17.3.2

Logistic Regression to Predict Diabetes

Now that you've gotten your feet wet with logistic regression, Jill believes that it's time to implement a model with a real dataset. In the next step, you will follow the familiar pattern as you instantiate a model, train it, create predictions, then validate the model.

Let's solve another classification problem with logistic regression. This time, we'll use a dataset on diabetes among Pima Indian women.

First, download the files you'll need.

Download 17-3-2-logistic_regression_dm.zip (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_17/17-3-2-logistic_regression_dm.zip)

Open `diabetes.ipynb`. We can see from the preview of the DataFrame that multiple variables (also called features), such as the number of previous pregnancies, blood glucose level, and age, can be used to predict the outcome: whether a person has diabetes (1) or does not have diabetes (0):

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

A common task in machine learning is data preparation. In previous examples, we assigned the label X to input variables, and used them to predict y, or the output. With this diabetes dataset, we need to categorize features from the target. We can do so by separating the `Outcome` column from the other columns.

NOTE

The terms features and variables are synonymous. Target and output are synonymous.

```
y = df["Outcome"]
X = df.drop(columns="Outcome")
```

- The `Outcome` column is defined as y, or the target.
- X, or features, is created by dropping the `Outcome` column from the DataFrame.

Later in this module, we'll be splitting the training and testing data, creating a logistic regression model, fitting the training data, and making a prediction. What steps can you do on your own in the following Skill Drills?

SKILL DRILL

Use the train_test_split module to split X and y into training and testing sets. You should end up with four sets in total: `X_train`, `X_test`, `y_train`, `y_test`.

SKILL DRILL

Import the Scikit-learn module for logistic learning, and instantiate a model. Use the following arguments for the model:

- `solver='lbfgs'`
- `max_iter=200` (this sets an upper limit on the number of iterations used by the solver)
- `random_state=1`

SKILL DRILL

Train the model with the training dataset. Then use `X_test` to make predictions for y values.

Let's retrace the steps taken so far. We first split the dataset into training and testing sets:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
   y, random_state=1, stratify=y)
```

Examining the shape of the training set with `X_train.shape` returned (576,8), meaning that there are 576 samples (rows) and eight features (columns).

The next step was to create a logistic regression model with the specified arguments for `solver`, `max_iter`, and `random_state`:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(solver='lbfgs',
   max_iter=200,
   random_state=1)
```

Next, we trained the model with the training data:

```
classifier.fit(X_train, y_train)
```

SKILL DRILL Use the testing set to create predictions of the outcome.

To create predictions for y-values, we used the `X_test` set:

```
y_pred = classifier.predict(X_test)
```

When the first 20 rows of the predicted y-values (`y_pred`) are compared with the actual y-values (`y_test`), we see that most of the predictions are correct, but that there are also some missed predictions, such as rows 14 and 15:

	Prediction	Actual
0	0	0
1	1	1
2	0	0
3	1	1
4	0	0
5	0	0
6	1	1
7	1	0
8	1	1
9	0	0
10	1	1
11	0	1
12	0	0
13	1	1
14	0	1
15	0	1
16	0	0
17	0	0
18	1	1
19	0	0

SKILL DRILL

Use the `accuracy_score()` method module to assess the performance of the model. What will you use as the arguments for this method? What is the accuracy score? What does the accuracy score mean?

The final step is to answer an important question: how well does our logistic regression model predict? We do so with `sklearn.metrics.accuracy_score`:

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

This method compares the actual outcome (y) values from the test set against the model's predicted values. In other words, `y_test` are the outcomes (whether or not a woman has diabetes) from the original dataset that were set aside for testing. The model's predictions, `y_pred`, were compared with these actual values (`y_test`). The accuracy score is simply the percentage of predictions that are correct. In this case, the model's accuracy score was 0.776, meaning that the model was correct 77.6% of the time.