kNN Classification using k-mer frequency representation of text

In this project, we will create a program to transform text into vectors using a slightly different technique than previously learned, and then perform kNN based classification on the resulting vectors. We will be using the badges UCI dataset (https://archive.ics.uci.edu/ml/machine-learning-databases/badges/badges.data (https://archive.ics.uci.edu/ml/machine-learning-databases/badges/badges.data)), which contains names of some conference attendees along with a "+" or "-" class for each name. We will first make the text lowercase and separate the class from the badge name for each object, e.g., "+ Naoki Abe" should be turned into the object "naoki abe" with class "+". We will keep track of the original name ("Naoki Abe") associated with the vector.

Our program will have two input parameters, c and k. Given the input parameter c, for each name, it will construct a vector of c-mer terms (usually called k-mers, but I am calling them c-mers since the input variable k is being used for kNN) of the required c length, by enumerating all subsequences of length c within the name. For example, if c=3, “naoki abe” becomes < “nao”, “aok”, “oki”, “ki “, “i a”, “ ab”, “abe” >. Finally, we will use the same technique we learned for word-based terms to construct sparse term-frequency vectors for each of the objects.

Using the constructed vectors and their associated classes, given the input parameter k, we will construct a program that should perform kNN based classification using cosine similarity and 10-fold cross-validation and report the average classification accuracy among all tests. The class of the test sample should be chosen by majority vote, with ties broken in favor of the class with the highest average similarity. In the rare case that the test sample does not have any neighbors (no features in common with any training samples), we will assign a predicted class label by drawing a random value from a uniform distribution over [0,1) and classifying the test sample as “+” if the value is greater than 0.5 and “-“ otherwise.

In [3]:
```
import numpy as np
import pandas as pd
import scipy.sparse as sp
from numpy.linalg import norm
from collections import Counter, defaultdict
from scipy.sparse import csr_matrix
```

The code below reads the badges dataset into the variable `df` and extracts the string data associated with each person in the `vals` variable. Write code to split the strings in vals into their component names and classes. The `names` and `cls` lists should hold those components such that the ith person's name will be in `names[i]` and their associated class in `cls[i]`.

In [10]:
```
# read in the dataset
df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases
    header=None,
    sep=',')

# separate names from classes
vals = df.iloc[:,:].values
#print(df)
#print(vals)

### FILL IN THE BLANKS ###

vals_list = vals.tolist()

cls = []
names = []

for x in vals_list:
    cls.append(x[0].split(' ', 1)[0])
    names.append(x[0].split(' ', 1)[1])

#print(cls)
#print(names)
```

Write a function that, given a name and a `c-mer` length parameter `c`, will create the list of `c-mers` for the name.

In [11]:
```
def cmer(name, c=3):
    r""" Given a name and parameter c, return the vector of c-mers associated wit
    """
    name = name.lower()

    ### FILL IN THE BLANKS ###

    v = []

    for x in range(0, len(name)-c+1):
        v.append(name[x:x+c])


    return v
```

The following functions will be useful in later tasks. Study them carefully.

In [12]:
```
def build_matrix(docs):
    r""" Build sparse matrix from a list of documents,
    each of which is a list of word/terms in the document.
    """
    nrows = len(docs)
    idx = {}
    tid = 0
    nnz = 0
    for d in docs:
        nnz += len(set(d))
        for w in d:
            if w not in idx:
                idx[w] = tid
                tid += 1
    ncols = len(idx)

    # set up memory
    ind = np.zeros(nnz, dtype=np.int)
    val = np.zeros(nnz, dtype=np.double)
    ptr = np.zeros(nrows+1, dtype=np.int)
    i = 0  # document ID / row counter
    n = 0  # non-zero counter
    # transfer values
    for d in docs:
        cnt = Counter(d)
        keys = list(k for k,_ in cnt.most_common())
        l = len(keys)
        for j,k in enumerate(keys):
            ind[j+n] = idx[k]
            val[j+n] = cnt[k]
        ptr[i+1] = ptr[i] + l
        n += l
        i += 1

    mat = csr_matrix((val, ind, ptr), shape=(nrows, ncols), dtype=np.double)
    mat.sort_indices()

    return mat

def csr_info(mat, name="", non_empy=False):
    r""" Print out info about this CSR matrix. If non_empy,
    report number of non-empty rows and cols as well
    """
    if non_empy:
        print("%s [nrows %d (%d non-empty), ncols %d (%d non-empty), nnz %d]" % (
                name, mat.shape[0],
                sum(1 if mat.indptr[i+1] > mat.indptr[i] else 0
                for i in range(mat.shape[0])),
                mat.shape[1], len(np.unique(mat.indices)),
                len(mat.data)))
    else:
        print( "%s [nrows %d, ncols %d, nnz %d]" % (name,
                mat.shape[0], mat.shape[1], len(mat.data)) )

def csr_l2normalize(mat, copy=False, **kargs):
    r""" Normalize the rows of a CSR matrix by their L-2 norm.
```

```
    If copy is True, returns a copy of the normalized matrix.
    """
    if copy is True:
        mat = mat.copy()
    nrows = mat.shape[0]
    nnz = mat.nnz
    ind, val, ptr = mat.indices, mat.data, mat.indptr
    # normalize
    for i in range(nrows):
        rsum = 0.0
        for j in range(ptr[i], ptr[i+1]):
            rsum += val[j]**2
        if rsum == 0.0:
            continue  # do not normalize empty rows
        rsum = 1.0/np.sqrt(rsum)
        for j in range(ptr[i], ptr[i+1]):
            val[j] *= rsum

    if copy is True:
        return mat

def namesToMatrix(names, c):
    docs = [cmer(n, c) for n in names]
    return build_matrix(docs)
```

Compare the sparse matrix statistics (via `csr_info`) for c-mer representations of the names given $c \in \{1, 2, 3\}$.

In [15]:
```
for c in range(1, 4):
    mat = namesToMatrix(names, c)
    csr_info(mat)
```

```
[nrows 294, ncols 30, nnz 3054]
[nrows 294, ncols 442, nnz 3739]
[nrows 294, ncols 1695, nnz 3527]

C:\Users\Checkout\AppData\Local\Temp\ipykernel_8824\571971379.py:18: Deprecatio
nWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this
warning, use `int` by itself. Doing this will not modify any behavior and is sa
fe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32`
to specify the precision. If you wish to review your current use, check the rel
ease note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  ind = np.zeros(nnz, dtype=np.int)
C:\Users\Checkout\AppData\Local\Temp\ipykernel_8824\571971379.py:20: Deprecatio
nWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this
warning, use `int` by itself. Doing this will not modify any behavior and is sa
fe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32`
 to specify the precision. If you wish to review your current use, check the re
lease note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  ptr = np.zeros(nrows+1, dtype=np.int)
```

We'll now define a function to search for the top-k neighbors for a given name (one of the objects the dataset), where proximity is computed via cosine similarity.

In [17]:
```
def findNeighborsForName(name, c=1, k=1):
    # first, find the document for the given name
    id = -1
    for i in range(len(names)):
        if names[i] == name:
            id = i
            break
    if id == -1:
        print("Name %s not found." % name)
        return []
    # now, compute similarities of name's vector against all other name vectors
    mat = namesToMatrix(names, c)
    csr_l2normalize(mat)
    x = mat[id,:]
    dots = x.dot(mat.T)
    dots[0,id] = -1 # invalidate self-similarity
    sims = list(zip(dots.indices, dots.data))
    sims.sort(key=lambda x: x[1], reverse=True)
    return [names[s[0]] for s in sims[:k] if s[1] > 0 ]
```

Let c=2 and k=5. Which are the closest neighbors for “Michael Kearns”, in decreasing order of similarity?

In [18]:
```
findNeighborsForName("Michael Kearns", c=2, k=5)
```

```
C:\Users\Checkout\AppData\Local\Temp\ipykernel_8824\571971379.py:18: Deprecatio
nWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this
warning, use `int` by itself. Doing this will not modify any behavior and is sa
fe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32`
to specify the precision. If you wish to review your current use, check the rel
ease note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  ind = np.zeros(nnz, dtype=np.int)
C:\Users\Checkout\AppData\Local\Temp\ipykernel_8824\571971379.py:20: Deprecatio
nWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this
warning, use `int` by itself. Doing this will not modify any behavior and is sa
fe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32`
 to specify the precision. If you wish to review your current use, check the re
lease note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  ptr = np.zeros(nrows+1, dtype=np.int)
```

Out[18]:
```
['Michael W. Barley',
 'Michael Redmond',
 'Michael Littman',
 'Michael A. meystel',
 'Michael I. Jordan']
```

Finally, we'll define a couple functions to perform d-fold cross-validation, defaulting to $d = 10$. Double-check the code for errors. What does the line

```
tc = Counter(clstr[s[0]] for s in sims[:k]).most_common(2)
```

do?

In [20]:
```
def splitData(mat, cls, fold=1, d=10):
    r""" Split the matrix and class info into train and test data using d-fold ho
    """
    n = mat.shape[0]
    r = int(np.ceil(n*1.0/d))
    mattr = []
    clstr = []
    # split mat and cls into d folds
    for f in range(d):
        if f+1 != fold:
            mattr.append( mat[f*r: min((f+1)*r, n)] )
            clstr.extend( cls[f*r: min((f+1)*r, n)] )
    # join all fold matrices that are not the test matrix
    train = sp.vstack(mattr, format='csr')
    # extract the test matrix and class values associated with the test rows
    test = mat[(fold-1)*r: min(fold*r, n), :]
    clste = cls[(fold-1)*r: min(fold*r, n)]

    return train, clstr, test, clste

def classifyNames(names, cls, c=3, k=3, d=10):
    r""" Classify names using c-mer frequency vector representations of the names
    cosine similarity and 10-fold cross validation
    """
    docs = [cmer(n, c) for n in names]
    mat = build_matrix(docs)
    # since we're using cosine similarity, normalize the vectors
    csr_l2normalize(mat)

    def classify(x, train, clstr):
        r""" Classify vector x using kNN and majority vote rule given training da
        """
        # find nearest neighbors for x
        dots = x.dot(train.T)
        sims = list(zip(dots.indices, dots.data))
        if len(sims) == 0:
            # could not find any neighbors
            return '+' if np.random.rand() > 0.5 else '-'
        sims.sort(key=lambda x: x[1], reverse=True)
        tc = Counter(clstr[s[0]] for s in sims[:k]).most_common(2)
        if len(tc) < 2 or tc[0][1] > tc[1][1]:
            # majority vote
            return tc[0][0]
        # tie break
        tc = defaultdict(float)
        for s in sims[:k]:
            tc[clstr[s[0]]] += s[1]
        return sorted(tc.items(), key=lambda x: x[1], reverse=True)[0][0]

    macc = 0.0
    for f in range(d):
        # split data into training and testing
        train, clstr, test, clste = splitData(mat, cls, f+1, d)
        # predict the class of each test sample
        clspr = [ classify(test[i,:], train, clstr) for i in range(test.shape[0])
        # compute the accuracy of the prediction
```

```
        acc = 0.0
        for i in range(len(clste)):
            if clste[i] == clspr[i]:
                acc += 1
        acc /= len(clste)
        macc += acc

    return macc/d
```

Given $c \in \{1, \ldots, 4\}$ and $k \in \{1, \ldots, 6\}$, which meta-parameters result in the highest accuracy?

In [21]:
```
c = [1, 2, 3, 4]
k = [1, 2, 3, 4, 5, 6]
list_accuracy = []
for x in range(0, len(c)):
    for y in range(0, len(k)):
        temp = classifyNames(names, cls, c[x], k[y], d=10)

        if(list_accuracy):
            if(temp > max(list_accuracy)):
                c_max = c[x]
                k_max = k[y]
        list_accuracy.append(temp)

print(list_accuracy)
print("\n")
print("Highest accuracy is " + str(max(list_accuracy)) + " for c = " + str(c_max)
```

```
C:\Users\Checkout\AppData\Local\Temp\ipykernel_8824\571971379.py:18: Deprecatio
nWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this
warning, use `int` by itself. Doing this will not modify any behavior and is sa
fe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32`
to specify the precision. If you wish to review your current use, check the rel
ease note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  ind = np.zeros(nnz, dtype=np.int)
C:\Users\Checkout\AppData\Local\Temp\ipykernel_8824\571971379.py:20: Deprecatio
nWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this
warning, use `int` by itself. Doing this will not modify any behavior and is sa
fe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32`
 to specify the precision. If you wish to review your current use, check the re
lease note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  ptr = np.zeros(nrows+1, dtype=np.int)

[0.6191666666666666, 0.6191666666666666, 0.6649999999999999, 0.663333333333333
3, 0.6799999999999999, 0.6758333333333333, 0.7308333333333333, 0.73083333333333
33, 0.7474999999999999, 0.7508333333333332, 0.7166666666666667, 0.7433333333333
333, 0.8008333333333333, 0.8008333333333333, 0.7508333333333332, 0.764166666666
6665, 0.7491666666666666, 0.75, 0.7575000000000001, 0.7483333333333333, 0.7575,
0.7508333333333332, 0.7241666666666666, 0.74]


Highest accuracy is 0.8008333333333333 for c = 3 and k = 1
```

In [ ]: