UNIVERSIDAD AUTÓNOMA DE TLAXCALA

Facultad de Ciencias Básicas Ingeniería y Tecnología

INGENIERÍA EN COMPUTACIÓN

ACTIVIDAD:

Vecinos más cercanos

PRESENTA:

DOCENTE:

M.C. Jorge Arturo Flores López

ALUMNO:

Christopher Rojano Jimenez

SEMESTRE Y GRUPO:

8 “B”

Correo:

20191414@uatx.mx

Apizaco, Tlaxcala, abril 2023

Introducción

K-vecinos más cercanos (k-NN) es un algoritmo de aprendizaje supervisado utilizado en problemas de clasificación y regresión. Es uno de los métodos más simples y efectivos de aprendizaje automático, y se basa en el concepto de encontrar los k ejemplos más cercanos en el conjunto de datos de entrenamiento a un nuevo ejemplo que se quiere clasificar o predecir.

En este proyecto, exploraremos el uso de k-NN para la clasificación de datos. La tarea de clasificación es una de las aplicaciones más comunes de la ciencia de datos, y se utiliza para identificar a qué categoría o clase pertenece un conjunto de datos dado. Por ejemplo, se podría utilizar k-NN para clasificar imágenes de animales en diferentes categorías, o para clasificar clientes según su comportamiento de compra.

Durante este proyecto, se utilizará Matlab para aplicar k-NN, Wk k-NN y Fuzzy k-nn en un conjunto de datos. Además, exploraremos cómo seleccionar el mejor valor de k, la cantidad de vecinos más cercanos a considerar, y cómo preprocesar los datos para mejorar la precisión de las predicciones al cambiar el número de vecindarios a tomar en cuenta.

En resumen, k-vecinos más cercanos es una herramienta valiosa para cualquier persona interesada en el aprendizaje automático y la ciencia de datos. A través de su aplicación, podremos comprender mejor el funcionamiento del algoritmo y su impacto en la clasificación de datos. Además, estaremos en una mejor posición para aplicar esta técnica en futuros proyectos y problemas de clasificación.

Antecedentes

El algoritmo k-vecinos más cercanos (k-NN) fue propuesto por primera vez en 1967 por los científicos Lee A. Galt y J.A. Heath en el contexto de la clasificación de patrones. La idea básica detrás del algoritmo es clasificar un nuevo ejemplo en función de las etiquetas de clase de los k ejemplos más cercanos en el conjunto de entrenamiento. A medida que se han ido desarrollando nuevas técnicas de aprendizaje automático, el algoritmo k-NN ha sido modificado y mejorado con variantes como wk k-nn y fuzzy knn.

La variante wk k-nn, propuesta por Kincaid en 1994, utiliza una ponderación para los k vecinos más cercanos basada en la distancia. En lugar de simplemente considerar los k vecinos más cercanos para la clasificación, la distancia a cada uno de los k vecinos se utiliza como un peso para la asignación de etiquetas de clase. Los vecinos más cercanos tendrán un mayor peso, mientras que los vecinos más alejados tendrán un peso menor. Esta técnica puede mejorar la precisión de la clasificación en algunos casos.

La variante fuzzy k-NN, propuesta por Bezdek en 1981, permite la asignación de un objeto a múltiples clases. En lugar de simplemente clasificar un objeto en una única clase, la asignación de etiquetas de clase se realiza en términos de grados de pertenencia a cada una de las posibles clases. De esta forma, se pueden manejar situaciones en las que un objeto no es claramente de una única clase, sino que puede pertenecer a varias clases con diferentes grados de confianza. Esta técnica ha sido utilizada en aplicaciones como la clasificación de imágenes y el reconocimiento de voz.

Metodología

Para el desarrollo de este proyecto se utilizó un conjunto de 360,177 puntos los cuales, por las características supervisadas del algoritmo, poseen una etiqueta. Nota, de estos se utilizaron solamente 144,071 (el 40%) debido a limitaciones de hardware.

Para encontrar lo valores óptimos se optó por utilizar dos conjuntos, entrenamiento y validación. La elección de los porcentajes óptimos para los conjuntos de validación y prueba en el entrenamiento de un modelo de clasificación depende en gran medida de la naturaleza y el tamaño del conjunto de datos y de los objetivos del proyecto de aprendizaje automático. Sin embargo, debido a experiencia en otros proyectos con algoritmos de clasificación (como lo fue en uno de imágenes), lo valores por los que se opto fueron 30% para entrenamiento y 15% para validación.

Ya definidos los conjuntos y su cantidad de elementos respecto a la muestra, se utilizó el conjunto de entrenamiento para encontrar el valor optimo de k-vecinos para cada método, en este caso se tomó aquel valor más pequeño de k con el menor porcentaje de falla en cada variante, siendo estas k-NN, wK-NN y Fuzzy k-NN.

```
Para k=3 se tuvo un 0.10 % de falla para k-NN, un 0.10 % de falla para MK-NN, un 0.10 % de falla para Fuzzy NN
Para k=5 se tuvo un 0.13 % de falla para k-NN, un 0.11 % de falla para MK-NN, un 0.13 % de falla para Fuzzy NN
Para k=7 se tuvo un 0.15 % de falla para k-NN, un 0.10 % de falla para MK-NN, un 0.15 % de falla para Fuzzy NN
Para k=9 se tuvo un 0.16 % de falla para k-NN, un 0.11 % de falla para MK-NN, un 0.16 % de falla para Fuzzy NN
Para k=11 se tuvo un 0.22 % de falla para k-NN, un 0.12 % de falla para MK-NN, un 0.22 % de falla para Fuzzy NN
Para k=13 se tuvo un 0.29 % de falla para k-NN, un 0.12 % de falla para MK-NN, un 0.29 % de falla para Fuzzy NN
Para k=15 se tuvo un 0.34 % de falla para k-NN, un 0.12 % de falla para MK-NN, un 0.34 % de falla para Fuzzy NN
Para k=17 se tuvo un 0.43 % de falla para k-NN, un 0.13 % de falla para MK-NN, un 0.43 % de falla para Fuzzy NN
Para k=19 se tuvo un 0.48 % de falla para k-NN, un 0.13 % de falla para MK-NN, un 0.48 % de falla para Fuzzy NN
Para k=21 se tuvo un 0.55 % de falla para k-NN, un 0.14 % de falla para MK-NN, un 0.55 % de falla para Fuzzy NN
Para k=23 se tuvo un 0.67 % de falla para k-NN, un 0.15 % de falla para MK-NN, un 0.67 % de falla para Fuzzy NN
Para k=25 se tuvo un 0.89 % de falla para k-NN, un 0.16 % de falla para MK-NN, un 0.89 % de falla para Fuzzy NN
Para k=27 se tuvo un 1.02 % de falla para k-NN, un 0.17 % de falla para MK-NN, un 1.02 % de falla para Fuzzy NN
Para k=29 se tuvo un 1.21 % de falla para k-NN, un 0.17 % de falla para MK-NN, un 1.21 % de falla para Fuzzy NN
Para k=31 se tuvo un 1.37 % de falla para k-NN, un 0.19 % de falla para MK-NN, un 1.37 % de falla para Fuzzy NN
Para k=33 se tuvo un 1.56 % de falla para k-NN, un 0.20 % de falla para MK-NN, un 1.56 % de falla para Fuzzy NN
```

Figura 1. Porcentaje de falla para cada k(3,5,…,33)

Como se puede observar en la figura 1, se comprobó el porcentaje de falla de distintos valores de k, empezando con 3 y terminando con 33, para hallar el que diera el menor porcentaje de falla al clasificar. Cabe aclarar que solo se tomaron valores impares de k para evitar empates entre clases a la hora de etiquetar un punto. Los valores que se decidieron finalmente fueron:

- k-NN: 3
- Wk-NN: 7
- Fuzzy k-NN: 3

La siguiente fase consistía en utilizar MDC (clasificador de distancia mínima), este método consiste en reducir el número de “vecindarios” de nuestros datos para evitar que la carga de computo se vuelva pesada al tratar con muestras de datos muy grandes.

```
Se tuvo un 99.88 % de exito para k-NN
Se tuvo un 99.96 % de exito para Wk-NN
Se tuvo un 99.96 % de exito para Fuzzy NN
Elapsed time is 6300.439849 seconds.
```

a. Clasificación con 122,459 vecindarios

```
Para MDC se tuvo un exito del 56.464001 % con 469 vecindarios en KNN
Para MDC se tuvo un exito del 79.159726 % con 469 vecindarios en WKNN
Para MDC se tuvo un exito del 56.464001 % con 469 vecindarios en Fuzzy KNN
Elapsed time is 31.704834 seconds.
```

b. Clasificación con 469 vecindarios

```
Para MDC se tuvo un exito del 29.937072 % con 20 vecindarios en KNN
Para MDC se tuvo un exito del 36.516750 % con 20 vecindarios en WKNN
Para MDC se tuvo un exito del 29.937072 % con 20 vecindarios en Fuzzy KNN
Elapsed time is 7.942776 seconds.
```

c. Clasificación con 20 vecindarios

Figura 2. Clasificación con distintos tamaños de vecindario

La eficacia de utilizar la técnica de MDC puede observarse en la figura 2, aquí podemos notar como con todos los puntos del conjunto de validación se obtiene un 99% de éxito en un tiempo de casi 2 horas, mientras que para 469 vecindarios (el número de clases del conjunto) toma solo 30 segundos el clasificar los más de 100,000 datos pero sacrificando un gran porcentaje de éxito, por último, en un caso extremo de solo 20 vecindarios se toma casi 8 segundos pero teniendo un tasa muy baja de éxito. Cabe aclarar que para determinar los nuevos vecindarios se tomo la clase que ya tenían y con la operación de modulo se les asigno una nueva en el rango establecido (20 en este ejemplo).

Código y resultados

Entrenamiento

En la figura 3 se muestra la primera parte del código, en esta parte se generan los conjuntos de entrenamiento y validación, así como los puntos sin los valores de dichos conjuntos.

```
%Generando puntos para entrenamiento y validación
pTrain = 0.30;
pVal = 0.15;
idx = randperm(m);
TrainingPoints   = data(idx(1:round(pTrain*m)),:);
ValidationPoints = data(idx(round((pTrain)*m)+1:(round((pTrain)*m)+1) + (round(pVal*m))),:);

%Generando conjunto sin los punto de entranamiento y validación
trainingM = data;
trainingM( idx(1:round(pTrain*m)),:) = [];

validationM = data;
validationM( idx(round((pTrain)*m)+1:(round((pTrain)*m)+1) + (round(pVal*m))),:) = [];
```

Figura 3. Preparación de los datos

El primar paso de este proyecto fue entrenar las variantes de k-vecinos para hallar el valor optimo de k para cada una de ellas. En la figura 4 se puede ver el código de la primera parte que consiste en calcular las distancias de un punto P a todos los puntos que no pertenecen al conjunto de entrenamiento. A su vez, para calcular dichas distancias se obtiene de una función de autoría propia que calcula la distancia euclidiana de un punto P dado hacia una matriz de puntos y devuelve dichas distancias ordenadas de menor a mayor.

```
P = TrainingPoints(i,:);
sortedDists = getSortedDist(b,P,trainingM);
```

a. Generación de las distancias hacia un punto

```
function [sortedDists] = getSortedDist(b,P,points)
    % Calcular distancias
    dist = zeros(b,5);
    for iP=1: b
      dist(iP,1) = points(iP,1,1);
      dist(iP,2) = points(iP,2,1);
      dist(iP,3) = points(iP,3,1);

      dist(iP,4) = sqrt( power(points(iP,1,1)-P(1),2) + power(points(iP,2,1)-P(2),2) );
      dist(iP,5) = iP;
    end

    % Ordenar distancias
    sortedDists = sortrows(dist,4);
end
```

b. Función genera las distancias hacia un punto P dado

Figura 4. Obtención de las distancias

En la figura 5 se observa como se invocan las funciones de las variantes de los k-vecinos y se valida si la clase que se calculo en dichas funciones corresponde con la que ya se tiene en el conjunto.

```
for j=3:2:maxKn
    knn = sortedDists(1:j,:);

    resp = KNN(knn);
    if resp{2}==P(3)
        success(1,k) = success(1,k)+1;
    end

    resp = WKNN(knn, j, nClass);
    if resp{2}==P(3)
        success(2,k) = success(2,k)+1;
    end

    resp = FUZZYKNN(knn, j, nClass, mu);
    if resp{2}==P(3)
        success(3,k) = success(3,k)+1;
    end

    k = k+1;
end
```

Figura 5. Comprobación de las variantes de k-vecinos

```
function [resp] = KNN(KNN)
claseP = mode(KNN(:, (3) ) );

resp{1} = KNN;
resp{2} = claseP(1);
```

a. Variante k-NN

```
function [resp] = WKNN(KNN, kn, a)
weigths = zeros(kn,1);
weightClass = zeros(a,1);
for i=1: kn
    if KNN(kn,4) == KNN(1,4)
        wei = 1;
    else
        wei = (KNN(kn,4)-KNN(i,4)) / ((KNN(kn,4)-KNN(1,4))+eps);
    end

    weigths(i) = wei;

    weightClass(KNN(i,3)) = weightClass(KNN(i,3))+wei;
end
classMax = find(weightClass==max(weightClass));


resp{1} = KNN;
resp{2} = classMax(1);
resp{3} = weigths;
```

b. Variante Wk-NN

```
function [resp] = FUZZYKNN(KNN, kn, a, m)
equi = zeros(1, a);
denominadorF = 0;
for i=1:kn
    denominadorF = denominadorF + ( 1 / ( power(KNN(i,4), 2/m-1) ) +eps);
    equi(KNN(i,3)) = equi(KNN(i,3)) + ( 1 / ( power(KNN(i,4), 2/m-1) ) +eps);
end

for i=1:a
    equi(i) = equi(i) / denominadorF;
end

classFin = find(equi == max(equi));
resp{1} = KNN;
resp{2} = classFin(1);
resp{3} = equi;
```

c. Variante Fuzzy k-NN

Figura 6. Funciones para calcular las variantes de k-vecinos

La figura 6a muestra la variante k-NN, en esta se toman los k-vecinos más cercanos al punto y se le asigna la clase que sea mayoría entre estos k-vecinos.

La figura 6b muestra la variante Wk-NN, esta le asigna un “peso” a cada clase en función de las distancias de los k-vecinos más cercanos y se le asigna al punto aquella clase con el mayor peso.

La figura 6c muestra la variante Fuzzy k-NN, en esta variante se calcula una pertenencia a cada clase tomando en cuenta las distancias y asi le asigna al punto aquella clase con mayor pertenencia.

Finalmente, en la figura 7 se muestra como se calcula finalmente el porcentaje de falla de cada variante para todos los valores de k probados, después simplemente se toma el mejor-menor valor de k para seleccionarlo como el valor optimo.

```
percentFail = 100-((success/pointsTR)*100);
```

Figura 7. Calculo de fallas en la clasificación

Validación

La siguiente fase del proyecto consiste en validar si los valores óptimos para k obtienen resultados apropiados para la clasificación. El proceso de validación es similar al de entrenamiento, solo que aquí se usa solamente el valor de k que se encontró optimo para cada variante. La figura 8 muestra dicho proceso

```
for i=1:pointsVL
    P = ValidationPoints(i,:);
    sortedDists = getSortedDist(b,P,validationM);

    knn = sortedDists(1:kOpti(1),:);
    resp = KNN(knn);
    if resp{2}==P(3)
        successV(1) = successV(1)+1;
    end
    P(3) = resp{2};
    finalKNN(i,:) = P;

    knn = sortedDists(1:kOpti(2),:);
    resp = WKNN(knn, kOpti(2), nClass);
    if resp{2}==P(3)
        successV(2) = successV(2)+1;
    end
    P(3) = resp{2};
    finalWKNN(i,:) = P;

    knn = sortedDists(1:kOpti(3),:);
    resp = FUZZYKNN(knn, kOpti(3), nClass, mu);
    if resp{2}==P(3)
        successV(3) = successV(3)+1;
    end
    P(3) = resp{2};
    finalFuzzy(i,:) = P;
end
```

Figura 8. Validación

Los resultados que se obtuvieron al realizar la validación fueron:

- Se tuvo un 99.88 % de exito para k-NN
- Se tuvo un 99.96 % de exito para Wk-NN
- Se tuvo un 99.96 % de exito para Fuzzy k-NN

Esto muestra que con 3 vecinos para k-NN y Fuzzy k-NN y 7 para Wk-NN se obtiene un porcentaje bastante alto de éxito al clasificar este tipo de datos.

En la figura 9 podemos ver la gráfica resultante de los puntos del conjunto de validación

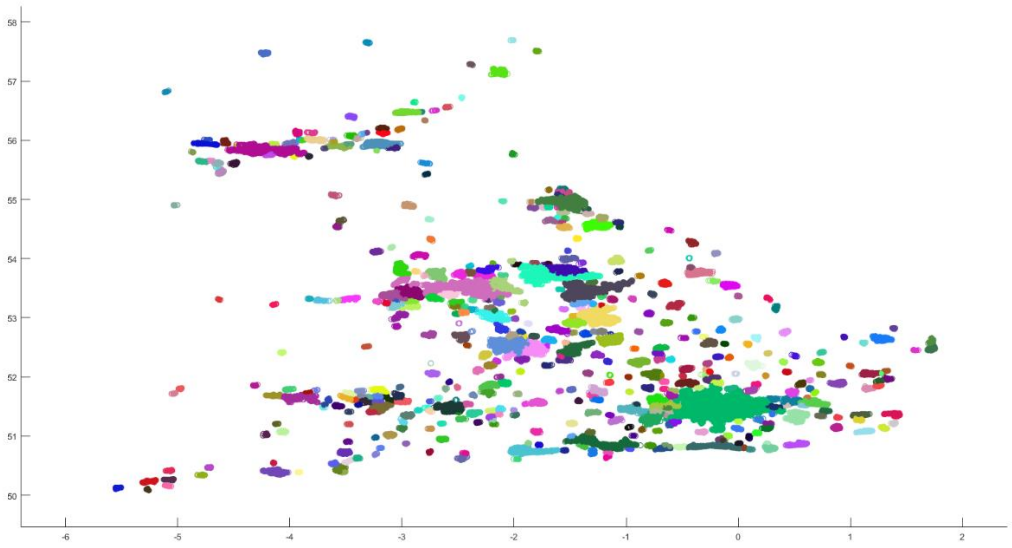
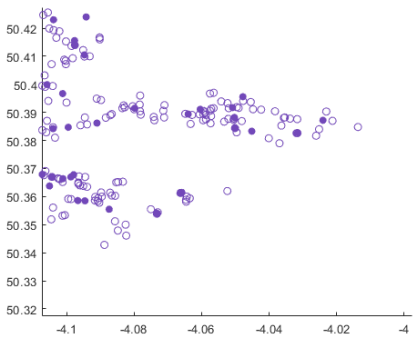
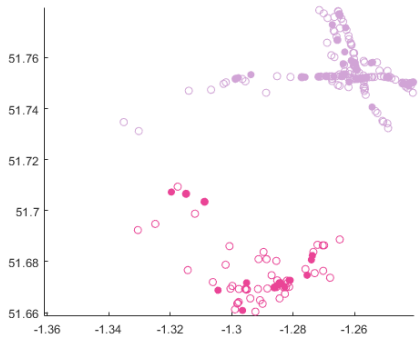


Figura 9. Conjunto de validación

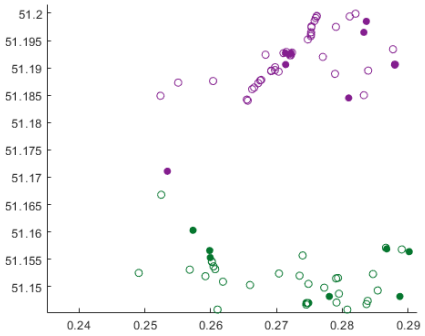
En la figura 10 se pueden observar ejemplos más detallados de como se les asignó una clase a los puntos del conjunto de validación (círculos rellenos) y como coinciden con la clase de sus vecinos (círculos huecos).



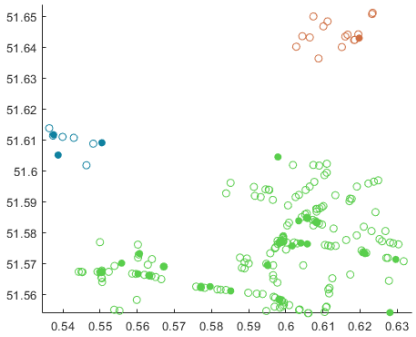
a. Ejemplo de clasificación



b. Ejemplo de clasificación



c. Ejemplo de clasificación



d. Ejemplo de clasificación

Figura 10. Ejemplos de clasificación

Clasificador de distancia mínima (MDC)

Otro algoritmo que se tomo en cuenta fue el de selección de prototipos conocido como Clasificador de distancia mínima (MDC), este consiste en cambiar el numero de vecindarios a uno más pequeño a fin de reducir el tiempo que le toma al algoritmo asignarles una clase a todos los puntos.

Para calcular los nuevos vecindarios y sus puntos clasificados se hace uso de una función de autoría propia, en la figura 11 se muestra el funcionamiento de dicha función:

```
function [resp] = MDC(nNeig, M, nSamples, Points, nPoints, kOpti, mu)
    nS = zeros(1,nNeig);
    sumP = zeros(nNeig,3);
    colorsMDC = zeros(nNeig,3);
    for i=1:nSamples
        positionNew = mod(M(i,3)-1,nNeig)+1;
        sumP(positionNew,1) = sumP(positionNew,1)+M(i,1);
        sumP(positionNew,2) = sumP(positionNew,2)+M(i,2);
        sumP(positionNew,3) = positionNew;

        nS(positionNew) = nS(positionNew)+1;
    end

    for i=1:nNeig
       sumP(i,1) = sumP(i,1)/nS(i);
       sumP(i,2) = sumP(i,2)/nS(i);

       colorsMDC(i,:) = getColors(sumP(i,3));
    end
```

Figura 11. Función MDC

Esta función inicialmente calcula la suma de todos puntos que pertenecen a una clase. La nueva clase que se genera se calcula con el módulo más uno de la clase anterior menos uno respecto a la cantidad de vecindarios a generar. El menos y más uno se utilizan para que los valores que genere este calculo siempre den un numero entre 1 y el numero de vecindarios a generar, esto se realizó así ya que en Matlab los índices empiezan en 1 y no en 0 como otros lenguajes.

Una vez se obtiene la suma de los puntos de cada clase se genera el promedio obteniendo así un punto central que representa la clase entera, a este nuevo punto se le conoce como “centroide”.

Finalmente se calculan las distancias de los puntos con la función mostrada en la figura 4b (en este caso se utilizaron los puntos que se tenían para la validación), se aplican las tres variantes de k-vecinos con las funciones previamente detalladas y se calcula el porcentaje de éxito en cada una de ellas. En la figura 12 se muestra el código que genera lo anterior:

```
for i=1: nPoints
    P = Points(i,:);
    sortedDists = getSortedDist(nNeig,P,sumP);
    newClass = mod(P(3)-1,nNeig)+1;

    knn = sortedDists(1:kOpti(1),:);
    resp = KNN(knn);
    if mod(resp{2}-1,nNeig)+1==newClass
        successV(1) = successV(1)+1;
    end
    P(3) = mod(resp{2}-1,nNeig)+1;
    mdcPointsKN(i,:) = P;

    knn = sortedDists(1:kOpti(2),:);
    resp = WKNN(knn, kOpti(2), nNeig);
    if mod(resp{2}-1,nNeig)+1==newClass
        successV(2) = successV(2)+1;
    end
    P(3) = mod(resp{2}-1,nNeig)+1;
    mdcPointsWK(i,:) = P;

    knn = sortedDists(1:kOpti(3),:);
    resp = FUZZYKNN(knn, kOpti(3), nNeig, mu);
    if mod(resp{2}-1,nNeig)+1==newClass
        successV(3) = successV(3)+1;
    end
    P(3) = mod(resp{2}-1,nNeig)+1;
    mdcPointsFuzzy(i,:) = P;
end
```

Figura 12. Proceso de clasificación y validación de los puntos

En la figura 13 se muestra el resultado obtenido de la clasificación con 469 vecindarios:

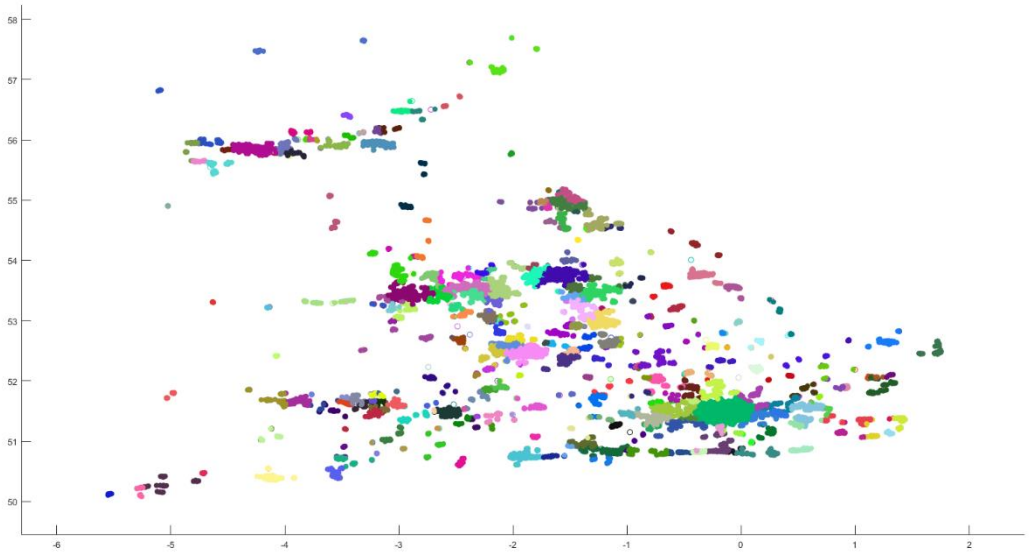
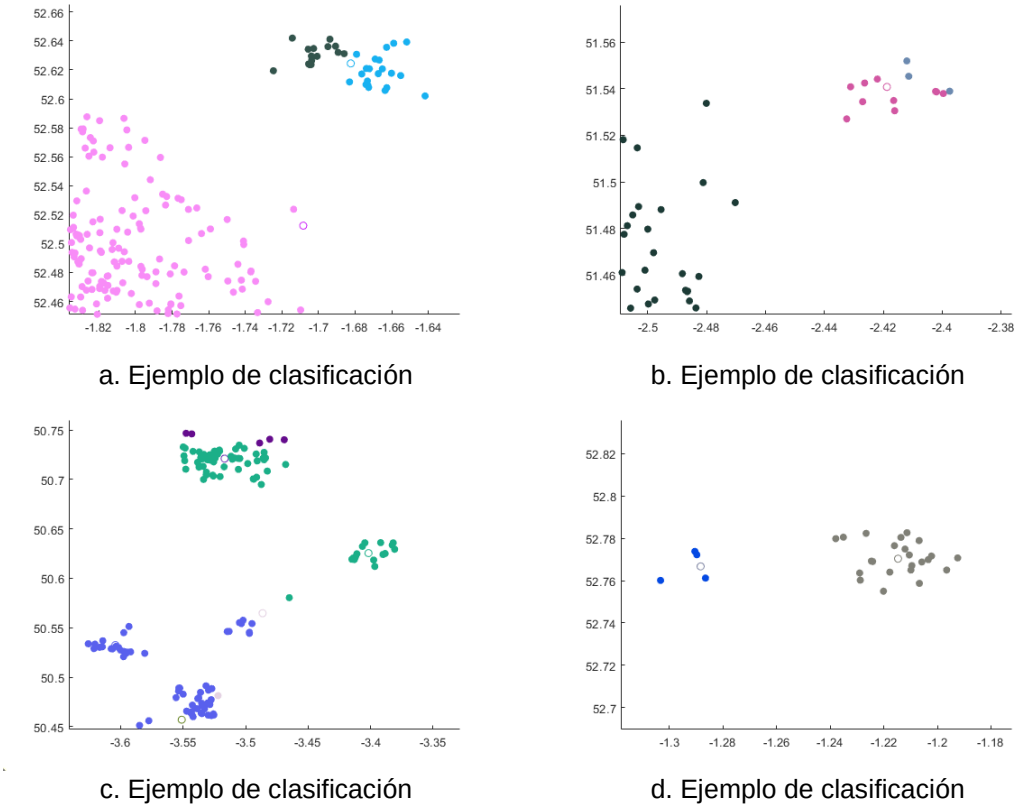


Figura 13. Clasificación con 469 vecindarios

En la figura 14 tenemos distintos ejemplos de como se asignaron las clases a los puntos (círculos rellenos) respecto a sus vecinos (círculos vacíos).



a. Ejemplo de clasificación

b. Ejemplo de clasificación

c. Ejemplo de clasificación

d. Ejemplo de clasificación

Figura 14. Ejemplos de clasificación con 469 vecindarios

En la figura 15 se muestra el resultado obtenido de la clasificación con 20 vecindarios:

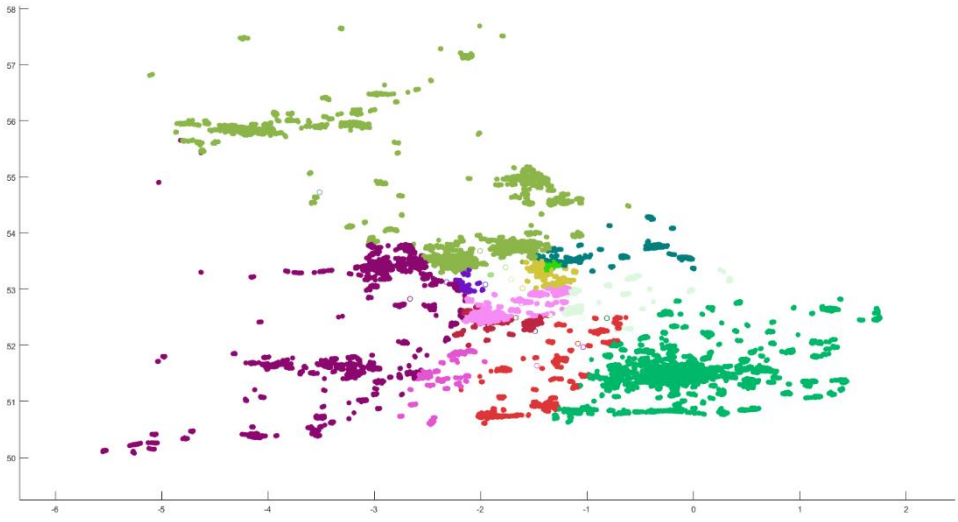
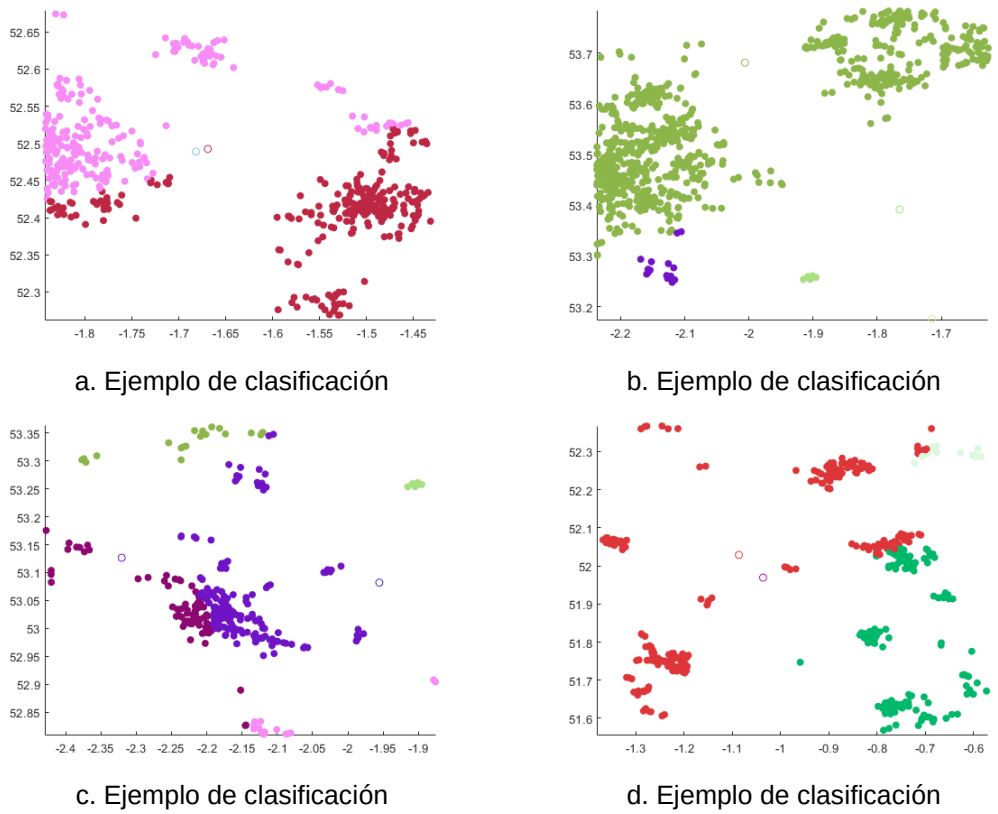


Figura 15. Clasificación con 20 vecindarios

En la figura 16 tenemos distintos ejemplos de cómo se asignaron las clases a los puntos (círculos rellenos) respecto a sus vecinos (círculos vacíos).



a. Ejemplo de clasificación

b. Ejemplo de clasificación

c. Ejemplo de clasificación

d. Ejemplo de clasificación

Figura 16. Ejemplos de clasificación con 20 vecindarios

Conclusiones

Trabajar con este algoritmo más de cerca nos da una visión más amplia de como se comportan sus distintas variantes. En este proyecto se obtuvo que valores pequeños para k suelen ser adecuados al momento de aplicar cualquiera de las variantes de k-vecinos. Además, el uso de técnicas de selección de prototipos, como fue mdc, puede reducir ampliamente el tiempo de ejecución de nuestro programa, sin embargo esto puede traer un impacto negativo en la efectividad al momento de clasificar, por lo que es necesario encontrar un balance entre ambas técnicas.

En general, el algoritmo k-vecinos más cercanos ha evolucionado con el tiempo y ha sido mejorado con variantes como las abordadas en este proyecto. Estas técnicas ofrecen diferentes enfoques para mejorar la precisión de la clasificación y permiten la asignación de clases. El k-NN sigue siendo una herramienta importante en el campo de la clasificación de patrones y el aprendizaje automático.