Logistic Regression

Hongchang Gao

Spring 2024

Linear Regression vs Logistic Regression

Given n samples: $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$

- Linear regression
  - Y is continuous
  - Model

$$f(x_i) = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \cdots + w_d x_{i,d}$$

  - Loss function

$$\min_w \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

- Logistic regression
  - Y is discrete
  - Model

$$f(\mathbf{x}_i) = \frac{1}{1+\exp(-\mathbf{w}^T \mathbf{x}_i)}$$

  - Loss function

$$\min_{\mathbf{w}} - \sum_{i=1}^{n} \{y_i \log p_i + (1 - y_i) \log(1 - p_i)\}$$

Evaluation for Binary Classification

- When the data is balanced, use the classification accuracy
  - Balanced data: #positive samples is almost the same with #negative samples

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Ground-truth	Prediction	Correct?
1	0	N
1	1	Y
0	0	Y
0	0	Y
1	1	Y
1	1	Y
0	1	N

Accuracy=5/7

Evaluation for Binary Classification

- When the data is imbalanced, does accuracy still work?
  - Imbalanced data: #positive samples is very different from #negative samples

Ground-truth	Prediction	Correct?
1	0	N
0	0	Y
0	0	Y
0	0	Y
0	0	Y
0	0	Y
0	0	Y

Accuracy=6/7

1. The classifier gives the same prediction for all samples,
But it still has a large accuracy!

2. Accuracy is not a good metric for imbalanced data!

Evaluation for Binary Classification

- Example
  - COVID-19 diagnosis: a good classifier should find all positive cases.
  - Fraud detection: a good classifier should find all fraud transactions.

Ground-truth	Prediction	Correct?
1	0	N
0	0	Y
0	0	Y
0	0	Y
0	0	Y
0	0	Y
0	0	Y

1. This classifier fails to find positive cases or fraud transactions
2. But it still has a large accuracy

Evaluation for Binary Classification

- True Positive (TP)
  - Prediction is positive, ground-truth is positive
- False Positive (FP)
  - Prediction is positive, ground-truth is negative
- False Negative (FN)
  - Prediction is negative, ground-truth is positive
- True Negative (TN)
  - Prediction is negative, ground-truth is negative

	Positive (ground-truth)	**Negative (ground-truth)**
Positive (prediction)	True Positive (TP)	False Positive (FP)
Negative (prediction)	False Negative (FN)	True Negative (TN)

Evaluation for Binary Classification

- Example: COVID-19 diagnosis

Ground-truth	Prediction
1	1
0	0
1	0
0	1
0	0
0	1
0	0
0	0

	Positive (ground-truth)	Negative (ground-truth)
Positive (prediction)	1 (TP)	2 (FP)
Negative (prediction)	1 (FN)	4 (TN)

Evaluation for Binary Classification

- When the data is imbalanced, use recall or precision
  - Recall: the proportion of actual positives is classified correctly

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

  - Precision: the proportion of positive predictions is actually correct

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

	Positive (ground-truth)	Negative (ground-truth)
Positive (prediction)	True Positive (TP)	False Positive (FP)
Negative (prediction)	False Negative (FN)	True Negative (TN)

Evaluation for Binary Classification

- Example

Ground-truth	Prediction
1	1
0	0
1	0
0	1
0	0
0	1
0	0
0	0

	Positive (ground-truth)	Negative (ground-truth)
Positive (prediction)	1 (TP)	2 (FP)
Negative (prediction)	1 (FN)	4 (TN)

Recall=1/2

Precision=1/3

Accuracy=5/8

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

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

Evaluation for Binary Classification

- Example

	Positive (ground-truth)	Negative (ground-truth)
Positive (predict)	1	1
Negative (predict)	8	90

Accuracy = (90+1)/100=0.91 But for the positive class, only one sample is classified correctly

Recall=1/(1+8)=0.11 It only correctly recognizes 11% of all positive samples

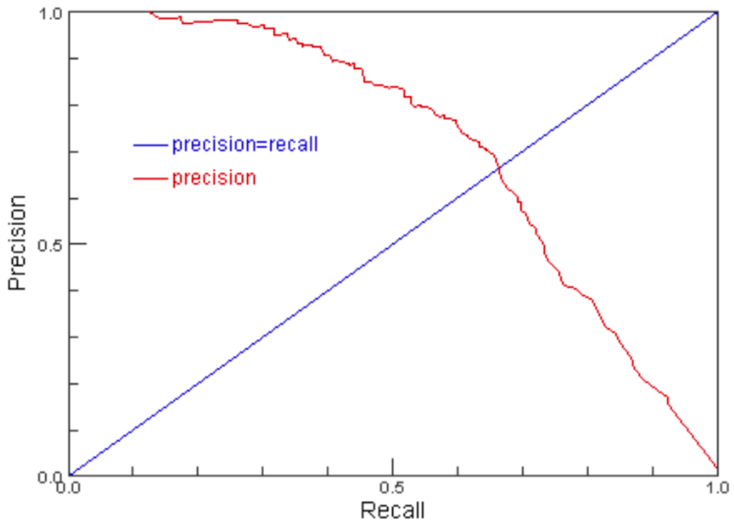
Precision=1/(1+1)=0.5 When the classifier gives a positive prediction, it is correct 50% of the time

Evaluation for Binary Classification

- When to care about recall?
  - It is important to find all positive samples
  - e.g. COVID-19 diagnosis, fraud detection
  - A larger recall means more ground-truth positive samples were founded by the classifier
- When to care about precision?
  - Users are sensitive to the prediction error
  - e.g. google search
  - A larger precision means the positive prediction has a large probability to be correct.

Evaluation for Binary Classification

- Relationship between recall and precision



- F1 score: harmonic mean of precision and recall
  - conveys the balance between the precision and the recall

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

Summary

- Build model
  - Linear classifier to separate positive and negative samples

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \{\log(1 + \exp(\mathbf{w}^T \mathbf{x}_i)) - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda \|\mathbf{w}\|_2^2$$

- Optimize the model

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \sum_{i=1}^{n} \mathbf{x}_i (y_i - \frac{\mathbf{w}_k^T \mathbf{x}_i}{1+\exp(\mathbf{w}_k^T \mathbf{x}_i)}) - 2\eta\lambda \mathbf{w}_k$$

- Evaluate the model
  - Balanced: Accuracy
  - Imbalanced: Recall, Precision, F1 score

Practical Steps

- 1. Preprocess data

```
# preprocess the dataset
X, y = datasets.load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.1,
                                                    random_state=0)

# normalize the dataset
normalizer = StandardScaler()
X_train = normalizer.fit_transform(X_train)
X_test = normalizer.transform(X_test)
```

Practical Steps

- 2. Train the model

```
# train logistic regression
clf = LogisticRegression(penalty='l2', C=1.0)
clf.fit(X_train, y_train)

y_train_pred = clf.predict(X_train)
acc = accuracy_score(y_train, y_train_pred)
print("Training accuracy: {:.4f}".format(acc))
```

```
Training accuracy: 0.9259
```

Practical Steps

- 3. Evaluate the model

```
# evaluate logistic regression
y_test_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_test_pred)
print("Testing accuracy: {:.4f}".format(acc))
```

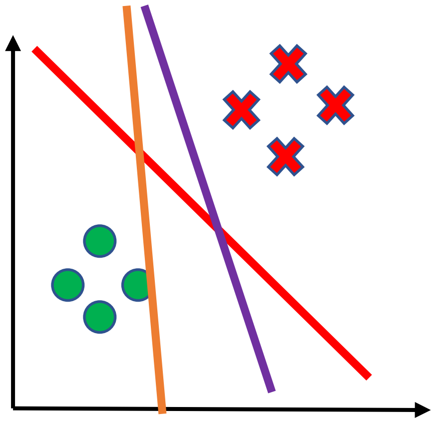
```
Testing accuracy: 0.8000
```

Model Selection

- Model selection
  - Different hyperparameters lead to different models/performance

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \{\log(1 + \exp(\mathbf{w}^T \mathbf{x}_i)) - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda \|\mathbf{w}\|_2^2$$

  - How to find a model with the best performance?

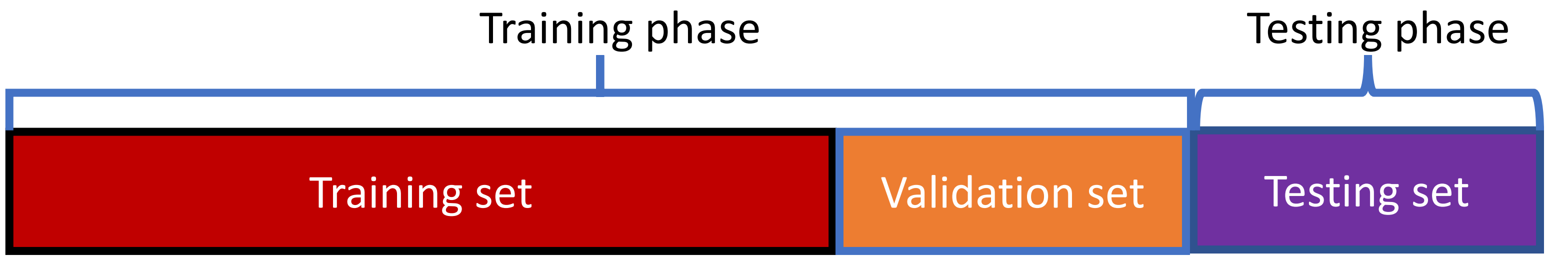


```python
regularization_coefficient = [0.1, 0.5, 1.0, 5.0, 10.0]

for reg in regularization_coefficient:
    clf = LogisticRegression(penalty='l2', C=reg)

    clf.fit(X_train, y_train)

    y_test_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_test_pred)

    print("reg_coeff: {}, acc: {:.3f}".format(1.0/reg, acc))
```

```
reg_coeff: 10.0, acc: 0.533
reg_coeff: 2.0, acc: 0.533
reg_coeff: 1.0, acc: 0.533
reg_coeff: 0.2, acc: 0.800
reg_coeff: 0.1, acc: 0.800
```

Model Selection: Threefold Split

- Threefold split:
  - Training set
    - Used for training the model, during the training phase
  - Validation set
    - Used for hyperparameter selection, during the training phase
  - Testing set
    - Used for evaluating the model, after obtaining the model



Do NOT use the testing set to select the hyperparameter

Model Selection: Threefold Split

- How to select the model (hyperparameters)?
  - For each hyperparameter
    - Train the model using the training set
    - Evaluate the model using the validation set
  - Use the performance on the validation set to select the best hyperparameters
- After obtaining the best hyperparameter:
  - Retrain the model with both the training and validation sets
  - Report the performance on the testing set

Model Selection: Threefold Split

```python
X, y = datasets.load_iris(return_X_y=True)
print(X.shape)

X_train_val, X_test, y_train_val, y_test = train_test_split(X, y,
                                                            test_size=0.1,
                                                            random_state=0)
print(X_train_val.shape, X_test.shape)

X_train, X_valid, y_train, y_valid = train_test_split(X_train_val,
                                                      y_train_val,
                                                      test_size=0.1,
                                                      random_state=0)
print(X_train.shape, X_valid.shape)
```

```
(150, 4)
(135, 4) (15, 4)
(121, 4) (14, 4)
```

Model Selection: Threefold Split

```
regularization_coefficient = [0.1, 0.5, 1.0, 5.0, 10.0]

best_acc = 0.0
best_reg = 0.0

for reg in regularization_coefficient:
    clf = LogisticRegression(penalty='l2', C=reg)

    clf.fit(X_train, y_train)

    y_valid_pred = clf.predict(X_valid)
    acc = accuracy_score(y_valid, y_valid_pred)

    if acc > best_acc:
        best_acc = acc
        best_reg = reg

    print("reg_coeff: {}, acc: {:.3f}".format(1.0/reg, acc))
```

Using the training set

Using the validation set

Model selection based on the validation set

```
reg_coeff: 10.0, acc: 0.571
reg_coeff: 2.0, acc: 0.643
reg_coeff: 1.0, acc: 0.643
reg_coeff: 0.2, acc: 0.714
reg_coeff: 0.1, acc: 0.714
acc: 0.800
```

```
clf = LogisticRegression(penalty='l2', C=best_reg)
clf.fit(X_train_val, y_train_val)
y_test_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_test_pred)

print("acc: {:.3f}".format(acc))
```

Retrain the model with the best hyperparameter,
Evaluate the model on the testing set

Model Selection: Threefold Split

- Pros:
  - Fast and simple
- Cons:
  - Large variance
    - Unreliable estimation of future performance
  - Bad use of data
    - Waste data

Model Selection: Cross-validation

- Training data:
  - Randomly partition it into K folds
  - K-1 folds for training set
  - 1 fold for validation set
- How to select the model?
  - For each hyperparameter
    - Train the model for K times
    - Evaluate the model for K times
  - Use the mean of K evaluation to select model

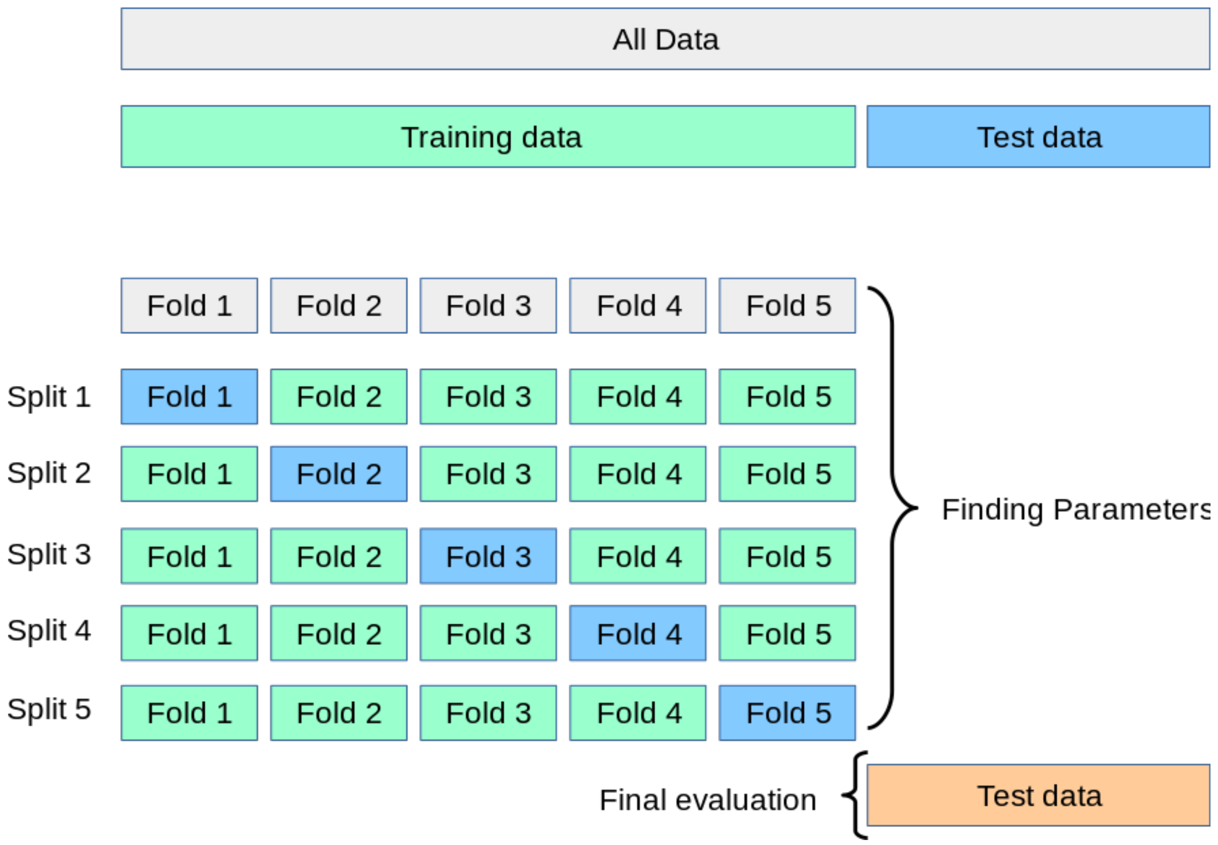


Fig ref: https://scikit-learn.org/stable/modules/cross_validation.html

```python
for reg in regularization_coefficient:

    sum_acc = 0.0
    for fold in range(5):

        index_of_folds_temp = index_of_folds.copy()

        valid_index = index_of_folds_temp[fold,:].reshape(-1)
        train_index = np.delete(index_of_folds_temp, fold, 0).reshape(-1)

        X_train = X_train_val[train_index]
        y_train = y_train_val[train_index]

        X_valid = X_train_val[valid_index]
        y_valid = y_train_val[valid_index]

        clf = LogisticRegression(penalty='l2', C=reg)
        clf.fit(X_train, y_train)

        y_valid_pred = clf.predict(X_valid)
        acc = accuracy_score(y_valid, y_valid_pred)

        sum_acc += acc

    cur_acc = sum_acc / 5.0

    print("reg_coeff: {}, acc: {:.3f}".format(1.0/reg, cur_acc))

    if cur_acc > best_acc:
        best_acc = cur_acc
        best_reg = reg
```

n

```
reg_coeff: 10.0, acc: 0.770
reg_coeff: 2.0, acc: 0.911
reg_coeff: 1.0, acc: 0.941
reg_coeff: 0.2, acc: 0.948
reg_coeff: 0.1, acc: 0.948
```

Model Selection: Cross-validation

```
clf = LogisticRegression(penalty='l2', C=best_reg)
clf.fit(X_train_val, y_train_val)

y_test_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_test_pred)

print("acc: {:.3f}".format(acc))
```

```
acc: 0.867
```

Model Selection: Cross-validation

- Pros:
  - More data
  - More stable
- Cons:
  - Slower