Using Sci-Kit Learn for Classification

Sci-Kit Learn can be installed from their web page, but I found it easiest to install by installing all of Anaconda, which is a larger group of Python packages. Go to their web page:

http://continuum.io/downloads

A lot of what I used for these examples came from their tutorials and user guide:

http://scikit-learn.org/stable/tutorial/index.html

but I also had some examples from Bei Yu and from questions to StackOverflow.

Classification Example demonstrating built-in text processing (NOT for final projects)

Let's start by using Bei Yu's example to load the twitter training data from Sentiment140 and demonstrating some of the built-in feature building tools.

In order to load data files, we can either go to the directory where the file is and run Python there, or we can use the complete file path. For our first example, we assume that the data file is where we started python.

Use the pandas package to read the csv file:

```
>>> import numpy
>>> import pandas
>>> FIELDNAMES = ('polarity', 'id', 'date', 'query', 'author', 'text')
>>> train = pandas.read_csv("training.1600000.processed.noemoticon.csv",
header=None, names=FIELDNAMES)
```

In this training set, the polarity field has either a 0 for negative or 4 for positive sentiment of the twitter text. We will ignore the neutral (2) labels.

```
>>> y = train['polarity'].values
>>> X = train['text'].values
>>> type(y)
<type 'numpy.ndarray'>
>>> y.shape
(1600000,)
>>> type(X)
<type 'numpy.ndarray'>
>>> X.shape
(1600000,)
```

Select the first 1000 positive and the first 1000 negative examples, putting the text into the X array and the labels into the y array. These are numpy arrays and not Python lists. Then put the positive and negative arrays together using numpy's hstack function.

```
>>> Xpos = X[y==4][:1000]
>>> ypos = y[y==4][:1000]
>>> Xneg = X[y==0][:1000]
>>> yneg = y[y==0][:1000]
>>> Xpn = numpy.hstack((Xpos, Xneg))
>>> ypn = numpy.hstack((ypos,yneg))
```

Among the built-in text processors is one that converts the text to word vectors with counts as the values and one that converts the text to word vectors with TFIDF weights as values. We create 4 different text to word vectors convertors and use the fit_transform function to convert the X (text) data.

```
>>> unigram_bool = CountVectorizer(encoding='latin-1', min_df=5, stop_words=None, binary=True)
>>> unigram_count = CountVectorizer(encoding='latin-1', min_df=5, stop_words=None)
>>> unigram_norm = TfidfVectorizer(encoding='latin-1', min_df=5, use_idf=False, smooth_idf=False, norm=u'l1')
>>> bigram_count = CountVectorizer(encoding='latin-1', min_df=5, ngram_range=(1,2), stop_words=None)

>>> unigram_bool_vec = unigram_bool.fit_transform(Xpn)
>>> unigram_count_vec = unigram_count.fit_transform(Xpn)
>>> unigram_norm_vec = unigram_norm.fit_transform(Xpn)
>>> unigram_tfidf_vec = unigram_tfidf.fit_transform(Xpn)
```

The convertors have vocabulary defined.

```
>>> unigram_bool_vec.shape
(2000, 675)
>>> unigram_count_vec[0].toarray()

>>> vocab = unigram_count.vocabulary_
>>> type(vocab)
<type 'dict'>
>>> len(vocab.keys())
675
>>> vocab.keys()
```

Finally, we use the SVM classifier with the Linear kernel to train a classifier and then we call the cross-validation scorer to see the result. The cross-validation scores are accuracy.

```
>>> from sklearn.svm import LinearSVC
```

```
>>> clf = LinearSVC(C=1, penalty='l1', dual=False).fit(unigram_bool_vec,ypn)
>>> from sklearn import cross_validation
>>> scores = cross_validation.cross_val_score(clf, unigram_bool_vec, ypn, cv=5)
>>> scores
array([ 0.675,  0.66 ,  0.67 ,  0.625,  0.665])
```

Classification Example using Movie Review feature sets file prepared in NLTK (this is the process for your Final Projects)

Now we demonstrate how to convert the same .csv file that we wrote for Weka input. We wrote the values of the word features as True and False and the values of the class labels as 'pos' and 'neg'

```
>>> filepath =
"/Users/njmccrac1/AAAdocs/NLPfall2014/labs/LabExamplesWeek12FinalProject/weka/
movies.csv"
>>> movies_train = pandas.read_csv(filepath)
>>> type(movies_train)
>>> movies_train.shape
(2000, 1955)
```

Now we still need to do some conversion because we read all the feature values as Strings from the .csv file, and this particular classifier, SVM, requires that all our features and class labels be integers. So we replace False by 0, True by 1, pos by 1 and neg by 0. Numpy keeps these as type 'object', so as a last step, we convert that type to an 'int'.

```
>>> movies_array = movies_train.values
>>> type(movies_array)
<type 'numpy.ndarray'>
>>> movies_y = movies_array[:,-1]
>>> movies_y.shape
(2000,)
>>> movies_y[0]
' neg'
>>> movies_X = movies_array[:,:-1]
>>> movies_X.shape
(2000, 1954)
>>> movies_X[0]
array([False, ' True', ' True', ..., ' False', ' False', ' False'], dtype=object)
>>> movies_X[0].shape
(1954,)
>>> movies_X[movies_X==' True'] = 1
>>> movies_X[0]
array([False, 1, 1, ..., ' False', ' False', ' False'], dtype=object)
>>> movies_X[movies_X==' False'] = 0
>>> movies_X[0]
```

```
array([False, 1, 1, ..., 0, 0, 0], dtype=object)
>>> movies_X[movies_X==False] = 0
>>> movies_X[0]
array([0, 1, 1, ..., 0, 0, 0], dtype=object)
>>> movies_y[movies_y==' pos'] = 1
>>> movies_y[movies_y==' neg'] = 0
>>> movies_y
array([0, 1, 0, ..., 1, 1, 1], dtype=object)

>>> mX = movies_X.astype(np.int)
>>> my = movies_y.astype(numpy.int)
```

Finally, we train the classifier and get our cross-validation scores:

```
>>> mclf = LinearSVC(C=1, penalty='l1', dual=False).fit(mX,my)
>>> scores = cross_validation.cross_val_score(mclf, mX, my, cv=5)
>>> scores
array([ 0.8075,  0.8125,  0.7925,  0.7725,  0.8425])
```

You can take the average of these scores as a final accuracy result.

```
>>> numpy.mean(scores)
0.80549999999999999
```