Logistic Regression

Implement logistic regression for classification tasks.

Chapter Goals:

- Learn about logistic regression for linearly separable datasets

A. Classification

Thus far we've learned about several linear regression models and implemented them with scikit-learn. The logistic regression model, despite its name, is actually a linear model for *classification*. It is called logistic regression because it performs regression on logits, which then allows us to classify the data based on model probability predictions.

For a more detailed explanation of logistic regression, check out the **Intro to Deep Learning** section of this course, which implements logistic regression via a single layer perceptron model in TensorFlow.

We implement logistic regression with the `LogisticRegression` object (part of the `linear_model` module). The default setting for `LogisticRegression` is *binary classification*, i.e. classifying data observations that are labeled with either a 0 or 1.

```
# predefined dataset
print('Data shape: {}\n'.format(data.shape))
# Binary labels
print('Labels:\n{}\n'.format(repr(labels)))

from sklearn import linear_model
reg = linear_model.LogisticRegression()
reg.fit(data, labels)

new_data = np.array([
  [  0.3,  0.5, -1.2,  1.4],
  [ -1.3,  1.8, -0.6, -8.2]])
print('Prediction classes: {}\n'.format(
  repr(reg.predict(new_data))))
```

The code above created a logistic regression model from a labeled dataset. The model predicts 1 and 0, respectively, as the labels for the observations in `new_data`.

For *multiclass classification*, i.e. when there are more than two labels, we initialize the `LogisticRegression` object with the `multi_class` keyword argument. The default value is `'ovr'`, which signifies a One-Vs-Rest strategy. In multiclass classification, we want to use the `'multinomial'` strategy.

The code below demonstrates multiclass classification. Note that to use the `'multinomial'` strategy, we need to choose a proper solver (see below for details on solvers). In this case, we choose `'lbfgs'`.

```
# predefined dataset
print('Data shape: {}\n'.format(data.shape))
# Multiclass labels
print('Labels:\n{}\n'.format(repr(labels)))

from sklearn import linear_model
reg = linear_model.LogisticRegression(
  solver='lbfgs',
  multi_class='multinomial')
reg.fit(data, labels)

new_data = np.array([
  [ 1.8, -0.5, 6.2, 1.4],
  [ 3.3,  0.8, 0.1, 2.5]])
print('Prediction classes: {}\n'.format(
  repr(reg.predict(new_data))))
```

B. Solvers

The `LogisticRegression` object uses a *solver* to obtain the optimal weight settings based on the input data and labels. The five solvers and their various properties are shown in the table below (which comes from the scikit-learn official website):

	Solvers				
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Table of the five solvers and their properties.

By default, the logistic regression is regularized through the L2 norm of weights. We can manually specify whether to use the L1 or L2 norm with the `penalty` keyword argument, by setting it as either `'l1'` or `'l2'`.

We can choose a particular solver using the `solver` keyword argument. The default solver is currently `'liblinear'` (although it will change to `'lbfgs'` in future version). For the `'newton-cg'`, `'sag'`, and `'lbfgs'` solvers, we can also set the maximum number of iterations the solver takes until the model's weights converge using the `max_iter` keyword argument. Since the default `max_iter` value is 100, we may want to let the solver run for a higher number of iterations in certain applications.

The code below demonstrates usage of the `solver` and `max_iter` keyword arguments.

```
from sklearn import linear_model
reg = linear_model.LogisticRegression(
  solver='lbfgs', max_iter=1000)
```

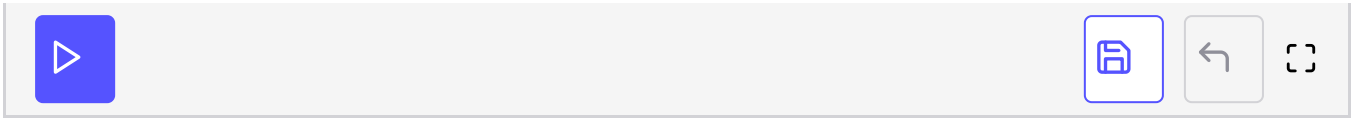
C. Cross-validated model

Like the ridge and LASSO regression models, the logistic regression model comes with a cross-validated version in scikit-learn. The cross-validated logistic regression object, `LogisticRegressionCV`, is initialized and used in the same way as the regular `LogisticRegression` object.

The code below demonstrates usage of the `LogisticRegressionCV` object.

```
from sklearn import linear_model
```

```
reg = linear_model.LogisticRegressionCV(
  solver='multinomial', max_iter=1000)
```



Time to Code!

The coding exercise in this chapter uses the `LogisticRegression` object of the `linear_model` module (imported in backend) for multiclass classification.

The function `multiclass_lr` will fit a logistic regression model to a dataset with multiclass labels.

Set `reg` equal to `linear_model.LogisticRegression` with the `solver` and `max_iter` keyword arguments set as `'lbfgs'` and `max_iter`, respectively. Also have the `multi_class` keyword argument set to `'multinomial'`.

Call `reg.fit` with `data` and `labels` as the two input arguments. Then return `reg`.

```
def multiclass_lr(data, labels, max_iter):
  # CODE HERE
  pass
```