- R^2: The amount of variance or variability in the dataset explained by the model. The score of 1 means that the model perfectly captures the variability in the dataset.

Classification evaluation metrics

- Accuracy: Is the ratio of correct predictions to the total number of predictions. The bigger the accuracy, the better the model.
- Logarithmic loss (a.k.a logistic loss or cross-entropy loss): Is the probability that an observation is correctly assigned to a class label. By minimizing the log-loss, conversely, the accuracy is maximized. So with this metric, values closer to zero are good.
- Area under the ROC curve (AUC-ROC): Used in the binary classification case. Implementation is not provided, but very similar in style to the others.
- Confusion matrix: More intuitive in the binary classification case. Implementation is not provided, but very similar in style to the others.
- Classification report: It returns a text report of the main classification metrics.

Regression Evaluation Metrics

The following code is an example of regression evaluation metrics implemented stand-alone.

```
# import packages
from sklearn.linear_model import LinearRegression
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

# load dataset
data = datasets.load_boston()
```

```
# separate features and target
X = data.data
y = data.target

# split in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True)

# create the model
# setting normalize to true normalizes the dataset before fitting the model
linear_reg = LinearRegression(normalize = True)

# fit the model on the training set
linear_reg.fit(X_train, y_train)
'Output': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
normalize=True)

# make predictions on the test set
predictions = linear_reg.predict(X_test)
# evaluate the model performance using mean square error metric
print("Mean squared error: %.2f" % mean_squared_error(y_test, predictions))
'Output':
Mean squared error: 14.46

# evaluate the model performance using mean absolute error metric
print("Mean absolute error: %.2f" % mean_absolute_error(y_test,
predictions))
'Output':
Mean absolute error: 3.63

# evaluate the model performance using r-squared error metric
print("R-squared score: %.2f" % r2_score(y_test, predictions))
'Output':
R-squared score: 0.69
```

The following code is an example of regression evaluation metrics implemented with cross-validation. The MSE and MAE metrics for cross-validation are implemented with the sign inverted. The simple way to interpret this is to have it in mind that the closer the values are to zero, the better the model.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# load dataset
data = datasets.load_boston()

# separate features and target
X = data.data
y = data.target

# initialize KFold - with shuffle = True, shuffle the data before splitting
kfold = KFold(n_splits=3, shuffle=True)

# create the model
linear_reg = LinearRegression(normalize = True)

# fit the model using cross validation - score with Mean square error (MSE)
mse_cv_result = cross_val_score(linear_reg, X, y, cv=kfold, scoring="neg_
mean_squared_error")
# print mse cross validation output
print("Negative Mean squared error: %.3f%% (%.3f%%)" % (mse_cv_result.
mean(), mse_cv_result.std()))
'Output':
Negtive Mean squared error: -24.275% (4.093%)

# fit the model using cross validation - score with Mean absolute error (MAE)
mae_cv_result = cross_val_score(linear_reg, X, y, cv=kfold, scoring="neg_
mean_absolute_error")
# print mse cross validation output
print("Negtive Mean absolute error: %.3f%% (%.3f%%)" % (mae_cv_result.
mean(), mae_cv_result.std()))
'Output':
Negtive Mean absolute error: -3.442% (4.093%)

# fit the model using cross validation - score with R-squared
r2_cv_result = cross_val_score(linear_reg, X, y, cv=kfold, scoring="r2")
# print mse cross validation output
```

```
print("R-squared score: %.3f%% (%.3f%%)" % (r2_cv_result.mean(), r2_cv_
result.std()))
'Output':
R-squared score: 0.707% (0.030%)
```

Classification Evaluation Metrics

The following code is an example of classification evaluation metrics implemented stand-alone.

```
# import packages
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss
from sklearn.metrics import classification_report

# load dataset
data = datasets.load_iris()

# separate features and target
X = data.data
y = data.target

# split in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True)

# create the model
logistic_reg = LogisticRegression()

# fit the model on the training set
logistic_reg.fit(X_train, y_train)
'Output':
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```