Classificació

Iloc:	Institut d'Ensenyaments a Distància de les Illes Balears	Imprès per:	Carlos Sanchez Recio
Curs:	Sistemes d'aprenentatge automàtic	Data:	dilluns, 28 d'octubre 2024, 16:06
Llibre:	Classificació		

Taula de continguts

1. Classificació

En aquest apartat de **classificació**, estudiarem una selecció dels algorismes més populars i potents que s'utilitzen tant en l'entorn acadèmic com a la indústria. Mentre aprendrem les diferències entre els diferents algorismes d'aprenentatge automàtic per a classificació, apreciarem les seves respectives fortaleses i febleses. Tot això ho farem aprofitant les implementacions de la llibreria **scikit-learn**, que ofereix una interfície amigable i consistent per poder usar els algorismes de forma eficient i productiva.

Els temes que cobrirem són els següents.

- Presentació dels **algorismes** més robustos per a classificació: **regressió logística**, **màquines de suport vectorial**, **arbres de decisió**, **veïns més propers**.
- Exemples i explicacions usant la llibreria d'aprenentatge automàtic **scikit-learn**, que ofereix una varietat d'algorismes d'aprenentatge a través de la seva **API de Python.**
- Discussions sobre punts forts i febles dels classificadors, amb **fronteres de decisió lineals i no lineals**.

2. Tria de l'algorisme

Triar un algorisme de classificació adequat per a una tasca-problema particular demana pràctica i experiència; cada algorisme té les seves peculiaritats i es basa en unes determinades suposicions. No hi ha cap classificador que superi tots els altres en tots els escenaris. A la pràctica, és recomanable comparar els resultats d'un grapat d'algorismes d'aprenentatge diferents per triar el millor model per a un problema particular; les situacions poden ser diferents segons el nombre de característiques o exemples, la quantitat de soroll en el conjunt de dades, o si les classes són **separables linealment** o no.

El rendiment d'un classificador, tant del punt de vista de rendiment computacional com de poder predictiu, depèn fortament de les dades subjacents que hi hagi disponibles per a l'aprenentatge. Les cinc passes principals de l'entrenament d'un algorisme d'aprenentatge supervisat són les següents.

1. Selecció de **característiques** i recopilació d'exemples d'**entrenament** etiquetats.
2. Tria d'una **mètrica** de rendiment.
3. Elecció d'un **classificador** i un algorisme d'**optimització**.
4. **Avaluació** del rendiment del model.
5. **Refinament** de l'algorisme.

3. Regressió logística

En aquest capítol veurem un algorisme senzill i potent per aplicar a problemes de classificació binària, la regressió logística. Malgrat el seu nom, la **regressió logística** és un model per a **classificació**, no per a regressió.

La regressió logística és un model de classificació molt senzill d'implementar i que funciona molt bé quan les classes són separables linealment. És un dels algorismes més utilitzats per a classificació a la indústria. El model de regressió logística és un model per a classificació binària que pot ampliar-se a la classificació multiclasse, per exemple, mitjançant la tècnica **OvR**.

Per entendre la idea de la regressió logística com a model probabilístic, convé primer veure la **raó de probabilitats** (en anglès, *odds ratio*). Suposem un esdeveniment positiu, per exemple detectar un determinat fenomen. Convé aclarir que *positiu*, aquí, no significa necessàriament *bo*. Per exemple, en diagnòstic mèdic, la detecció d'una determinada malaltia s'entén com a esdeveniment positiu (cert, present), mentre que la situació de salut, sense la malaltia, es considera negativa. A aquest esdeveniment positiu se li assigna una **etiqueta de classe** $y = 1$. Si la probabilitat de l'esdeveniment positiu és p, aleshores la probabilitat de l'esdeveniment negatiu és $1 - p$. La raó d'aquestes dues probabilitats és $\frac{p}{1-p}$, i el seu logaritme és la funció logit.

$$logit(p) = \log\frac{p}{1-p}$$

La funció *log* aquí és el logaritme natural, de base e. La funció *logit* pren valors de p entre 0 i 1 (són probabilitats) i els transforma en tot el rang de nombres reals, que podem usar per expressar una relació lineal entre els valors d'entrada i el logaritme de la raó de probabilitats.

Expressem la probabilitat condicional que una mostra en concret pertanyi a la classe 1, donades les seves característiques $\mathbf{x}$ com a $p(y = 1 \mid x)$

Aleshores,

$$logit(p(y = 1 \mid x)) = w_1x_1 + w_2x_2 + \ldots + w_mx_m + b = \sum_{i=0}^{m} w_ix_i + b = \mathbf{w}^T\mathbf{x} + b$$

Analitzem amb una mica de detall l'expressió compacta $\mathbf{w}^T\mathbf{x} + b$. Començam amb vectors columna $\mathbf{w}$ i $\mathbf{x}$, formats pels pesos des de w_1 fins a w_m i les característiques des de x_1 fins a x_m.

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}$$

$$\mathbf{w} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

Quan aplicam l'operació de transposició, T, el vector columna $\mathbf{w}$ es transforma en un vector fila, $\mathbf{w}^T$.

$$\mathbf{w}^T = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}^T = \begin{pmatrix} w_1, w_2, \ldots, w_N \end{pmatrix}$$

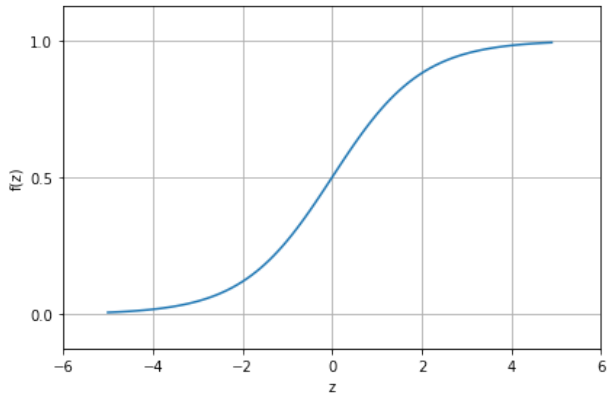
Llavors, el producte d'aquest vector fila $\mathbf{w}^T$ pel vector columna $\mathbf{x}$ constiteix en la suma dels productes dels elements corresponents dels dos vectors. Podem pensar que és un cas particular de multiplicació de matrius. La primera matriu és de dimensions $1 \times N$ i la segona matriu és de dimensions $N \times 1$. El resultat tendrà tantes files com la primera matriu i tantes columnes com la segona. Per tant, serà de dimensions \(1\times 1), un únic escalar.

$$\mathbf{w}^T\mathbf{x} = \begin{pmatrix} w_1, w_2, \cdots, w_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = w_1x_1 + w_2x_2 + \ldots + w_nx_N$$

Aquest resultat s'obté multiplicant el primer element de $\mathbf{w}$ amb el primer de $\mathbf{x}$, el segon de $\mathbf{w}$ amb el segon de $\mathbf{w}$, fins al darrer element de tots dos vectors. Tots aquests productes elementals se sumen i s'obté un únic valor, escalar, que sumarem finalment al biaix b.

Ara, el que ens interessa realment és predir la probabilitat que una determinada mostra pertanyi a una classe concreta, que és la forma inversa de la funció *logit*. També se li diu funció sigmoide logística, sovint abreujada funció sigmoide, a causa de la seva forma característica en S.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



La variable z és l'entrada de la funció, la combinació lineal de les característiques de la mostra, x_i, ponderada pels pesos w_i, més el biaix b. Aquest terme afegit permet desplaçar la funció, independentment del valor de les entrades.

$$z = \mathbf{w}^T\mathbf{x} + b = w_1x_1 + w_2x_2 + \ldots + w_mx_m + b$$

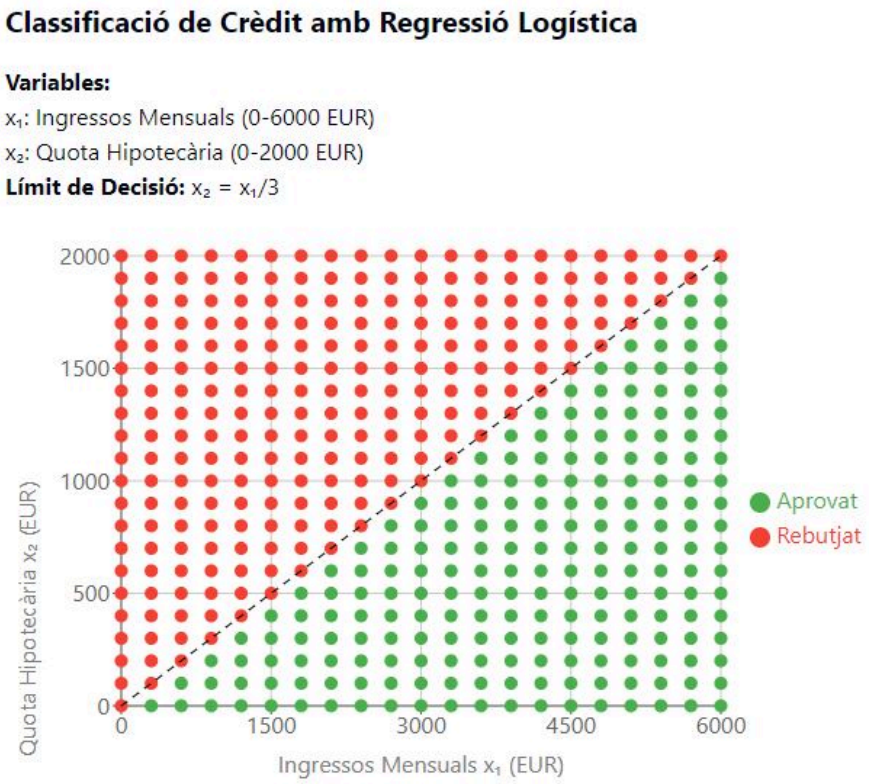
4. Exemple: classificació de crèdit

La regressió logística origina una frontera de decisió lineal en l'espai de les variables d'entrada. Si només hi ha dues variables d'entrada, som al pla, i aquesta frontera de decisió és una línia recta. Amb tres variables d'entrada, la frontera de decisió és un plans. En casos de dimensionalitat més alta, es parla d'hiperplans.

Vegem un exemple que per ventura ens serà familiar, que és el de la concessió d'un crèdit hipotecari. Una criteri general bastant senzill perquè la sol·licitud sigui acceptable és que els ingressos com a mínim tripliquin la quota de la hipoteca. Podem veure la forma de la regressió logística resultant a la següent visualització de WolframAlpha.

Regressió logística per a la concessió de crèdit hipotecari

Al pla, representant en verd la zona d'acceptació i vermell la zona de rebuig de la sol·licitud, obtenim el gràfic bidimensional següent.



Les entitats bancàries poden aplicar models d'anàlisi de risc molt més complexos que tenguin en compte un perfil més afinat del client, amb més variables d'entrada que poden influir en la decisió final. Aquest exemple senzill ens ha permès visualitzar una regressió logística de dues variables d'entrada, i la divisió del pla que genera en dues zones, separades per la frontera lineal de decisió.

5. Ajust dels paràmetres

Podem usar la regressió logística per predir probabilitats i etiquetes de classe. Ara, vegem com s'ajusten els paràmetres del model, els pesos \(\boldsymbol{w}\) i el biaix \(b\). Per començar, definim la funció de cost quadràtica següent.

\[J(\boldsymbol{w},b) = \frac{1}{2}\sum_{i} (\phi(z^{(i)})-y^{(i)})^2 \]

Recordem que la \(\phi(z) \) és la funció sigmoide de l'apartat anterior i \(y^{i}\) les etiquetes de classe. L'índex \(i \) s'estén d'\(1\) a \(N\), on \(N\) és la mida del conjunt de dades de què disposam. A partir d'aquí, definim la funció **log-likelihood** que es maximitzarà. Podem traduir el terme anglès *likelihood* per **versemblança**, el qual representa com de bé els paràmetres del model \(\boldsymbol{w}\) i \(b\) descriuen les nostres dades. Té l'expressió matemàtica següent.

\[l(\boldsymbol{w},b) = log L(\boldsymbol{w},b) = \sum_{i=1}^N \left[y^{(i)}log(\phi(z^{(i)})) + (1-y^{(i)})log(1-\phi(z^{(i)})))\right] \]

S'hi fa servir la funció logarítmica per dues raons: per estabilitat numèrica i per poder derivar-la (per obtenir-ne després el màxim) aplicant la suma de les derivades dels termes. Aquesta operació és més senzilla que no si la féssim sobre la funció *likelihood* original, que té un producte en lloc d'una suma. Ara convé transformar aquesta funció *log-likelihood* en una funció de cost, simplement canviant-li el signe. D'aquesta manera, podrem minimitzar-la (en lloc de maximitzar la funció de *likelihood*, que seria equivalent) utilitzant el descens de gradient. La funció de cost queda així.

\[J(\boldsymbol{w},b) = -\sum_{i=1}^N \left[y^{(i)}log(\phi(z^{(i)})) + (1-y^{(i)})log(1-\phi(z^{(i)})))\right] \]

Per poder interpretar-la més fàcilment, observem el seu valor per a un sol punt de dades, descrit per \(\phi(z)\) i \(y\), en lloc de la suma de tots ells.

\[J(\phi(z),y;\boldsymbol{w},b) = -ylog(\phi(z)) -(1-y) log(1-\phi(z)) \]

Observem que quan \(y = 0\) el primer terme val zero, i quan \(y = 1\) el segon terme val zero.

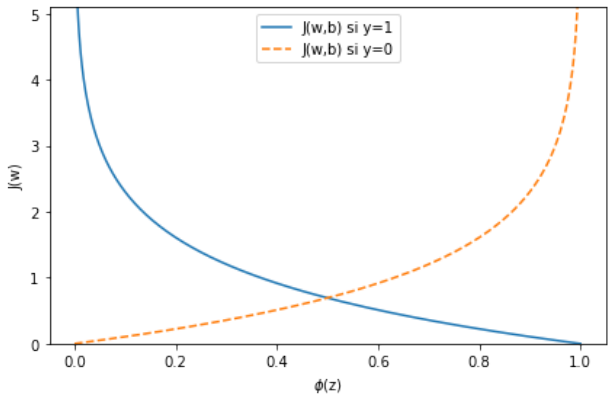
\[J(\phi(z),y; \boldsymbol{w},b) = -log(\phi(z)) \]

si \(y = 1\);

\[J(\phi(z),y; \boldsymbol{w},b) = -log(1-\phi(z)) \]

si \(y = 0 \).

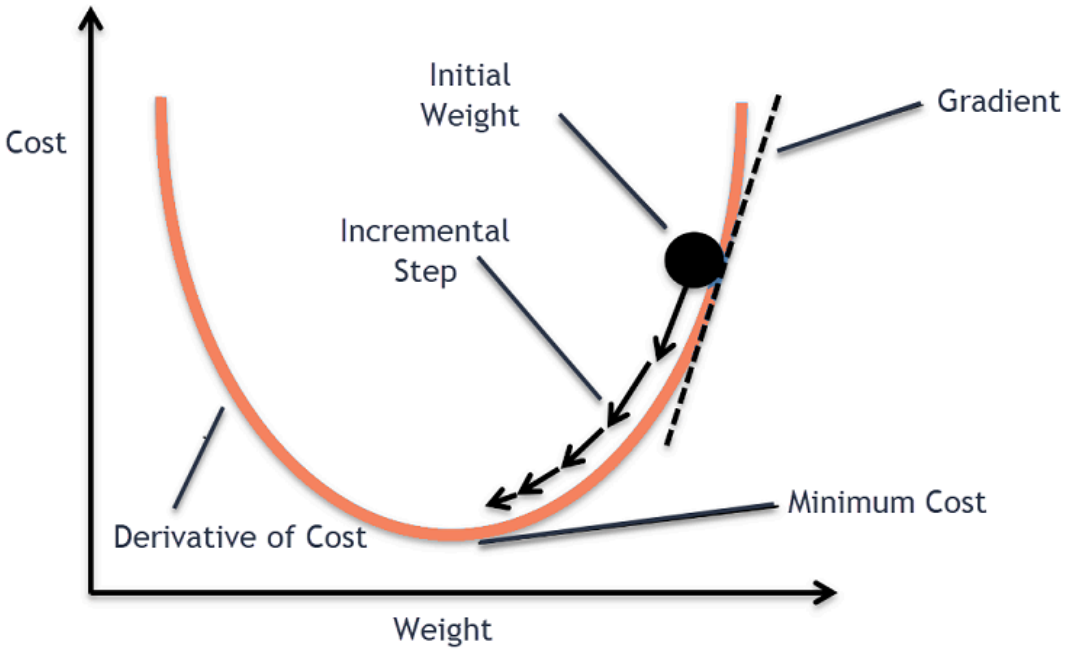
Vegem un gràfic que il·lustra la funció de cost per a un únic exemple d'entrenament segons el valor de l'etiqueta de classe. La figura mostra la funció d'activació sigmoide a l'eix de les x en el marge entre \(0\) i \(1\) (les entrades a la funció sigmoide eren valors de \(z\) entre \(-10\) i \(10\)) i el cost logístic associat a l'eix de les \(y\).



Veim que el cost s'acosta a zero (línia contínua) si prediem correctament que l'exemple pertany a la classe \(1\). Igualment, veim a l'eix de les \(y\) que el cost també és zero si prediem l'etiqueta de classe \(y=0\) (línia discontínua). En canvi, si la predicció és incorrecta, el cost tendeix a infinit. La qüestió important és que penalitzam les prediccions incorrectes amb un cost més gran.

6. Descens de gradient

La idea principal del descens de gradient es pot pensar com davallar una muntanya fins que s'arriba al punt més baix de la vall. En cada iteració, es fa una passa en la direcció oposada al gradient. La mida de la passa depèn de la taxa d'aprenentatge i del pendent de la funció de cost.



Imatge: *Descens de gradient*

Podem veure l'algorisme en funcionament amb el visualitzador https://uclaacm.github.io/gradient-descent-visualiser

6.1. Deducció de les fórmules (opcional)

Fent servir càlcul, obtindrem la regla d'aprenentatge del descens de gradient per a la regressió logística. Aquest apartat és fora dels objectius del curs, però és interessant des del punt de vista matemàtic, i es pot seguir amb una base de derivades. De tota manera, no és necessari per assolir els objectius del curs, de forma que es pot obviar sense més problemes.

Començarem calculant la derivada parcial de la funció *log-likelihood* respecte del pes j-èsim.

$$\frac{\partial}{\partial w_j} l(\boldsymbol{w}) = \left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{1-\phi(z)} \right) \frac{\partial}{\partial w_j}\phi(z)$$

Abans de continuar, calculem també la derivada parcial de la funció sigmoide.

$$\frac{\partial}{\partial z} \phi(z) = \frac{\partial}{\partial z} \frac{1}{1+e^{-z}} = \frac{1}{(1+e^{-z})^2}e^{-z} = \frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right) = \phi(z)(1-\phi(z))$$

Ara substituïm aquest resultat a l'equació de dalt.

$$\left(y\frac{1}{\phi(z)} -(1-y)\frac{1}{1-\phi(z)} \right) \frac{\partial}{\partial w_j}\phi(z)$$
$$= \left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{1-\phi(z)} \right) \phi(z)\left(1-\phi(z) \right) \frac{\partial} {\partial w_j}z$$
$$= \left(y\left(1-\phi(z) \right) -(1-y)\phi(z)\right)x_j$$
$$= (y-\phi(z))x_j$$

L'objectiu és trobar els pesos que maximitzen la *log-likelihood*, de forma que actualitzam els pesos de la forma següent.

$$w_j \leftarrow w_j + \eta \sum_{i=1}^n \left(y^{(i)} - \phi(z^{(i)}) \right)x_j^{(i)}$$

7. Implementació

En aquest quadern de Colab tenim una implementació completa d'un classificador mitjançant regressió logística entrenat mitjançant descents de gradient.

Dins la classe **LogisticRegressionGD** trobam implementats els mètodes següents.

- **__init__** és el constructor de la classe, al qual se li introdueixen els paràmetres de l'entrenament.
- **fit** és el mètode que ajusta el model. Rep com a paràmetres les variables d'entrada X i les etiquetes de sortida y.
- **compute_net_activation** calcula l'entrada a la funció d'activació, la combinació lineal de les variables d'entrada multiplicades pels seus pesos corresponents més el biaix.
- **compute_activation**, a partir del valor d'entrada a la funció d'activació, calcula la sortida de la funció logística.
- **predict** passa el resultat de la funció d'activació a través d'un esglaó unitari, per obtenir una predicció de la categoria de la sortida.

8. Aplicació a dades reals

Ara que tenim descrita la regressió logística, és el moment d'aplicar-la a un conjunt de dades reals.

Treballarem aquest classificador sobre el conjunt de dades Iris, referit a tipus de flors. Hi ha tres etiquetes de classe:

1. Iris Setosa
2. Iris Versicolour
3. Iris Virginica

I els atributs a partir dels quals es poden predir les etiquetes de classe són quatre: l'amplada i la longitud del sèpal i l'amplada i la longitud del pètal.

La mostra es divideix en dues parts:

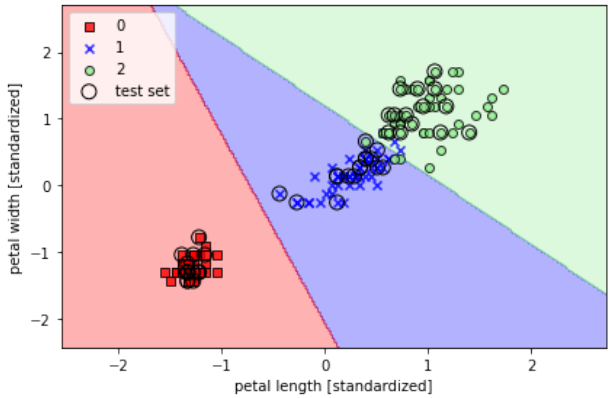
1. Un 70% per a l'entrenament
2. Un 30% per a l'avaluació

Després, les dades es normalitzen amb un escalat estàndard, que les transforma de manera que la seva mitjana passa a ser \(0\) i la desviació típica passa a ser \(1\).

Els paràmetres de la normalització (mitjana i desviació típica) es calculen a partir del conjunt d'entrenament i la normalització s'aplica als dos conjunts, d'entrenament i de test.

És important avaluar els models amb unes dades que no s'hagin utilitzat durant l'entrenament. Així mesuram la seva capacitat de generalització.

Una vegada seleccionades les dades i transformades, el resultat d'aplicar la regressió logística és el següent.

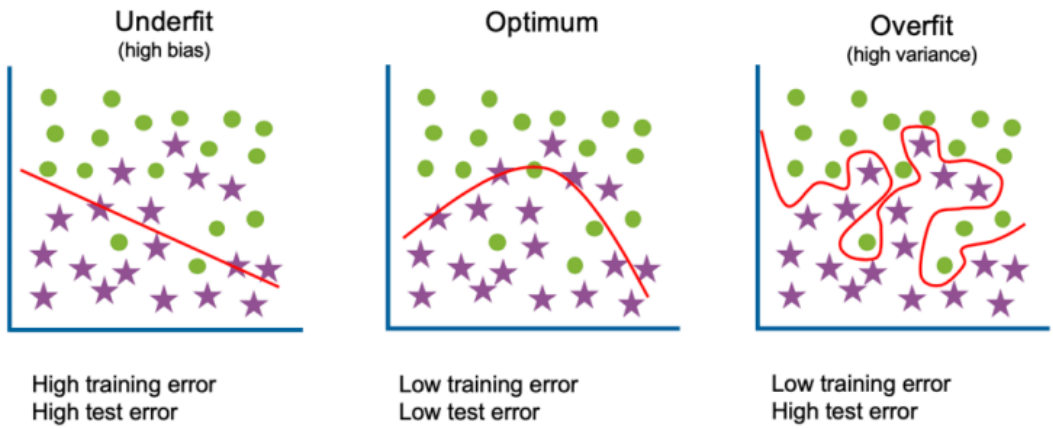


Al quadern de Colab de classificació hi ha el codi d'aquest exemple i de la resta de tècniques que es mostren a continuació.

9. Regularització: compromís biaix-variància

Un problema habitual de l'aprenentatge automàtic és el **sobreajust** (*overfitting*), en què un model funciona bé amb les dades d'entrenament però no generalitza prou bé a dades no vistes abans (dades de prova). Si un model pateix sobreajust, també deim que el model té variància alta, que pot ser deguda a tenir massa paràmetres, això vol dir massa complexitat donades les dades d'entrenament. D'una forma semblant, un model també pot tenir **infraajust** (*underfitting*). En aquest cas el model no és prou complex per capturar la distribució de les dades d'entrada i com a resultat també presenta un baix rendiment sobre dades noves.

Podem il·lustrar les situacions de sobreajust, ajust adequat i infraajust amb les següents fronteres de decisió.



Imatge: *Infraajust, ajust adequat, sobreajust*

https://www.linkedin.com/pulse/overfitting-underfitting-machine-learning-ml-concepts-com/

AMPLIACIÓ

Sovint s'usen els termes **biaix**, **variància**, o **compromís biaix-variància** per descriure el comportament d'un model. Podem trobar que un model presenta **variància alta** (*high variance*) o **biaix alt** (*high bias*). Pero això què vol dir? En general, podem dir que variància alta correspon a sobreajust i biaix alt a infraajust.

En el context de models d'aprenentatge automàtic, la **variància** mesura la consistència (o variabilitat) de la predicció del model per classificar un exemple particular si es reentrena el model moltes vegades, per exemple, sobre diferents subconjunts del conjunt d'entrenament. Podem dir que el model és sensible a l'aleatorietat de les dades d'entrenament. En canvi, el **biaix** mesura com d'enfora són les prediccions són dels valors correctes en general si reconstruïm el model sobre diferents conjunts d'entrenament; el biaix és la mesura de l'error sistemàtic que no és degut a l'aleatorietat.

Una forma de trobar un bon compromís biaix-variància és ajustar la complexitat del model a través del que s'anomena **regularització**. La regularització és un mètode molt útil per gestionar la col·linealitat (correlació elevada entre característiques), filtrar el soroll de les dades, i prevenir el sobreajust.

La idea de la regularització és introduir una informació adicional (biaix) per penalitzar valors extrems dels paràmetres, dels pesos. La forma més habitual de regularització és l'anomenada regularització L2, que correspon a l'expressió següent.

$$ \frac{\lambda}{2} ||\boldsymbol{w}||^2 = \frac{\lambda}{2} \sum_{j=1}^{m} = {w_j}^2 $$

Aquí, λ és l'anomenat **paràmetre de regularització**.

Regularització i normalització de característiques

> La regularització és una altra raó per la qual l'escalat de característiques, com per exemple l'estandardització, és important. Perquè la regularització funcioni correctament, cal assegurar que totes les característiques estan en escales comparables.

La funció de cost de la regressió logística es regularitza afegint-hi el terme de regularització, que reduirà els pesos durant l'entrenament del model.

\[J(\boldsymbol{w},b) = \sum_{i=1}^n \left[-y^{(i)}log(\phi(z^{(i)})) -(1-y^{(i)})log(1-\phi(1-z^{(i)})) \right] +\frac{\lambda}{2}|| \boldsymbol{w}||^2\]

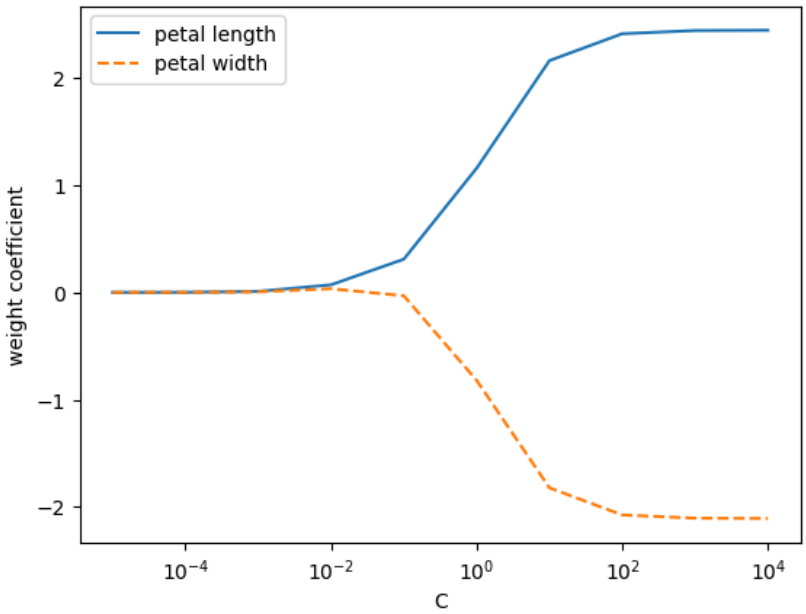
A través del paràmetre de regularització, \(\lambda \), podem controlar l'ajust del model a les dades, mantenint els pesos petits. Pujant el valor de \(\lambda\) puja la força de la regularització.

A la classe de scikit-learn LogisticRegression, el paràmetre \(C\) ve d'una convenció de les màquines de suport vectorial, que veurem a l'apartat següent. El terme \(C\) està relacionat amb el paràmetre de regularització \ (\lambda\), que n'és l'invers. D'aquesta forma, reduir el paràmetre invers de regularització \(C\) és augmentar la força de regularització . Vegem-ho amb el següent gràfic.

```
weights, params = [], []
for c in np.arange(-5, 5):
     lr = LogisticRegression(C=10.**c, random_state=1,
                             solver='lbfgs', multi_class='ovr')
     lr.fit(X_train_std, y_train)
     weights.append(lr.coef_[1])
     params.append(10.**c)
weights = np.array(weights)
plt.plot(params, weights[:, 0],
          label='petal length')
plt.plot(params, weights[:, 1], linestyle='--',
          label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='upper left')
plt.xscale('log')
plt.show()
```

Al codi anterior, hem ajustat deu models de regressió amb diferents valors del paràmetre invers de regularització \ (C\).

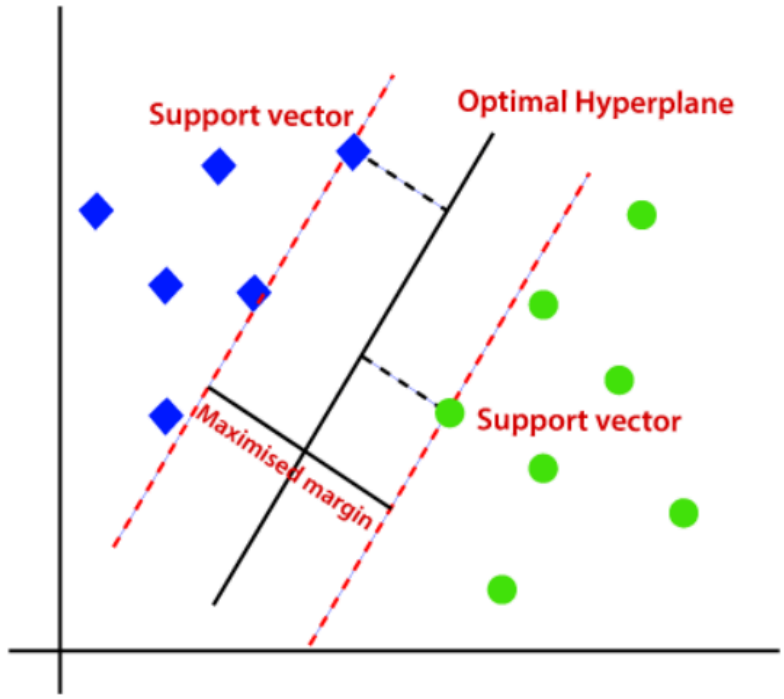
Com podem veure al gràfic següent, els coeficients dels pesos es fan petits si reduïm el paràmetre \(C\), és a dir, si augmentam la força de la regularització.



Imatge: *Regularització*

10. Màquines de suport vectorial

El següent algorisme de classificació que tractam és també potent i àmpliament usat. Són les **màquines de suport vectorial** (en anglès *Support Vector Machines*). En les màquines de suport vectorial, es maximitza el marge de seguretat en la classificació dels punts de dades. El marge es defineix com la distància entre l'hiperplà de separació (la frontera de decisió) i els exemples d'entrenament més propers a aquest hiperplà, que reben el nom de **vectors de suport**. Ho il·lustram amb la figura següent.



La motivacio de tenir fronteres de decisió amb marges grans és que tendeixen a tenir un error de generalització més baix, mentre que els models amb marges petits tendeixen al sobreajust (*overfitting*). Per tenir una idea de la maximització del marge, vegem els hiperplans positiu i negatiu paral·lels a la frontera de decisió, que podem expressar de la forma següent.

$$ w_0 + w^Tx_{pos} = 1 $$

$$ w_0 + w^Tx_{neg} = -1 $$

Si restam aquestes dues equacions, obtenim la següent.

$$ w^T(x_{pos}-x_{neg}) = 2 $$

Aquesta equació es pot normalitzar per la longitud del vector de pesos w, definit de la forma següent

$$ ||w|| = \sqrt{\sum_{j=1}^m w_{j}^{2}} $$

El costat esquerre d'aquesta equació es pot interpretar com la distància entre l'hiperplà positiu i el negatiu: és el **marge** que volem maximitzar.

Ara, entrenar la SVM consisteix en maximitzar aquest marge, $ \frac{2}{||w|| } $ complint la restricció que tots els exemples estiguin classificats correctament. Això ho podem escriure de la forma següent.

$$ w_0 + w^Tx^{(i)} \geq 1 $$

si $ y ^{(i)} = 1 $, i

$$ w_0 + w^Tx^{(i)} \leq -1 $$

si $ y ^{(i)} = -1 $, per a tots els exemples del conjunt d'entrenament $x^{(i)}$, des d'1 a N.

Aquestes equacions signifiquen que tots els exemples de la classe negativa han de caure a un costat de l'hiperplà negatiu, mentre que tots els exemples de la classe positiva han de caure a l'altre costat de l'hiperplà positiu. Les dues condicions es poden agrupar en una de sola de la forma següent.

$$ y^{(i)} (w_0 + w^Tx^{(i)}) \geq 1 $$

per a tots els i d'1 a N.

A la pràctica, però, és més senzill minimitzar el terme recíproc, $ \frac {||w||^2} {2} $, mitjançant programació quadràtica. No aprofundirem més en les matemàtiques dels SVM.

Cas no separable linealment amb variables auxiliars

No sempre estam en una situació que les classes siguin separables linealment mitjançant un hiperplà. Aleshores, interessa introduir una variable auxiliar $ eps $, que du el model a ser un classificador **soft-margin**. La motivació per introduir aquesta variable auxiliar és que les restriccions lineals (que tots els exemples quedin a la banda correcta del seu hiperplà) s'han de relaxar quan hi ha errors de classificació, amb una penalització de cost adequada.

Aquesta variable auxiliar, positiva, s'afegeix a les restriccions lineals.

$$ w_0 + w^T x^{(i)} \geq 1 - \xi^{(i)} $$

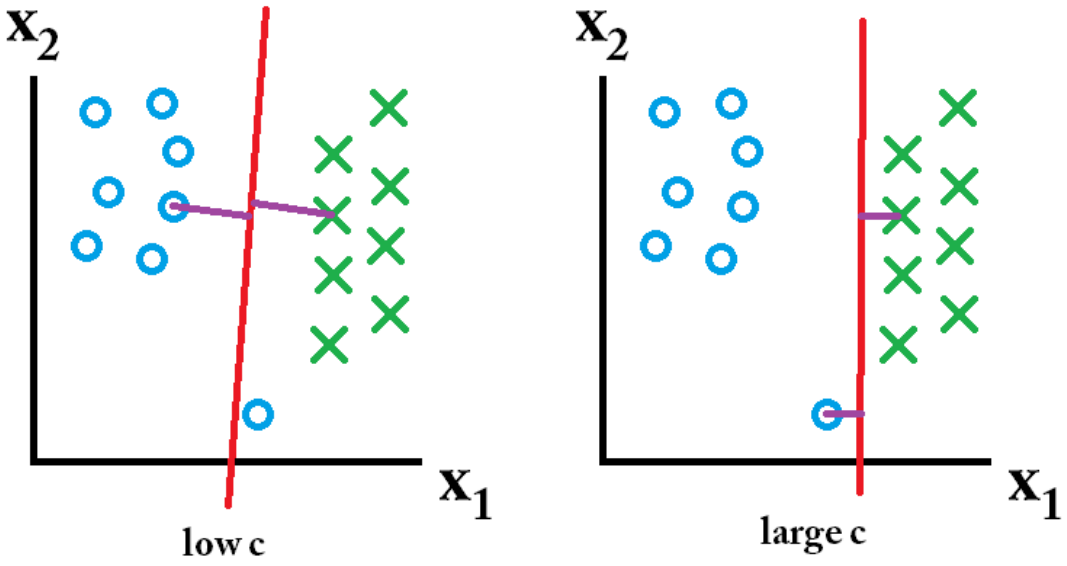
si $ y^{(i)} = 1$, la classe positiva, i

$$ w_0 + w^T x^{(i)} \leq -1 + \xi^{(i)} $$

si $ y^{(i)} = 1$, la classe negativa, per a tots els exemples $i $, de l'$1 $ a l'N. Ara, la nova funció objectiu que l'ha de minimitzar (obeint les restriccions) és la següent

$$ \frac{||w||^2}{2} + C \sum_i \xi^{(i)} $$

La variable $ C $ controla la penalització dels errors de classificació. Amb valors grans de C, es penalitzen molt els errors de classificació, mentre que si som més tolerants, feim servir valors de C petits. Amb C, doncs, es controla el marge del classificador.



Després d'haver vist les bases de la màquina de suport vectorial SVM, podem aplicar-la per classificar les flors del conjunt de dades Iris.

```
from sklearn.svm import SVC

svm = SVC(kernel='linear',C=1.0, random_state=1)

svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined, classifier = svm,
                      test_idx = range(105,150))

plt.xlabel('petal length [standardized] ')
plt.ylabel('petal width [standardized] ')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

11. Kernel SVM

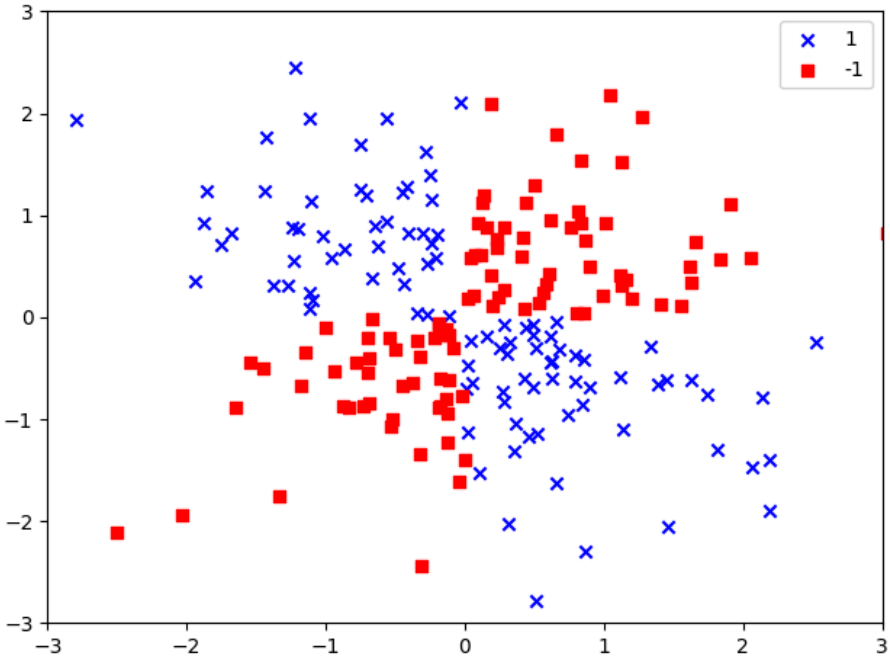
Una de les raons per les quals les SVM són populars en aprenentatge automàtic és per la facilitat amb què es poden **kernelitzar** per resoldre problemes de classificació no-lineal. Abans de veure el concepte de kernel SVM, crearem un conjunt de dades sintètic per veure quin aspecte pot tenir un problema de classificació no lineal com aquest.

11.1. Dades no separables linealment

Usant el codi següent, crearem un conjunt de dades simple que pren la forma de la porta \(XOR\), amb la funcio **logical_xor** de NumPy. 100 exemples tindran assignada l'etiqueta de classe \(1\), i 100 exemples tendran assignada l'etiqueta de classe \(-1\).

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(1)
X_xor = np.random.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0,
                       X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)
plt.scatter(X_xor[y_xor == 1, 0],
            X_xor[y_xor == 1, 1],
            c='b', marker='x',
            label='1')
plt.scatter(X_xor[y_xor == -1, 0],
            X_xor[y_xor == -1, 1],
            c='r',
            marker='s',
            label='-1')
plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Després d'executar el codi, obtindrem un conjunt de dades \(XOR\) amb soroll aleatori, com podem veure al gràfic següent.



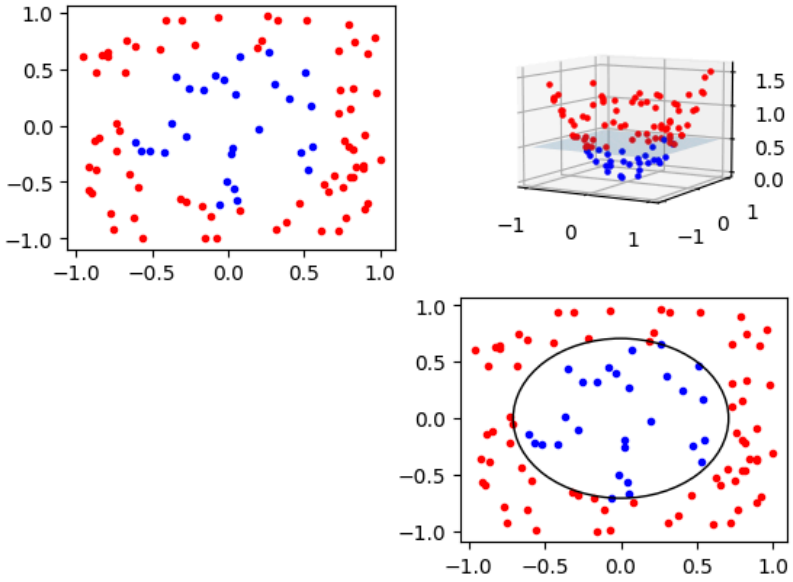
Imatge: *Dades no separables linealment*

No podrem separar els exemples de les classes positiva i negativa usant un hiperplà lineal com a frontera de decisió, ni amb la regressió logística ni amb una màquina de suport vectorial.

La idea principal dels mètodes de kernel és tractar aquestes dades no separables linealment creant combinacions no lineals de les característiques d'entrada per projectar-les en un espai de més dimensions a través d'una funció d'aplicació \(\phi\), de forma que les dades tornin separables linealment. Com podem veure al gràfic següent, podem transformar un conjunt bidimensional en un espai de característiques tridimensionals, en què les classes passen a ser separables a través de la projecció següent.

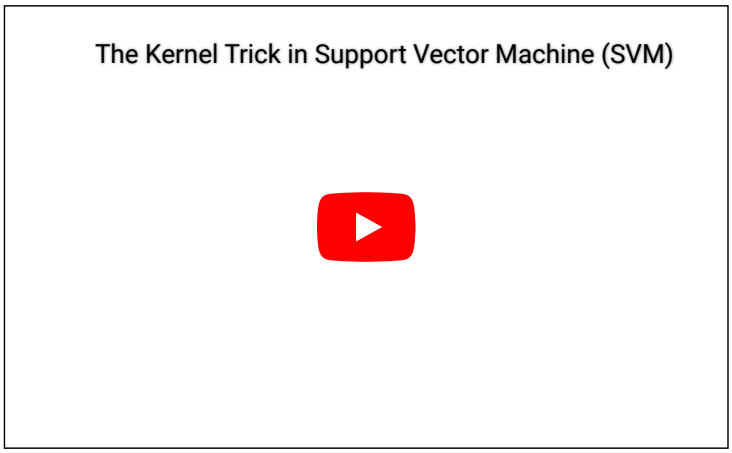
$$ \phi(x_1,x_2) = (z_1,z_2,z_3) = (x_1,x_2,{x_1}^2+{x_2}^2) $$

Això permet separar les dues classes que es mostren a la figura, a través d'un hiperplà que esdevé una frontera de decisió no lineal (una circumferència) quan es projecta en l'espai original bidimensional.



Imatge: *Separació no lineal de dues classes amb el mètode del nucli*

Podeu veure uns gràfics semblants als de la imatge en el següent vídeo sobre el mètode del nucli en màquines de suport vectorial.



Vídeo: *Mètode del nucli al canal Visually Explained*

11.2. Hiperplans de separació en espais multidimensionals

Per resoldre un problema no lineal usant un SVM, transformam les dades d'entrenament dins un espai de característiques multidimensional a través d'una funció d'aplicació \(\phi\), i entrenam un model lineal SVM per classificar les dades en aquest nou espai. Després, usam aquesta mateixa funció \(\phi\) per transformar dades noves i poder-les classificar amb el model.

No obstant això, un problema que hi ha amb aquesta aproximació d'aplicació és que la construcció de noves característiques és molt costosa computacionalment, sobretot si treballam amb dades multidimensionals. Aquí és on entra en joc el **mètode del nucli** (*kernel trick*).

A la pràctica, per entrenar un SVM, basta que substituïm el producte escalar \(x^{(i)T}x^{(j)} \) pel producte de funcions \(\phi(x^{(i)})^T\phi(x^{(j)})\).

Un dels nuclis (*kernels*) més usats és el de **funció de base radial** (*Radial Basis Function*, RBF), que es pot anomenar simplement **nucli gaussià**.

\[\kappa(x^{(i)},x^{(j)}) = exp \left(-\frac{|| x^{(i)} - x^{(j)} ||^2}{2\sigma^2} \right)\]

Això es pot simplificar com:

\[\kappa(x^{(i)},x^{(j)}) = exp \left(-{\gamma || x^{(i)} - x^{(j)} ||^2} \right) \]

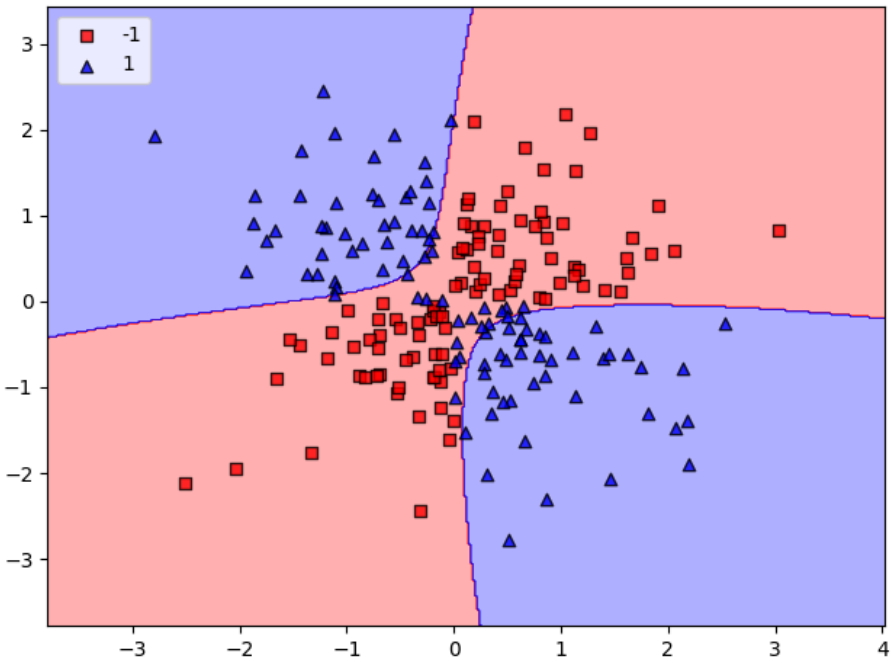
Aquí \(\gamma = \frac{1}{2{\sigma}^2} \) és un paràmetre lliure que s'ha d'optimitzar.

Podem interpretar el terme kernel com a funció de similitud entre un parell d'exemples. El signe menys transforma la mesura de distància en una mesura de semblança. A causa del terme exponencial, la mesura de semblança serà entre \(1\) (per a exemples exactament semblants) i \(0\) (per a exemples molt dissemblants).

Després d'aquesta descripció, vegem si podem entrenar un kernel-SVM que pugui traçar una frontera de decisió no lineal que separi bé les dades XOR. Al codi, usarem simplement la classe SVC de scikit-learn que hem importat abans i substituïm el paràmetre `kernel='linear'` per `kernel='rbf'`.

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(1)
X_xor = np.random.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0,
                       X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)
plt.scatter(X_xor[y_xor == 1, 0],
            X_xor[y_xor == 1, 1],
            c='b', marker='x', label='1')
plt.scatter(X_xor[y_xor == -1, 0],
            X_xor[y_xor == -1, 1],
            c='r', marker='s', label='-1')
plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Com podem veure al gràfic resultant, el kernel-SVM separa les dades XOR relativament bé.

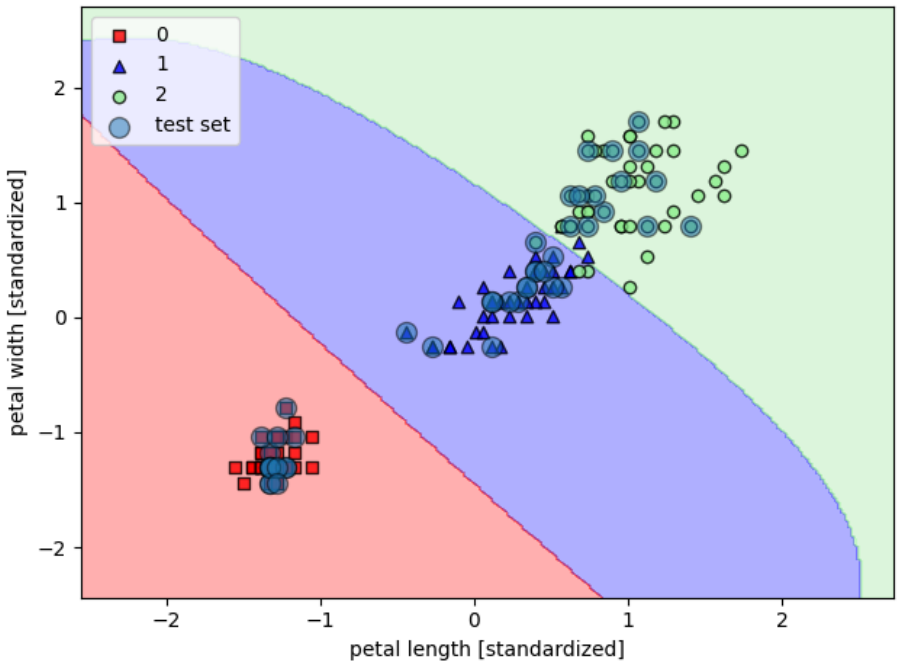


Imatge: *SVM amb kernel aconsegueix separar regions complexes*

El paràmetre \(\gamma\), que podem fixar a gamma=0.1, es pot entendre com un paràmetre de tall per a l'esfera gaussiana. Si augmentam el valor de \(\gamma\), augmentam la influència o abast dels exemples d'entrenament, cosa que du a una frontera de decisió més ajustada i accidentada. Per entendre més bé aquest paràmetre \(\gamma\), apliquem una SVM amb kernel RBF al conjunt de dades florals Iris.

```
svm = SVC(kernel='rbf', random_state=1, gamma=0.2, C=1.0)
svm.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std,
                      y_combined, classifier=svm,
                      test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

Com que hem triat un valor relativament petit per a \(\gamma\), la frontera de decisió que en resulta és relativament suau, com es veu al gràfic.

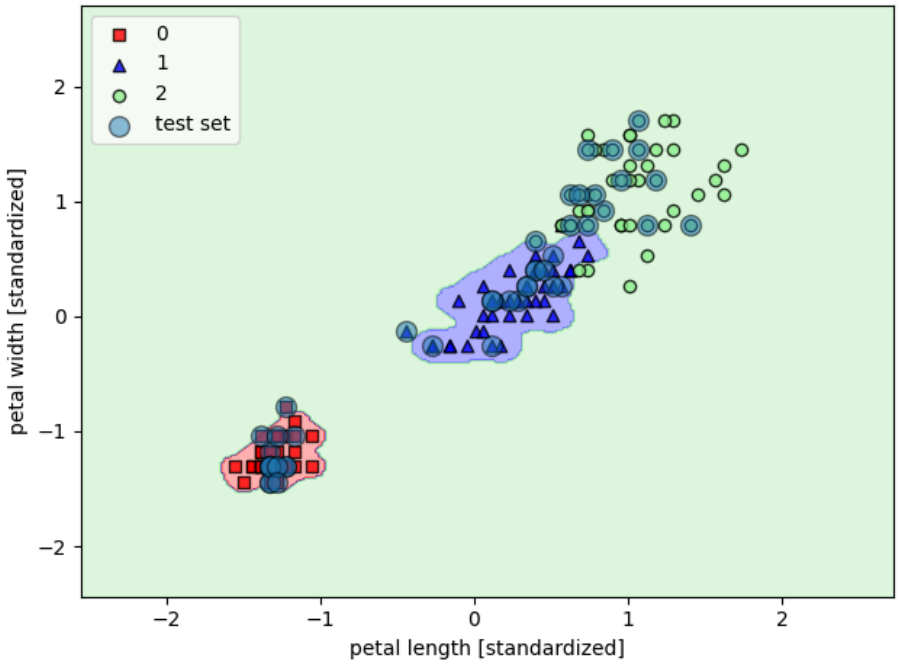


Imatge: *Kernel SVM amb gamma petit*

A continuació, incrementem el valor de \(\gamma\) i vegem l'efecte sobre la frontera de decisió.

```
svm = SVC(kernel='rbf', random_state=1, gamma=100.0, C=1.0)
svm.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std,
                      y_combined, classifier=svm,
                      test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

En aquest cas trobam que la frontera de decisió que envolta les classes 0 i 1 és molt més ajustada quan s'empra un valor relativament alt de \(\gamma\).



Imatge: *Kernel SVM amb gamma gran*

Tot i que el model s'ajusta a les dades d'entrenament molt bé, un classificador d'aquesta forma probablement presentarà un error de generalització elevat sobre dades noves. Això il·lustra que el paràmetre \(\gamma\) juga un paper important a l'hora de controlar el sobreajust o la variància quan l'algorisme és massa sensible a fluctuacions en el conjunt d'entrenament.

12. Arbres de decisió

Els classificadors basat en **arbres de decisió** són atractius si ens interessa la **interpretabilitat** del model. Aquest tipus de classificador segmenta les dades en grups cada vegada més petits realitzant una seqüència de preguntes.

En funció de les característiques (*features*) del nostre conjunt de dades d'entrenament, l'arbre de decisió aprèn una sèrie de preguntes per inferir les etiquetes de classe dels exemples. Per exemple, al conjunt de dades Iris, podríem definir un valor de tall sobre l'eix de la característica "amplada del sèpal" (*sepal width*) i fer la següent pregunta binària: "L'amplada del sèpal, és més gran o igual que 2.8cm?"

Usant l'algorisme de decisió, es comença pel node arrel, amb tot el conjunt de dades d'entrenament agrupat inicialment. Es divideixen les dades amb la característica (*feature*) que aporti un major guany d'informació (*information gain*), que veurem tot seguit. Seguint un procediment iteratiu, es pot continuar el procés de divisió en cada node fill fins que les fulles són pures. A la pràctica, això pot donar com a resultat un arbre molt profund amb molts de nodes, cosa que pot representar un sobreajust (*overfitting*), un ajust massa precís al conjunt particular de dades d'entrenament, i que no pugui generalitzar fàcilment a unes dades distintes. Per tant, sovint volem porgar (en anglès, *prune*) l'arbre fixant un límit per a la profunditat màxima de l'arbre.

12.1. Maximització del guany d'informació

Per dividir els nodes a les característiques més informatives, hem de definir la funció objectiva que volem optimitzar a través de l'algorisme d'aprenentatge de l'arbre. Aquí, la nostra funció objectiva és el **guany d'informació** (IG, *information gain*), que volem maximitzar en cada subdivisió. El definim de la forma següent:

$$ IG(D_p,f) = I(D_p) - \sum_{j=1}^m \frac{N_j}{N_p}I(D_j) $$

En aquesta expressió, $ f $ és la característica on es realitza la partició; $ D_p $ i $ D_j $ són el conjunt de dades del node pare i del node fill j-èsim; I és la mesura d'impuresa; $ N_p $ és el nombre d'exemples d'entrenament al node pare; $ N_j $ és el nombre d'exemples al nòde fill j-èsim. Com podem veure, el guany d'informació és simplement la diferència entre la impuresa del node pare i la suma de les impureses dels nodes fills. Per simplicitat i per reduir l'espai de cerca combinatòria, la majoria de llibreries, com scikit-learn, implementen els arbres de decisió binaris. Això significa que cada node pare es divideix en dos nodes fills, $ D_{left} $ i $ D_{right} $.

$$ IG(D_p,f) = I(D_p) -\frac{N_{left}}{N_p} I (D_{left}) - \frac{N_{right}}{N_p} I(D_{right}) $$

Aquesta fórmula és un cas particular de l'anterior per al cas binari, amb dos nodes fills per node, és a dir, $ m=2 $.

En els arbres de decisió binaris s'utilitzen habitualment tres mesures o criteris de partició: la **impuresa de Gini** (I_g), l'**entropia** (I_H) i l'**error de classificació** (I_E).

Comencem amb la definició d'**entropia**.

$$ I_H = -\sum_{i=1}^c p(i|t)log_2p(i|t) $$

Aquí, $ p(i|t) $ és la proporció d'exemples que pertanyen a la classe $ i $ en un node $ t $. L'entropia és $ 0 $ si tots els exemples del node pertanyen a la mateixa classe, i en canvi l'entropia és màxima si estan distribuïts equitativament, si tenim una distribució de classe uniforme. Es pot dir que el criteri de l'entropia cerca maximitzar la informació mútua a l'arbre.

La **impuresa de Gini** és un criteri per minimitzar la probabilitat d'error de classificació.

$$ I_G(t) = \sum_ {i=1}^c p(i|t)(1-p(i|t)) = 1 - \sum_{i=1}^c p(i|t)^2 $$

De forma semblant a l'entropia, la impuresa de Gini és màxima si les classes estan perfectament mesclades, i mínima si les classes estan perfectament separades.

A la pràctica, la impuresa de Gini i l'entropia solen donar resultats molt semblants, i normalment no val la pena comparar els diferents criteris, sinó que és més pràctic experimentar amb diferents profunditats de tall.

Encara una altra mesura d'impuresa és l'**error de classificació**

$$ I_E(t) = 1 - max \{p(i|t) \} $$

Aquest criteri és útil per retallar l'arbre, més que no per construir-lo fent-lo créixer, ja que és més poc sensible als canvis en les probabilitats de classe dels nodes.

ALERTA

En aquest apartat cal anar alerta a no confondre IG i I_G. Són diferents.

IG és el guany d'informació, *information gain*.

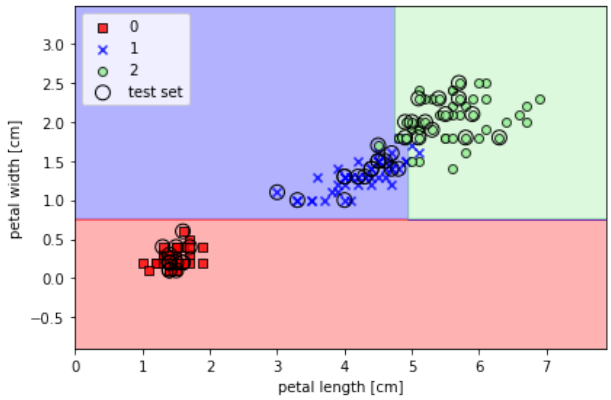
I_G és una de les tres mesures d'impuresa que podem fer servir per avaluar el guany d'informació IG. Les altres dues són l'entropia (I_H) i l'error de classificació (I_E).

12.2. Construcció de l'arbre

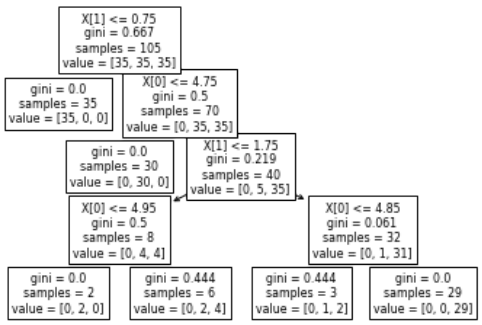
Els arbres de decisió poden aconseguir fronteres de decisió complexes dividint l'espai de característiques en rectangles. Però, hem d'anar alerta perquè com més profund sigui l'arbre, més complexa serà la frontera de decisió, i això augmenta el perill de **sobreajust** (*overfitting*): un molt bon funcionament amb les dades d'entrenament però una capacitat de generalització dolenta. A continuació utilitzarem scikit-learn per entrenar un arbre de decisió amb una profunditat màxima de 4, utilitzant com a criteri la impuresa de Gini. L'escalat de característiques no és necessari en els arbres de decisió. El codi és el següent.

```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier(criterion='gini',max_depth=4,
                                    random_state=1)
tree_model.fit(X_train,y_train)
X_combined = np.vstack((X_train,X_test))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions (X_combined, y_combined, classifier = tree_model,
test_idx=range(105,150))

plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Després d'executar aquest codi, obtenim les típiques fronteres de decisió paral·leles als eixos típiques dels arbres de decisió.



Una característica útil de scikit-learn és que permet veure directament el model de l'arbre de decisió després de l'entrenament.



També podem obtenir una visualització més agradosa amb el programa Graphviz. Podeu consultar el codi per obtenir aquests diagrames al quadern Colab de classificació.

12.3. Random Forest

Els mètodes d'aprenentatge per conjunts (*ensemble learning*) han guanyat popularitat en aplicacions d'aprenentatge automàtic la darrera dècada a causa del seu bon rendiment en classificació i robustesa contra l'*overfitting*. A continuació veurem l'algorisme de **random forest** (bosc aleatori), basat en els arbres de decisió i conegut per la seva facilitat d'ús i la seva bona escalabilitat. Un **random forest** es pot considerar un conjunt d'arbres de decisió. La idea rere el random forest és fer el promig de molts d'arbres de decisió profunds, cadascun d'alta variança, per construir un model que té una millor capacitat de generalització i és més poc susceptible d'*overfitting*. Podem interpretar alta variància com molta dependència del model de les dades particulars usades en l'entrenament. L'algorisme de random forest es pot desglossar en aquestes quatre etapes:

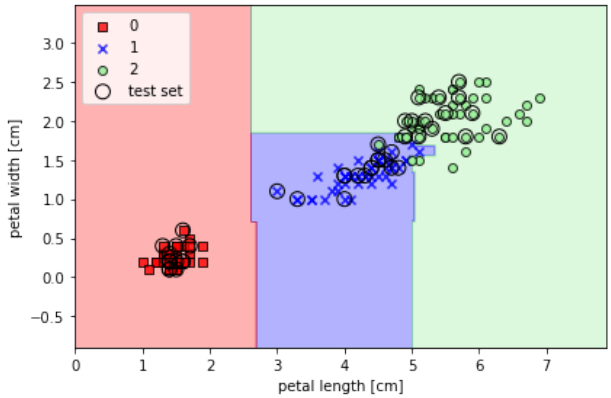
1. Extreure una mostra de bootstrap de mida n: triar a l'atzar n exemples del conjunt d'entrenament, amb reemplaçament.
2. Construir un arbre de decisió amb la mostra bootstrap. En cada node: a) Triar a l'atzar d característiques (variables d'entrada) sense reemplaçament. b) Dividir el node usant la característica que dona la millor divisió d'acord amb la funció objectiva, per exemple el guany d'informació.
3. Repetir els passos anteriors 1 i 2 k vegades.
4. Agregar la predicció de cadascun dels arbres per assignar l'etiqueta de classe mitjançant **votació per majoria** (*majority vote*).

Tot i que els random forests no donen el mateix nivell d'interpretabilitat que els arbres de decisió, un gran avantatge és que no fa falta preocupar-se tant per la tria dels hiperparàmetres. Normalment no cal esporgar (*prune*) el *random forest*, ja que el model de conjunt és bastant robust a l'aleatorietat de dels arbres de decisió individuals. L'únic paràmetre que fa falta tenir en compte a la pràctica és el nombre d'arbres, k, que triam per al *random forest*. Típicament, com més arbres en el conjunt, més gran serà el rendiment del classificador *random forest*, a canvi d'un cost computacional més gran.

Hi ha una implementació del *random forest* a la llibreria **scikit-learn**, de forma que no cal que construim nosaltres el classificador a partir d'arbres individuals. El codi que la utilitzarà és el següent.

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(criterion='gini',
                                n_estimators=25,
                                random_state = 1,
                                n_jobs=2)

forest.fit(X_train,y_train)
plot_decision_regions(X_combined,y_combined,
                      classifier=forest,
                      test_idx=range(105,150))

plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Després d'executar aquest codi, veim les regions de decisió obtingudes pel conjunt d'arbres del random forest, com mostra el gràfic següent.



Al codi, hem entrenament un random forest amb \(25\) arbres de decisió (paràmetre `n_estimators`) i el criteri d'impuresa de Gini. Tot i que és un conjunt petit a partir de poques dades d'entrenament, hem usat el paràmetre `n_jobs` per demostrar com podem paral·lelitzar l'entrenament del model aprofitant múltiples *cores* de la nostra computadora (aquí, dos).

13. k-nearest neighbors

El darrer algorisme d'aprenentatge supervisat que veurem en aquest lliurament és classificador **KNN** (*k-nearest neighbors*, k veïns més propers), que és molt interessant perquè és fonamentalment diferents dels algorismes que hem vist fins ara.

KNN és un exemple típic del que s'anomena **lazy learner**. S'anomena *lazy* (mandrós, peresós) perquè no elabora cap model a partir de les dades, sinó que simplement memoritza tot el conjunt d'entrenament.

L'algorisme KNN és bastant directe i consisteix en els passos següents.

1. Triar el nombre de veïns \(k\) i una mesura de distància.
2. Trobar els k veïns més propers al punt de dades que volem classificar.
3. Assignar l'etiqueta de classe mitjançant vot per majoria.

El principal avantatge d'aquesta solució basada en memòria és que el classificador s'adapta immediatament a mesura que recull noves dades d'entrenament. Però, la contrapartida és que la complexitat computacional de classificar nous exemples creix linealment amb el nombre d'exemples en el conjunt d'entrenament, en el pitjor cas. A més, no es poden descartar exemples d'entrenament, ja que no hi ha realment cap passa d'entrenament. Per tant, l'espai d'emmagatzematge pot ser una dificultat quan es treballa amb aquest model amb grans conjunts de dades.

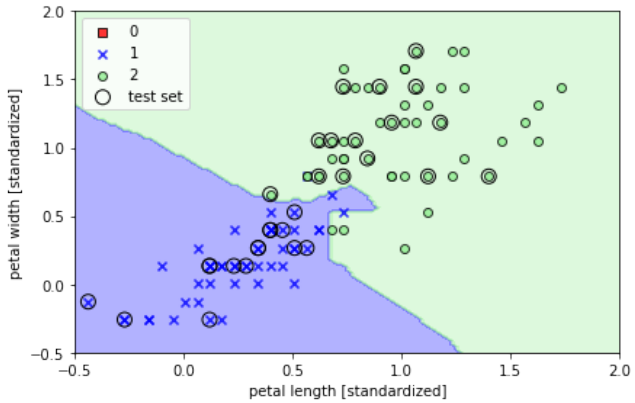
En el codi següent, implementam un model KNN en scikit-learn que utilitza una mesura de distància euclidiana. Aquesta distància és l'arrel dels quadrats de les diferències entre les característiques, tots els quadrats sumats. Vegem l'expressió per calcular la distància euclidiana entre dos punts \(i\) i \(j\), representats pels vectors \(x^{(i)}\) i \(x^{(j)} \). Cada vector està representat per \(K \) característiques, cadascuna definida pel subíndex \(k\), com a \ (x_k\).

\[d(x^{(i)},x^{(j)}) = (\sum_k (x_k^{(i)} - x_k^{(j)})^2) ^\frac{1}{2} \]

A continuació teniu el codi d'exemple.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3, p=2, metric='minkowski')

knn.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std,y_combined,classifier=knn,test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()


ax=plt.gca()
ax.set_xlim([-.5,2])
ax.set_ylim([-.5,2])
plt.show()
```

I al gràfic següent un detall de la frontera de decisió obtinguda. S'hi observa que la corba pot ser bastant complexa, cosa que també comporta el perill de sobreajust (*overfitting*). Com més veïns es tenguin en compte, més poc sobreajust hi haurà, però també més poca resolució.


0
1
2
test set
petal width [standardized]
petal length [standardized]
2.0
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5
2.0

14. Transformació de característiques

En ocasions és útil una transformació de les característiques per facilitar la seva modelització. Per exemple, considerem l'índex de massa corporal (en anglès **body mass index**, BMI). Es defineix d'acord amb l'expressió següent.

$$ BMI = \frac{m}{h^2} $$

on m és la massa en kilograms i h l'alçada en metres.

La frontera entre pes normal i sobrepès s'estableix al valor 25. Vegem com aquesta frontera no és lineal, sinó que presenta una corbatura. (La corba és molt lleu, s'hauria de representar amb una recta damunt per apreciar la diferència).

Per linearitzar-la, es pot efectuar una transformació logarítmica. Ara, després d'aplicar propietats dels logaritmes, la frontera de decisió queda de la forma següent.

$$ \log(25) = \log(m) - 2\log(h) $$

La seva representació és estrictament una recta en l'espai transformat definit pels logaritmes de la massa $\log(m)$ i l'alçada, $\log(h)$.

15. Resum

En aquest lliurament sobre **classificació**, hem après diversos algorismes per resoldre problemes lineals i no lineals.

Hem vist que els **arbres de decisió** són especialment atractius si ens interessa la **interpretabilitat** del model.

La **regressió logística** és un model útil per a l'aprenentatge en línia a través de **SGD** (*Stochastic Gradient Descent*), a més de ser capaç de predir la probabilitat d'un esdeveniment determinat.

Els **SVM** són models lineals potents que es poden estendre a problemes no lineals a través del **truc del kernel**, però tenen molts de paràmetres que s'han d'ajustar per poder realitzar bones prediccions.

En canvi, els **mètodes de conjunt** (*ensemble methods*), com ***random forest***, no necessiten gaire ajust de paràmetres i no sobreajusten tan fàcilment com els arbres de decisió, cosa que els fa models atractius en molts de dominis d'aplicació pràctica.

El classificador **KNN** ofereix una classificació alternativa amb **aprenentatge mandrós** que no necessita l'entrenament de cap model, a canvi d'una etapa de predicció computacionalment costosa.