Example of 'hard voting.'

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
%matplotlib inline
from pandas import DataFrame
# generate 2d classification dataset
X, y = make_moons(n_samples=100, noise=0.1)
log_clf=LogisticRegression()
rnd_clf=RandomForestClassifier()
svm_clf=SVC()
voting_clf=VotingClassifier(estimators=[('lr',log_clf), ('rf',rnd_clf),('svc', svm_clf)]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y, random_state=123)
voting_clf.fit(X_train, y_train)
```

```
VotingClassifier(estimators=[('lr',
                              LogisticRegression(C=1.0, class_weight=None,
                                                 dual=False, fit_intercept=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=100,
                                                 multi_class='auto',
                                                 n_jobs=None, penalty='l2',
                                                 random_state=None,
                                                 solver='lbfgs', tol=0.0001,
                                                 verbose=0, warm_start=False)),
                             ('rf',
                              RandomForestClassifier(bootstrap=True,
                                                     ccp_alpha=0.0,
                                                     class_weight=None,
                                                     cr...
                                                     oob_score=False,
                                                     random_state=None,
                                                     verbose=0,
                                                     warm_start=False)),
                             ('svc',
                              SVC(C=1.0, break_ties=False, cache_size=200,
                                  class_weight=None, coef0=0.0,
                                  decision_function_shape='ovr', degree=3,
                                  gamma='scale', kernel='rbf', max_iter=-1,
                                  probability=False, random_state=None,
                                  shrinking=True, tol=0.001, verbose=False))],
                 flatten_transform=True, n_jobs=None, voting='hard',
                 weights=None)
```

```
from sklearn.metrics import accuracy_score
for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
  clf.fit(X_train, y_train)
  y_pred=clf.predict(X_test)
  print(clf.__class__, round(accuracy_score(y_test, y_pred),10))
```

```
<class 'sklearn.linear_model._logistic.LogisticRegression'> 0.84
<class 'sklearn.ensemble._forest.RandomForestClassifier'> 1.0
<class 'sklearn.svm._classes.SVC'> 1.0
<class 'sklearn.ensemble._voting.VotingClassifier'> 0.96
```

This example illustrates the use of bootstrap and out of bag instances (oob). During training, a predictor will not see any of the oob instances. So, the evaluation can happen without needing a separate cross validation. The oob evaluation will tell us that an accuracy of 92% will be achieved on the test set with bag_clf.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
bag_clf=BaggingClassifier(DecisionTreeClassifier(), n_estimators=500,
                          max_samples= 10, bootstrap=True,
                          n_jobs=-1, oob_score=True)
bag_clf.fit(X_train, y_train)
y_pred=bag_clf.predict(X_test)
# Overall accuracy score for the Bagging Classifier
print('Accuracy Score: \n', round(accuracy_score(y_test, y_pred),3))
# The BaggingClassifier samples m instances using replacement
# Around 63% of all instances are sampled for each of the predictors
# The remainder of the instances not sampled are out of bag instances
print('Out of Bag Evaluation Score:\n', round((bag_clf.oob_score_),3))
```

```
Accuracy Score:
 0.84
Out of Bag Evaluation Score:
 0.867
```

Example of 'soft voting.'

```
print('Predict class probabilities for X test: \n', bag_clf.predict_proba(X_test))
```

```
Predict class probabilities for X test:
 [[0.828 0.172]
 [0.934 0.066]
 [0.044 0.956]
 [0.09  0.91 ]
 [0.834 0.166]
 [0.506 0.494]
 [0.826 0.174]
 [0.438 0.562]
 [0.826 0.174]
 [0.422 0.578]
 [0.836 0.164]
 [0.368 0.632]
 [0.57  0.43 ]
 [0.784 0.216]
 [0.952 0.048]
 [0.834 0.166]
 [0.82  0.18 ]
```

```
# In this example, there is 5.2% chance that the first instance belongs to a
# positive class.
print('Decision function computed with out-of-bag estimate on the training set: \n',bag_
```

```
Decision function computed with out-of-bag estimate on the training set:
 [[0.0523918  0.9476082 ]
 [0.45701357 0.54298643]
 [0.9372093  0.0627907 ]
 [0.5372093  0.4627907 ]
 [0.81118881 0.18881119]
 [0.09862385 0.90137615]
 [0.24       0.76      ]
 [0.94444444 0.05555556]
 [0.36238532 0.63761468]
 [0.80136986 0.19863014]
 [0.45205479 0.54794521]
 [0.09885057 0.90114943]
 [0.95183486 0.04816514]
 [0.08675799 0.91324201]
 [0.10297483 0.89702517]
 [0.03464203 0.96535797]
 [0.57568807 0.42431193]
 [0.01354402 0.98645598]
 [0.94050343 0.05949657]
 [0.02941176 0.97058824]
 [0.75342466 0.24657534]
 [0.84137931 0.15862069]
 [0.39344262 0.60655738]
 [0.29908676 0.70091324]
 [0.81118881 0.18881119]
 [0.63364055 0.36635945]
 [0.10114943 0.89885057]
 [0.89953271 0.10046729]
 [0.01834862 0.98165138]
 [0.01339286 0.98660714]
 [0.62045455 0.37954545]
 [0.48081264 0.51918736]
 [0.80493274 0.19506726]
 [0.93793103 0.06206897]
 [0.10227273 0.89772727]
 [0.72706935 0.27293065]
 [0.60222222 0.39777778]
 [0.8590604  0.1409396 ]
 [0.59598214 0.40401786]
 [0.79587156 0.20412844]
 [0.38875878 0.61124122]
 [0.67126437 0.32873563]
 [0.01357466 0.98642534]
 [0.81105991 0.18894009]
 [0.03811659 0.96188341]
 [0.10273973 0.89726027]
 [0.07482993 0.92517007]
 [0.84246575 0.15753425]
 [0.10633484 0.89366516]
 [0.70731707 0.29268293]
 [0.05681818 0.94318182]
 [0.10697674 0.89302326]
 [0.01385681 0.98614319]
 [0.3972912  0.6027088 ]
 [0.10244989 0.89755011]
 [0.43429844 0.56570156]
```

```
[0.96962617 0.03037383]
[0.01354402 0.98645598]
[0.06944444 0.93055556]
[0.63785047 0.36214953]
[0.96590909 0.03409091]
[0.56853933 0.43146067]
[0.66974596 0.33025404]
[0.01877934 0.98122066]
[0.81165919 0.18834081]
[0.77116705 0.22883295]
[0.34515366 0.65484634]
```