A2: Classification with SVM, BP and MLR

Professors: SERGIO GÓMEZ JIMÉNEZ, JORDI DUCH GAVALDÀ
Student: SEBASTIAN-EUGEN BUZDUGAN

Table of Contents

Introduction

This project explores the application of advanced computational methods - Support Vector Machines (SVM), Back-Propagation (BP), and Multiple Linear Regression (MLR) - in the realm of classification tasks.

The goal is to examine and compare the efficacy of these methods across various datasets, including Ring, Bank Marketing, and Banknote Authentication. Each technique offers unique approaches to classification problems: SVM is known for its effectiveness in high-dimensional spaces, BP is a cornerstone of neural network training, enhancing the learning process, and MLR provides a baseline for comparison with its linear approach to regression analysis.

Data Description

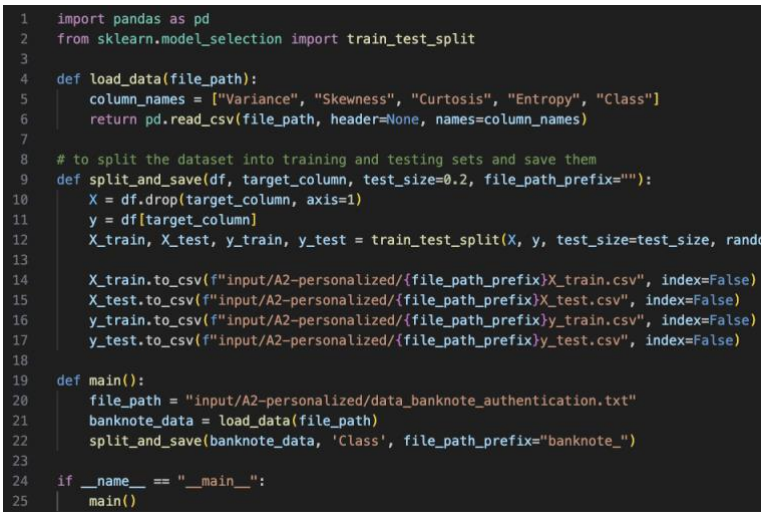
- **Ring Datasets:** Two training sets: separable (easier) and merged (more difficult). One test set applicable for both training sets. Features: 2 input features and 1 class identifier (0/1). Data size: 10,000 patterns in each file. Recommendation: Plot the data due to low dimensionality.
- **Bank Marketing Dataset:** Data files: bank-additional.csv (4,119 patterns) or bank-additional-full.csv (41,188 patterns). Training/Test Split: First 80% for training, last 20% for testing. Features: 20, mostly categorical, requiring numerical representation. Prediction Target: Last feature (yes/no) for term deposit subscription. Note: Missing information tagged as “unknown”.
- **Banknote Authentication Dataset:** 1372 rows with five columns representing various attributes from banknote images. Final column: Class identifier for genuine/forged banknotes. Data Split: 80% for training/validation, 20% for testing, with a shuffle to avoid bias.

Data preprocessing steps

- **Ring Datasets:** No preprocessing required. Suitable for direct application of classification models.
- **Bank Marketing Dataset:** Preprocessed using Python. 'Unknown' values replaced with most frequent value. Categorical features transformed into numerical form using label encoding. Dataset split into 80% training and 20% testing sets, saved as CSV files.

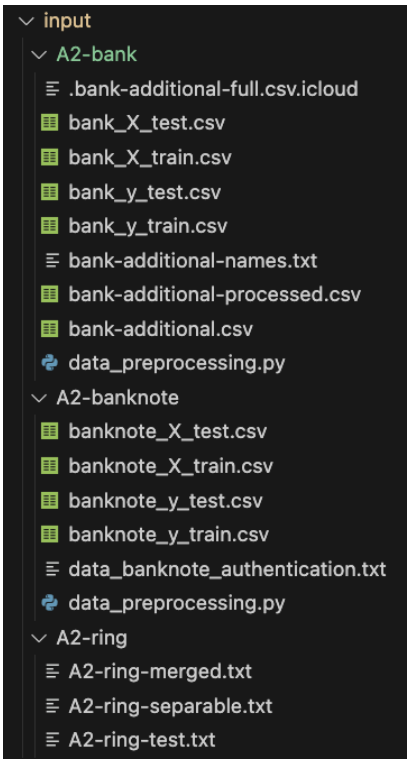
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

def load_data(file_path):
    return pd.read_csv(file_path, sep=";")

def preprocess_data(df):
    # replace 'unknown' with the most frequent value
    for column in df.columns:
        most_frequent = df[column].mode()[0]
        df[column] = df[column].replace('unknown', most_frequent)

    # convert categorical features into numerical form using label encoding
    for column in df.columns:
        if df[column].dtype == 'object':
            le = LabelEncoder()
            df[column] = le.fit_transform(df[column])
    return df

# to split the dataset into training and testing sets and save them
def split_and_save(df, target_column, test_size=0.2, file_path_prefix=""):
    X = df.drop(target_column, axis=1)
    y = df[target_column]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_siz

    X_train.to_csv(f"input/A2-bank/{file_path_prefix}X_train.csv", index=False)
    X_test.to_csv(f"input/A2-bank/{file_path_prefix}X_test.csv", index=False)
    y_train.to_csv(f"input/A2-bank/{file_path_prefix}y_train.csv", index=False)
    y_test.to_csv(f"input/A2-bank/{file_path_prefix}y_test.csv", index=False)

def main():
    file_path = "input/A2-bank/bank-additional.csv"
    bank_dataset = load_data(file_path)
    preprocessed_data = preprocess_data(bank_dataset)
    split_and_save(preprocessed_data, 'y', file_path_prefix="bank_")

if __name__ == "__main__":
    main()
```

- **Banknote Authentication Dataset:** Preprocessed using Python. Columns named for clarity: ["Variance", "Skewness", "Curtosis", "Entropy", "Class"]. Randomized split into 80% training and 20% testing sets, ensuring shuffled data. Split datasets saved as separate CSV files for training and testing.



```python
import pandas as pd
from sklearn.model_selection import train_test_split

def load_data(file_path):
    column_names = ["Variance", "Skewness", "Curtosis", "Entropy", "Class"]
    return pd.read_csv(file_path, header=None, names=column_names)

# to split the dataset into training and testing sets and save them
def split_and_save(df, target_column, test_size=0.2, file_path_prefix=""):
    X = df.drop(target_column, axis=1)
    y = df[target_column]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, rando

    X_train.to_csv(f"input/A2-personalized/{file_path_prefix}X_train.csv", index=False)
    X_test.to_csv(f"input/A2-personalized/{file_path_prefix}X_test.csv", index=False)
    y_train.to_csv(f"input/A2-personalized/{file_path_prefix}y_train.csv", index=False)
    y_test.to_csv(f"input/A2-personalized/{file_path_prefix}y_test.csv", index=False)

def main():
    file_path = "input/A2-personalized/data_banknote_authentication.txt"
    banknote_data = load_data(file_path)
    split_and_save(banknote_data, 'Class', file_path_prefix="banknote_")

if __name__ == "__main__":
    main()
```

All these values that are pre-processed and normalized are saved in different files based on X, the feature of the dataset and y, the target variable used for prediction. This is how my project input structure looks like after the files are generated:



Methodology

Description of the SVM, BP, and MLR models

- **SVM (Support Vector Machine):** In your implementation, the SVM model is used with linear and radial basis function (RBF) kernels. The model's complexity is controlled through the regularization parameter C. A 4-fold cross-validation method is employed to optimize these parameters, ensuring the model's robustness. It's tailored to handle classification tasks efficiently in your dataset.
- **BP (Back-Propagation):** The BP model employs a Multilayer Perceptron (MLP) Classifier. It's optimized using GridSearchCV to find the best combination of hidden layers, learning rate, and maximum iterations. The use of cross-validation in parameter optimization highlights its capability to minimize overfitting and improve generalization.
- **MLR (Multiple Linear Regression):** The MLR model, though traditionally used for regression, is adapted for classification in your project. It involves using a linear regression model followed by a threshold cutoff to make class predictions. The process includes cross-validation and optimization of the threshold and data scaling, showcasing its adaptability in a classification context.

Explanation of parameter selection and optimization techniques

Back-Propagation (BP):

- **Bank dataset:** Optimal parameters were hidden_layer_sizes: (10,), learning_rate_init: 0.1, max_iter: 1000. This configuration likely provided a balance between model complexity and training efficiency, leading to a high accuracy.

```
Best parameters for Bank: {'hidden_layer_sizes': (10,), 'learning_rate_init': 0.1, 'max_iter': 1000}
Best score (accuracy) for Bank: 0.9098634294385433
Confusion Matrix for Bank:
 [[692  40]
 [ 44  48]]
```

- **Banknote Authentication**: Optimal parameters included hidden_layer_sizes: (50,), suggesting a slightly more complex model was effective. A lower learning rate (learning_rate_init: 0.001) and fewer iterations (max_iter: 500) were sufficient, indicating quick convergence.

```
Best parameters for Banknote Authentication: {'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.001, 'max_iter': 500}
Best score (accuracy) for Banknote Authentication: 1.0
Confusion Matrix for Banknote Authentication:
 [[148   0]
 [  0 127]]
```

- **Separable and Merged Ring Datasets:** Larger hidden_layer_sizes (100 for separable, 50 for merged) and higher learning rates (0.01 for both) were optimal. This implies the need for more complex models to capture the patterns in these datasets.

```
Best parameters for Separable Ring Dataset: {'hidden_layer_sizes': (100,), 'learning_rate_init': 0.01, 'max_iter': 1000}
Best score (accuracy) for Separable Ring Dataset: 0.9884000000000001
Confusion Matrix for Separable Ring Dataset:
 [[5276   57]
 [  39 4628]]
```

```
Best parameters for Merged Ring Dataset: {'hidden_layer_sizes': (50,), 'learning_rate_init': 0.01, 'max_iter': 1000}
Best score (accuracy) for Merged Ring Dataset: 0.7737999999999999
Confusion Matrix for Merged Ring Dataset:
 [[5248   85]
 [ 441 4226]]
```

Support Vector Machine (SVM):

- **Bank and Banknote Datasets:** The linear kernel with C: 0.1 and the rbf kernel with C: 10 respectively showed the best performance. The choice of kernel impacts how the data is separated, with linear being simpler and rbf more capable of handling non-linear relationships. The C parameter controls the trade-off between a smooth decision boundary and classifying training points correctly.

Best Parameters for Bank dataset

```
Kernel: linear, C: 0.1, Mean CV Accuracy: 0.9107740595028844
Kernel: linear, C: 1, Mean CV Accuracy: 0.9050091277471717
Kernel: linear, C: 10, Mean CV Accuracy: 0.9053117885075912
Kernel: rbf, C: 0.1, Mean CV Accuracy: 0.891048761929479
Kernel: rbf, C: 1, Mean CV Accuracy: 0.9007574997935566
Kernel: rbf, C: 10, Mean CV Accuracy: 0.9022748587337353
Best Model Parameters: {'kernel': 'linear', 'C': 0.1}, Best Accuracy: 0.9107740595028844
```

Best Parameters for Banknote authentication dataset

```
Kernel: linear, C: 0.1, Mean CV Accuracy: 0.98814200398142
Kernel: linear, C: 1, Mean CV Accuracy: 0.9890577305905773
Kernel: linear, C: 10, Mean CV Accuracy: 0.9890577305905773
Kernel: rbf, C: 0.1, Mean CV Accuracy: 0.9881453218314531
Kernel: rbf, C: 1, Mean CV Accuracy: 0.9954379562043795
Kernel: rbf, C: 10, Mean CV Accuracy: 1.0
Best Model Parameters: {'kernel': 'rbf', 'C': 10}, Best Accuracy: 1.0
```

- **Ring Datasets:** The rbf kernel with C: 10 was most effective, indicating non-linear separability of the data where a higher penalty on misclassifications (higher C value) was beneficial.

```
Cross-validation for Separable Dataset:
Kernel: linear, C: 0.1, Mean CV Accuracy: 0.5203
Kernel: linear, C: 1, Mean CV Accuracy: 0.5203
Kernel: linear, C: 10, Mean CV Accuracy: 0.5203
Kernel: rbf, C: 0.1, Mean CV Accuracy: 0.962
Kernel: rbf, C: 1, Mean CV Accuracy: 0.9715
Kernel: rbf, C: 10, Mean CV Accuracy: 0.9929999999999999
Best Model Parameters: {'kernel': 'rbf', 'C': 10}, Best Accuracy: 0.9929999999999999

Cross-validation for Merged Dataset:
Kernel: linear, C: 0.1, Mean CV Accuracy: 0.5515000000000001
Kernel: linear, C: 1, Mean CV Accuracy: 0.5515000000000001
Kernel: linear, C: 10, Mean CV Accuracy: 0.5515000000000001
Kernel: rbf, C: 0.1, Mean CV Accuracy: 0.7695
Kernel: rbf, C: 1, Mean CV Accuracy: 0.7713
Kernel: rbf, C: 10, Mean CV Accuracy: 0.7807000000000001
Best Model Parameters: {'kernel': 'rbf', 'C': 10}, Best Accuracy: 0.7807000000000001
```

Multiple Linear Regression (MLR):

- For all datasets, the selection of scaler (StandardScaler in all cases) and threshold (varying between 0.4 to 0.6) was crucial. The threshold determines the cut-off for classifying the continuous output of MLR into binary classes. The StandardScaler likely helped in normalizing feature scales, improving model performance. The variance in optimal thresholds across datasets indicates the sensitivity of MLR's performance to this parameter in different data contexts.

MLR for bank:

```
Optimal parameters: {'regressor__threshold': 0.4, 'scaler': StandardScaler()}
Best cross-validation score: 0.9144157814871017
Mean Classification Error in Cross-Validation: 0.08558421851289832
```

MLR for banknote:

```
Optimal parameters: {'regressor__threshold': 0.6, 'scaler': StandardScaler()}
Best cross-validation score: 0.9844997924449979
Mean Classification Error in Cross-Validation: 0.015500207555002143
```

MLR for merged ring:

```
Optimal parameters for Merged Dataset:
Best Parameters: {'regressor__threshold': 0.5, 'scaler': StandardScaler()}
Best Cross-validation Score: 0.5515
Mean Classification Error in Cross-Validation: 0.4485
```

MLR for separable ring:

```
Optimal parameters for Separable Dataset:
Best Parameters: {'regressor__threshold': 0.6, 'scaler': StandardScaler()}
Best Cross-validation Score: 0.5203
Mean Classification Error in Cross-Validation: 0.4797
```

Details of the implementation process and tools/libraries used

1. Data Processing: Data was loaded and preprocessed using Pandas, a powerful data manipulation library in Python. This step included splitting data into training and testing sets.

2. Model Implementation:
    - **Back-Propagation:** Implemented using the MLPClassifier from sklearn.neural_network, with parameter tuning via GridSearchCV.
    - **SVM:** Utilized SVC from sklearn.svm, optimizing kernel types and the C parameter. Cross-validation was performed using KFold from sklearn.model_selection.
    - **MLR:** Implemented through sklearn's LinearRegression, with a custom MLRClassifier class to adapt linear regression for classification tasks.

3. Performance Evaluation: Model performance was evaluated using metrics like accuracy, confusion matrix, and ROC AUC, calculated using sklearn.metrics.

4. Visualization: Matplotlib was used for plotting, especially for ROC curves.

Experimental Setup

Configuration of Experiments for Each Model and Dataset:

- **BP:** Outline the setup for the neural network using MLPClassifier, detailing the grid of parameters chosen for optimization, such as the number of neurons in hidden layers, learning rate, and maximum iterations. Include the reasoning behind these specific choices for each dataset.
- **SVM:** Explain the experimental setup for the SVM classifier, including the choice of kernel functions (linear and rbf) and the range of values for the regularization parameter C. Describe how these were configured differently for the bank and banknote datasets.
- **MLR:** Detail the conversion process of the MLR model into a binary classifier using a threshold mechanism. Describe the data normalization step and the rationale for the selection of the threshold values tested.

Description of Cross-Validation Strategy and Other Evaluation Methods:

- **BP:** Discuss the use of 5-fold cross-validation for the bank dataset and how it provided robustness to the model by assessing its performance on different subsets of the data.

```python
from sklearn.model_selection import cross_val_score

# define the model (using the best parameters found, from above)
mlp_cv = MLPClassifier(hidden_layer_sizes=(10,), learning_rate_init=0.1, max_iter=100, random_state=42)

# Perform k-fold cross-validation
cv_scores = cross_val_score(mlp_cv, X_train_bank, y_train_bank.values.ravel(), cv=5, scoring='accuracy')

# Calculate average classification error in cross-validation
cv_classification_error = np.mean(1 - cv_scores)
print(f"Average Classification Error (Cross-Validation): {cv_classification_error}")
```

- **SVM:** Describe the 4-fold cross-validation process used for the SVM model, explaining how the training and validation splits were made and why 4-fold was chosen.

```python
# the set of parameters defined
kernel_options = ['linear', 'rbf']
C_options = [0.1, 1, 10]

# to store the best model and its performance
best_model = None
best_accuracy = 0
best_params = {}

# 4-fold cross-validation
kf = KFold(n_splits=4)

for kernel in kernel_options:
    for C in C_options:
        cv_accuracy = []

        for train, validation in kf.split(X_train_bank):
            clf = SVC(kernel=kernel, C=C, random_state=0, probability=True)
            clf.fit(X_train_bank.iloc[train], y_train_bank.iloc[train].values.ravel())
            y_pred = clf.predict(X_train_bank.iloc[validation])
            accuracy = accuracy_score(y_train_bank.iloc[validation], y_pred)
            cv_accuracy.append(accuracy)

        mean_accuracy = np.mean(cv_accuracy)
        print(f"Kernel: {kernel}, C: {C}, Mean CV Accuracy: {mean_accuracy}")

        if mean_accuracy > best_accuracy:
            best_accuracy = mean_accuracy
            best_model = clf
            best_params = {'kernel': kernel, 'C': C}

print(f"Best Model Parameters: {best_params}, Best Accuracy: {best_accuracy}")
```

- **MLR:** For the MLR model, elaborate on the use of GridSearchCV with cross-validation to find the optimal threshold and scaling technique, and how this method ensures the model is not tailored to specificities of the training data.

```python
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold
from sklearn.metrics import make_scorer, accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression

class MLRClassifier:
    def __init__(self, threshold=0.5):
        self.regressor = LinearRegression()
        self.threshold = threshold

    def fit(self, X, y):
        self.regressor.fit(X, y)

    def predict(self, X):
        predictions = self.regressor.predict(X)
        return (predictions > self.threshold).astype(int)

    def get_params(self, deep=True):
        return {"threshold": self.threshold}

    def set_params(self, **parameters):
        for param, value in parameters.items():
            setattr(self, param, value)
        return self
```

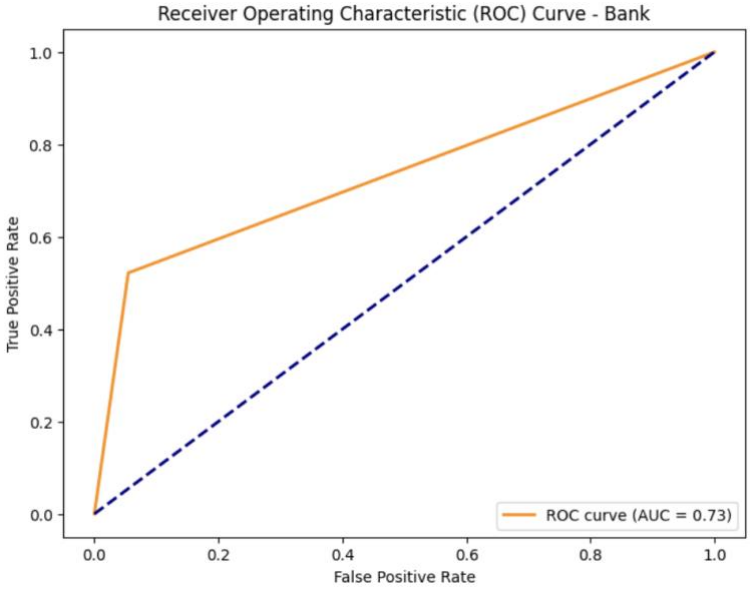
```python
# custom scoring function for accuracy
def calculate_accuracy(y_true, y_pred):
    return accuracy_score(y_true, y_pred)

# set up 5-fold cross-validation
cross_validation = KFold(n_splits=5)

# assemble a pipeline with data scaling and the custom classifier
data_processing_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # initial scaler, will be tuned
    ('regressor', MLRClassifier())
])

# scoring method based on accuracy
accuracy_scoring = make_scorer(calculate_accuracy)

# grid for tuning parameters including scalers and threshold
parameter_grid = {
    'scaler': [StandardScaler(), MinMaxScaler()],
    'regressor__threshold': [0.4, 0.5, 0.6],
}

# conduct grid search to find the best parameters
parameter_search = GridSearchCV(data_processing_pipeline, parameter_grid, cv=cross_validation, scoring=accuracy_scoring)
parameter_search.fit(X_train_bank, y_train_bank)

# display the best parameters and cross-validation score
print("Optimal parameters:", parameter_search.best_params_)
print("Best cross-validation score:", parameter_search.best_score_)

# calculate and display the mean classification error from cross-validation
mean_classification_error = 1 - parameter_search.best_score_
print("Mean Classification Error in Cross-Validation:", mean_classification_error)
```

Evaluation of the results

Model	Dataset	Classification Error	True Positives	False Positives	False Negatives	True Negatives	ROC AUC	Accuracy
BP	Bank	10.19%	48	40	44	692	0.7335	89.81%
BP	Banknote Authentication	0.0%	127	0	0	148	1.0	100%
BP	Separable Ring	0.96%	4628	57	39	5276	0.9905	99.04%
BP	Merged Ring	5.26%	4226	85	441	5248	0.9448	94.74%
SVM	Bank	9.10%	35	18	57	714	0.92	90.90%
SVM	Banknote Authentication	0.0%	127	0	0	148	1.0	100%
SVM	Separable Ring	0.52%	4658	43	9	5290	1.0	99.48%
SVM	Merged Ring	2.12%	4480	25	187	5308	1.0	97.88%
MLR	Bank	9.22%	29	13	63	719	0.6487	90.78%
MLR	Banknote Authentication	2.18%	127	6	0	142	0.9797	97.82%
MLR	Separable Ring	47.11%	0	44	4667	5289	0.4959	52.89%
MLR	Merged Ring	46.67%	0	0	4667	5333	0.5	53.33%

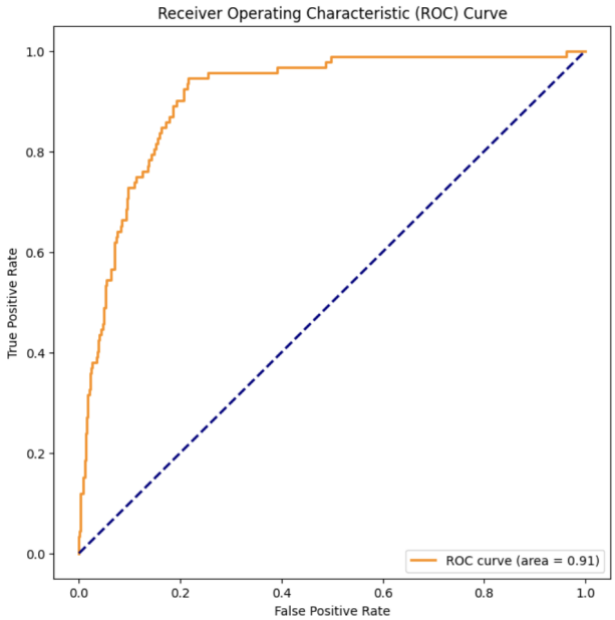
Comparison and analysis of the results:

Bank Dataset:
BP: Shows good performance with 89.81% accuracy and a decent ROC AUC of 0.7335.

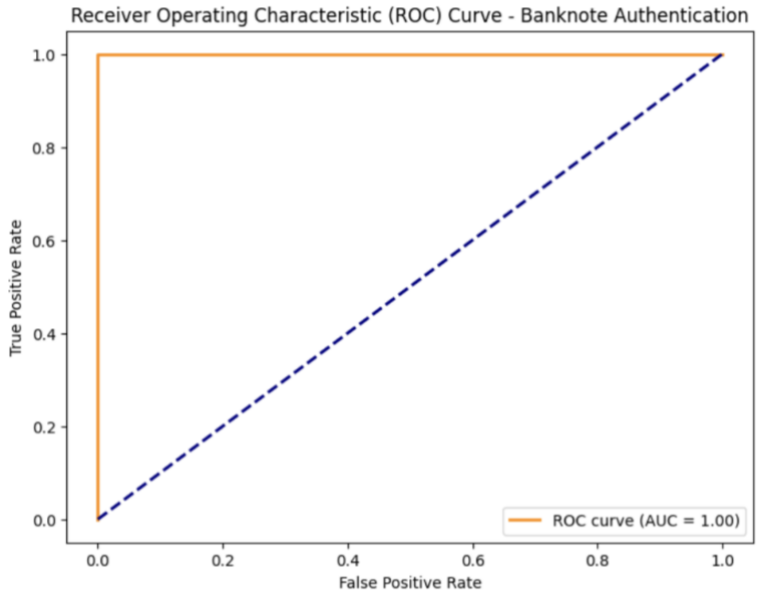


SVM: Comparable accuracy to BP (90.90%) with a higher ROC AUC of 0.92, indicating better discrimination between classes.

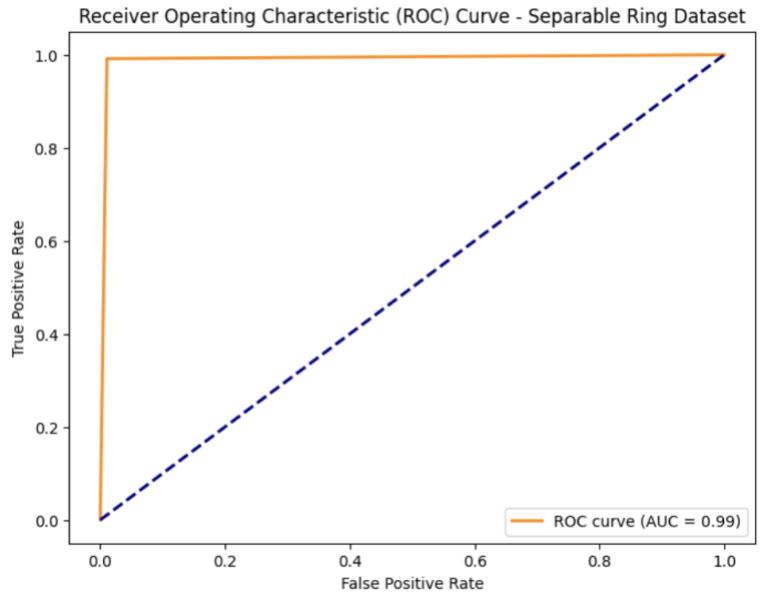
MLR: Similar accuracy to BP and SVM but lower ROC AUC (0.6487), indicating weaker performance in distinguishing between classes.



Banknote Authentication:
All models perform exceptionally well, with perfect accuracy (100%) and ROC AUC scores (1.0 for BP and SVM, 0.9797 for MLR).

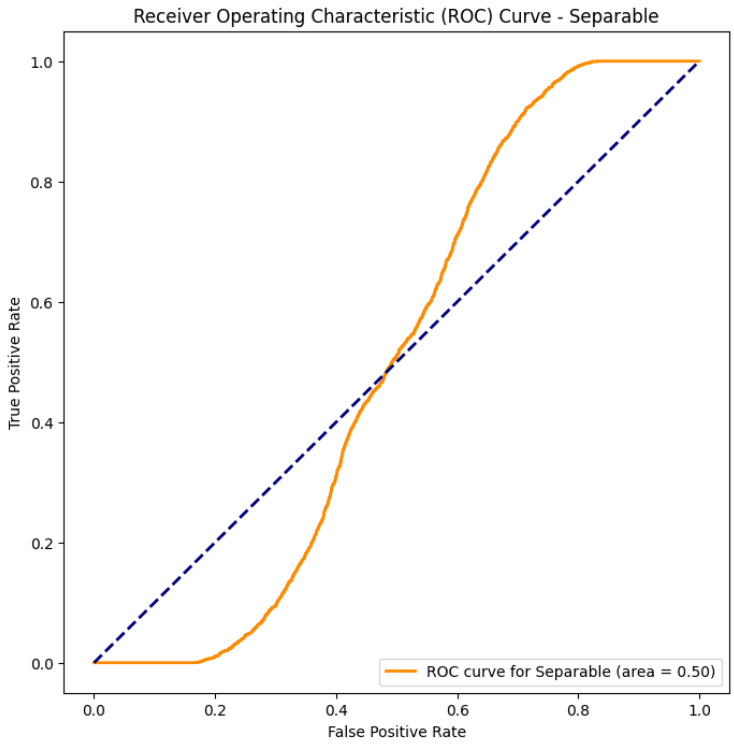


Separable Ring Dataset:
BP: Shows excellent performance with nearly perfect accuracy and ROC AUC.

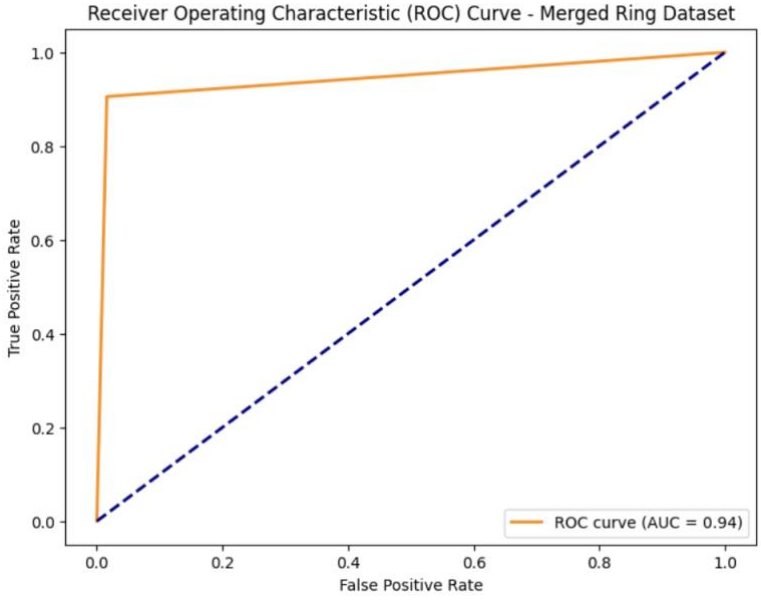


SVM: Also exhibits near-perfect metrics, aligning with BP's performance.

MLR: Significantly lower performance, with only 52.89% accuracy and a ROC AUC score barely better than random guessing (0.4959).

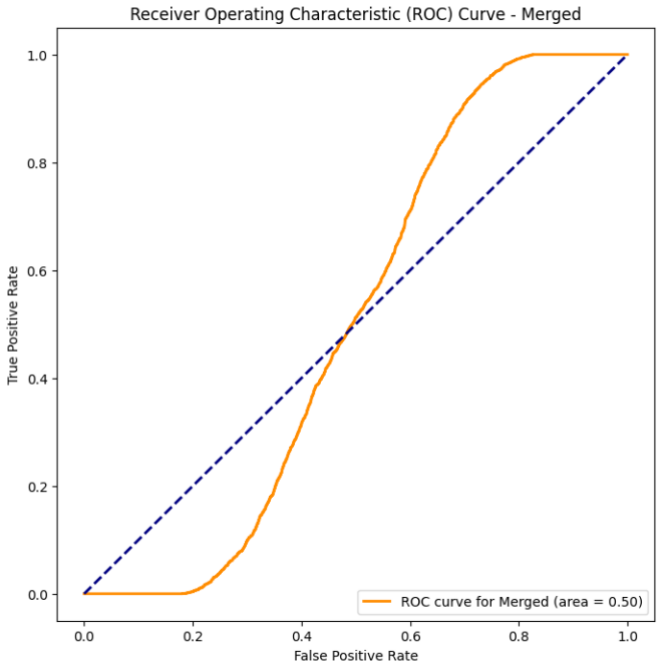


Merged Ring Dataset:
BP: Good accuracy (94.74%) and ROC AUC score (0.9448).



SVM: Highest accuracy (97.88%) and perfect ROC AUC (1.0), indicating superior performance.

MLR: Like with the Separable Ring, MLR lags behind, with low accuracy (53.33%) and ROC AUC (0.5).



Discussion

Interpretation of Results: The SVM model exhibited superior performance across most datasets, particularly in the ring datasets where its ability to handle complex, non-linear separations was evident. The BP model, with its neural network architecture, also performed well, especially in capturing intricate patterns in data. However, its performance was less consistent compared to SVM. The MLR model excelled in simpler, linearly separable tasks like the banknote dataset but struggled with the more complex ring datasets.

Comparison between SVM, BP, and MLR: SVM's robustness and versatility across different types of datasets highlight its strength as a classifier. BP, while effective, showed variability, suggesting its dependency on the nature of the dataset and the fine-tuning of its parameters. MLR's inconsistent performance underscores its limitations in handling complex classification tasks, suggesting its suitability for more straightforward, linear problems.

Challenges Encountered and Solutions:

I had trouble with understanding the ROC curve for our SVM model, but Professor Jordi helped me clear it up. Finding the right parameters for my models was hard because I had to test many different options to see what worked best with our data and also integrate them in the code was a challenge itself. Besides that, I was surprised when the banknote test showed perfect results. I had to double-check to make sure our data and results were reliable and not just because of a mistake or the model being too specific to the training data.

GITHUB LINK: https://github.com/sebastianbuzdugan/A2-NeuralNetworks