Assignment 3

Bryan and Thanmaye

Part 1:

The spread on each of the classes (except group 1) is quite large, this would indicate that the pre-defined classification might not be the best. One thing to consider is that neither the inner product nor the polynomial kernel are adjusted for file size, so the grouping could still be fine.

Results

```
dot kernel      You, 18 hours ago • print
class_matrix: 1
Mean: 44688507.0
Median: 44688507.0
Standard Deviation: 0.0
Minimum Value: 44688507.0
Maximum Value: 44688507.0

class_matrix: 2
Mean: 93199874.6122449
Median: 11544829.0
Standard Deviation: 311484467.5106672
Minimum Value: 592990.0
Maximum Value: 2120831379.0

class_matrix: 3
Mean: 443357.08
Median: 339719.0
Standard Deviation: 338497.8522679776
Minimum Value: 127770.0
Maximum Value: 2882809.0

class_matrix: 6
Mean: 681860.4375
Median: 479433.5
Standard Deviation: 658661.7063671579
Minimum Value: 143651.0
Maximum Value: 2834812.0

class_matrix: 7
Mean: 127975.72
Median: 71977.0
Standard Deviation: 249007.74030114326
Minimum Value: 0.0
Maximum Value: 961139.0
```

```
poly kernel
class_matrix: 1
Mean: 1997062747266064.0
Median: 1997062747266064.0
Standard Deviation: 0.0
Minimum Value: 1997062747266064.0
Maximum Value: 1997062747266064.0

class_matrix: 2
Mean: 1.0570879031454181e+17
Median: 133283099728900.0
Standard Deviation: 6.36172905665562e+17
Minimum Value: 351638326081.0
Maximum Value: 4.4979257423927045e+18

class_matrix: 3
Mean: 311147183091.32
Median: 115409678400.0
Standard Deviation: 849747417959.1481
Minimum Value: 16325428441.0
Maximum Value: 8310593496100.0

class_matrix: 6
Mean: 898770263384.0625
Median: 229905651982.5
Standard Deviation: 1920249292627.89
Minimum Value: 20635897104.0
Maximum Value: 8036164744969.0

class_matrix: 7
Mean: 78382895591.84
Median: 5180832484.0
Standard Deviation: 249328835289.98972
Minimum Value: 1.0
Maximum Value: 923790099600.0
```

Part 2:

Some stemming was done, as you can see in the picture there was some stemming for various arithmetic operations, jump and move operations. This seems to have to have improved the average kernel by cluster as well as the standard deviation.

```
    add = [term for term in vocab if "add" in term]
    sub = [term for term in vocab if "sub" in term]
    mul = [term for term in vocab if "mul" in term]
    div = [term for term in vocab if "div" in term]
    jump = [term for term in vocab if term.startswith("j")]
    push = [term for term in vocab if "push" in term]
    mov = [term for term in vocab if "mov" in term]
    num = [term for term in vocab if term.isnumeric()]

    new_vocab = {}
    new_vocab.update(dict.fromkeys(add, "add"))
    new_vocab.update(dict.fromkeys(sub, "sub"))
    new_vocab.update(dict.fromkeys(mul, "mul"))
    new_vocab.update(dict.fromkeys(div, "div"))
    new_vocab.update(dict.fromkeys(jump, "jump"))
    new_vocab.update(dict.fromkeys(push, "push"))
    new_vocab.update(dict.fromkeys(mov, "mov"))
    new_vocab.update(dict.fromkeys(num, "num"))
    return new_vocab
```

Results

```
dot kernel
class_matrix: 1
Mean: 36343885.0
Median: 36343885.0
Standard Deviation: 0.0
Minimum Value: 36343885.0
Maximum Value: 36343885.0

class_matrix: 2
Mean: 84707314.1632653
Median: 10033767.0
Standard Deviation: 267630438.51785988
Minimum Value: 497384.0
Maximum Value: 1790510980.0

class_matrix: 3        You, 19 hours ago •
Mean: 342426.13
Median: 245530.0
Standard Deviation: 302749.2850652386
Minimum Value: 101411.0
Maximum Value: 2527424.0

class_matrix: 6
Mean: 562501.625
Median: 370874.0
Standard Deviation: 542192.5743252893
Minimum Value: 90363.0
Maximum Value: 2294845.0

class_matrix: 7
Mean: 112369.32
Median: 63744.0
Standard Deviation: 231279.40933558613
Minimum Value: 0.0
Maximum Value: 961127.0
```

```
poly kernel
class_matrix: 1
Mean: 1320878049580996.0
Median: 1320878049580996.0
Standard Deviation: 0.0
Minimum Value: 1320878049580996.0
Maximum Value: 1320878049580996.0

class_matrix: 2
Mean: 7.880138086343074e+16
Median: 100676500277824.0
Standard Deviation: 4.5429963931276634e+17
Minimum Value: 247391838225.0
Maximum Value: 3.2059295730815826e+18

class_matrix: 3
Mean: 208913468967.55
Median: 60285471961.0
Standard Deviation: 655014752454.431
Minimum Value: 10284393744.0
Maximum Value: 6387877130625.0

class_matrix: 6
Mean: 610381990785.375
Median: 141305710429.0
Standard Deviation: 1263621849076.1223
Minimum Value: 8165652496.0
Maximum Value: 5266318163716.0

class_matrix: 7
Mean: 66117253999.52
Median: 4063425025.0
Standard Deviation: 216835858539.06482
Minimum Value: 1.0
Maximum Value: 923767032384.0
```

Part 3:

A stronger remapping of the terms was applied in part 3. Though I don't think this was a significant improvement

```
    arith = [
        term for term in vocab if any(s in term for s in ["add", "sub", "mul", "div"])
    ]
    jump = [term for term in vocab if term.startswith("j")]
    data = [term for term in vocab if any(s in term for s in ["mov", "push"])]
    num = [term for term in vocab if term.isnumeric()]

    new_vocab = {}
    new_vocab.update(dict.fromkeys(num, "num"))
    new_vocab.update(dict.fromkeys(arith, "arith"))
    new_vocab.update(dict.fromkeys(jump, "jump"))
    new_vocab.update(dict.fromkeys(data, "data"))

    return new_vocab
```

Results

```
dot kernel
class_matrix: 1
Mean: 36343885.0
Median: 36343885.0
Standard Deviation: 0.0
Minimum Value: 36343885.0
Maximum Value: 36343885.0

class_matrix: 2
Mean: 84707314.1632653
Median: 10033767.0
Standard Deviation: 267630438.51785988
Minimum Value: 497384.0
Maximum Value: 1790510980.0

class_matrix: 3
Mean: 342426.13
Median: 245530.0
Standard Deviation: 302749.2850652386
Minimum Value: 101411.0
Maximum Value: 2527424.0

class_matrix: 6
Mean: 562501.625
Median: 370874.0
Standard Deviation: 542192.5743252893
Minimum Value: 90363.0
Maximum Value: 2294845.0

class_matrix: 7
Mean: 112369.32
Median: 63744.0
Standard Deviation: 231279.40933558613
Minimum Value: 0.0
Maximum Value: 961127.0
```

```
poly kernel
class_matrix: 1
Mean: 1320878049580996.0
Median: 1320878049580996.0
Standard Deviation: 0.0
Minimum Value: 1320878049580996.0
Maximum Value: 1320878049580996.0

class_matrix: 2
Mean: 7.880138086343074e+16
Median: 100676500277824.0
Standard Deviation: 4.5429963931276634e+17
Minimum Value: 247391838225.0
Maximum Value: 3.2059295730815826e+18

class_matrix: 3
Mean: 208913468967.55
Median: 60285471961.0
Standard Deviation: 655014752454.431
Minimum Value: 10284393744.0
Maximum Value: 6387877130625.0

class_matrix: 6
Mean: 610381990785.375
Median: 141305710429.0
Standard Deviation: 1263621849076.1223
Minimum Value: 8165652496.0
Maximum Value: 5266318163716.0

class_matrix: 7
Mean: 66117253999.52
Median: 4063425025.0
Standard Deviation: 216835858539.06482
Minimum Value: 1.0
Maximum Value: 923767032384.0
```

Part 4:

In this part we analyze our own spectral clusters using silhouettes. Using each dictionary and both kernels here are the silhouettes. The interesting thing about the results is that the two best performing models are d1, inner product with k=2 and D2, polynomial with k=3. Another interesting note is that there doesn't seem to be any correlation between the number of clusters and the performance, and that while the original had 5 different cluster, the best performance is with less clusters.

D0, inner product

```
spectral clustering k=2
silhouette score: 0.5424089619095536

spectral clustering k=3
silhouette score: -0.520366289024476

spectral clustering k=4
silhouette score: -0.3939736143467031
```

D0, polynomial

```
spectral clustering k=2
silhouette score: 0.4468168836852343

spectral clustering k=3
silhouette score: -0.6187588948716103

spectral clustering k=4
silhouette score: -0.6245173996640986
```

D1, inner product

```
spectral clustering k=2
silhouette score: 0.7114378991749604

spectral clustering k=3
silhouette score: 0.5191548145181609

spectral clustering k=4
silhouette score: -0.34234570543561926
```

D0, polynomial

```
spectral clustering k=2
silhouette score: 0.063232394792473

spectral clustering k=3
silhouette score: -0.12574724967241038

spectral clustering k=4
silhouette score: -0.8123473161830655
```

D2, inner product

```
spectral clustering k=2
silhouette score: 0.08320696226484069

spectral clustering k=3
silhouette score: -0.5812461066454849

spectral clustering k=4
silhouette score: -0.4103802675750889
```

D2, polynomial

```
spectral clustering k=2
silhouette score: 0.29983986384897

spectral clustering k=3
silhouette score: 0.7629542041328409

spectral clustering k=4
silhouette score: -0.7615254569227531
```

Code

```
“””main.py”””

import kernel

import spectral

import vectorize

import vocab

import numpy as np

import os

from typing import List



def silhouette(K: np.ndarray, clusters: np.ndarray) -> float:
    """
    Compute the silhouette score
    """
    # compute average distane between points in the same cluster
    d = np.zeros(K.shape[0])
    for i in range(K.shape[0]):
        d[i] = np.mean(K[i, clusters == clusters[i]])

    # compute the smallest average distance between a point and any other cluster
    D = np.zeros(K.shape[0])
    for i in range(K.shape[0]):
        D[i] = np.min(
            [np.mean(K[i, clusters == j])
             for j in set(clusters) if j != clusters[i]]
```

```
    )

  # compute the silhouette score
  s = (D - d) / np.maximum(d, D)
  return np.mean(s)


if __name__ == "__main__":

  # list of filenames
  files = [f"./data/{f}" for f in os.listdir("./data")]
  files.remove("./data/.DS_Store")
  class_sizes = vectorize.classCounts(files)
  class_keys = sorted(list(class_sizes.keys()))
  results = open("results.txt", "w")
  """part 1"""
  results.writelines("PART 1!!:\n")
  # dictionary of terms across all files
  D0 = vocab.dictionary(files)
  # re-vectorize with revised dictionay
  vectors: List[np.ndarray] = [vectorize.vectorize(f, D0) for f in files]

  # D is the Document-term matrix
  D = np.vstack(vectors)

  # K is the kernel matrix
```

```
K0 = kernel.buildKernel(D, "dot")
results.writelines("dot kernel\n")
sub_matrices = kernel.extractSubmatrices(K0, class_sizes, class_keys)
[
    kernel.descriptiveStats(kernel_mat, class_label, results)
    for class_label, kernel_mat in sub_matrices.items()
]

for k in range(2, 5):
    clusters = spectral.spectralClustering(K0, k)
    silhouette_score = silhouette(K0, clusters)
    results.writelines(f"spectral clustering k={k}\n")
    results.writelines(f"silhouette score: {silhouette_score}\n")
    results.writelines("\n")

Kp = kernel.buildKernel(D, "poly")

results.writelines("poly kernel\n")
sub_matrices = kernel.extractSubmatrices(Kp, class_sizes, class_keys)
[
    kernel.descriptiveStats(kernel_mat, class_label, results)
    for class_label, kernel_mat in sub_matrices.items()
]

for k in range(2, 5):
    clusters = spectral.spectralClustering(Kp, k)
```

```
    silhouette_score = silhouette(Kp, clusters)
    results.writelines(f"spectral clustering k={k}\n")
    results.writelines(f"silhouette score: {silhouette_score}\n")
    results.writelines("\n")

"""part 2"""
results.writelines("PART 2!!:\n")
# dictionary of terms across all files
D1 = vocab.dictionary(files, stem=True)
# re-vectorize with revised dictionay
vectors: List[np.ndarray] = [vectorize.vectorize(f, D1) for f in files]

# D is the Document-term matrix
D = np.vstack(vectors)

# K is the kernel matrix
K0 = kernel.buildKernel(D, "dot")
results.writelines("dot kernel\n")
sub_matrices = kernel.extractSubmatrices(K0, class_sizes, class_keys)
[
    kernel.descriptiveStats(kernel_mat, class_label, results)
    for class_label, kernel_mat in sub_matrices.items()
]

for k in range(2, 5):
    clusters = spectral.spectralClustering(K0, k)
```

```
    silhouette_score = silhouette(K0, clusters)
    results.writelines(f"spectral clustering k={k}\n")
    results.writelines(f"silhouette score: {silhouette_score}\n")
    results.writelines("\n")

Kp = kernel.buildKernel(D, "poly")

results.writelines("poly kernel\n")
sub_matrices = kernel.extractSubmatrices(Kp, class_sizes, class_keys)
[
    kernel.descriptiveStats(kernel_mat, class_label, results)
    for class_label, kernel_mat in sub_matrices.items()
]

for k in range(2, 5):
    clusters = spectral.spectralClustering(Kp, k)
    silhouette_score = silhouette(Kp, clusters)
    results.writelines(f"spectral clustering k={k}\n")
    results.writelines(f"silhouette score: {silhouette_score}\n")
    results.writelines("\n")

"""part 3"""
results.writelines("PART 3!!:\n")
# dictionary of terms across all files
D2 = vocab.dictionary(files, stem=True)
# re-vectorize with revised dictionay
```

```
vectors: List[np.ndarray] = [vectorize.vectorize(f, D2) for f in files]

# D is the Document-term matrix
D = np.vstack(vectors)

# K is the kernel matrix
K0 = kernel.buildKernel(D, "dot")
results.writelines("dot kernel\n")
sub_matrices = kernel.extractSubmatrices(K0, class_sizes, class_keys)
[
  kernel.descriptiveStats(kernel_mat, class_label, results)
  for class_label, kernel_mat in sub_matrices.items()
]

for k in range(2, 5):
  clusters = spectral.spectralClustering(K0, k)
  silhouette_score = silhouette(K0, clusters)
  results.writelines(f"spectral clustering k={k}\n")
  results.writelines(f"silhouette score: {silhouette_score}\n")
  results.writelines("\n")

Kp = kernel.buildKernel(D, "poly")

results.writelines("poly kernel\n")
sub_matrices = kernel.extractSubmatrices(Kp, class_sizes, class_keys)
[
```

```
        kernel.descriptiveStats(kernel_mat, class_label, results)
        for class_label, kernel_mat in sub_matrices.items()
    ]

    for k in range(2, 5):
        clusters = spectral.spectralClustering(Kp, k)
        silhouette_score = silhouette(Kp, clusters)
        results.writelines(f"spectral clustering k={k}\n")
        results.writelines(f"silhouette score: {silhouette_score}\n")
        results.writelines("\n")

    results.close()

“””kernel.py”””
import numpy as np
from collections import Counter
from typing import List, Literal, TextIO



def buildKernel(D: np.ndarray, type: Literal["dot", "poly"]):
    if type == "dot":

        return np.matmul(D, D.T)
    else:
        return (np.matmul(D, D.T) + 1) ** 2
```

```python
def extractSubmatrices(
    K: np.ndarray, class_sizes: Counter, class_keys: List[int]
) -> np.ndarray:

    starts = [0]
    for key in class_keys[:-1]:
        starts.append(starts[-1] + class_sizes[key])

    sub_matrices = {
        key: K[
            starts[i]: starts[i] + class_sizes[key],
            starts[i]: starts[i] + class_sizes[key],
        ]
        for i, key in enumerate(class_keys)
    }
    return sub_matrices



def descriptiveStats(kernel_mat: np.ndarray, class_label: str, results: TextIO) -> None:
    mean_value = np.mean(kernel_mat)
    median_value = np.median(kernel_mat)
    std_deviation = np.std(kernel_mat)
    min_value = np.min(kernel_mat)
    max_value = np.max(kernel_mat)
```

```
    results.writelines(f"class_matrix: {class_label}\n")
    results.writelines(f"Mean: {mean_value}\n")
    results.writelines(f"Median: {median_value}\n")
    results.writelines(f"Standard Deviation: {std_deviation}\n")
    results.writelines(f"Minimum Value: {min_value}\n")
    results.writelines(f"Maximum Value: {max_value}\n")
    results.writelines("\n")

“””spectral.py”””
import numpy as np



class Graph:
    def __init__(self, K: np.ndarray):
        self.W = self.similairyGraph(K)

    def similairyGraph(self, K: np.ndarray, threshold: float = 0.5):
        """
        Create a similarity graph from a kernel matrix
        """
        distances = K.copy()
        d_max = distances.max()
        distances = d_max - distances

        distances[distances < threshold * d_max] = 0
        return distances
```

```
def spectralClustering(W: np.ndarray, k: int):
    """
    Spectral Clustering
    """

    sim_graph = Graph(W)
    W = sim_graph.W
    L = np.diag(W.sum(axis=1)) - W

    # compute the eigenvalues and eigenvectors
    eigvals, eigvecs = np.linalg.eigh(L)
    U = eigvecs[:, :k]
    clusters = kmeans(U, k)
    return clusters


def kmeans(X: np.ndarray, k: int):
    """
    K-means
    """

    centroids = X[np.random.choice(X.shape[0], k, replace=False)]
    prev_centroids = np.zeros(centroids.shape)
    clusters = np.zeros(X.shape[0])
```

```
  error = np.linalg.norm(centroids - prev_centroids)
  while error != 0:
    for i in range(X.shape[0]):
      distances = np.linalg.norm(X[i] - centroids, axis=1)
      clusters[i] = np.argmin(distances)
    prev_centroids = centroids
    for i in range(k):
      centroids[i] = np.mean(X[clusters == i], axis=0)
    error = np.linalg.norm(centroids - prev_centroids)
  return clusters
“””spectral.py”””
import numpy as np



class Graph:
  def __init__(self, K: np.ndarray):
    self.W = self.similairyGraph(K)

  def similairyGraph(self, K: np.ndarray, threshold: float = 0.5):
    """
    Create a similarity graph from a kernel matrix
    """
    distances = K.copy()
    d_max = distances.max()
    distances = d_max - distances
```

```
        distances[distances < threshold * d_max] = 0
        return distances



def spectralClustering(W: np.ndarray, k: int):
    """
    Spectral Clustering
    """

    sim_graph = Graph(W)
    W = sim_graph.W
    L = np.diag(W.sum(axis=1)) - W

    # compute the eigenvalues and eigenvectors
    eigvals, eigvecs = np.linalg.eigh(L)
    U = eigvecs[:, :k]
    clusters = kmeans(U, k)
    return clusters



def kmeans(X: np.ndarray, k: int):
    """
    K-means
    """

    centroids = X[np.random.choice(X.shape[0], k, replace=False)]
```

```
    prev_centroids = np.zeros(centroids.shape)
    clusters = np.zeros(X.shape[0])
    error = np.linalg.norm(centroids - prev_centroids)
    while error != 0:
        for i in range(X.shape[0]):
            distances = np.linalg.norm(X[i] - centroids, axis=1)
            clusters[i] = np.argmin(distances)
        prev_centroids = centroids
        for i in range(k):
            centroids[i] = np.mean(X[clusters == i], axis=0)
        error = np.linalg.norm(centroids - prev_centroids)
    return clusters

“””vectorize.py”””
from collections import Counter
import numpy as np
from typing import List
import re



def classCounts(files: List[str]) -> Counter:
    # Regular expression pattern to extract class information
    pattern = r"(\d+)-"

    # Use list comprehension to extract class information from each file name
    class_numbers = [int(re.search(pattern, file_name).group(1))
```

```
            for file_name in files]
    return Counter(class_numbers)



def vectorize(filename: str, dict: List[str]) -> np.ndarray:
    f = open(filename)
    content = f.read().split()
    vector = [content.count(word) for word in dict]
    np_vector = np.array(vector, dtype=np.float64)
    return np_vector

“””vocab.py”””
from typing import List, Optional
import numpy as np
import vectorize



def part2_stem(vocab: List[str]):
    add = [term for term in vocab if "add" in term]
    sub = [term for term in vocab if "sub" in term]
    mul = [term for term in vocab if "mul" in term]
    div = [term for term in vocab if "div" in term]
    jump = [term for term in vocab if term.startswith("j")]
    push = [term for term in vocab if "push" in term]
    mov = [term for term in vocab if "mov" in term]
    num = [term for term in vocab if term.isnumeric()]
```

```
    new_vocab = {}
    new_vocab.update(dict.fromkeys(add, "add"))
    new_vocab.update(dict.fromkeys(sub, "sub"))
    new_vocab.update(dict.fromkeys(mul, "mul"))
    new_vocab.update(dict.fromkeys(div, "div"))
    new_vocab.update(dict.fromkeys(jump, "jump"))
    new_vocab.update(dict.fromkeys(push, "push"))
    new_vocab.update(dict.fromkeys(mov, "mov"))
    new_vocab.update(dict.fromkeys(num, "num"))
    return new_vocab


def part3_stem(vocab: List[str]):
    arith = [
        term for term in vocab if any(s in term for s in ["add", "sub", "mul", "div"])
    ]
    jump = [term for term in vocab if term.startswith("j")]
    data = [term for term in vocab if any(s in term for s in ["mov", "push"])]
    num = [term for term in vocab if term.isnumeric()]

    new_vocab = {}
    new_vocab.update(dict.fromkeys(num, "num"))
    new_vocab.update(dict.fromkeys(arith, "arith"))
    new_vocab.update(dict.fromkeys(jump, "jump"))
    new_vocab.update(dict.fromkeys(data, "data"))
```

```python
    return new_vocab


def dictionary(
    files: List[str],
    stem: Optional[bool] = False,
    adv_stem: Optional[bool] = False,
):
    assert type(stem) in [bool, False], "stem must be boolean"
    assert type(adv_stem) in [bool, False], "adv_stem must be boolean"

    # dictionary of terms across all files
    vocab = set()
    [vocab := vocab.union(set(open(f).read().split())) for f in files]
    vocab = list(vocab)

    vectors: List[np.ndarray] = [vectorize.vectorize(f, vocab) for f in files]
    # revise dictionary
    stop_indexes = findStopWords(vectors)

    vocab = [vocab[i] for i in range(len(vocab)) if i not in stop_indexes]

    if stem:
        # TODO: implement stemming for part 2
        vocab = part2_stem(vocab)
```

```
    elif adv_stem:
        # TODO: implement stemming for part 3
        vocab = part3_stem(vocab)

    return vocab



def findStopWords(vectors: List[np.ndarray]) -> list:
    """returns the index of the stop words"""
    top_ten_sets = [set(np.argsort(v)[:5]) for v in vectors]
    common_indexes = set.intersection(*top_ten_sets)
    return list(common_indexes)
```