Regressió

lloc:	Institut d'Ensenyaments a Distància de les Illes Balears	Imprès per:	Carlos Sanchez Recio
Curs:	Sistemes d'aprenentatge automàtic	Data:	dimarts, 26 de novembre 2024, 09:06
Llibre:	Regressió		

Taula de continguts

1. Regressió

El primer tipus d'aprenentatge supervisat que tractarem és la **regressió**. Els models de regressió s'utilitzen per predir variables de destinació (*target*) en una escala contínua. Això els fa útils per resoldre múltiples qüestions en el camp científic, així com aplicacions en la indústria: entendre les relacions entre variables, avaluar tendències o fer pronòstics.

Alguns exemples d'aplicació de regressió poden ser els següents.

- Predicció de les vendes d'una empresa el proper mes
- Predicció de l'evolució d'una població
- Estimació del preu d'un habitatge a partir de les seves característiques (superfície, nombre d'habitacions, planta del pis, exterior/interior, superfície de terrassa, nombre d'ascensors a la finca, distància al centre de la ciutat...)

En aquest tema tractarem els conceptes principals dels models de regressió amb aquests temes:

- Explorar i visualitzar conjunts de dades
- Veure diferents mètodes per implementar models de regressió lineals
- Entrenar models de regressió robustos davant valors extrems o *outliers*
- Avaluar models de regressió i diagnosticar problemes habituals
- Ajustar models de regressió a dades no lineals

L'objectiu de la regressió lineal és modelar la relació entre una o múltiples característiques i una variable de destinació o sortida contínua. L'anàlisi de regressió és una subcategoria de l'aprenentatge automàtic supervisat. Comparat amb la classificació, que també forma part de l'aprenentatge supervisat, la regressió prediu resultats en una escala contínua, en comptes d'etiquetes de classe categòriques.

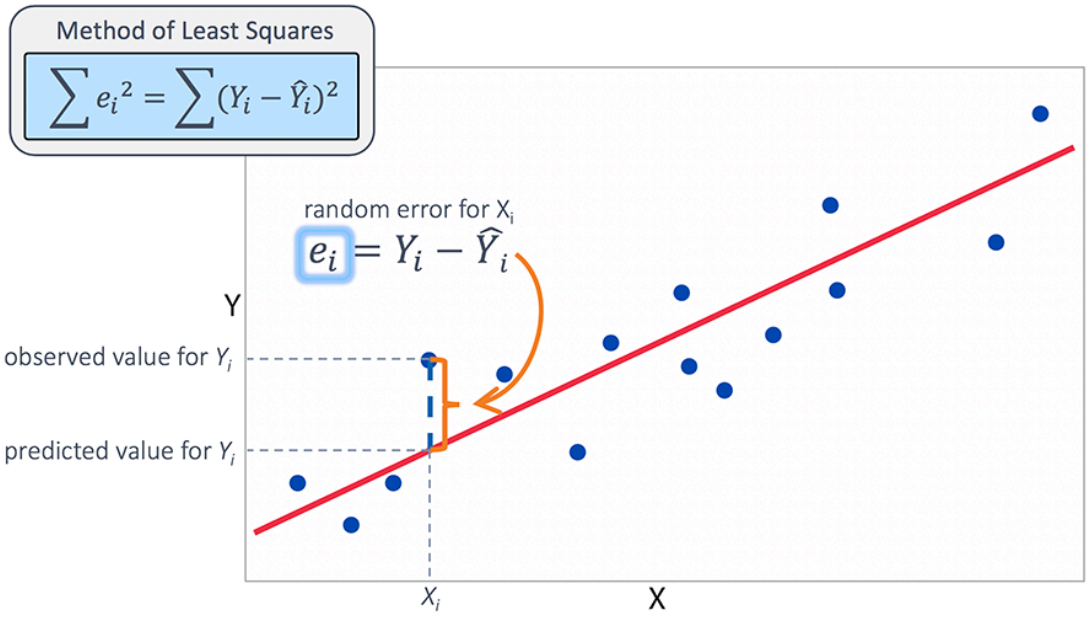
2. Regressió lineal simple

L'objectiu de la regressió lineal simple (univariada) és modelitzar una relació entre una característica simple (**variable explicativa** *x*) i una resposta de valor continu (**variable de destinació** *y*). L'equació d'un model lineal amb una variable explicativa es defineix de la forma següent.

$$y = wx + b$$

En aquest cas, el **biaix** *b* (en anglès, *bias*) representa l'ordenada a l'origen, el valor de *y* quan *x* val zero. El **pes** w (en anglès, *weight*) és el pendent de la recta.

Aquesta recta d'ajust òptim és la **recta de regressió**, i els segments verticals que van dels punts de mostra a la línia representen els residus, els errors de la predicció.



3. Regressió lineal múltiple

El cas especial de la regressió lineal amb una única variable explicativa de la secció anterior es denomina **regressió lineal simple**.

Quan s'utilitzen diverses variables explicatives, x_1, x_2, ... es tracta d'una **regressió lineal múltiple**.

En aquest cas, tenim un model amb l'expressió següent.

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_m x_m + b = \sum_{i=1}^{m} w_i x_i + b = \boldsymbol{w}^T \boldsymbol{x} + b$$

Escrivim els pesos $\boldsymbol{w}$ i les variables d'entrada $\boldsymbol{x}$ en negreta per indicar que es tracta de vectors, en aquest cas vectors columna. El vector columna dels pesos, transposat, es transforma en un vector filera, $\boldsymbol{w}^T$ $\boldsymbol{x}$. El producte escalar d'un vector filera i un altre de columna, de la mateixa longitud, dona com a resultat un escalar. Sumat a l'escalar b, el resultat és un altre escalar y. En aquesta notació algebraica compacta, sempre hem de tenir clar què són escalars, vectors o matrius, i quines dimensions tenen.

Si en el cas de regressió lineal simple parlam de recta de regressió, quan hi ha dues variables explicatives i una variable de sortida les podem representar en un espai tridimensional, amb un pla de regressió. Amb més de de dues variables explicatives, es parla d'hiperplans de regressió, però la seva visualització és més difícil.

4. Descens de gradient

Com s'obté aquesta recta ajustada a les dades? Se sol usar el mètode de **mínims quadrats**, que estima els paràmetres de la recta de regressió (els pesos $\boldsymbol{w}$ i el biaix b) que minimitzen la suma de les distàncies verticals al quadrat (residus o errors) en els punts de mostra.

Per arribar a definir la funció de cost necessitarem algunes definicions prèvies.

y és la variable de destinació o de sortida. La nostra mostra està formada per una sèrie de valors d'aquesta variable, diguem-ne *n* valors. Cada un d'ells, el valor i-èsim, en general, el denotarem amb $y^{(i)}$. El model intenta obtenir un valor predit tan a prop com sigui possible del valor real. El valor predit per a l'i-èsim punt de dades, l'anomenam $y^{\hat{(i)}}$. L'error quadràtic de predicció, per tant, correspon a la diferència següent.

$$(y^{(i)} - y^{\hat{(i)}})^2$$

La funció de cost, que depèn dels pesos i el biaix, $J(w, b)$, ha de tenir en compte la contribució de tots els errors. El denominador 2 és un simple factor d'escala que s'utilitza per conveniència. Aquesta funció, després, s'haurà de derivar, i la contribució de l'exponent 2 i aquest denominador es cancel·laran. L'expressió que es fa servir per a la funció de cost és la següent.

$$J(\boldsymbol{w}, b) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - y^{\hat{(i)}}))^2$$

Ara, per poder optimitzar de forma iterativa les estimacions que tinguem del pes *w* i el biaix *b*, hem d'obtenir una expressió de la variació de la funció de cost respecte d'aquests paràmetres. Com que es tracta d'una funció de dues variables, si volem fer-ho rigorosament hem d'expressar aquesta variació com a derivada parcial. Van apareixent idees de càlcul i d'àlgebra amb les quals podem estar més o menys familiaritzats. No serà necessari que les acabem implementant nosaltres mateixos en codi, ja que estan disponibles a les llibreries, per exemple scikit-learn. Però és bo que tenguem una comprensió tan profunda dels algorismes com sigui possible, per entendre què està passant per sota i poder interpretar les dificultats que trobem a l'hora d'executar-los.

Tenint en compte l'expressió de la predicció de la variable de sortida,

$$y^{\hat{(i)}} = \boldsymbol{w}^T x^{(i)} + b$$

podem obtenir que

$$\frac{\partial J(w, b)}{\partial \boldsymbol{w}} = -\sum_{i=1}^{n}(y^{(i)} - y^{\hat{(i)}})x^{(i)}$$

i

$$\frac{\partial J(w, b)}{\partial b} = -\sum_{i=1}^{n}(y^{(i)} - y^{\hat{(i)}})$$

Les dues expressions són molt semblants, tot i que no idèntiques. Observem que l'única diferència ve del factor x addicional a causa de la derivada respecte del pendent, que ve del producte wx de la funció de predicció.

Una vegada coneixem com varia la funció de cost respecte dels paràmetres del model, podem aplicar iterativament les expressions següents per obtenir valors cada vegada més propers als òptims. En general,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\frac{\partial J(\boldsymbol{w}, b)}{\partial \boldsymbol{w}}$$

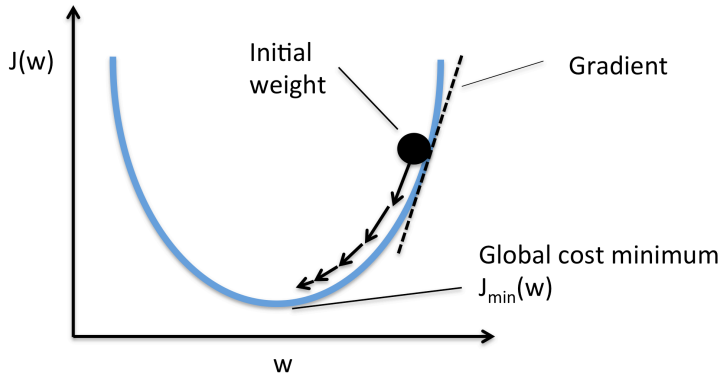
$$b \leftarrow b - \alpha \frac{\partial J(\boldsymbol{w}, b)}{\partial b}$$

Substituint-hi les expressions que hem obtingut més amunt,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \sum_{i=1}^{n} (y^{(i)} - y^{\hat{(i)}}) x^{(i)}$$

$$b \leftarrow b + \alpha \sum_{i=1}^{n} (y^{(i)} - y^{\hat{(i)}})$$

Al mateix temps que l'àlgebra, és important tenir una idea intuïtiva de com funciona el descens de gradient. Primer, notem que la funció de cost és convexa i té un sol mínim. Té una forma de copa, o de mitja síndria buida. Per a un valor qualsevol de $\boldsymbol{w}$ i b, obtenim un valor de $J(\boldsymbol{w}, b)$, que en general no serà el mínim. Si ens movem en la direcció oposada a la que marca el gradient, davallarem per la superfície. Quan ho hàgim fet prou vegades, en les direccions tant de $\boldsymbol{w}$ com de b, serem tan a prop del mínim com vulguem.



És important la taxa d'aprenentatge α (*learning rate*). Una taxa d'aprenentatge massa grossa farà que l'algorisme divergeixi. Una taxa d'aprenentatge massa petita farà que l'algorisme convergeixi molt lentament. Habitualment es comença a executar amb un valor al voltant de 0.01 i a partir d'aquí s'ajusta cap a dalt o cap a baix en funció de l'evolució de la reducció del cost. Considerarem que l'algorisme ha convergit quan el canvi en el cost és inferior a un llindar predeterminat, o s'ha arribat a un nombre màxim d'iteracions.

5. Avaluació: error quadràtic i coeficient de correlació

Després d'haver obtingut els paràmetres del model (w i b), l'hem de provar amb dades que no s'hagin utilitzat durant la fase d'entrenament. D'aquesta forma, tindrem una idea del comportament del model davant informació no coneguda.

En un cas com la regressió lineal, en què no hi hagi hiperparàmetres (mida del model, per exemple), es poden dividir les dades en dues fraccions, conjunt d'entrenament i conjunt de test. Una proporció típica orientativa és dedicar un 70% per a l'entrenament i un 30% de test. Quan hi ha hiperparàmetres, una divisió prudent és el 60% per a entrenament, un 20% per a desenvolupament (optimització dels hiperparàmetres) i un 20% per a test.

Gràfics de residus

En casos de regressió múltiple, no és fàcil representar l'hiperplà de regressió. Per avaluar visualment l'ajust del model, són útils els **gràfics de residus**, amb els residus en funció dels valors predits.

Error quadràtic mitjà

L'error quadràtic mitjà (***MSE***, de l'anglès *Mean Square Error*) és una mesura quantitativa del rendiment del model. És el valor mitjà del cost que s'ha minimitzar per ajustar el model de regressió lineal. L'MSE és útil per comparar diferents models de regressió. L'expressió matemàtica de l'MSE és la següent.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

En Python, el codi pot ser el següent.

```
from sklearn.metrics import mean_squared_error
print('MSE train: %.2f' % mean_squared_error(y_train,y_train_pred))
print('MSE train: %.2f' % mean_squared_error(y_test,y_test_pred))
```

Si l'error en el conjunt de test és molt més gran que en el conjunt d'entrenament, això indica que el nostre model té sobreajust (*overfitting*): depèn massa dels detalls irrellevants del conjunt d'entrenament i mostra poca capacitat de generalització al conjunt de test.

Coeficient de determinació

De vegades pot ser més útil obtenir el coeficient de determinació (R^2), que es pot entendre com una versió estandarditzada de l'MSE, que permet interpretar més bé el rendiment del model. R^2 és la fracció de la variança de la resposta capturada pel model, i es defineix amb l'expressió següent.

$$R^2 = 1 - \frac{SSE}{SST}$$

SSE és la suma dels errors quadràtics:

$$SSE = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

i SST la suma total dels quadrats de la diferència dels valors de sortida i la seva mitjana μ_y.

$$SSE = \sum_{i=1}^{n}(y^{(i)} - \mu_y)^2$$

Es pot demostrar que \(*R^{2}*\) és una versió reescalada de l'MSE

$$R^2 = 1 - \frac{MSE}{Var(y)}$$

Arribam a aquesta conclusió perquè la variància d'\(*y*\) està directament relacionada amb l'SST.

$$Var(y) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \mu_y)^2$$

En Python, calculam el coeficient de determinació de la forma següent.

```
from sklearn.metrics import r2_score
print('R^2 train: %.2f' %r2_score(y_train,y_train_pred))
print('R^2 test: %.2f' %r2_score(y_test,y_test))
```

6. RANSAC

La presència d'*outliers* pot tenir un impacte negatiu en la regressió lineal. De vegades, un subconjunt molt petit de les dades pot tenir un gran efecte sobre els valors estimats dels coeficients. Hi ha molts de tests estadístics que poden fer-se servir per a detectar *outliers*, però són fora de l'abast d'aquest curs. Tanmateix, l'eliminació d'outliers sempre necessita el nostre criteri com a tècnics de dades així com el coneixement del domini d'aplicació.

Com a alternativa a l'eliminació d'outliers, veurem un mètode robust de regressió usant l'algorisme **RANSAC** (***RAndom SAmple Consensus***). Aquest algorisme ajusta un model de regressió a un subconjunt de les dades, els anomenats *inliers*.

Es pot resumir l'algorisme iteratiu RANSAC de la forma següent.

1. Seleccionar un nombre aleatori de dades d'exemple com a inliers i ajustar-hi el model.
2. Testejar tots els altres punts respecte del model ajustat i, aquells que hi estiguin prou a prop, afegir-los al conjunt d'inliers.
3. Reentrenar el model usant tots els *inliers*.
4. Estimar l'error del model ajustat respecte dels *inliers*.
5. Acabar l'algorisme si el rendiment arriba a un cert llindar definit per l'usuari o s'hi s'ha arribat a un determinat nombre d'iteracions. Si no, tornar a la primera passa.

A continuació farem servir un model lineal en combinació amb l'algorisme RANSAC tal com ve implementat en la classe RANSACRegressor de scikit-learn.

```
from sklearn.linear_model import RANSACRegressor
ransac = RANSACRegressor(LinearRegression(),
                         max_trials=100,
                         min_samples=50,
                         loss='absolute_error',
                         residual_threshold=5.0,
                         random_state=0)
ransac.fit(X,y)
```

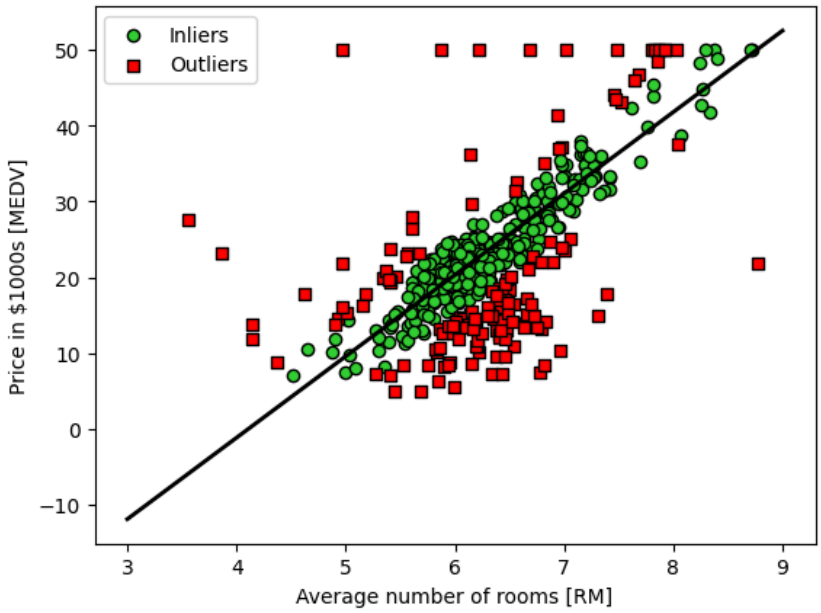
Fixam el nombre màxim d'iteracions del Regressor RANSAC a 100, i usant min_samples=50, fixam el nombre mínim d'exemples d'entrenament a 50. Usant 'absolute_loss' com a argument del paràmetre loss, l'algorisme computa les distàncies verticals absolutes entre la línia ajustada i els exemples d'entrenament. Fixant el paràmetre residual_threshold a 5.0, només permetem que s'afegeixin al conjunt d'*inliers* els exemples d'entrenament que quedin no més enfora de 5 unitats, un valor adequat per a aquest conjunt de dades concret.

Per defecte, scikit-learn fa servir l'estimació MAD per seleccionar el llindar dels *inliers*. Les sigles MAD corresponen a Median Absolute Deviation, la desviació absoluta mediana (no mitjana) dels valors de sortida y. La tria d'un valor adequat com a llindar d'*inliers* depèn de cada problema, i això és un inconvenient del mètode RANSAC. S'han desenvolupat moltes solucions diferents per triar automàticament un bon llindar.

Havent ajustat el model RANSAC, podem obtenir els inliers i outliers del model de regressió lineal RANSAC i mostrar-los alhora que l'ajust lineal.

```
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
line_X = np.arange(3, 10, 1)
line_y_ransac = ransac.predict(line_X[:, np.newaxis])
plt.scatter(X[inlier_mask], y[inlier_mask],
            c='limegreen', edgecolor='black',
            marker='o', label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask],
            c='red', edgecolor='black',
            marker='s', label='Outliers')
plt.plot(line_X, line_y_ransac, color='black', linewidth=2)
plt.xlabel('Average number of rooms [RM]')
plt.ylabel('Price in $1000s [MEDV]')
plt.legend(loc='upper left')
plt.show()
```

Al gràfic següent podem veure com el model de regressió lineal s'ha ajustat al conjunt d'inliers, representats com a cercles.



Imatge: *Regressió lineal robusta amb RANSAC*

Quan imprimim els valors del pendent i l'ordenada a l'origen del model, amb el codi següent, trobam que la recta és lleugerament diferent que la que hem trobat a l'apartat anterior sense fer servir RANSAC.

```
print('Slope: %.3f' % ransac.estimator_.coef_[0])
print('Intercept: %.3f' % ransac.estimator_.intercept_)
```

```
Slope: 10.735
Intercept: -44.089
```

Usant RANSAC, hem reduït l'efecte potencial dels *outliers* en aquest conjunt de dades, però no sabem si aquesta tècnica tendrà un efecte positiu en la capacitat predictiva del model davant dades noves. A la secció següent tractam diferents formes d'avaluar el model de regressió, una part fonamental de la construcció de sistemes de modelització predictiva.

7. Regularització

La **regularització** és un mètode per fer front al problema del **sobreajust** (*overfitting*). Consisteix a reduir els valors dels paràmetres del model per penalitzar-ne la complexitat.

Els dos tipus principals són L2 (Ridge) i L1 (LASSO), segons la norma del vector de pesos que minimitzen. Vegem-ho amb detall.

$$J(w,b)_{Ridge} = \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2 + \lambda ||w||_2^2$$

on

$$\lambda ||w||_2^2 = \sum_{j=1}^{m} w_j^2$$

Augmentant el valor de l'hiperparàmetre λ, augmentam la força de regularització i reduïm els pesos del model. Hem de tenir en compte que no regularitzam el terme de l'ordenada a l'origen *b*. Per això, en un cas extrem de λ molt gran, el model es reduirà a una constant, precisament la *b* del model lineal $y = wx + b$. A l'altre extrem, $\lambda=0$ correspon a no introduir cap regularització.

Un altre mètode, que pot produir models dispersos, és **LASSO** (de l'anglès *Least Absolute Shrinkage and Selection Operator*). Segons la força de la regularització, determinats pesos poden adoptar el valor zero, de forma que LASSO és també una tècnica útil de **selecció de característiques** (*feature selection*) supervisada.

$$J(w,b)_{LASSO} = \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2 + \lambda ||w||_1$$

Aquí, en lloc de la suma dels quadrats dels pesos, es minimitza la suma dels valors absoluts.

$$\lambda ||w||_1 = \sum_{j=1}^{m} |w_j|$$

Una limitació de LASSO és que selecciona com a màxim *n* variables si $m>n$.

Implementació amb scikit-learn

El model de regressió Ridge es pot inicialitzar de la manera següent

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)
```

El paràmetre alpha fa la funció de lambda, λ. De la mateixa manera, podem inicialitzar un regressor LASSO a partir del submòdul linear_model.

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=1.0)
```

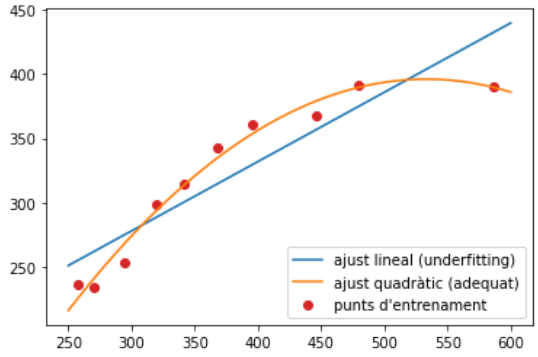
Hi ha informació més detallada sobre models lineals a https://scikit-learn.org/stable/modules/linear_model.html

8. Regressió polinòmica

Tant a la regressió lineal simple com múltiple, se suposa una relació lineal entre les variables explicatives i la variable de sortida. Si el model lineal és insuficient per modelitzar la relació entre les variables x i y, es pot usar, per exemple, una regressió polinòmica. En el cas d'una única variable, l'expressió és la següent.

$$y = w_1 x + w_2 x^2 + w_3 x^3 + \ldots w_d x^d + b$$

Aquí, d és el grau del polinomi. Tot i que sortim de la linealitat pel fet d'usar exponents més grans que la unitat, se segueix considerant un model lineal pel fet que pondera les potències d'x.



Al quadern de Colab següent tenim codi d'exemple d'implementació de la regressió lineal i polinòmica.

9. Regressió amb arbres de decisió

Un avantatge de realitzar la regressió amb arbre de decisió és que no fa falta cap transformació de les dades si tenim dades no lineals, perquè l'arbre de decisió avalua les característiques d'una en una, en lloc de tenir en compte combinacions lineals de les variables d'entrada. D'una forma semblant, tampoc no fa falta normalitzar o estandarditzar les característiques. Feim créixer un arbre de decisió xapant iterativament els seus nodes fins que les fulles són pures o bé s'ha arribat a un criteri d'aturada. Quan vàrem usar els arbres de decisió per a classificació, vàrem definir l'**entropia** com a mesura d'impuresa per determinar quina característica maximitza el **guany d'informació** de la divisió, que es defineix de la forma següent.

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

En aquesta fórmula, x_i és la característica respecte de la qual es divideix el node, N_p és el nombre d'exemples d'entrenament en el node pare, D_p és el conjunt de dades d'entrenament del node pare, D_{left} i D_{right} són els subconjunts dels nodes esquerre i dret després de la divisió. Recordem que el nostre objectiu és trobar la divisió que maximitza el guany d'informació, és a dir, que redueix al màxim les impureses en els nodes fill. Al capítol de classificació, vàrem veure la impuresa de Gini i l'entropia com a mesures d'impuresa, que són criteris útils en classificació.

Per usar l'arbre com a regressor, en canvi, necessitam una mesura d'impuresa adequada per a variables contínues, de forma que definim la mesura d'impuresa d'un node t com a MSE, Mean Square Error.

$$I(t) = MSE(t) = \frac{1}{N_t} \sum_{iinD_t} \left(y^{(i)} - \hat{y}_t\right)^2$$

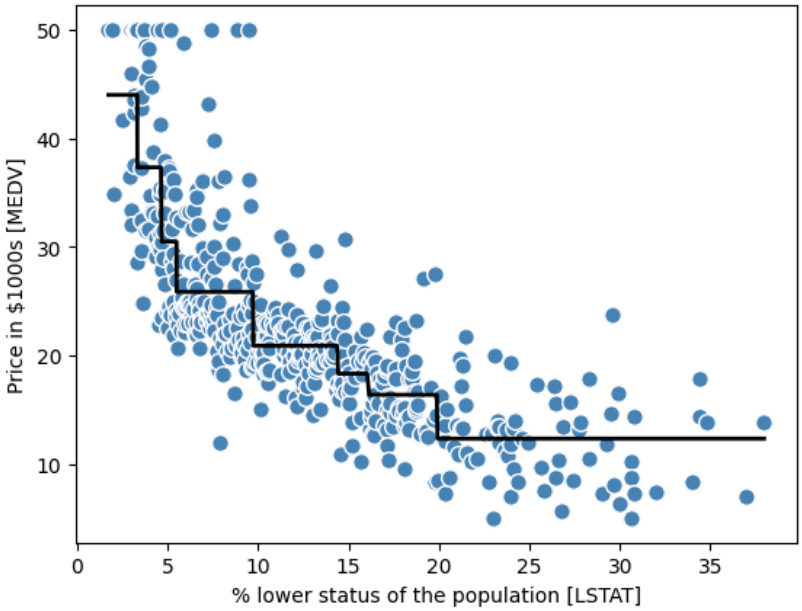
Aquí N_t és el nombre d'exemples d'entrenament al node t, D_t és el conjunt d'entrenament al node t, $y^{(i)}$ és el valor cert de la sortida, i $\hat{y}_t$ és el valor predit de la sortida, la mitjana de les mostres:

$$\hat{y}_t = \frac{1}{N_t} \sum_{iinD_t} y^{(i)}$$

En regressió amb arbres, del MSE se'n sol dir **variància intranode**. Per això, aquest criteri de divisió també es coneix com a **reducció de variància**. Per veure quin aspecte té una funció de regressió d'un arbre de decisió, usarem el DecisionTreeRegressor implementat a scikit-learn per modelitzar la relació no lineal que hi ha entre les variables de la base de dades Housing entre les variables MEDV i LSTAT.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

X = df[['LSTAT']].values
y = df['MEDV'].values
tree = DecisionTreeRegressor(max_depth=3)

tree.fit(X, y)
sort_idx = X.flatten().argsort()
lin_regplot(X[sort_idx], y[sort_idx], tree)
plt.xlabel('% lower status of the population [LSTAT]')
plt.ylabel('Price in $1000s [MEDV]')
plt.show()
```

Com es pot veure al gràfic inferior, l'arbre de decisió captura la tendència general de les dades. No obstant això, una limitació d'aquest model és que la predicció no és contínua ni, per tant, derivable. A més, hem d'anar amb compte amb la tria d'un valor de la profunditat de l'arbre per no caure en infraajust ni en sobreajust. En aquest cas, una profunditat de 3 ha semblat una bona opció.



Imatge: *Regressió amb arbre de decisió*

A la secció següent, tractam una tècnica més robusta per ajustar arbres de regressió: els *random forests*.

10. Regressió amb random forests

L'algorisme de random forest és una tècnica de conjunt (*ensemble*) que combina múltiples arbres de decisió. Un random forest sol tenir una capacitat de generalització més gran que un arbre individual a causa de l'aleatorietat, que ajuda a reduir la variància del model. Uns altres avantatges dels random forests són la seva baixa sensibilitat als *outliers* i que no cal gaire ajust dels paràmetres. L'únic paràmetre amb que sol fer falta experimentar en els random forests és el nombre d'arbres del conjunt. L'algorisme bàsic dels random forests per a regressió és gairebé idèntic al de classificació. L'única diferència és que aquí es fa servir el criteri del MSE per créixer els arbres de decisió individuals, i que el valor de sortida predit es calcula com a promig de les prediccions de tots els arbres individuals.

A continuació, usarem totes les característiques del conjunt de dades Housing per ajustar una regressió random forest sobre un 60 dels exemples i l'avaluarem sobre l'altre 40. Aquest és el codi.

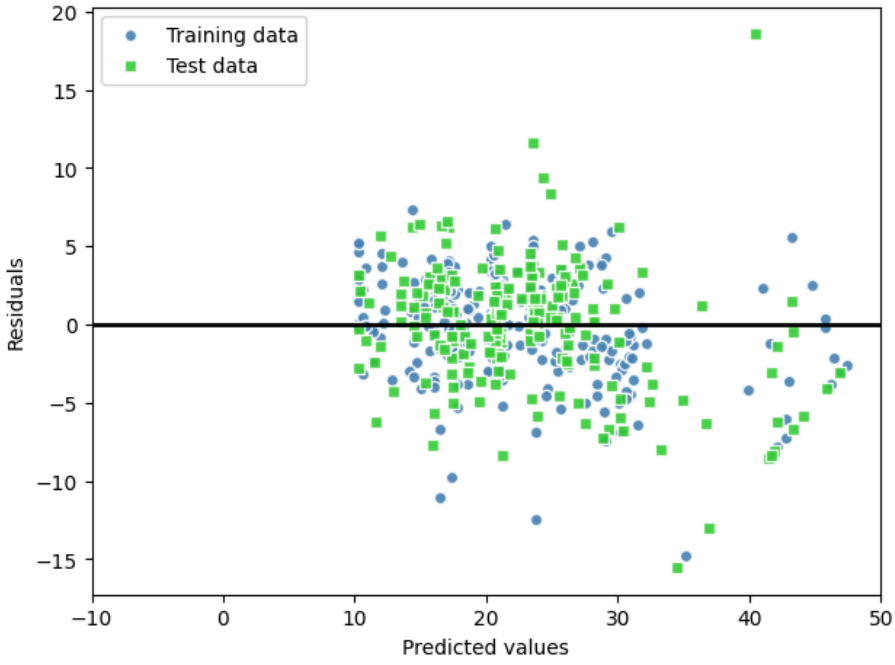
```
X = df.iloc[:, :-1].values
y = df['MEDV'].values
X_train, X_test, y_train, y_test = \
  train_test_split(X, y,
                   test_size=0.4,
                   random_state=1)

forest = RandomForestRegressor(n_estimators = 8000,
                               criterion='squared_error',
                               max_depth=3,
                               random_state=1,
                               n_jobs=-1)

forest.fit(X_train, y_train)
y_train_pred = forest.predict(X_train)
y_test_pred = forest.predict(X_test)
print('R^2 train: %.3f, test: %.3f' % (
    r2_score(y_train, y_train_pred),
    r2_score(y_test, y_test_pred)))
```

```
R^2 train: 0.879, test: 0.822
```

Malauradament, veim que el random forest sobreajusta a les dades d'entrenament. Així i tot, és capaç d'explicar la relació entre les variables explicatives i objectiu relativament bé, amb $R^2 = 0.822$ en el conjunt de prova.

Finalment, vegem els residus de la predicció.

```
plt.scatter(y_train_pred,
            y_train_pred - y_train,
            c='steelblue',
            edgecolor='white',
            marker='o',
            s=35,
            alpha=0.9,
            label='Training data')
plt.scatter(y_test_pred,
            y_test_pred-y_test,
            c='limegreen',
            edgecolor='white',
            marker='s',
            s=35,
            alpha=0.9,
            label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, lw=2, color='black')
plt.xlim([-10, 50])
plt.tight_layout()
plt.show()
```



Imatge: *Residus en funció del valor predit amb regressió basada en random forest*

Com hem vist amb el coeficient R^2, el model s'ajusta més bé a les dades d'entrenament que no a les de prova, com indiquen els *outliers* en la direcció de l'eix y. La distribució dels residus tampoc no sembla completament aleatòria al voltant del punt zero central, cosa que indica que el model no és capaç d'explotar tota la informació explicativa. De tota manera, el gràfic dels residus indica una gran millora sobre el gràfic del model lineal que havíem vist abans.

Idealment, l'error del model hauria de ser aleatori o impredictible. En altres paraules, l'error a les prediccions no s'hauria de poder relacionar amb cap informació continguda en les variables explicatives, sinó que hauria de reflectir l'aleatorietat de les distribucions o patrons del món real. Una raó habitual podria ser que la informació explicativa s'estigués filtrant en els residus.

Malauradament, no hi ha una solució universal per tractar amb la no-aleatorietat en els gràfics de residus, i això requereix experimentació. Segons les dades que hi hagi disponibles, podem millorar el model transformant les variables, ajustant els hiperparàmetres de l'algorisme d'aprenentatge, triant models més simples o més complexos, eliminant outliers o incloent variables addicionals.

11. Regressió no paramètrica

En aquest apartat de regressió no paramètrica seguirem el llibre de Russell i Norvig *Artificial Intelligence: A Modern Approach*. A continuació veurem tres sistemes de regressió no paramètrica.

- Regressió no paramètrica lineal a trossos
- Regressió *k-nearest-neighbors*
- Regressió ponderada localment

11.1. Regressió lineal a trossos

El mètode més simple es coneix col·loquialment com unir els punts o, més formalment, com regressió no paramètrica lineal a trossos. Aquest model crea una funció d'interpolació $h(x)$ que, donat un punt demanat x_q, pren els exemples d'entrenament a l'esquerra i a la drata de x_q i hi fa una interpolació. Quan el soroll és baix, aquest mètode trivial no funciona malament. Per això és una solució estàndard de visualització en els programes de full de càlcul. Ara bé, quan la funció conté soroll, la funció que resulta és punxeguda i no generalitza bé.

11.2. Regressió de k veïns propers

La regressió k-nearest-neighbours és una millora respecte de la lineal a trossos. En lloc d'usar només els dos veïns a l'esquerra i a la dreta del punt demanat x_q, se n'usen els k més propers. A les gràfiques d'exemple, usam $k = 3$. Un valor més gran de k suavitza la magnitud dels pics, tot i que la funció resultat té discontinuïtats. Oferim dues versions de la regressió k-nearest-neighbors.

A la primera, tenim el promig dels veïns més propers. Notem que als punts extrems, a prop de $x = 0$ i $x = 14$, les estimacions són pobres perquè tota l'evidència ve d'un costat (l'interior), i s'ignora la tendència.

A la segona, tenim la regressió lineal k-nearest-neighbors, que troba la millor línia per als k exemples. Aquest sistema captura més bé la tendència als extrems, tot i que la funció segueix essent discontínua.

11.3. Regressió ponderada localment

La regressió ponderada localment té els avantatges dels veïns propers, però sense les discontinuïtats. Per evitar les discontinuïtats a la funció $h(x)$, hem d'evitar les discontinuïtats en el conjunt d'exemples que usam per estimar $h(x)$. La idea de la regressió ponderada localment és donar més pes als exemples més propers i més poc als exemples més llunyans, fins arribar a zero als més llunyans de tots. La reducció del pes amb la distància és gradual, no sobtada.

Es decideix la ponderació de cada exemple amb una funció anomenada kernel. La seva entrada és la distància entre el punt demanat i l'exemple. Una funció de kernel K és una funció decreixent amb la distància amb un màxim a 0, de forma que $K(Dist(x_j, x_q))$ dona més pes als exemples x_j propers al punt demanat x_q, per al qual es fa la predicció del valor de la funció.

L'exemple de la gràfica s'ha generat amb un kernel quadràtic

$$K(d) = max(0, 1 - (2\,|\,d\,|\,/w)^2,$$

amb amplada de kernel $w = 10$. També se'n poden fer servir d'altres formes, com per exemple gaussians. Normalment l'amplada és més important que no la forma. Això és un hiperparàmetre del model. Si els kernels són massa amples obtenim infraajust i si són massa estrets, sobreajust. A l'exemple, l'amplada del kernel 10 dona una corba suau amb un ajust adequat.

A partir d'això, realitzar la regressió ponderada amb kernel és directe. Per a cada punt demanat x_q es resol el següent problema de regressió ponderada.

$$w^* = argmin_w \sum_j K(Dist(x_q, x_j))(y_j, w \cdot x_j)^2$$

on Dist és una mètrica de distància. Aleshores la resposta és

$$h(x_q) = w^* \cdot x_q$$

Observem que cal resoldre un nou problema de regressió per a cada punt demanat (això és el que significa que sigui local). La part positiva és que per a cada punt demanat el nombre de punts de dades que intervenen és més petit, només aquells que tenen un pes més gran que zero. Amb kernels estrets, això poden ser només alguns punts.

12. Resum

En aquest lliurament hem vist com la regressió lineal simple modelitza la relació entre una única variable explicativa i una variable de resposta contínua.

Després hem vist una tècnica útil d'anàlisi de dades explicativa per trobar patrons i anomalies a les dades, una primera passa important en tascas de modelització predictiva.

Hem construït un primer model de regressió lineal utilitzant una optimització basada en el gradient. Després hem vist com emprar els models lineals de scikit-learn per a regressió.

Hem implementat una tècnica de regressió robusta (RANSAC) per fer cara als valors anòmals (*outliers*).

Per avaluar la capacitat predictiva dels models de regressió, hem calculat el promig de l'error quadràtic i el coeficient de regressió. També hem vist una tècnica gràfica útil per valorar els models de regressió: el gràfic de residus.

Després d'utilitzar la regularització per reduir la complexitat del model i minimitzar el sobreajust, hem vist com modelitzar relacions no lineals, com ara la regressió polinòmica i els regressors amb *random forest*.

Als lliuraments anteriors hem estudiat l'aprenentatge supervisat, al lliurament 2 la classificació i al lliurament 3 la regressió. Al proper lliurament veurem diverses tècniques d'aprenentatge no supervisat, clústering i reducció de la dimensionalitat.