Arbres de decisió

Coneixement, Raonament i Incertesa

Enginyeria Informàtica 2021/2022 - UAB

Martí Caixal Joaniquet - 1563587

Ricard López Olivares - 1571136

Sergi Bons Fuses - 1571359

Introducció

La pràctica tracta de dissenyar, programar i testejar un arbre de decisió. Està dividida en 3 subapartats:

1) Dissenyar un arbre de decisió utilitzant diferents heurístiques (ID3, C4.5, Gini index). Per el moment no cal fer un preprocessat de les dades gaire avançat. Les mostres que tinguin valors nuls s'esborren de la base de dades. Per altra banda, els atributs amb dades continues són classificats segons intervals preestablerts de la mateixa mida.
   Finalment també es dissenya un K-Fold Cross Validation per avaluar les prediccions realitzades.

2) Tractament dels valors nuls de forma avançada, tant per la fase d'aprenentatge com de predicció. Ja no s'esborraran les mostres, sinó que s'aplicaran diferents mètodes per no perdre tantes dades.

3) Tractament d'atributs continus més idoni tenint en compte la informació que es guanya a l'hora de classificar cada valor.

Sense entrar en detall, la següent llista mostra el que s'ha programat:

- DecisionTree
- Preprocessat de dades
- Heurístiques: ID3, C4.5, Gini
- Cross validation, train_test_split i KFold
- Probabilistic Approach
- Tractament d'atributs continus: amb intervals i 2-way partitioning
- Random Forest

Si bé l'objectiu era dissenyar nosaltres un arbre de decisió des de 0, fem ús de les següents llibreries per facilitar la feina:

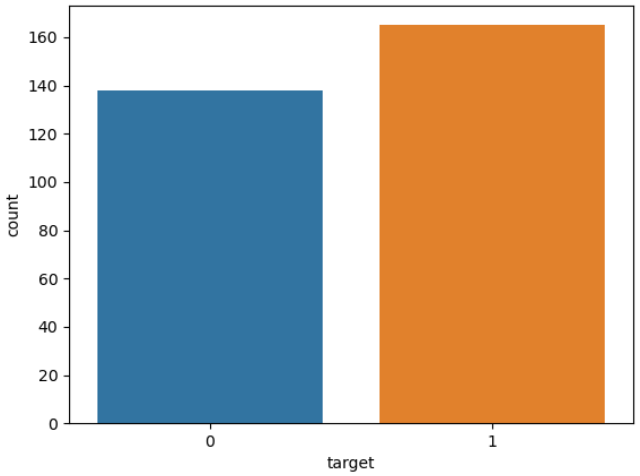
- Pandas: llegir i emmagatzemar en memòria el dataset
- Numpy: diversos càlculs matemàtics per l'heurística
- Scipy.stats: diversos càlculs matemàtics per fer la votació al Random Forest
- Seaborn: mostrar plots
- Matplotlib: mostrar plots
- Sklearn.metrics: extreure mètriques de rendiment
- sklearn.model_selection: per comparar la nostra versió amb una ja creada
- sklearn.tree: per comparar la nostra versió amb una ja creada
- PrintTree: mostrar per terminal un arbre

Exploratory Data Analysis

Abans de començar a dissenyar i implementar algosrimes de Machine Learning, sempre és recomanable veure amb quines dades s'haurà de treballar.

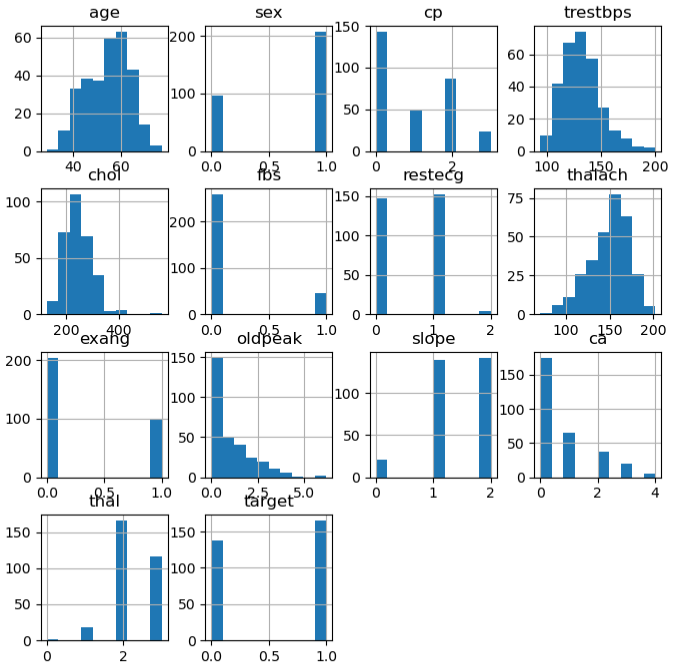
El dataset consta de 303 mostres amb dades per predir si un pacient té problemes de cor.

El primer punt a tractar és veure si la base de dades està balancejada. Significa que hi hagi, aproximadament, la mateixa proporció de les diferents classes objectius. El nostre cas està suficientment balancejat com per no suposar un problema.



Quan les dades no estan balancejades, el model acostuma a aprendre més sobre la classe majoritària i, per tant, fer més bones prediccions per aquesta classe que no pas per les altres. Un altre problema que es pot presentar és, que durant la fase de test, hi hagi moltes mostres d'una mateixa classe i les mètriques de rendiment puguin enganyar. Per afrontar aquest problema hi ha mètriques com el "Oversampling" i el "Undersampling" que permeten equilibrar les classes.

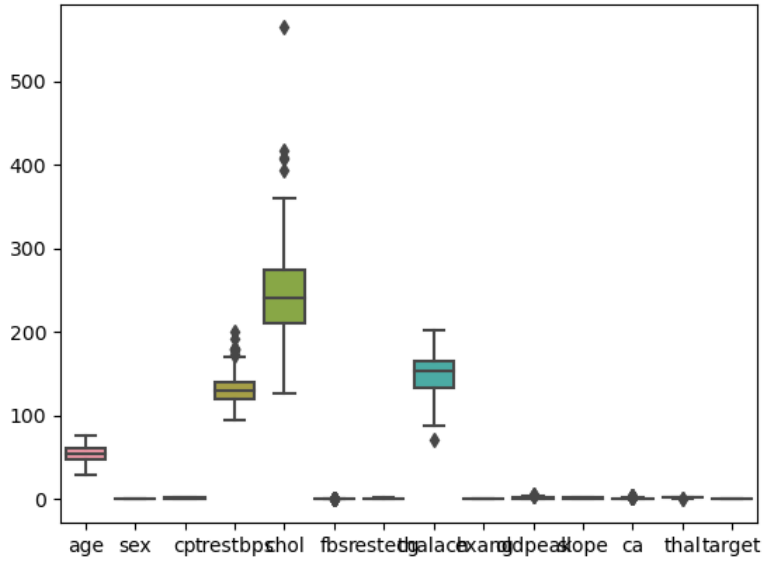
El següent diagrama permet veure de forma més detallada la distribució de cada atribut i quin tipus de valors hi ha.



El primer fet a destacar és que hi ha atributs categòrics i continus. Si els categòrics són idonis pels arbres de decisió, els continus, per exemple “trestbps”, són tot el contrari. Més endavant es comenta els problemes que poden comportar i com tractar-los.

L’histograma també permet veure si ha ha valors fora del seu rang corresponent. Per exemple, l’atribut “CA” només ha de tenir valors entre el 0 i el 3, però es veu que té alguns amb un 4. Aquests també es tractaran més endavant durant el preprocessat de les dades.

Amb un diagrama de caixes, tot i ser molt semblant a un histograma, es pot veure amb molta claredat els outliers de cada atribut. No són altre cosa que anomalies als valors, per exemple una persona amb alçada de 250 cm. També es tractaran durant el preprocessat de dades.

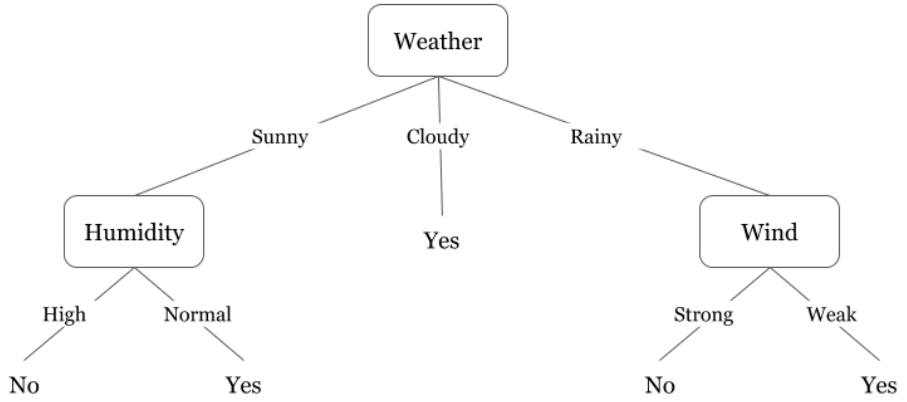


Apartat 1

Estructura de dades

El primer que s'ha hagut de decidir és com seria l'estructura de dades per guardar l'arbre de decisió que s'està creant. Al no haver de fer recorreguts complicats per l'arbre, que simplement és recórre'l de dalt a baix, hem decidit utilitzar diccionaris anidats.

Per mostrar-ho millor, a continuació es veu un arbre de decisió i com queda guardat a l'estructura de dades:




```
tree = \
{'Weather': {
    'Sunny': {'Humidity': {
                    'High' : 'NO',
                    'Normal': 'YES'
                }
    },
    'Cloudy' : 'YES',
    'Rainy' : { 'Wind': {
                    'Strong' : 'NO',
                    'Weak' : 'YES'
                }
    }
}}
```

Més endavant es veurà com aquesta estructura no és suficient i caldrà ser modificada.

Arbre de decisió

A l'hora de generar un arbre de decisió ens trobem amb 2 tipus d'atributs principals:

1) Discrets / categòrics: Com bé el seu nom indica, són atributs amb valors discrets o directament no numèrics. Són els més indicats pels arbre de decisió doncs són propensos a no generar gaires branques. Per exemple, si l'atribut és la direcció del vent, els valors poden ser Nord, Sud, Est, Oest.

2) Continus: Els valors d'aquests atributs són numèrics continus. Això significa que, al tenir valors infinits, l'arbre de decisió pot arribar a crear una nova branca a un atribut per cada mostra que utilitzi durant l'entrenament. Per seguir el mateix exemple anterior, ara els valors per la direcció del vent podrien ser en graus, minuts i segons.

Com ja es pot intuir, els atributs amb valors continus presenten un greu problema per la generació de l'arbre. El que s'ha de fer, doncs, és discretitzar aquests valors. En aquest primer apartat es fa un enfocament senzill i ràpid, basant-se en crear N intervals d'igual mida on es classifiquen els varis valors de l'atribut.

Un exemple clar a la nostra base de dades és el nivell de colesterol. Es té valors entre el 150 i el 300. Si es creen 15 intervals de mida 10, cada mostra es classificarà a un d'ells. Un cop realitzada aquesta classificació, es passa de tenir molts valors diferents a tenir-ne només 15, un nombre ja més acceptable. La qüestió és trobar el nombre d'intervals òptim, més endevant s'abordarà.

Les dades també es codifiquen per tal de no haver de tractar amb variables de tipus interval. Així doncs, a cada interval se li assigna un valor numèric únic començant pel 0.

Com a punt a destacar d'aquest primer apartat, cal parlar sobre com es decideixen algunes prediccions. De moment no s'està implementant cap tractament de valors nuls o erronis. Durant l'aprenentatge no es presenta cap problema, en canvi, durant les prediccions sorgeix un problema que s'ha de tractar. Hi ha valors de les mostres a predir que no existeixen a l'arbre de decisió. Quan s'arriba a un node i el valor no existeix com a branca, es decideix predir que té la enfermetat. Més endavant es veurà una altra aproximació a aquest problema i les avantatges i desavantatges que comporta.

Heurístiques

Per a trobar els atributs òptims sobre els quals basar el nostre arbre de decisió, necessitem especificar heurístiques que escullin els atributs que ens aportin més informació. Aquesta decisió es basa en el guany (gain) que ens aporta cada atribut, i ens permet generar arbres més bons que segurament també generalitzaran millor.

En el nostre programa les heurístiques que hem implementat es poden escollir passant el nom de l'heurística com a paràmetre per a l'atribut "heuristic".

Els algorismes disponibles són:

ID3

En aquest algorisme, escollim el millor atribut a cada iteració del programa. Per a fer això, avaluem la entropía dels atributs i el guany d'informació que ens proporcionen. Per a calcular aquest guany per cada atribut, restem l'entropia de l'atribut a l'entropia de la base de dades, i ens quedem amb l'atribut amb la menor entropia (és a dir, la major informació).
La fórmula del càlcul de l'entropia i el guany ve determinada per:

$$Gain(S,A) = -\sum_{x=1}^{m}\left(p(x) * log_2 p(x)\right) - \sum_{v \epsilon A} \frac{|S_v|}{|S|} * \sum_{x \epsilon v}\left(p(x) * log_2 p(x)\right)$$

on:

$$Entropy(S) = -\sum_{x=1}^{m}\left(p(x) * log_2 p(x)\right)$$

quan $x \epsilon S$, on S és la base de dades i x cada valor que agafa un atribut en concret, és a dir, cada classe.

$$Entropy(S,A) = \sum_{v \epsilon A}\left(\frac{|S_v|}{|S|} * \sum_{x=1}^{m}\left(p(x) * log_2 p(x)\right)\right) = \sum_{v \epsilon A}\left(\frac{|S_v|}{|S|} * Entropy(S_v)\right)$$

quan S es la base de dades, A un atribut concret, v un valor que agafa (es a dir una classe) i x tots els valors possibles de l'atribut

C4.5

C4.5 és la millora de l'algorisme ID3. Ara, a part de tenir en compte la informació que guanyem per cada atribut, tenim en compte les vegades que ens hem subdividit per a valorar millor el guany real de cada moment concret respecte al global.
En aquest cas, a cada iteració del programa calculem, per a cada atribut, el guany d'informació normalitzada de la divisió d'aquest. Aquest concepte es veu reflectit en la fórmula següent:

$$GainRatio(S,A) = \frac{Entropia(S) - Entropia(S,A)}{-\sum_{v \in A}\left(\frac{|S_v|}{|S|} log_2\left(\frac{|S_v|}{|S|}\right)\right)}$$

On S és la base de dades, A es l'atribut en concret d'aquesta iteració, i v són els valors que pot prendre aquest atribut (o dit d'una altra forma les classes possibles d'aquest atribut).

La part superior d'aquesta fórmula és la que fem servir per a calcular el guany, i la part inferior és el valor de la partició sobre la que es calcula el guany, aquest valor (split_info(S,A)) resulta en un valor entre 0 i 1, com més particions hi hagi, més gran serà, i menys rellevant serà el seu gain ratio.

En el cas de la implementació de l'algorisme s'ha de tenir en compte que el split_info mai pot ser de 0, ja que probable que generi errors en el programa, per a evitar aquest escenari, quan obtenim un 0 el convertim a un valor que anomenem "eps", que no és més que el valor més baix possible, un substitut del 0 perquè a casos pràctics resulti igual.

Gini

En aquest cas, a cada iteració del programa calculem, per a cada atribut, la distribució de les classes d'aquell atribut, utilitzant l'índex de Gini.
En el nostre context, aquest index correspon a un valor entre 0 i 1, on un valor de 0 vol dir que només apareix un valor en aquell atribut, i un valor de 1 vol dir que hi ha infinites tuples on cada tupla té un valor diferent (es a dir, disposem d'infinites classes i cap classe es repeteix). Aquest concepte es veu reflexat en la següent formula:

$$Gini(S,A) = 1 - \sum_{v \in A}\left(\frac{|S_v|}{|S|}^2\right)$$

On S és la base de dades, A es l'atribut en concret d'aquesta iteració, i v són els valors que pot prendre aquest atribut (o dit d'una altra forma les classes possibles d'aquest atribut)

Mètriques a utilitzar

Si bé el dataset amb el que s'està treballant ara està suficientment balancejat, res ens assegura que al moment de la veritat les prediccions que s'hagin de fer també estaran balancejades. És molt important, doncs, saber quines mètriques utilitzar. Per fer l'explicació més senzilla, es farà ús de la taula inferior.

El punt més destacable d'aquestes prediccions és que de 30 persones que tenen la malaltia de cor, només se'n ha predit 5. Clarament són uns resultats molt dolents, però algunes mètriques diuen el contrari

Malaltia?	**Actual YES**	**Actual NO**	**Total**
Predicted YES	TP = 5	FP = 0	5
Predicted NO	FN = 25	TN = 270	295
Total	30	270	300

Accuracy	92%
Precision	100%
Recall	16%
F1 Score	27%

L'accuracy té en compte el nombre de prediccions correctes respecte el total de prediccions. És una mesura bastant generalista i només funciona quan tenim dades balancejades, pot servir per fer-se una idea bàsica de si un model és bo o no. Si més no, no és gens idònia pels casos amb dades no balancejades. L'exemple superior mostra una Accuracy molt alta, però ja s'ha vist que les prediccions realment no són òptimes.

Una altra mètrica que, tot i ser més adequada quan tenim dades no balancejades, en aquest cas no seria de gaire ajuda és Precision. Aquesta dada diu de totes les prediccions marcades com a "Yes", quantes s'han fet correctament. La raó per la qual no ens és d'ajuda és que, segons el nostre criteri, és més valuós estar segur que si diem no té malaltia, realment no la té. En aquest exemple, només troba 5 pacients malalts, sent els 5 realment malalts. Té una precisió, doncs, del 100%, però seguim sense poder predir la majoria dels pacients que realment estan malalts.

La tercera mètrica a parlar és el Recall. Aquesta ens permet saber de tots els pacients malalts, quants s'han predit correctament. Òbviament aquesta mètrica és la més important ja que ens permet saber si hi haurà gaire pacients malalts que el model es pensarà que no ho estan.

Per últim, queda analitzar la F1 Score. Aquesta mètrica es basa en la mitjana harmònica del Precision i del Recall. Així aconseguim que només sigui alta si les altres 2 mètriques són altes. Per exemple, si Precision és 1 i Recall és 0.01, el F1 score és 0.02. En dades no balancejades, aquesta mètrica és molt important i sempre substituirà al accuracy.

Havent parlat de les 4 mètriques principals, les que ens hem de fixar per aquest dataset són Recall i F1 Score.

K-Fold Cross-Validation

Explicació

Quan s'entrena un model sense tenir en compte la variació de les dades, com estan ordenades, si estan balancejades o no, és molt possible que els resultats obtinguts no representin realment al comportament que tindrà el predictor a l'hora de la veritat.

Hi ha, doncs, una tècnica anomenada Cross Validation que permet avaluar de forma més òptima i fidel a la realitat un conjunt de dades, evitant així generalització quan s'entrena sobre un sol conjunt.
Es basa en entrenar diversos models en subconjunts de les dades disponibles i avalua cada un d'ells en el subconjunt complementari restant. Mitjançant aquesta tècnica, hi ha moltes probabilitats que puguem detectar overfitting amb facilitat.

Una manera d'aplicar el Cross Validation és la coneguda per K-Fold Cross Validation. És la més popular d'entre elles i acostuma a donar un resultat bastant acceptable per la gran majoria de casos.

Abans de començar a explicar la tècnica, cal tenir clar una regla molt important. Mai s'ha de mesclar les dades Training i Test. Per tant, el pas previ és isolar les dades de Test i només utilitzar-les per l'avaluació final. Les dades que resten a Training seran les usades per aplicar el F-Fold Cross Validation.

Es comença partint les dades de forma aleatoria en K subconjunts iguals. El primer subconjunt és l'utilitzat per Testing, i els K-1 restants són utilitzats per entrenar el model (Training). Finalment el model és posat a prova amb les dades Testing.
Aquest procés es repeteix K vegades, cada cop emprant un subconjunt diferent per Testing. D'aquesta manera cada subconjunt té la mateixa oportunitat de ser inclòs al subconjunt d'entrenament.

El valor de K acostuma a ser entre 3 i 5 per la majoria de casos. Si més no, pot ser estès fins a valors com 10 0 15, tenint com a punt negatiu que el cost computacional incrementa considerablement.

Implementació

Per programar un cross validation s'ha necessitat les 3 funcions detallades a continuació:

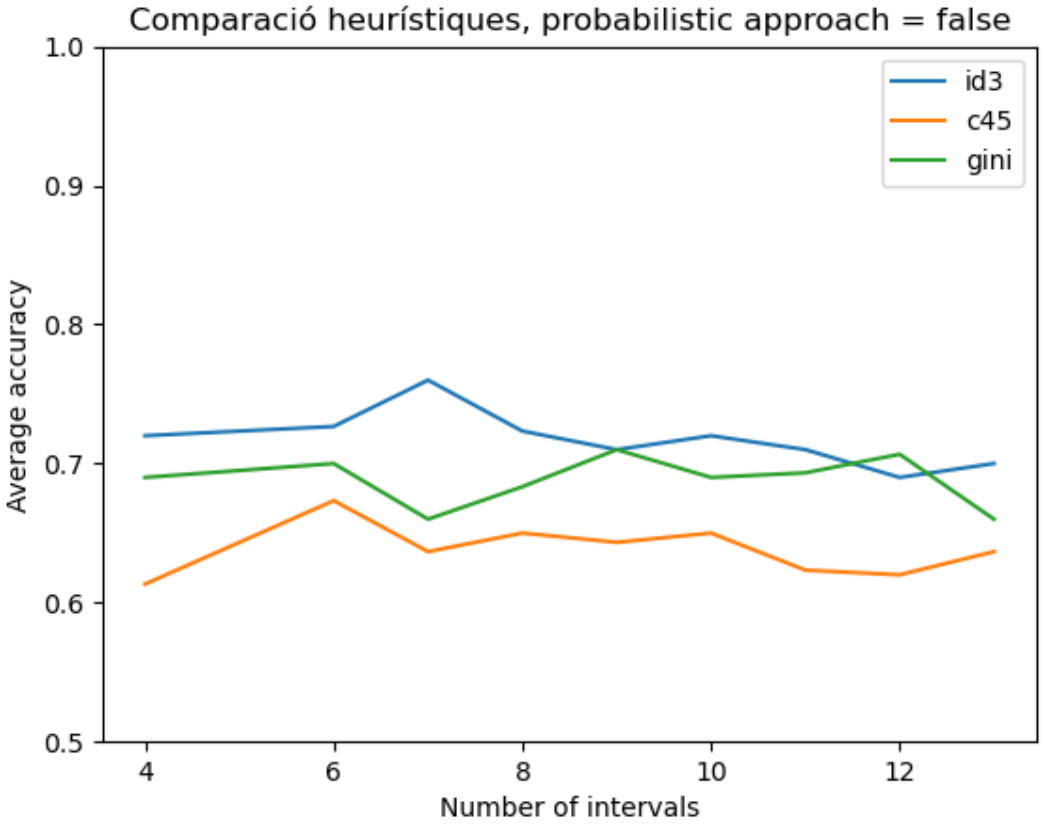
- `trainTestSplit(df, trainSize=0.8, shuffle=True):`
  És la utilitzada per separar el dataset en dos subconjunts. Com a punts a destacar, accepta un paràmetre per fer "shuffle" abans de separar el conjunt.

- `kFoldSplit(df, n_splits=10):`
  La funció permet crear K subconjunts, retornarnt-los tots en una llista. És la utilitzada durant el cross validation.

- `crossValidation(model, df, n_splits=10, shuffle=True):`
  És la funció principal per executar el cross-validation. Com a paràmetres rep el model a utilitzar, el número de folds que ha de fer i un flag per si s'ha de fer "shuffle" al dataset.
  A part de fer la funció principal d'executar K vegades el model amb subconjunts diferents per Train i Test, també guarda els resultats. Retorna un diccionari on les claus són el nom de la mètrica i els valors una llista on cada element és el resultat d'una execució. És realment útil per més tard poder crear plots i mostrar els resultats fàcilment.

Comparació de resultats

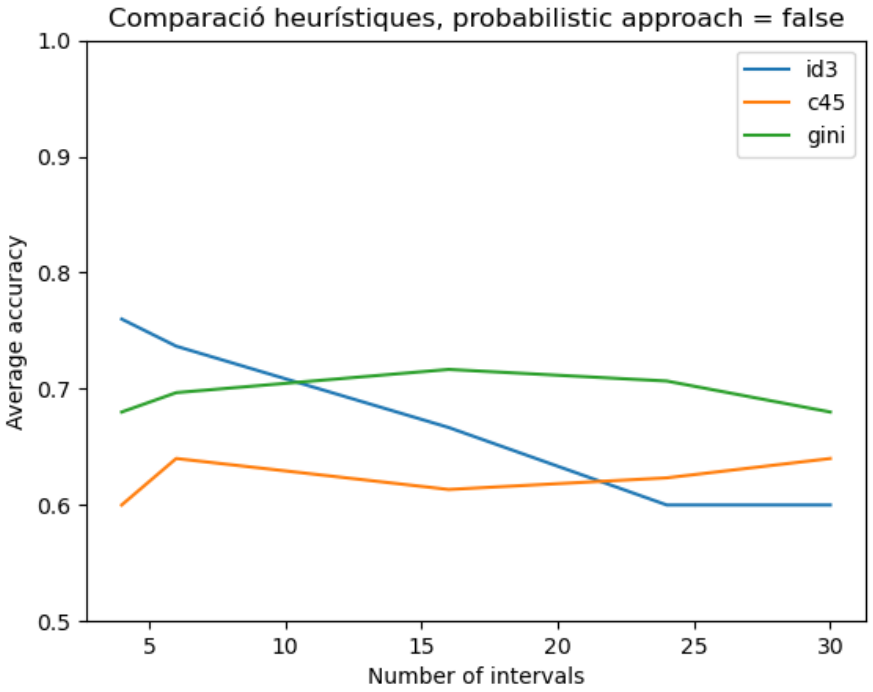
A continuació es mostren un seguit de proves portades a terme per avaluar cadascuna de les heurístiques. El valor de cada interval i heurística és la mitjana de fer un cross validation amb 10 folds.



En base als resultats que hem obtingut, podem realitzar diverses observacions, per exemple, si ens fixem en la imatge anterior observarem que id3 sembla donar millor resultats tot i ser l'algorisme més senzill, però s'ha de tenir en compte que estem visualitzant poca diferencia en el nombre d'intervals dels atributs.

És a dir, com que estem observant la mostra molt discretitzada, i com que les altres heuristiques donen resultats més precisos, son menys confiables en discretitzacions que abarquen grans rangs.

Sabent això, si ampliem el nombre d'intervals en els quals discretitzem (augmentant la qualitat de la previsió, i reduint el rang entre intervals), obtenim uns resultats bastants diferents:



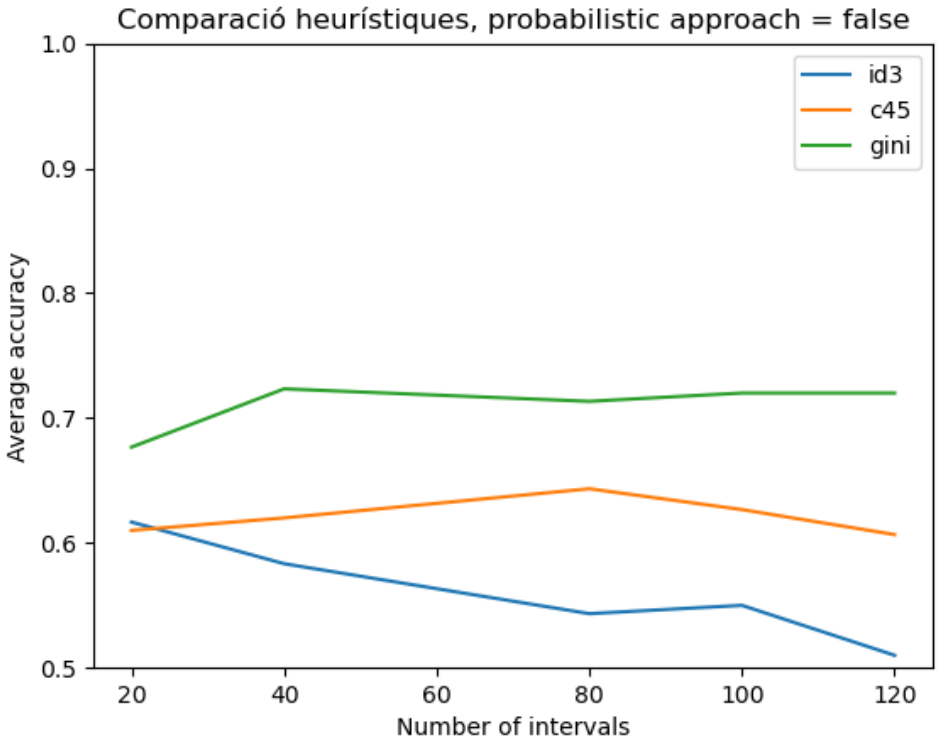
Es pot observar que id3 decau, mentre que gini i c4.5 es mantenen estables, és a dir, les prediccions han de ser molt més precises.

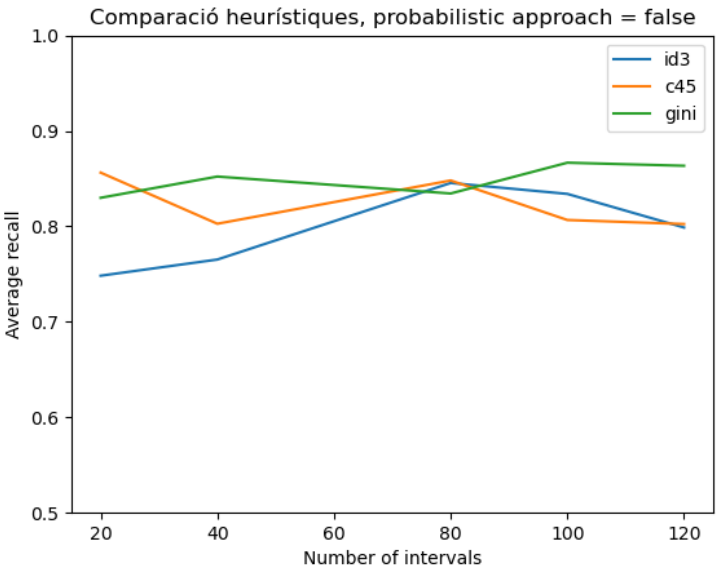
En el cas de c4.5 aquesta precisió es deu al fet que cada atribut és seleccionat en base a la split_info, és a dir, la informació que dona els atributs té en compte la quantitat de valors que es classifiquen, permetent prioritzar els que valoren millor la majoria de valors, donant una predicció millor per a casos amb moltes possibles combinacions d'atributs.

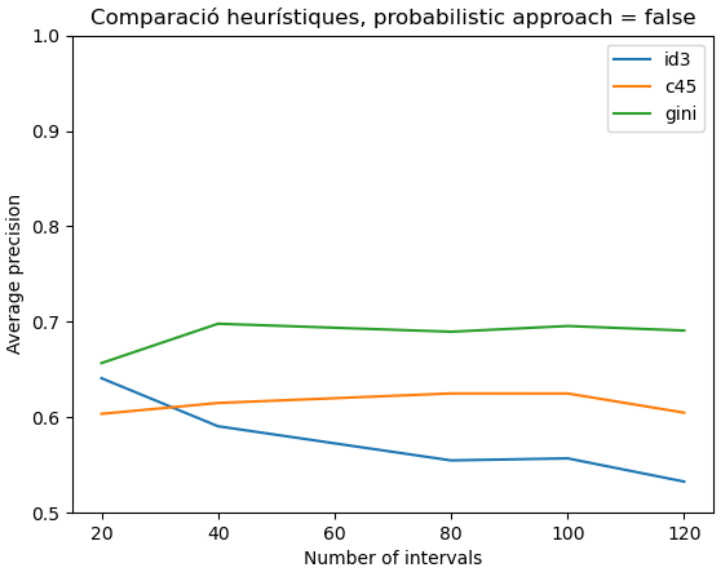
En el cas de heurística que implementa l'índex de Gini, és molt semblant a l'anterior, ja que a major nombre d'elements d'una classe, més precís és l'índex, i com que es tracta d'una mesura de desigualtat és extrapolable a problemes amb major nombre d'intervals.

En base a aquestes dades, podem hipotetitzar que Gini es el millor mètode per al nostre problema, o per al nostre approach al mateix, tot i que c4.5 sembla donar també resultats bastant estables. Si més no, quan s'utilitzen pocs intervals seria recomanable l'heurística id3 ja que dona resultats més clarament diferenciats.

Finalment. Per comprovar que les nostres observacions són correctes, comprovem encara més intervals, i n'avaluem l'accuracy, la precision i el recall.



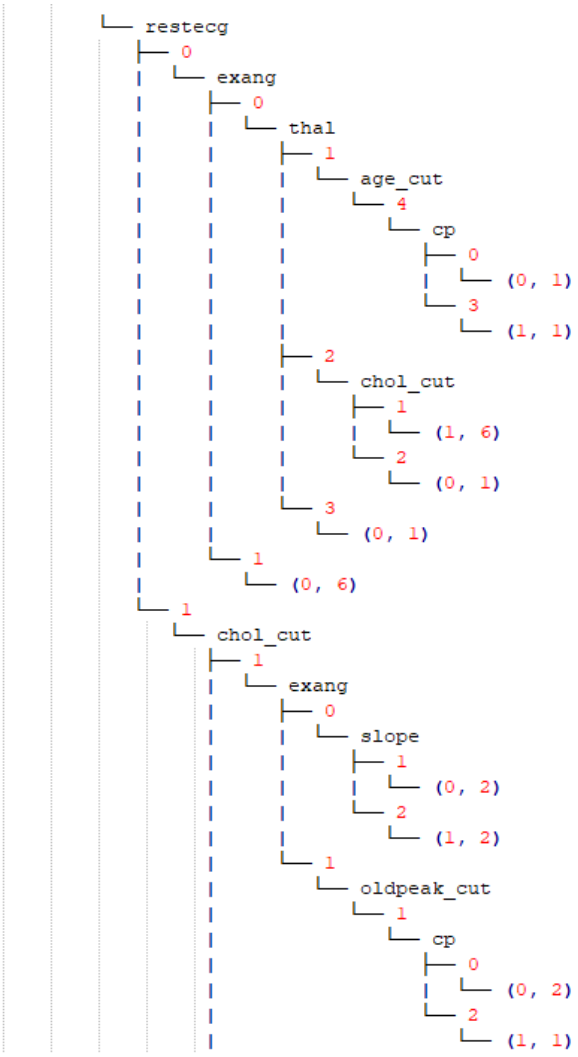




Val la pena mencionar que la diferencia entre puntuacions de precision i recall es deguda al fet de que tots els pacients que no es poden decidir emprant l'arbre els preveiem per defecte com a positius, així que el recall i la precision es veuen distorsionats, afavorint el recall.
Aquest problema el resolem posteriorment amb un probabilistic approach.

Visualització de l'arbre

Degut a que l'arbre resultant té unes dimensions molt elevades, no es pot mostrar sencer al document. A continuació es mostra una part d'ell.



Al tenir valors numèrics, ja sigui perquè s'han codificat a posteriori o perquè originalment són així, ens permet veure molt bé la diferència entre nodes i branques.

A cada fulla es troba la classe resultant junt amb el nombre de mostres que s'hi han classificat durant el Training.

S'inclou un arxiu txt amb l'arbre generat durant l'apartat 3

Apartat 2

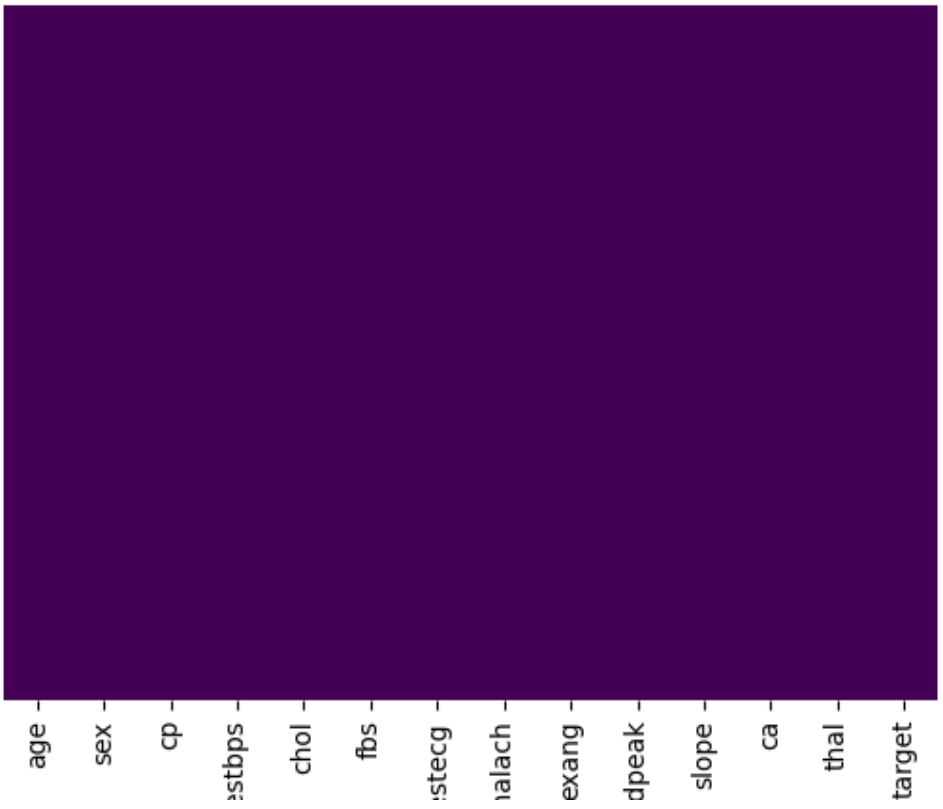
Fase aprenentatge

Fins ara s'han descartat totes les mostres que presenten problemes de valors. El problema és que es perd tota una mostra quan només hi ha un atribut erroni. A continuació s'explica com tractar cada un dels problemes per tal de no perdre tant informació.

Valors nuls

El dataset amb el que es treballa no conté ningun valor nul. Si més no, aquest problema es pot tractar
de dos maneres depenent del tipus d'atribut.

Pels categòrics s'omple els valors nuls amb la moda de l'atribut. Per altra banda, pels continus s'omple el valor amb la mitjana.



Plot característic per mostrar valors nuls, apareixent en groc si hi ha algun

Valors no vàlids

Un altre problema més difícil de detectar són els valors existents però amb dades no vàlides. Un clar exemple és una persona amb 348 anys.

El nostre dataset conté valors d'aquest tipus. Per solucionar-ho s'ha fet un filtre mitjançant la llibreria "Pandas" de totes les mostres que tenen valors fora del rang especificat a la descripció de les dades.

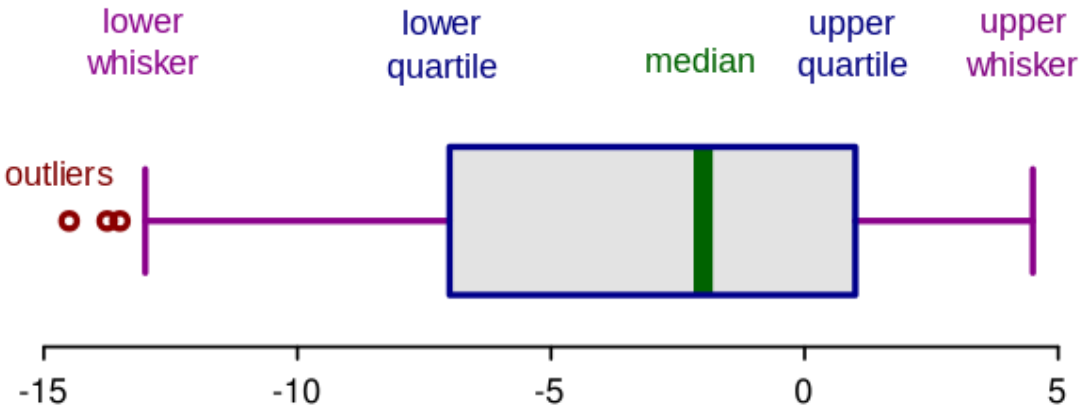
Les dades errònies es solucionen igual que anteriorment, amb la mitjana i la moda.

Tant el subapartat anterior com aquest estan tractats a la funció:

```
fixMissingAndWrongValues(df)
```

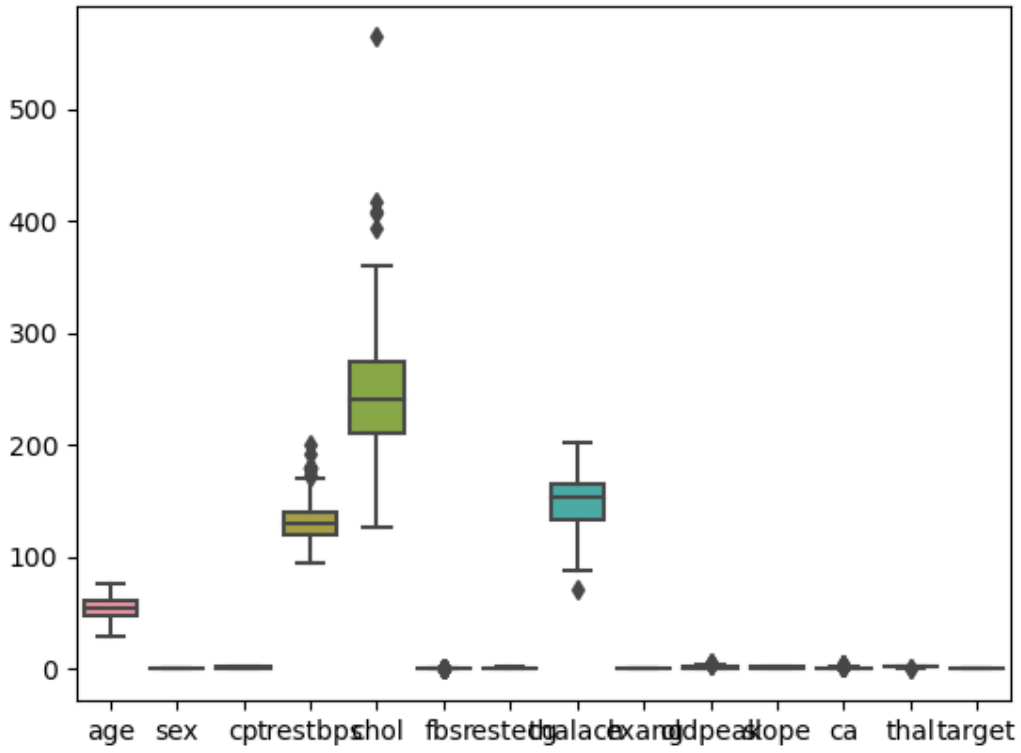
Outliers

Per últim, cal tractar també les anomalies. Tot i haver-hi moltes maneres de fer-ho, pel nostre programa s'ha decidit trobar-les mitjançant la amplitud interquartílica (IQR). Per entendre-ho ràpidament, són els valors que queden fora dels bigotis en un diagrama de caixes

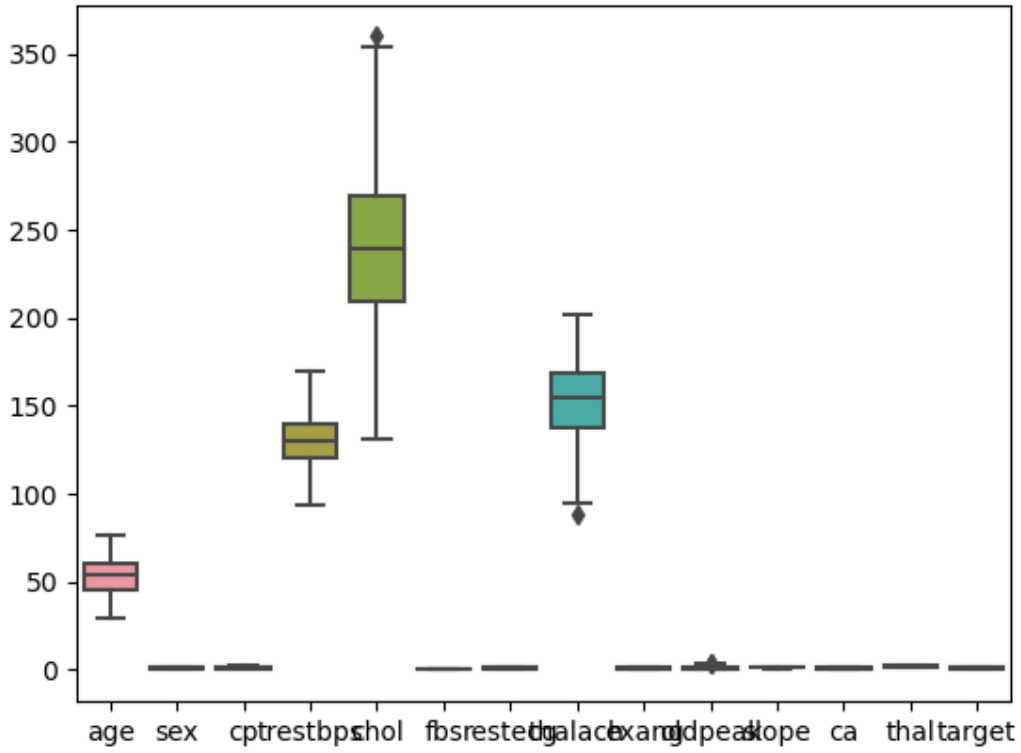


S'ha creat la funció `detectOutliers(df)` que retorna una llista amb els índex de les mostres amb anomalies. Un cop es té aquests índexs, es pot decidir eliminar-los o modificar els valor amb la moda o mitjana.

El següent diagrama de caixes mostra les dades sense haver tractat els outliers



Un cop havent-los tractat, queda així



Tot i semblar que encara hi ha outliers, els que apareixen són els nous després d'haver eliminat els originals. Es pot comprovar com el rang d'alguns atributs ha disminiut.

Fase predicció

Explicació

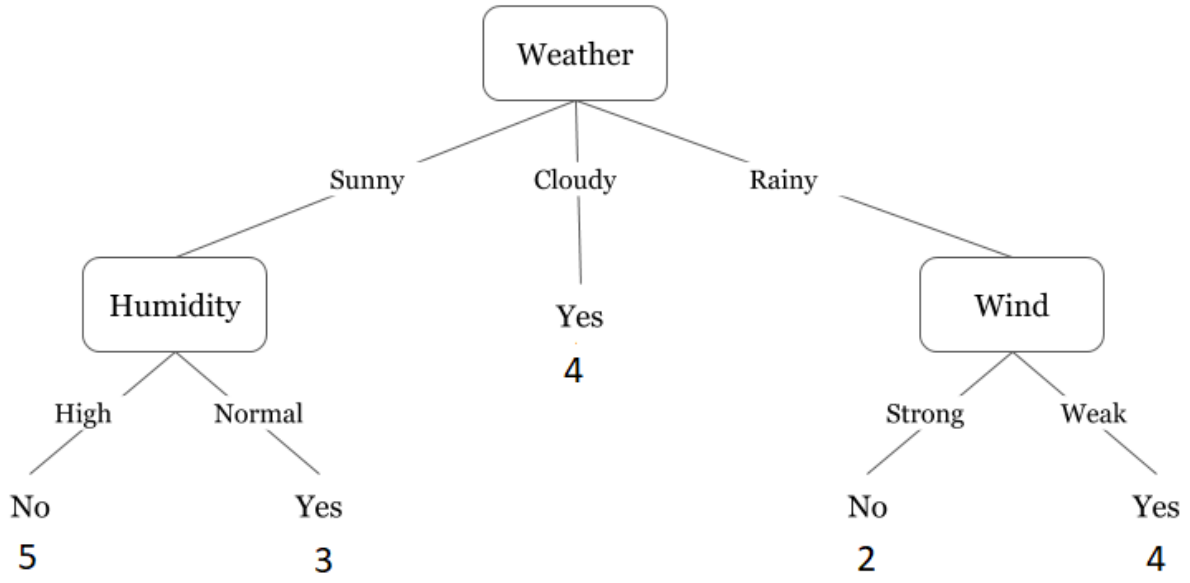
Al subapartat anterior s'ha vist com els valors nuls poden afectar durant la fase d'aprenentatge del model i els inconvenients que pot comportar. La fase de predicció també és sensible als valors de les mostres i cal aplicar un seguit d'accions per afrontar els problemes. La gran diferència és que, mentre a la fase anterior es modificaven les mostres, ara no es fa.

El principal problema durant aquesta fase es troba quan s'ha de predir una mostra amb valors que mai han estat vistos per l'arbre de decisió. Per posar un exemple fàcil, un arbre de decisió té un node "Temps" del que surten les branques "Sol" i "Pluja". La mostra que s'ha de classificar té per l'atribut "Temps" el valor "Neu". L'arbre no és capaç d'arribar a cap fulla ja que no té cap branca pel valor "Neu".

Fins ara, aquest problema s'ha tractat dient que el pacient té la enfermetat, ja que davant d'incertesa, és millor predir que té enfermetat que no pas que no la té.

El nou mètode aplicat en aquest apartat el "probabilistic approach". Sempre que s'arriba a un node pel qual no es pot seguir per ninguna branca, es pot fer un càlcul de probabilitat de ser una classe o una altra. Per entendre-ho més bé, es pot veure el mateix arbre utilitzat abans d'exemple. Ara, si més no, es té a cada fulla quantes mostres de l'entrenament hi estan classificades.

Seguim amb la mostra on l'atribut "Temps" té el valor de "Neu". Com que no podria seguir per cap branca, caldria explorar tots els camins possibles fins arribar a totes les fulles. Un cop haguem arribat a totes, caldrà veure quin resultat apareix més vegades i definir-lo com a predicció final. En aquest exemple hi hauria 7 NO i 11 SÍ, classificant-se com a SÍ.



També s'ha hagut de programar el cas en que l'arbre no tingui ningun valor existent per l'atribut "Temps", però sí tingui un valor existent per "Vent". En aquesta situació, quan arribi al node "Vent" només ha de seguir pel camí que li correspon segons el seu valor. Suposant que "Vent" és "Strong", la predicció quedaria 7 NO i 8 SÍ, tornant a sortir SÍ, però ara amb menys seguretat.

Veient aquest comportament i sabent la importància de saber quan un pacient realment està malalt, pot no ser el més idoni fer les prediccions d'aquesta manera. A l'últim exemple s'ha classificat com a "SÍ" tenint només un 8/15 de probabilitats d'encertar. Caldria, doncs, trobar un equilibri entre falsos positius i la importància d'enviar un pacient a casa quan realment té la malaltia.

Com s'ha vist a aquest apartat, si bé les mètriques milloren, mèdicament parlant pot ser més útil directament classificar com a positius tots els que tinguin incertesa. En canvi, si es donés la mateixa importància a ambdues classes, aquest mètode ofereix una gran millora.

Implementació

Anteriorment s'ha comentat que l'estructura per guardar l'arbre s'hauria de modificar. És necessari per poder guardar el número de mostres del Training que arriben a cada fulla. Després d'implementar aquesta funció, l'arbre queda estructurat de la següent manera:

```
tree = \
{'Weather': {
    'Sunny': {'Humidity': {
                    'High' : ('NO', 5),
                    'Normal': ('YES', 3)
                }
    },
    'Cloudy' : ('YES', 4),
    'Rainy' : { 'Wind': {
                    'Strong' : ('NO', 2),
                    'Weak' : ('YES', 4)
                }
    }
}}
```

Les fulles, en comptes de guardar únicament la predicció, ara també guarden en forma de tupla el número de mostres que s'han classificat a la fulla durant el Training.

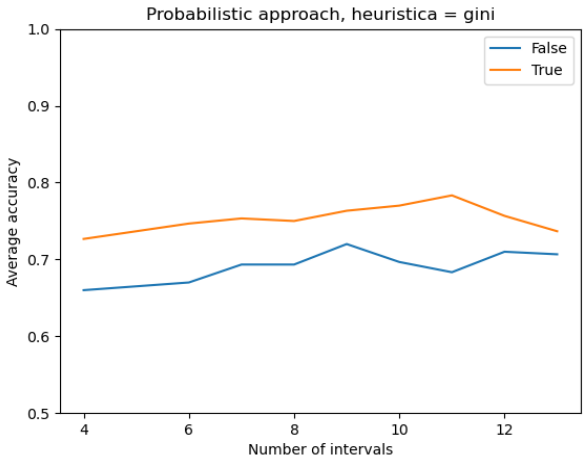
Es pot indicar al model utilitzar aquesta tècnica mitjançant un hipermaràmetre:

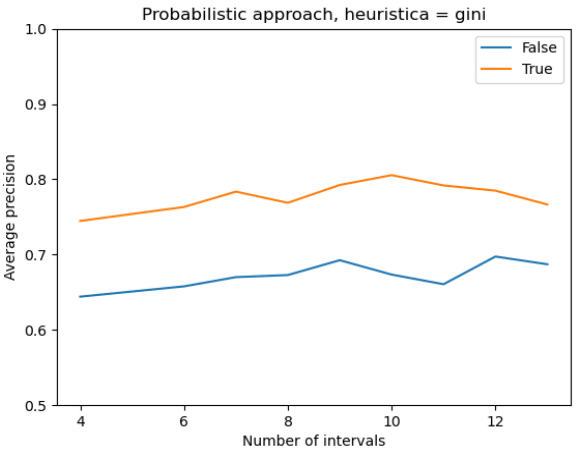
```
DecisionTree(enableProbabilisticApproach=True)
```

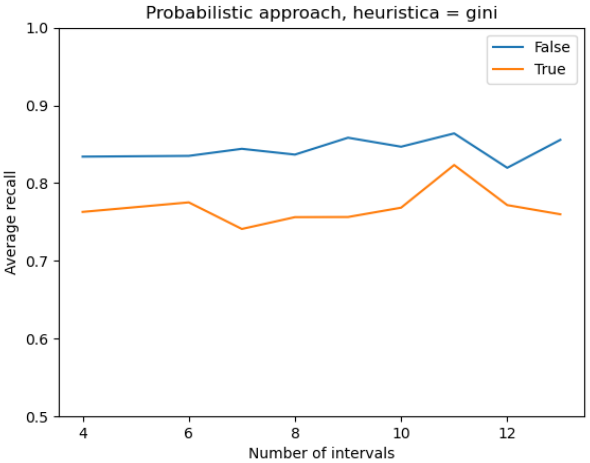
Quan té el valor True, en trobar-se amb un valor no existent comença a explorar els possibles camins i calcula la probabilitat final de ser un resultat o l'altre.

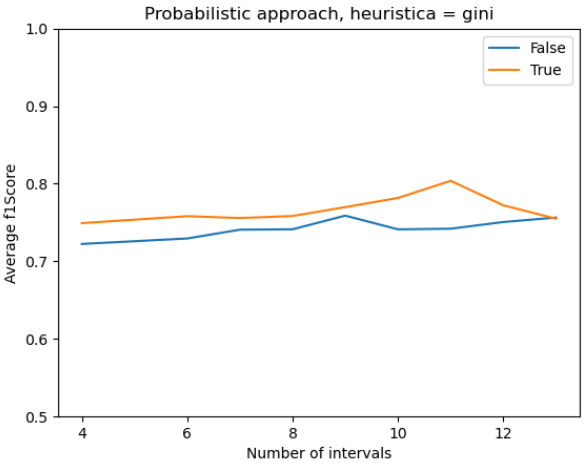
Resultats

Les següents gràfiques mostren el resultat d'aplicar el "probabilistic approach". S'observa com el resultat millora per totes les mètriques, tret de Recall. La raó és la explicada anteriorment, utilitzant el nou mètode les prediccions milloren en general. Tot i això, abans es classificaven totes les indecises com a "malalts" i feia pujar molt la mètrica recall. Com que prèviament s'ha decidit que la mètrica Recall és la més important en un predictor d'aquest tipus, aquesta millora no és útil.



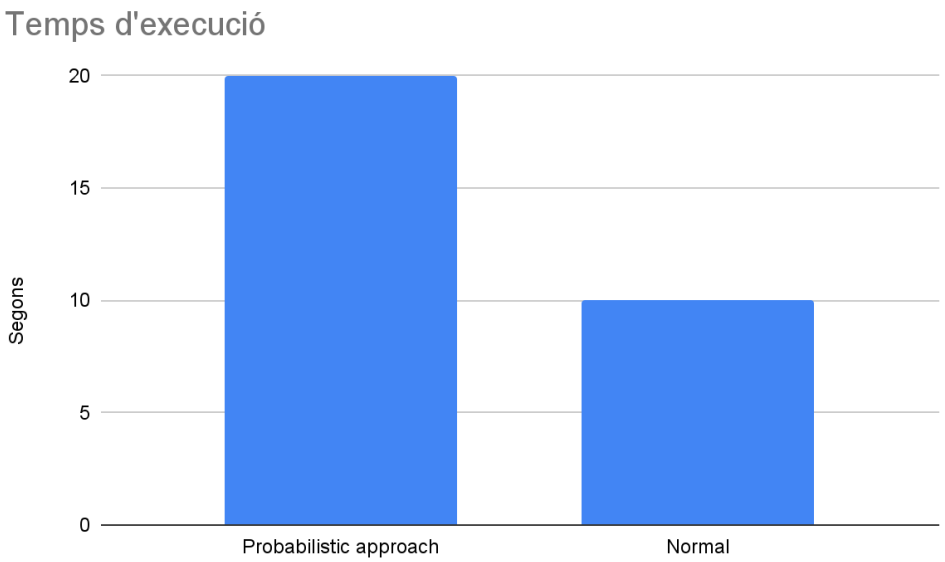






Per últim també cal parlar dels temps d'execució. Com es veu a la gràfica inferior, utilitzant la nova tècnica es triga molt més a obtenir les prediccions.

Aquest increment de temps es deu al fet d'haver d'explorar varis camins quan s'arriba a un node on el valor de l'atribut no correspon a cap branca.



Anotació: ambdues execucions són el resultat d'un cross validation amb 10 folds i classificant els atributs continus en 7 intervals d'igual mida.

Apartat 3

Explicació

Fins ara s'ha estat discretitzant els atributs continus en N intervals de la mateixa mida. Una altra tècnica per tractar valors continus es l'anomenada 2-way partitioning. El principi d'aquest mètode és trobar un punt de partició dels valors de l'atribut que ens aporti el màxim guany d'informació. Així s'aconsegueix que el node generat a partir de l'atribut només tingui 2 branques.

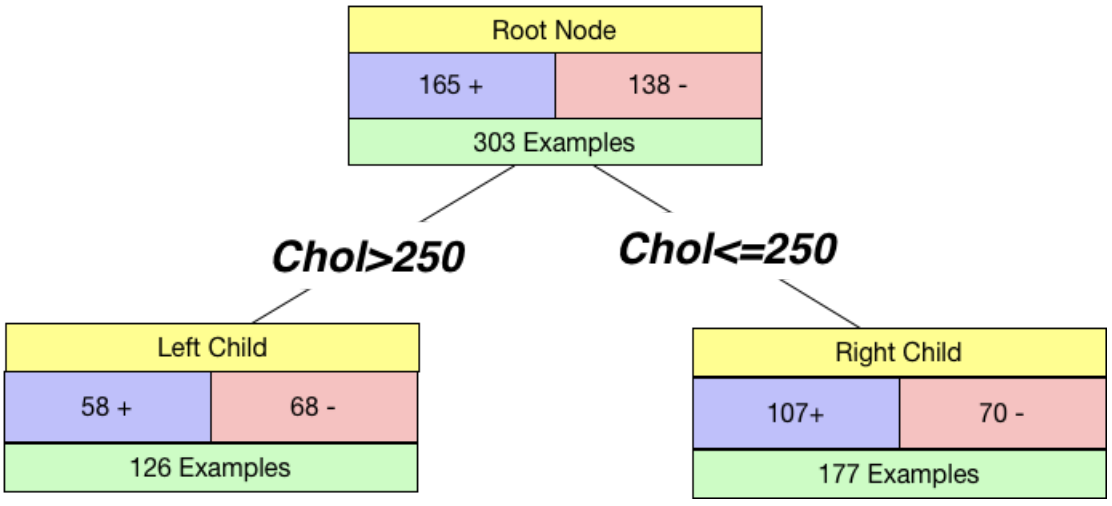
Per trobar el punt de partició cal seguir els següent passos:

1) Ordenar tots els valors de l'atribut de forma ascendent

2) Buscar els punts intermitjos de tots els valors. Si per exemple es té {0, 2, 4}, els punts intermitjos són {1, 3}

3) Per cada punt intermig, separar les mostres ens dos grups (més petites que el punt intermig, i més grans que el punt intermig). Avaluar l'entropia de la classe objectiu per la separació feta.

4) Quedar-se amb el punt intermig que proporciona l'entropia més baixa, és a dir, el màxim guany d'informació respecte la classe objectiu



Entropia ponderada pel punt intermig 250 = 0.97.

Implementació

A l'hora d'implementar-ho, ràpidament s'ha vist que si el nombre de valors és molt elevat el temps d'execució creix.

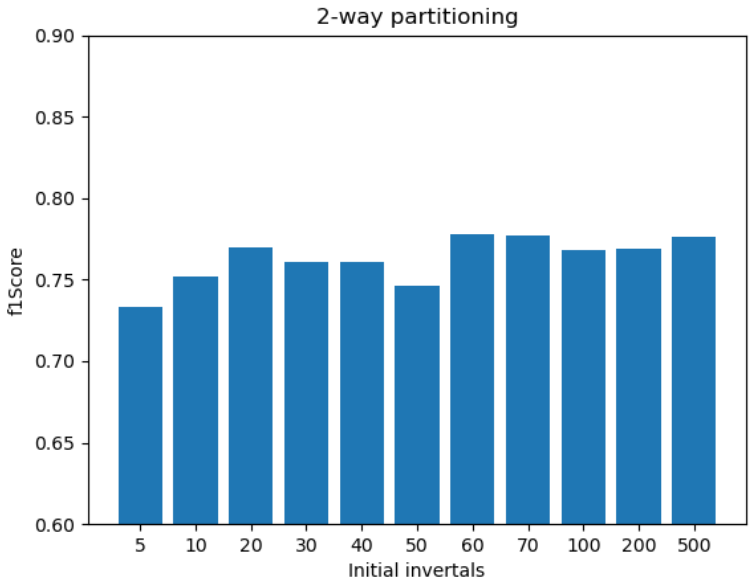
Per intentar reduir-lo, la funció "TwoWaySplit" accepta un hipermaràmetre "initialIntervals". Aquest valor permet preestablir uns intervals inicials des dels quals buscar un punt intermig òptim. D'aquesta manera, si hi ha 1000 valors i per paràmetre es passa un 100, es passarà a tenir 100 números des dels quals buscar el millor punt intermig.
Per evitar inconsistències, quan el valor de l'hiperparàmetre és més gran que el nombre de valors únics per un atribut, el màxim número d'intervals serà igual al nombre de valors únics.

Un valor molt gran permetrà trobar un punt intermig més òptim, però triga més a executar. En canvi, un valor petit no podrà arribar a un punt òptim que no pertany a les particions establertes pels intervals, però trigarà menys a executar.

De totes maneres, s'ha vist que el temps d'execució pel "TwoWaySplit" es pot menysprear degut a que després el model triga molt més a executar-se.

Resultats

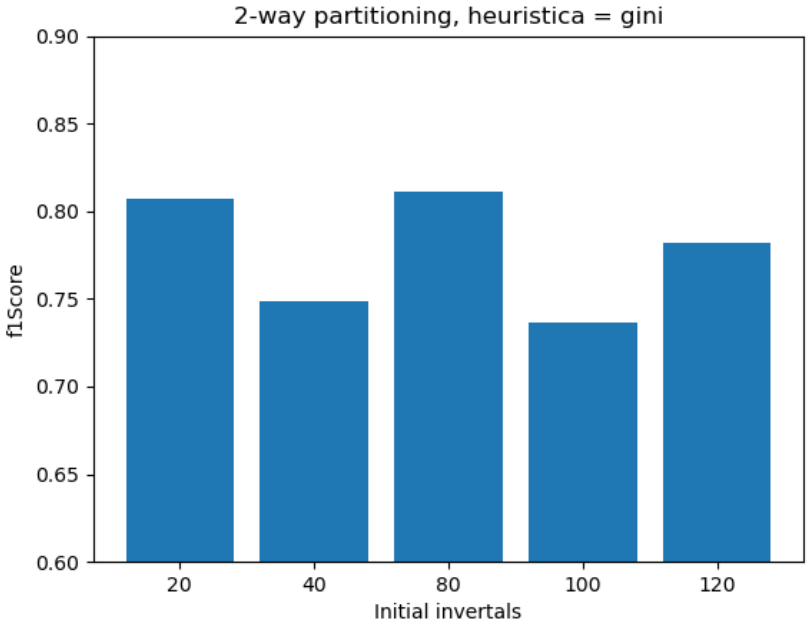
Tenint una base de dades amb 303 mostres, es veu clarament que a partir d'un cert valor d'intervals inicials el resultat s'estanca, arribant a un punt on busca tots els punts mitjos dels valors. Cal destacar, també, que de les 303 mostres, moltes tenen atributs repetits i per tant hi ha menys valors únics.



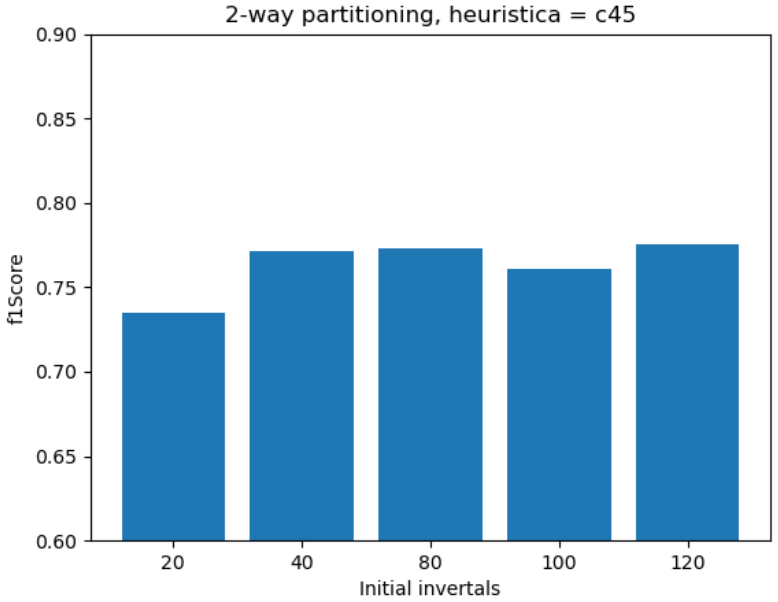
Anotació: Totes les execucions són el resultat d'un cross validation amb 10 folds, arbre generat utilitzant el mètode Gini

Comparant els resultats amb els mètodes anteriors, les mètriques segueixen sent molt similars, amb certes diferències.

Per exemple, en el cas de gini, els resultats són poc estables depenent de les particions que fem, ja que gini principalment divideix a partir de a la diferència entre la distribució de classes de cada atribut, però com que la informació és molt similar, el que fa que el criteri de decisió de gini es torna més difús.

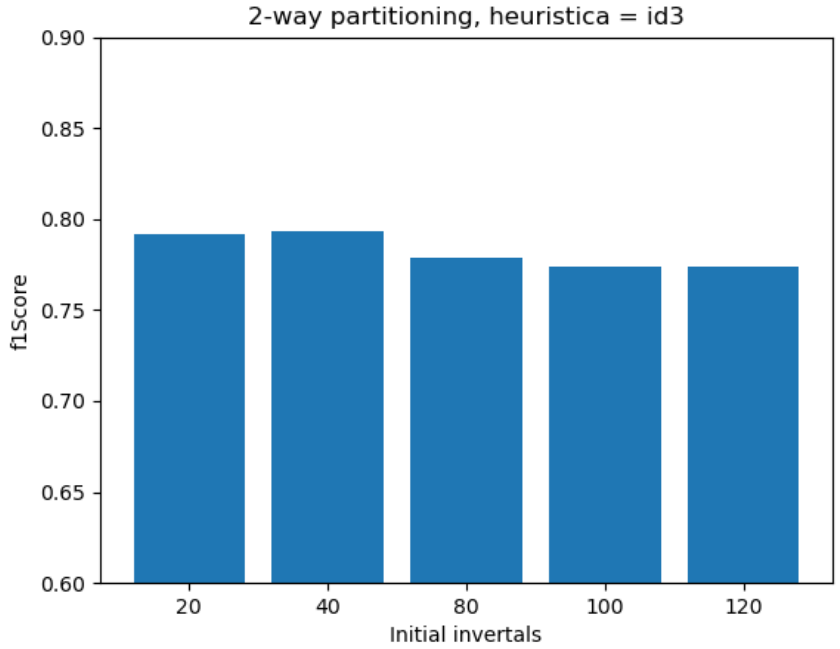


En el cas de c4.5, sí que obtenim un benefici més estable ja que tenim en compte el nombre de dades que tenim per partició, i la distribució de la mateixa té menys pes, de manera que només afegim capes i no tenim problemes amb el funcionament concret de l'algorisme.

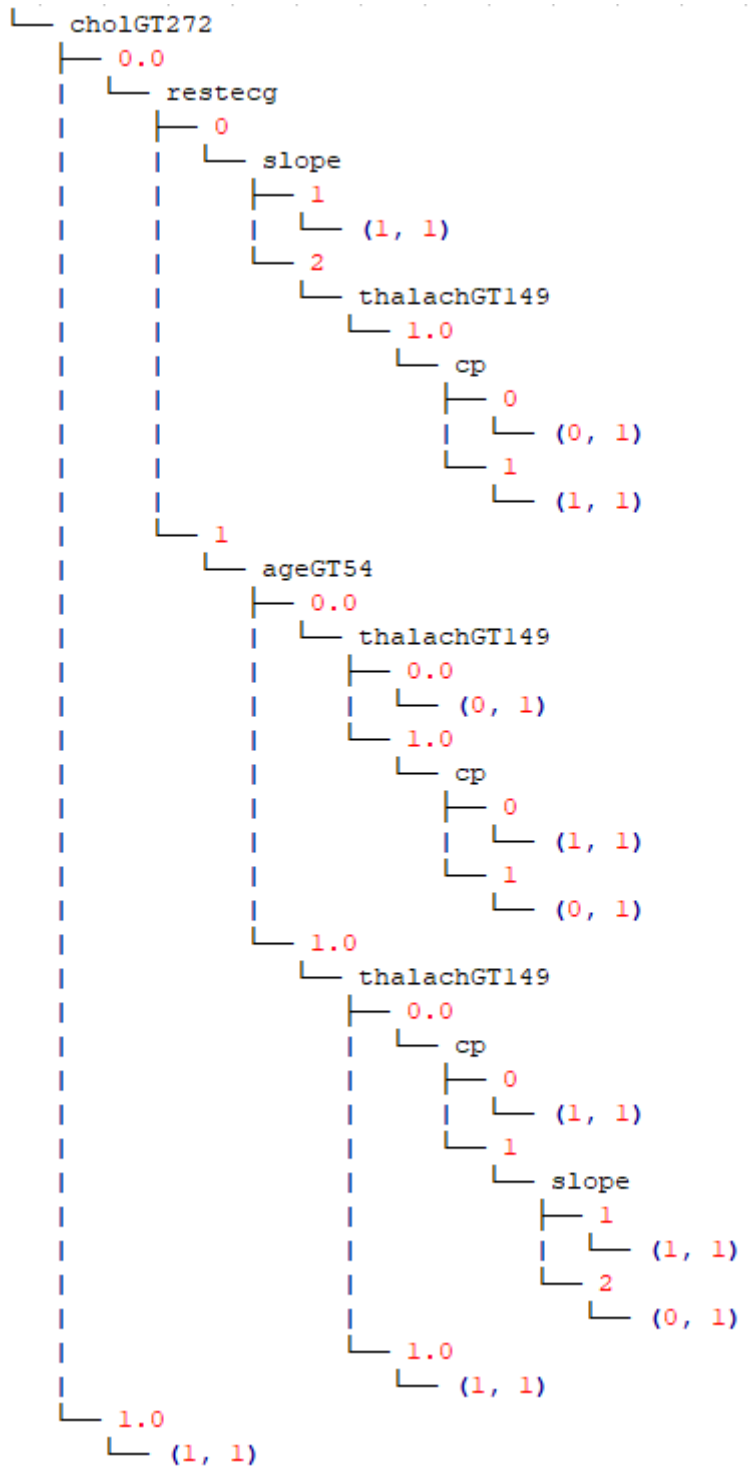


Això segurament és degut al fet que les particions de les dades contenen quantitats molt similars d'informació, suavitzant l'impacte que normalment tindrien algorismes que ja tenen en compte la informació de cada partició.
És a dir, l'algorisme que més es beneficia d'aquesta mètrica, és id3, ja que al ser un algorisme que no contempla el valor de cada partició, li simplifica molt el treball el fet que totes les particions tinguin la mateixa informació.

Si contemplem el resultat d'aquest per a una quantitat major d'intervals, podem apreciar que el rendiment d'id3 no cau dràsticament com feia en apartats anteriors.



La següent imatge mostra a el nou arbre que es genera. Es veu que ara els atributs continus són diferents. Per exemple, l'atribut "thalach" ara és "thalachGT149", significant que, com a màxim, es creen dos branques a partir d'ell. Una és per les mostres que el valor de l'atribut thalach sigui més petit o igual que 149, l'altra branca és per les mostres que el valor sigui més gran.



Apartat 4

Random Forest

Els arbres són models bastant propers a generar overfitting. Per evitar que passi hi ha tècniques que fusionen resultats de diferents arbres per generar una predicció final més robusta.

El Random Forest és una d'aquests tècniques i es basa en generar N arbres de decisió idèntics però entrenant-se sobre diferents dades. A diferència del baggin, que cada model s'entrena amb un subconjunt de Train, els Random Forest ho fan amb conjunts de la mateixa mida que el Train original. El que canvia, doncs, és el contingut d'ells. Cada nou conjunt és triat agafant valors aleatoris de l'original permeten repetició.

Per posar un exemple, hi ha el conjunt Train original = [1, 2, 3, 4, 5, 6]. Un dels nous conjunts podria ser [1, 2, 2, 3, 6, 6]. Si ens fixem bé, ambdues llistes tenen la mateixa mida, però la segona té alguns elements repetits. Es repeteix el mateix procés per cada arbre del Random Forest.

Un cop es tenen tots entrenats, la predicció es fa per votació. Es dóna per bona la que hagi sortit més vegades als arbres que formen el Random Forest

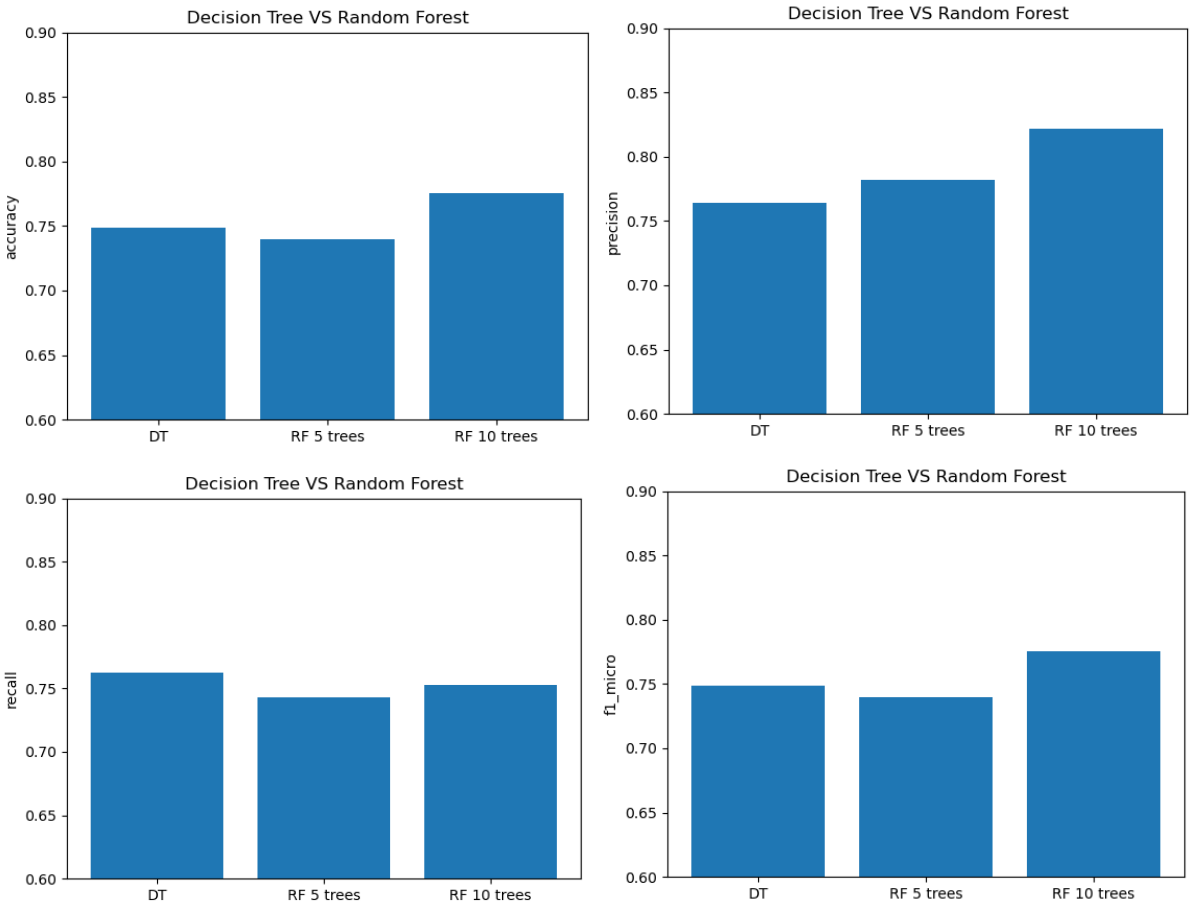
Per implementar-ho s'ha creat una classe "RandomForest" que no fa altra cosa que crear diferents conjunts Train i entrenar N arbres, cada un amb un conjunt diferent. En acabat retorna la predicció fent votació.

Resultats

S'ha fet un seguit de proves per avaluar i comparar el Random Forest dissenyat. Totes elles s'han fet utilitzant cross validation amb 10 folds.

Els resultats són lleugerament millors quan el Random Forest té 10 arbres. En canvi, quan té només 5 arbres els resultats són molt semblants al Decision Tree. Veient aquesta millora, és molt probable que un sol decision tree estigui generat overfitting.

L'únic mètrica que no millora, però tampoc empitjora, és la recall.



Apartat extra

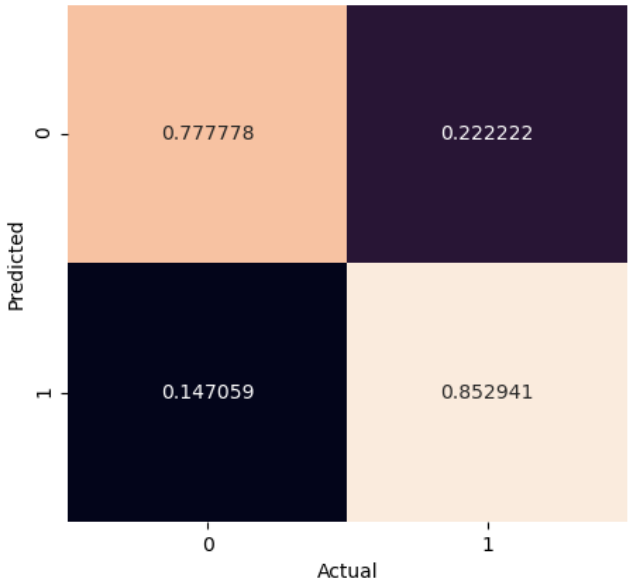
A continuació decidim fer una comparació del model programat per nosaltres i el de Sklearn.

El nostre model s'executa amb les opcions que hem trobat que donen millor resultat:

- 2-way split amb 15 intervals inicials
- heurística Gini
- Utilitzant Probabilistic Approach

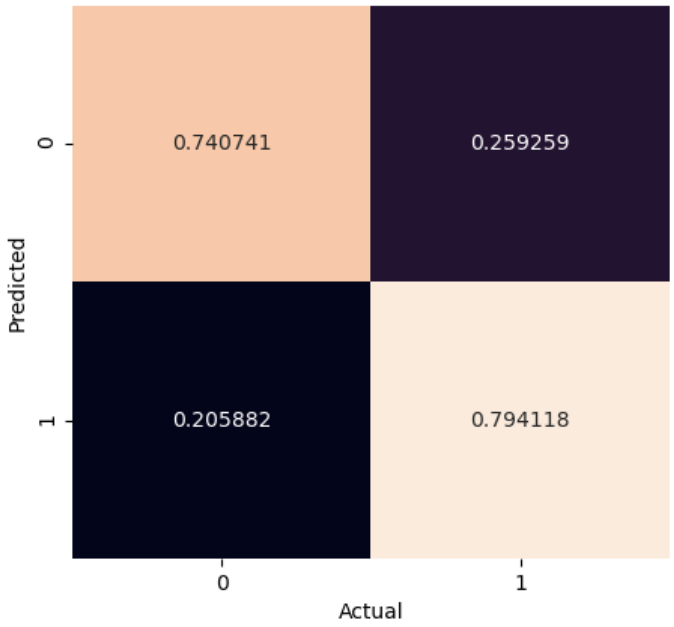
Per l'altre banda, el model de Sklearn s'executa sense ningun paràmetre de configuració i deixem que ell sol apliqui el que cregui.

Model propi



	precision	recall	f1-score	support
0	0.81	0.78	0.79	27
1	0.83	0.85	0.84	34
accuracy			0.82	61
macro avg	0.82	0.82	0.82	61
weighted avg	0.82	0.82	0.82	61

Model Sklearn



	precision	recall	f1-score	support
0	0.74	0.74	0.74	27
1	0.79	0.79	0.79	34
accuracy			0.77	61
macro avg	0.77	0.77	0.77	61
weighted avg	0.77	0.77	0.77	61

Si bé el nostre model ha obtingut millors resultats, som conscients que no funcionaria igual de bé per qualsevol dataset. De la mateixa manera, hem dedicat també molt temps a trobar configuracions, tant del model com de les dades, que ofereixin el millor resultat. En canvi, el model de Sklearn troba unes bones prediccions sense ni tant sols dedicar temps al "hyperparameter search".

Tanmateix, es pot dir que el model dissenyat per nosaltres té un bon rendiment i, fent els canvis oportuns, pot arribar al de prediccions que el de Sklearn.

Treballs futurs

Com a principal millora que es podria aplicar és fer pruning a l'arbre. Es basa en esborrar seccions de l'arbre redundants o no crítiques a l'hora de classificar les mostres. S'aconseguiria un millor temps d'execució.

Una altra millora que es podria fer és optimitzar alguns càlculs durant l'entrenament de l'arbre, sobretot el C4.5. Ara mateix tenen un temps molt elevat.

Conclusions

Com a aspecte més important, hem entès la necessitat d'analitzar el problema i fer un bon disseny i estructura general del programa. És un dels primers treballas que se'ns demana fer des de 0. Fins ara sempre ens donaven l'estructura del programa i només havíem d'omplir el contingut de les funcions. No érem conscient de la mà de problemes que pot presentar un programa mal estructurat.

Havent hagut de dissenyar un arbre de decisió ens hem adonat que realment hi ha moltes coses a tenir en compte. Fins ara només s'havia vist com utilitzar models ja creats, on només cal fer un "Fit" i un "Predict", no érem conscients de tot el que es fa internament.

Si bé programar un arbre de decisió no té molta dificultat, sí que és difícil decidir com tractar algunes situacions. Per exemple, com escollir el millor atribut o què fer amb valors no existents a l'arbre. Una mala decisió pot fer que el model no es comporti correctament.