T73 - Multiclass

March 6, 2021

1 Evaluation matrices (multiclass classification)

1.1 Setting up

- Iris data
- 3 classes
- 2 features
- Logistic regression

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Iris data
dataObj = load_iris()

# X data (features)
X = dataObj.data[:, [1, 2]]

# y data
y = dataObj.target

print(np.unique(y))

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1, stratify=y)

# Constructing a pipeline object
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LogisticRegression(random_state=0, C=1))])

pipe_lr.fit(X_train, y_train)
```

1.2 Confusion matrix

```
[ ]: from sklearn.metrics import confusion_matrix

y_pred = pipe_lr.predict(X_test)
confusion_matrix(y_true=y_test, y_pred=y_pred)
```

```
[ ]: import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(estimator=pipe_lr, X=X_test, y_true=y_test)
plt.show()
```

Compute class-wise (default) or sample-wise (samplewise=True) multilabel confusion matrix to evaluate the accuracy of a classification, and output confusion matrices for each class or sample.

```
[ ]: from sklearn.metrics import multilabel_confusion_matrix

#Multilabel confusion matrix
multilabel_confusion_matrix(y_true=y_test, y_pred=y_pred)
```

1.3 Accuracy, Precision, Recall, F1

```
[ ]: from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

# Accuracy
ACC = accuracy_score(y_true=y_test, y_pred=y_pred)
print(f"Accuracy:{ACC:6.3f}")

# Precision
PRE = precision_score(y_true=y_test, y_pred=y_pred, average='macro')
print(f"Precision (macro):{PRE:6.3f}")

# Recall
REC = recall_score(y_true=y_test, y_pred=y_pred, average='macro')
print(f"Recall (macro):{REC:6.3f}")

# F1
F1 = f1_score(y_true=y_test, y_pred=y_pred, average='macro')
print(f"F1 Score (macro):{F1:6.3f}")
```

```
[ ]: # Precision
PRE = precision_score(y_true=y_test, y_pred=y_pred, average='micro')
print(f"Precision (micro):{PRE:6.3f}")

# Recall
REC = recall_score(y_true=y_test, y_pred=y_pred, average='micro')
```

```
print(f"Recall (micro):{REC:6.3f}")

# F1
F1 = f1_score(y_true=y_test, y_pred=y_pred, average='micro')
print(f"F1 Score (micro):{F1:6.3f}")
```

1.4 ROC AUC

[ ]:
```
from sklearn.metrics import roc_auc_score

proba = pipe_lr.predict_proba(X_test)

#AUC Value
auc_score = roc_auc_score(y_true=y_test, y_score=proba, average='macro',␣
 ↪multi_class="ovr")
print(f"AUC:{auc_score:6.3f}")
```

1.5 Using precision in grid search

[ ]:
```
from sklearn.metrics import make_scorer

# Making score.
scorer = make_scorer(precision_score, average='micro')
```

[ ]:
```
pipe_lr.get_params()
```

[ ]:
```
from sklearn.model_selection import GridSearchCV

param_grid = { 'clf__C': [0.001, 0.01, 0.1, 1] }

# Grid search. Note the "scoring" argument
gs = GridSearchCV(estimator=pipe_lr,
                  param_grid=param_grid,
                  scoring=scorer,
                  cv=10,
                  n_jobs=-1)

gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

[ ]:
```
df = pd.DataFrame(gs.cv_results_)
df = df.sort_values(by=['rank_test_score'])
display(df)
```