tuning!

ok ok. ~~what~~ you can't just inverse what you see in test or pre-train rebalancing

- **Tuning**, determined by a hyperparameter search such as a grid search.
- **Heuristic**, specified using a general best practice.

test = 1:100
weighting = 100:1
Inverse is best practice

A best practice for using the class weighting is to use the inverse of the class distribution present in the training dataset. For example, the class distribution of the ~~test~~ dataset is a 1:100 ratio for the minority class to the majority class. The inversion of this ratio could be used with 1 for the majority class and 100 for the minority class; for example:

majority
minority

0, seen 1/100 x

minority: majority, seen it test
~~majority~~ minority = 1, seen 100 x

this should be training author based updating on my feedback

Inverse of training set!

```
...
# define model
weights = {0:1.0, 1:100.0}
model = LogisticRegression(solver='lbfgs', class_weight=weights)
```

Listing 16.10: Example of defining the imbalanced weighting for logistic regression as integers.

We might also define the same ratio using fractions and achieve the same result; for example:

these are quite confusing

```
...
# define model
weights = {0:0.01, 1:1.0}
model = LogisticRegression(solver='lbfgs', class_weight=weights)
```

Listing 16.11: Example of defining the imbalanced weighting for logistic regression as fractions.

We can evaluate the logistic regression algorithm with a class weighting using the same evaluation procedure defined in the previous section. We would expect that the class-weighted version of logistic regression to perform better than the standard version of logistic regression without any class weighting. The complete example is listed below.

dict
0: value
1: value

```
# weighted logistic regression model on an imbalanced classification dataset
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
# define model
weights = {0:0.01, 1:1.0}
model = LogisticRegression(solver='lbfgs', class_weight=weights)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

weights

stochastic algorithm needs to be run several times

actual ret_add
0: 0.99
1: 0.01
therefor, inverse should be
{
0: .01
1: .99
}

Listing 16.12: Example of evaluating a class-weighted logistic regression algorithm on the imbalanced classification dataset.

Running the example prepares the synthetic imbalanced classification dataset, then evaluates the class-weighted version of logistic regression using repeated cross-validation.

Note: Your specific results may vary given the stochastic nature of the learning algorithm. Consider running the example a few times and compare the average performance.

the dict keys ~~values~~ tie to the actual column values.