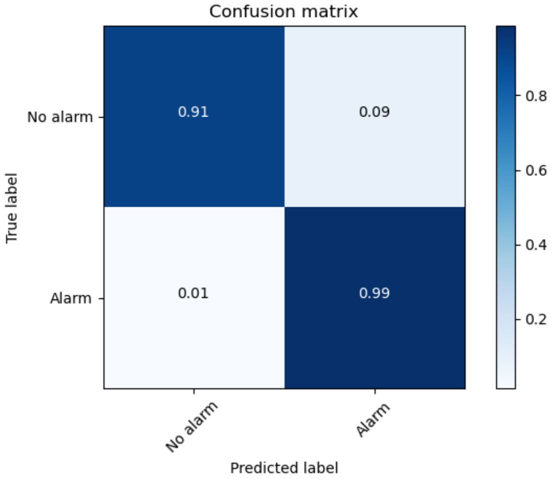

Fig. 21. MLP with LDA features - normalized confusion matrix

H. Model comparison

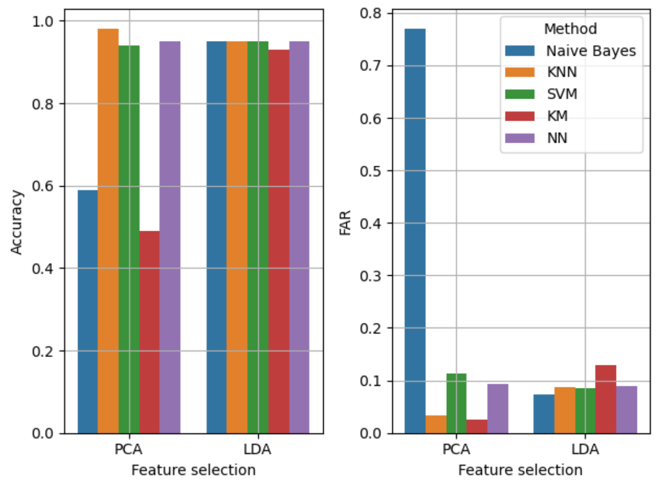

Fig. 22. Model comparison

All the discussed results are summarized in figure 22. It is immediately apparent that using LDA for dimensionality reduction results in a much more predictable performance than using PCA. Furthermore, using LDA features consistently produces classifiers with high accuracy and low FAR. This is due to the fact that LDA collapses the feature space on a single dimension, in which finding a decision boundary between the classes becomes much easier. Some models however can reach a sharp decrease of the FAR when using PCA features, in one case (KNN) even improving accuracy by about 3%.

By the comparison it appears then that the best performing model is the K-Nearest Neighbors classifier using PCA for dimensionality reduction, using the Euclidean distance metric with $K = 7$ and assigning weights to the samples based on their distance.

VII. Conclusions

The main purpose of this project was to obtain a comprehensive comparison between a number of classical machine learning methods for binary classification in the context of smoke detection, namely Naive Bayes, K-Nearest Neighbors, Support Vector Machines, K-Means clustering and Multi-Layer Perceptron. This was achieved by using a procedure of hyperparameter tuning with cross-validation, which produced a view on how the hyperparameters chosen influence each model, as well as allowing for between-model comparisons. The results indicate that, in the context of smoke detection, K-Nearest Neighbors slightly outperforms the other models in terms of accuracy and False Alarm Rate. The second best model was found to be the MLP classifier, whose performance however probably suffered from the limited amount of hyperparameter combinations explored in this project.

The secondary purpose was to implement two feature dimensionality reduction methods, Principal Component Analysis and Linear Discriminant Analysis, in order to highlight their impact on model performance. The results showed that by using PCA the resulting models, except for KNN, had on average lower accuracy and lower FAR. The usage of LDA produced a much more predictable performance in the models, which all produced accuracies around 0.95 and FARs around 0.09.

The last purpose of this project was to determine which model, feature dimensionality reduction method and hyperparameter set combination produced the best performing classifier in the context of smoke detection. The resulting model was the K-Nearest Neighbors algorithm using PCA for dimensionality reduction, using the Euclidean distance metric with $K = 7$ and assigning weights by distance. The model achieved an accuracy of 98% and a FAR of 3.4%.

References

[1] Fire safety statistics. https://www.modernbuildingalliance.eu/fire-safety-statistics/. Accessed: 2022-12-2.

[2] Home office statistics highlight fire false alarms remain an issue. https://www.bafe.org.uk/bafe-news/home-office-statistics-highlight-fire-false-alarms-remain-an-issue. Accessed: 2022-12-2.

[3] Shubhangi Chaturvedi, Pritee Khanna, and Aparajita Ojha. A survey on vision-based outdoor smoke detection techniques for environmental safety. *ISPRS Journal of Photogrammetry and Remote Sensing*, 185:158–187, 2022.

[4] Smoke detection dataset. https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset. Accessed: 2022-12-2.

[5] V. Murino. Lecture slides of the machine learning and artificial intelligence course, master's degree in computer engineering for robotics and smart industry, 2022.

[6] J. Heaton. *Introduction to Neural Networks with Java*, pages 158–159.

[7] Smoke detector. https://github.com/Salazar99/Smoke-detector.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Cross-validation. https://scikit-learn.org/stable/modules/cross_validation.html. Accessed: 2023-1-18.