Lab 4 report for Machine Learning

Organtzoglou Evangelos Haralambos Malakoudis
Christodoulos Zerdalis

31/10/2024

1 Introduction and Background of the given paper

The paper "Comparative Analysis of Machine Learning Algorithms for Prediction of Smart Grid Stability" evaluates various machine learning (ML) algorithms to determine the optimal approach for predicting smart grid (SG) stability, a critical aspect of modern energy management. Using a publicly available dataset from the UCI machine learning repository, it explores the performance of Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Logistic Regression, Naive Bayes, Neural Networks, and Decision Trees. The study emphasizes the unique role ML models play in SG stability, especially when compared to deep learning models, given the size constraints of many SG datasets

2 Paper results

Classifier	Testing Accuracy
SVM(Polynomial Kernel)	91.20
SVM (RBF Kernel)	87.46
SVM (Sigmoid Kernel)	39.33
KNN	78.06
Logistic Regression	88.23
Naive Bayes	97.53
Decision Tree	99.96
Neural Network	98.13

Figure 1: Classifier Accuracy Results

3 Implementing the methods to the same dataset using the professor's code

Using the professor's code that implements the same methods to the same dataset we get some interesting results.

Python Code from Jupyter Notebook

```
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score,
    classification_report

# Load the dataset
file_path = our\file_path
data = pd.read_csv(file_path)

# Split the dataset into features and target variable
X = data.drop(columns='stabf')
y = data['stabf']
y = y.map({'unstable': 0, 'stable': 1})

# Split the data 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)

# List of classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(),
    "Logistic Regression": LogisticRegression(max_iter=1000)
        ,
    "Random Forest": RandomForestClassifier(),
    "Naive Bayes": GaussianNB(),
    "Support Vector Machine": SVC(kernel='linear'),
    "XGBoost": XGBClassifier(use_label_encoder=False,
        eval_metric='logloss'),
    "Neural Networks": MLPClassifier(max_iter=1000)
}

# Create a function to evaluate and report accuracy
```

```
def evaluate_classifier(classifier, X_train, X_test, y_train
    , y_test):
    # Fit the model
    classifier.fit(X_train, y_train)
    # Make predictions
    y_pred = classifier.predict(X_test)
    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    # Get classification report
    report = classification_report(y_test, y_pred,
        output_dict=True)
    return accuracy, report

# Evaluate each classifier and store results
accuracy_report = {}
for name, clf in classifiers.items():
    accuracy, report = evaluate_classifier(clf, X_train,
        X_test, y_train, y_test)
    accuracy_report[name] = {
        "Accuracy": accuracy,
        "Report": report
    }

# Prepare a summary DataFrame for accuracy
accuracy_summary = pd.DataFrame({
    "Classifier": list(accuracy_report.keys()),
    "Accuracy": [value["Accuracy"] for value in
        accuracy_report.values()],
})

# Display the accuracy summary
print(accuracy_summary)
```

4 Results from the code

Classifier	Accuracy (Your Results)
Decision Tree	99.95%
Logistic Regression	90.00%
Random Forest	99.95%
Naive Bayes	97.35%
SVM	94.55%
XGBoost	99.65%
Neural Networks	99.10%

Figure 2: Classifier Accuracy Results

5 Implementing the methods to our dataset

We have another dataset about Credit Approvals from Australia (named Australia.csv aka Australian Credit Approval dataset). Implementing that same methods that the paper used with its dataset which are : SVM, KNN, Logistic Regression, Naive Bayes, Neural Networks, and Decision Trees with Decision Trees showing the best results on the UCI dataset. We are going to be using this code:

Python Code from Jupyter Notebook

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score,
    classification_report

# Load the Australian dataset

file_path = \our\file_path
data = pd.read_csv(file_path)

# Assuming the last column is the target variable
X = data.iloc[:, :-1]  # All columns except the last one
y = data.iloc[:, -1]   # Last column as the target

# Split the data 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)

# List of classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(),
```

```
    "Logistic Regression": LogisticRegression(max_iter=1000)
        ,
    "Random Forest": RandomForestClassifier(),
    "Naive Bayes": GaussianNB(),
    "Support Vector Machine": SVC(kernel='linear'),
    "XGBoost": XGBClassifier(use_label_encoder=False,
        eval_metric='logloss'),
    "Neural Networks": MLPClassifier(max_iter=1000)
}

# Create a function to evaluate and report accuracy
def evaluate_classifier(classifier, X_train, X_test, y_train
    , y_test):
    # Fit the model
    classifier.fit(X_train, y_train)
    # Make predictions
    y_pred = classifier.predict(X_test)
    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    # Get classification report
    report = classification_report(y_test, y_pred,
        output_dict=True)
    return accuracy, report

# Evaluate each classifier and store results
accuracy_report = {}
for name, clf in classifiers.items():
    accuracy, report = evaluate_classifier(clf, X_train,
        X_test, y_train, y_test)
    accuracy_report[name] = {
        "Accuracy": accuracy,
        "Report": report
    }

# Prepare a summary DataFrame for accuracy
accuracy_summary = pd.DataFrame({
    "Classifier": list(accuracy_report.keys()),
    "Accuracy": [value["Accuracy"] for value in
        accuracy_report.values()],
})

# Display the accuracy summary
print(accuracy_summary)
```

6 Results from the code

We see that Decision Trees have exceptional results here too, with the Random Forest algorithm having the best results out of every other algorithm.

	Classifier	Accuracy
0	Decision Tree	0.855072
1	Logistic Regression	0.847826
2	Random Forest	0.884058
3	Naive Bayes	0.818841
4	Support Vector Machine	0.855072
5	XGBoost	0.862319
6	Neural Networks	0.724638

Figure 3: Classifier Accuracy Results

7 Conclusion

Studying the paper we see that Decision Tree classification delivers the best performance for SG stability prediction, achieving 99.96% accuracy with high precision, recall, and F1-scores. This method outperforms others such as Neural Networks, SVM, and KNN, attributed largely to the dataset's manageable size and the algorithm's suitability for classification tasks in SGs which is the subject studied in the paper.

The results provided by the professor's implementation, which employs the same algorithms on the same dataset as the paper, exhibit remarkably high accuracies across almost all models. Notably, both the Decision Tree and Random Forest classifiers achieve near-perfect accuracies, suggesting an extremely effective fit to the dataset characteristics or possibly an overfitting scenario if not cross-validated. The Neural Networks and XGBoost also show very high performance, indicating that advanced ensemble methods and neural architectures are well-suited for this data. In contrast, traditional models like Logistic Regression and Naive Bayes, while still performing admirably, do not reach the same heights, potentially due to their linear nature and simpler decision boundaries. These results underscore the importance of choosing appropriate algorithms based on the specific features and complexity of the data being analyzed.

The results from our implementation using the Australian dataset show a variation in model performance, with Decision Tree and Random Forest classifiers achieving the highest accuracy scores among the tested models. This could suggest that tree-based methods are particularly well-suited for the structure and nature of this dataset, potentially due to their ability to handle non-linear relationships and feature interactions effectively. The lower performance of the

Neural Networks might indicate issues related to overfitting, the need for more training epochs, or the requirement for hyperparameter tuning.

Comparing these results to those from the paper, which used a different dataset focused on Smart Grid Stability, we observe a generally higher performance across all classifiers in the paper's results. This difference can be attributed to several factors such as the nature of the data, the complexity of the problems being solved, and possibly the tuning of the machine learning models. For example, the sigmoid SVM's significantly lower accuracy in the paper suggests a poor fit for that data's distribution or possible issues with hyperparameter settings, which seems less pronounced in the results from our Australian dataset testing. These comparisons highlight the importance of context-specific model tuning and validation to ensure optimal performance across different datasets.

References

[1] Ali Kashif Bashir, Suleman Khan, Prabadevi B, N. Deepa, Waleed S. Alnumay, Thippa Reddy G, and Praveen Kumar Reddy M. "Comparative Analysis of Machine Learning Algorithms for Prediction of Smart Grid Stability."