DS 7331 In Class Assignment 3

Ben Goodwin, Andre Mauldin

In [1]:
```
#  Ebnable HTML/CSS
from IPython.core.display import HTML
HTML("<link href='https://fonts.googleapis.com/css?family=Passion+One' rel='stylesheet' type='text/css'><style>div.attn { font-family: 'Helvetica Neue'; font-size: 30px; line-height: 40px; color: #FFFFFF; text-align: center; marg
```

Out[1]:

Enter Team Member Names here (double click to edit):

- Name 1: Ben Goodwin
- Name 2: Andre Mauldin
- Name 3:

In Class Assignment Three

In the following assignment you will be asked to fill in python code and derivations for a number of different problems. Please read all instructions carefully and turn in the rendered notebook (or HTML of the rendered notebook) before the end of class.

Contents

Back to Top

Downloading the Document Data

Please run the following code to read in the "20 newsgroups" dataset from sklearn's data loading module.

In [2]:
```
from sklearn.datasets import fetch_20newsgroups_vectorized
import numpy as np
from __future__ import print_function

# this takes about 30 seconds to compute, read the next section while this downloads
ds = fetch_20newsgroups_vectorized(subset='train')

# this holds the continuous feature data (which is tfidf)
print('features shape:', ds.data.shape) # there are ~11000 instances and ~130k features per instance
print('target shape:', ds.target.shape)
print('range of target:', np.min(ds.target),np.max(ds.target))
print('Data type is', type(ds.data), float(ds.data.nnz)/(ds.data.shape[0]*ds.data.shape[1])*100, '% of the data is non-zero')
```

```
features shape: (11314, 130107)
target shape: (11314,)
range of target: 0 19
Data type is <class 'scipy.sparse.csr.csr_matrix'> 0.1214353154362896 % of the data is non-zero
```

Understanding the Dataset

Look at the description for the 20 newsgroups dataset at http://qwone.com/~jason/20Newsgroups/. You have just downloaded the "vectorized" version of the dataset, which means all the words inside the articles have gone through a transformation that binned them into 130 thousand features related to the words in them.

Question Set 1:

- How many instances are in the dataset?
- What does each instance represent?
- How many classes are in the dataset and what does each class represent?
- Would you expect a classifier trained on this data would generalize to documents written in the past week? Why or why not?
- Is the data represented as a sparse or dense matrix?

- There are 11314 instances.
- Each instance represents an article.
- There are 20 classes representing 20 newsgroups
- No. This data set was last updated Jan 14, 2008. The data is too stale.
- This is a sparse matrix. Only 0.12% of the data is non-zero.

Back to Top

Measures of Distance

In the following block of code, we isolate three instances from the dataset. The instance " `a` " is from the group *computer graphics*, " `b` " is from from the group *recreation autos*, and " `c` " is from group *recreation motorcycle*. **Exercise for part 2**: Calculate the:

- (1) Euclidean distance
- (2) Cosine distance
- (3) Jaccard similarity

between each pair of instances using the imported functions below. Remember that the Jaccard similarity is only for binary valued vectors, so convert vectors to binary using a threshold.

Question for part 2: Which distance seems more appropriate to use for this data? **Why**?

In [30]:
```
from scipy.spatial.distance import cosine
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import jaccard
import numpy as np

# get first instance (comp)
idx = 550
a = ds.data[idx].todense()
a_class = ds.target_names[ds.target[idx]]
print('Instance A is from class', a_class)

# get second instance (autos)
idx = 4000
b = ds.data[idx].todense()
b_class = ds.target_names[ds.target[idx]]
print('Instance B is from class', b_class)

# get third instance (motorcycle)
idx = 7000
c = ds.data[idx].todense()
c_class = ds.target_names[ds.target[idx]]
print('Instance C is from class', c_class)

# Enter distance comparison below for each pair of vectors:
p = 'Placeholder'
print('\n\nEuclidean Distance\n ab:', np.linalg.norm(a - b), 'ac:', np.linalg.norm(a - c), 'bc:', np.linalg.norm(b - c))
print('\n ab:',euclidean(a,b), 'ac:',euclidean(a,c), 'bc:',euclidean(b,c))
print('Cosine Distance\n ab:', cosine(a, b), 'ac:', cosine(a, c), 'bc:', cosine(b, c))
print('Jaccard Dissimilarity (vectors should be boolean values)\n ab:', jaccard(a>0,b>0), 'ac:', jaccard(a>0,c>0), 'bc:', jaccard(b>0,c>0))

print('\n\nThe most appropriate distance is...')
print('It is cosine because this is a sparse data set.  Euclidean does not work well with sparse data.')
```

```
Instance A is from class comp.graphics
Instance B is from class rec.autos
Instance C is from class rec.motorcycles


Euclidean Distance
 ab: 1.0985184671870858 ac: 1.1891405425398236 bc: 0.9177794226661625

 ab: 1.0985184671870858 ac: 1.1891405425398234 bc: 0.9177794226661624
Cosine Distance
 ab: 0.6033714113755322 ac: 0.7070276149559529 bc: 0.4211595343347173
Jaccard Dissimilarity (vectors should be boolean values)
 ab: 0.8821138211382114 ac: 0.8754716981132076 bc: 0.9087947882736156


The most appropriate distance is...
It is cosine because this is a sparse data set.  Euclidean does not work well with sparse data.
```

Start of Live Session Assignment

Back to Top

Using scikit-learn with KNN

Now let's use stratified cross validation with a holdout set to train a KNN model in `scikit-learn`. Use the example below to train a KNN classifier. The documentation for `KNeighborsClassifier` is here: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Exercise for part 3: Use the code below to test what value of `n_neighbors` works best for the given data. *Note: do NOT change the metric to be anything other than `'euclidean'`. Other distance functions are not optimized for the amount of data we are working with.*

Question for part 3: What is the accuracy of the best classifier you can create for this data (by changing only the `n_neighbors` parameter)?

In [7]:
```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import cross_validate
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from IPython.html import widgets

cv = StratifiedShuffleSplit(n_splits=5, test_size = 0.5, train_size=0.5, random_state=3)

# fill in the training and testing data and save as separate variables
for trainidx, testidx in cv.split(ds.data,ds.target):
    # note that these are sparse matrices
    X_train = ds.data[trainidx]
    X_test = ds.data[testidx]
    y_train = ds.target[trainidx]
    y_test = ds.target[testidx]

# fill in your code  here to train and test
# calculate the accuracy and print it for various values of K
kays = [1,2,3,4,5,6,7,8,9,10]
for K in kays:
    clf = KNeighborsClassifier(n_neighbors=K, weights='uniform', metric='euclidean')
    clf.fit(X_train,y_train)
    y_hat = clf.predict(X_test)

    acc = accuracy_score(y_test,y_hat)

    #=====================================

    print('Accuracy of classifier with %d neighbors is: %.2f'%(K,acc))
```

```
Accuracy of classifier with 1 neighbors is: 0.61
Accuracy of classifier with 2 neighbors is: 0.55
Accuracy of classifier with 3 neighbors is: 0.52
Accuracy of classifier with 4 neighbors is: 0.51
Accuracy of classifier with 5 neighbors is: 0.50
Accuracy of classifier with 6 neighbors is: 0.50
Accuracy of classifier with 7 neighbors is: 0.49
Accuracy of classifier with 8 neighbors is: 0.48
Accuracy of classifier with 9 neighbors is: 0.48
Accuracy of classifier with 10 neighbors is: 0.47
```

The best accuracy is .61 with 1 neighbors.

Question for part 3: With sparse data, does the use of a KDTree representation make sense? Why or Why not?

The cost of measuring distance with truly sparse data is small--usually smaller than the cost of creating a tree. Sparse data also will not branch well in a KDTree because there are so many zeros. The number of "zero" comparisons will mean each tree needs to be very deep. Therefore the benefit of the KDTree is drastically reduced.

KNN extensions - Centroids

Now lets look at a very closely related classifier to KNN, called nearest centroid. In this classifier (which is more appropriate for big data scenarios and sparse data), the training step is used to calculate the centroids for each class. These centroids are saved. Unknown attributes, at prediction time, only need to have distances calculated for each saved centroid, drastically decreasing the time required for a prediction.

Exercise for part 4: Use the template code below to create a nearest centroid classifier. Test which metric has the best cross validated performance: Euclidean, Cosine, or Manhattan. In `scikit-learn` you can see the documentation for NearestCentroid here:

- http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestCentroid.html#sklearn.neighbors.NearestCentroid

and for supported distance metrics here:

- http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.distance_metrics.html#sklearn.metrics.pairwise.distance_metrics

In [12]:
```
from sklearn.neighbors.nearest_centroid import NearestCentroid

# the parameters for the nearest centroid metric to test are:
#    l1, l2, and cosine (all are optimized)
params = ['l1','l2','cosine']
X = ds.data
y = ds.target
for i in params:
    clf = NearestCentroid(metric=i)
    clf.fit(X_train,y_train)
    y_hat = clf.predict(X_test)

    acc = accuracy_score(y_test,y_hat)
    print(i, acc)

print('The best distance metric is: ', 'Cosine')
```

```
/Applications/anaconda3/lib/python3.8/site-packages/sklearn/neighbors/_nearest_centroid.py:150: UserWarning: Averaging for metrics other than euclidean and manhattan not supported. The average is set to be the mean.
  warnings.warn("Averaging for metrics other than "
l1 0.3372812444758706
l2 0.4007424429909846
cosine 0.47816864062223796
The best distance metric is:  Cosine
/Applications/anaconda3/lib/python3.8/site-packages/sklearn/neighbors/_nearest_centroid.py:150: UserWarning: Averaging for metrics other than euclidean and manhattan not supported. The average is set to be the mean.
  warnings.warn("Averaging for metrics other than "
/Applications/anaconda3/lib/python3.8/site-packages/sklearn/neighbors/_nearest_centroid.py:150: UserWarning: Averaging for metrics other than euclidean and manhattan not supported. The average is set to be the mean.
  warnings.warn("Averaging for metrics other than "
```

Back to Top

Naive Bayes Classification

Now let's look at the use of the Naive Bayes classifier. The 20 newsgroups dataset has 20 classes and about 130,000 features per instance. Recall that the Naive Bayes classifer calculates a posterior distribution for each possible class. Each posterior distribution is a multiplication of many conditional distributions:

$$\arg\max_j \left(p(class=j) \prod_i p(attribute=i|class=j) \right)$$

where $p(class=j)$ is the prior and $p(attribute=i|class=j)$ is the conditional probability.

Question for part 5: With this many classes and features, how many different conditional probabilities need to be parameterized? How many priors need to be parameterized?

Total conditionals = 130107 * 20 = 2602140. There is one "prior" for each class.

Naive Bayes in Scikit-learn

Scikit has several implementations of the Naive Bayes classifier: `GaussianNB`, `MultinomialNB`, and `BernoulliNB`. Look at the documentation here: http://scikit-learn.org/stable/modules/naive_bayes.html Take a look at each implementation and then answer this question:

Questions for part 6:

- If the instances contain mostly continuous attributes, would it be better to use Gaussian Naive Bayes, Multinomial Naive Bayes, or Bernoulli? And Why?
- What if the data is sparse, does this change your answer? Why or Why not?

Sparse matrices are much harder to find realistic Gaussian models for because they always have a mean near zero.

For sparse data, it is probably better (and faster) to use multinomial naive Bayes.

An argument can also be made for Bernoulli if binarizing the feature data helps to reduce the complexity of the problem.

Naive Bayes Comparison

For the final section of this notebook let's compare the performance of Naive Bayes for document classification. Look at the parameters for `MultinomialNB`, and `BernoulliNB` (especially `alpha` and `binarize`).

Exercise for part 7: Using the example code below, change the parameters for each classifier and see how accurate you can make the classifiers on the test set.

Question for part 7: Why are these implementations so fast to train? What does the `'alpha'` value control in these models (*i.e.*, how does it change the parameterizations)?

In [32]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB

alphaVals = [0.0,0.0001,0.0025,0.0050,0.0075,1.0]
binVals = [0.0,0.01,0.02,0.04,0.08,1.0]
b=0
for a in alphaVals:

    clf_mnb = MultinomialNB(alpha=a)
    clf_bnb = BernoulliNB(alpha=a, binarize=binVals[b])
    b = b + 1
    c = 0 # used for labeling
    for clf in [clf_mnb, clf_bnb]:
        clf.fit(X_train, y_train)
        y_hat = clf.predict(X_test)
        acc = accuracy_score(y_test,y_hat)

        if c == 0:
            label = "Multinomial"
        else:
            label = "Bernoulli"

        print(label,acc)
        c = c + 1


print('\nThese classifiers are so fast because binarization is the process of transforming data features of any entity into vectors of binary numbers to make classifier algorithms more efficient. In a simple example transforming
print('\nThe alpha values control Laplace smoothing. And Laplace smoothing helps with zero probability in a naive bases classifier.')
```

```
/Applications/anaconda3/lib/python3.8/site-packages/sklearn/naive_bayes.py:511: UserWarning: alpha too small will result in numeric errors, setting alpha = 1.0e-10
  warnings.warn('alpha too small will result in numeric errors, '
/Applications/anaconda3/lib/python3.8/site-packages/sklearn/naive_bayes.py:511: UserWarning: alpha too small will result in numeric errors, setting alpha = 1.0e-10
  warnings.warn('alpha too small will result in numeric errors, '
Multinomial 0.8651228566377939
Bernoulli 0.8532791232101821
Multinomial 0.8833303871309881
Bernoulli 0.8690118437334277
Multinomial 0.8891638677744388
Bernoulli 0.8753756407990101
Multinomial 0.8886335513523069
Bernoulli 0.8667138059041894
Multinomial 0.8873961463673326
Bernoulli 0.75923634435213
Multinomial 0.7097401449531554
Bernoulli 0.0530316422131872

These classifiers are so fast because binarization is the process of transforming data features of any entity into vectors of binary numbers to make classifier algorithms more efficient. In a simple example transforming an imag
e's gray-scale from the 0-255 spectrum to a 0-1 spectrum is binarization.

The alpha values control Laplace smoothing. And Laplace smoothing helps with zero probability in a naive bases classifier.
```

That's all! Please **upload your rendered notebook to blackboard** and please include **team member names** in the notebook submission.

In [ ]: