```
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)
```

```
# make predictions on the test set
predictions = logistic_reg.predict(X_test)

# evaluate the model performance using accuracy
print("Accuracy score: %.2f" % accuracy_score(y_test, predictions))
'Output':
Accuracy score: 0.89

# evaluate the model performance using log loss

### output the probabilities of assigning an observation to a class
predictions_probabilities = logistic_reg.predict_proba(X_test)
print("Log-Loss likelihood: %.2f" % log_loss(y_test, predictions_
probabilities))
'Output':
Log-Loss likelihood: 0.39

# evaluate the model performance using classification report
print("Classification report: \n", classification_report(y_test,
predictions, target_names=data.target_names))
'Output':
Classification report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        12
  versicolor       0.85      0.85      0.85        13
   virginica       0.85      0.85      0.85        13

avg / total        0.89      0.89      0.89        38
```

Let's see an example of classification evaluation metrics implemented with cross-validation. Evaluation metrics for log-loss using cross-validation is 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 LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

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

# 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
logistic_reg = LogisticRegression()

# fit the model using cross validation - score with accuracy
accuracy_cv_result = cross_val_score(logistic_reg, X, y, cv=kfold,
scoring="accuracy")
# print accuracy cross validation output
print("Accuracy: %.3f%% (%.3f%%)" % (accuracy_cv_result.mean(), accuracy_
cv_result.std()))
'Output':
Accuracy: 0.953% (0.025%)

# fit the model using cross validation - score with Log-Loss
logloss_cv_result = cross_val_score(logistic_reg, X, y, cv=kfold,
scoring="neg_log_loss")
# print mse cross validation output
print("Log-Loss likelihood: %.3f%% (%.3f%%)" % (logloss_cv_result.mean(),
logloss_cv_result.std()))
'Output':
Log-Loss likelihood: -0.348% (0.027%)
```

Pipelines: Streamlining Machine Learning Workflows

The concept of pipelines in Scikit-learn is a compelling tool for chaining a bunch of operations together to form a tidy process flow of data transforms from one state to another. The operations that constitute a pipeline can be any of Scikit-learn's