Bài 4: HUẤN LUYỆN MÔ HÌNH VÀ LỖI

Hướng dẫn 3: Các độ đo đánh giá hiệu năng của mô hình phân lớp – Metrics

(Lưu ý: Bài này chỉ xét trường hợp có 2 nhãn lớp – Phân lớp nhị phân – Binary classification)

1. Độ chính xác dự đoán nhãn lớp – Accuracy score

Đối với bài toán phân lớp (lưu ý: mô hình hồi quy logistic thực chất là mô hình phân lớp) có đặc trưng là vector "nhãn lớp" – y – chứa các giá trị rời rạc ($y = [y_0;\ y_1;\ \ldots; y_{m-1}]\ với\ y_i \in \{c_j | j = \overline{1,k}\}$). Do vậy, các phép đo hiệu năng dựa vào sai số của mô hình hồi quy không áp dụng được trong trường hợp này. Vì vậy, chúng ta có một lớp các phép đo khác chuyên dụng để đánh giá hiệu năng dự đoán của mô hình phân lớp. Trong đó, phép đo độ chính xác – Accuracy – là một trong những phép đo phổ biến nhất.

Cho $y \in R^m$ và $\hat{y} \in R^m$ lần lượt là vector "nhãn lớp" thực tế và vector chứa nhãn lớp dự đoán của mô hình phân lớp *h*. Độ chính xác dự đoán của mô hình *h* được tính theo công thức sau:

$$accuracy(y, \hat{y}) = \frac{1}{m}\sum_{i=0}^{m-1} \mathbf{1}(y_i = \hat{y}_i)$$

Viết lại công thức:

$$accuracy(y, \hat{y}) = \frac{1}{m}\sum \mathbf{1}(y_i = \hat{y}_i)$$

2. Lập trình tính Accuracy

Công thức Toán	Lập trình Python	Sử dụng sklearn

Độ đo Accuracy:
$$accuracy(y, \hat{y}) = \frac{1}{\cancel{m}} \sum_{i=0}^{m-1} \mathbf{1}(y_i = \hat{y}_i)$$
Viết lại công thức
$$accuracy(y, \hat{y}) = \frac{1}{m} \sum \mathbf{1}(y_i = \hat{y}_i)$$
	`def acc_score(y, y_hat):`
	`  m = y.shape[0]`
	`  result = (1/m)*np.sum(y == y_hat)`
	`  return  result`
		`from sklearn.metrics import accuracy_score`
		`...`
		`print('\t\tSử dụng sklearn, Acc: ', accuracy_score(y_test.flatten(), y_hat.flatten()))`

Tham khảo chương trình gợi ý trong classroom.

3. Các độ đo phổ biến đánh giá hiệu năng mô hình phân lớp

Ngoài độ đo Accuracy, thư viện sklearn còn cung cấp nhiều độ đo khác dùng để đánh giá hiệu năng của mô hình phân lớp (link). Hãy hoàn thành bảng sau và viết chương trình đánh giá hiệu năng mô hình hồi quy logistic bằng các thang đo mới. Hãy xây dựng bảng như trên, với yêu cầu:

- Liệt kê tất cả các độ đo phổ biến của mô hình hồi quy theo tài liệu tham khảo;
- Cột 1: Viết công thức Toán gốc và công thức Toán chuyển đổi theo cú pháp vector/ma trận;
- Cột 2: Tự lập trình Python theo công thức Toán đã chuyển đổi;
- Cột 3: Sử dụng thư viện sklearn.metrics

Công thức Toán	**Mã lệnh Python**	**Sử dụng sklearn.metrics**

Độ đo Top k accuracy:
$$top\ k\ accuracy(y, \hat{f}) = \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=1}^{k} \mathbf{1}(\hat{f}_{i,j} = y_i)$$
	`def top_k_accuracy_score2(y_test, y_score, k):`
	`  sorted_index = np.argsort(y_score, axis=1)[:, ::-1]`
	`  top_k = sorted_index[:, :k].T`
	`  result = np.sum(y_test == top_k)`
	`  return result`
		`from sklearn.metrics import top_k_accuracy_score`
		`...`
		`print(top_k_accuracy_score(y_test, y_score, k=2,`
		`  normalize=False))`

Confusion matrix:
Confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một phân lớp và được dự đoán rơi vào một phân lớp (Chúng ta có thể hình dung nó như một cross table vậy)

```
def confusion_matrix(y_test, y_pred):
  y_test = np.array(y_test)
  y_pred = np.array(y_pred)
  class_num = np.unique(y_test).shape[0]
  result = np.zeros((class_num, class_num))
  n = y_test.shape[0]
  for i in range(n):
    result[y_test[i], y_pred[i]] += 1
  return result
```

```
from sklearn.metrics import confusion_matrix
…
print(confusion_matrix(y_test, y_pred))
```

Độ đo balanced accuracy

$$balanced\ accuracy = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

$\frac{TP}{TP+FN}$: tỉ lệ số điểm true positive trong số những điểm thật sự là positive

$\frac{TN}{TN+FP}$: tỉ lệ số điểm true negative trong số những điểm thật sự là negative

Hai giá trị trên cao tức là tỉ lệ bỏ xót các điểm đúng thấp

```
def true_negative_rate(y_test, y_pred, positive=None):
  cm = confusion matrix2(y test, y pred)
  sum cross = np.sum(cm.diagonal())
  if positive:
    tp = cm[positive,positive]
    tn = sum cross - tp
    fp = np.sum(cm[:,positive]) - tp
    return tn/(tn+fp)
  else:
    class num = np.unique(y test).shape[0]
    result = []
    for i in range(class num):
      tp = cm[i,i]
      tn = sum cross - tp
      fp = np.sum(cm[:,i]) - tp
      result.append(tn/(tn+fp))
    return np.array(result)
def balanced accuracy score2(y test, y pred, positive=No
ne):
  temp1 = recall_score2(y_test, y_pred, positive)
  temp2 = true_negative_rate(y_test, y_pred, positive)
  return 1/2 * (temp1 + temp2)
```

```
from sklearn.metrics import balanced_accuracy_score
print(balanced accuracy score(y test, y pred))
```

Độ đo Precision

$$Precision = \frac{TP}{TP + FP}$$

Precision được định nghĩa là tỉ lệ số điểm true positive trong số những điểm được phân loại là positive (TP + FP). Precision cao đồng nghĩa với độ chính xác của các điểm tìm được là cao.

```
def precision_score2(y_test, y_pred, positive = None):
  cm = confusion_matrix2(y_test, y_pred)
  if positive:
    return cm[positive,positive]/np.sum(cm[:, positive])
  else:
    class_num = np.unique(y_test).shape[0]
    result = []
    for i in range(class_num):
      result.append(cm[i,i]/np.sum(cm[:, i]))
    return np.array(result)
```

```
from sklearn.metrics import precision_score
…
print(precision_score(y_test, y_pred))
```

Độ đo Recall

$$Recall = \frac{TP}{TP + FN}$$

Recall được định nghĩa là tỉ lệ số điểm true positive trong số những điểm thực sự là positive (TP + FN). Recall cao đồng nghĩa với việc tỉ lệ bỏ xót các điểm thực sự positive là thấp

```
def recall_score2(y_test, y_pred, positive=None):
  cm = confusion matrix(y test, y pred)
  if positive:
    return cm[positive,positive]/np.sum(cm[positive])
  else:
    class num = np.unique(y test).shape[0]
    result = []
    for i in range(class num):
      result.append(cm[i,i]/np.sum(cm[i]))
    return np.array(result)
```

```
from sklearn.metrics import recall_score
…
print(recall_score(y_test, y_pred))
```

Độ đo F1 (F-measure)

$$F1 = 2\frac{Precision \times Recall}{Precision + Recall}$$

```
def f1 score2(y test, y pred, positive=None):
  precision = precision_score2(y test, y pred,
                               positive)
  recall = recall score2(y test, y pred, positive)
  result = 2 * (precision * recall) \
               / (precision + recall)
  return result
```

```
from sklearn.metrics import f1_score
…
print(f1_score(y_test, y_pred))
```

Độ đo Haming loss:

$$L_{Hamming}(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1} \mathbf{1}(\hat{y}_i \neq y_i)$$

```
def hamming loss2(y test, y pred):
  n = y test.shape[0]
  return np.sum(y test != y pred)/n
```

```
from sklearn.metrics import hamming loss
…
print(hamming loss(y test, y pred))
```

Viết lại công thức: $L_{Hamming}(y, \hat{y}) = \frac{1}{n}\sum \mathbf{1}(\hat{y}_i \neq y_i)$		