UNIVERSIDAD DE SONORA

"Clasificación de Tiros en la NBA"

Licenciatura en Ciencias de la Computación

Cesar Ernesto Salazar Buelna

Reconocimiento de Patrones

Director del proyecto: Dr. Ramón Soto de la Cruz

diciembre de 2017

Contenido

1. Introducción

El Reconocimiento de Patrones (RP) se ha convertido en una de las líneas de investigación más atractiva. El RP es la ciencia que se encarga de la descripción y clasificación de objetos, personas, señales, etc. El margen de aplicaciones del RP es muy amplio.

El esquema general de un sistema se RP cuenta con varias etapas, un sensor, la extracción de características y un clasificador, que es donde el sistema toma decisiones para asignar los patrones de clase desconocida a la categoría apropiada.

En muchos ámbitos se puede usar RP, reconocimiento de voz, de imágenes, canciones, ¿tiros de baloncesto?... Saber si un tiro entro o no por sus características como distancia, reloj de tiro, el numero de dribles que realizo el jugador, entre otros. Es interesante y útil conocer cuáles son las características de un tiro encestado.

Con el fin de identificar mejor las características que influyen en un tiro, se analizará, limpiara y aplicaran distintos algoritmos de clustering y clasificación a un conjunto de datos sobre tiros del 2015 en la liga de baloncesto estadounidense, la NBA.

2. Descripción de los datos

Los datos fueron tomados de: https://www.kaggle.com/dansbecker/nba-shot-logs

Son datos sobre los tiros realizados durante la temporada 2014-2015 de la NBA (National Basketball Association), liga de baloncesto estadounidense. Tomados de la API de la NBA.

La fuente de datos consiste en 128069 renglones con 21 columnas de información. Para este proyecto se tomaron solo 8 columnas numéricas:

a) SHOT_NUMBER: el número de tiro del jugador en el partido.

b) PERIOD: el periodo en el que se lanzó.

c) SHOT_CLOCK: el segundo en el reloj de tiro al momento del lanzamiento.

d) DRIBBLES: la cantidad de dribles que realizo el jugador antes de tirar.

e) TOUCH_TIME: el tiempo que el jugador toca el balón antes del tiro.

f) SHOT_DIST: la distancia del tiro en pies.

g) CLOSE_DEF_DIST: la distancia del defensor al hacer el tiro.

h) FMG: 1 si el tiro entro y 0 si fue fallado.

En la siguiente imagen se muestra la información del archivo usado y también los tipos asociados a cada columna.

```
RangeIndex: 128069 entries, 0 to 128068
Data columns (total 8 columns):
SHOT_NUMBER       128069 non-null int64
PERIOD            128069 non-null int64
SHOT_CLOCK        122502 non-null float64
DRIBBLES          128069 non-null int64
TOUCH_TIME        128069 non-null float64
SHOT_DIST         128069 non-null float64
CLOSE_DEF_DIST    128069 non-null float64
SHOT_RESULT       128069 non-null int64
dtypes: float64(4), int64(4)
memory usage: 7.8 MB
```

3. Limpieza de datos

En el conjunto de datos se encontraron problemas en dos de las columnas seleccionadas, en la de SHOT_CLOCK y en la de TOUCH_TIME.

a) Valores Inconsistentes

	SHOT_NUMBER	PERIOD	SHOT_CLOCK	DRIBBLES
count	128069.000000	128069.000000	122502.000000	128069.000000
mean	6.506899	2.469427	12.453344	2.023355
std	4.713260	1.139919	5.763265	3.477760
min	1.000000	1.000000	0.000000	0.000000
25%	3.000000	1.000000	8.200000	0.000000
50%	5.000000	2.000000	12.300000	1.000000
75%	9.000000	3.000000	16.675000	2.000000
max	38.000000	7.000000	24.000000	32.000000

	TOUCH_TIME	SHOT_DIST	CLOSE_DEF_DIST	SHOT_RESULT
count	128069.000000	128069.000000	128069.000000	128069.000000
mean	2.765901	13.571504	4.123015	0.452139
std	3.043682	8.888964	2.756446	0.497706
min	-163.600000	0.000000	0.000000	0.000000
25%	0.900000	4.700000	2.300000	0.000000
50%	1.600000	13.700000	3.700000	0.000000
75%	3.700000	22.500000	5.300000	1.000000
max	24.900000	47.200000	53.200000	1.000000

Como se puede ver en la imagen anterior, en SHOT_CLOCK el valor mínimo en los datos es de -163.6 segundos, lo cual es imposible ya que es el tiempo que el jugador toco el balón antes del tiro, el cual no puede ser de 0 o menos.

Como el porcentaje de valores de 0 o menores es muy bajo, se utilizó la estrategia de descartarlos. Si el valor era 0 o menor, se remplazó por NaN para su posterior eliminación.

El resto de las columnas no tenía valores con falta de coherencia como lo anterior.

b) Valores Faltantes

Para verificar los valores faltantes se imprimió la información de cada columna con los siguientes resultados:

```
SHOT_NUMBER       False
PERIOD            False
SHOT_CLOCK         True
DRIBBLES          False
TOUCH_TIME        False
SHOT_DIST         False
CLOSE_DEF_DIST    False
SHOT_RESULT       False
```

Como se puede ver en la única columna en los que hay valores faltantes es en la de SHOT_CLOCK.

```
Porcentaje de datos nulos en la columna SHOT_CLOCK 4.34687551242 %
```

En términos generales, se suelen considerar los siguientes grados de impacto, dependiendo del porcentaje de valores faltantes (*dumb rules*):

- Menos de 1%: Trivial (no relevante)
- 1-5%: Manejable
- 5-15%: Manejable mediante métodos sofisticados
- Más de 15%: Crítico, con impacto severo en cualquier tipo de interpretación

Como en la columna SHOT_CLOCK el porcentaje faltante es de 4.34… el impacto es manejable, por tal razón la decisión fue descartar esos renglones donde hubiera valores faltantes.

Ya teniendo los valores inconsistentes como NaN y los valores faltantes, se aplicó el descarte de estos valores quedando los siguientes resultados:

	SHOT_NUMBER	PERIOD	SHOT_CLOCK	DRIBBLES
count	119386.000000	119386.000000	119386.00000	119386.000000
mean	6.471027	2.471111	12.32351	2.041094
std	4.682672	1.138258	5.66551	3.440073
min	1.000000	1.000000	0.00000	0.000000
25%	3.000000	1.000000	8.10000	0.000000
50%	5.000000	2.000000	12.20000	1.000000
75%	9.000000	3.000000	16.40000	3.000000
max	37.000000	7.000000	24.00000	32.000000

	TOUCH_TIME	SHOT_DIST	CLOSE_DEF_DIST	SHOT_RESULT
count	119386.000000	119386.000000	119386.000000	119386.000000
mean	2.824904	13.614446	4.145090	0.455841
std	2.938674	8.738203	2.741207	0.498048
min	0.100000	0.000000	0.000000	0.000000
25%	0.900000	4.900000	2.300000	0.000000
50%	1.700000	13.900000	3.700000	0.000000
75%	3.800000	22.500000	5.300000	1.000000
max	24.900000	43.300000	53.200000	1.000000

Como se ve, ya no hay valores “raros” en la columna SHOT_CLOCK.

```
SHOT_NUMBER       False
PERIOD            False
SHOT_CLOCK        False
DRIBBLES          False
TOUCH_TIME        False
SHOT_DIST         False
CLOSE_DEF_DIST    False
SHOT_RESULT       False
dtype: bool
Porcentaje de datos nulos en la columna SHOT_CLOCK 0.0 %
```

Y no hay valores faltantes en ninguna columna después del descarte.

El código asociado a la sección de limpieza de datos se puede visualizar en archivo limpieza.py.

4. Clustering

El clustering consiste en agrupar objetos en grupos de tal manera que los objetos pertenecientes a un grupo (o "*clister*") son más semejantes entre sí que a otros objetos no pertenecientes al grupo.

a) Dendogramas

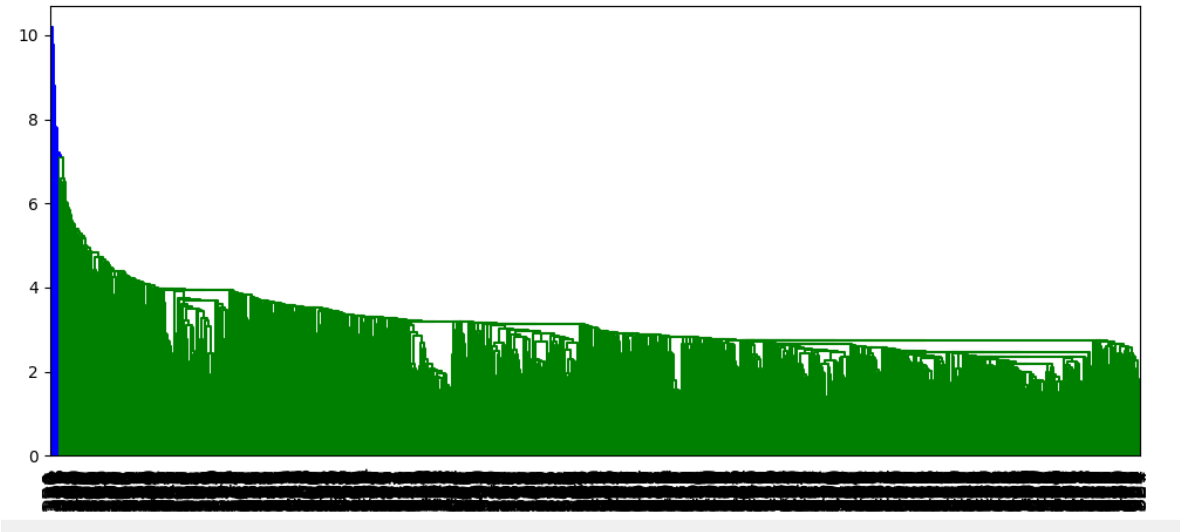
Un dendrograma es un tipo de representación gráfica o diagrama de datos en forma de árbol que organiza los datos en subcategorías que se van dividiendo en otros hasta llegar al nivel de detalle deseado. Este tipo de representación permite apreciar claramente las relaciones de agrupación entre los datos e incluso entre grupos de ellos, aunque no las relaciones de similitud o cercanía entre categorías.

Viendo las divisiones en las gráficas se puede hacer una idea de los grupos presentes en una muestra de datos.

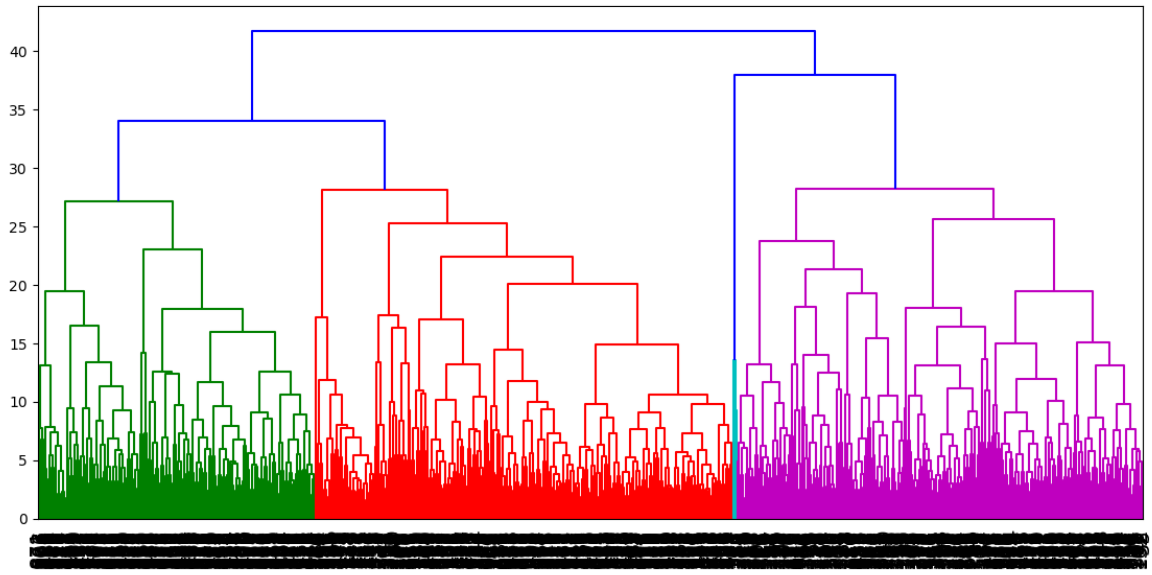
Consideremos un conjunto de datos formado por los primeros 1000 elementos de nuestros datos, ya limpios.

A continuación, se presenta el dendrograma para los 1000 datos de muestra que tomamos de los datos originales:

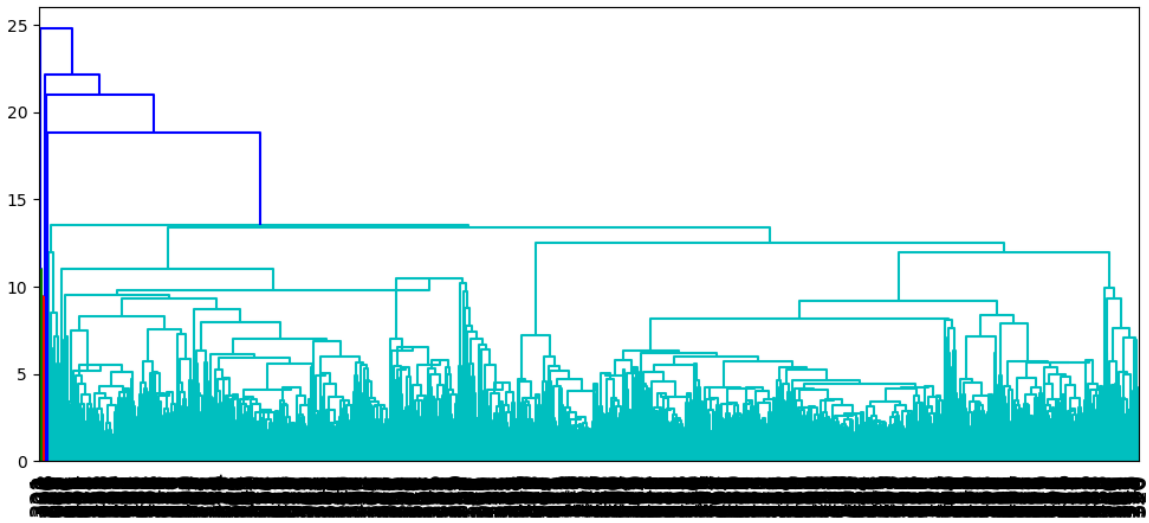
- Método Single



- Método Complete:

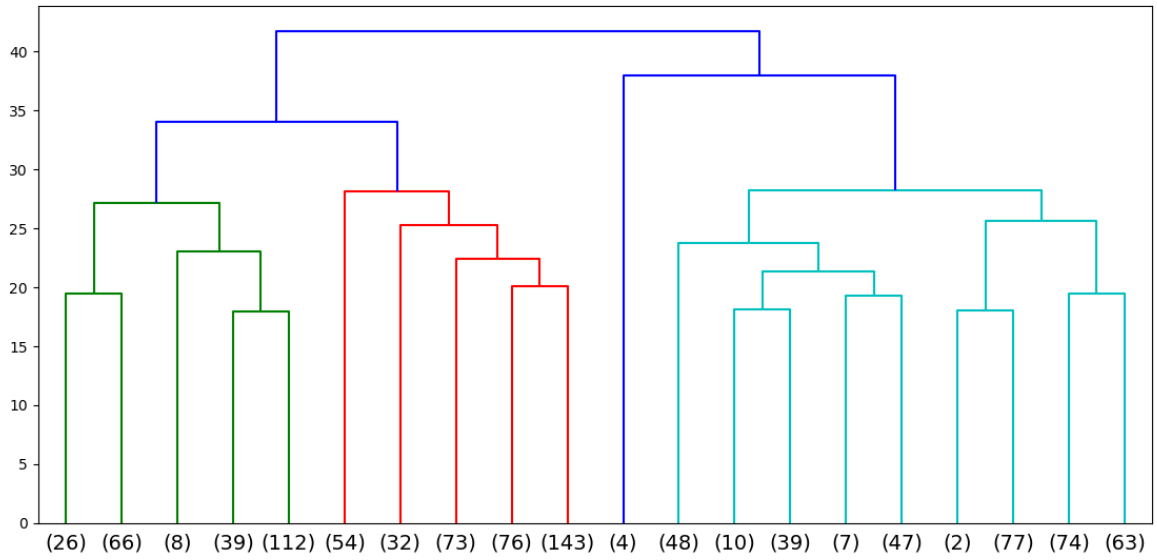


- Método Centroid:



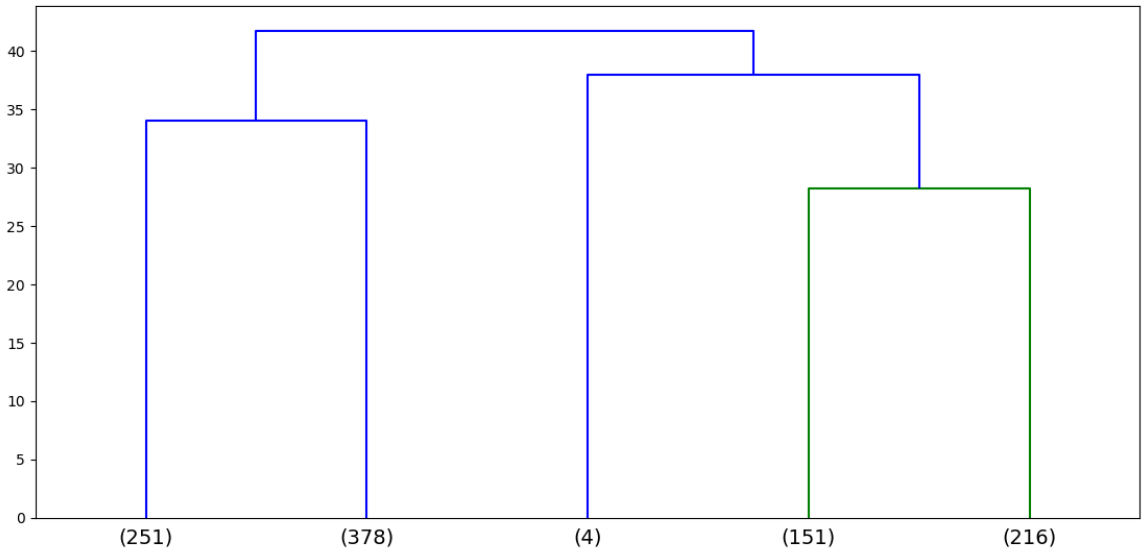
Como se ve el que mejor se ve es el del método *complete*, pero no es viable analizar la gráfica completa con tantos datos. Para eso se truncó la gráfica para ver solo los últimos agrupamientos, con los siguientes resultados:

- Método Complete, truncada a los últimos 20 agrupamientos:



Analizando los resultados parece natural pensar que hay de 2 a 5 grupos de datos.

- Método Complete, truncada a los últimos 5 agrupamientos:



Con el uso de estos dendrograma es posible notar más caramente los valores atípicos.

El código asociado a la sección de dendrogramas se puede visualizar en archivo dendograma.py.

b) k-means

El siguiente método de agrupamiento utilizado es k-means, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano.

Se tomo el valor de k=5, debido a los resultados observados en el método de dendogramas, los resultados fueron los siguientes:

- Los centroides finales fueron:

[5.8090197264509191, 2.35873912020701, 15.804015861813998, 1.0274221191652384, 1.8537991060928256, 22.094051819739892, 5.9821520986659174, 0.39153812548307959]

[9.3892866274610078, 2.8463308616722065, 11.512758885195629, 11.970851444643314, 10.784402965993349, 12.69351828176938, 3.425722321656866, 0.41843518281769371]

[6.8121617081637984, 2.5922206875956548, 7.0039941767143343, 0.98783082608533357, 2.0100115719139562, 21.362805629176179, 5.2477920041808188, 0.37071185934525364]

[6.1947821053502024, 2.3691391714214833, 18.903125775241847, 1.6500041346233358, 2.2838129496402693, 4.1937319110229057, 2.9281526502935669, 0.60427520052923178]

[6.2898936170212769, 2.4591333679294238, 8.6306661909703823, 1.7222042034250129, 2.8753373118837455, 6.3218409444732169, 2.5510962636222141, 0.48491826673585886]

- Los clusters quedaron de la siguiente manera:

```
El cluster  0  incluye  29757 miembros.
El cluster  1  incluye  7822 miembros.
El cluster  2  incluye  26789 miembros.
El cluster  3  incluye  24186 miembros.
El cluster  4  incluye  30832 miembros.
```

Al igual que en los dendogramas hubo un cluster en el que había muchos menos valores que en los demás. Esto puede deberse a la presencia de valores atípicos.

El código asociado a la sección de k-means se puede visualizar en archivo kmeans.py.

c) ISODATA

El método ISODATA es un algoritmo similar al k-means, su objetivo es particionar un conjunto de datos en subconjuntos. Sin embargo, a diferencia de k-means, éste maneja una seria de heurísticas con tres objetivos que le permiten optimizar el número de clusters:

- Eliminar clusters con pocos ejemplares.
- Unir clusters muy cercanos.
- Dividir clusters dispersos.

Se inicializó el número de clusters en 8, los resultados fueron los siguientes:

- Centroides Iniciales:

[1.0, 1.0, 10.800000000000001, 2.0, 1.8999999999999999, 7.7000000000000002, 1.3, 1.0]

[2.0, 1.0, 3.3999999999999999, 0.0, 0.80000000000000004, 28.199999999999999, 6.0999999999999996, 0.0]

[4.0, 2.0, 10.300000000000001, 2.0, 1.8999999999999999, 17.199999999999999, 3.3999999999999999, 0.0]

[5.0, 2.0, 10.9, 2.0, 2.7000000000000002, 3.7000000000000002, 1.1000000000000001, 0.0]

[6.0, 2.0, 9.0999999999999996, 2.0, 4.4000000000000004, 18.399999999999999, 2.6000000000000001, 0.0]

[7.0, 4.0, 14.5, 11.0, 9.0, 20.699999999999999, 6.0999999999999996, 0.0]

[8.0, 4.0, 3.3999999999999999, 3.0, 2.5, 3.5, 2.1000000000000001, 1.0]

[9.0, 4.0, 12.4, 0.0, 0.80000000000000004, 24.600000000000001, 7.2999999999999998, 0.0]

- Centroides Finales:

[5.9516747313517246, 2.3679381637654746, 14.438314585559038, 1.7937954293135592, 2.581185194495125, 4.6350614801315366, 2.6793900165483322, 0.5605479796392886]

[5.8867195403229067, 2.4175332820782582, 12.34269210536956, 0.84973495730991788, 1.7281956864809094, 22.421933800024341, 5.8584489944563876, 0.38259215797353618]

[8.8864587552315371, 2.8119346564061023, 7.7374555600557535, 5.2223122271724947, 5.787903334683401, 13.315084829665585, 3.4829485621709244, 0.39381665991629539]

- Clusters iniciales:

```
El cluster  0  incluye  10177 miembros.
El cluster  1  incluye  6260 miembros.
El cluster  2  incluye  16652 miembros.
El cluster  3  incluye  34751 miembros.
El cluster  4  incluye  9333 miembros.
El cluster  5  incluye  6990 miembros.
El cluster  6  incluye  9274 miembros.
El cluster  7  incluye  25949 miembros.
```

- Clusters finales:

```
El cluster  0  incluye  46773 miembros.
El cluster  1  incluye  50426 miembros.
El cluster  2  incluye  22187 miembros.
```

En la sección de dendogramas se apuntó que los grupos que se distinguían eran de 2 a 5. El resultado de este algoritmo fue que después de su ejecución solamente quedaron 3.

El código asociado a la sección de ISODATA se puede visualizar en archivo isodata.py .

d) DBSCAN

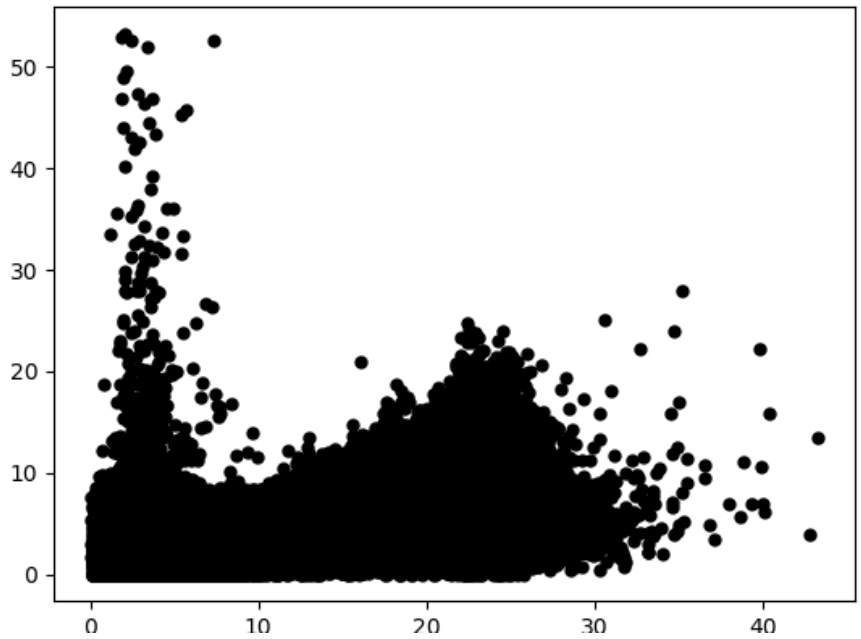
El método DBSCAN visualiza los clusters como áreas de alta densidad separadas por áreas de baja densidad. Gracias a este enfoque, DBSCAN es capaz de identificar adecuadamente cualquier forma de cluster.

Otras características importantes del algoritmo DBSCAN son:

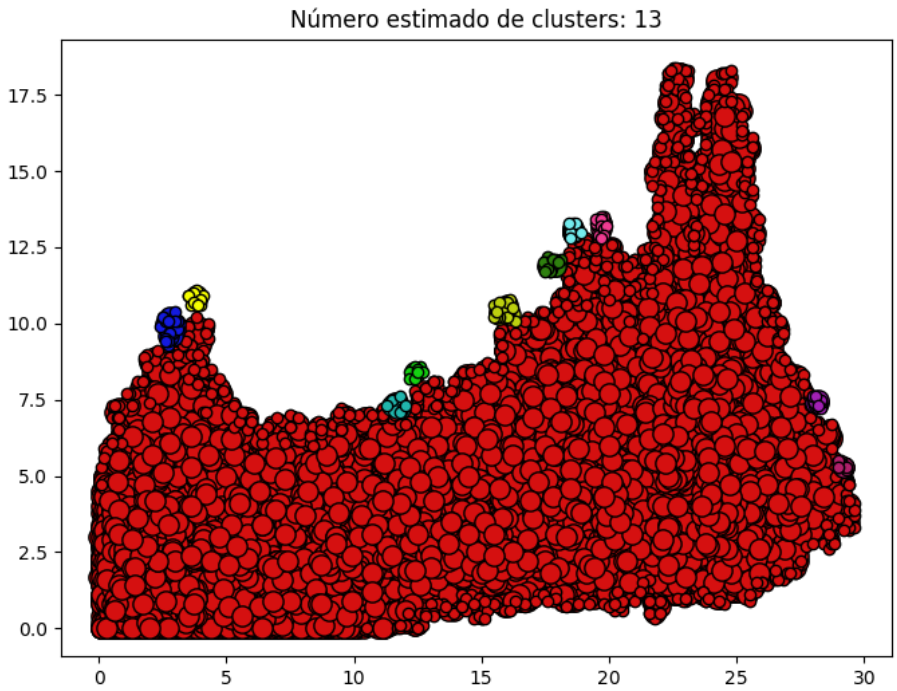
- DBSCAN no requiere que el usuario especifique el número de clusters a *priori*.
- DBSCAN es capaz de identificar valores atípicos.

Se hizo un análisis del comportamiento de los datos por parejas de características. Se tomo una muestra comparando los datos de las parejas y después con el algoritmo DBSCAN. A continuación, se muestran los resultados de comparar y agrupar las columnas SHOT_DIST con CLOSE_DEF_DIST y SHOT_CLOCK con CLOSE_DEF_DIST

- SHOT_DIST y CLOSE_DEF_DIST

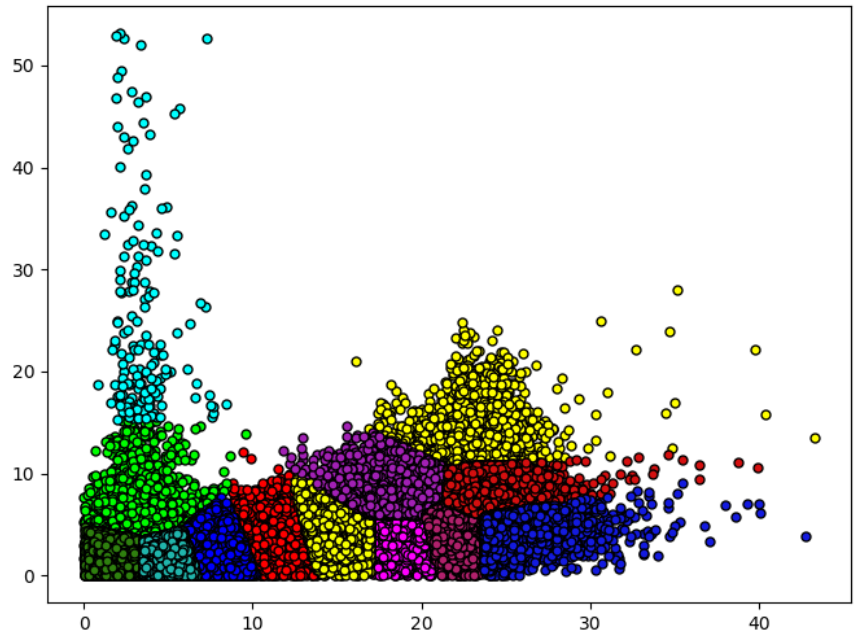


Se puede observar una gran cantidad de valores dispersos, que son los atípicos.



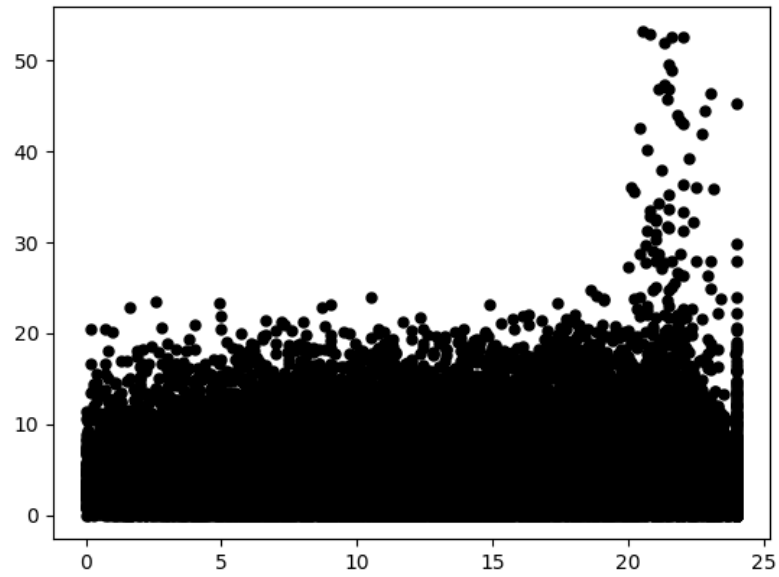
Los resultados de aplicarle DBSCAN a estas dos columnas dan como resultado la imagen anterior. Se eliminaron los datos dispersos y se aplicó clustering. No parece ser una buena agrupación ya que el área de color rojo es muy desproporcionada comparada con las demás y no se ve una clara división.

Con el mismo número de clusters se aplicó k-means a esas dos mismas columnas con los siguientes resultados:

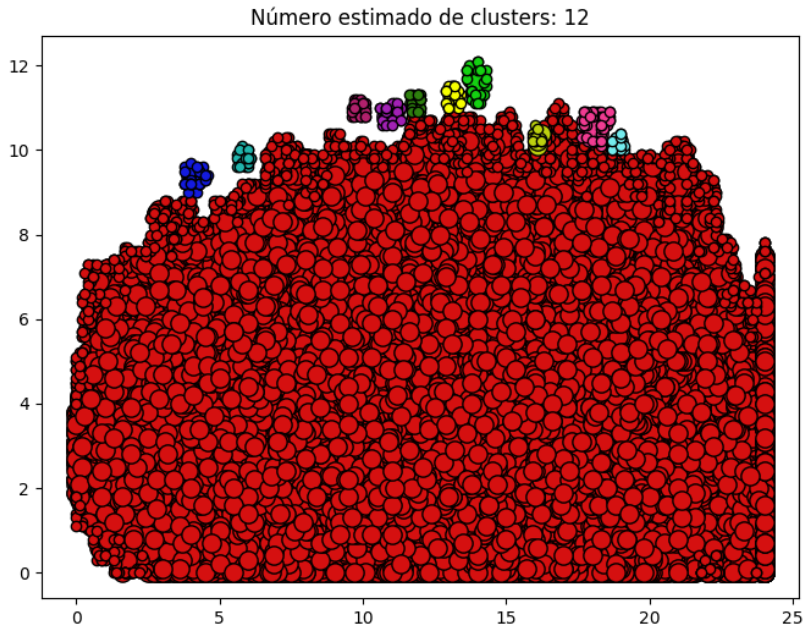


Como se ve, los valores atípicos son tomados en cuenta, pero la división se aprecia de una manera más clara.

- SHOT_CLOCK y CLOSE_DEF_DIST

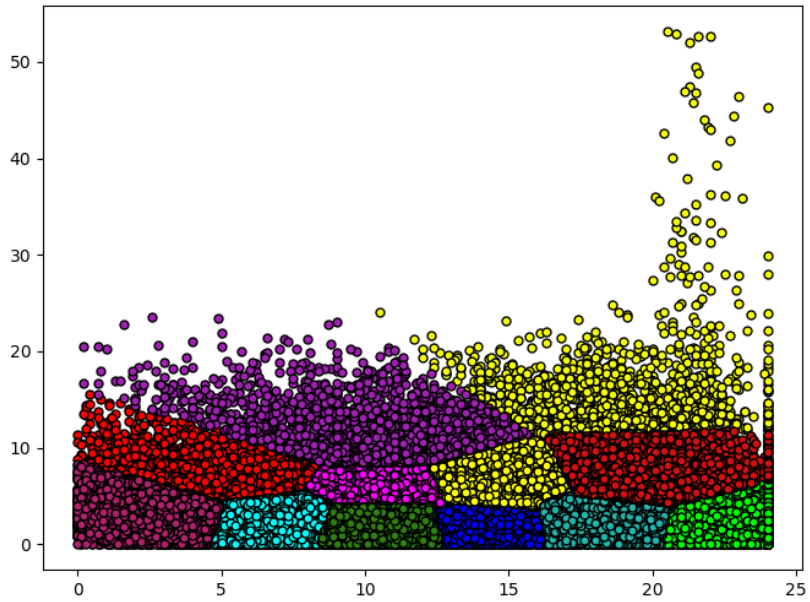


Se puede ver mediante la comparación, que los valores atípicos de CLOSE_DEF_DIST se encuentran cuando el reloj de tiro está por terminar, que es a los 24 segundos.



El resultado de agrupar por DBSCAN con los parámetros dados no aporta mucho, ya que da pequeñas agrupaciones y una gran agrupación. Esto se puede deber a que los datos están muy agrupados en una sola masa de datos.

Con el mismo número de clusters se aplicó k-means a esas dos mismas columnas con los siguientes resultados:



La división se ve más clara y equitativa pero no a simple vista sin los colores no sería esta una clara división.

El código asociado a la sección de DBSCAN se puede visualizar en archivo dbscan.py .

5. Clasificación

Se entiende por clasificación el proceso de identificar a qué conjunto de categorías o *clases* pertenece una nueva observación, sobre la base de un conjunto de elementos cuya pertenencia a una categoría ya es conocida.

Entre los métodos más populares de clasificación se encuentran los siguientes:

- El método de k-vecinos próximos
- El clasificador ingenuo de Bayes
- Los árboles de decisión
- Las máquinas de soporte vectorial
- Las redes neuronales

a) k-Vecinos próximos

El método de k-vecinos próximos (o simplemente kNN) es un método de *aprendizaje vago* (*lazy learning*). Esto significa que el aprendizaje no conduce a una generalización: no existe una fase de entrenamiento (o es muy breve) y en su lugar el método mantiene todos los datos disponibles y los emplea para realizar la clasificación.

En este método se tomó el primer elemento de los datos que corresponde a un tiro encestado y se intentó encontrar su clasificación. Se tomaron los 5 vecinos más cercanos a este elemento y más adelante se tomaron 172 debido a que correspondía con la cantidad sugerida: 1/2 $\sqrt{n}$ donde n es la cantidad de datos.

- Caso de prueba: [3,4,12.1,14,11.9,14.6,1.8] Pertenece a la clase 1

Los 172 vecinos más próximos y sus pesos ponderados son:

Vecino 0: peso=0.9973253951916758, clase: 0

Vecino 1: peso=0.9969976567018114, clase: 0

Vecino 2: peso=0.9967096349908479, clase: 0

Vecino 3: peso=0.9962409323125656, clase: 0

Vecino 4: peso=0.99619007451933, clase: 0

Vecino 5: peso=0.9961774657960137, clase: 0

Vecino 6: peso=0.9960862428190449, clase: 0

Vecino 7: peso=0.9958379808951034, clase: 0

Vecino 8: peso=0.9957597154842908, clase: 0

Vecino 9: peso=0.9957578246103882, clase: 0

Vecino 10: peso=0.9957559345789335, clase: 0

Vecino 11: peso=0.9957126938147862, clase: 0

Vecino 12: peso=0.9956996198818915, clase: 0

Vecino 13: peso=0.9956569397669726, clase: 0

Vecino 14: peso=0.9956532482573958, clase: 0

Vecino 15: peso=0.9956091922153916, clase: 0

Vecino 16: peso=0.9955673827112125, clase: 0

Vecino 17: peso=0.9954884787976016, clase: 0

Vecino 18: peso=0.9954372199089199, clase: 0

Vecino 19: peso=0.9954004579942227, clase: 0

Vecino 20: peso=0.9953260856948722, clase: 0

Vecino 21: peso=0.9952175333034304, clase: 0

Vecino 22: peso=0.9952091562315059, clase: 0

Vecino 23: peso=0.995197452879089, clase: 0

Vecino 24: peso=0.9950852132631265, clase: 0

Vecino 25: peso=0.9950689229077623, clase: 0

Vecino 26: peso=0.9950187613574948, clase: 0

Vecino 27: peso=0.9950026876189065, clase: 0

Vecino 28: peso=0.9949978755692782, clase: 0

Vecino 29: peso=0.9949914666948616, clase: 0

Vecino 30: peso=0.994950007259178, clase: 0

Vecino 31: peso=0.9949325688396325, clase: 0

Vecino 32: peso=0.9949246620924923, clase: 0

Vecino 33: peso=0.9948556067018338, clase: 0

Vecino 34: peso=0.9948509321045057, clase: 0

Vecino 35: peso=0.9948167805412247, clase: 0

Vecino 36: peso=0.9948105953129396, clase: 0

Vecino 37: peso=0.9948013312686351, clase: 0

Vecino 38: peso=0.9947920837035474, clase: 0

Vecino 39: peso=0.9947020090022108, clase: 0

Vecino 40: peso=0.9947020090022108, clase: 0

Vecino 41: peso=0.9946929345034152, clase: 0

Vecino 42: peso=0.9946778447217612, clase: 0

Vecino 43: peso=0.9946253641074081, clase: 0

Vecino 44: peso=0.9946104635080016, clase: 0

Vecino 45: peso=0.9945985726242875, clase: 0

Vecino 46: peso=0.9945527406139333, clase: 0

Vecino 47: peso=0.9945453844196485, clase: 0

Vecino 48: peso=0.9945321682096105, clase: 0

Vecino 49: peso=0.9945160583180799, clase: 0

Vecino 50: peso=0.9944883430485992, clase: 0

Vecino 51: peso=0.9944723608807527, clase: 0

Vecino 52: peso=0.9944189385481779, clase: 0

Vecino 53: peso=0.9944175017965897, clase: 0

Vecino 54: peso=0.9944131937587498, clase: 0

Vecino 55: peso=0.9943774218076685, clase: 0

Vecino 56: peso=0.9943759956622575, clase: 0

Vecino 57: peso=0.9943731444558195, clase: 0

Vecino 58: peso=0.9943660227524956, clase: 0

Vecino 59: peso=0.9943617540483242, clase: 0

Vecino 60: peso=0.994354646714163, clase: 0

Vecino 61: peso=0.9943206553184847, clase: 0

Vecino 62: peso=0.9942854626116311, clase: 0

Vecino 63: peso=0.9942770485642938, clase: 0

Vecino 64: peso=0.9942770485642938, clase: 0

Vecino 65: peso=0.994275647425471, clase: 0

Vecino 66: peso=0.9942588604317358, clase: 0

Vecino 67: peso=0.9942421223799858, clase: 0

Vecino 68: peso=0.9942032548766688, clase: 0

Vecino 69: peso=0.9941935786529641, clase: 0

Vecino 70: peso=0.9941894366326915, clase: 0

Vecino 71: peso=0.9941825398191454, clase: 0

Vecino 72: peso=0.9941618982634083, clase: 0

Vecino 73: peso=0.9941550339438867, clase: 0

Vecino 74: peso=0.994144067769037, clase: 0

Vecino 75: peso=0.9941413294325623, clase: 0

Vecino 76: peso=0.9941385923753855, clase: 0

Vecino 77: peso=0.9941276569038923, clase: 0

Vecino 78: peso=0.9941235613485209, clase: 0

Vecino 79: peso=0.9941112917729626, clase: 0

Vecino 80: peso=0.9941004069055831, clase: 0

Vecino 81: peso=0.9940746354830344, clase: 0

Vecino 82: peso=0.9940638178299996, clase: 0

Vecino 83: peso=0.994058416389237, clase: 0

Vecino 84: peso=0.9940530198544182, clase: 0

Vecino 85: peso=0.9940489756648004, clase: 0

Vecino 86: peso=0.9940261103045643, clase: 0

Vecino 87: peso=0.9940261103045643, clase: 0

Vecino 88: peso=0.9940033321295634, clase: 0

Vecino 89: peso=0.9939993214271617, clase: 0

Vecino 90: peso=0.993955379335091, clase: 0

Vecino 91: peso=0.993948749278623, clase: 0

Vecino 92: peso=0.9939460992891281, clase: 0

Vecino 93: peso=0.9939289025519324, clase: 0

Vecino 94: peso=0.9939183441136112, clase: 0

Vecino 95: peso=0.9939130717624811, clase: 0

Vecino 96: peso=0.993897282038839, clase: 0

Vecino 97: peso=0.9938828439290643, clase: 0

Vecino 98: peso=0.9938762924052086, clase: 0

Vecino 99: peso=0.9938749829413649, clase: 0

Vecino 100: peso=0.9938658245213378, clase: 0

Vecino 101: peso=0.9938449421614374, clase: 0

Vecino 102: peso=0.9938319265879292, clase: 0

Vecino 103: peso=0.9938267280422005, clase: 0

Vecino 104: peso=0.9938228320039726, clase: 0

Vecino 105: peso=0.9937904600812334, clase: 0

Vecino 106: peso=0.9937878776149464, clase: 0

Vecino 107: peso=0.9937467031964775, clase: 0

Vecino 108: peso=0.9937402941584076, clase: 0

Vecino 109: peso=0.9937313325133872, clase: 0

Vecino 110: peso=0.9937198291889237, clase: 0

Vecino 111: peso=0.9936994306916497, clase: 0

Vecino 112: peso=0.9936994306916497, clase: 0

Vecino 113: peso=0.9936905270866913, clase: 0

Vecino 114: peso=0.9936740250671843, clase: 0

Vecino 115: peso=0.9936689561762004, clase: 0

Vecino 116: peso=0.9936638913403313, clase: 0

Vecino 117: peso=0.9936613604400268, clase: 0

Vecino 118: peso=0.9936537737951091, clase: 0

Vecino 119: peso=0.9936499838716869, clase: 0

Vecino 120: peso=0.9936298090277554, clase: 0

Vecino 121: peso=0.993628550221109, clase: 0

Vecino 122: peso=0.9936184887070972, clase: 0

Vecino 123: peso=0.9936084430317832, clase: 0

Vecino 124: peso=0.9936046799717173, clase: 0

Vecino 125: peso=0.9935959080972706, clase: 0

Vecino 126: peso=0.9935609400968459, clase: 0

Vecino 127: peso=0.9935236838945016, clase: 0

Vecino 128: peso=0.9934952653147251, clase: 0

Vecino 129: peso=0.9934940325402072, clase: 0

Vecino 130: peso=0.9934841787406039, clase: 0

Vecino 131: peso=0.9934718824072293, clase: 0

Vecino 132: peso=0.9934387974135239, clase: 0

Vecino 133: peso=0.9934290264437228, clase: 0

Vecino 134: peso=0.9934241464032757, clase: 0

Vecino 135: peso=0.9934107449276675, clase: 0

Vecino 136: peso=0.9933973706532193, clase: 0

Vecino 137: peso=0.9933840234149691, clase: 0

Vecino 138: peso=0.9933828113646548, clase: 0

Vecino 139: peso=0.9933815995363074, clase: 0

Vecino 140: peso=0.9933646571779066, clase: 0

Vecino 141: peso=0.9933489636316177, clase: 0

Vecino 142: peso=0.9933212884625083, clase: 0

Vecino 143: peso=0.9932937275011744, clase: 0

Vecino 144: peso=0.9932829735606383, clase: 0

Vecino 145: peso=0.9932829735606383, clase: 0

Vecino 146: peso=0.9932805861333586, clase: 0

Vecino 147: peso=0.9932781995540375, clase: 0

Vecino 148: peso=0.9932710448948051, clase: 0

Vecino 149: peso=0.9932710448948051, clase: 0

Vecino 150: peso=0.9932686616983057, clase: 0

Vecino 151: peso=0.9932591373379684, clase: 0

Vecino 152: peso=0.9932579477389717, clase: 0

Vecino 153: peso=0.9932496264159021, clase: 0

Vecino 154: peso=0.9932436888885126, clase: 0

Vecino 155: peso=0.9932389426209012, clase: 0

Vecino 156: peso=0.9932377565745139, clase: 0

Vecino 157: peso=0.9932365707361136, clase: 0

Vecino 158: peso=0.9932235402096447, clase: 0

Vecino 159: peso=0.9931869518998984, clase: 0

Vecino 160: peso=0.9931775416079822, clase: 0

Vecino 161: peso=0.9931611047544421, clase: 0

Vecino 162: peso=0.9931435375658667, clase: 0

Vecino 163: peso=0.9931435375658667, clase: 0

Vecino 164: peso=0.9931213501986288, clase: 0

Vecino 165: peso=0.9931073739231933, clase: 0

Vecino 166: peso=0.9931027214582174, clase: 0

Vecino 167: peso=0.9931003964021792, clase: 0

Vecino 168: peso=0.9930945871870406, clase: 0

Vecino 169: peso=0.9930934259303582, clase: 0

Vecino 170: peso=0.9930667707086143, clase: 0

Vecino 171: peso=0.9930609896442467, clase: 0

```
Votación ponderada:
El nuevo punto es asignado a la clase 0 con una votación de 170.99999999999997.
```

Como los datos son muchos encuentra hay muchos datos muy cercanos, pero, aunque la clase a la que pertenece es la 1, el resultado da que pertenece a la clase 0.

- Caso de prueba: [1,4,4.4,0,0.8,26.4,4.4] Pertenece a la clase 0

Los 172 vecinos más próximos y sus pesos ponderados son:

Vecino 0: peso=1.0, clase: 1

Vecino 1: peso=0.9973067063994908, clase: 0

Vecino 2: peso=0.9969766778380206, clase: 0

Vecino 3: peso=0.9966866435763116, clase: 1

Vecino 4: peso=0.9962146658396268, clase: 1

Vecino 5: peso=0.9961634526777304, clase: 1

Vecino 6: peso=0.996150755851002, clase: 0

Vecino 7: peso=0.9960588954537857, clase: 0

Vecino 8: peso=0.995808898800478, clase: 1

Vecino 9: peso=0.9957300865103681, clase: 0

Vecino 10: peso=0.9957281824239904, clase: 1

Vecino 11: peso=0.9957262791859472, clase: 1

Vecino 12: peso=0.9956827362771045, clase: 0

Vecino 13: peso=0.9956695709901472, clase: 1

Vecino 14: peso=0.9956265926480676, clase: 1

Vecino 15: peso=0.9956228753440799, clase: 1

Vecino 16: peso=0.995578511460629, clase: 1

Vecino 17: peso=0.9955364098126788, clase: 0

Vecino 18: peso=0.9954569545582345, clase: 1

Vecino 19: peso=0.9954053374982389, clase: 1

Vecino 20: peso=0.9953683187097854, clase: 1

Vecino 21: peso=0.995293426734241, clase: 0

Vecino 22: peso=0.9951841158333258, clase: 0

Vecino 23: peso=0.995175680226639, clase: 0

Vecino 24: peso=0.9951638950970862, clase: 1

Vecino 25: peso=0.9950508712071812, clase: 0

Vecino 26: peso=0.9950344670230132, clase: 1

Vecino 27: peso=0.9949839549690803, clase: 1

Vecino 28: peso=0.9949677689152971, clase: 1

Vecino 29: peso=0.9949629232414882, clase: 1

Vecino 30: peso=0.9949564695850829, clase: 0

Vecino 31: peso=0.9949147204517307, clase: 0

Vecino 32: peso=0.9948971601812867, clase: 0

Vecino 33: peso=0.9948891981857771, clase: 0

Vecino 34: peso=0.9948196602708066, clase: 0

Vecino 35: peso=0.9948149530097452, clase: 0

Vecino 36: peso=0.9947805628125297, clase: 1

Vecino 37: peso=0.9947743343649825, clase: 0

Vecino 38: peso=0.9947650055881994, clase: 0

Vecino 39: peso=0.9947556934057814, clase: 0

Vecino 40: peso=0.9946649893077695, clase: 1

Vecino 41: peso=0.9946649893077695, clase: 0

Vecino 42: peso=0.9946558514009437, clase: 0

Vecino 43: peso=0.9946406561794907, clase: 1

Vecino 44: peso=0.9945878088570196, clase: 1

Vecino 45: peso=0.9945728041397245, clase: 1

Vecino 46: peso=0.9945608301685, clase: 1

Vecino 47: peso=0.9945146779071262, clase: 1

Vecino 48: peso=0.9945072703114582, clase: 1

Vecino 49: peso=0.9944939617531968, clase: 0

Vecino 50: peso=0.9944777392938527, clase: 1

Vecino 51: peso=0.9944498303640189, clase: 0

Vecino 52: peso=0.9944337365208279, clase: 1

Vecino 53: peso=0.9943799408998822, clase: 0

Vecino 54: peso=0.9943784941089971, clase: 1

Vecino 55: peso=0.9943741559687572, clase: 1

Vecino 56: peso=0.9943381340612869, clase: 0

Vecino 57: peso=0.9943366979506897, clase: 0

Vecino 58: peso=0.9943338268214561, clase: 0

Vecino 59: peso=0.994326655355254, clase: 1

Vecino 60: peso=0.9943223568235263, clase: 0

Vecino 61: peso=0.9943151998268913, clase: 1

Vecino 62: peso=0.9942809709164491, clase: 0

Vecino 63: peso=0.9942455323006737, clase: 1

Vecino 64: peso=0.9942370594602086, clase: 0

Vecino 65: peso=0.9942370594602086, clase: 1

Vecino 66: peso=0.9942356485309329, clase: 1

Vecino 67: peso=0.9942187442381349, clase: 0

Vecino 68: peso=0.9942018892293041, clase: 0

Vecino 69: peso=0.9941627501394372, clase: 1

Vecino 70: peso=0.9941530063031516, clase: 0

Vecino 71: peso=0.9941488353405262, clase: 1

Vecino 72: peso=0.9941418903355184, clase: 1

Vecino 73: peso=0.9941211045469794, clase: 1

Vecino 74: peso=0.994114192263048, clase: 1

Vecino 75: peso=0.9941031494620881, clase: 1

Vecino 76: peso=0.9941003919914965, clase: 0

Vecino 77: peso=0.9940976358091417, clase: 1

Vecino 78: peso=0.9940866239260778, clase: 0

Vecino 79: peso=0.994082499753027, clase: 1

Vecino 80: peso=0.9940701444438504, clase: 1

Vecino 81: peso=0.9940591835184963, clase: 1

Vecino 82: peso=0.9940332320182185, clase: 0

Vecino 83: peso=0.9940223387768692, clase: 1

Vecino 84: peso=0.9940168995935572, clase: 1

Vecino 85: peso=0.9940114653504692, clase: 0

Vecino 86: peso=0.9940073929020901, clase: 1

Vecino 87: peso=0.9939843677702256, clase: 0

Vecino 88: peso=0.9939843677702256, clase: 1

Vecino 89: peso=0.9939614304328027, clase: 0

Vecino 90: peso=0.9939573917056314, clase: 0

Vecino 91: peso=0.9939131425683387, clase: 0

Vecino 92: peso=0.9939064661843732, clase: 0

Vecino 93: peso=0.9939037976780856, clase: 1

Vecino 94: peso=0.9938864807787461, clase: 0

Vecino 95: peso=0.9938758485633723, clase: 0

Vecino 96: peso=0.9938705393717062, clase: 1

Vecino 97: peso=0.9938546393174228, clase: 1

Vecino 98: peso=0.9938401003214041, clase: 0

Vecino 99: peso=0.9938335030187972, clase: 1

Vecino 100: peso=0.9938321844050794, clase: 1

Vecino 101: peso=0.9938229619906228, clase: 0

Vecino 102: peso=0.9938019337153021, clase: 0

Vecino 103: peso=0.9937888271955174, clase: 1

Vecino 104: peso=0.9937835923249679, clase: 1

Vecino 105: peso=0.9937796690631858, clase: 0

Vecino 106: peso=0.9937470709417444, clase: 0

Vecino 107: peso=0.9937444704304828, clase: 1

Vecino 108: peso=0.9937030083059021, clase: 0

Vecino 109: peso=0.9936965544847001, clase: 1

Vecino 110: peso=0.9936875302202148, clase: 0

Vecino 111: peso=0.99367594651631, clase: 1

Vecino 112: peso=0.9936554054846043, clase: 1

Vecino 113: peso=0.9936554054846043, clase: 0

Vecino 114: peso=0.993646439665736, clase: 0

Vecino 115: peso=0.9936298223384226, clase: 1

Vecino 116: peso=0.9936247180285799, clase: 0

Vecino 117: peso=0.993619617802187, clase: 0

Vecino 118: peso=0.993617069217225, clase: 1

Vecino 119: peso=0.9936094295606513, clase: 0

Vecino 120: peso=0.9936056131551518, clase: 1

Vecino 121: peso=0.9935852973395656, clase: 1

Vecino 122: peso=0.9935840297370119, clase: 0

Vecino 123: peso=0.9935738979182045, clase: 1

Vecino 124: peso=0.9935637820487677, clase: 1

Vecino 125: peso=0.9935599926943322, clase: 1

Vecino 126: peso=0.9935511595264419, clase: 0

Vecino 127: peso=0.9935159471872308, clase: 1

Vecino 128: peso=0.9934784306572968, clase: 0

Vecino 129: peso=0.9934498135027888, clase: 0

Vecino 130: peso=0.9934485721142631, clase: 0

Vecino 131: peso=0.9934386494612679, clase: 0

Vecino 132: peso=0.9934262672073049, clase: 0

Vecino 133: peso=0.9933929510323162, clase: 1

Vecino 134: peso=0.9933831117878961, clase: 0

Vecino 135: peso=0.9933781976481826, clase: 1

Vecino 136: peso=0.9933647025298082, clase: 0

Vecino 137: peso=0.9933512348026621, clase: 0

Vecino 138: peso=0.9933377943006291, clase: 0

Vecino 139: peso=0.9933365737811172, clase: 0

Vecino 140: peso=0.9933353534851233, clase: 0

Vecino 141: peso=0.9933182927420495, clase: 1

Vecino 142: peso=0.9933024895371583, clase: 1

Vecino 143: peso=0.993274620987897, clase: 0

Vecino 144: peso=0.9932468674444376, clase: 1

Vecino 145: peso=0.9932360383607778, clase: 0

Vecino 146: peso=0.9932360383607778, clase: 1

Vecino 147: peso=0.9932336342513579, clase: 0

Vecino 148: peso=0.9932312309958218, clase: 0

Vecino 149: peso=0.9932240263434319, clase: 0

Vecino 150: peso=0.9932240263434319, clase: 0

Vecino 151: peso=0.9932216264943549, clase: 1

Vecino 152: peso=0.9932120355825814, clase: 0

Vecino 153: peso=0.9932108376712657, clase: 0

Vecino 154: peso=0.9932024582029789, clase: 1

Vecino 155: peso=0.9931964791871365, clase: 1

Vecino 156: peso=0.9931916997549961, clase: 0

Vecino 157: peso=0.9931905054211135, clase: 0

Vecino 158: peso=0.9931893112966714, clase: 0

Vecino 159: peso=0.9931761897194421, clase: 1

Vecino 160: peso=0.9931393457490034, clase: 0

Vecino 161: peso=0.9931298697027039, clase: 0

Vecino 162: peso=0.9931133179967044, clase: 1

Vecino 163: peso=0.9930956280574574, clase: 1

Vecino 164: peso=0.9930956280574574, clase: 1
Vecino 165: peso=0.9930732856560649, clase: 0
Vecino 166: peso=0.9930592117214513, clase: 1
Vecino 167: peso=0.993054526747392, clase: 1
Vecino 168: peso=0.9930521854450324, clase: 1
Vecino 169: peso=0.9930463356380224, clase: 0
Vecino 170: peso=0.9930451662670629, clase: 0
Vecino 171: peso=0.9930183247920458, clase: 1

```
Votación ponderada:
El nuevo punto es asignado a la clase 1 con una votación de 85.50760679397708.
```

El dato de prueba tiene muchos vecinos cercanos, el resultado indica que pertenece a la clase 1, pero en realidad pertenece a la clase 0.

Los resultados indican que los datos no son fáciles de clasificar.

El código asociado a la sección de k-vecinos próximos se puede visualizar en archivo kvecinos.py.

b) Clasificador de Bayes Ingenuo

El método Bayes ingenuo o *Naive Bayes* describe a una familia de clasificadores que utilizan el Teorema de Bayes para realizar la clasificación de nuevas observaciones de una manera ingenua: asumiendo que existe total independencia entre los diferentes atributos de un objeto.
La columna de SHOT_RESULT es tomada como la clase, si el tiro entro o no entro.
Para el clasificador de Bayes se dividieron los datos en los totales, los datos de entrenamiento y datos de prueba y se utilizó la librería sklearn.naive_bayes para poder hacer las pruebas con las distribuciones: Bernoulli, Gaussiana y Multinomial.
Los resultados fueron los siguientes:

- Distribución Gaussiana:
  Puntos mal clasificados en el conjunto completo: 48590 de 119386 (40.699914562846566%)
  Puntos mal clasificados en el conjunto de entrenamiento: 32546 de 79590 (40.89207186832517%)
  Puntos mal clasificados en el conjunto de prueba: 16379 de 39796 (41.157402754045634%)

- Distribución Multinomial:
  Puntos mal clasificados en el conjunto completo: 48849 de 119386 (40.91685792303955%)
  Puntos mal clasificados en el conjunto de entrenamiento: 32534 de 79590 (40.87699459731122%)

Puntos mal clasificados en el conjunto de prueba: 16256 de 39796 (40.84832646497135%)

- Distribución de Bernoulli:
  Puntos mal clasificados en el conjunto completo: 54420 de 119386 (45.583234215067094%)
  Puntos mal clasificados en el conjunto de entrenamiento: 36479 de 79590 (45.83364744314613%)
  Puntos mal clasificados en el conjunto de prueba: 17942 de 39796 (45.08493315911147%)

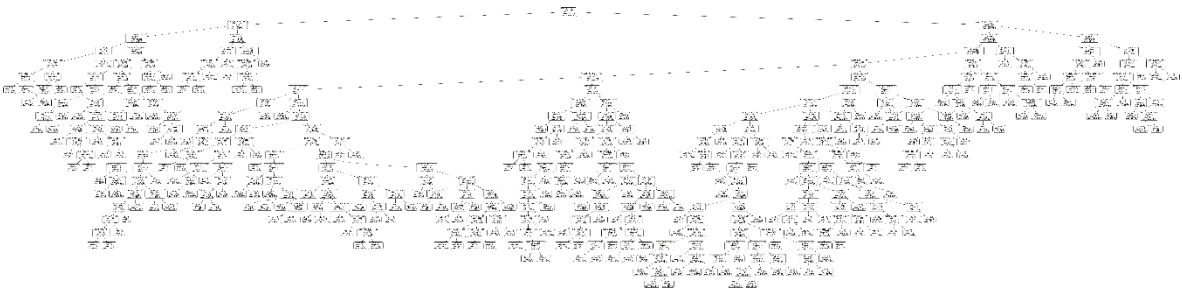
Como se puede ver los errores son bastante grandes, eso indica que la clasificación del dataset no es sencilla. Con la distribución de Bernoulli el error fue más grande.

El código asociado a la sección del clasificador de bayes ingenuo próximos se puede visualizar en archivo bayes.py .

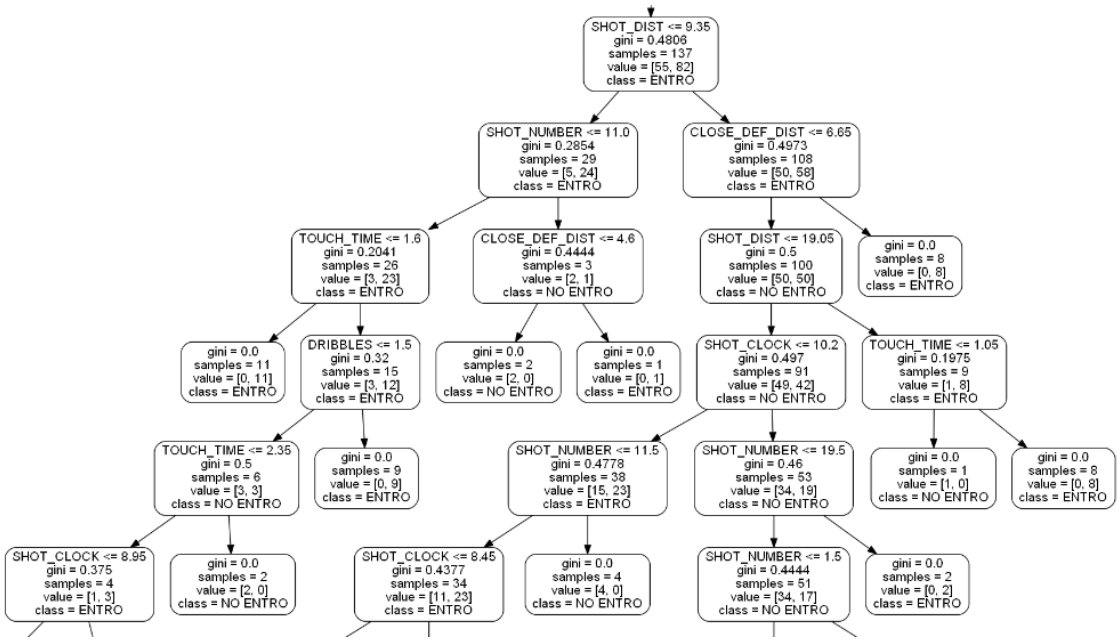
c) Árboles de decisión

Los árboles de decisión son uno de los métodos de clasificación más antiguos y robustos y, por lo tanto, de las más utilizados. Nuevamente, se trata de un método fácil de entender y cuyo mecanismo de razonamiento es transparente.

- Primero para 1000 datos se probó con los siguientes resultados:
  - **Objetivos: [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0]**
  - **Resultados: [0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1]**
  - **Puntos mal clasificados en el conjunto de prueba: 45 de 100 (45.0%)**
  - **El árbol resultante fue el siguiente:**



A simple vista se ve que es un árbol demasiado grande, por eso se hizo un acercamiento.



Como se aprecia en los resultados, el error de clasificación es muy similar a los de los métodos anteriores.

- Con los datos completos:
  Como los datos para probar son demasiados solo se muestran los primeros 20 objetivos y resultados de la muestra:
  - **Objetivos: [0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0...]**
  - **Resultados: [1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1...]**
  - **Puntos mal clasificados en el conjunto de prueba: 5440 de 11939 (45.56495518887679%)**

  Los resultados fueron similares a la clasificación con solo 1000 de los datos.

  Como se ve en el resultado de los puntos mal clasificados es mayor o similar al error en que en el clasificador de bayes ingenio.

El árbol de clasificación no se muestra porque por el tamaño es imposible colocarlo aquí.

El código asociado a la sección de árboles de decisión próximos se puede visualizar en archivo arboles.py .

d) Máquinas de Vectores de Soporte

Las SVM se basan en la idea de definir planos de decisión que separen a los objetos pertenecientes a diferentes clases. El objetivo de clasificación es encontrar el hiperplano que separe mejor las regiones en el espacio de características ocupadas por cada una de las clases.

Para todas las pruebas se tomó una muestra de 10000 datos de tiros por cuestiones de recursos computacionales.

A continuación, se presentan los resultados de la aplicación de SVM lineal a los datos de tiros con diferentes valores de penalización.

PENALIZACION C	PUNTOS DE ENTRENAMIENTO MAL CLASIFICADOS	PUNTOS DE PRUEBA MAL CLASIFICADOS
.1	39.85%	40.6%
.5	40.25%	40.8%
1	42.6625%	43.7%
5	54.0%	52.55%
10	50.1625%	50.25%

Los resultados son muy parecidos y van en aumento conforme el valor de C crece.

A continuación, se presentan los resultados de la aplicación de SVM con función kernel RBF a una muestra de 10000 datos de tiros con diferentes valores de penalización.

PENALIZACION C	PUNTOS DE ENTRENAMIENTO MAL CLASIFICADOS	PUNTOS DE PRUEBA MAL CLASIFICADOS
.1	41.7875%	42.3%
.5	28.6%	39.7%
1	19.075%	42.05%
5	7.775%	45.6%
10	4.925%	45.6%

Los resultados de entrenamiento mejoran conforme aumenta el valor de penalización, pero los resultados de los puntos de prueba empeoran o se mantienen.

A continuación, se presentan los resultados de la aplicación de SVM con función kernel RBF a una muestra de 10000 datos de tiros con diferentes valores de gamma.

GAMMA	PUNTOS DE ENTRENAMIENTO MAL CLASIFICADOS	PUNTOS DE PRUEBA MAL CLASIFICADOS
.1	30.875%	39.65%
.5	23.675%	42.5%
1	28.2625%	43.2%
5	32.875%	44.7%
10	37.9125%	45.5%

Modificando el valor de gamma los resultados obtenidos no son estables, conforme aumenta se nota un ligero incremento del porcentaje de error en los puntos de prueba.

El código asociado a la sección Maquinas de Vectores de soporte se puede visualizar en archivo svm.py .

e) Redes Neuronales

a. Perceptrón

El perceptrón es un modelo de neurona obtenido al combinar el modelo de la neurona de McCulloch-Pitts con una extensión de la regla de Hebb.

La regla de entrenamiento consta de los siguientes pasos:

1. Inicializar los pesos con valores aleatorios pequeños.
2. Para cada vector de entrenamiento xj con valor de clase yj (el valor de salida esperado):
   - Calcular el valor de salida y*j de la función de activación
   - Actualizar los pesos de acuerdo con la regla de Hebb modificada:

$$\Delta w = n\left(y_j - y_j^{\dot{c}}\right)x_j$$

Se utilizaron los primeros 100 datos revueltos para entrenar el perceptrón, 50 para probarlo. Se obtuvieron los siguientes resultados en 4 corridas:

CORRIDA	RESULTADOS
1	15 vectores mal clasificados de 50 (30.0%)
2	17 vectores mal clasificados de 50 (34.0%)
3	33 vectores mal clasificados de 50 (66.0%)
4	16 vectores mal clasificados de 50 (32.0%)

Como puede observarse, el porcentaje de error va desde 30% hasta 66%. Esto puede deberse a que solo se toma en cuenta si la clasificación fue correcta en términos del valor discreto de clase, pero no se toman en cuenta los aciertos que estuvieron cerca de fallar y se magnifican los errores cometidos por un margen pequeño.

Se realizo una segunda prueba agregando dos rondas de entrenamiento, la primera con la mitad de los datos totales y la tercera con los datos completos, se obtuvieron los siguientes resultados:

- Corrida 1:
  - PRIMERA RONDA.
    - **25322 vectores mal clasificados de 59693 (42.42038429966663%)**
  - SEGUNDA RONDA con la mitad de los datos.
    - **27997 vectores de entrenamiento mal clasificados de 59693 (46.901646759251506%)**
    - **26667 vectores mal clasificados de 59693 (44.67357981672893%)**
  - TERCERA RONDA con datos completos.
    - **55424 vectores de entrenamiento mal clasificados de 119386 (46.42420384299666%)**
    - **24971 vectores mal clasificados de 59693 (41.83237565543699%)**
- Corrida 2:
  - PRIMERA RONDA.
    - **23211 vectores mal clasificados de 59693 (38.88395624277553%)**
  - SEGUNDA RONDA con la mitad de los datos
    - **27917 vectores de entrenamiento mal clasificados de 59693 (46.76762769503962%)**
    - **23836 vectores mal clasificados de 59693 (39.93098018193088%)**
  - TERCERA RONDA con datos completos
    - **55437 vectores de entrenamiento mal clasificados de 119386 (46.43509289196388%)**
    - **28089 vectores mal clasificados de 59693 (47.055768683095174%)**
- Corrida 3:
  - PRIMERA RONDA.
    - **31902 vectores mal clasificados de 59693 (53.4434523310941%)**
  - SEGUNDA RONDA con la mitad de los datos
    - **27822 vectores de entrenamiento mal clasificados de 59693 (46.60848005628801%)**
    - **24601 vectores mal clasificados de 59693 (41.21253748345703%)**
  - TERCERA RONDA con datos completos

- 55388 vectores de entrenamiento mal clasificados de 119386 (46.394049553548996%)
- 26400 vectores mal clasificados de 59693 (44.22629118992177%)

Los resultados ya no varían tanto como en la primera prueba, pero aun el error es grande.

El código asociado a la sección del Perceptrón se puede visualizar en archivo perceptron.py .

b. Método de descenso de gradiente por lotes

Se probo el algoritmo con 200 iteraciones y cambiando el valor de la tasa de aprendizaje eta:

ETA	RESULTADO
.01	11491 vectores mal clasificados de 25000 (45.964%)
.001	11487 vectores mal clasificados de 25000 (45.948%)
.0001	13499 vectores mal clasificados de 25000 (53.996%)
.00001	9877 vectores mal clasificados de 25000 (39.507999999999996%)
.000001	9942 vectores mal clasificados de 25000 (39.768%)

Modificando el valor de penalización el mejor resultado dio con un valor de .000001.

El código asociado a la sección del método de gradiente por lotes se puede visualizar en archivo gradienteLotes.py .

c. Método de descenso de gradiente por estocástico

También llamado “aprendizaje en línea”. En este método la actualización de los pesos se realiza un vector a la vez. Esta versión suele tener mejor tiempo de convergencia.

- 11318 vectores mal clasificados de 25000 (45.272%)
- 11211 vectores mal clasificados de 25000 (44.844%)
- 11299 vectores mal clasificados de 25000 (45.196%)

Los resultados son muy similares en todas las corridas aun cuando deberían ser oscilatorios debido a que depende de como se vayan presentando los ejemplos de entrenamiento.

El código asociado a la sección del gradiente estocástico se puede visualizar en archivo gradienteEstocastico.py .

d. Feed-forward entrenada con Backpropagation

Se trata simplemente de un método descendente de gradiente, con el cual podemos minimizar el error cuadrático total de la salida calculada por la red.

El proceso de entrenamiento de una red backpropagation involucra tres fases: la feedforward de los patrones de entrenamiento de entrada, el cálculo y la propagación inversa del error asociado, y el ajuste de los pesos.

Se corrió el algoritmo con diferentes combinaciones de función de activación y de la variable solver con los siguientes resultados:

- Solver=lbfgs y activation=tanh

  9091 vectores mal clasificados de 23878 (38.07270290644107%)

- Solver=lbfgs y activation=relu

  9081 vectores mal clasificados de 23878 (38.03082335203953%)

- Solver=sgd y activation=tanh

  9249 vectores mal clasificados de 23878 (38.734399865985424%)

- Solver=sgd y activation=relu

  9279 vectores mal clasificados de 23878 (38.86003852919005%)

- Solver=adam y activation=tanh

  9173 vectores mal clasificados de 23878 (38.416115252533714%)

- Solver=adam y activation=relu

  9241 vectores mal clasificados de 23878 (38.700896222464195%)

El código asociado a la sección del algoritmo backpropagation se puede visualizar en archivo backpropagation.py .

6. Conclusiones

Algo que fue muy importante durante el proceso, fue la limpieza de datos, ya que eran muchos los datos faltantes en la columna SHOT_CLOCK, y datos que no tenían coherencia en la columna TOUCH_TIME ya que el tiempo no puede ser negativo.

En el clustering, el método de dendogramas daba a ver que los clusters estaría bien que fueran de 2 a 5, que al final son dos las clases de los datos usados. En isodata el resultado eran 3 clusters. El que mostró los peores resultados es el DBSCAN.

En cuanto a los métodos de clasificación, la mayoría de los métodos dieron resultados muy similares alrededor de 40% de error en todos hasta 45%. El que mejores resultados y con mayor estabilidad en cada corrida es el Feed-forward entrenada con Backpropagation que los resultados fueron del 38% de errores en todas sus combinaciones. Es probable que los errores altos de error en todos los métodos se deban a los valores atípicos, ya que se observó en el método DBSCAN que había bastantes. Esto puede solucionarse incluyendo otra ronda de limpieza de datos.

Son muchos los factores que pueden influir en si un tiro entra o no, como el animo del jugador, el defensor, si se plantó bien al tirar, el clima, lo que ceno el día anterior... se intuye que otra de las razones para que el error de clasificación fuera tan alto.

7. Referencias

- Kaggle. 2016. *NBA Shot logs*. [ONLINE] Available at: https://www.kaggle.com/dansbecker/nba-shot-logs/. [Accessed 8 December 2017].

- RSOTOC/PATTERN-RECOGNITION:

  rsotoc/pattern-recognition. [online] Available at: https://github.com/rsotoc/pattern-recognition/ [Accessed 8 Dec. 2017].