Implementation:

```
import numpy as np
from sklearn.model_selection import LeavePOut

X = np.array([[1,2],[3,4],[5,6],[7,8]])
y = np.array([1,2,3,4])
lpo = LeavePOut()

for train_index, test_index in lpo.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Stratified k-Fold → Is a variation of the standard k-Fold CV technique which is designed to be effective in such cases of target imbalance.
All mentioned about k-Fold CV is true for Stratified k-Fold.

Algorithm

1. Pick a number of folds - k
2. Split the dataset into k folds. Each fold must contain approximately the same percentage of samples of each target class as the complete set.
3. Choose k-1 folds which will be the training set. The remaining fold will be the test set.
4. Train the model on the training set. The remaining fold will be the test set.
5. Validate on the test set.
6. Save the result of the validation.
7. Repeat steps 3-6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.

Implementation:

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

X = np.array([[1,2],[3,4],[1,2],[3,4]])
y = np.array([0,0,1,1])
skf = StratifiedKFold(n_splits=2)

for train_index, test_index in skf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Repeated k-Fold cross-validation → Is probably the most robust of all CV technique. It is a variation of k-Fold but in the case of Repeated k-Folds k is not the number of folds. It is the number of times we train the model. Repeated k-Fold has clear advantages over standard k-Fold CV. Firstly, the proportion of train/test split is not dependent on the number of iterations. Secondly, we can even set unique proportions for every iteration. Thirdly, random selection of samples from the dataset makes Repeated k-Fold even more robust to selection bias.
Still, there are some disadvantages. k-Fold CV guarantees that the model will be tested on all samples, whereas Repeated k-Fold is based on randomization which means that some samples may never be selected to be in the test set at all. At the same time, some samples might be selected multiple times.

Algorithm

1. Pick k - a number of times the model will be trained.
2. Pick a number of samples which will be the test set.
3. Split the dataset.
4. Train on the training set. On each iteration of cross-validation, a new model must be trained.
5. Validate on the test set.
6. Save the result of validation.
7. Repeat steps 3-6 k times.
8. To get the final score average the results that you got on step 6.

Implementation:

```
import numpy as np
from sklearn.model_selection import RepeatedKFold

X = np.array([[1,2],[3,4],[1,2],[3,4]])
y = np.array([0,0,1,1])
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=42)

for train_index, test_index in rkf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Nested k-Fold cross-validation → Unlike the other CV techniques, which are designed to evaluate the quality of an algorithm, Neasted k-Fold CV is the most popular way to tune the parameters of an algorithm.
This technique is computionally expensive because throughout steps 1-10 plenty of models should be trained and evaluated.

Algorithm

1. Pick k - a number of folds, for example, 10 - let's assume that we've picked this number.
2. Pick a parameter p. Let's assume that our algorithm is Logistic Regression and p is the parameter p = ['l1', 'l2', 'elasticnet', 'none'].
3. Divide the dataset into 10 (k) folds and reserve one of them for test.
4. Reserve one of the training folds for validation.
5. For each value of p train on the 8 remaining training folds and evaluate on the validation fold. Now you have 4 measurements.
6. Repeat steps 4-5 9 times. Rotate which training fold is the validation fold. You now have 9x4 measurements.
7. Choose p that minimizes the average training error over 9 folds. Use that p to evaluate on the test set.
8. Repeat 10 times from step 2, using each fold in turn as test fold
9. Save the mean and standard deviation of the evaluation measure over the 10 test folds.
10. The algorithm that performed the best was the one with the best average out-of-sample performance across the 10 test folds.

Implementation

There is not built-in method in sklearn.

Complete Cross-Validation → Is the least used CV. The general idea is that we choose a number of k - the length of the training set - and validate on every possible split containing k samples in the training set.

Algorithm

1. Pick a number k - length of the training set.
2. Split the dataset.
3. Train on the training set.
4. Validate on the test set.
5. Save the result of the validation.
6. Repeat steps 2-5 C_n^k times
7. To get the final score average the results that you got on step 5.

DL → tricky because most of the CV techniques require training the model at least a couple of times

Keras → allows to pass one of two parameters for the fit function that performs training.

1. validation_split: percentage of data that should be held out for validation
2. validation_data: a tuple of (X, y) which should be used for validation. This parameter overrides the validation_split parameter which means you can use only one of these parameters at once.

PyTorch and MxNet → same approach. They also suggest splitting the dataset into three parts.

1. Training: a part of the dataset to train
2. Validation: a part of the dataset to validate on while training.
3. Testing: a part of the dataset for final validation of the model.