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), 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 [56]:

```
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 [57]:

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

names = []
cls = []
# separate names from classes
vals = df.iloc[:,:].values
for a in vals:
    temp = a[0].split()
    names.append(" ".join(temp[1:]))
    cls.append(temp[0])
### FILL IN THE BLANKS ###
print(names)
print(cls)
```

```
['Naoki Abe', 'Myriam Abramson', 'David W. Aha', 'Kamal M. Ali',
'Eric Allender', 'Dana Angluin', 'Chidanand Apte', 'Minoru Asad
a', 'Lars Asker', 'Javed Aslam', 'Haralabos Athanassiou', 'Jose
L. Balcazar', 'Timothy P. Barber', 'Michael W. Barley', 'Cristina
Baroglio', 'Peter Bartlett', 'Eric Baum', 'Welton Becket', 'Shai
Ben-David', 'George Berg', 'Neil Berkman', 'Malini Bhandaru', 'Bi
r Bhanu', 'Reinhard Blasig', 'Avrim Blum', 'Anselm Blumer', 'Just
in Boyan', 'Carla E. Brodley', 'Nader Bshouty', 'Wray Buntine',
'Andrey Burago', 'Tom Bylander', 'Bill Byrne', 'Claire Cardie',
'Richard A. Caruana', 'John Case', 'Jason Catlett', 'Nicolo Cesa-
Bianchi', 'Philip Chan', 'Mark Changizi', 'Pang-Chieh Chen', 'Zhi
xiang Chen', 'Wan P. Chiang', 'Steve A. Chien', 'Jeffery Clouse',
'William Cohen', 'David Cohn', 'Clare Bates Congdon', 'Antoine Co
rnuejols', 'Mark W. Craven', 'Robert P. Daley', 'Lindley Darden',
'Chris Darken', 'Bhaskar Dasgupta', 'Brian D. Davidson', 'Michael
de la Maza', 'Olivier De Vel', 'Scott E. Decatur', 'Gerald F. DeJ
ong', 'Kan Deng', 'Thomas G. Dietterich', 'Michael J. Donahue',
'George A. Drastal', 'Harris Drucker', 'Chris Drummond', 'Hal Dun
can', 'Thomas Ellman', 'Tapio Elomaa', 'Susan L. Epstein', 'Bob E
vans', 'Claudio Facchinetti', 'Tom Fawcett', 'Usama Fayyad', 'Aar
on Feigelson', 'Nicolas Fiechter', 'David Finton', 'John Fische
r', 'Paul Fischer', 'Seth Flanders', 'Lance Fortnow', 'Ameur Foue
d', 'Judy A. Franklin', 'Yoav Freund', 'Johannes Furnkranz', 'Mer
rick L. Furst', 'Jean Gabriel Ganascia', 'William Gasarch', 'Rica
rd Gavalda', 'Melinda T. Gervasio', 'Yolanda Gil', 'David Gillma
```

n', 'Attilio Giordana', 'Kate Goelz', 'Paul W. Goldberg', 'Sally Goldman', 'Diana Gordon', 'Geoffrey Gordon', 'Jonathan Gratch', 'Leslie Grate', 'William A. Greene', 'Russell Greiner', 'Marko Grobelnik', 'Tal Grossman', 'Margo Guertin', 'Tom Hancock', 'Earl S. Harris Jr.', 'David Haussler', 'Matthias Heger', 'Lisa Hellerstein', 'David Helmbold', 'Daniel Hennessy', 'Haym Hirsh', 'Jonathan Hodgson', 'Robert C. Holte', 'Jiarong Hong', 'Chun-Nan Hsu', 'Kazushi Ikeda', 'Masayuki Inaba', 'Drago Indjic', 'Nitin Indurkhya', 'Jeff Jackson', 'Sanjay Jain', 'Wolfgang Janko', 'Klaus P. Jantke', 'Nathalie Japkowicz', 'George H. John', 'Randolph Jones', 'Michael I. Jordan', 'Leslie Pack Kaelbling', 'Bala Kalyanasundaram', 'Thomas E. Kammeyer', 'Grigoris Karakoulas', 'Michael Kearns', 'Neela Khan', 'Roni Khardon', 'Dennis F. Kibler', 'Jorg-Uwe Kietz', 'Efim Kinber', 'Jyrki Kivinen', 'Emanuel Knill', 'Craig Knoblock', 'Ron Kohavi', 'Pascal Koiran', 'Moshe Koppel', 'Daniel Kortenkamp', 'Matevz Kovacic', 'Stefan Kramer', 'Martinch Krikis', 'Martin Kummer', 'Eyal Kushilevitz', 'Stephen Kwek', 'Wai Lam', 'Ken Lang', 'Steffen Lange', 'Pat Langley', 'Mary Soon Lee', 'Wee Sun Lee', 'Moshe Leshno', 'Long-Ji Lin', 'Charles X. Ling', 'Michael Littman', 'David Loewenstern', 'Phil Long', 'Wolfgang Maass', 'Bruce A. MacDonald', 'Rich Maclin', 'Sridhar Mahadevan', 'J. Jeffrey Mahoney', 'Yishay Mansour', 'Mario Marchand', 'Shaul Markovitch', 'Oded Maron', 'Maja Mataric', 'David Mathias', 'Toshiyasu Matsushima', 'Stan Matwin', 'Eddy Mayoraz', 'R. Andrew McCallum', 'L. Thorne McCarty', 'Alexander M. Meystel', 'Michael A. meystel', 'Steven Minton', 'Nina Mishra', 'Tom M. Mitchell', 'Dunja Mladenic', 'David Montgomery', 'Andrew W. Moore', 'Johanne Morin', 'Hiroshi Motoda', 'Stephen Muggleton', 'Patrick M. Murphy', 'Sreerama K. Murthy', 'Filippo Neri', 'Craig Nevill-Manning', 'Andrew Y. Ng', 'Nikolay Nikolaev', 'Steven W. Norton', "Joseph O'Sullivan", 'Dan Oblinger', 'Jong-Hoon Oh', 'Arlindo Oliveira', 'David W. Opitz', 'Sandra Panizza', 'Barak A. Pearlmutter', 'Ed Pednault', 'Jing Peng', 'Fernando Pereira', 'Aurora Perez', 'Bernhard Pfahringer', 'David Pierce', 'Krishnan Pillaipakkamnatt', 'Roberto Piola', 'Leonard Pitt', 'Lorien Y. Pratt', 'Armand Prieditis', 'Foster J. Provost', 'J. R. Quinlan', 'John Rachlin', 'Vijay Raghavan', 'R. Bharat Rao', 'Priscilla Rasmussen', 'Joel Ratsaby', 'Michael Redmond', 'Patricia J. Riddle', 'Lance Riley', 'Ronald L. Rivest', 'Huw Roberts', 'Dana Ron', 'Robert S. Roos', 'Justinian Rosca', 'John R. Rose', 'Dan Roth', 'James S. Royer', 'Ronitt Rubinfeld', 'Stuart Russell', 'Lorenza Saitta', 'Yoshifumi Sakai', 'William Sakas', 'Marcos Salganicoff', 'Steven Salzberg', 'Claude Sammut', 'Cullen Schaffer', 'Robert Schapire', 'Mark Schwabacher', 'Michele Sebag', 'Gary M. Selzer', 'Sebastian Seung', 'Arun Sharma', 'Jude Shavlik', 'Daniel L. Silver', 'Glenn Silverstein', 'Yoram Singer', 'Mona Singh', 'Satinder Pal Singh', 'Kimmen Sjolander', 'David B. Skalak', 'Sean Slattery', 'Robert Sloan', 'Donna Slonim', 'Carl H. Smith', 'Sonya Snedecor', 'Von-Wun Soo', 'Thomas G. Spalthoff', 'Mark Staley', 'Frank Stephan', 'Mandayam T. Suraj', 'Richard S. Sutton', 'Joe Suzuki', 'Prasad Tadepalli', 'Hiroshi Tanaka', 'Irina Tchoumatchenko', 'Brian Tester', 'Chen K. Tham', 'Tatsuo Unemi', 'Lyle H. Ungar', 'Paul Utgoff', 'Karsten Verbeurgt', 'Paul Vitanyi', 'Xuemei Wang', 'Manfred Warmuth', 'Gary Weiss', 'Sholom Weiss', 'Thomas Wengerek', 'Bradley L. Whitehall', 'Alma Whitten', 'Robert Williamson', 'Janusz Wnek', 'Kenji Yamanishi', 'Takefumi Yamazaki', 'Holly Yanco', 'John M. Zelle', 'Thomas Zeugmann', 'Jean-Daniel Zucker', 'Darko Zupanic']

```
['+', '-', '+', '+', '-', '+', '-', '+', '+', '+', '+', '+', '+',
'+', '-', '+', '-', '+', '-', '+', '+', '+', '+', '+', '-', '-',
'+', '+', '+', '-', '-', '+', '+', '-', '+', '+', '+', '+', '-',
'+', '+', '-', '+', '-', '+', '+', '+', '-', '-', '+', '+', '+',
'-', '-', '-', '+', '-', '-', '+', '+', '-', '+', '+', '+', '-',
'+', '-', '+', '+', '+', '-', '+', '-', '+', '+', '+', '+', '+',
'+', '+', '-', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+',
'-', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+',
'+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '-', '+',
'+', '-', '+', '+', '+', '+', '-', '+', '+', '+', '+', '+', '+',
'-', '-', '+', '+', '+', '+', '+', '-', '-', '-', '-', '+', '+',
'+', '+', '+', '-', '+', '+', '-', '-', '+', '+', '-', '+', '+',
'+', '+', '+', '-', '+', '+', '-', '+', '-', '+', '-', '-', '+',
'+', '-', '-', '+', '+', '+', '-', '-', '-', '-', '-', '+', '-',
'+', '+', '+', '+', '-', '+', '+', '-', '+', '-', '+', '-', '-',
'+', '-', '+', '+', '+', '-', '+', '+', '+', '-', '+', '+', '+',
'+', '+', '-', '+', '+', '+', '-', '+', '-', '+', '+', '-', '-',
'+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+',
'-', '+', '+', '+', '+', '-', '-', '+', '+', '+', '+', '+', '+',
'-', '+', '+', '-', '+', '+', '+', '+', '+', '+', '+', '+', '+',
'+', '+', '-', '+', '-', '+', '+', '+', '-', '+', '-', '-', '-',
'+', '-', '+', '+', '+', '+', '+', '+', '-', '-', '-', '-', '+',
'+', '+', '+', '+', '+', '-', '+', '+']
```

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 [58]:

```
def cmer(name, c=3):
    r""" Given a name and parameter c, return the vector of c-mer
    """
    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 [59]:

```
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
    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-empt
                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 [60]:

```
mat1 = namesToMatrix(names,1)
mat2 = namesToMatrix(names,2)
mat3 = namesToMatrix(names,3)

print(csr_info(mat1))
print(csr_info(mat2))
print(csr_info(mat3))
```

```
 [nrows 294, ncols 30, nnz 3054]
None
 [nrows 294, ncols 442, nnz 3739]
None
 [nrows 294, ncols 1695, nnz 3527]
None
<ipython-input-59-0ec70edba228>:18: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warn
ing, use `int` by itself. Doing this will not modify any behavior
and is safe. When replacing `np.int`, you may wish to use e.g. `n
p.int64` or `np.int32` to specify the precision. If you wish to r
eview your current use, check the release note link for additiona
l information.
Deprecated in NumPy 1.20; for more details and guidance: https://
numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  ind = np.zeros(nnz, dtype=np.int)
<ipython-input-59-0ec70edba228>:20: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warn
ing, use `int` by itself. Doing this will not modify any behavior
and is safe. When replacing `np.int`, you may wish to use e.g. `n
p.int64` or `np.int32` to specify the precision. If you wish to r
eview your current use, check the release note link for additiona
l information.
Deprecated in NumPy 1.20; for more details and guidance: 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 [61]:

```
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 othe
    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 [62]:

```
findNeighborsForName("Michael Kearns", c=2, k=5)
```

```
<ipython-input-59-0ec70edba228>:18: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warn
ing, use `int` by itself. Doing this will not modify any behavior
and is safe. When replacing `np.int`, you may wish to use e.g. `n
p.int64` or `np.int32` to specify the precision. If you wish to r
eview your current use, check the release note link for additiona
l information.
Deprecated in NumPy 1.20; for more details and guidance: https://
numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  ind = np.zeros(nnz, dtype=np.int)
<ipython-input-59-0ec70edba228>:20: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warn
ing, use `int` by itself. Doing this will not modify any behavior
and is safe. When replacing `np.int`, you may wish to use e.g. `n
p.int64` or `np.int32` to specify the precision. If you wish to r
eview your current use, check the release note link for additiona
l information.
Deprecated in NumPy 1.20; for more details and guidance: https://
numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  ptr = np.zeros(nrows+1, dtype=np.int)
```

Out[62]:
```
['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 [63]:
```
def splitData(mat, cls, fold=1, d=10):
    r""" Split the matrix and class info into train and test data
    """
    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 th
    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 representati
    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 g
        """
        # 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=Tru

    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 rang
        # 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 [64]:

```
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
```

```
<ipython-input-59-0ec70edba228>:18: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warn
ing, use `int` by itself. Doing this will not modify any behavior
and is safe. When replacing `np.int`, you may wish to use e.g. `n
p.int64` or `np.int32` to specify the precision. If you wish to r
eview your current use, check the release note link for additiona
l information.
```

```
Deprecated in NumPy 1.20; for more details and guidance: https://
numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  ind = np.zeros(nnz, dtype=np.int)
<ipython-input-59-0ec70edba228>:20: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warn
ing, use `int` by itself. Doing this will not modify any behavior
and is safe. When replacing `np.int`, you may wish to use e.g. `n
p.int64` or `np.int32` to specify the precision. If you wish to r
eview your current use, check the release note link for additiona
l information.
Deprecated in NumPy 1.20; for more details and guidance: 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, 0.66
33333333333333, 0.6799999999999999, 0.6758333333333333, 0.7308333
333333333, 0.7308333333333333, 0.7474999999999999, 0.750833333333
3332, 0.7166666666666667, 0.7433333333333333, 0.8008333333333333,
0.8008333333333333, 0.7508333333333332, 0.7641666666666665, 0.749
1666666666666, 0.75, 0.7625000000000001, 0.7691666666666668, 0.7
4, 0.7583333333333333, 0.7416666666666667, 0.7350000000000001]


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

In [ ]: