Applied Machine Learning

Assignment 2

Submitted By

Mallidi Akhil Reddy

2024TM93056

Problem Definition:

The goal of this assignment is to perform a classification analysis on two real-world datasets obtained from the UCI Machine Learning Repository:

Banknote Authentication Dataset: The goal here is to classify whether a banknote is authentic or not.

Haberman's Survival Dataset: The goal here is to predict patient survival status based on surgery data.

The following machine learning algorithms are employed for both tasks:

1) Naive Bayes
2) Logistic Regression
3) Support Vector Machine (SVM)
4) Random Forest

Mathematical Details:

Naive Bayes: Naive Bayes is probabilistic classifier algorithm based on Bayes' Theorem. It assumes independence among features.

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Logistic Regression: It's a linear model algorithm for binary classification. In this Model parameters are estimated using maximum likelihood estimation.

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Support Vector Machine (SVM): This algorithm Maximizes the margin between classes.

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{subject to:} \quad & y^i(\mathbf{w}^T\mathbf{x}^i + b) \geq 1 \quad \forall i \end{aligned}$$

Dataset Description:

Banknote Authentication Dataset:

- Number of features: 4 (variance, skewness, curtosis, entropy)
- Number of instances: 1372
- Number of classes: 2 (authentic, counterfeit)
- Preprocessing steps:
    1) Scaling the features using StandardScaler to have zero mean and unit variance. This can help improve the performance of algorithms like Logistic Regression and SVM.
    2) No missing values were found in the dataset.

Haberman's Survival:

- Number of features: 3 (age, year of operation, number of positive axillary nodes)
- Number of instances: 306
- Number of classes: 2 (survived 5 years or more, died within 5 years)
- Preprocessing steps:
  1) Scaling the features using StandardScaler, as it is important because the features have different ranges, and scaling can prevent features with larger ranges.
  2) No missing values were found.

Experiments:

Experimental Setup and Train-Test Split:

For each dataset, I used an 80/20 train-test split. This means that 80% of the data was used for training the models, and 20% was held out for evaluating their performance.

Hyperparameter Tuning:

Naive Bayes: Naive Bayes generally doesn't have hyperparameters that require extensive tuning.

Logistic Regression:

- Hyperparameter: Regularization strength (C)
- Tuning method: Grid search
- Parameter range: C = [0.001, 0.01, 0.1, 1, 10, 100]
- Regularization type: l2

SVM:

- Hyperparameters: Kernel type, C (regularization), gamma (for RBF kernel)
- Tuning method: Grid search
- Parameter ranges:
    - Kernel: ['linear', 'rbf']
    - C: [0.1, 1, 10, 100]
    - gamma (for RBF): [0.001, 0.01, 0.1, 1]

Results on Test Data:

Metrics are rounded to 4 decimals

Banknote Authentication Dataset:

Model	**Accuracy**	**Macro F1-Score**	**Macro Precision**	**Macro Recall**
Naive Bayes	0.8582	0.856	0.8571	0.8551
Logistic Regression	0.9855	0.9853	0.9841	0.9869
Support Vector Machine (SVM)	1.0	1.0	1.0	1.0

Haberman Survival Dataset:

Model	**Accuracy**	**Macro F1-Score**	**Macro Precision**	**Macro Recall**
Naive Bayes	0.7581	0.57	0.686	0.572
Logistic Regression	0.7581	0.5338	0.7147	0.5516
Support Vector Machine (SVM)	0.7258	0.5127	0.5772	0.5299

Conclusion:

The classification of the Banknote Authentication dataset yielded remarkably high performance across all evaluated models. Support Vector Machine (SVM) achieved a perfect score with 100% accuracy, F1-Score, precision, and recall on the test data. Logistic Regression also demonstrated strong performance with an accuracy of 0.9855, a macro F1-Score of 0.9853, a macro precision of 0.9841, and a macro recall of 0.9869. Naive Bayes, while slightly lower, still provided a respectable accuracy of 0.8582, a macro F1-Score of 0.856, a macro precision of 0.8571, and a macro recall of 0.8551.

The classification of the Haberman's Survival dataset proved to be a more challenging task. All three models exhibited lower performance compared to the banknote dataset. Logistic Regression and Naive Bayes achieved the same accuracy of 0.7581. However, Logistic Regression showed a slightly better macro F1-Score (0.5338) compared to Naive Bayes (0.57), although Naive Bayes had a higher macro precision (0.686 vs 0.7147 for Logistic Regression) but a lower macro recall (0.572 vs 0.5516 for Logistic Regression). SVM performed slightly worse on this dataset with an accuracy of 0.7258 and a macro F1-Score of 0.5127.

Future Work:

Incorporate feature selection or transformation like PCA to explore latent patterns.

Employing stratified k-fold cross-validation to ensure stable performance estimates across imbalanced data splits.

Code

```
# importing all required libraries

import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score

# Load the datasets

# Banknote Authentication Dataset

banknote_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-
learning-databases/00267/data_banknote_authentication.txt',
header=None, names=['variance', 'skewness', 'curtosis', 'entropy',
'class'])
X_banknote = banknote_data.drop('class', axis=1)
y_banknote = banknote_data['class']

print("\n Banknote Dataset Description:")
print(f"  Number of features: {X_banknote.shape[1]}")
print(f"  Number of instances: {X_banknote.shape[0]}")
print(f"  Number of classes: {len(y_banknote.unique())}")
print("  Features:", list(X_banknote.columns))
```

```
 Banknote Dataset Description:
  Number of features: 4
  Number of instances: 1372
  Number of classes: 2
  Features: ['variance', 'skewness', 'curtosis', 'entropy']
```

```
# Haberman Dataset

haberman_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-
learning-databases/haberman/haberman.data', header=None, names=['age',
'year', 'nodes', 'survival'])
X_haberman = haberman_data.drop('survival', axis=1)
y_haberman = haberman_data['survival'].map({1: 1, 2: 0})

# Dataset Description
print("\n Haberman Dataset Description:")
print(f"  Number of features: {X_haberman.shape[1]}")
print(f"  Number of instances: {X_haberman.shape[0]}")
print(f"  Number of classes: {len(y_haberman.unique())}")
print("  Features:", list(X_haberman.columns))
```

```
 Haberman Dataset Description:
  Number of features: 3
  Number of instances: 306
  Number of classes: 2
  Features: ['age', 'year', 'nodes']

# Preprocessing datasets

scaler_banknote_data = StandardScaler()
X_banknote_scaled_data =
scaler_banknote_data.fit_transform(X_banknote)

scaler_haberman_data = StandardScaler()
X_haberman_scaled_data =
scaler_haberman_data.fit_transform(X_haberman)

# Train-test split with 80% train data and 20% test data

X_banknote_train, X_banknote_test, y_banknote_train, y_banknote_test =
train_test_split(X_banknote_scaled_data, y_banknote, test_size=0.2,
random_state=42, stratify=y_banknote)
X_haberman_train, X_haberman_test, y_haberman_train, y_haberman_test =
train_test_split(X_haberman_scaled_data, y_haberman, test_size=0.2,
random_state=42, stratify=y_haberman)

# Model Training and Evaluation

results = {}

results['banknote'] = {}

# Banknote Authentication Models

# 1. Naive Bayes

print("Banknote Authentication Dataset \n")
print("Training Naive Bayes \n")

nb_banknote = GaussianNB()
nb_banknote.fit(X_banknote_train, y_banknote_train)
y_banknote_pred_nb = nb_banknote.predict(X_banknote_test)
results['banknote']['Naive Bayes'] = {
    'Accuracy': round(accuracy_score(y_banknote_test,
y_banknote_pred_nb), 4),
    'Macro F1-Score': round(f1_score(y_banknote_test,
y_banknote_pred_nb, average='macro'), 4),
    'Macro Precision': round(precision_score(y_banknote_test,
y_banknote_pred_nb, average='macro'), 4),
    'Macro Recall': round(recall_score(y_banknote_test,
y_banknote_pred_nb, average='macro'), 4)
```

```
}

print(f"Naive Bayes Results: {results['banknote']['Naive Bayes']} \n")
print("\n----------------------------------------------------------\n")

# 2. Logistic Regression

print("Training Logistic Regression \n")

lr_banknote = LogisticRegression(random_state=42)
param_grid_lr_banknote = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
'solver': ['liblinear']}
grid_search_lr_banknote = GridSearchCV(lr_banknote,
param_grid_lr_banknote, cv=5, scoring='f1_macro')
grid_search_lr_banknote.fit(X_banknote_train, y_banknote_train)
best_lr_banknote = grid_search_lr_banknote.best_estimator_
y_banknote_pred_lr = best_lr_banknote.predict(X_banknote_test)
results['banknote']['Logistic Regression'] = {
    'Accuracy': round(accuracy_score(y_banknote_test,
y_banknote_pred_lr), 4),
    'Macro F1-Score': round(f1_score(y_banknote_test,
y_banknote_pred_lr, average='macro'), 4),
    'Macro Precision': round(precision_score(y_banknote_test,
y_banknote_pred_lr, average='macro'), 4),
    'Macro Recall': round(recall_score(y_banknote_test,
y_banknote_pred_lr, average='macro'), 4)
}

print(f"Logistic Regression Results: {results['banknote']['Logistic
Regression']} \n")
print(f"Best Logistic Regression parameters:
{grid_search_lr_banknote.best_params_} \n")
print("\n----------------------------------------------------------\n")

# 3. Support Vector Machine (SVM)

print("Training SVM \n")

svm_banknote = SVC(random_state=42)
param_grid_svm_banknote = {'C': [0.1, 1, 10, 100], 'kernel':
['linear', 'rbf'], 'gamma': ['scale', 'auto']}
grid_search_svm_banknote = GridSearchCV(svm_banknote,
param_grid_svm_banknote, cv=5, scoring='f1_macro')
grid_search_svm_banknote.fit(X_banknote_train, y_banknote_train)
best_svm_banknote = grid_search_svm_banknote.best_estimator_
y_banknote_pred_svm = best_svm_banknote.predict(X_banknote_test)
results['banknote']['SVM'] = {
    'Accuracy': round(accuracy_score(y_banknote_test,
y_banknote_pred_svm), 4),
    'Macro F1-Score': round(f1_score(y_banknote_test,
```

```
y_banknote_pred_svm, average='macro'), 4),
    'Macro Precision': round(precision_score(y_banknote_test,
y_banknote_pred_svm, average='macro'), 4),
    'Macro Recall': round(recall_score(y_banknote_test,
y_banknote_pred_svm, average='macro'), 4)
}
print(f"SVM Results: {results['banknote']['SVM']} \n")
print(f"Best SVM parameters: {grid_search_svm_banknote.best_params_} \
n")
print("\n---------------------------------------------------------\n")

print("Haberman Dataset \n")

results['haberman'] = {}

# 1. Naive Bayes

print("Training Naive Bayes \n")

nb_haberman = GaussianNB()
nb_haberman.fit(X_haberman_train, y_haberman_train)
y_haberman_pred_nb = nb_haberman.predict(X_haberman_test)
results['haberman']['Naive Bayes'] = {
    'Accuracy': round(accuracy_score(y_haberman_test,
y_haberman_pred_nb), 4),
    'Macro F1-Score': round(f1_score(y_haberman_test,
y_haberman_pred_nb, average='macro'), 4),
    'Macro Precision': round(precision_score(y_haberman_test,
y_haberman_pred_nb, average='macro'), 4),
    'Macro Recall': round(recall_score(y_haberman_test,
y_haberman_pred_nb, average='macro'), 4)
}

print(f"Naive Bayes Results: {results['haberman']['Naive Bayes']} \n")
print("\n---------------------------------------------------------\n")

# 2. Logistic Regression

print("Training Logistic Regression \n")

lr_haberman = LogisticRegression(random_state=42)
param_grid_lr_haberman = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
'solver': ['liblinear']}
grid_search_lr_haberman = GridSearchCV(lr_haberman,
param_grid_lr_haberman, cv=5, scoring='f1_macro')
grid_search_lr_haberman.fit(X_haberman_train, y_haberman_train)
best_lr_haberman = grid_search_lr_haberman.best_estimator_
y_haberman_pred_lr = best_lr_haberman.predict(X_haberman_test)
results['haberman']['Logistic Regression'] = {
    'Accuracy': round(accuracy_score(y_haberman_test,
```

```
y_haberman_pred_lr), 4),
    'Macro F1-Score': round(f1_score(y_haberman_test,
y_haberman_pred_lr, average='macro'), 4),
    'Macro Precision': round(precision_score(y_haberman_test,
y_haberman_pred_lr, average='macro'), 4),
    'Macro Recall': round(recall_score(y_haberman_test,
y_haberman_pred_lr, average='macro'), 4)
}

print(f"Logistic Regression Results: {results['haberman']['Logistic
Regression']} \n")
print(f"Best Logistic Regression parameters:
{grid_search_lr_haberman.best_params_} \n")
print("\n---------------------------------------------------------\n")

# 3. Support Vector Machine (SVM)

print("Training SVM \n")

svm_haberman = SVC(random_state=42)
param_grid_svm_haberman = {'C': [0.1, 1, 10, 100], 'kernel':
['linear', 'rbf'], 'gamma': ['scale', 'auto']}
grid_search_svm_haberman = GridSearchCV(svm_haberman,
param_grid_svm_haberman, cv=5, scoring='f1_macro')
grid_search_svm_haberman.fit(X_haberman_train, y_haberman_train)
best_svm_haberman = grid_search_svm_haberman.best_estimator_
y_haberman_pred_svm = best_svm_haberman.predict(X_haberman_test)
results['haberman']['SVM'] = {
    'Accuracy': round(accuracy_score(y_haberman_test,
y_haberman_pred_svm), 4),
    'Macro F1-Score': round(f1_score(y_haberman_test,
y_haberman_pred_svm, average='macro'), 4),
    'Macro Precision': round(precision_score(y_haberman_test,
y_haberman_pred_svm, average='macro'), 4),
    'Macro Recall': round(recall_score(y_haberman_test,
y_haberman_pred_svm, average='macro'), 4)
}

print(f"SVM Results: {results['haberman']['SVM']} \n")
print(f"Best SVM parameters: {grid_search_svm_haberman.best_params_} \
n")
print("\n---------------------------------------------------------\n")
```

```
Banknote Authentication Dataset

Training Naive Bayes

Naive Bayes Results: {'Accuracy': 0.8582, 'Macro F1-Score': 0.856,
'Macro Precision': 0.8571, 'Macro Recall': 0.8551}
```

```
----------------------------------------------------------
Training Logistic Regression
Logistic Regression Results: {'Accuracy': 0.9855, 'Macro F1-Score':
0.9853, 'Macro Precision': 0.9841, 'Macro Recall': 0.9869}
Best Logistic Regression parameters: {'C': 100, 'solver': 'liblinear'}


----------------------------------------------------------
Training SVM
SVM Results: {'Accuracy': 1.0, 'Macro F1-Score': 1.0, 'Macro
Precision': 1.0, 'Macro Recall': 1.0}
Best SVM parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}

----------------------------------------------------------
Haberman Dataset
Training Naive Bayes
Naive Bayes Results: {'Accuracy': 0.7581, 'Macro F1-Score': 0.57,
'Macro Precision': 0.686, 'Macro Recall': 0.572}

----------------------------------------------------------
Training Logistic Regression
Logistic Regression Results: {'Accuracy': 0.7581, 'Macro F1-Score':
0.5338, 'Macro Precision': 0.7147, 'Macro Recall': 0.5516}
Best Logistic Regression parameters: {'C': 0.001, 'solver':
'liblinear'}

----------------------------------------------------------
Training SVM
SVM Results: {'Accuracy': 0.7258, 'Macro F1-Score': 0.5127, 'Macro
Precision': 0.5772, 'Macro Recall': 0.5299}
Best SVM parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
```

Thank you