a2-multiclass-classification

March 26, 2024

1 Multi-Class Classification

[576]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from sklearn.metrics import accuracy_score, precision_score, f1_score,␣
 ↪confusion_matrix, recall_score
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
```

[577]:
```
data = pd.read_csv('DataB1.csv')
```

[578]:
```
data.head()
```

[578]:
```
   sepallength  sepalwidth  petallength  petalwidth        class
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
```

[579]:
```
data.columns
```

[579]:
```
Index(['sepallength', 'sepalwidth', 'petallength', 'petalwidth', 'class'],
dtype='object')
```

[580]:
```
data.shape
```

[580]:
```
(150, 5)
```

[581]:
```
data.isnull().sum()
```

[581]:
```
sepallength    0
sepalwidth     0
```

```
petallength    0
petalwidth     0
class          0
dtype: int64
```

[582]:
```
# Assuming the last column is the label
X = data.drop(columns=['class'])
y = data['class']
```

[583]:
```
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
```

[584]:
```
# Encode class labels to integers
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

[585]:
```
# Splits the dataset into training (70%) and test (30%) sets.
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y_encoded,␣
  ↪test_size=0.3, random_state=42)
```

2 1. Describe and develop the training and classification procedures by using the "one-versus-all" and "one-versus-one" strategy for SVM.

One-versus-One (OvO) Strategy for SVM

- Description: In the one-versus-one (OvO) strategy, a binary classifier is trained for every pair of classes. If there are N classes, the results in $\frac{N\times(N-1)}{2}$ classifiers.
- Procedure:
  1. Training: Train a binary SVM classifier for each pair of classes.
  2. Classification: To classify a new sample, run it through all classifiers. The class that wins the most duels (is chosen most frequently by the classifiers) is assigned to the sample.

One-versus-All (OvA) Strategy for SVM

- Description: In the one-versus-all (OvA) strategy, a single classifier is trained per class to distinguish the samples of that class from samples of all other classes. For a problem with N classes, N separate binary classifiers are trained. For each classifier, the class it represents is treated as a positive class, while all other classes are merged into a negative class.
- Procedure:
  1. Training: For each class i, train a binary SVM classifier where class i samples are considered positive, and all other samples are considered negative.
  2. Classification: To classify a new sample, run it through all N classifiers. The classifier that outputs the highest confidence score (distance from the decision boundary) assigns its class to the sample.

3 2. Classify data set B by using binary Support Vector Machine classifiers with linear kernel and default parameters. Randomly split the data into 70% training and 30% test set. Report the classification overall accuracy, precision, recall, F1-score, and the confusion matrix of the classification results on the test set.

[586]:
```
# Initialize the SVM One-vs-One classifier with a linear kernel
model_one_one = SVC(decision_function_shape='ovo', kernel = 'linear')
# Train the classifier
model_one_one.fit(X_train, y_train)
# Predict on the test set
y_pred_one_one = model_one_one.predict(X_test)

# Calculate evaluation metrics
accuracy_one_one = accuracy_score(y_test, y_pred_one_one)
precision_one_one = precision_score(y_test, y_pred_one_one, average='weighted')
recall_one_one = recall_score(y_test, y_pred_one_one, average='weighted')
f1_one_one = f1_score(y_test, y_pred_one_one, average='weighted')
conf_matrix_one_one = confusion_matrix(y_test, y_pred_one_one)
```

[587]:
```
# Initialize the SVM One-vs-All classifier with a linear kernel
model_one_all = SVC(decision_function_shape='ovr', kernel = 'linear')
# Train the classifier
model_one_all.fit(X_train, y_train)
# Predict on the test set
y_pred_one_all = model_one_all.predict(X_test)

# Calculate evaluation metrics
accuracy_one_all = accuracy_score(y_test, y_pred_one_all)
precision_one_all = precision_score(y_test, y_pred_one_all, average='weighted')
recall_one_all = recall_score(y_test, y_pred_one_all, average='weighted')
f1_one_all = f1_score(y_test, y_pred_one_all, average='weighted')
conf_matrix_one_all = confusion_matrix(y_test, y_pred_one_all)
```

[588]:
```
print("One-vs-One:")
print(f"Accuracy for one-versus-one: {accuracy_one_one}")
print(f"Precision for one-versus-one: {precision_one_one}")
print(f"Recall for one-versus-one: {recall_one_one}")
print(f"F1-Score for one-versus-one: {f1_one_one}")
print("Confusion Matrix for one-versus-one:")
print(conf_matrix_one_one)
```

```
One-vs-One:
Accuracy for one-versus-one: 0.9777777777777777
Precision for one-versus-one: 0.9793650793650793
Recall for one-versus-one: 0.9777777777777777
```

```
F1-Score for one-versus-one: 0.9777448559670783
Confusion Matrix for one-versus-one:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
```

[589]:
```
print("One-vs-All:")
print(f"Accuracy for one-versus-all: {accuracy_one_all}")
print(f"Precision for one-versus-all: {precision_one_all}")
print(f"Recall for one-versus-all: {recall_one_all}")
print(f"F1-Score for one-versus-all: {f1_one_all}")
print("Confusion Matrix for one-versus-all:")
print(conf_matrix_one_all)
```

```
One-vs-All:
Accuracy for one-versus-all: 0.9777777777777777
Precision for one-versus-all: 0.9793650793650793
Recall for one-versus-all: 0.9777777777777777
F1-Score for one-versus-all: 0.9777448559670783
Confusion Matrix for one-versus-all:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
```

When using a Support Vector Machine (SVM) with a 'linear' kernel or without specifying kernel settings, choosing between One-vs-One (OvO) and One-vs-All (OvA) strategies can result in different classification outcomes due to their distinct approaches to handling multi-class classification

[590]:
```
# Initialize the SVM classifier with linear kernel and default parameters
svm_classifier = SVC(kernel='linear')

# Train the SVM classifier on the training data
svm_classifier.fit(X_train, y_train)

# Predict labels for the test set
y_pred = svm_classifier.predict(X_test)
```

[591]:
```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_pred)


print(f"SVM classifier Accuracy: {accuracy}")
print(f"SVM classifier Precision: {precision}")
print(f"SVM classifier Recall: {recall}")
print(f"SVM classifier F1-Score: {f1}")
print("SVM classifier Confusion Matrix:")
print(conf_matrix)
```

```
SVM classifier Accuracy: 0.9777777777777777
SVM classifier Precision: 0.9793650793650793
SVM classifier Recall: 0.9777777777777777
SVM classifier F1-Score: 0.9777448559670783
SVM classifier Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
```

The results for One-vs-One and One-vs-All strategies mentioned above are based on a normalized dataset. Without data normalization, the outputs for both strategies could differ due to variations in feature scales.

The effectiveness of One-vs-One and One-vs-All strategies in SVM classification can be significantly influenced by whether the dataset is normalized. Without normalization, the disparities in scale among features can lead to different outcomes for both strategies, potentially affecting their performance and accuracy.

4 3. How does the decision tree classifier deal with the multi-class problem? Classify data set B using decision tree with default parameters, report the classification results. Comment and compare the methods of SVM and decision tree.

For Decision Tree classifiers, both One-vs-One (OvO) and One-vs-All (OvA) strategies can be applied:

- **OvO**: A binary decision tree is created for each class pair, each trained to differentiate between its two classes. The final prediction is made based on the majority vote from all trees.
- **OvA**: A binary decision tree is created for each class to distinguish it from all others. The final prediction is the class that a decision tree identifies with the highest confidence.

A Decision Tree classifier handles multi-class problems directly by:

- **Natively Supporting Multi-Class**: Can classify instances into multiple classes without needing special strategies.
- **Hierarchical Splitting**: Uses feature values to recursively split the data, creating branches and leaves that correspond to different classes.
- **Purity-Based Decisions**: Chooses splits based on measures like Gini impurity or entropy to best separate classes.
- **Binary Splits for Multi-Class**: Although splits are binary, the hierarchical nature allows for efficient multi-class classification.
- **Leaf Nodes Represent Classes**: Each leaf node predicts a class, guiding instances to the appropriate class based on their features.

[592]:
```
# Splits the dataset into training (70%) and test (30%) sets without feature␣
↪scaling
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.
↪3, random_state=42)
```

[593]:
```
# Initialize the Decision Tree classifier
dt_classifier  = DecisionTreeClassifier()

# Train the classifier
dt_classifier  = dt_classifier .fit(X_train, y_train)

# Predict on the test se
y_pred_dt  = dt_classifier .predict(X_test)
```

[594]:
```
# Calculate evaluation metrics
accuracy_dt = accuracy_score(y_test, y_pred_dt )
precision_dt = precision_score(y_test, y_pred_dt , average='weighted')
recall_dt = recall_score(y_test, y_pred_dt , average='weighted')
f1_dt = f1_score(y_test, y_pred_dt , average='weighted')
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt )


print(f"Decision Tree Accuracy: {accuracy_dt}")
print(f"Decision Tree Precision: {precision_dt}")
print(f"Decision Tree Recall: {recall_dt}")
print(f"Decision Tree F1 Score: {f1_dt}")
print(f"Decision Tree Confusion Matrix:\n{conf_matrix_dt}")
```

```
Decision Tree Accuracy: 1.0
Decision Tree Precision: 1.0
Decision Tree Recall: 1.0
Decision Tree F1 Score: 1.0
Decision Tree Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

- **Multi-Class Handling**:
  - **SVM**: Uses One-versus-All or One-versus-One strategies for multi-class classification, increasing complexity.
  - **Decision Tree**: Natively supports multi-class classification without additional complexity.
- **Performance**:
  - **SVM**: Often excels in precision and recall for linearly separable data, with the ability to handle high-dimensional spaces effectively.
  - **Decision Tree**: Can capture complex patterns but may overfit, affecting precision and recall.
- **Interpretability**:

  - **SVM**: Less intuitive due to abstract concepts like hyperplanes and support vectors.
  - **Decision Tree**: Highly interpretable with a clear visualization of decision paths.
- **Model Complexity and Overfitting**:
  - **SVM**: Requires careful selection of kernel and regularization to balance bias and variance.
  - **Decision Tree**: Prone to overfitting, especially with deep trees; requires pruning or constraints to ensure generalizability.

Overall, the choice between SVM and Decision Trees depends on the specific needs for interpretability, the nature of the dataset, and the trade-off between performance and model complexity.