```python
# Appendix C7 - ovr_handler.py

from nltk import compat
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.feature_extraction import DictVectorizer
from sklearn import cross_validation
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.cross_validation import train_test_split

class OvrHandler():
    def __init__(self, featuresets):
        self.mlb = MultiLabelBinarizer()
        self.featuresets = featuresets
        self._vectorizer = DictVectorizer(dtype=float, sparse=True)
        self.X, self.y = self.prepare_scikit_x_and_y(self.featuresets)
        self.classifiers = {
            "MultinomialNB": OneVsRestClassifier(MultinomialNB()),
            "BernoulliNB": OneVsRestClassifier(BernoulliNB()),
        }

    def prepare_scikit_x_and_y(self, labeled_featuresets):
        X, y = list(compat.izip(*labeled_featuresets))
        X = self._vectorizer.fit_transform(X)

        set_of_labels = []
        for label in y:
            set_of_labels.append(set(label))

        y = self.mlb.fit_transform(set_of_labels)

        return X, y

    def train_classifiers(self):
        for name, clf in self.classifiers.items():
            clf.fit(self.X, self.y)

    def train_classifiers(self, X, y):
        for name, clf in self.classifiers.items():
            clf.fit(X, y)

    def cross_validate(self):
        results = {}
        for name, clf in self.classifiers.items():
            scores = cross_validation.cross_val_score(
                clf, self.X, self.y, cv=10
            )
            results[name] = {"cross score": scores.mean(), "cross variance": scores.std() * 2}
        return results

    def calculate_accuracy(self):
        results = {}
        X_train, X_test, y_train, y_test = train_test_split(self.X, self.y, random_state=0)

        for name, clf in self.classifiers.items():

            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_test)
            prob_pos = clf.predict_proba(X_test)[:, 1]
            precision = precision_score(y_test, y_pred, average='weighted')
            recall = recall_score(y_test, y_pred, average='weighted')
            f1 = f1_score(y_test, y_pred, average='weighted')

            results[name] = {"precision": precision, "recall": recall, "f1": f1}
        return results

    # Not used. For future implementation.
    # Feed a document's bag of word to this method to obtain recommended classes
    def predict_for_random(self, doc_with_bag_of_words):
        doc, bag_of_words = doc_with_bag_of_words
        print("Predicting for:", doc.title)
        print("Item is labeled to:")
        print(set(doc.topic_titles()))
        print("====> Predictions:")

        X = self._vectorizer.fit_transform(bag_of_words)

        for name, clf in self.classifiers.items():
            predicted_labels = (clf.predict(X))[0]
            probabilities =  clf.predict_proba(X)[0]
            named_classes = self.mlb.classes_

            print("Using %s:" % name)

            # If no labels are predicted for the item:
            if not 1 in predicted_labels:
                print("No label suggested for item")
                return

            for idx, label in enumerate(predicted_labels):
                confidence = round(float(probabilities[idx] * 100), 2)
                if confidence > 10:
                    print(named_classes[idx] + " - Confidence: ", end="")
                    print(str(confidence) + "%")

```