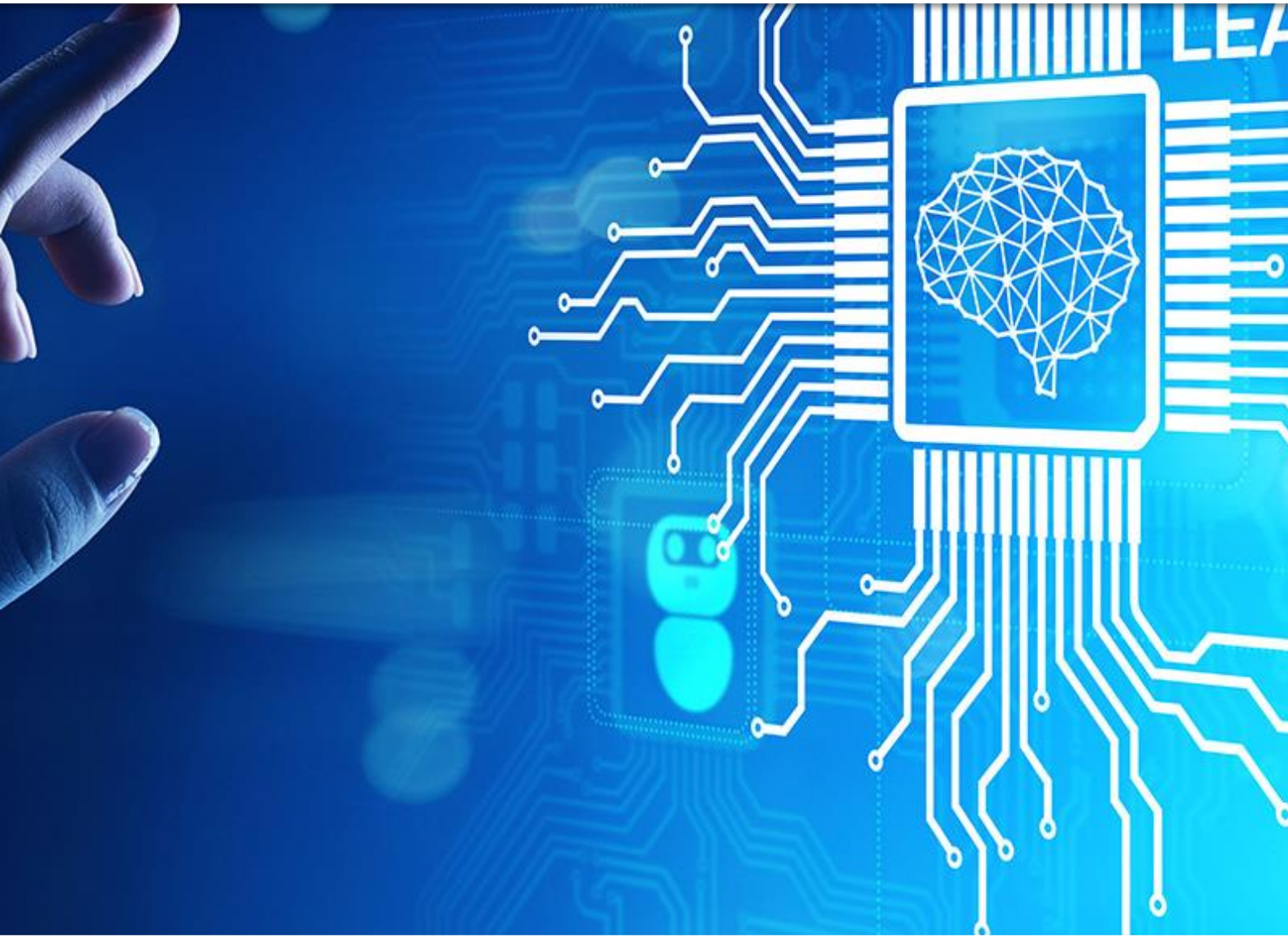
# PRÀCTICA 3
# CAS KAGGLE



**APRENENTATGE COMPUTACIONAL**

**Serena Sánchez**
**1599952**

# Índex

# Introducció

La base de dades amb la qual es treballa en aquest informe tracta de peixos comuns en una peixateria. Es un conjunt de dades sobre característiques dels peixos on comptem amb 7 espècies diferents que es venen al mercat. Aquestes mostres s'han agafat de peixos d'un mateix llac: Laengelmavesi, a Finlandia.

L'objectiu que proposen els autors de la base de dades a la plataforma web Kaggle és predir el pes ("weight") dels diferents peixos mitjançant una regressió lineal. De fet, la majoria de documentació a la pàgina tracta sobre aquest tema. Tot i així, a mi des d'un principi hem va cridar més l'atenció intentar predir l'atribut "Species" que fa referència a la espècie de peix que es una mostra concreta mitjançant una classificació multi classe.  Per tant, el nostre target pot prendre 7 valors diferents en referència a les 7 espècies diferents de peixos que trobem a la base de dades.

A mesura que s'avanci en l'informe i es realitzin diferents probes es decidirà amb quines dades i quin model es farà servir per aconseguir un bon predictor. De moment, es començarà explicant la base de dades. L'informe, el notebook en format .ipynb i .py executable utilitzats es troben referenciats en l'apartat conclusions.

# EDA & Preprocessament de dades

## Atributs de la base de dades

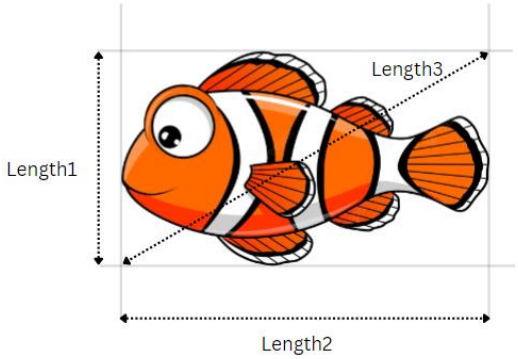
La base de dades descarregada compta amb 159 mostres (files) i 7 característiques (columnes). Fent una visualització ràpida dels atributs observem que n'hi ha de dos tipus: object (Species) i float64 (la resta). Abans d'entrar en l'anàlisi avançat de dades, mostrarem un llistat amb tots els atributs i el que signifiquen.

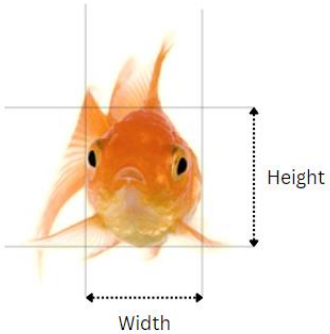
| **Atribut** | **Descripció** | **Tipus** |
|---|---|---|
| `Species` | Nom de l'espècie del peix | object |
| `Weight` | Pes del peix en grams | float64 |
| `Lenght1` | Longitud vertical en centímetres | float64 |
| `Length2` | Longitud diagonal en centímetres | float64 |
| `Length3` | Longitud de la creu en centímetres | float64 |
| `Height` | Alçada del peix en centímetres | float64 |
| `Width` | Ample diagonal en centímetres | float64 |

Un cop sabem el nom de les dades amb que treballarem, caldria entendre tots els atributs abans de començar l'anàlisi, ja que 5 dels 7 atributs que hi ha tracten sobre mides del peix i la definició que apareix sobre aquests pot resultar confusa. Investigant per la pàgina del Kaggle, en l'apartat discussion[1] vaig veure que ja s'havia parlat sobre aquest tema i que el mateix creador de la base de dades ja havia donat informació més concreta sobre aquests atributs. Mitjançant aquesta informació s'han extret les següents conclusions:

- Els atributs LengthX fan referència a les mides totals de cada peix.
- Els atributs Height i Width fan referència al cos proteic del peix, sense tenir en compte les aletes, per exemple.

De forma més aclaridora amb un parell d'imatges:



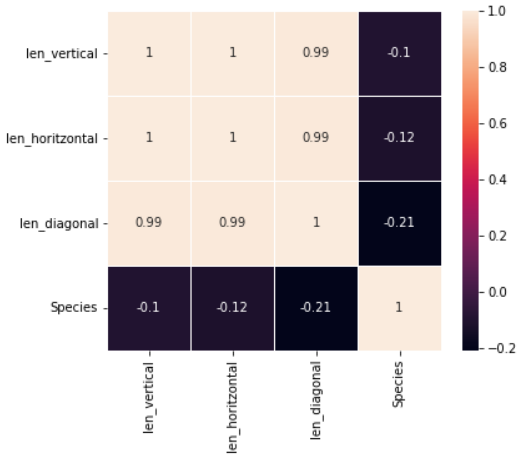

[1] https://www.kaggle.com/datasets/aungpyaeap/fish-market/discussion/97243

Sabent això, comença a tenir poc sentit el fet de tenir tantes mesures de longitud diferent. Per exemple, gràcies a la diferència entre Height i Length1, podrem saber la llargada de l'aleta superior d'un peix, que pot variar depenent l'espècie.

Amb aquesta informació farem dues coses: en primer lloc, canviar el nom de les diferents LengthX per poder treballar millor amb els noms dels atributs. Els canvis seran els següents:

- Length1 → len_vertical
- Length2 → len_horitzontal
- Length3 → len_diagonal

En segon lloc, mirarem la correlació entre aquestes tres longituds (imatge de la dreta), perquè matemàticament resulta molt senzill calcular una d'aquestes amb el valor de les altres dues. Per exemple, si volguéssim calcular Length3 a partir de Length1 i Length2, i hem entès bé les diferents mesures, només caldria aplicar el teorema de Pitàgores per aconseguir el seu valor.




Efectivament, veiem que les correlacions són molt altes, per tant, amb un dels tres atributs ja tenim suficient per que el nostre model funcioni igual de bé que amb el data set original però amb dos atributs menys.

Per decantar-nos per una de les mides, s'ha decidit visualitzar la correlació amb Species, que és el nostre atribut objectiu, i tenint en compte els resultats de matriu de correlació, eliminarem len_vertical i len_horitzontal del dataset amb què estem treballant. Aquesta eliminació no generarà cap millora computacional en aquest cas, ja que tenim molts pocs atributs, però, si es tractés d'un cas més gran i hi haguessin correlacions molt altes entre bastants atributs, això seria un bon pas cap a un model més eficient.

# Preparació de les dades

Per tal de fer un bon preprocessament de dades abans d'escollir els models que utilitzarem per predir l'atribut objectiu, haurem d'analitzar diferents aspectes de la base de dades: valors nuls, valors categòrics, diferències dels rangs de valors entre atributs i desbalancejament de classes. Desenvoluparem aquests apartats en aquest ordre.

### Tractament de valors nuls

Respecte aquest apartat hem tingut sort i no tenim cap atribut a null a la base de dades.

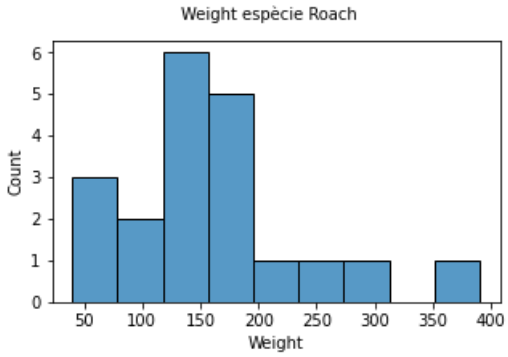
Tot i així, trobem un valor a 0 en la columna Weigth. Tenint en compte que considerem que un peix no pot pesar 0 grams, potser es tracta d'un error de transcripció. Una primera idea seria eliminar aquesta fila. Si tenim en compte que és de l'espècie Roach i tenim 20 mostres d'aquesta espècie, eliminaríem un 2% de la informació sobre aquesta. No penso que sigui una gran pèrdua, però com tenim poques mostres a la base de dades i es tracta només d'un valor, he decidit utilitzar el mètode d'inferència KNNImputer, per donar-li un valor raonable a l'atribut d'aquesta mostra. D'aquesta forma, no informació de la resta de columnes sobre aquesta mostra i, si el KNNImputer fa una bona feina, tindrem un valor que correspongui aproximadament al valor real.

Per fer això, s'ha hagut de transformar el valor 0 a nan i aplicar la funció KNNImputer amb el paràmetres n_neighbors a 10, ja que m'ha semblat un bon número donat a que hi ha 20 mostres de la mateixa espècie i weigths a 'distance' per ponderar la importància d'un veí en major mesura si aquest està molt a prop.

Si comparem els valors de weigth d'aquesta espècie sense inferència i el valor inferit. Tenint en compte les mesures de weight de l'espècie Roach que veiem en la gràfica de barres de la dreta, penso que el valor ha estat correctament inferit degut a que es troba dins dels valors mes comuns de l'espècie.

| | Species | Weight | len_diagonal | Height | Width |
|---|---|---|---|---|---|
| 40 | Roach | 0.000 | 22.800 | 6.475 | 3.352 |

| | Species | Weight | len_diagonal | Height | Width |
|---|---|---|---|---|---|
| 40 | Roach | 119.636 | 22.800 | 6.475 | 3.352 |




### Tractament de dades categòriques

En el cas de la nostra base de dades l'atribut objectiu és l'únic categòric. Aquest tipus de dades no ens permeten dur a terme operacions numèriques ni entrenaments de regressió. Per tant, abans de posar-se a estudiar les dades, s'ha hagut de modificar aquesta columna. En aquest punt existeixen dues possibilitats per convertir-los a numèrics: label encoding i one hot encoding.

El label encoding és un mètode que assigna a cada valor categòric un de numèric, com podem veure a la imatge de la dreta. Aquest mètode s'acostuma a utilitzar quan aquests valors són ordenables (per exemple, si l'atribut fos: nivell d'anglès i els valors possibles: 'baix', 'mitjà' i 'alt', és poden ordenar de menor -0- a major nivell -2-) i la quantitat de valors possibles de la columna és molt alta. En canvi, el

one hot encoding consisteix en crear una columna per cada resposta diferent que aparegui en un atribut i, per cada registre, posar un 1 a la columna a la que pertany aquest registre i la resta posar 0's. S'acostuma a utilitzar quan tenim valors no ordenables i pocs valors possibles en l'atribut, ja que sinó, acabaríem amb un número d'atributs (columnes) intractable.

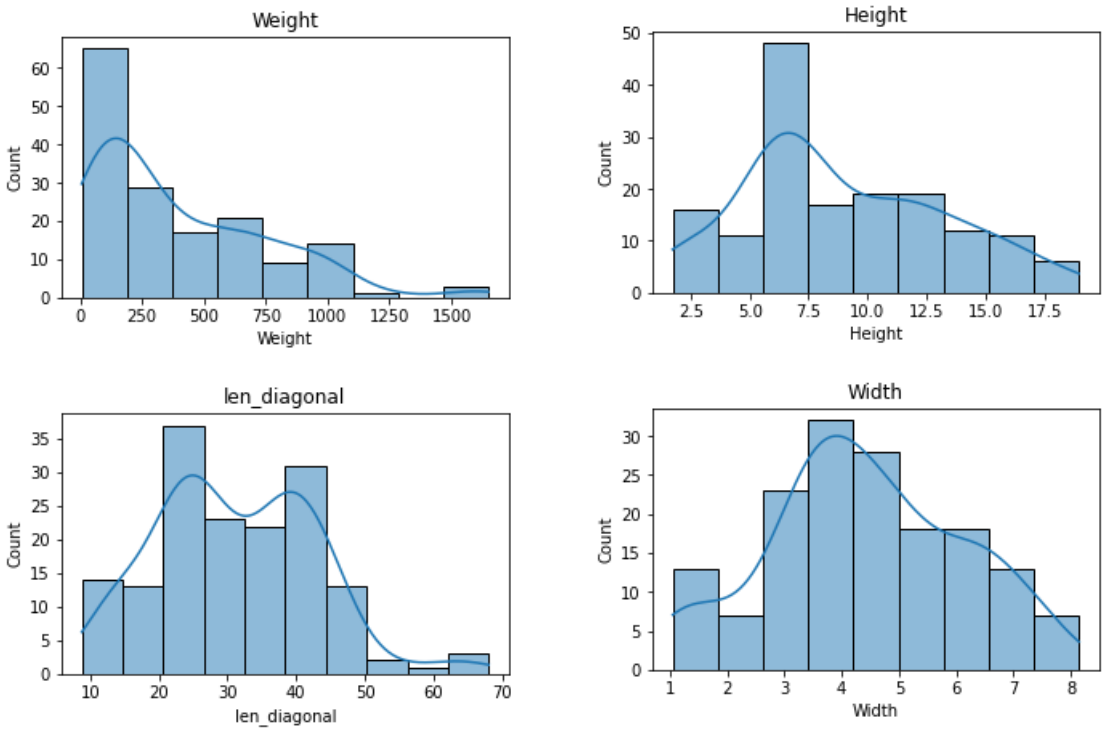

En aquest cas, al tractar-se de classes no ordenables i pocs valors possibles dels atributs la meva primera idea era aplicar one hot encoding. Després de fer aquest pas, vaig veure que l'atribut objectiu de la base de dades quedava repartit en 7 columnes, és a dir, com si fossin 7 atributs diferents i avançant-me en el temps, vaig tenir dubtes sobre com podria crear un model que predigués 7 columnes diferents, que es el que provocaria el one hot encoding. Per tant, vaig decidir fer un pas enrere i canviar l'estratègia: aplicar label encoding, numerant del 1 al 7, les diferents espècies de la base de dades.

| **Species before label encoding** | **Species after label encoding** |
|---|---|
| Bream | 0 |
| Roach | 4 |
| Whitefish | 6 |
| Parkki | 1 |
| Perch | 2 |
| Pike | 3 |
| Smelt | 5 |

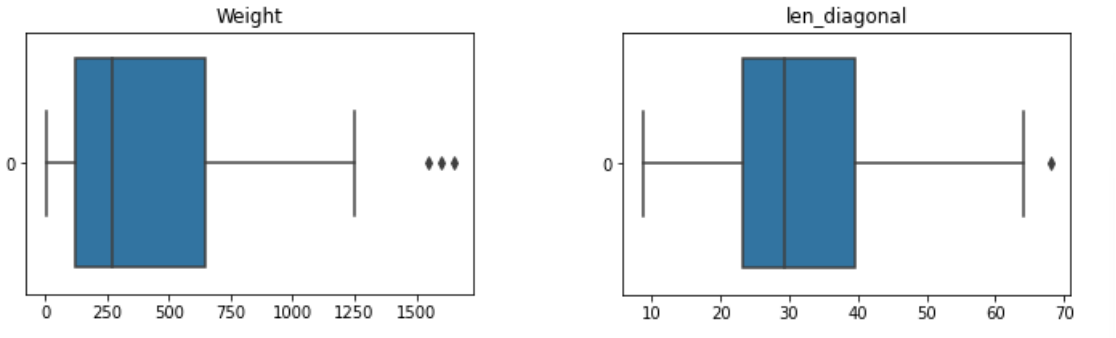
### Outliers

Els outliers són valors numèrics de cada atribut del dataset que prenen valors molt diferents a la resta de la columna, això pot causar problemes en l'anàlisi de les dades i la posterior creació d'un model que funcioni. Es aconsellable treure'ls abans de continuar amb el preprocessament de dades. Per aquest motiu anem a analitzar si tenim outliers, a on els tenim i com podem tractar-los.

En primer lloc, observem amb histogrames la distribució de les dades per tenir una primera intuïció d'on podem trobar els outliers.




En aquest cas, podem intuir casos d' outlier en dos atributs diferents: de forma bastant clara en els valors més alts de l'eix X en l'atribut weigth, ja que es tracta d'una funció exponencial i en els valors més alts de len_diagonal, que potser també trobem algun valor outlier. Visualitzarem aquests dos atributs amb diagrames de caixa, per veure-ho de forma més clara:




Confirmem el que havíem intuït dels histogrames, existeixen outliers en aquests atributs i es representen amb els punts que es troben fora dels bigotis del diagrama de caixes. Abans de possar-nos a eliminar o tractar aquests outliers, ens interessa saber on es troben aquests outliers, ja que al tenir poques files, potser formen part d'una mateixa espècie amb poca representació.

Per aconseguir aquesta informació realitzarem per els dos atributs els mateixos càlculs:

- Càlcul del quartil 1 (Q1)
- Càlcul del quartil 3 (Q3)
- Càlcul del rang interquartil (IQR = Q3-Q1)
- Càlcul bigoti superior (Q3 + 1.5*IQR)

D'aquesta forma sabrem on es troben els valors límits dels bigotis del diagrama de caixa i determinar quins valors de la nostra base de dades estan sobrepassant aquests límits. Com hem vist a les imatges,

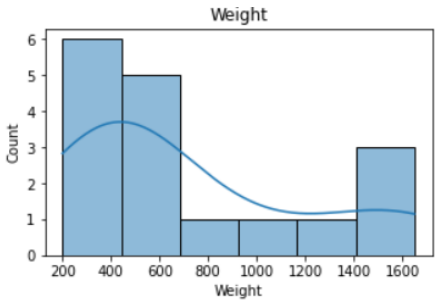
només necessitarem el càlcul del bigoti superior, perquè els punts de valors inusuals es troben a la dreta del diagrama.

Atribut weight

Si realitzem els càlculs anteriors per aquesta columna, trobem que hi ha 3 files que es consideren outliers i són les següents:

| | Species | Weight | len_diagonal | Height | Width |
|---|---|---|---|---|---|
| 142 | 3 | 1600.000 | 64.000 | 9.600 | 6.144 |
| 143 | 3 | 1550.000 | 64.000 | 9.600 | 6.144 |
| 144 | 3 | 1650.000 | 68.000 | 10.812 | 7.480 |

Com podem observar, els tres outliers són de la mateixa espècie: Pike (3). Analitzarem ara el valor que pren l'atribut Weight per aquesta espècie amb un diagrama de barres. Com podem observar, els valors d'aquests són molt semblants entre ells i, de fet, els valors que s'han calculat com outliers són més probables que d'altres que no se'n consideren perquè estan dins del rang del diagrama de caixes.




Atribut len_diagonal

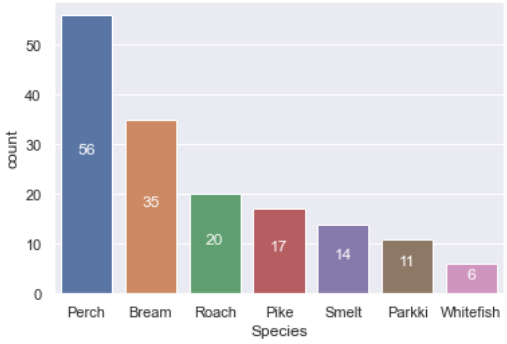
En canvi si realitzem els càlculs anteriors per la columna len_diagonal, trobem que hi ha 1 fila que es considera outlier i és la següent:

| | Species | Weight | len_diagonal | Height | Width |
|---|---|---|---|---|---|
| 144 | 3 | 1650.000 | 68.000 | 10.812 | 7.480 |

Aquesta fila també es de l'espècie 3 i coincideix amb una de les files que té outlier a weight. Una possible idea seria eliminar la fila que és outlier per dos atributs. Però al tenir poques dades i havent confirmat que aquestes dades estan dins de les característiques normals de l'espècie, deixarem el dataset igual, és a dir, encara conservarà les dades que hem catalogat com outliers.

### Balancejament de les dades

Un primer anàlisi senzill de les dades es veure si hi ha un balancejament entre les classes de l'atribut Species que prenen 7 valors diferents. Així sabrem si totes les classes compten amb la mateixa proporció d'informació i, per tant, tindran la mateixa importància dins dels models que entrenem. Ho visualitzem en un diagrama de barres:

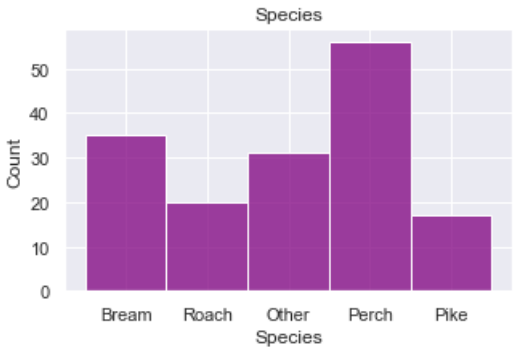



Veiem que clarament la distribució queda molt desbalancejada. Això pot afectar en el nostre anàlisi ja que hi ha molta diferència i, com s'ha comentat abans, la desproporció d'informació entre classes afavorirà l'aprenentatge del les espècies Perch i Bream que són les més nombroses.

Per solucionar aquest problema hi ha dues opcions: recaptar informació sobre possibles agrupacions diferents d'aquests tipus de peixos o agrupar les tres espècies menys nombroses en una nova anomenada "Other" i així, aconseguir una classe amb més representació. Envers la primera proposta,

no s'ha trobat cap agrupament per família o condicions diferent, per tant, crearem un data set nou amb les espècies més nombroses en una nova categoria. Per veure més endavant la distribució de les dades, treballarem amb el dataset amb 7 espècies i el dataset amb 5 espècies en paral·lel. El nou dataset comptaria amb les següents classes i mostres per classe:

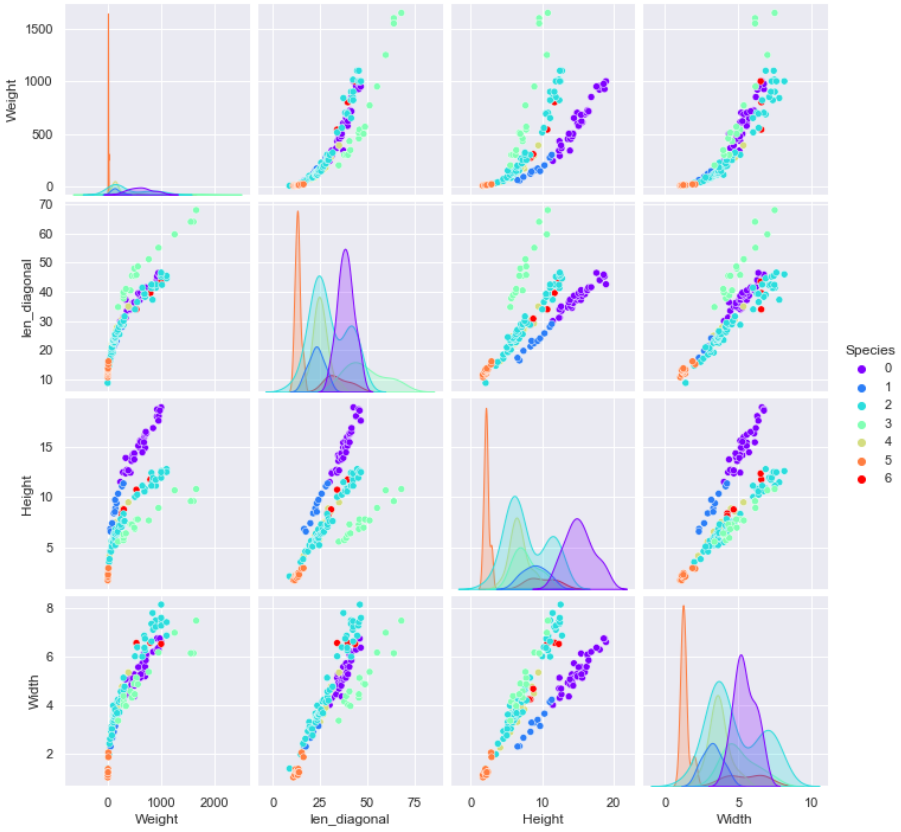
| Tipus de peix | Classe | Nº de mostres |
|---|---|---|
| Perch | Perch | 56 |
| Bream | Bream | 35 |
| Roach | Roach | 20 |
| Pike | Pike Species | 17 |
| Smelt, Parkki, Whitefish | Other | 31 |




Com tot i així no acaba de estar del tot balancejat, haurem d'anar amb compte a l'hora d'analitzar els resultats dels models o utilitzar mètodes que no afavoreixin aquest desbalancejament.

### Distribució de les dades

Mitjançant l'eina del pairplot, anem a fer una primera visualització de les distribucions que tenim amb els diferents atributs i tenint en compte l'espècie.

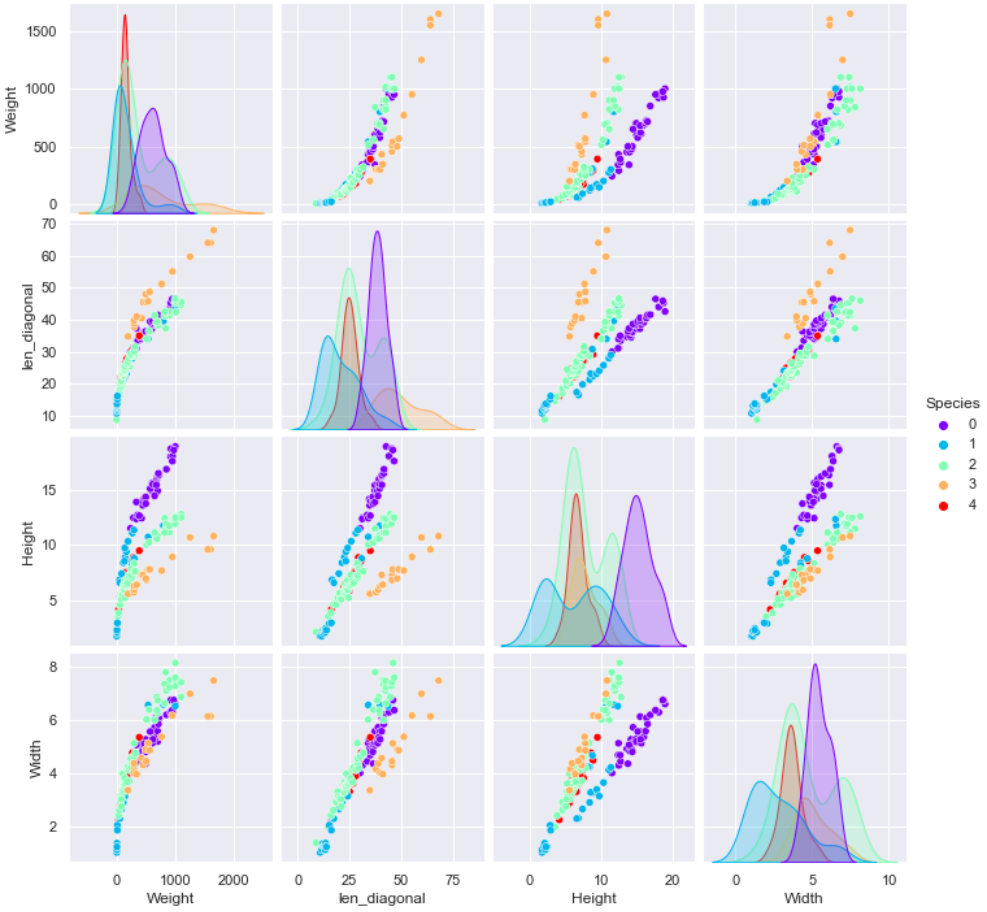



La primera visualització és amb el dataset amb les 7 espècies i veiem que amb algunes parelles d'atributs, les classes 3 (color ver), 0 (color lila) i 1 (blau fosc) tenen una clara diferenciació amb les altres. De forma una més confusa però encara amb diferenciació: la classe 5 (taronja). Tot i així, hi ha algunes classes una mica confuses en dues dimensions. Ara farem la mateixa visualització amb el dataset agrupat. D'aquesta forma veurem si les agrupacions cobren algun tipus de sentit, ja que, per exemple, si hem agrupat la classe 0 (lila) amb la classe 5 (taronja), prenen un valor molt diferents per tots els atributs i farem que el model perdi precisió.

Amb el dataset amb els valors agrupats obtenim la següent visualització:

Si ho comparem amb la imatge de l'altra dataset, no es veuen moltes diferències a simple vista. El que més ens pot sobtar i que pot fer que el model empitjori és la classe 1, que la tenim repartida a dos llocs diferents però ben diferenciada. Tot i així, en 2 dimensions, visualitzar les diferents classes resulta complicat.

Per fer-la més aclaridora utilitzarem un PCA, un anàlisi de components principals. Però abans normalitzarem les dades.




## Normalització de les dades

En el cas de les dades que tenim és necessari fer una normalització, ja que fent un anàlisi de les mitjanes dels atributs veiem que n'hi ha que prenen el valor de 4.417 ("width") i d'altres on el valor de la mitjana és 398.326 com "weight". Això pot significar que a l'hora de crear un model i entrenar-lo, les dades amb un rang de valors més alts prenguin una major importància i dominin el model envers a les que prenen valors petits, fent que l'estimador no pugui aprendre correctament o com s'havia esperat.

Es fa la normalització amb la eina de preprocessament de dades StandardScaler i la funció transform on s'elimina la mitjana i s'escalen les dades de forma que la seva variança sigui igual a 1. A la imatge de l'esquerra les dades sense normalitzar i a la de la dreta normalitzades.

| | Species | Weight | len_diagonal | Height | Width |
|---|---|---|---|---|---|
| **0** | 0 | 242.000 | 30.000 | 11.520 | 4.020 |
| **1** | 0 | 290.000 | 31.200 | 12.480 | 4.306 |
| **2** | 0 | 340.000 | 31.100 | 12.378 | 4.696 |
| **3** | 0 | 363.000 | 33.500 | 12.730 | 4.455 |
| **4** | 0 | 430.000 | 34.000 | 12.444 | 5.134 |

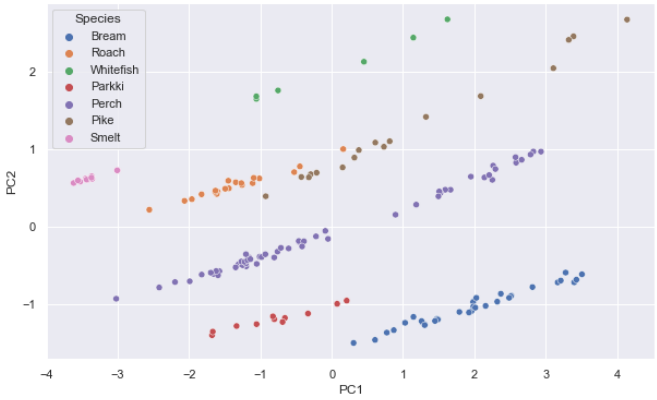
| | Species | Weight | len_diagonal | Height | Width |
|---|---|---|---|---|---|
| **0** | -1.333 | -0.441 | -0.106 | 0.597 | -0.237 |
| **1** | -1.333 | -0.306 | -0.002 | 0.821 | -0.067 |
| **2** | -1.333 | -0.166 | -0.011 | 0.797 | 0.166 |
| **3** | -1.333 | -0.101 | 0.196 | 0.880 | 0.023 |
| **4** | -1.333 | 0.087 | 0.240 | 0.813 | 0.426 |

## PCA

Com en el pairplot explicat anteriorment, només teníem una visualització per parelles d'atributs en dos dimensions, una forma de utilitzar els 4 atributs de la base de dades per veure la distribució però sense haver-ho de representar en 4 dimensions és l'anàlisi principal de components.

Resumidament, és un mètode d'anàlisi de dades multivariades que permet estudiar i visualitzar conjunts de dades multidimensionals. Consisteix en projectar mostres d'un espai p-dimensional amb número de variables p en un espai k-dimensional on k < p de forma que es conservi el màxim d'informació possible. D'aquesta forma, quan es treballa amb un conjunt de dades grans i volem contextualitzar-les en un espai (com pot ser una gràfica), es redueix la dimensionalitat aplicant un PCA que es queda amb 2 o 3 components principals de variables no correlacionades permetent així la visualització de les dades.

Pel nostre cas, farem la visualització en dues dimensions del dataset amb 7 espècies (imatge esquerra) i amb el de 5 (imatge de la dreta).

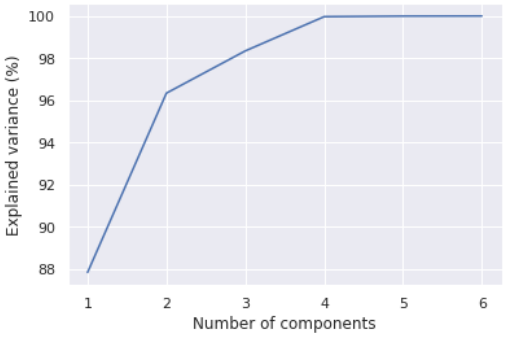




Com podem observar, en la primera imatge si que hi ha dues classes que generen una mica de confusió (Roach i Pike), però aquesta es veu totalment solventada quan agrupem les espècies a la segona imatge on les classes estan totalment diferenciades. En l'últim apartat ("Comparativa amb un altre treball") utilitzarem la transformació a dues components que ens ha proporcionat el PCA per entrenar un model de Regressió Logística ja que la separació de classes amb aquestes es molt bona.

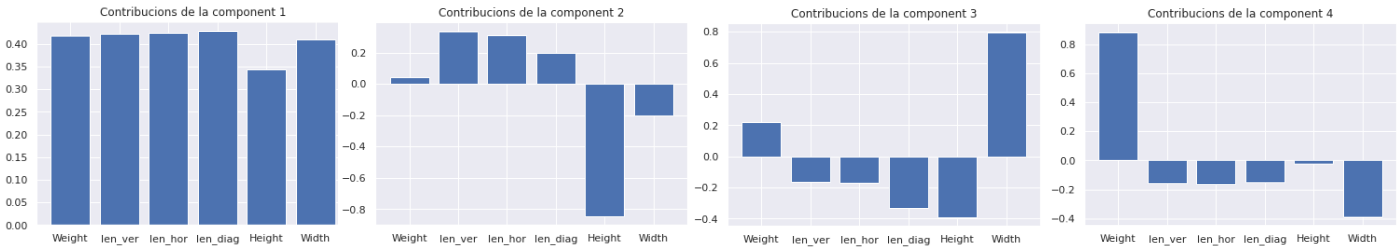
Més informació que ens aporta l'anàlisi de components principals amb el dataset original és la quantitat d'informació que ens proporciona cada atribut a cada component. Això ho analitzem a continuació.

Primer de tot tindrem en compte la variança de cada component, és a dir, en quines components obtenim la variabilitat de les dades que volem, per tal perdre el mínim d'informació possible. Per això podem fer un gràfic amb les 6 components que crea el PCA amb el dataset original i veure amb quines obtenim gran part de la informació. Tenint en compte la variància de cada component, és a dir, tenint en compte les components que perden menys informació:




Amb la gràfica anterior podem concloure que amb les primeres 4 components no perdem informació de les dades originals ja que conservem el 100% de la variància. De fet, amb només una component ja tindríem un 88% de la informació total, per tant aquesta component ens està donant la majoria

d'informació. Ara visualitzarem quins atributs són els que contribueixen en aquestes 4 primeres components principals.




Com podem observar, en la primera component tots els atributs contribueixen més o menys de la mateixa forma, però en les altres components hi ha atributs que destaquen sobre els altres. La component 2 tenim com a protagonista l'atribut Height, que ens aporta bastanta informació donat que la component 2 és la segona més important, en la component 3 l'atribut Width i en l'última component l'atribut Weight. Per tant, en resum, els atributs que menys informació generaran pel nostre model són les longituds. Tot i així, ja hem possat remei a aquesta fet quan hem observat les correlacions entre els atributs.

# Selecció del model

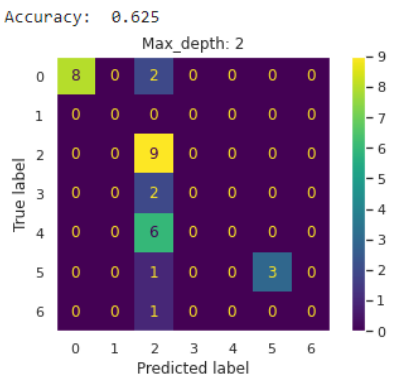
Ara que ja tenim la base de dades preparada, hem de plantejar-nos quin model utilitzar per predir l'atribut Species de les nostres mostres i per mostres futures de forma correcta. El nostre problema és clarament una classificació multiclasse, per tant, no tindria sentit començar a provar models de regressió lineal, ja que no correspondrien al nostre problema. En canvi, els models de predicció que penso que si s'adapten millor al problema que hem estem desenvolupant son: arbre de decisió, SVM i KNN. A més, tot i ser un model més complicat, afegirem les xarxes neuronals com a algoritme a analitzar ja que s'ha estudiat a teoria i em sembla un dataset coherent per entrenar les dases. Per últim, cap al final de l'informe es detallarà la Regressió Logística com a model ja utilitzat en un altre treball de Kaggle. En aquest apartat, visualitzarem com es comporta cada model dels escollits amb les nostres dades i amb les classes de l'atribut objectiu. D'aquesta forma podrem estudiar quin és el model que es pugui adaptar millor a les nostres dades.

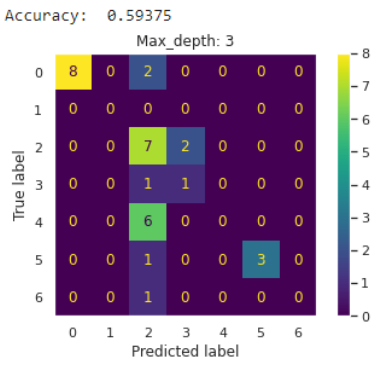
El valor dels hyperparàmetres serà escollit mitjançant prova i error, i es farà l'estudi de cada model canviant algun hyperparàmetre concret de cada model per veure el seu comportament amb les dades. Per tant, aquests valors s'han anat modificant a mà per fer diferents proves i obtenir una visualització. Però, la major accuracy que trobem en aquest apartat no vol dir que sigui la millor que podem trobar. Quina és la millor combinació d'hyperparàmetres per cada model és dura a terme en l'apartat següent.

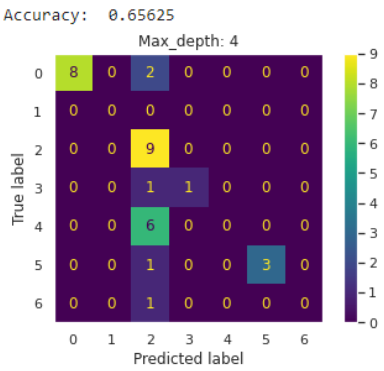
Cal destacar que tots els models s'entrenaran amb el dataset amb les espècies originals i el dataset amb algunes de les espècies agrupades, ambdós preprocessats i normalitzats.

## Arbre de decisió

Un arbre de decisió és un algoritme utilitzat per classificació, en el nostre cas, multiclasse. Aquest planteja un conjunt de preguntes sobre els atributs al conjunt de dades d'entrenament. A l'arrel de l'arbre i en cadascun dels nodes que crea es planteja una pregunta, que correspon a un atribut i aquets node es divideix en diferents branques o en es troben els valors possibles de cada atribut. Les fulles de l'arbre fan referència a les diferents classes en que es divideix el conjunt de dades. Utilitzarem l' arbre de decisió de la llibreria scikit-learn.

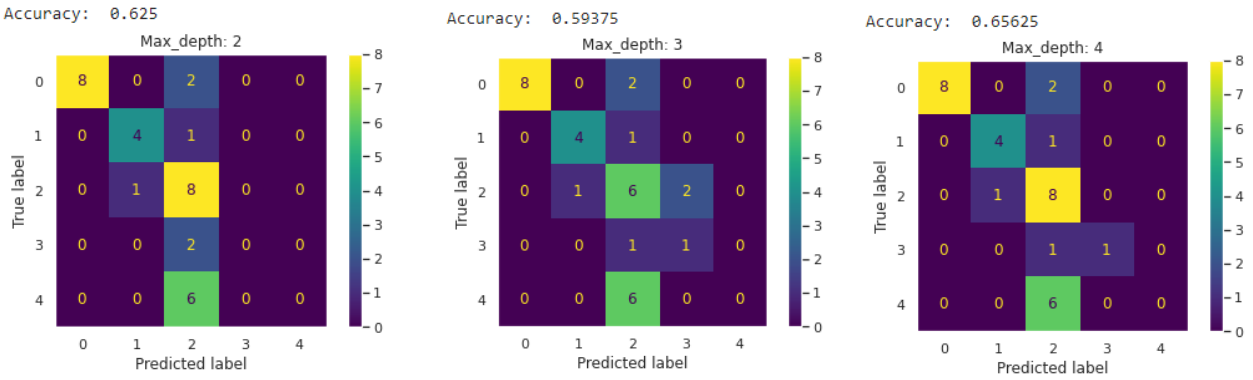



Per poder analitzar aquestes matrius de confusió es important tenir present a què espècie correspon cada etiqueta i quina representació té aquesta espècie en el conjunt de dades. Aquí veiem que l'etiqueta 0 i 5 en la majoria dels casos l'algoritme és capaç de predir-la bé i corresponen a l'espècie Bream i Smelt respectivament. Mentre que per la classe Bream tenim una representació del 35 mostres i és la segona

classe més nombrosa, per tant, l'algorisme a pogut aprendre correctament les característiques d'aquesta, l'espècie Smelt té una representació de 14 mostres. Si analitzem aquesta etiqueta en els pairplots de distribució (punts taronges) veiem que les característiques d'aquesta espècie són bastant especifiques i tot i tenir poca representació són bastant diferenciables de la resta. En canvi, si mirem que ha passat amb la classe Perch que es a la que pertany la majoria de mostres, correspon a l'etiqueta 2 que és on els arbres estàn fallant. Això es degut a que l'arbre està pràcticament especialitzat en aquesta etiqueta confonent així les altres classes. Aquest és un dels problemes que ens causa el desbalancejament de classes. Veurem si això també passa amb el dataset en el que hem agrupat algunes




espècies:

Les espècies que formen part de l'etiqueta 1 Other son: Whitefish, Smelt, Parkki. Una mica en la mateixa línia de l'explicació anterior, trobem que l'etiqueta 0 i l'etiqueta 1 (a la que pertany Smelt amb característiques molt especifiques) resulta més fàcil de classificar per aquest model que les altres etiquetes. Sobre tot té confusió amb la classe Perch (classe amb més representació) i la resta de classes que tenen menys mostres.

## SVM

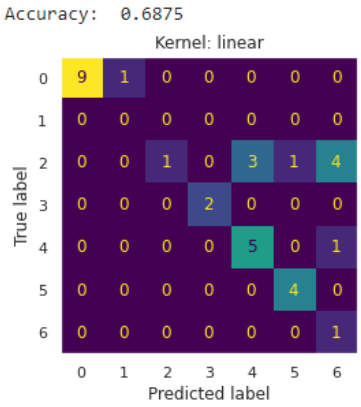
Una màquina de vectors de suport és un model d'aprenentatge automàtic supervisat que utilitza algorismes de classificació originalment per a problemes de classificació de dos grups, però que avui dia s'ha estès fins a problemes de classificació múltiple i regressió. Tot i així, segons la documentació pot no funcionar bé perquè tenim un vector de característiques de poques dimensions, és a dir, pocs atributs.

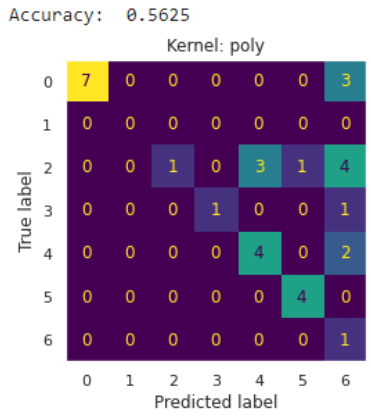
Aquest mètode es basa en buscar quin hiperplà separa millor les nostres dades per tenir una bona classificació. Aquest SVM més senzill dona bons resultats quan la separació és lineal, però si no ho és, el rendiment es molt baix. Per canviar això, s'empra una estratègia per expandir les dimensions del espai original i observar si així les dades són linealment separables. Per fer això, s'usa la funció kernel que retorna el valor del producte escalar enter dos vectors realitzant un nou espai diferent a l'original on es troben aquests vectors, obtenint un millor resultat.

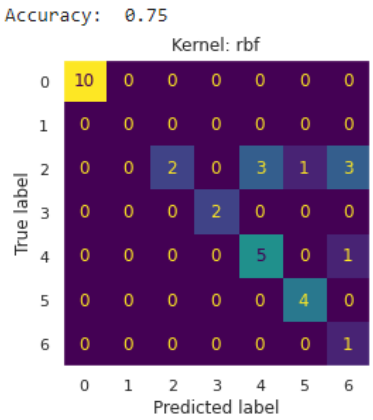
Cal destacar que en la llibreria que utilitzem hi ha diferents tipus de classificadors de vectors de suport com: LinearSVC o SGDClassifier que es recomanen quan tenim una base de dades amb moltes mostres. En el nostre cas, no les farem servir, ja que comptem amb poques dades.

Pel mètode SVM amb diferents kernels obtenim els següents resultats:

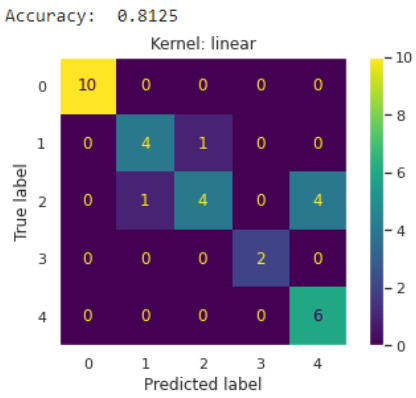
Classes no agrupades

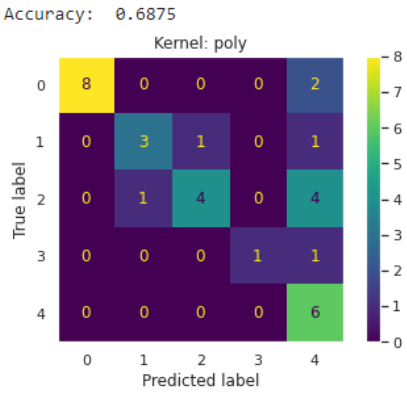


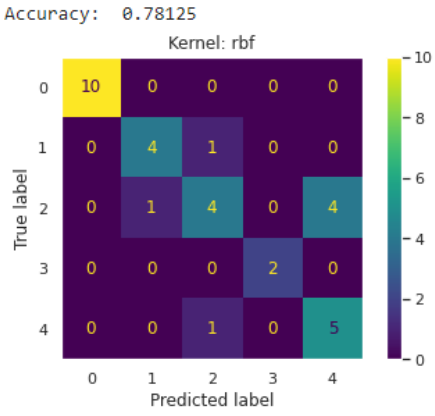



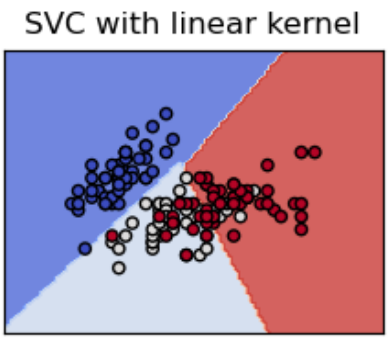

Classes agrupades

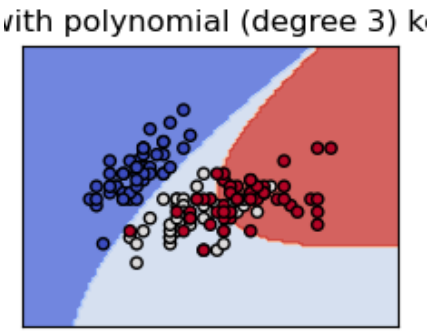


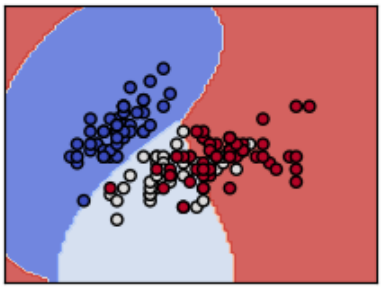




Kernels



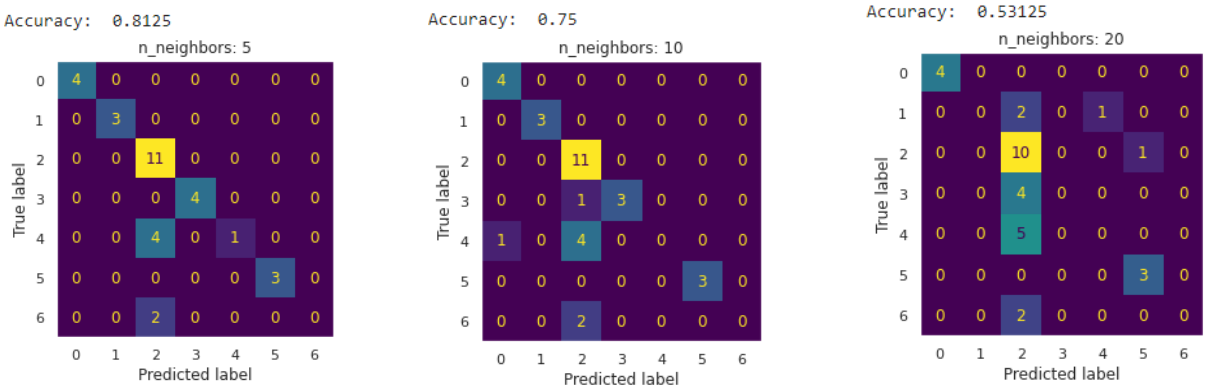

A la imatge de dalt podem observar que obtenim millors classificacions que amb els arbres de decisió i que en alguns casos aconseguim l'espècie Bream de forma completament correcta. Tot i així, per les altres espècies encara hi ha confusions. Cal destacar que en aquesta execució, per les dades no agrupades no ha hagut en el conjunt de test cap espècie 1 (Parkki), ja que les 11 mostres estaven al conjunt de train.

Envers a l'ús de diferents kernels veiem que pel primer cas, s'ajusta millor el kernel rbf que tracta d'una funció radial i que s'ajusta a dades no separables linealment. Com no podem visualitzar en 4 dimensions

el que està passant amb les divisions que crea aquest kernel, suposarem que les nostres dades no agrupades funcionen millor amb aquest nucli donat la separació que tenen entre elles. En canvi, en el segon cas, veiem que el kernel amb millors resultats és el lineal. Tot i que podem intuir que les dades no es separen linealment en un espai 4-dimensional, és la forma que millor s'adapta a les dades en aquesta execució. Tot i així, veiem que la diferència de rendiment del model no es gaire amb els diferents nuclis ja que tot i que l'accuracy canviï en un percentatge notable, el que varia es només la classificació d'una o dues mostres.

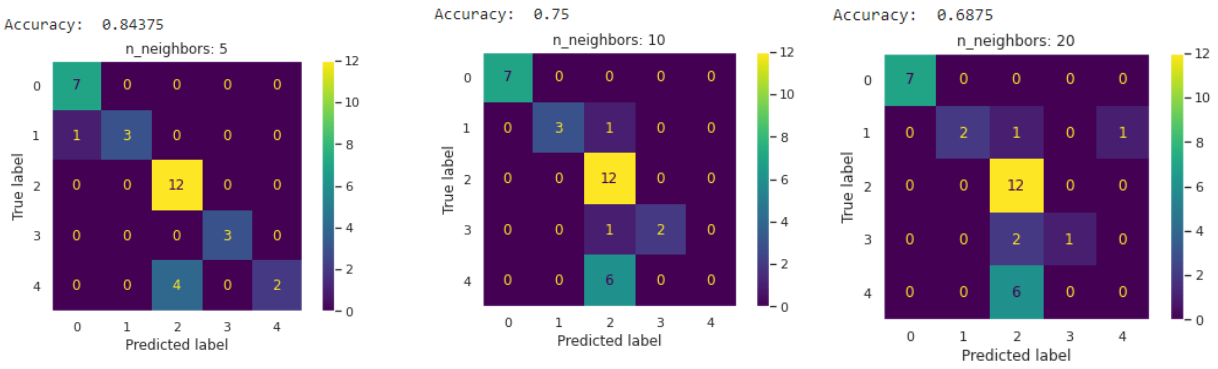
## KNN

L'algorisme K-nearest neighbours és el classificador més simple. Aquest algorisme col·loca totes les mostres del conjunt d'entrenament en un espai i quan arriba una nova mostra a classificar, aquest examina les k mostres més properes i li assigna la classe més nombrosa dels k punts veïns trobats.




En aquest cas visualitzem les matrius de confusió dels resultats del knn amb número de veïns 5,10,20. Aquests números no han estat escollits a l'atzar, ja que només mirant el número d'espècies de la classe menys nombrosa: Whitefish amb 6 mostres, es molt complicat aconseguir encertar la classe amb 20 veïns, ja que hi haurà una influència de mínim 15 mostres diferents al conjunt de traint que faria variar la predicció. En el cas que les classes siguin bastant compactes, en el cas d'aquestes mostres i tal i com es veu a la primera imatge, necessitaríem un número de veïns petit per a que el model sigui més fiable. En aquest cas si podem observar que tenim classes bastant compactes com les de l'etiqueta 0 i l'etiqueta 5 que tot i escollir un número de veïns molt alt, els true positives funcionen perfectament.

Si fem una visualització per les dades amb espècies agrupades trobem els següents resultats:

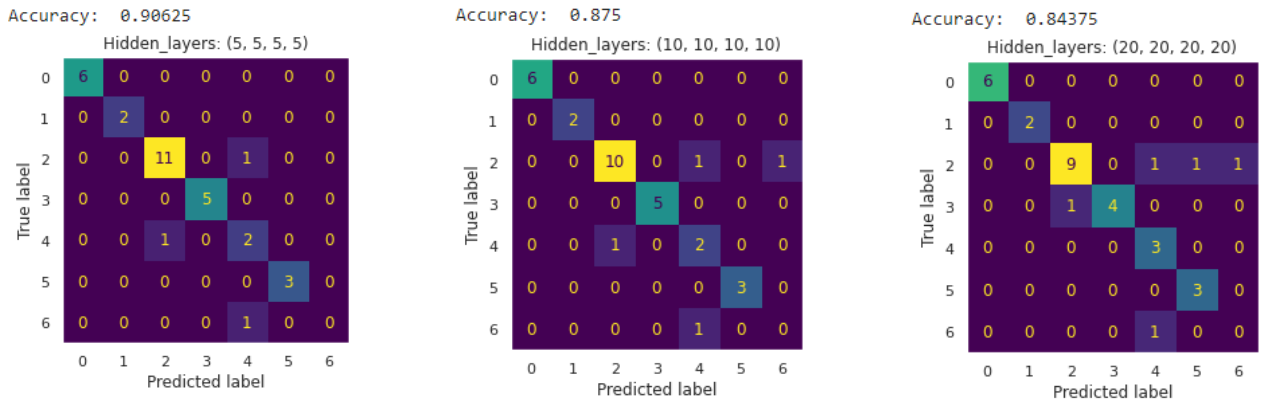



En aquest cas trobem una situació similar, a mesura que augmentem el número de veïns, disminuïm l'accuracy, ja que seguim tenint poques dades. També trobem el mateix error en la classe Roach

(etiqueta 4), que tindria sentit que en l'espai 4-dimensional on trobem les dades, tampoc sigues separable de les altres classes i per això tinguéssim problemes per predir-la.

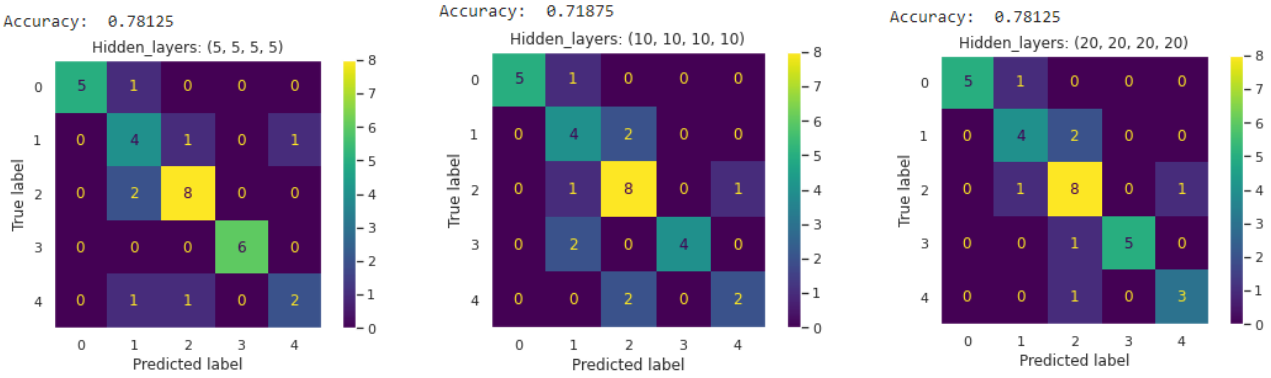
## Xarxes neuronals

Les xarxes neuronals són una estructura de capes de perceptrons que permet resoldre problemes de classificació. Els perceptrons són un neurones artificials que simulen el comportament d'una neurona biològica. Aquest mètode ens permet resoldre problemes on les dades siguin linealment separables en n dimensions. En el nostre cas, tal i com hem vist, no tenim dades separables en dues dimensions ja que contem amb 4 atributs i cap parella d'aquests ens conforma una separació de classes neta. Per tant, farem ús d'aquest algorisme. Utilitzarem diferents valors per els perceptrons de cada capa, però sempre amb 4 capes, ja que tenim 4 atributs.




Cal destacar que aquest és un algorisme més complex que els anteriors i tal i com podem observar obtenim els millors resultats. Destacar també que a mesura que augmentem el número de perceptrons per capa estem anant cap a un clar overfitting perquè no tenim tantes dades ni tants atributs com per entrenar una xarxa d'aquesta mida de forma correcta. Més endavant, triarem els hyperparàmetres de la millor xarxa que poguem entrenar.

Amb les espècies agrupades, en canvi, no obtenim uns resultats tant bons.




En general, podem dir que cada execució dels diferents algorismes variaven bastant els resultats d'accuracy. Això es degut a que tenim molt poques dades i depenent del traint i test que agafem els resultats poden ser molt diferents.

# Optimització d'hyperparàmetres

En l'apartat anterior, hem vist els resultats d'algunes combinacions de paràmetres que ens proporcionen els 4 models diferents que hem provat. Els valors d'aquests hyperparàmetres han estat escollits mitjançant Prova i Error ja que han sigut diferents proves manuals. En aquest apartat volem optimitzar aquests paràmetres de forma més acurada amb una altra tècnica, principalment podem provar dos: Grid Search i Random Search. Tenint en compte que tenim un dataset petit, per tant, computacionalment fàcil de treballar i farem ús de la tècnica del Grid Search que és un mètode de força bruta on l'ordinador prova totes les possibles combinacions de hyperparàmetres i utilitza el millor per entrenar el model. En canvi, amb el Random Search només algunes de les combinacions es provarien i ens quedaríem amb la millor d'aquestes, però no podríem assegurar que fos la millor. Aquest mètode esta bé quan tenim moltes dades. Per tant, amb Grid Search, al haver provat totes les possibilitats ens assegurem que els hyperparàmetres escollits siguin la millor combinació i doni els millors resultats. El mètode més comú per fer això és el k-fold cross-validation, que també ens servirà per controlar l'overfitting que segurament generin els models degut a la quantitat de mostres que tenim.

## Cross-validation

La validació creuada o cross-validation, és una tècnica que avalua un model a través de dividir-lo en subconjunts de dades d'entrada. És molt important fer validació creuada amb el conjunt de dades, ja que fer-ho ens ajudarà a detectar overfitting, i garantitzarà una millor generalització del model. Per tant, l'objectiu principal del cross-validation és mostrar una idea de com de bé funcionarà el nostre model per predir dades que encara no s'han vist.

Per dur a terme aquesta tècnica dividim el conjunt de training en 'k' subconjunts, de tots aquests 1 s'escull per validar el model (set de validació) i els altres 'k-1' subconjunts s'utilitzen per entrenar el model. Aquest procés es repeteix mentre que dels 'k' conjunts tots s'hagin fet servir per validació. Es calcula l'accuracy per cada iteració i per determinar l'accuracy final és fa la mitja. Això podem estendre-ho a la optimització de paràmetres ja que es realitza una validació creuada per cada combinació d'hyperparàmetres. La combinació amb millor accuracy, serà la considerada la millor per aquest model. Per cadascun d'aquests, assenyalarem en negreta els paràmetres que l'algorisme ha considerat els millors en la majoria d'execucions.

Com comptem amb poques dades, farem servir validació creuada de k = 2 a k = 10 per les dades agrupades (ja que hi ha més de 10 mostres per classe) i de k = 2 a k= 5 per les dades no agrupades (ja que l'atribut amb menys representació té 6 mostres). Per cadascuna d'aquestes mirarem quina és en general la millor combinació de paràmetres per cadascun dels models. Per fer això utilitzarem la funció GridSearchCV de sckit-learn. Cal destacar que amb el dataset amb les dades no agrupades, podem generar resultats fins a k = 6 ja que el nombre de mostres de la classe menys nombrosa es 6 i utilitzem un mètode d'estratificació pel qual mantenim la proporció original de mostres de cada classe en cada subset.

## Arbre de decisió

Dins d'aquest classificador trobem alguns hyperparàmetres interessants que haurem de provar per saber quin es el que dona millors resultats a l'hora d'entrenar el model. Penso que els paràmetres més determinants per entrenar el model son:

- criterion: la mesura que utilitzem per calcular la qualitat de la divisió d'un node.
    - **entropy** i gini
- Splitter: la estratègia que utilitzem per calcular la millor divisió.
    - **best**, random
- max_depth: la profunditat màxima que arribarà l'arbre.
    - 1, 2, 3, **4** (que és el número màxim d'atributs que tenim)
- max_features: el número d'atributs que es tenen en compte per mirar per la millor divisió.
    - qrt, log2, **None**
- class_weigth: el pes que se li atribueix a cada classe, si hi ha classes desbalancejades, aquest pes per cada classe serà l'inversament proporcional a la freqüència d'aquesta al conjunt de traint.
    - balanced, **None**

## SVM

Farem proves amb els hyperparàmetres:

- decision_function_shape: indica si utilitzem estratègia one-vs-rest o one-vs-one.
    - **ovo** , ovr
- class_weight: indica si volem ponderar les classes de forma inversa a la seva freqüència.
    - balanced , **None**
- kernel: especifica quin kernel utilitzarem per a entrenar el nostre algorisme.
    - **linear**, poly, rbf, sigmoid
- degree: paràmetre per indicar el grau si usem kernel polinomial.
    - **1**,2,3
- gamma: coeficient del kernel per kernels diferents al lineal.
    - **scale** , auto

## KNN

Els hyperparàmetres que s'han de d'analitzar en aquest cas són:

- n_neighbours: número de veïns utilitzats per determinar la classe d'una nova mostra.
    - De 1 a 20 (**depèn de l'execució i del dataset)**
- weights: ponderacions utilitzades per l'algorisme segons la distància que hi hagi a la mostra.
    - **Uniform** (dataset 7 espècies), **distance** (dataset 5 espècies)
- algorithm: algorisme matemàtic utilitzat per calcular els veïns més propers
    - **auto**, ball_tree, kd_tree, brute

## Xarxes neuronals

En aquest cas tenim molts hyperparàmetres possibles, només farem proves amb:

- hidden_layer_sizes: representa el número de neurones a cada capa oculta i el número de capes ocultes.
    - Pel dataset amb 7 espècies:

(5,5,5), (5,5,7), (5,5,5,7), (5,5,5,5), (10,10,10), (10,10,7), (10,10,10,7), (10,10,10,10), **depèn**

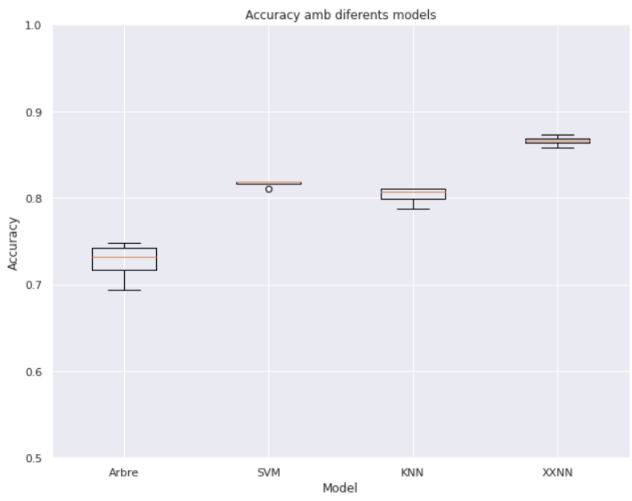
    - Pel dataset amb 5 espècies:

(5,5,5), , (5,5,5,5), **(10,10,10), (10,10,5),** (10,10,10,5), **(10,10,10,10)**

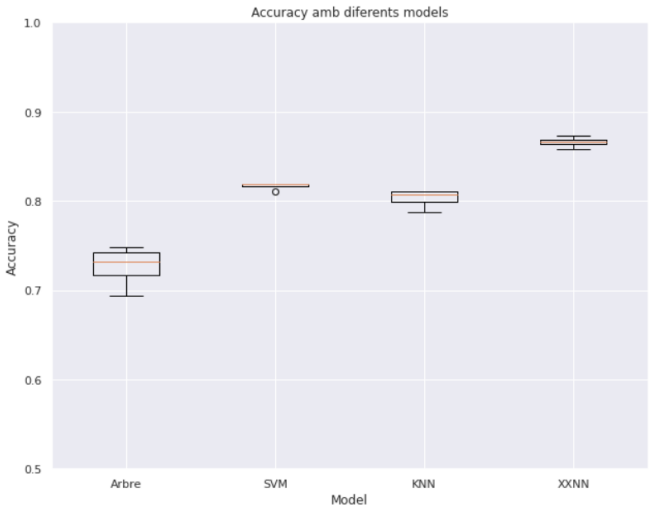
- activation: funció d'activació de la capa oculta.
    - **identity**, logistic, **tanh**, relu
- solver: solucionador de l'optimització de pes.
    - sgd, **adam**
- max_iter: màxim d'iteracions que deixem la xarxa entrenar-se, si no convergeix abans
    - 5000, **10000**

En aquesta dues imatges de baix podem veure les diferents precisions que ens han donat els 4 models en les execucions del cross-validation amb els millors hyperparàmetres. Tot i que tenim accuracy's similars, els millors resultats els veiem amb les Xarxes Neuronals en ambdós casos. Tot i així, cal destacar que també ha sigut l'algorisme més costos en temps. Mentre que tots els algoritmes estaven en l'ordre d'un segon aproximadament, les xarxes neuronals han trigat 20 minuts en el primer cas i prop de 45 minuts en el segon cas.

**Espècies no agrupades:**




**Espècies agrupades:**



# Lazy Predictor

Lazy Predictor és una llibreria de Python que semiautomatitza l'aprenentatge automàtic. Construeix 29 models bàsics diferents i ens proporciona informació sobre els resultats que aquets donen sense tenir en compte l'optimització de paràmetres.

En el nostre cas, només hem provat 4 models que m'han semblat que funcionarien millor tenint en compte l'aprenentatge teòric d'aquests. Tot i així, considero una bona idea utilitzar aquesta llibreria per descobrir si he estat encertada amb els models escollits o si existeixen altres mètodes que poden fer millorar l'accuracy aconseguida fins al moment. Per problemes de classificació s'utilitza la funció LazyClassifier i per problemes de regressió LazyRegressor. Nosaltres treballarem amb la primera esmentada.
Farem una visualització dels 10 millors resultats d'accuracy pel dataset amb les 7 espècies originals (que dona millors resultats que el dataset on hem manipulat el número d'espècies):

| Model | Accuracy | Balanced Accuracy | ROC AUC | F1 Score | Time Taken |
|---|---|---|---|---|---|
| QuadraticDiscriminantAnalysis | 0.91 | 0.94 | None | 0.91 | 0.01 |
| KNeighborsClassifier | 0.84 | 0.72 | None | 0.81 | 0.02 |
| XGBClassifier | 0.81 | 0.76 | None | 0.80 | 0.06 |
| LinearDiscriminantAnalysis | 0.78 | 0.69 | None | 0.77 | 0.02 |
| SGDClassifier | 0.78 | 0.68 | None | 0.74 | 0.01 |
| CalibratedClassifierCV | 0.78 | 0.67 | None | 0.70 | 0.14 |
| Perceptron | 0.78 | 0.67 | None | 0.70 | 0.01 |
| RandomForestClassifier | 0.78 | 0.62 | None | 0.75 | 0.18 |
| DecisionTreeClassifier | 0.78 | 0.83 | None | 0.78 | 0.03 |
| PassiveAggressiveClassifier | 0.75 | 0.65 | None | 0.67 | 0.02 |

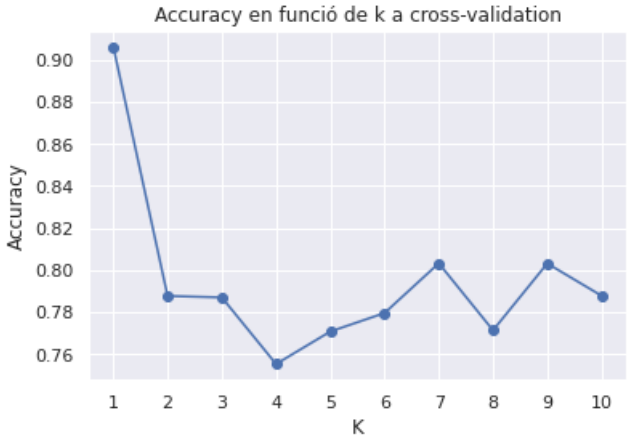
Podem observar que entre els millors trobem el KNN i l'arbre de decisió, dos dels algorismes escollits i desenvolupats durant l'informe. Per tal d'aconseguir el millor accuracy possible, intentarem aplicar el GridSearch explicat anteriorment amb l'algorisme que dona millor accuracy: el QuadraticDiscriminantAnalysis. Cal destacar que tot i que amb el dataset amb les espècies agrupades obtenim un 0.03 menys d'accuracy, també és el classificador que millor funciona.

## Quadratic Discriminant Analysis

L'anàlisi discriminant és un mètode de classificació supervisat en que un conjunt de mostres es classifiquen per classes segons les seves característiques. Aquest fa ús del Teorema de Bayes estimant la probabilitat de que una observació, donats uns valors concrets, pertanyi a cadascuna de les classes de l'atribut objectiu, d'aquesta forma la classe més probable serà la que doni com a resultat d'aquella mostra el predictor. Existeix anàlisi discriminant lineal que forma plans per separar les dades, per tant, resol problemes on les classes són linealment separables, però no es el nostre cas. Per tant, quan els

límits que s'han d'establir són corbes, perquè les classes no es separen de forma lineal, utilitzem el anàlisi discriminant quadràtic .

En la següent gràfica analitzem el resultat d'accuracy amb l'algorisme bàsic (k = 1) i amb optimització d'hyperparàmetres fent us del GridSearch i per diferents K's pel cross-validation. Observem que obtenim pitjors resultats que quan no utilitzem la optimització de paràmetres.




Com a última prova, s'ha modificat el valor del paràmetre test_size quan fem l'split i el millor resultat és amb test_size 0.1 que arribem fins a un 0.93% d'accuracy. Però, si afègissim noves mostres pel mateix model, probablement l'accuracy baixaria.

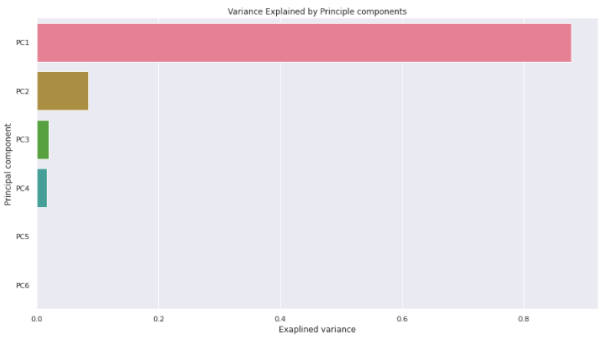
# Comparativa amb un altre treball

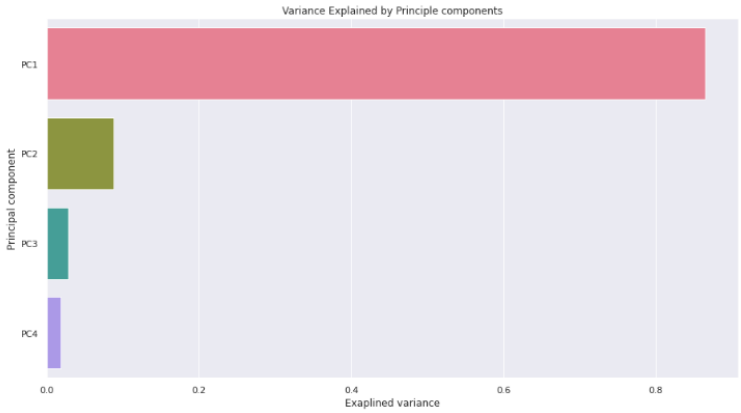
El següent apartat tractarà d'utilitzar un treball del Kaggle sobre la mateixa base de dades i amb al mateix objectiu per, principalment, acabar de completar el meu informe. Aquest treball s'anomena Multiple_Classification_Models i està escrit per l'usuari Mayur Kagathara (https://www.kaggle.com/code/mayurkagathara/multiple-classification-models).
Es tractaran dos temes principals: PCA i Regressió Logística.

## PCA

Una forma de fer més completa el meu anàlisi de components principals és, tal i com es fa en aquest treball, utilitzar la informació que ens proporciona per assegurar-nos que hem escollit de forma correcta els atributs a utilitzar al model, és a dir, que la decisió d'eliminar atributs molt correlacionats entre ells ha estat correcta. Per fer això, farem una visualització de la variança de cada component , és a dir, de la informació que ens aporta cada atribut.

Tal i com podem observar a la imatge de la dreta, hi ha dos atributs que no ens estan aportant gens d'informació i, juntament amb la informació sobre les correlacions podem afirmar que hem pres una decisió correcta. Tot i així, visualitzarem la mateixa informació però amb el dataset reduït (4 atributs).

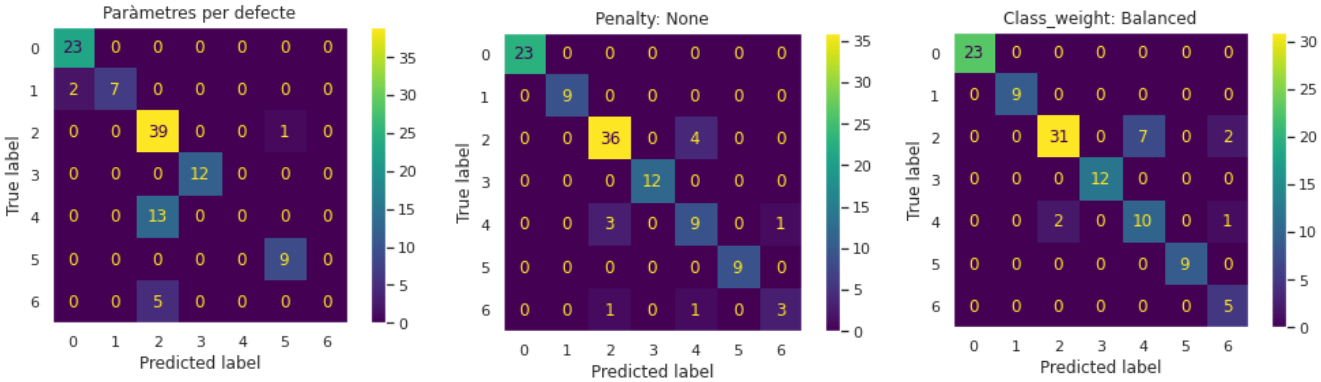



En aquest cas, les 4 components (atributs) que ens hem quedat en el dataset definitiu aporten cadascuna, en major o menor mesura, informació al model. Amb aquesta petita anàlisi extra, podem concloure i estar segurs de que hem pres una decisió que no perjudica als models que hem creat.

## Regressió Logística

Un model del que no s'ha parlat en aquest informe i que també pot encaixar perfectament dins dels nostres objectius de predicció és la Regressió Logística. No s'ha explicat aquest model des d'un principi ja que la majoria de treballs sobre aquesta base de dades utilitzaven aquest model i em cridava l'atenció l'idea de trobar altres models que no s'haguessin utilitzat i també funcionessin bé. Tot i així, en aquest apartat treballarem aquest model ja que dona bastants bons resultats. Així que, tal i com hem fet en els altres apartats, visualitzarem algunes imatges on els podrem observar.




Com podem veure obtenim millors resultats a la imatge d'enmig amb una accuracy de 0.8958. A la primer imatge podem observar que el model no ha sapigut aprendre correctament la classe amb etiqueta 4 (Roach), tot i tenir 20 mostres de representació (bastant més que altres), si mirem les distribucions al pairplot veiem que els punts estàn difosos barrejant-se per totes les parelles d'atributs amb altres classes. Per tant, al no tenir característiques especifiques, és més complicada una bona classificació. En canvi, quan tenim dades balancejades, observem que aquesta classe s'aconsegueix classificar en major mesura ja que se i dona més importància que a les que tenen més representació, per tant, hi ha un balancejament entre la ponderació i la representació de cada classe al model.

Al treball, a més, s'indica que tot i que l'accuracy sigui més baixa amb l'hyperparàmetre class_weigth a Balanced, utilitzar-lo previndrà el model d'overfitting i, conseqüentment, tindrem una millor generalització per classificar noves mostres.

# Suggerencies col·laboratives

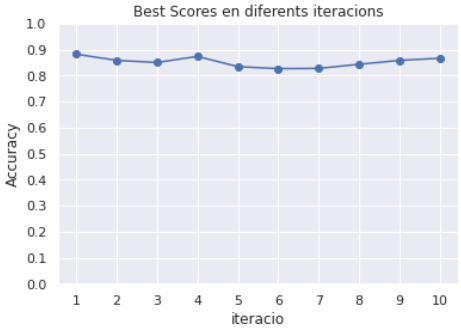
Un parell de d'elements a afegir en relació a la Regressió Lógistica realitzada per l'autor esmentat al principi seria: l'optimització d'hyperparàmetres, feature importance i entrenar el model amb les components del PCA. D'aquests tres aspectes de complementació parlarem a continuació.

## Optimització d'hyperparàmetres

Per aquest apartat hem fet les següents proves amb els hyperparàmetres:

- Penalty: especifica el tipus de penalització que s'aplica
  - l1,l2
- C: invers de la força de regularització
  - De 0.1 a 1
- class_weight: pes associat a cada classe per ajustar el desbalancejament
  - balanced, None

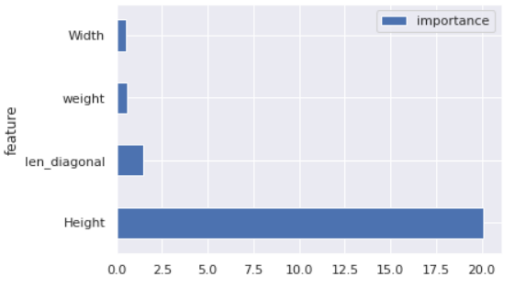
Com els resultats en altres models han variat molt segons les execucions, he decidit que per aquest model volia assegurar-me de que els paràmetres òptims s'ajustaven a diferents execucions. Per fer això, he 10 divisions en train i test diferents i he executat la funció GridSearchCV amb els diferents paràmetres amb una k del cross-validation igual a 4. Hem aconseguit unes accuracy's similars, com veiem en la gràfica de la dreta. Els millors paràmetres han estat:



balanced, l1, liblinear per totes les execucions, l'únic paràmetre que no podem assegurar que sigui un valor el millor és la C, ja que ha sigut variant en la majoria d'execucions.

## Feature Importance

Quan parlem de feature d'importance ens referim a un mètode utilitzat per saber dins d'un model, quins són els atributs que més ponderen la decisió de classificació d'aquest.

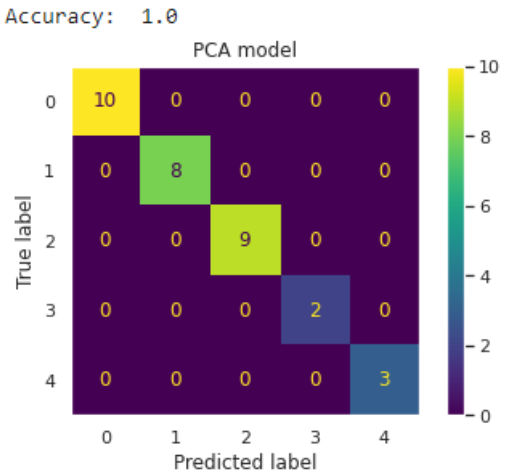


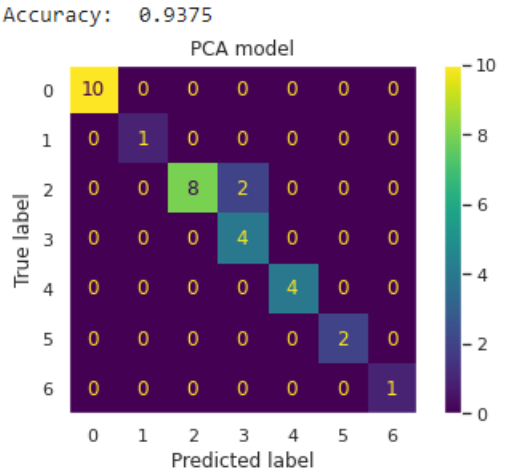

En aquest cas, hem visualitzat pel model de regressió logística amb el dataset amb 7 espècies quina és la importància de cada atribut. En aquest cas l'atribut que influeix en major mesura és clarament Height. Aquest resulta determinant per fer una classificació correcta. Això ho podíem intuir amb el pairplot de l'apartat de preprocessament ja que qualsevol parella d'atributs que compti amb Height, la seva representació resulta més senzilla de classificar, és a dir, les classes és veuen més separades que a les visualitzacions.

Podríem realitzar el mateix procediment amb la resta de models però molt probablement obtindríem els mateixos resultats.

## Regressió Logística amb PCA

Com a complementació d'aquest apartat, i com a última prova d'aquest informe tenim: entrenament de regressió logística amb les components del PCA. Tal i com s'ha esmentat a un dels primers apartats, amb el PCA la diferenciació de classes amb el dataset agrupat era perfecta, és per això, que penso que es una bona idea crear un model definitiu amb aquestes dues components com a dades d'entrenament. Visualitzarem la matriu de confusió tal i com hem mostrat en els apartats anteriors i veiem: a la esquerra els resultats amb el dataset amb 5 espècies, completament correcte i, a la dreta els resultats amb el dataset de 7 espècies que té errors en algunes mostres. En aquest últim, tal i com hem vist en la representació del PCA, esperaríem que l'algorisme confongués les classes Roach (4) i Pike (3) que són les que es solapen el l'eix de les Y. Però en la matriu de confusió ens indica que l'algorisme prediu com a etiqueta 3 (Pike) alguns dels peixos de l'espècie 2 (Perch). Això podria donar-se a que les mostres d'aquestes dues classes també és solapen en l'eix de les X.



# Conclusions

En primer lloc, m'ha semblat molt interessant realitzar aquesta pràctica. Tot i així, a diferència de les altres ha sigut més complicat prendre decisions ja que t'enfrontaves sol i lluitaves contra la teva pròpia opinió i això sempre resulta més complicat que comentar-ho amb altres companys. Però, ha servit per acabar de consolidar tots els coneixements teòrics de l'assignatura de forma individual, per tant, una pràctica d'enriquiment personal.

Un altre aspecte important en aquesta base de dades es que hem creat models que, degut a la poca quantitat de dades, són hipersensibles a les inicialitzacions. Per tant, tal i com s'ha comentat en algun apartat, cada cop que s'executaven els diferents models donava una precisió diferent. Això ha resultat un problema a l'hora d'extreure conclusions i, podria ser, que algun comentari en relació a la interpretació dels resultats dels diferents models hagi estat equivocat.

Per una persona com jo, aquets tipus de problemes es podrien allargar infinitament, ja que sempre se t'acudeixen milers de proves diferents a fer i noves idees, per tant, saber on parar i donar per finalitzat ell treball ha sigut el més complicat per mi. Però, tot i semblar un treball infinit, no ha sigut pesat de construir.

Resumint de forma tècnica la pràctica, durant tot l'informe i el notebook he intentat seguir un ordre lògic dels esdeveniments aconseguint així un treball fàcil de seguir i amb sentit. Tot i que els resultats no han sigut els millors degut, principalment, a l'escassa quantitat de dades que tenia la de base de dades que se m'ha assignat, estic satisfeta amb totes les proves que he realitzat i explicat en l'informe.

Per últim, s'ha inclòs la documentació i el notebook utilitzat a un repositori GitHub públic.[2]

[2] https://github.com/sereenaasg/Cas-Kaggle