Implementación y Análisis de k-Nearest Neighbors en Imágenes Multibanda

Gustavo Murillo Vega

Introducción

En este proyecto implementamos el algoritmo de k-Nearest Neighbors como un módulo de Python (`knn.py`), y lo aplicamos para hacer inferencia en una imagen satelital de 4 bandas.

Tenemos cuatro archivos binarios correspondientes a cada banda de una imagen satelital, y un conjunto de entrenamiento `rsTrain.dat`, que es tabular separado por espacios.

Requisitos para correr el código del proyecto

Mi versión de python en 3.12.3, en un entorno de pip instalé los siguientes paquetes:

```
numpy matplotlib pandas sklearn
```

los cuales se pueden obtener con `pip install`. En sí el modulo de `knn.py` solo necesita `numpy` y una implementación de la moda estadística, python ya incluye `statistics.mode`; pero algo tan simple como `mean(y) > 0.5` podría haber funcionado también, pues nuestra clasificación binaria está codificada por 0 y 1.

Los directorios extra en la raíz del proyecto no se tienen que eliminar porque la creación de imágenes no crea el directorio si no existe y detiene el programa.

Código fuente

Archivo `main.py`

En `main.py` sucede la validación de modelos y la inferencia de la clasificación de pixeles de la imagen satelital. Hay una sobra de métricas estadísticas porque me entretuve pensando cuál sería la mejor para escoger el mejor modelo, solo unas pocas se imprimen. Al final decidí por reportar el coeficiente de correlación de Matthews porque toma en cuenta ambas clases como igual de importantes, penalizando si solo una tiene buena exactitud.

La impresión de las tablas comparativas es solo funcional y no bonita, haciendo la alineación con tabs al tanteo. Se pudo haber usado diccionarios para crear un dataframe de `pandas`, pero para el desarrollo del proyecto decidí dejarlo así.

Se llenan los directorios vacíos que se mencionaron en la sección de requisitos de este reporte, para la inferencia en cada transformación de datos. “per_char” para normalización por columnas, “global” para normalización global, y original para el modelo con los datos originales.

```
import numpy as np
import pandas as pd
```

```
from matplotlib import pyplot, image
import os

# import de modulos propios
from datasets import X_global_normal, X, X_normal, y
from datasets import normal, bands, train, test
from knn import KNN

def inference(x, train_X, train_y, k_list, prefix):
        knn = KNN(x, train_X, train_y)
        for k in k_list:
                inference = knn.fit(k).reshape((512,512))
                image.imsave(f"{prefix}inferencia_k_{k}.png",
                        inference, cmap='gray')

try:
        for dataset,nombre in zip((X,X_normal,X_global_normal),
                        ("Sin transformar",
                        "Normalizar por columna",
                        "Normalizar todo el dataset")):
                knn_train = KNN(dataset[train], dataset[train], y[train])

                knn = KNN(dataset[test], dataset[train], y[train])
                print(f"Resultado con transformación: {nombre}")
                print("k\t\tTraining Accuracy\tAccuracy\tMCC")
                for k in [3,7,100,150]:
                        y_prediction = knn.fit(k)
                        y_train_prediction = knn_train.fit(k)
                        training_accuracy = np.mean(
                                y_train_prediction == y[train])
                        accuracy = np.mean(y_prediction == y[test])

                        fp = np.sum(
                                y_prediction[y[test] == 0] == 1
                        )
                        tp = np.sum(
                                y_prediction[y[test] == 1] == 1
                        )
                        fn = np.sum(
                                y_prediction[y[test] == 1] == 0
                        )
                        tn = np.sum(
                                y_prediction[y[test] == 0] == 0
                        )

                        precision = tp / (tp + fp) if tp + fp > 0 else 1
                        recall = tp / (tp + fn) if tp + fn > 0 else 1

                        f1_score = 2/(1/precision + 1/recall)\
                                if precision > 0 and recall > 0 else 0
```

```
                        true_positive_rate = tp / (tp + fn)
                        true_negative_rate = tn / (tn + fp)
                        balanced_accuracy =\
                                (true_positive_rate + true_negative_rate)/2

                        numerator = (tp*tn - fp*fn)
                        denominator = np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
                        mcc = numerator/denominator if denominator > 0 else 0

                        print(f"{k}\t\t{training_accuracy}\t\t\t{accuracy}\t\t{mcc}")


        # image reconstruction
        x = np.zeros((4,512*512))
        for i,band in enumerate(bands):
                x[i] = band.reshape((512*512,))

        x = x.T
        x_global_norm = (x - X.min())/(X.max() - X.min())
        x_norm = normal.transform(x)

        for directory in ("original","per_char","global"):
                os.makedirs(directory, exist_ok=True)

        k_list = [3,7,100,150]
        if True: # cambiar a False para solo generar el análisis estadístico
                inference(x, X, y, k_list, "original/")
                inference(x_norm, X_normal, y, k_list, "per_char/")
                inference(x_global_norm, X_global_normal, y, k_list, "global/")

except Exception as e:
        print(e)
```

Archivo `datasets.py`

En `datasets.py` ocurre la lectura de los archivos, limpieza (que no se necesitó según nuestro análisis), su transformación y carga. No se encontraron outliers o valores NA. Todos los valores pudieron ser interpretados como enteros de 8 bits, el objetivo `y` podría haberse guardado como booleano pero por simplicidad todas las columnas son `np.int8`.

Parte del código siguiente se comentó porque en el mismo archivo se probó más código, y además se importa como modulo desde `main.py`. Lo que escribí en el párrafo anterior es todo lo que basta concluir del código comentado.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot, image
from os.path import isfile

bands_filenames = "band1.irs band2.irs band3.irs band4.irs".split(" ")
```

```
bands = list[np.ndarray]() # 0 corresponds to band1
for i,filename in enumerate(bands_filenames):
        bands.append(np.fromfile(filename, dtype=np.int8))
        bands[i] = bands[i].reshape((512,512))
        if not isfile(f"banda{i+1}.png"):
                image.imsave(f"banda{i+1}.png", bands[i], cmap='gray')
        # else:
        #         print("Already saved")

# Load training set in pandas to check
colnames = ["band1", "band2", "band3", "band4", "is_water"]
df = pd.read_csv('rsTrain.dat', sep=r'\s+', names=colnames)

# obvious conversion
# df["is_water"] = df["is_water"].astype(np.int8)
df = df.astype(np.int8)
df.dtypes
len(df)

# comenté este codigo a una función para que no tenga que correr
# mientras hago pruebas
def analysis(df):
        # check for possible conversion of all columns to int8
        df_int8 = df.astype(np.int8)
        # no difference, floating point precision not needed
        np.sum(np.abs(df_int8 - df))
        df = df_int8
        # no NA
        df.isna().sum()

        # no blatant outliers
        df.hist(figsize=(6,7))
        pyplot.savefig('hist.png', dpi=150)

# to numpy
X = df.iloc[:,:4].to_numpy(dtype=np.int8)
y = df.iloc[:,4:5].to_numpy(dtype=np.int8)

# train-test split
import random
random.seed(590)

indexes = list(range(200))
random.shuffle(indexes)

train, test = indexes[:160], indexes[160:200]

y[train].mean() # 0.5
y[test].mean() # 0.5
```

```
# X[train].mean(axis=0) - X[test].mean(axis=0)
# [0.7     0.38125 0.79375 0.7    ]
# X.std(axis=0)
# [3.7018779  3.64540464 5.24956189 7.87437458]

from sklearn.preprocessing import MinMaxScaler
normal = MinMaxScaler()
X_normal = normal.fit_transform(X)

X_global_normal = (X - X.min()) / (X.max() -  X.min())
X_global_normal.max(axis=0)
```

Archivo `knn.py`

Finalmente la parte más importante del proyecto es la implementación de k-Nearest Neighbors en `knn.py`. Se escribió como modulo imitando el funcionamiento de bibliotecas como `sklearn`, en la cual se asigna una instancia de la clase de modelos pasando el conjunto de entrenamiento, y se hace el ajuste con los parámetros deseados.

En el caso de `KNN`, como probamos distintos `k`, hice un cache para guardar las distancias, por eso las instancias de `KNN` toman como argumento también el conjunto al que se va a hacer inferencia, porque el modelo depende de ese conjunto para "entrenar" el modelo. Donde se pudo hacer sin perder la simplicidad, se usó la vectorización de numpy. Se pudo haber manejado mejor el encontrar las distancias y ordenarlas, posiblemente con `np.argpartition`, pero al final esta implementación fue lo suficientemente rápida para mi computadora.

La función `fit` de la clase `KNN` simplemente calcula todas las distancias Manhattan de `X_train` a los renglones de `X_train`, se hace con las operaciones optimizadas de numpy. Subsecuentemente se ordenan con `sorted` enumerando las distancias, después tomando los indices de la numeración para extraer la clase que le corresponde. Al final se toman `k` y se regresa la moda de estos.

Los datos por si solos, observamos, que pueden sesgar los resultados con la distancia Manhattan. Estos modelos son sensibles a que una sola banda domine el resultado si esta tiene mayor contrastes, por eso no se vio necesidad de probar otra distancia como la euclidiana.

```
import numpy as np
from statistics import mode

class KNN:
        '''Warning: throws exception, catch with try'''
        def __init__(self,
                        X: np.ndarray, # inference from this var
                        X_train: np.ndarray,
                        y_train: np.ndarray):
                if len(X_train) == len(y_train):
                        self.nearest = None
                        self.X = X.copy()
                        self.X_train = X_train.copy()
                        self.y_train = y_train.copy()
                else:
                        raise Exception("X debe tener tantos renglones como y")
```

```
'''Returns inference from X with k nearest neighborhoods'''
def fit(self, k: int) -> np.ndarray:
        y = np.zeros((len(self.X),1), dtype=np.int8)
        if not self.nearest: # checar si existe el cache
                self.nearest = list[list[int]]()
                for i,x in enumerate(self.X):
                        distances = np.sum(np.abs(self.X_train - x), axis=1)

                        sorted_distances = sorted(
                                list(enumerate(distances)),
                                key=lambda x: x[1])

                        # obtener indice, e insertar a nearest el valor
                        # de la clase correspondiente
                        self.nearest.append(list[int]())
                        for j,_ in sorted_distances:
                                self.nearest[i].append(
                                        int(self.y_train[j][0]))
                                ######################################
                                # !! corrección !!
                                # la línea anterior tenía antes
                                # sorted_distances[j][0] en vez de j
                                # mezclando mal el orden
                                ######################################

        if self.nearest: # usar cache
                for i in range(len(self.X)):
                        y[i] = mode(self.nearest[i][0:k])

        return y
```

Inspección preliminar de los datos para decidir un análisis

Cargando cada imagen con `numpy` y guardándola como PNG con `matplotlib`, se puede apreciar que cada banda tiene distintos contrastes de la misma imagen en cuanto a nosotros respecta convirtiéndolo a escala de grises. Digo en cuanto nos respecta porque, al provenir de distintos sensores una imagen, es posible que un análisis más preciso pueda darse con esa información. En nuestro caso, tenemos el conjunto de entrenamiento de `rsTrain.dat`. Las primeras cuatro columnas deben corresponder a cada banda respectivamente, la quinta columna es su clase.

De esta forma tenemos la información suficiente para crear un modelo de aprendizaje de máquina. Para comprobar un poco si el modelo va a generalizar, apartamos un 20% del conjunto de entrenamiento para probarlo. Hay "n-fold cross validation" que podríamos hacer fácilmente siguiendo la documentación de alguna librería, pero por mantener el reporte corto lo omitimos.

Split de entrenamiento vs test para validar modelos

Primero intenté dividir el conjunto de entrenamiento apartando los últimos 40 de 200 registros, pero no era representativo en ambas clases el resultado del split, luego con los primeros 40 para test y los restantes para training. Pero la información está distribuida de cierta manera que no podemos hacer eso satisfactoriamente para validar el modelo.

```
# split sencillo sin aleatorizar
X_train, X_test = X[40:200], X[:40]
y_train, y_test = y[40:200], y[:40]

np.mean(y) # 0.5

y[0:100].sum() # 0
y[100:200].sum() # 1
```

Los primeros 100 son todos clase 0 y los últimos son clase 1. Por lo que usamos la librería estándar de python para aleatoriedad.

```
import random
random.seed(590)

indexes = list(range(200))
random.shuffle(indexes)

train, test = indexes[:160], indexes[160:200]

y[train].mean() # 0.5
y[test].mean() # 0.5

# X[train].mean(axis=0) - X[test].mean(axis=0)
# [0.7     0.38125 0.79375 0.7    ]
# X.std(axis=0)
# [3.7018779  3.64540464 5.24956189 7.87437458]
```

Este seed la encontré a prueba y error (con el mismo `randint`), para tener misma proporción de clase 0 y clase 1 en entrenamiento y prueba. Una inspección estadística de la desviación estándar y la diferencia de los promedios de cada columna muestra que este split (individualmente el test y el training set) es representativo del conjunto entero de datos.

Resultados de KNN

Para analizar qué tan bueno es cada modelo muestro la exactitud (accuracy) por clase y la exactitud total. También hago una comparación de las transformaciones que se pueden realizar: Se puede dejar la información en su escala original, se puede normalizar por característica, y también normalizar en todo el conjunto.

```
Resultado con transformación: Sin transformar
k       Training Accuracy   Accuracy    MCC
3       0.98125             0.95        0.9045340337332909
7       0.975               1.0         1.0
100     0.9375              0.95        0.9045340337332909
```

```
150     0.89375             0.875       0.7745966692414834

Resultado con transformación: Normalización global
k       Training Accuracy   Accuracy    MCC
3       0.98125             0.975       0.9511897312113419
7       0.975               1.0         1.0
100     0.9375              0.95        0.9045340337332909
150     0.89375             0.875       0.7745966692414834

Resultado con transformación: Normalizar por columna
k       Training Accuracy   Accuracy    MCC
3       0.975               0.875       0.7745966692414834
7       0.975               0.975       0.9511897312113419
100     0.75                0.75        0.5103103630798288
150     0.84375             0.8         0.6546536707079772
```

Análisis cualitativo en un conjunto no etiquetado

Podemos graficar las cuatro bandas con `matplotlib` y también el resultado de hacer inferencia en estas bandas.

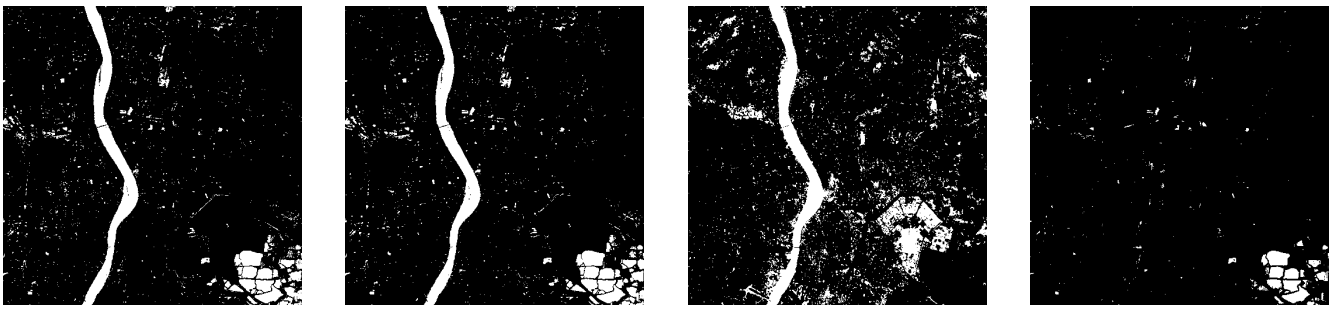
```python
from matplotlib import pyplot, image
from os.path import isfile

bands_filenames = "band1.irs band2.irs band3.irs band4.irs".split(" ")

bands = list[np.ndarray]() # 0 corresponds to band1
for i,filename in enumerate(bands_filenames):
    bands.append(np.fromfile(filename, dtype=np.int8))
    bands[i] = bands[i].reshape((512,512))
    if not isfile(f"banda{i+1}.png"):
        image.imsave(f"banda{i+1}.png", bands[i], cmap='gray')
```

El código anterior guarda las imágenes siguientes.

(a) Banda 1 (b) Banda 2 (c) Banda 3 (d) Banda 4

Figura 1: Visualización de las cuatro bandas en que aplicamos los modelos.

De la normalización por característica (columna) de `sklearn`, `MinMaxScaler`, se infieren las siguientes imágenes con nuestra implementación.

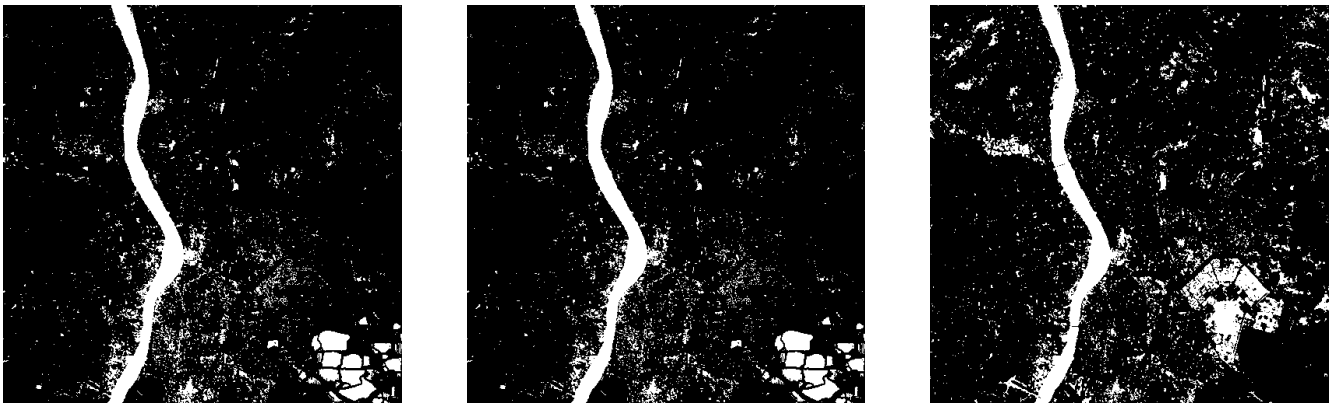


(a) k = 3 (b) k = 7 (c) k = 100 (d) k = 150

Figura 2: Imagen inferida con conjunto de entrenamiento normalizado por columnas por KNN.

k=3 y k=7 se ven bastante similares, k=100 tiene bastante ruido lo cual corresponde con su MCC bajo de 0.51.

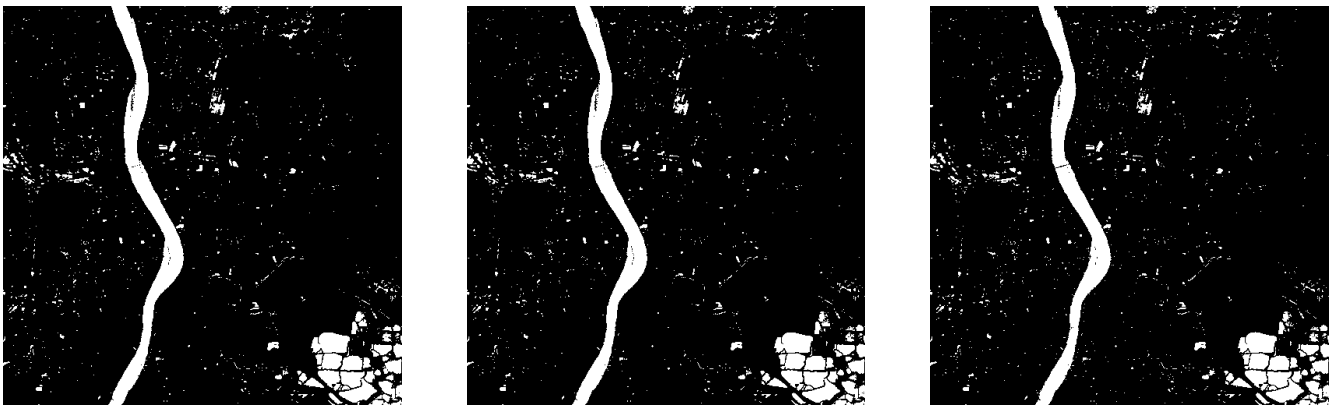
k=150 está claramente "mal", y a pesar de eso tiene exactitud de 80% en validación, esto a pesar de tener mejor MCC que k=100. Es muy sensible a detectar un pixel como Tierra, solo las zonas más extremadamente parecidas a la clase "1" (agua) son marcadas como tal. Esto se podría ver en la matriz de confusión, y podría ser muy útil en otros contextos cuando se quiere ser muy precavido y minimizar falsos positivos.



(a) Dataset original (b) Normalización global (c) Normalización por columna

Figura 3: Modelos 100-Nearest Neighbors.

Ya que k=100 tiene una calificación baja para la normalización por columnas, comparé con las otras transformaciones. Se puede ver que la normalización por columna tiene más ruido. Esto no necesariamente quiere decir que esta transformación es la peor, incluso puede significar que separa mejor los datos con un menor k, lo cuál si podemos evitar sobre-ajuste con un k lo suficientemente bajo, nos da un modelo que es computacionalmente más eficiente. Comprobamos con k=7 en la siguiente figura.



(a) Dataset original (b) Normalización global (c) Normalización por columna

Figura 4: Modelos 7-Nearest Neighbors.

Se ven bastante similar, pero si recordamos la calificación estadística, la normalización por columna sería la peor, ya que el dataset original y la normalización global llegaron al mejor posible MCC de 1. Obviamente puede ser sobr-eajuste al conjunto de entrenamiento, sin embargo al ser cualitativamente tan similar al modelo con una calificación de 0.95 no necesariamente significa que no generalicen igual. Para saber si en verdad en la instancia de `k=7` una transformación es mejor que la otra sería más preciso utilizar algo como "n-fold cross validation".

Conclusión

Cabe destacar que la filosofía que mostré en mi análisis para escoger un modelo es que la transformación de los datos es parte del modelo, como si fuera un parámetro más. En este caso que el número de datos es bajo es posible hacer esa comparación sin mucho costo. Aunque por lo general se sabe heurísticamente qué funciona de antemano para cada problema.

El análisis cualitativo parece concordar mayormente con nuestro análisis estadístico en la validación por el split en conjuntos de entrenamiento y prueba. Sin embargo podría escogerse con mejor precisión el modelo que generalice mejor con "n-fold cross validation" para evitar las idiosincrasias de un solo split.