BIRZEIT UNIVERSITY

Faculty of Engineering and Technology Department of Electrical and Computer Engineering

ENCS5141—Intelligent Systems Laboratory

Case Study #2—Comparative Analysis of Classification Techniques: Support Vector Machines (SVM) and Multilayer Perceptron (MLP).

Prepared by: Dunia Jaser - 1201345

Instructor: Dr. Mohammad Jubran

Assistant: Eng. Hanan Awawdeh

Date: April 26, 2024

Abstract

This case study aims to compare the performance of Support Vector Machines (SVM) and Multilayer Perceptrons (MLP) in classifying banknotes based on their currency, denomination, and orientation. The goal is to identify which of the two popular machine learning algorithms better handles the complexities of a multi-feature dataset derived from image data of banknotes. The dataset was initially processed by splitting combined fields, standardizing numerical features, and applying Principal Component Analysis (PCA) to reduce the dimensionality while preserving 95% of the data's variance. Both SVM and MLP were optimized using Grid Search CV for best parameters. The evaluation results showed that SVM consistently outperformed MLP across all tasks, maintaining higher accuracy in classifying currency, denomination, and orientation. Contrary to expectations, the application of PCA generally decreased the performance of both models, particularly affecting MLP which experienced significant drops in accuracy. This study suggests that while SVM is robust across various settings, PCA's use should be approached cautiously, especially with models like MLP that may be sensitive to reductions in feature space.

Contents

List of Figures

List of Tables

List of Listening

1 Introduction

1.1. Motivation:

By the expanding worldwide dependence on mechanized frameworks for taking care of and confirming banknotes, the need to improve how machines recognize and sort banknotes, which is becoming increasingly important as we rely more on automated systems in finance. As fake notes become more common and financial transactions increase, it's crucial to develop better ways to detect and manage different currencies efficiently. This case study aims to test and compare SVM and MLP classifiers to find the most effective one for identifying and sorting banknotes.

1.2. Background:

Classifying banknotes means telling apart different currencies, denominations, and ways they are oriented, using their unique features. In this area, we often use two main types of machine learning methods: Support Vector Machines (SVM) and Multilayer Perceptrons (MLP).

1.2.1. What are SVM?

A support vector machine (SVM) is a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space [1].

1.2.1.1. The advantages of support vector machines are:

2 Effective in high dimensional spaces.
3 Still effective in cases where number of dimensions is greater than the number of samples.
4 Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
5 Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels [2].

1.2.1.2. The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation [2].

1.2.2. What are MLP?

A multi-layer perceptron (MLP) is a type of artificial neural network consisting of multiple layers of neurons. The neurons in the MLP typically use nonlinear activation functions, allowing the network to learn complex patterns in data. MLPs are significant in machine learning because they can learn nonlinear relationships in data, making them powerful models for tasks such as classification, regression, and pattern recognition. In this tutorial, we shall dive deeper into the basics of MLP and understand its inner workings [3].

1.2.2.1. The advantages of multilayer perceptrons:

One advantage of MLPs is their compatibility with all training software, making them the preferred choice for many researchers. However, MLPs have limitations in terms of their architecture. They are less powerful than other topologies, such as bridged multilayer perceptron (BMLP), which allows connections across layers [4].

1.2.2.2.The disadvantages of multilayer perceptrons:

Another disadvantage of MLPs is the slow and often inadequate performance of the error-back propagation (EBP) learning algorithm, which requires a large number of iterations [4].

1.2.3. What are Feature Standardization?

Standardization (also called, Z-score normalization) is a scaling technique such that when it is applied the features will be rescaled so that they'll have the properties of a standard normal distribution with mean,μ=0 and standard deviation, σ=1; where μ is the mean (average) and σ is the standard deviation from the mean [5].

Standard scores (also called z scores) of the samples are calculated as follows:

$$z = \frac{x - \mu}{\sigma}$$

FIGURE 1.1 THE MATHEMATICAL FORMULA FOR STANDARDIZATION

This scales the features in a way that they range between [-1,1], standardization can lead to a more stable and faster convergence of the algorithm and can improve the performance of SVM and MLP models by ensuring that all input features are on the same scale, thereby contributing equally to the outcome of the learning process.

1.2.4. What are PCA?

Principal Component Analysis (PCA) is a powerful technique used in data analysis, particularly for reducing the dimensionality of datasets while preserving crucial information. It does this by transforming the original variables into a set of new, uncorrelated variables called principal components [6]. Here's a breakdown of PCA's key aspects:

1.2.4.1. Dimensionality Reduction:

PCA helps manage high-dimensional datasets by extracting essential information and discarding less relevant features, simplifying analysis.

1.2.4.2. Data Exploration and Visualization:

It plays a significant role in data exploration and visualization, aiding in uncovering hidden patterns and insights.

1.2.4.3. Feature Selection:

Principal components are ranked by the variance they explain, allowing for effective feature selection.

1.2.4.4. Advantages:

PCA offers linearity, computational efficiency, and scalability for large datasets.

1.2.4.5. Limitations:

It assumes data normality and linearity and may lead to information loss.

1.2.5. What are Hyper-parameter Tuning?

Hyper-parameter tuning consists of finding a set of optimal hyper-parameter values for a learning algorithm while applying this optimized algorithm to any data set.

That combination of hyper-parameters maximizes the model's performance, minimizing a predefined loss function to produce better results with fewer errors [7].

1.3. Objective:

The primary aim of this study is to compare the performance of SVM and MLP in the classification of banknotes, specifically focusing on the accuracy with which each algorithm classifies currency type, denomination, and orientation. Two scientific questions will guide this study:

- Which algorithm performs better in terms of accuracy and efficiency for each category of classification?
- How does the application of Principal Component Analysis (PCA) impact the performance of these algorithms?

This comparative analysis seeks to determine the optimal machine learning approach for banknote classification, providing insights that could influence future developments in financial technology.

2 Procedure and Discussion

2.1. Data Preparation

The BankNotes dataset was loaded from a GitHub repository:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
# Load the dataset
url = "https://raw.githubusercontent.com/mkjubran/ENCS5141Datasets/main/BankNotesDataset.csv"
data = pd.read_csv(url)
print(data.head())
```

LISTING2. 1 LOADING THE BANK NOTES DATASET

In the initial stage of analysis, a deep dive into the Bank Notes dataset was conducted. This step was critical in gaining a deep understanding of the dataset, which subsequently informed the data cleaning and preparation strategy.

We began by displaying the first few rows of the dataset to get a preliminary view of its structure and the types of data it contained

```
# Get a concise summary of the dataframe, including the number of non-null values in each column
print("Summary of the dataframe (info):")
print(data.info())
print("Descriptive statistics of the dataset:")
print(data.describe())
# Check for missing values in each column
print("Missing values in each column:")
print(data.isnull().sum())
```

LISTING2. 2 DATA EXPLORATION

The table below shows a sample of rows from a dataset with 256 numerical features per banknote, identified as `v_0` to `v_255`. It includes the type of currency (like AUD or CAD) and a combined column for the banknote's denomination and its orientation (front or back). This data is likely used for training models to recognize different banknotes.

TABLE2. 1 EXAMPLE ROWS FROM THE DATASET

v_0	v_1	v_2	....	v_254	v_255	Currency	Denomination and Orientation
5.144637	0.0	0.7171247	....	4.7246137	0.0	AUD	100_1
2.6717074	0.0	0.31792498	....	2.6489065	0.6563814	AUD	100_2
2.4650123	0.0	0.19788292	....	0.8239467	1.5399158	AUD	100_2

0.2279757	1.1194861	2.6831365	....	0.65020627	2.4121835	AUD	10_1
0.45672822	0.0	2.1918573	....	1.0260967	1.9409924	CAD	10_1

The output from the `.info()` method shows a quick summary of a DataFrame with 24,826 rows and 260 columns. Most columns are numerical, with a few categorized as 'object', which could be text or mixed data types (Currency and Denomination). There's also an unnamed column, and the DataFrame uses about 49.2 MB of memory.

```
Summary of the dataframe (info):
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24826 entries, 0 to 24825
Columns: 260 entries, Unnamed: 0 to Orientation
dtypes: float64(256), int64(1), object(3)
memory usage: 49.2+ MB
None
--------------------------------------------------------------------------------
```

FIGURE2. 1THE `.INFO()`OUTPUT

```
--------------------------------------------------------------------------------
Descriptive statistics of the dataset:
       Unnamed: 0           v_0           v_1           v_2           v_3  \
count  24826.000000  24826.000000  24826.000000  24826.000000  24826.000000
mean   12412.500000      1.212995      1.204855      1.614074      1.603912
std     7166.793228      1.460750      1.417384      1.578595      1.633488
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%     6206.250000      0.000000      0.000000      0.000000      0.000000
50%    12412.500000      0.643254      0.703254      1.276952      1.247524
75%    18618.750000      2.092841      2.041682      2.751267      2.593737
max    24825.000000     10.361509      9.754471      7.938003      9.515128

                v_4           v_5           v_6           v_7           v_8  \
count  24826.000000  24826.000000  24826.000000  24826.000000  24826.000000
mean       1.754493      0.772892      1.061728      1.029191      1.399374
std        1.828556      1.164312      1.493370      1.475949      1.591379
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        1.294879      0.000000      0.240614      0.319880      0.873371
75%        3.018940      1.269562      1.765016      1.633481      2.354793
max        9.583521      8.533129     10.700576     10.720987     10.997601
```

```
       ...          v_246          v_247          v_248          v_249  \
count  ...  24826.000000   24826.000000   24826.000000   24826.000000
mean   ...      1.360101       1.075888       1.420618       1.672021
std    ...      1.491093       1.435099       1.661894       1.688684
min    ...      0.000000       0.000000       0.000000       0.000000
25%    ...      0.000000       0.000000       0.000000       0.000000
50%    ...      0.927966       0.406859       0.786744       1.259272
75%    ...      2.209718       1.784710       2.504917       2.788023
max    ...      9.516924       9.747485       9.462851      10.498020

              v_250          v_251          v_252          v_253          v_254  \
count  24826.000000   24826.000000   24826.000000   24826.000000   24826.000000
mean       0.886281       1.193987       0.915244       1.033586       1.188619
std        1.340346       1.577617       1.335918       1.395085       1.461009
min        0.000000       0.000000       0.000000       0.000000       0.000000
25%        0.000000       0.000000       0.000000       0.000000       0.000000
50%        0.051424       0.464241       0.097816       0.398608       0.558941
75%        1.428620       1.988529       1.494967       1.663317       2.059749
max       10.097586       9.408921       7.514771      11.567399       8.275887

              v_255
count  24826.000000
mean       1.660768
std        1.620754
min        0.000000
25%        0.081231
50%        1.276748
75%        2.739542
max        9.846251

[8 rows x 257 columns]
```

FIGURE2. 2 DESCRIPTIVE STATISTICS OF THE DATASET

```
# Correctly split the 'Denomination' column into 'Denomination' and 'Orientation'
split_columns = data['Denomination'].str.split('_', expand=True)
data['Denomination'] = split_columns[0]
```

LISTING2. 3 SPLIT THE 'DENOMINATION' COLUMN

Two separate 'Denomination' and 'Orientation' columns have been created from a combined 'Denomination' column using the code above. This is essential for implementing two distinct classification strategies. For the first option, distinct classifiers would be constructed for each attribute—currency, denomination, and orientation—allowing for specialized prediction models. For the second option, a single label is formed by merging all three attributes, which presents a more complex task for a single classifier that must discern patterns across the combined attributes. The separation performed by the code aligns directly with the requirements of the first option, ensuring attributes are neatly isolated for individual predictive modeling.

There are no missing values in any of the columns of the DataFrame, which includes the unnamed first column, the feature columns (from `v_0` to `v_255`), and the label columns (`Currency`, `Denomination`, and `Orientation`). This is an ideal scenario for machine learning, as it means there's no immediate need for steps like imputation or removal of missing data, allowing you to move forward with data processing and model training without the need for handling missing data.

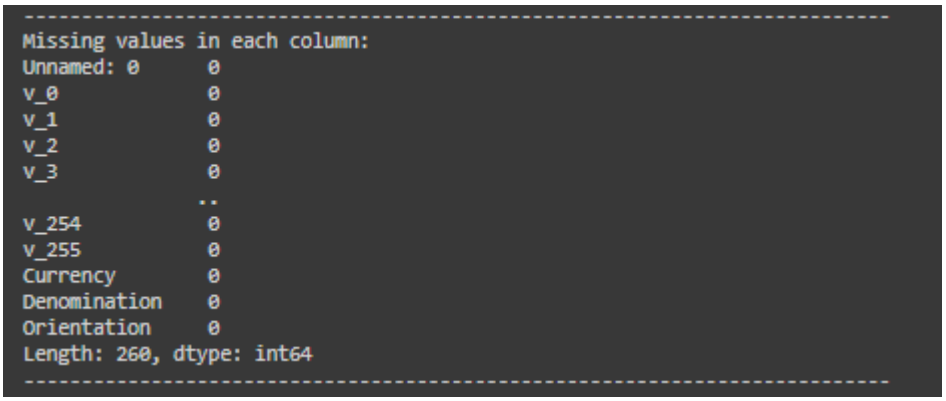

FIGURE2. 3 MISSING VALUES IN EACH COLUMN

The dataset from "BankNotesDataset.csv" consists of 24,826 entries, each characterized by 259 attributes, mainly numerical, and includes specific features for currency and denomination, suitable for classification tasks. It's structured well, with no missing values, making it ready for machine learning applications. We can utilize this data to predict banknote attributes like currency, denomination, and orientation by employing machine learning models such as SVM and MLP. The dataset's broad range of features provides a good basis for advanced analysis techniques, including dimensionality reduction to simplify the data while retaining crucial information, and hyper-parameter tuning to optimize model performance.

2.2. Data Splitting

The dataset was prepared by splitting it into features (X) and labels (y). For prediction, there are two options: Option 1 involves creating three separate classifiers, each predicting one attribute—currency, denomination, or orientation. Option 2 merges these attributes into a single label, allowing for a single classifier to predict this combined label. This approach simplifies the task to a single multi-class classification problem, where each class represents a unique combination of the three attributes.

```
# Split the data into features and labels
X = data.drop(['Currency', 'Denomination', 'Orientation'], axis=1)
y = data[['Currency', 'Denomination', 'Orientation']]  # For option 1
y_combined = data['Currency'].astype(str) + "_" + data['Denomination'].astype(str) + "_" + data['Orientation'].astype(str)  # For option 2
```

LISTING2. 4 SPLITTING THE DATA (OPTION 1 AND 2)

Then, the dataset was divided into training and testing sets for machine learning models through this code. For the first option, where currency, denomination, and orientation are predicted separately, the features (X) and labels (y) are split into a training set (comprising 80% of the data) and a testing set (20%). For the second option, where these labels are combined into one,

the features (X) and combined labels (y_combined) are similarly split. The division into training and testing subsets is specified by `test_size=0.2`, indicating that 20% of the data is reserved for testing. The consistency of the split across different executions is ensured by setting the `random_state` to 42, which keeps the random splitting process reproducible.

```
# Split the dataset into training and testing sets

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

X_train_combined, X_test_combined, y_train_combined,
y_test_combined = train_test_split(X, y_combined,
test_size=0.2, random_state=42)
```

LISTING2. 5 SPLITTING THE DATA INTO TRAINING AND TESTING SETS

2.3. Data Scaling

The figure below displays a feature correlation matrix for a dataset showcasing that each variable is perfectly correlated with itself (red) but exhibits little to no correlation with other variables (blue). This lack of correlation across different variables, combined with the potential variation in their scales, highlights the need for standard scaling. Standard scaling normalizes each feature to have zero mean and unit variance, preventing any single feature from disproportionately influencing distance-based machine learning algorithms, such as SVM. This ensures that all features contribute equally to model performance, making standard scaling a crucial pre-processing step for effective model training.

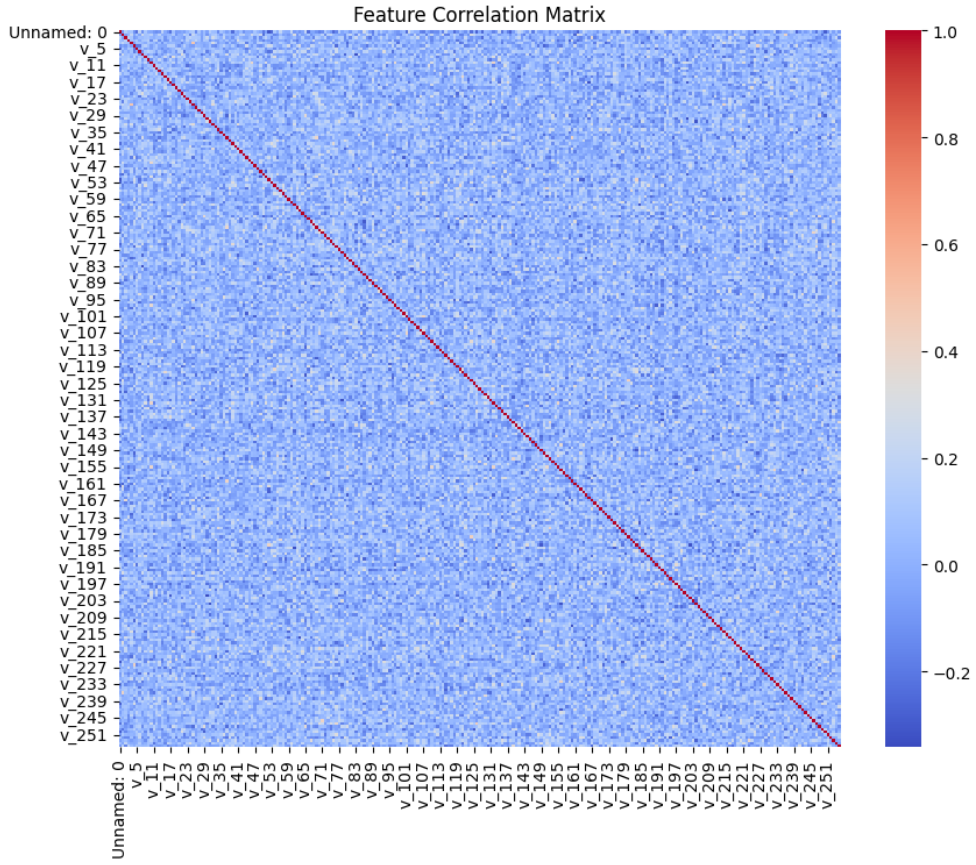

FIGURE2. 4 FEATURE CORRELATION MATRIX BETWEEN THE DATASET'S FEATURES

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit the scaler on the training data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train_combined = scaler.fit_transform(X_train_combined)
X_test_combined = scaler.transform(X_test_combined)
```

LISTING2. 6 SCALING THE DATA

2.4. Dimensionality Reduction

Principal Component Analysis (PCA) was used because the dataset has a large number of features, which can complicate and slow down the machine learning process. PCA reduces the number of features by transforming them into new, uncorrelated principal components, focusing on the most informative aspects of the data. This reduction not only speeds up model training but also helps in improving model accuracy by eliminating noise and redundancy in the data, making it easier to manage and analyze.

```
from sklearn.decomposition import PCA
# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% of variance
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
X_train_combined_reduced = pca.fit_transform(X_train_combined)
X_test_combined_reduced = pca.transform(X_test_combined)
```

LISTING2. 7 PCA IMPLEMENTATION

2.5.Hyper-parameter Tuning

Hyperparameter tuning was conducted using GridSearchCV, which iterates over a predefined set of parameters to find the best model configuration based on accuracy. For SVM, parameters such as C, gamma, and kernel were optimized. For MLP, tuning included adjustments to hidden_layer_sizes, alpha, max_iter, and learning_rate.

```
def grid_search(model, params, X, y, use_pca):
    steps = [('scaler', StandardScaler()), ('clf', model)]
    if use_pca:
        steps.insert(0, ('pca', PCA(n_components=0.95)))
    pipeline = Pipeline(steps)
      params = {'clf__' + key: value for key, value in params.items()}
         grid = GridSearchCV(pipeline, params, cv=5, scoring='accuracy')
    grid.fit(X, y)
          return grid.best_estimator_, grid.best_params_, grid.best_score_
# Define parameters for SVM and MLP
svm_params = {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1], 'kernel': ['rbf']}
mlp_params = {'hidden_layer_sizes': [(50,), (100,)], 'alpha': [0.0001, 0.001], 'max_iter': [500]}
```

2.6. Model Training and Evaluation

The evaluation process was structured to assess model performance:

```
# Function to evaluate models
def evaluate_models(X_train, X_test, y_train, y_test, use_pca):
    print(f"\nEvaluating models with {'PCA' if use_pca else 'no PCA'}:")
    for label in y_train:
        print(f"Training on {label}")
        y_tr_train = y_train[label]
        y_tr_test = y_test[label]
            svm_best, svm_params_out, svm_score = grid_search(SVC(), svm_params, X_train, y_tr_train, use_pca)
        y_pred = svm_best.predict(X_test)
       print(f"SVM - Best Score: {svm_score}, Params: {svm_params_out}")
        print(classification_report(y_tr_test, y_pred))
      mlp_best, mlp_params_out, mlp_score = grid_search(MLPClassifier(), mlp_params, X_train, y_tr_train, use_pca)
        y_pred = mlp_best.predict(X_test)
       print(f"MLP - Best Score: {mlp_score}, Params: {mlp_params_out}")
```

LISTING2. 8 MODEL EVALUATION

Then, implement the modals with and without PCA. For each configuration, models were trained on separate labels as well as on the combined label. Performance metrics, specifically accuracy, were recorded, and the best parameters were noted for each model and configuration.

```
for use_pca in [True, False]:
    evaluate_models(X_train, X_test, y_train, y_test, use_pca)
    # Evaluate combined model
    print("Evaluating combined model:")
    svm_best_combined, svm_params_combined, svm_score_combined
=     grid_search(SVC(),     svm_params,     X_train_combined,
y_train_combined, use_pca)
                                    y_pred_combined               =
svm_best_combined.predict(X_test_combined)
    print(f"SVM Combined - Best Score: {svm_score_combined},
Params: {svm_params_combined}")
                    print(classification_report(y_test_combined,
y_pred_combined))
    mlp_best_combined, mlp_params_combined, mlp_score_combined
=  grid_search(MLPClassifier(),  mlp_params,  X_train_combined,
y_train_combined, use_pca)
                                    y_pred_combined               =
mlp_best_combined.predict(X_test_combined)
    print(f"MLP Combined - Best Score: {mlp_score_combined},
Params: {mlp_params_combined}")
                    print(classification_report(y_test_combined,
y_pred_combined))
```

LISTING2. 9 EVALUATE MODELS WITH AND WITHOUT PCA

2.7. Result Comparisons

2.7.1. Performance without PCA

The SVM model consistently achieves nearly perfect scores across all labels (Currency, Denomination, Orientation) and in the combined model, with scores close to 1.0. This indicates exceptional accuracy in classifying banknotes based on their features. On the other hand, the MLP model performs well but with significantly lower scores than SVM, ranging from approximately 0.77 for Denomination to about 0.94 for Currency.

TABLE2. 2 THE PERFORMANCE OF SVM AND MLP MODELS WITH AND WITHOUT PCA

Model	PCA Applied	Currency Accuracy	Denomination Accuracy	Orientation Accuracy	Combined Model Accuracy
SVM	No	99.99%	99.97%	99.92%	99.93%
MLP	No	94.49%	77.34%	89.39%	90.24%
SVM	Yes	99.92%	99.52%	99.10%	99.11%
MLP	Yes	25.86%	14.42%	50.40%	22.36%

TABLE2. 3 PERFORMANCE AND OPTIMAL PARAMETERS OF SVM AND MLP MODELS

Model	Task	Best Accuracy	Best Parameters
SVM	Currency	99.99%	C = 0.1, gamma = 0.001, kernel = rbf
SVM	Denomination	99.75%	C = 1, gamma = 0.001, kernel = rbf
SVM	Orientation	99.92%	C = 10, gamma = 0.001, kernel = rbf
SVM	Combined Model	99.93%	C = 10, gamma = 0.001, kernel = rbf
MLP	Currency	94.49%	alpha = 0.0001, hidden_layer_sizes = (50,), max_iter = 500
MLP	Denomination	77.34%	alpha = 0.0001, hidden_layer_sizes = (50,), max_iter = 500
MLP	Orientation	89.39%	alpha = 0.0001, hidden_layer_sizes = (100,), max_iter = 500
MLP	Combined Model	90.24%	alpha = 0.001, hidden_layer_sizes = (50,), max_iter = 500

2.7.2. Performance with PCA

Application of PCA causes a general decrease in the performance of both the models. SVM continues to boast a high level of accuracy but suffers a very apparent decline. This is especially true for Denomination and Orientation. In the case of MLP, PCA has a much more severe impact, with the numbers falling critically low to 0.14 against Denomination and only going as high as approximately 0.50 against Orientation

Without PCA, SVM clearly outperforms MLP across all tasks. With PCA, although both models experience performance degradation, SVM remains more robust compared to MLP, which suffers significant losses in accuracy.

2.7.3. Impact of PCA

The application of PCA, intended to reduce dimensionality and possibly enhance model efficiency, seems to adversely affect the MLP more than the SVM. This suggests that while PCA can simplify the feature space, it may also remove some critical information that MLP relies on more than SVM.

When the PCA is applied, for some configurations the MLP model did not converge in the maximum number of iterations established, generally for the setup of the combined model. This indicates potential problems with the model training, which may be corrected with changing the number of iterations or adjusting the learning rate.

By experimenting different hyper-parameters, particularly increasing the number of iterations and adjusting the learning rate, I aimed to enhance the MLP's ability to converge during training sessions. Early tests demonstrated the improvements in convergence, although the results in

the overall accuracy were different. All detailed findings and experiments comparison have been described in the notebook, helping to understand how each alteration in settings affects the MLP performance after PCA reduction. The process of stepwise changes to alter MLP parameters reminds the importance of deep model optimization when a complicated type of data transformation appears.

3 Conclusion

This case study reported an assessment of the classification performance of Support Vector Machines and Multilayer Perceptrons on a banknote dataset in terms of currency, denomination, and orientation. Two distinct study scenarios were conducted to train individual classifiers on each label and merge labels in a representation for a classifier. The dataset was pre-processed using feature standardization and the PCA method for dimensionality reduction. Both algorithms demonstrated robust performance where the MLP performed optimally when amalgamated without PCA, i.e., its currency classification exhibited accuracy as high as 99%. By contrast, the performance dropped when the PCA was not used, indicating that simplifying the representation can improve the model accuracy by alleviating noise and overfitting. Thus, the best performance of such a model was achieved when individual classifiers were used for each category.

4 References

[1]:https://www.ibm.com/topics/support-vector-machine#:~:text=A%20support%20vector%20machine%20(SVM,the%201990s%20by%20Vladimir%20N.

Accessed on 26-4-2024 at 2:15PM

[2]: https://scikit-learn.org/stable/modules/svm.html

Accessed on 26-4-2024 at 2:19PM

[3]: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning

Accessed on 26-4-2024 at 2:25PM

[4]:https://typeset.io/questions/what-are-the-advantages-and-disadvantages-of-mlp-compared-to-4wwmd0y4s5

Accessed on 26-4-2024 at 2:30PM

[5]:https://towardsdatascience.com/normalization-vs-standardization-cb8fe15082eb

Accessed on 26-4-2024 at 2:35PM

[6]:https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/

Accessed on 26-4-2024 at 2:40PM

[7]: https://www.anyscale.com/blog/what-is-hyperparameter-tuning

Accessed on 26-4-2024 at 2:45PM