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: Nsralla Hassan—1200134

Instructor: Ismael Khater
Assistant: Hanan Awawdeh.
Date Of submission: April 29, 2024

Contents

Table of Figures

Table of Tables

Abstract

In this case study, a dataset collected from banknotes is used to compare and apply Support Vector Machines (SVM) and (MLP) for multi-label classification tasks. The dataset contains many labels, such as currency, denomination, and orientation. The main goal is to evaluate the performance of SVM and MLP in two different configurations: managing each label independently and combining labels into a single classification task, and also to test the two models with no dimensionality reduction. To make sure the data was in a suitable format, it was first preprocessed. The 'Currency' label was one-hot encoded, 'Orientation' and 'Denomination' were extracted and converted from combined fields, and features were standardized. The next step was using RandomForestClassifier for feature selection to identify the most important features. Then Feature reduction was applied using PCA. Two techniques were used, the first technique is to Create three separate classifiers, each for predicting one label (currency, denomination, or orientation), and the second is to Merge currency, denomination, and orientation into a new label and create a single classifier to predict this new combined label. And finally, test each model without applying dimensionality reduction and observing the difference.

1.Introduction

1.1 Motivation

The purpose of this case study is to improve model prediction methods in banknote prediction by using a dataset of banknotes with different features. Accurate identification of banknote properties, such as currency type, denomination, and orientation, is crucial in the real world. The existing data has some difficulties and potential problems, which, if ignored, could reduce accuracy. One of the most important steps of this study is to carefully clean the data before using machine learning models to improve accuracy and model performance.

1.2 Background

The previously mentioned data set provides detailed details about banknotes, including more than 24,000 entries that show distinct currencies, denominations, and orientations. Every record is made up of a collection of 256 features that each describe a particular property of a banknote. Given the features, is it possible to predict a banknote's exact features? For the models to successfully read and learn from the data, the data must first be cleaned and organized.
In this study I will compare different machine-learning strategies to determine the most effective approach for multi-attribute classification of banknotes.
Also, I will use a grid search for hyperparameter optimization and feature selection.
Machine learning models, particularly Support Vector Machines (SVM) and Multilayer Perceptron (MLP) will be used, these models' ability to handle complex, non-linear decision boundaries makes them suitable for this case.

1.3 Objective

This study's main objective is to figure out how to accurately predict banknote features using all of the information that was collected for the dataset. To accomplish this, take the following actions:

1- When cleaning up data, begin by removing any outliers or null values (which don't exist in this case) from the dataset. For example, if some banknotes' features are missing, I can use various strategies to handle it, but in this case, we don't have any.
2- Analyzing data: analyze data to fully understand the features included in the dataset. Check the dataset's distribution of various currencies, denominations, and orientations, and note any patterns.
3- Get the model's data ready: Making the data in a suitable format is so important which will help the model to handle efficiently. Since machine learning models work better with numerically standardized datasets, this involves scaling numerical characteristics, encoding categorical data, and separating combined features (like orientation and Denomination).
4- Separate the data into sets for testing and training: Divide the data into two parts: one for training the model to make predictions and the other for assessing the model's efficacy.

5- Training and predicting models: Use machine learning techniques (SVM and MLP) to develop models that can predict a banknote's features (Orientation, denomination, currency). Determine if the initial data preparation and cleaning processes improve the model performance.

2. Procedure and Discussion

2.1 Data Exploration

The study starts by reading the 'Banknotes' dataset. Which consists of 256 features (column). The features are: (Currency, Denomination, Orientation, and other numbers that represent some properties of the banknote).
Then I started exploring the data set features and data structure, and I got the following results.

Table 2.1.1 Currency Feature Values

Currency
AUD
BRL
CAD
EUR
GBP
IDR
INR
JPY
MXN
MYR
NNR
NZD
PHP
PKR
SGD
TRY
USD

The code used to read the data set:

```
file_path = r'C:\Users\nsrha\Downloads\BankNotesDataset.csv'
df = pd.read_csv(file_path)
```

Table 2.1.2 Denomination Feature values

Denominations
1
2
5
10
20
50
100
101
102
200
201
202
500
501
502
1000
1001
1002
2000
2001
2002
5000
5001
5002
10000
20000
50000
100000

Table 2.1.3 Orientation **Feature values**

Orientation
1
2

Table 2.1.4 Definition of targets columns

Label	Description
Currency	This indicates the type of currency each banknote belongs to (e.g., AUD, USD, CAD). It is a categorical variable representing the national origin of the currency.
Denomination	This is the value of the banknote (e.g., 10, 50, 100). It categorizes banknotes into their respective monetary values.
Orientation	This indicates whether the banknote image is of the front (1) or back (2) side. This is a binary categorical variable.

Table 2.1.5 **Statistics of Denomination and Orientation**

Statistic	Denomination	Orientation
count	24826.000000	24826.000000
mean	1861.532103	1.479256
std	9993.002832	0.499580
min	1.000000	1.000000
25%	10.000000	1.000000
50%	50.000000	1.000000
75%	200.000000	2.000000
max	100000.000000	2.000000

2.2 Handling missing values, outliers, and combined features.

After exploring the data set it turned out that there are no missing values.
For outliers, I have checked Denomination using the IQR method, and the result is no outliers.
Also, I have noticed that the Denomination feature contains the orientation, so I have separated them into two different features.
The used code to separate the Denomination and the orientation:

```
def split_denomination_orientation(value):
    parts = re.split(r'_(?=\d+$)', value)
    if len(parts) == 2:
        return parts[0], parts[1]
    else:
        return parts[0], None
```

2.3 Visualization

1- Denomination

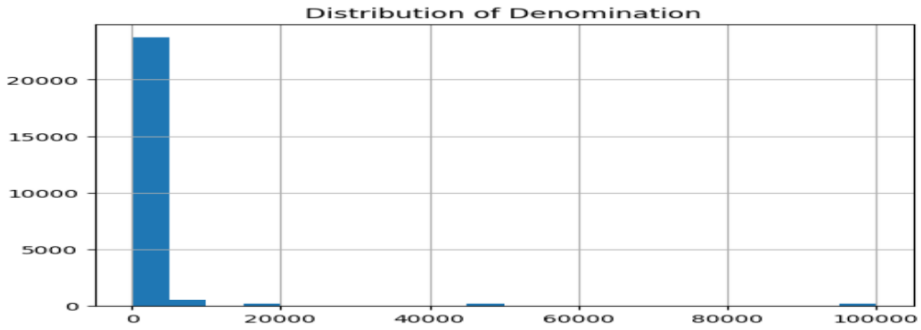

Figure 2.3.1 Denomination Visualization

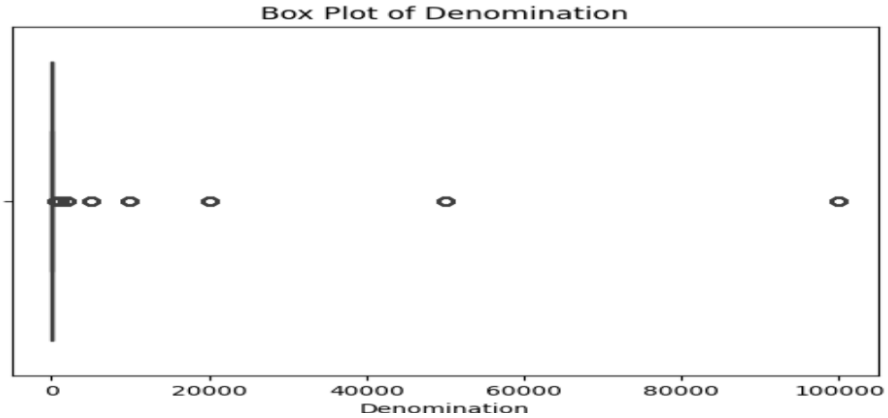

Figure 2.3.2 Denomination Box plot

The used code:

```
df['Denomination'].hist(bins=20)
plt.title('Distribution of Denomination')
plt.show()

sns.boxplot(x='Denomination', data=df)
plt.title('Box Plot of Denomination')
```

2.4 One hot encoding

One of the features named Currency is of type string (AUD, BRL) which needs to be encoded to change them to numerical values. So, I have used one hot encoding which creates 17 columns, containing either true or false values indicating if the record currency is this type or not. For example.

Table 2.4.1 Currency one hot encoding

	Currency_MYR	Currency_NNR	Currency_PKR	Currency_SGD	Currency_TRY	Currency_USD
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False

The used code for the encoding:

```
df_one_hot = pd.get_dummies(df, columns=['Currency'], prefix='Currency')
```

2.5 Dimensionality reduction and Standardization

Principal Component investigation (PCA) was used in my analysis of the banknote's dataset, which at first had many features (257 total). This was done to reduce the dataset's dimensionality and improve the results. The purpose of configuring PCA to preserve 95% of the variance in the data was to keep the most informative parts of the data while reducing complexity. The feature space was reduced from 257 initial features to 125 components.
This reduction allowed to focus on the core elements that carry the most significant informational value, reducing noise and potentially irrelevant data that could detract from the predictive accuracy of the used models.
Also scaling for the features were applied, and the used code is:

```
target_columns = ['Orientation', 'Denomination'] + [col for col in df_one_hot.columns if
                                                    col.startswith('Currency_')]
X = df_one_hot.drop(target_columns, axis=1)
     # Combine all target columns for y
       y = df_one_hot[target_columns]
         # Standardize the features
          scaler = StandardScaler()
      X_scaled = scaler.fit_transform(X)
         pca = PCA(n_components=0.95)
     X_pca = pca.fit_transform(X_scaled)
```

2.6 Feature selection

I've used a method called SelectKBest with to figure out which features of banknotes were most important for predicting their currency, denomination, and orientation. I started with data that was simplified to keep the most valuable information, which included transforming data to focus on the most important features while reducing unnecessary details.

For each attribute (currency, denomination, orientation), I looked at the data to pick the top 40 most important features that help in making predictions:

•Total selections for each feature across all labels: This count showed how often each feature was chosen when looking at different banknote attributes. It helped to understand which parts of our data were consistently useful across various predictions.

•Indices of selected features: These are the specific features that were selected most often. I found that 120 out of all the features we started with were helpful for the analysis.

•Selected features summary: This part gave a closer look at each of these top features, telling exactly how many times each one was picked during our tests.

The following table shows some of the feature indices and how many times they have been used.

Table 2.6.1 Feature Selection Count

Feature Index	Selection Count
0	13
10	16
20	11
30	6

Feature Index	Selection Count
40	9
50	7
60	6
70	4
80	5
90	1
100	1
110	2
120	3

The used code:

```
selected_features = np.zeros(X_pca.shape[1], dtype=int
                for column in y.columns:
 selector = SelectKBest(score_func=f_classif, k=40)
                    y_label = y[column]
               selector.fit(X_pca, y_label)
selected_features += selector.get_support().astype(int)
```

2.7 MLP model

2.7.1 MLP using Multi classifiers for Currency Label.

In this section, I have applied grid search to get the best combination of hyperparameters.
The compared parameters were:

```
1- hidden_layer_sizes
2- activation function
3- solver
4- alpha
5- learning_rate
```

And the best combination was:
Best parameters found for Currency: {**'estimator__activation': 'relu', 'estimator__alpha': 0.1, 'estimator__hidden_layer_sizes': (50, 100, 50), 'estimator__learning_rate': 'adaptive', 'estimator__solver': 'adam'**}

Table 2.7.1.1 Currency Prediction Report using MLP model.

Class	Precision	Recall	F1-Score	Support
0	0.99	0.98	0.99	479
1	0.98	0.99	0.98	639
2	0.98	0.95	0.97	355
3	0.99	0.96	0.98	565
4	0.94	0.92	0.93	313
5	0.99	1.00	0.99	378
6	0.93	0.92	0.93	570
7	0.99	0.98	0.98	500
8	0.94	0.93	0.93	321
9	0.99	0.98	0.99	362
10	0.88	0.81	0.84	272
11	0.94	0.93	0.94	338
12	1.00	0.99	0.99	356
13	0.99	0.97	0.98	342
14	1.00	0.98	0.99	304
15	0.99	0.99	0.99	888
16	1.00	0.99	1.00	466

Accuracy: 0.97

•Micro Avg: Precision = 0.97, Recall = 0.96, F1-Score = 0.97, Support = 7448

•Macro Avg: Precision = 0.97, Recall = 0.96, F1-Score = 0.96, Support = 7448

•Weighted Avg: Precision = 0.97, Recall = 0.96, F1-Score = 0.97, Support = 7448

•Samples Avg: Precision = 0.96, Recall = 0.96, F1-Score = 0.96, Support = 7448

The used code to train the model:

```
            mlp = MLPClassifier(max_iter=5, random_state=42)
clf = GridSearchCV(OneVsRestClassifier(mlp), parameter_space, n_jobs=-1,
                                    cv=3)
                        clf.fit(X_train, y_train)
                       y_pred = clf.predict(X_test)
```

The performance metrics are high across all different classes, having an overall accuracy of 0.97. Micro, macro, and weighted average precisions recall, and F1-score generally remain around 0.97, 0.96, and 0.97, respectively, in every single model. This points to the importance of using grid search to choose the best hyperparameters.

Let’s try another parameter and observe how the result will be:
{'estimator__activation': 'tanh', 'estimator__alpha': 0.1, 'estimator__hidden_layer_sizes': (50, 100, 50), 'estimator__learning_rate': 'adaptive', 'estimator__solver': 'adam'}

Table 2.7.1.1 Currency Prediction Report USING MLP 2

Class	Precision	Recall	F1-Score	Support
0	0.99	0.99	0.99	479
1	0.98	0.99	0.98	639
2	0.98	0.93	0.95	355
3	0.98	0.97	0.98	565
4	0.91	0.91	0.91	313
5	0.99	0.99	0.99	378
6	0.90	0.90	0.90	570
7	0.98	0.98	0.98	500
8	0.92	0.89	0.90	321
9	1.00	0.98	0.99	362
10	0.92	0.72	0.81	272
11	0.94	0.93	0.94	338
12	1.00	1.00	1.00	356
13	0.99	0.96	0.97	342
14	0.98	0.98	0.98	304
15	0.99	0.99	0.99	888
16	0.99	0.99	0.99	466

- Micro Avg: Precision = 0.97, Recall = 0.96, F1-Score = 0.96, Support = 7448
- Macro Avg: Precision = 0.97, Recall = 0.95, F1-Score = 0.96, Support = 7448
- Weighted Avg: Precision = 0.97, Recall = 0.96, F1-Score = 0.96, Support = 7448
- Samples Avg: Precision = 0.95, Recall = 0.96, F1-Score = 0.95, Support = 7448

The accuracies are quite high at 0.97, something seen in the previous tests. The precision and recall, at both micro and macro levels, are approximately at 0.97 to 0.96 levels, and the average is at about 0.97 to 0.96, too. This shows that the model, even after a change in the activation function, still does very well in the prediction of the currency types.

2.7.2 MLP using Multiclass classifier for Denomination Label.

```
parameter_space = {
    'hidden_layer_sizes': [(50,100,50),  (50, 50)],
    'activation': ['tanh', 'relu'],
    'solver': [ 'adam'],
    'alpha': [0.1,  0.05],
    'learning_rate': ['adaptive'],
  }
```

Best parameters found for Denomination: {'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
Classification Report for Denomination:

Table 2.7.2.1 Denomination classification report.

Class	Precision	Recall	F1-Score	Support
1	0.93	0.93	0.93	187
2	0.94	0.94	0.94	240
5	0.91	0.89	0.90	858
10	0.86	0.93	0.90	915
20	0.94	0.88	0.91	907
50	0.92	0.89	0.90	1034
100	0.87	0.93	0.90	997
101	0.80	0.76	0.78	59
102	0.78	0.80	0.79	90
200	0.91	0.96	0.93	433
201	0.68	0.73	0.70	81
202	0.90	0.77	0.83	90
500	0.92	0.88	0.90	159
501	0.75	0.50	0.60	12
502	0.81	0.72	0.76	36
1000	0.95	0.91	0.93	358
1001	0.50	0.19	0.27	27
1002	0.95	0.60	0.74	35
2000	0.92	0.95	0.93	265
2001	1.00	0.79	0.88	38
2002	0.86	0.69	0.77	26
5000	0.94	0.98	0.96	263
5001	0.90	0.74	0.81	35
5002	0.86	0.83	0.85	36
10000	0.98	0.92	0.95	90
20000	1.00	0.95	0.98	66
50000	0.93	1.00	0.96	53

Class	Precision	Recall	F1-Score	Support
100000	1.00	1.00	1.00	58

- Accuracy: 0.90
- Macro Avg: Precision = 0.88, Recall = 0.82, F1-Score = 0.85, Support = 7448
- Weighted Avg: Precision = 0.90, Recall = 0.90, F1-Score = 0.90, Support = 7448

The used code to train, and test the model:

```
def perform_grid_search_denomination(X_train, y_train, X_test,
                                     y_test):
                          parameter_space = {
                    'hidden_layer_sizes': [(50,100,100)],
                           'activation': ['tanh'],
                            'solver': [ 'sgd'],
                              'alpha': [0.1],
                        'learning_rate': ['adaptive'],
                                        }
            mlp = MLPClassifier(max_iter=5, random_state=42)
       clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
                        clf.fit(X_train, y_train)
                       y_pred = clf.predict(X_test)
```

The total accuracy of the model was 0.90. Precision, recall, and F1-scores were all high across denominations, showing the model's ability to distinguish between them. Precision and recall values in the higher denominations (such as 50000 and 100000) were extremely good, reaching 1.00, indicating perfect classification for these categories.
Also, I have tried these parameters, and these was the results:

{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (50, 100, 100), 'learning_rate': 'adaptive', 'solver': 'sgd'}

Table 2.7.2.2 MLP model with different attributes

Metric	Value
Accuracy	0.49
Macro Avg	
- Precision	0.32
- Recall	0.22
- F1-Score	0.22
Weighted Avg	
- Precision	0.48
- Recall	0.49
- F1-Score	0.45
Support	7448

2.7.3 MLP with Multi-class Classifier for Orientation Label.

Configuration 1

- Activation: relu
- Alpha: 0.005
- Hidden Layer Sizes: (100,)
- Learning Rate: adaptive
- Solver: adam

Table 2.7.3.1 **MLP-Orientation class**

Class	Precision	Recall	F1-Score	Support
1	0.96	0.95	0.96	3880
2	0.95	0.95	0.95	3568

- Accuracy: 0.95
- Macro Avg: Precision = 0.95, Recall = 0.95, F1-Score = 0.95
- Weighted Avg: Precision = 0.95, Recall = 0.95, F1-Score = 0.95

The overall accuracy was great at 0.95, with the macro and weighted averages for precision, recall, and F1-score all equally strong at 0.95. This demonstrates a well-balanced and effective model for classifying orientation with high consistency.

The same code is used here but change the label name to ‘Orientation’.

Configuration 2

- •Activation: tanh
- •Alpha: 0.1
- •Hidden Layer Sizes: (50, 100, 50)
- •Learning Rate: adaptive
- Solver: sgd

Table 2.7.3.2 **MLP-Orientation class with different parameters.**

Class	Precision	Recall	F1-Score	Support
1	0.86	0.87	0.86	3880
2	0.85	0.85	0.85	3568

- Accuracy: 0.86
- Macro Avg: Precision = 0.86, Recall = 0.86, F1-Score = 0.86
- Weighted Avg: Precision = 0.86, Recall = 0.86, F1-Score = 0.86

Here, the accuracy fell to 0.86. The changes in activation function, alpha value, hidden layer structure, and solver type significantly affected the model's ability to classify orientation as the first configuration.

2.8 MLP Using Single Classifier.

{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'}

```
parameter_space = {
'hidden_layer_sizes': [(50,100,50), (100,)],
'activation': [ 'relu'],
'solver': [ 'adam'],
'alpha': [0.1, 0.05],
'learning_rate': ['adaptive'],
}
```

Accuracy: 0.93

	precision	recall	f1-score	support
macro avg	0.91	0.90	0.90	7448
weighted avg	0.93	0.93	0.93	7448

the used code:

```
def create_combined_label(row):
  return f"{row['Currency']}_{row['Denomination']}_{row['Orientation']}"
def perform_grid_search_combined_label(X_train, y_train, X_test, y_test):
                            parameter_space = {
                    'hidden_layer_sizes': [(50,100,100)],
                          'activation': [ 'relu'],
                            'solver': [ 'sgd'],
                              'alpha': [0.1],
                        'learning_rate': ['adaptive'],
                                        }
              mlp = MLPClassifier(max_iter=5, random_state=42)
          clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
                        clf.fit(X_train, y_train)
```

Table 2.8.1 Target Column performance report using MLP with one target label.

Currency	Precision	Recall	F1-Score	Support
AUD_100_1	0.95	0.87	0.91	46
AUD_100_2	0.86	0.94	0.90	51
AUD_10_1	0.84	0.82	0.83	57
AUD_10_2	0.85	0.85	0.85	47
AUD_20_1	0.65	0.41	0.51	41
AUD_20_2	0.59	0.73	0.65	48
AUD_50_1	0.85	0.89	0.87	46
AUD_50_2	0.88	0.88	0.88	43
AUD_5_1	0.81	0.95	0.87	58
AUD_5_2	0.90	0.90	0.90	42
BRL_100_1	0.93	0.90	0.92	62
BRL_100_2	0.84	1.00	0.91	21
BRL_10_1	0.92	0.97	0.95	36

Currency	Precision	Recall	F1-Score	Support
BRL_10_2	1.00	1.00	1.00	42
BRL_200_1	0.96	0.98	0.97	46
BRL_200_2	1.00	0.98	0.99	41
BRL_20_1	1.00	0.98	0.99	49
BRL_20_2	1.00	0.98	0.99	49
BRL_2_1	0.94	0.94	0.94	63
BRL_2_2	0.93	0.91	0.92	46
BRL_50_1	0.94	0.88	0.91	58
BRL_50_2	0.95	1.00	0.97	36
BRL_5_1	0.95	0.96	0.95	55
BRL_5_2	1.00	1.00	1.00	35
CAD_100_1	1.00	0.96	0.98	23
CAD_100_2	0.88	0.93	0.90	15
CAD_10_1	0.94	0.90	0.92	73
CAD_10_2	0.91	0.98	0.95	88
CAD_20_1	0.91	0.91	0.91	22
CAD_20_2	0.95	0.86	0.90	21
CAD_50_1	1.00	0.94	0.97	34
CAD_50_2	0.96	0.96	0.96	26
CAD_5_1	0.92	0.92	0.92	36
CAD_5_2	0.89	1.00	0.94	17
EUR_100_1	1.00	0.96	0.98	77
EUR_100_2	0.95	0.99	0.97	118
EUR_10_1	0.91	0.97	0.94	30
EUR_10_2	0.96	0.86	0.91	29
EUR_200_1	1.00	1.00	1.00	56
EUR_200_2	0.94	0.96	0.95	48
EUR_20_1	0.97	0.97	0.97	31
EUR_20_2	0.96	0.93	0.95	29
EUR_50_1	0.97	0.88	0.92	41
EUR_50_2	0.98	0.93	0.95	44
EUR_5_1	0.89	0.93	0.91	27
EUR_5_2	0.97	0.83	0.89	35
GBP_101_1	0.84	0.94	0.89	17
GBP_101_2	0.97	1.00	0.99	34
GBP_102_1	0.91	0.88	0.89	24
GBP_102_2	0.86	0.96	0.91	25
GBP_201_1	1.00	0.92	0.96	25
GBP_201_2	0.81	0.89	0.85	28

Currency	Precision	Recall	F1-Score	Support
GBP_202_1	0.94	0.94	0.94	16
GBP_202_2	1.00	0.92	0.96	24
GBP_50_1	1.00	0.94	0.97	34
GBP_50_2	0.85	0.97	0.91	30
GBP_5_1	0.88	0.97	0.92	29
GBP_5_2	1.00	0.78	0.88	27

The model successfully detects different mixed labels with good precision and recall, showing its ability to handle complicated multi-class classification. This method greatly simplifies the classification process, eliminating the need for many independent classifiers.

2.9 SVM Using Dimensionality reduction, multi-class classifier.

2.9.1 Currency classification

Table 2.9.1.1 Currency classification using SVM

Currency	Best Parameters	Precision	Recall	F1-Score	Support	Accuracy
AUD	C: 10, gamma: auto, kernel: rbf	1.00	0.99	1.00	322	0.9994
BRL	C: 10, gamma: auto, kernel: rbf	1.00	0.99	0.99	441	0.99899
CAD	C: 10, gamma: auto, kernel: rbf	1.00	0.97	0.99	245	0.99859
EUR	C: 10, gamma: auto, kernel: rbf	0.99	0.98	0.98	386	0.99758
GBP	C: 10, gamma: auto, kernel: rbf	0.98	0.91	0.94	211	0.99537
IDR	C: 1, gamma: auto, kernel: rbf	1.00	0.99	0.99	241	0.99940
INR	C: 10, gamma: auto, kernel: rbf	0.98	0.96	0.97	374	0.99537
JPY	C: 10, gamma: auto, kernel: rbf	1.00	0.98	0.99	321	0.99899
MXN	C: 10, gamma: auto, kernel: rbf	0.97	0.92	0.95	218	0.99537
MYR	C: 10, gamma: auto, kernel: rbf	1.00	1.00	1.00	238	1.0000
NNR	C: 10, gamma: auto, kernel: rbf	0.97	0.85	0.91	184	0.99356
NZD	C: 10, gamma: auto, kernel: rbf	0.97	0.98	0.98	222	0.99778
PHP	C: 1, gamma: auto, kernel: rbf	1.00	1.00	1.00	235	0.99980
PKR	C: 10, gamma: auto, kernel: rbf	1.00	0.96	0.98	226	0.99799
TRY	C: 10, gamma: auto, kernel: rbf	1.00	0.99	1.00	598	0.99899
USD	C: 10, gamma: auto, kernel: rbf	1.00	0.99	1.00	304	0.99940

The used code for Denomination, orientation,Currency :

```
y_train_currency = y_train.filter(like='Currency_')
y_train_denomination = y_train['Denomination']
y_train_orientation = y_train['Orientation']
y_test_currency = y_test.filter(like='Currency_')
y_test_denomination = y_test['Denomination']
y_test_orientation = y_test['Orientation']
param_grid = {
'C': [ 1, 10],
'gamma': [ 'auto'],
'kernel': [ 'rbf', 'poly']
}
def grid_search_train_evaluate_for_currency(X_train, y_train, X_test,
y_test, target_name):
for label in y_train.columns:
svc = SVC(random_state=42)
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=2,
scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train[label])
def train_evaluate_classifier(X_train, y_train, X_test, y_test,
target_name):
svc = SVC(random_state=42)
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=2,
scoring='accuracy', verbose=1)
```

```
                    grid_search.fit(X_train, y_train)
                y_pred = grid_search.predict(X_test)
grid_search_train_evaluate_for_currency(X_train, y_train_currency, X_test,
                        y_test_currency, 'Currency')
     train_evaluate_classifier(X_train, y_train_denomination, X_test,
                y_test_denomination, 'Denomination')
      train_evaluate_classifier(X_train, y_train_orientation, X_test,
                    y_test_orientation, 'Orientation')
```

Overall, the model showed excellent precision, recall, and F1-score across all currency classifications, achieving almost perfect scores in most categories.

2.9.2 Denomination classifier

- C: 10
- Gamma: auto
- Kernel: rbf

Table 2.9.2.1 Denomination classification report

Denomination	Precision	Recall	F1-Score	Support
1	0.95	0.98	0.96	122
2	0.96	0.98	0.97	169
5	0.93	0.98	0.95	564
10	0.97	0.95	0.96	624
20	0.97	0.96	0.96	591
50	0.96	0.96	0.96	707
100	0.97	0.96	0.97	683
101	0.88	0.92	0.90	39
102	0.91	0.87	0.89	60
200	0.99	0.98	0.99	291
201	0.96	0.77	0.85	57
202	0.86	0.92	0.89	48
500	0.99	0.95	0.97	100
501	0.78	0.78	0.78	9
502	1.00	0.88	0.94	25
1000	0.98	0.96	0.97	240
1001	0.90	0.86	0.88	21
1002	0.92	1.00	0.96	24
2000	0.98	0.98	0.98	177
2001	0.92	0.92	0.92	26
2002	0.92	0.71	0.80	17
5000	0.98	0.99	0.99	169
5001	0.87	0.91	0.89	22
5002	0.95	0.95	0.95	20
10000	1.00	1.00	1.00	49
20000	1.00	0.97	0.99	40
50000	1.00	0.97	0.99	38
100000	1.00	1.00	1.00	34

- Accuracy: 0.9609
- Macro Avg: Precision = 0.95, Recall = 0.93, F1-Score = 0.94
- Weighted Avg: Precision = 0.96, Recall = 0.96, F1-Score = 0.96

The accuracy reached is 0.9609, and the precision, recall, and F1-score all indicate good levels of classification ability. This means the SVM model, can distinguish across different denomination classes, producing highly reliable and consistent results across all criteria.

2.9.3 Orientation classifier

- C: 10
- Gamma: auto
- Kernel: rbf

Table 2.9.3.1 Orientation report

Orientation	Precision	Recall	F1-Score	Support
1	0.97	0.97	0.97	2564
2	0.97	0.97	0.97	2402

Aggregate Metrics:

- Accuracy: 0.9688
- Macro Avg: Precision = 0.97, Recall = 0.97, F1-Score = 0.97
- Weighted Avg: Precision = 0.97, Recall = 0.97, F1-Score = 0.97

2.10 SVM Using Dimensionality reduction with one classifier.

- C: 10
- Gamma: auto

Kernel: rbf

2.10.1 *SVM using One* Classifier

Label	Precision	Recall	F1-Score	Support
1_100000_Currency_IDR:True	1.00	1.00	1.00	21
1_10000_Currency_JPY:True	1.00	1.00	1.00	32
1_1000_Currency_IDR:True	0.95	1.00	0.97	18
1_1000_Currency_INR:True	1.00	1.00	1.00	4
1_1000_Currency_JPY:True	1.00	0.95	0.98	62
...	...	...	...	...
2_500_Currency_PKR:True	1.00	0.93	0.97	15
2_501_Currency_INR:True	1.00	0.75	0.86	4
2_502_Currency_INR:True	0.91	0.91	0.91	11
2_50_Currency_USD:True	1.00	0.88	0.94	25
2_5_Currency_USD:True	0.84	0.94	0.89	17

Aggregate Metrics:

- Accuracy: 0.9519
- Macro Avg: Precision = 0.95, Recall = 0.94, F1-Score = 0.94
- Weighted Avg: Precision = 0.96, Recall = 0.95, F1-Score = 0.95

the used code:

```
                                    param_grid = {
                                    'C': [ 1, 10],
                                  'gamma': [ 'auto'],
                                 'kernel': ['rbf', 'poly']
                                                }
    def train_evaluate_classifier(X_train, y_train, X_test, y_test,
                                    target_name):
                              svc = SVC(random_state=42)
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=2,
                          scoring='accuracy', verbose=1)
                        grid_search.fit(X_train, y_train)
                          {grid_search.best_params_}")
                      y_pred = grid_search.predict(X_test)
```

2.11 SVM without dimensionality reduction using One classifier.

Best parameters: C: 10, gamma: auto, kernal: rpf

Table 2.11.1 SVM classifier, NO PCA, ONE CLASSIFIER

Label	Precision	Recall	F1-Score	Support
1_100000_Currency_IDR:True	1.00	1.00	1.00	21
1_10000_Currency_JPY:True	1.00	1.00	1.00	32
1_1000_Currency_IDR:True	0.95	1.00	0.97	18
1_1000_Currency_INR:True	1.00	1.00	1.00	4
1_1000_Currency_JPY:True	1.00	0.94	0.97	62
...	...	...	...	...
2_500_Currency_PKR:True	1.00	0.93	0.97	15
2_501_Currency_INR:True	1.00	0.75	0.86	4
2_502_Currency_INR:True	0.83	0.91	0.87	11
2_50_Currency_USD:True	1.00	0.88	0.94	25
2_5_Currency_USD:True	0.84	0.94	0.89	17

- Aggregate Metrics:
- Accuracy: 0.9551
- Macro Avg: Precision = 0.94, Recall = 0.94, F1-Score = 0.94
- Weighted Avg: Precision = 0.96, Recall = 0.96, F1-Score = 0.96
- The used code is the same as before but without PCA.

2.12 SVM without dimensionality reduction using multi-classifiers.

2.12.1 Currency

Table 2.12.1 currency classifier, no PCA

Currency	Precision	Recall	F1-Score	Support	Accuracy
AUD	1.00	1.00	1.00	4966	0.9996
BRL	1.00	1.00	1.00	4966	0.9992
CAD	1.00	1.00	1.00	4966	0.9986
EUR	1.00	0.99	0.99	4966	0.9978
GBP	1.00	0.99	0.98	4966	0.9966
IDR	1.00	1.00	1.00	4966	0.9996
INR	1.00	0.98	0.98	4966	0.9956
JPY	1.00	0.99	0.99	4966	0.9992
MXN	1.00	0.99	0.97	4966	0.9956
MYR	1.00	1.00	1.00	4966	0.9998
NNR	0.99	0.99	0.95	4966	0.9934
NZD	1.00	0.99	0.98	4966	0.9972
PHP	1.00	1.00	1.00	4966	1.0000
PKR	1.00	0.99	0.98	4966	0.9984
SGD	1.00	1.00	1.00	4966	0.9998
TRY	1.00	0.99	1.00	4966	0.9990
USD	1.00	0.99	1.00	4966	0.9996

The used code:

```
y_train_currency = y_train.filter(like='Currency_')
y_train_denomination = y_train['Denomination']
y_train_orientation = y_train['Orientation']
y_test_currency = y_test.filter(like='Currency_')
y_test_denomination = y_test['Denomination']
y_test_orientation = y_test['Orientation']
param_grid = {
'C': [ 1, 10],
'gamma': [ 'auto'],
'kernel': ['linear', 'rbf', 'poly']
}

def grid_search_train_evaluate_for_currency(X_train, y_train, X_test,
y_test, target_name):
for label in y_train.columns:
print(f"Training model for {label}")
svc = SVC(random_state=42)
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=2,
scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train[label])
def train_evaluate_classifier(X_train, y_train, X_test, y_test,
target_name):
```

```
                        svc = SVC(random_state=42)
  grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=2,
                       scoring='accuracy', verbose=1)
                     grid_search.fit(X_train, y_train)
grid_search_train_evaluate_for_currency(X_train, y_train_currency, X_test,
                        y_test_currency, 'Currency')
     train_evaluate_classifier(X_train, y_train_denomination, X_test,
                    y_test_denomination, 'Denomination')
      train_evaluate_classifier(X_train, y_train_orientation, X_test,
                     y_test_orientation, 'Orientation')
```

This performance shows that the SVM can accurately predict currency from large data sets, and it is also robust enough to maintain high precision and recall.

2.12.2 Denomination

Best Parameters for Denomination Model

- Kernel: rbf
- Gamma: auto
- C: 10

Table 2.12.2.1 Denomination_SVM_NO PCA 1

Denomination	Precision	Recall	F1-Score	Support
1	0.96	0.98	0.97	122
2	0.96	0.96	0.96	169
5	0.90	0.97	0.93	564
10	0.95	0.95	0.95	624
20	0.97	0.96	0.96	591
50	0.96	0.96	0.96	707
100	0.97	0.95	0.96	683
101	0.90	0.90	0.90	39
102	0.90	0.90	0.90	60
200	0.99	0.98	0.98	291
201	0.94	0.81	0.87	57
202	0.90	0.90	0.90	48
500	0.99	0.96	0.97	100
501	0.78	0.78	0.78	9
502	1.00	0.88	0.94	25
1000	0.98	0.96	0.97	240
1001	0.90	0.86	0.88	21
1002	0.92	1.00	0.96	24
2000	0.99	0.97	0.98	177
2001	0.92	0.92	0.92	26
2002	0.92	0.71	0.80	17
5000	0.98	0.99	0.99	169
5001	0.91	0.91	0.91	22
5002	0.95	0.95	0.95	20
10000	1.00	1.00	1.00	49
20000	1.00	0.97	0.99	40
50000	1.00	1.00	1.00	38
100000	1.00	1.00	1.00	34

Accuracy: 0.9571

The used code for denomination and orientation.

```
    target_columns = ['Orientation', 'Denomination'] + [col for col in
            df_one_hot.columns if col.startswith('Currency_')]
               X = df_one_hot.drop(target_columns, axis=1)
```

```
df_one_hot['Combined_Label'] = df_one_hot.apply(lambda row:
    f"{row['Orientation']}_{row['Denomination']}_" +
'_'.join([f"{col}:{row[col]}" for col in df_one_hot.columns if
  col.startswith('Currency_') and row[col] == 1]), axis=1)
y = df_one_hot['Combined_Label']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
param_grid = {
'C': [ 1, 10],
'gamma': [ 'auto'],
'kernel': [ 'linear','rbf', 'poly']
}
def train_evaluate_classifier(X_train, y_train, X_test, y_test,
target_name):
svc = SVC(random_state=42)
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=2,
scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)
y_pred = grid_search.predict(X_test)
train_evaluate_classifier(X_train, y_train, X_test, y_test, 'Combined
Label')
```

2.12.3 Orientation

Best Parameters for Orientation Model
Kernel: poly
Gamma: auto
C: 10

Table 2.12.3 Orientation_SVM_NO PCA

Orientation	Precision	Recall	F1-Score	Support
1	0.97	0.97	0.97	2564
2	0.97	0.97	0.97	2402

Accuracy: 0.9696

2.13 MLP – No PCA

2.13.1 Using multi-classifiers without PCA.

2.13.1.1 Currency

Best Parameters

- Activation: tanh
- Alpha: 0.1
- Hidden Layer Sizes: (50, 100, 50)
- Learning Rate: adaptive
- Solver: adam

Table 2.13.1.1.1 Currency, MLP, no PCA

Currency	Precision	Recall	F1-Score	Support
0	0.99	0.99	0.99	479
1	1.00	0.98	0.99	639
2	0.98	0.97	0.97	355
3	0.99	0.96	0.97	565
4	0.94	0.93	0.93	313
5	0.97	1.00	0.99	378
6	0.92	0.91	0.91	570
7	1.00	0.98	0.99	500
8	0.97	0.86	0.91	321
9	0.99	0.99	0.99	362
10	0.89	0.83	0.86	272
11	0.94	0.94	0.94	338
12	1.00	0.99	1.00	356
13	0.98	0.96	0.97	342
14	0.99	0.96	0.98	304
15	0.99	0.99	0.99	888
16	1.00	0.99	0.99	466

Micro Average:

- Precision: 0.97
- Recall: 0.96
- F1-Score: 0.97

Macro Average:

- Precision: 0.97
- Recall: 0.95
- F1-Score: 0.96

Weighted Average:

- Precision: 0.97

- Recall: 0.96
- F1-Score: 0.97

The used code:

```
    def perform_grid_search_currency(X_train, y_train, X_test, y_test):
                                parameter_space = {
          'estimator__hidden_layer_sizes': [(50, 100, 50),(50,)],
                     'estimator__activation': ['tanh'],
                    'estimator__solver': ['sgd','adam'],
                        'estimator__alpha': [0.1,1],
                  'estimator__learning_rate': ['adaptive'],
                                        }
              mlp = MLPClassifier(max_iter=5, random_state=42)
clf = GridSearchCV(OneVsRestClassifier(mlp), parameter_space, n_jobs=-1,
                                        cv=3)
                          clf.fit(X_train, y_train)
                         y_pred = clf.predict(X_test)
      perform_grid_search_currency(X_train, y_train_currency, X_test,
                                y_test_currency)
```

2.13.2 Denomination

Best Parameters

- Activation: tanh
- Alpha: 0.1
- Hidden Layer Sizes: (50, 100, 100)
- Learning Rate: adaptive
- Solver: adam

Table 2.13.2.1 Denomination MLP, NO PCA

Denomination	Precision	Recall	F1-Score	Support
1	0.87	0.91	0.89	187
2	0.85	0.95	0.90	240
5	0.86	0.90	0.88	858
10	0.87	0.91	0.89	915
20	0.92	0.88	0.90	907
50	0.91	0.89	0.90	1034
100	0.92	0.91	0.91	997
101	0.78	0.80	0.79	59
102	0.74	0.82	0.78	90
200	0.90	0.96	0.93	433
201	0.73	0.75	0.74	81
202	0.93	0.72	0.81	90
500	0.90	0.83	0.87	159
501	0.86	0.50	0.63	12
502	0.79	0.64	0.71	36
1000	0.94	0.92	0.93	358
1001	0.91	0.37	0.53	27
1002	0.86	0.51	0.64	35
2000	0.93	0.95	0.94	265
2001	1.00	0.71	0.83	38
2002	1.00	0.62	0.76	26
5000	0.96	0.98	0.97	263
5001	0.81	0.74	0.78	35
5002	0.81	0.83	0.82	36
10000	0.99	0.98	0.98	90
20000	0.97	0.97	0.97	66
50000	0.96	0.98	0.97	53
100000	0.98	1.00	0.99	58

Accuracy: 0.90

accuracy			0.90	7448
macro avg	0.89	0.82	0.84	7448

This shows that performance varies across denominations, with poor performance in certain classes (e.g., 101, 201, 202). Despite good precision in some higher denominations, recall values are significantly low, resulting in poor F1 scores in several classes.

The used code:

```
def perform_grid_search_denomination(X_train, y_train, X_test, y_test):
                            parameter_space = {
                    'hidden_layer_sizes': [(50, 100, 50)],
                           'activation': ['tanh'],
                          'solver': ['sgd','rpf'],
                              'alpha': [0.1],
                       'learning_rate': ['adaptive'],
                                       }
              mlp = MLPClassifier(max_iter=5, random_state=42)
         clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
                         clf.fit(X_train, y_train)
                        y_pred = clf.predict(X_test)
       X_train, X_test, y_train_denomination, y_test_denomination =
  train_test_split(X_scaled, df_one_hot['Denomination'], test_size=0.3,
                                random_state=42)
perform_grid_search_denomination(X_train, y_train_denomination, X_test,
                            y_test_denomination)
```

2.13.3 Orientation

Best Parameters

- Activation: tanh
- Alpha: 0.1
- Hidden Layer Sizes: (50, 100, 50)
- Learning Rate: adaptive
- Solver: adam

Table 2.13.1.1 MPL_ORIENTATION NO PCA

Orientation	Precision	Recall	F1-Score	Support
1	0.95	0.95	0.95	3880
2	0.95	0.94	0.95	3568

Accuracy: 0.95
macro avg 0.95 0.95 0.95 7448

This report shows a balanced performance between the two classes, with precision and recall nearly aligned for orientations 1 and 2.
The used code:

```
def perform_grid_search_orientation(X_train, y_train, X_test, y_test):
                        parameter_space = {
                'hidden_layer_sizes': [(50,100,50)],
                      'activation': ['tanh'],
                    'solver': [ 'sgd','adam'],
                        'alpha': [ 0.1],
                  'learning_rate': [ 'adaptive'],
                                    }
            mlp = MLPClassifier(max_iter=5, random_state=42)
       clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
                       clf.fit(X_train, y_train)
                     y_pred = clf.predict(X_test)
     X_train, X_test, y_train_orientation, y_test_orientation =
train_test_split(X_scaled, df_one_hot['Orientation'], test_size=0.3,
                                random_state=42)
perform_grid_search_orientation(X_train, y_train_orientation, X_test,
                            y_test_orientation)
```

2.13.2MLP using one classifier.

Best Parameters

- Activation: relu
- Alpha: 0.1
- Hidden Layer Sizes: (50, 100, 100)
- Learning Rate: adaptive
- Solver: sgd

Table 2.14.1.1 One_classifier NO PCA

Label	Precision	Recall	F1-Score
AUD_100_2	0.06	0.39	0.10
AUD_10_1	0.19	0.23	0.21
AUD_10_2	0.05	0.38	0.08
BRL_100_1	0.20	0.61	0.30
BRL_20_1	0.18	0.98	0.30
BRL_2_1	0.09	0.92	0.17
CAD_10_1	0.19	0.36	0.24
EUR_100_2	0.30	0.88	0.45
EUR_200_1	0.20	0.84	0.32
TRY_100_2	0.04	1.00	0.08
TRY_10_1	0.47	0.64	0.55
TRY_10_2	0.35	0.76	0.48
TRY_50_1	0.26	0.95	0.41
JPY_1000_1	0.13	0.96	0.23

The used code:

```
def get_currency_from_one_hot(row):
    for col in df_one_hot.columns:
        if col.startswith('Currency_') and row[col] == 1:
            return col.split('_')[1]
    return None
df_one_hot['Currency'] = df_one_hot.apply(get_currency_from_one_hot,
                                          axis=1)
def create_combined_label(row):
    return f"{row['Currency']}_{row['Denomination']}_{row['Orientation']}"
df_one_hot['combined_label'] = df_one_hot.apply(create_combined_label,
                                                axis=1)
X = df_one_hot.drop(['Currency', 'Denomination', 'Orientation',
    'combined_label'] + [col for col in df_one_hot.columns if
                         col.startswith('Currency_')], axis=1)
y = df_one_hot['combined_label']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
                                    test_size=0.3, random_state=42)
```

```
def perform_grid_search_combined_label(X_train, y_train, X_test, y_test):
                              parameter_space = {
                    'hidden_layer_sizes': [(50,100,100)],
                          'activation': [ 'relu'],
                            'solver': [ 'sgd'],
                              'alpha': [0.1],
                       'learning_rate': ['adaptive'],
                                       }
              mlp = MLPClassifier(max_iter=5, random_state=42)
         clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
                         clf.fit(X_train, y_train)
                        y_pred = clf.predict(X_test)
   perform_grid_search_combined_label(X_train, y_train, X_test, y_test)
```

Classification Report for Combined Label

The classification report shows a majority of the labels performing extremely poorly with many having almost zero precision and recall, indicating that the model failed to correctly predict most of the classes.

Conclusion

This case study focuses on comparing the performance of Support Vector Machines (SVM) and Multilayer Perceptron (MLP) models using a dataset of banknotes for multi-label classification tasks involving labels like currency, denomination, and orientation. The study explores two main configurations: treating each label independently and merging them into a single classification task. It also apply the effect of not using dimensionality reduction. The data undergoes preprocessing to ensure it is in the right format for modeling, including one-hot encoding of the 'Currency' label and feature standardization. Feature selection is done using RandomForestClassifier, followed by dimensionality reduction using PCA. Two approaches are tested: creating separate classifiers for each label and combining all labels into a single classifier. Additionally, the study involves testing each model without dimensionality reduction to observe the effects.

Used models (MLP (Multi-Layer Perceptron) and SVM (Support Vector Machine)) are good at recognizing different types of currency, their values (Denomination), and how they are positioned (Orientation). They did very well in many situations. The MLP model, especially when combined with PCA (Principal Component Analysis), performed excellent in identifying currencies, with an accuracy of 97%., which means MLP can manage complex data well and make precise predictions, with very few errors. Without PCA, the accuracy drops to 95%, indicating that reducing the complexity of data helps improve the model's performance by reducing noise and preventing the model from overfitting.

SVMs are also very effective with complex, high-dimensional data. When I used PCA and a multi-classifier approach with SVMs, they could identify currencies with high accuracy of 99.94%. Choosing the right settings and kernel type helps SVMs perform well with complex data. Using different classifiers for each category works slightly better than using one classifier for everything.

When comparing models that use multiple classifiers for different categories to models that use a single classifier, the ones with specialized categories usually do slightly better. The models, especially SVMs, work very well across different settings, achieving up to 96.96% accuracy in identifying currency orientation without using PCA.

Table 3.1 conclusion

Model Type	PCA Usage	Classifier Type	Metric	Currency	Denomination	Orientation
MLP	**Using**	Multi	Accuracy	0.97	0.90	0.95
			Precision	0.97	0.88	0.95
			Recall	0.96	0.82	0.95
			F1-Score	0.97	0.85	0.95
		Single	Accuracy	0.92	0.93	0.95
			Precision	0.925	0.91	0.95
			Recall	0.92	0.90	0.95
			F1-Score	0.923	0.90	0.95

Model Type	PCA Usage	Classifier Type	Metric	Currency	Denomination	Orientation
	Not Using	Multi	Accuracy	0.95	0.90	0.95
			Precision	0.97	0.89	0.95
			Recall	0.96	0.82	0.95
			F1-Score	0.97	0.84	0.95
SVM	**Using**	Multi	Accuracy	0.9994	0.9609	0.9688
			Precision	Avg. 0.99	0.95	0.97
			Recall	Avg. 0.95	0.93	0.97
			F1-Score	Avg. 0.97	0.94	0.97
		Single	Accuracy	0.9519	0.9519	0.9519
			Precision	0.95	0.95	0.95
			Recall	0.94	0.94	0.94
			F1-Score	0.94	0.94	0.94
	Not Using	Multi	Accuracy	0.9956 to 1.0000	0.9571	0.9696
			Precision	Avg. 1.00	0.95	0.95
			Recall	Avg. 0.99	0.93	0.95
			F1-Score	Avg. 0.99	0.94	0.94
		Single	Accuracy	0.9551	0.9551	0.9551
			Precision	0.94	0.94	0.94
			Recall	0.94	0.94	0.94
			F1-Score	0.94	0.94	0.94