Alberto Barbosa

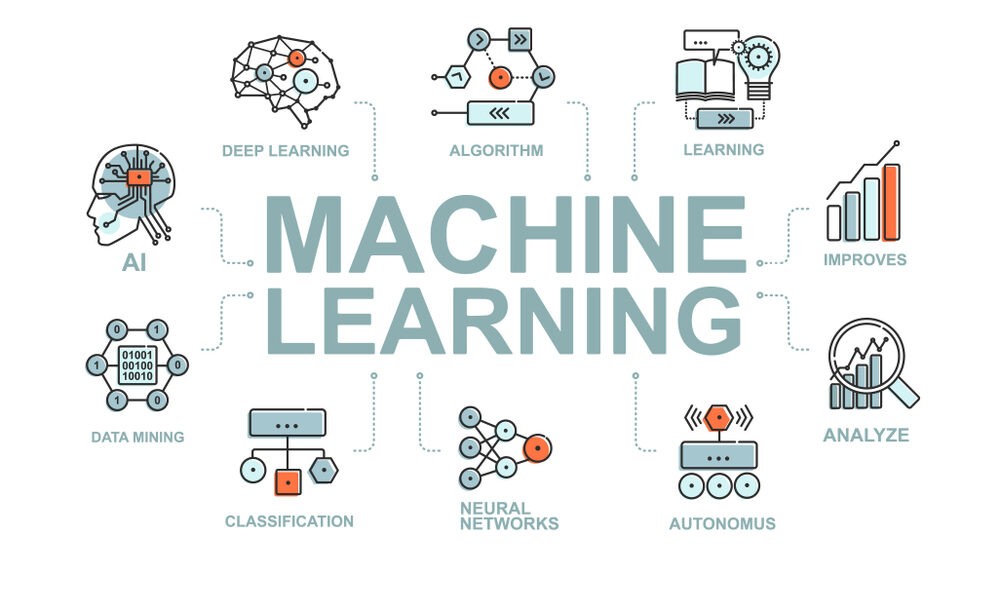
Universidade da Maia *alberto.barbosa@umaia.pt*

Sistemas Inteligentes - Machine Learning

May 26, 2025

# Machine Learning

Podemos dizer que um agente inteligente est´a a aprender quando melhora o seu comportamento ap´os fazer observac¸˜oes sobre o mundo. Quando este agente ´e um computador, falamos de machine learning, onde um computador analisa dados, constr´oi um modelo baseado nesses dados e usa o modelo como uma hip´otese sobre o mundo.



Machine Learning: Porquˆe?

Porquˆe deixar a m´aquina aprender quando podemos simplesmente fazer um program´a-la da forma correcta para resolver o problema? Primeiro, podemos estar a falar de um problema com demasiada variabilidade, o que torna imposs´ıvel a capacidade de antecipar todos os casos.

Segundo, podemos n˜ao saber como resolver o problema, pelo que tentemos que a m´aquina aprenda a existˆencia de padr˜oes que n´os n˜ao conseguimos descortinar ainda.

# Tarefas em Machine Learning

Genericamente podemos falar de trˆes tipos de tarefas de aprendizagem:

Aprendizagem supervisionada.



Aprendizagem n˜ao supervisionada.

Reinforcement Learning.

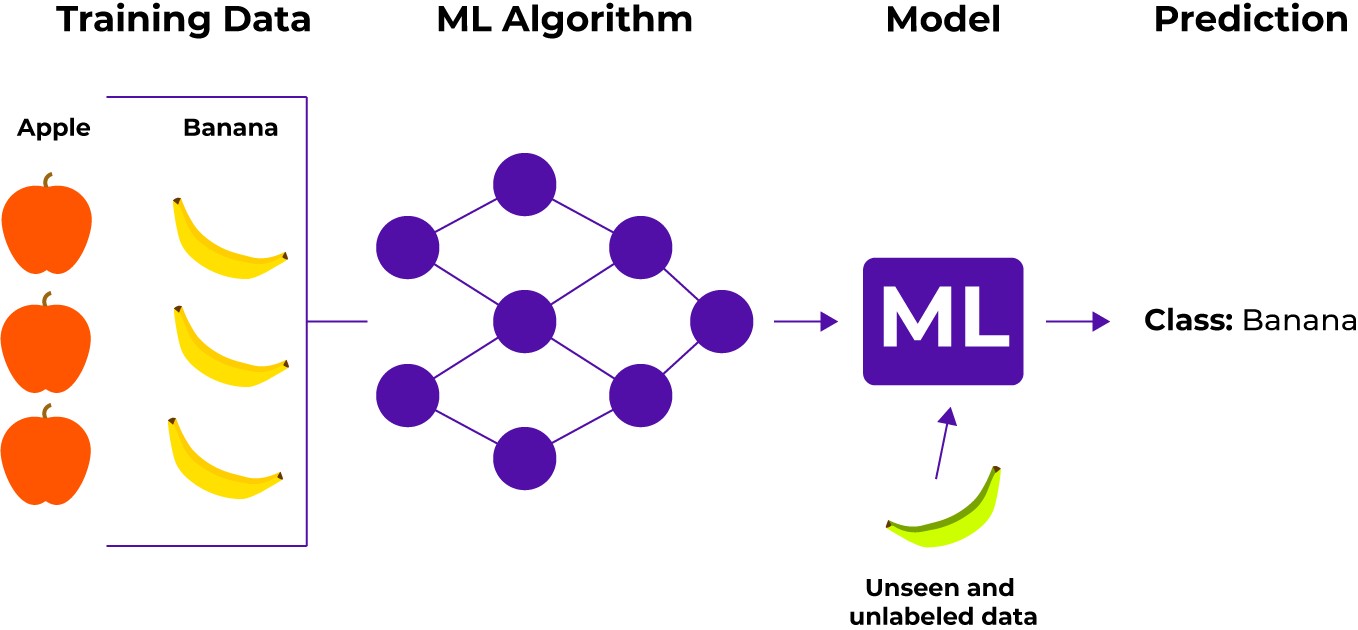
# Aprendizagem Supervisionada

Na aprendizagem supervisionada, o agente observa pares de input-output e aprende uma func¸˜ao que mapeia o input no output. Aos outputs chamamos labels.

O agente ir´a aprender uma func¸˜ao que quando recebe um novo elemento de input, prevˆe uma nova label para esse input.

Um exemplo deste tipo de aprendizagem ´e receber uma imagem e dizer se a imagem retrata um gato.

# Aprendizagem Supervisionada

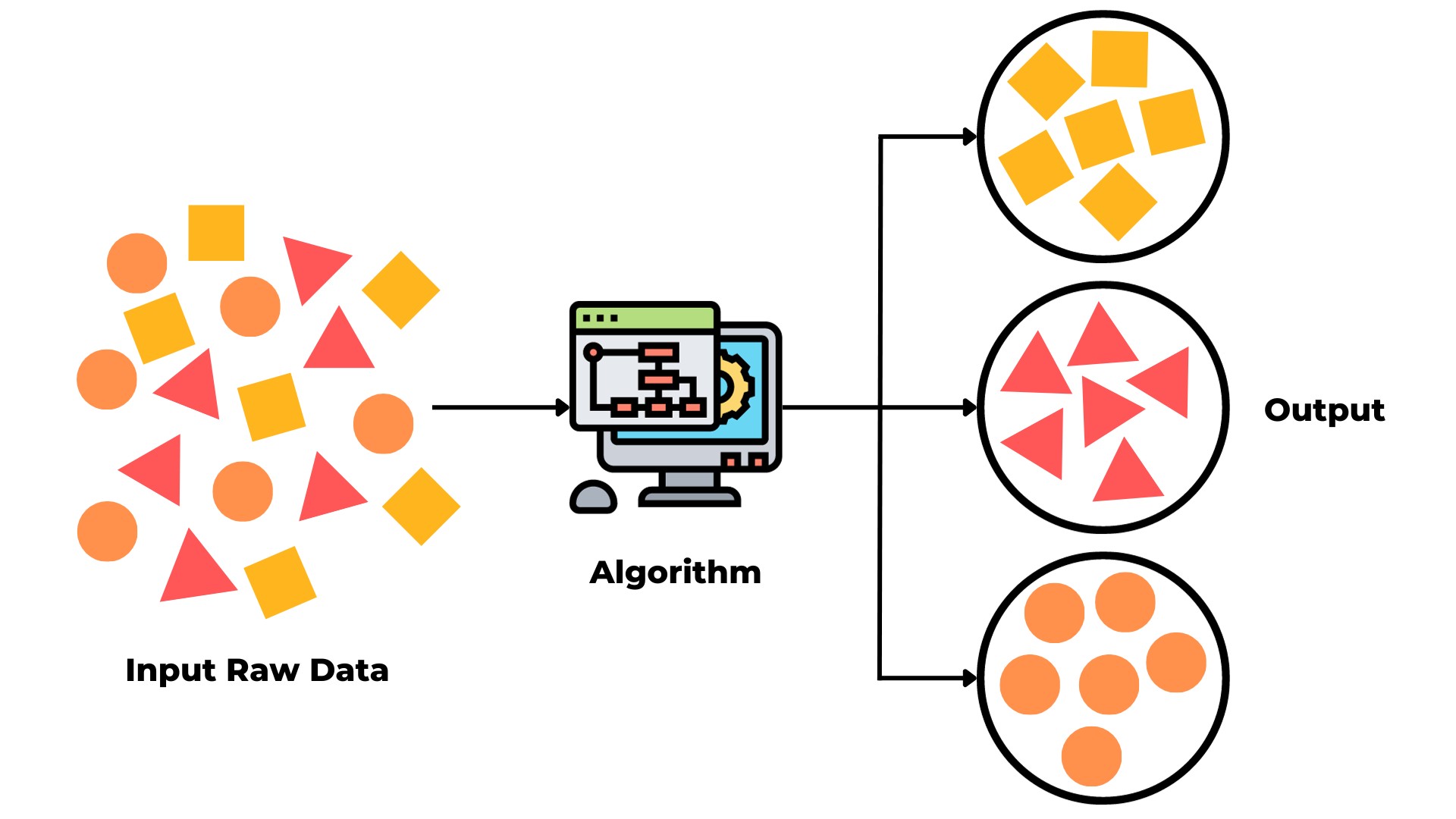


# Aprendizagem n˜ao supervisionada

Na aprendizagem n˜ao supervisionada, o agente aprende padr˜oes no input sem qualquer label a suportar a sua decis˜ao.

Um exemplo deste tipo de aprendizagem ´e clustering onde o conjunto de objectos dados como input ´e divido em grupos de acordo com um determinado crit´erio.

# Aprendizagem n˜ao supervisionada

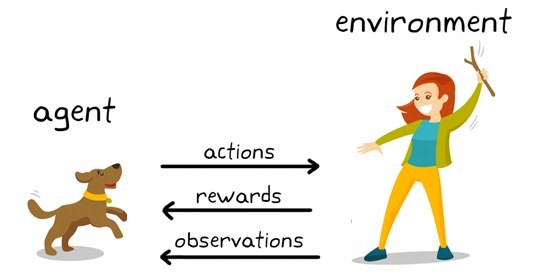


# Reinforcement Learning

No caso de reinforcement learning, o agente aprende a partir de reforc¸o positivo e negativo.

Por exemplo, no final de um jogo de xadrez o agente ´e reforc¸ado positivamente caso ganhe e refor¸cado negativamente caso perca.

# Reinforcement Learning



# Aprendizagem Supervisionada

Uma tarefa de aprendizagem supervisionada pode ser formalmente definida da seguinte forma:

Dado um conjunto de treino com *N* pares input-output

(*x*1*,y*1)*,*(*x*2*,y*2)*,...*(*xN,yN*) onde cada par foi gerado por uma fun¸c˜ao desconhecida *y* = *f* (*x*), queremos descobrir uma fun¸c˜ao *h* que aproxima a verdadeira fun¸c˜ao *f* .

# Aprendizagem Supervisionada

A essa func¸˜ao *h* chamamos hip´otese e ´e parte integrante do espac¸o de hip´oteses H de poss´ıveis fun¸c˜oes.

Tamb´em chamamos *h* de modelo dos dados, obtido atrav´es da classe de modelos H. Ao output *yi* chamamos ground truth pois ´e a resposta verdadeira que estamos a tentar que o nosso modelo preveja.

# Avalia¸c˜ao do modelo

A melhor forma de avaliar o nosso modelo n˜ao ser´a pela sua performance no conjunto de treino, mas sim como ele se comporta em dados n˜ao vistos anteriormente. Assim sendo, por norma existe um segundo conjunto de dados chamado conjunto de teste composto por pares (*xi,yi*) semelhante ao conjunto de treino.

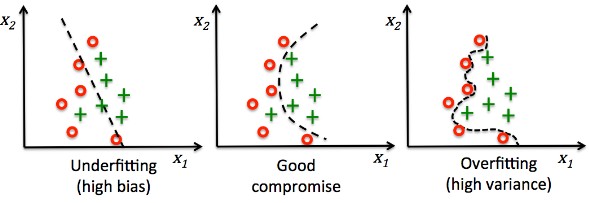
Caso o modelo preveja de forma correcta os outputs no conjunto de teste, dizemos que o modelo generaliza bem.

Dois conceitos importantes em Machine Learning s˜ao o bias e a variˆancia. Bias ´e a tendˆencia de um modelo se desviar de um valor esperado quando exposto a diferentes conjuntos de treino. O bias normalmente resulta de restric¸˜oes impostas pelo espac¸o de poss´ıveis modelos.

Por exemplo, o espa¸co de hip´oteses das func¸˜oes lineares imp˜oe um bias aos modelos na medida em que o pr´oprio modelo apenas pode ser constitu´ıdo por uma linha recta. Caso existam padr˜oes nos dados n˜ao capturados por uma linha recta, o modelo n˜ao ser´a capaz de os representar e aprender. Chamamos de underfitting `a incapacidade do modelo de encontrar os devidos padr˜oes nos dados.

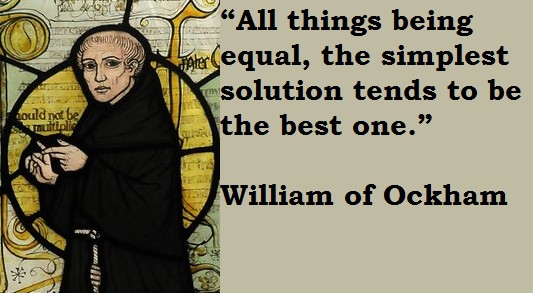
A variˆancia consiste na quantidade de mudanc¸as nos dados devido a flutuac¸˜oes nos dados de treino.

Chamamos de overfitting `a excessiva capacidade de adapta¸c˜ao do modelo a um conjunto de dados em particular no qual foi treinado, fazendo com que o modelo tenha baixa performance em dados novos.

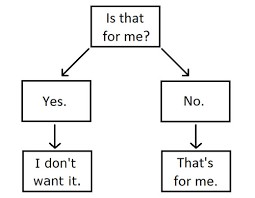


Normalmente tentamos equilibrar estes dois componentes, tendo em conta o bias-variance tradeoff: uma escolha entre modelos mais complexos com baixo bias que se adaptam bem aos dados de treino e modelos mais simples com pouca variˆancia que generalizem melhor.

Um princ´ıpio u´til de seguir ´e o da Ockham’s Razor: Se temos duas poss´ıveis respostas, mas uma ´e mais simples, provavelmente ´e a mais correcta.



# Decision Trees

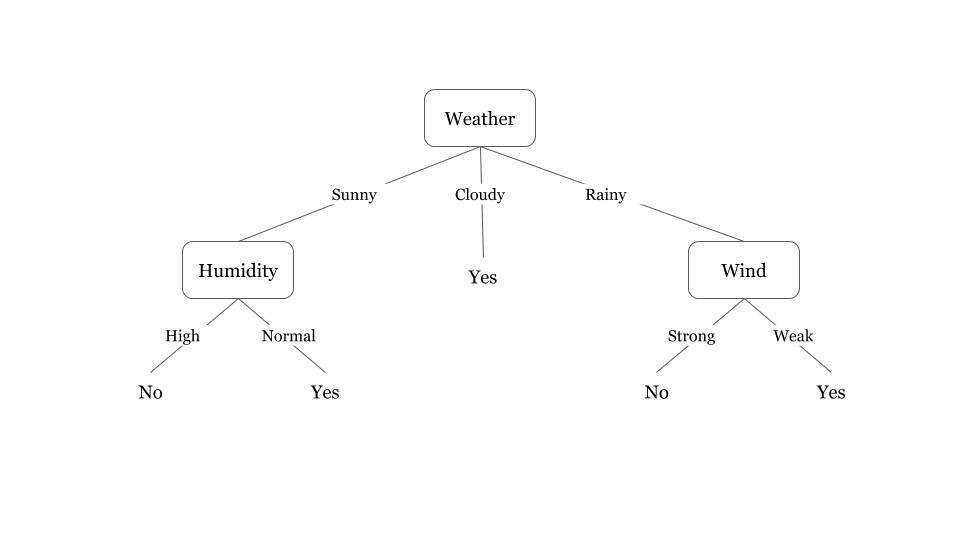
Uma Decision Tree ´e uma representac¸˜ao de uma fun¸c˜ao que mapeia um vector de valores de atributos num u´nico valor de output.

Uma ´arvore de decis˜ao chega a um output ap´os fazer uma sequˆencia de testes comec¸ando na ra´ız da ´arvore e percorrendo os ramos certos at´e chegar a uma folha.

# Decision Trees

Cada n´o interno corresponde a um teste ao valor de um dos atributos, os ramos correspondem aos poss´ıveis valores desse atributo e os as folhas especificam que valor deve ser retornado pela fun¸c˜ao.

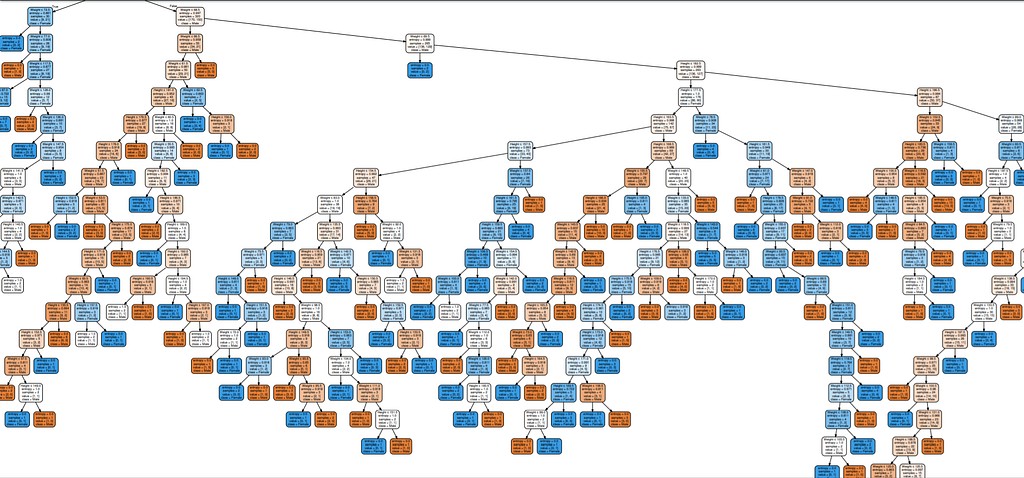
Tanto o input como o output de uma ´arvore de decis˜ao pode ser um valor discreto ou cont´ınuo.



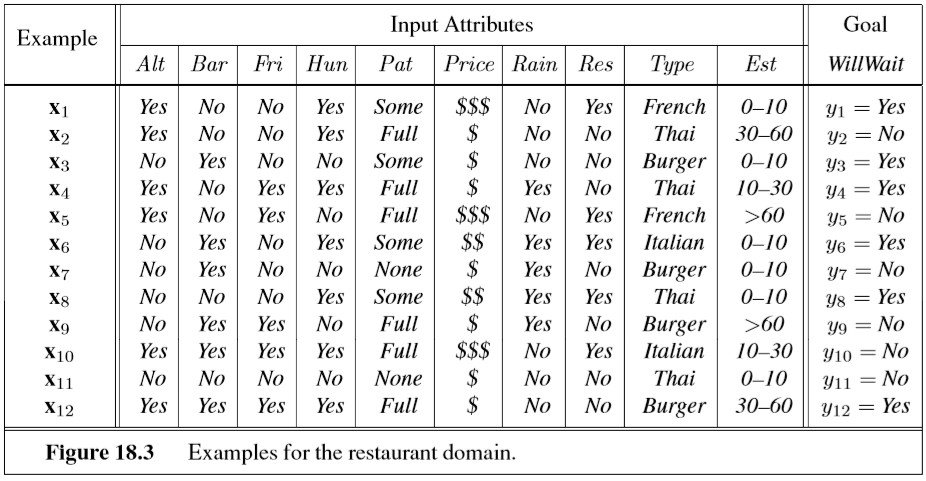
# Expressividade

Para muitos casos, as ´arvores de decis˜ao representam um resultado bastante f´acil de perceber.

No entanto, existem casos onde as ´arvores de decis˜ao n˜ao s˜ao a melhor forma de representa¸c˜ao do modelo (fun¸c˜ao de paridade, por exemplo).



Vamos analisar o seguinte conjunto de dados:



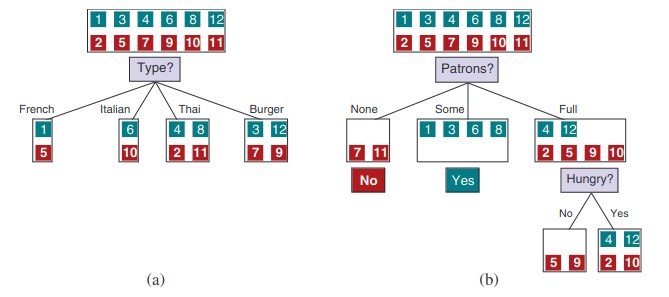
O nosso objectivo ´e gerar uma ´arvore de decis˜ao que seja consistente com o conjunto de dados que temos e que seja o mais pequena poss´ıvel. Vamos falar de um algoritmo greedy que ir´a escolher sempre o atributo mais importante para estar mais perto da ra´ız da ´arvore e depois ir´a construir a restante ´arvore recursivamente.

O atributo mais importante ´e aquele que melhor nos ajuda a classificar os exemplos.

Neste caso, podemos ver que o atributo mais importante dos dois ´e o Patrons.

Com Type, mantemos exactamente a mesma distribuic¸˜ao ap´os a separa¸c˜ao dos exemplos pelos valores de atributos.

Com Patrons, conseguimos logo dar uma resposta relativamente a bastantes exemplos, ficando apenas por resolver o caso em que o valor de Patrons ´e ”Full”.



# Como gerar ´arvores de decis˜ao: ID3

Ap´os escolhermos o atributo mais importante, existem quatro casos poss´ıveis:

Se todos os exemplos que restam s˜ao todos positivos ou negativos, ent˜ao j´a podemos dar uma resposta: Sim ou N˜ao. Por exemplo, *Patrons* = *None*.



Se existem alguns exemplos positivos ou negativos ent˜ao voltamos a escolher o melhor atributo para os separar. Por exemplo em *Patrons* = *Full* ´e escolhido o atributo *Hungry*.



Se j´a n˜ao existem exemplos, ent˜ao quer dizer que ainda n˜ao foi visto nenhum caso com aquela combina¸c˜ao de atributos. Assim sendo, retornamos o output mais comum do conjunto de exemplos que foi usado na construc¸˜ao do n´o pai.



Se j´a n˜ao existem atributos para usar, mas ainda temos exemplos positivos e negativos, ent˜ao quer dizer que estes exemplos tˆem a mesma descric¸˜ao, mas diferentes classifica¸c˜oes. Neste caso, retornamos o valor de output mais comum neste conjunto de exemplos.



# Algoritmo ID3

Algorithm 1 ID3(*examples*,*attributes*,*parent examples*)

if *examples* is empty then return PLURALITY-VALUE(*parent examples*)

end if

if all *examples* have the same classification then return the classification

end if

if *attributes* is empty then return PLURALITY-VALUE(*examples*)

end if

*A* ← argmax*a*∈*attributes* IMPORTANCE(*a*,*examples*) *tree* ← a new decision tree with root test A for all value *v* of *A* do

*exs* ←{*e* : *e* ∈ *examples* and *e.A* = *v*}

*subtree* ← LEARN-DECISION-TREE(*exs*,*attributes* − *A*,*examples*) add a branch to *tree* with label (*A* = *v*) and subtree *subtree*

end for return *tree*

Para determinar a importˆancia, vamos usar a noc¸˜ao de ganho de informa¸c˜ao, que deriva da definic¸˜ao de entropia.

Entropia ´e uma medida de incerteza de uma vari´avel aleat´oria: quanto mais informa¸c˜ao, menos entropia.

Se uma vari´avel aleat´oria tem apenas um valor poss´ıvel, a entropia ´e 0. Uma moeda equilibrada com 2 faces tem 1 bit de entropia (2 resultados igualmente poss´ıveis).

Se uma moeda est´a viciada e 99% das vezes que a atiramos, recebemos ”cara”, ent˜ao a entropia ser´a muito menor que 1, uma vez que a incerteza ´e tamb´em ela muito menor.

Podemos definir entropia como:

*H*(*V*) = −X*P*(*vk*)*log*2*P*(*vk*) (1)

*k*

onde *V* ´e uma vari´avel aleat´oria com valores *vk* com probabilidades *P*(*vk*) Podemos calcular a entropia de uma moeda equilibrada:

*H*(*equilibrada*) = −(0*.*5*log*20*.*5 + 0*.*5*log*20*.*5) = 1 (2)

Para uma moeda viciada:

*H*(*viciada*) = −(0*.*99*log*20*.*99 + 0*.*01*log*20*.*01) = 0*.*08 (3)

Vamos ent˜ao definir a seguinte func¸˜ao para ajudar na representa¸c˜ao o c´alculo do ganho de informa¸c˜ao de uma vari´avel:

*B*(*q*) = −(*qlog*2*q* + (1 − *q*)*log*2(1 − *q*)) (4)

Assim, podemos dizer que *H*(*viciada*) = *B*(0*.*99) = 0*.*08. Se tivermos um dataset com *p* exemplos positivos e *n* exemplos negativos, ent˜ao a entropia da vari´avel de output no conjunto inteiro ´e:

*p*

*H*(*output*) = *B*( ) (5)

*p* + *n*

No exemplo do restaurante, temos que *p* = 6 e *n* = 6. Logo, a entropia pode ser calculada como *B*(0*.*5) = 1.

A ideia ´e escolher um atributo *A* de tal forma que a entropia do conjunto de dados desc¸a. Medimos esta redu¸c˜ao calculando a entropia que resta depois de efectuado o teste ao atributo.

Um atributo *A* com *d* valores diferentes divide o conjunto de treino *E* em subconjuntos *E*1*,...,Ed*. Cada subconjunto *Ek* tem *pk* exemplos positivos e *nk* exemplos negativos, pelo que precisaremos de *B*(*pkp*+*knk* ) bits de informa¸c˜ao para responder `a quest˜ao.

Um exemplo escolhido aleatoriamente tem probabilidade *ppk*++*nnk* de pertencer a *Ek*, pelo que a restante entropia depois de escolhido o atributo pode ser calculada da seguinte forma:

*d*

X *pk* + *nk pk*

*Resto*(*A*) = *B*( ) (6) *p* + *n pk* + *nk*

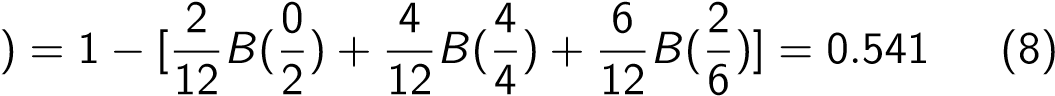
*k*=1

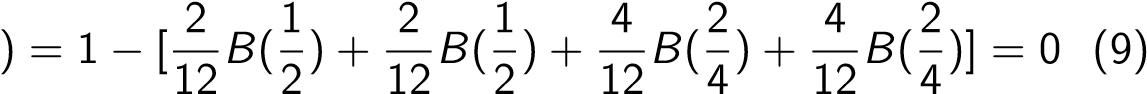
O ganho de informa¸c˜ao ´e ent˜ao calculado da seguinte forma para um atributo *A*:

*p*

*Ganho*(*A*) = *B*( ) − *Resto*(*A*) (7) *p* + *n*

A fun¸c˜ao *Ganho*(*A*) ´e, na verdade a fun¸c˜ao IMPORTANCE. Voltando ao exemplo do restaurante:

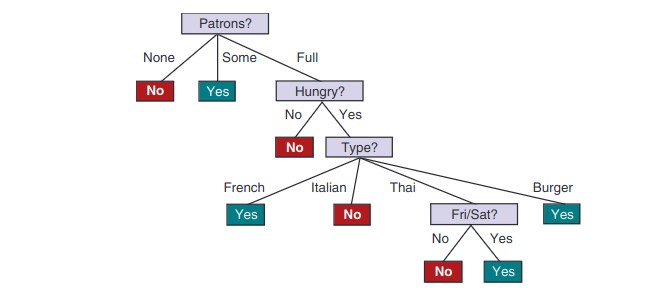
*Ganho*(*Patrons*

*Ganho*(*Type*

Confirmando a intui¸c˜ao de que Patrons ´e um atributo mais importante que Type.

# Arvore de Decis˜ao´

A ´arvore gerada pelo algoritmo para este exemplo seria:



Vamos comec¸ar por ver o caso mais simples de uma regress˜ao linear:

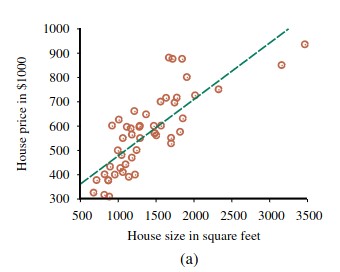
regress˜ao linear com uma fun¸c˜ao linear univariada, ou seja, vamos modelar um dados com uma linha recta.

Uma fun¸c˜ao linear univariada com input *x* e output *y* tem a forma *y* = *w*1*x* + *w*0, onde *w*0 e *w*1 s˜ao coeficientes que temos de determinar. Estes coeficientes funcionam como pesos: o valor de *y* varia consoante o peso relativo de um termo ou outro.

Vamos assumir que *w* ´e o vector ⟨*w*0*,w*1⟩ e a func¸˜ao linear com esses pesos ´e:

*hw*(*x*) = *w*1*x* + *w*0 (10)

Vamos explorar este exemplo onde cada ponto relaciona o tamanho e o pre¸co de uma casa para venda.



O objectivo ´e encontrar a func¸˜ao *hw*(*X*) que melhor se ajusta aos dados.

A esta tarefa chamamos regress˜ao linear.

A tarefa resume-se a encontrar valores para os pesos ⟨*w*0*,w*1⟩ que minimizem uma loss function.

Uma loss function cl´assica em casos de regress˜ao linear ´e a Squared-Error:

*N*

*Loss*(*hw*) = X(*yj* − (*w*1*xj* + *w*0))2 (11)

*j*=1

O objectivo ´e minimizar a func¸˜ao *Loss*(*hw*). A func¸˜ao ´e m´ınima quando as suas derivadas parciais s˜ao zero:

|  |  |
| --- | --- |
| X 2  (*yj* − (*w*1*xj* + *w*0)) = 0  *∂w*0  *j*=1 | (12) |
| *N*  *∂* X 2  (*yj* − (*w*1*xj* + *w*0)) = 0 | (13) |

*N ∂*

*∂w*1

*j*=1

Estas equa¸c˜oes tˆem uma solu¸c˜ao u´nica:

*N*(P*xjyj*) − (P*xj*)(P*yj*)

*w*1 = P(*xj*)2) − (P*xj*)2 (14)

*N*(

*w*0 = (X*yj* − *w*1(X*xj*))*/N* (15)

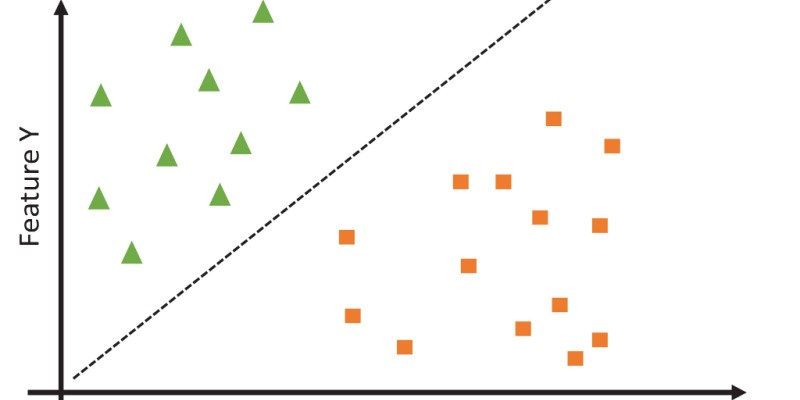
Para o exemplo das casas, a solu¸c˜ao seria *w*1 = 0*.*232 e *w*0 = 246 e a linha est´a representada a tracejado na figura.

Com as devidas extens˜oes, ´e poss´ıvel aplicar regress˜oes lineares a dom´ınios com mais vari´aveis.

# Support Vector Machines

SVMs s˜ao uma classe muito flex´ıvel e poderosa de algoritmos para classifica¸c˜ao e regress˜ao.

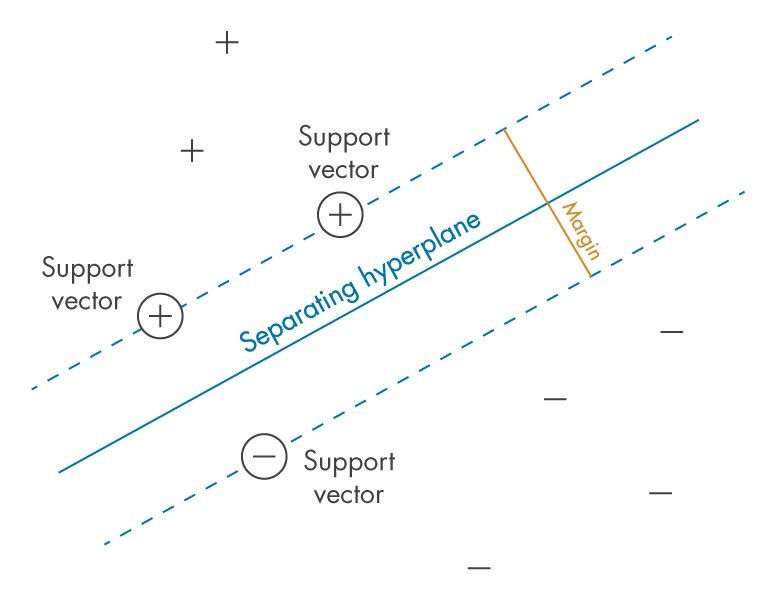
O objectivo ´e tentar encontrar uma linha (no caso de dados bidimensionais) ou uma variedade (em dados multidimensionais) capaz de dividir as classes.



A tarefa da SVM ser´a ent˜ao encontrar uma linha que separe os dois conjuntos de dados. No entanto, temos um problema: existem formas poss´ıveis de trac¸ar uma linha entre os dois conjuntos de dados. Isso poder´a influenciar a forma como iremos atribuir labels a dados novos.

# Support Vector Machines

Para que possamos desenhar a melhor linha poss´ıvel, as SVMs oferecem uma estrat´egia: cada linha ter´a associada uma margem at´e ao seu ponto mais pr´oximo e a linha que oferecer a maior margem ser´a a escolhida.



from sklearn.svm import SVC

model = SVC(kernel='linear',C=1E10) model.fit(X,y)

# Support Vector Machines: Kernels

Para que possamos utilizar todo o poder que as SVMs nos podem dar, poderemos utilizar kernels que s˜ao func¸˜oes capazes de mapear os nossos dados para espac¸os com mais dimens˜oes.

Isto ir´a permitir que, por exemplo, possamos dividir classes suja separa¸c˜ao n˜ao segue um padr˜ao linear, mapeando-as numa dimens˜ao onde essa divis˜ao passe a ser linear.

Podemos utilizar diferentes kernels no scikit (que mapear˜ao os nossos dados para outras dimens˜oes de formas diferentes). Para tal, basta alterar o valor do argumento kernel.

model = SVC(kernel='rbf',C=1E6) model.fit(X,y)

# Support Vector Machines: Kernels

model = SVC(kernel='rbf',C=1E6) model.fit(X,y)

Este uso de kernels ´e usado em m´etodos lineares para que possam tamb´em funcionar em ambientes n˜ao lineares como ´e o caso das SVMs e at´e das regress˜oes lineares.

A vari´avel C controla qu˜ao r´ıgida ´e a margem. Isto ´e, qu˜ao tolerante a divis˜ao ser´a relativamente a pontos que possam n˜ao cumprir a rigidez da separa¸c˜ao linear das classes. Um valor de C muito grande significa que a margem ser´a bastante r´ıgida; um valor de C mais baixo significar´a que a margem ser´a mais relaxada.

# Support Vector Machines

As SVMs tˆem algumas vantagens:

S˜ao modelos muito compactos, necessitando de pouca mem´oria, uma vez que dependem de um nu´mero relativamente pequeno de vectores de suporte.



A previs˜ao ´e normalmente muito r´apida.



Trabalham bem com dados com muitas dimens˜oes, uma vez que apenas s˜ao afectadas por pontos perto da margem.

A utiliza¸c˜ao de kernels torna-as muito vers´ateis.

No entanto, para datasets muito grandes, o processo de treino pode demorar bastante tempo, os resultados tamb´em dependem bastante do valor fornecido para a constante C e o modelo n˜ao ´e facilmente interpret´avel.

O conceito de Random Forests utiliza um m´etodo de ensemble chamado bagging. Bagging utiliza um conjunto de estimadores que potencialmente d˜ao overfit aos dados e faz a m´edia dos resultados para conseguir uma melhor classificac¸˜ao. Quando fazemos bagging a um conjunto de decision trees geradas para subconjuntos aleat´orio do conjunto de dados, temos uma Random Forest.

Podemos gerar uma Random Forest usando o Scikit da seguinte forma: from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=100) model.fit(X,y)

Tamb´em podemos usar Random Forests para regress˜ao, utilizando o RandomForestRegressor:

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n\_estimators = 100) model.fit(X,y)

As Random Forests tˆem imensas vantagens:

Treino e previs˜ao s˜ao relativamente r´apidos, pela simplicidade da gera¸c˜ao das ´arvores de decis˜ao.



As ´arvores permitem at´e classificac¸˜oes probabil´ısticas, podendo estimar a probabilidade de cada label de acordo com os resultados de cada ´arvore individualmente.

E um modelo muito´ flex´ıvel, podendo ser u´til em tarefas em que facilmente existe um underfit por parte de outros modelos.

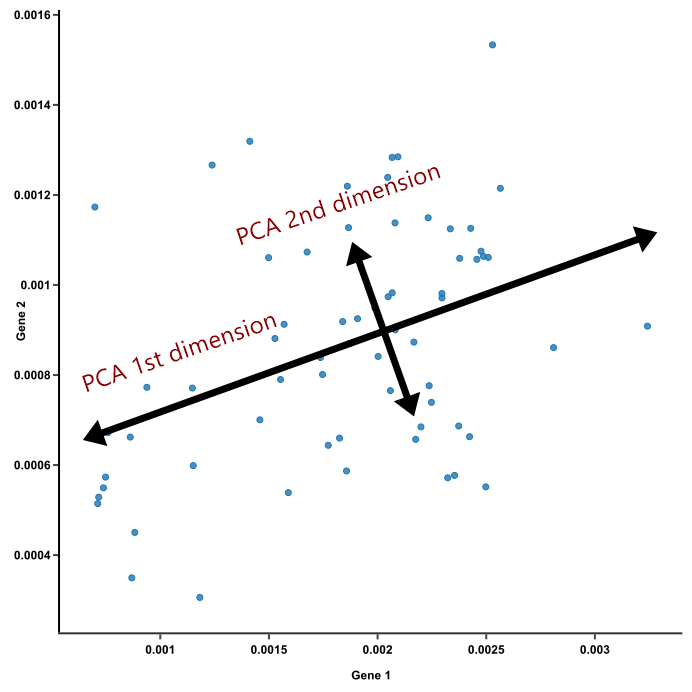


No entanto, ao utilizar Random Forests, perdemos bastante a interpretabilidade que as Decision Trees oferecem.

Um dos algoritmos n˜ao supervisionados mais utilizados ´e o PCA. Essencialmente, o PCA reduz a dimensionalidade do nosso conjunto de dados, podendo tamb´em ser u´til em tarefas de reduc¸˜ao de ru´ıdo ou extracc¸˜ao de features, por exemplo.

Enquanto que num algoritmo superivisonado n´os queremos prever valores para *y* atrav´es dos valores de *X*, aqui n´os queremos perceber a rela¸c˜ao entre *x* e *y*.

Assim sendo, n´os tentamos encontrar quais s˜ao os principais atributos dos nossos dados e utilizamos esses atributos para descrever o dataset. from sklearn.decomposition import PCA



Alberto Barbosa (UMAIA)

SI - Introdu¸c˜ao

May 26, 2025

45

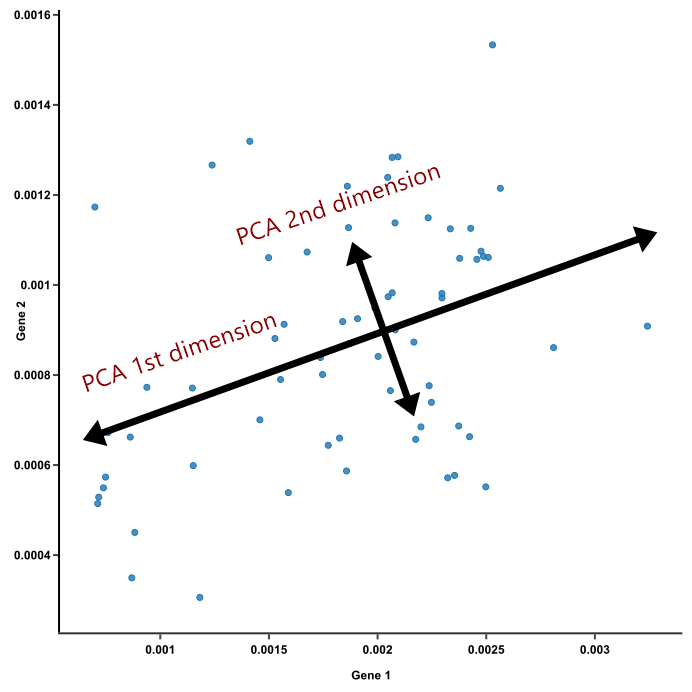
/

60

pca = PCA(n\_componentes = 2) pca.fit(X)

Podemos aceder aos parˆametros mais importantes:

print(pca.components\_) print(pca.explained\_variance\_)



Os vectores obtidos s˜ao os principais eixos dos nosso dados e o tamanho mede a ”importˆancia” desse eixo na distribui¸c˜ao dos nossos dados, sendo uma medida de variˆancia dos dados quando projectados nesse eixo. A projecc¸˜ao de cada ponto nos eixos principais s˜ao os componentes principais dos dados e ´e feita atrav´es de uma func¸˜ao afim.

A utiliza¸c˜ao do PCA enquanto algoritmo para reduc¸˜ao da dimensionalidade do dataset envolve descartar os componentes principais mais pequenos, resultando numa projec¸c˜ao dos dados para menos dimens˜oes preservando a variˆancia maximal dos mesmos.

pca = PCA(n\_components=1) pca.fit(X)

X\_pca = pca.transform(X) *#ter´a apenas uma dimens~ao* print(X\_pca.shape) *#ser´a do tipo (L,1)*

Embora tenhamos falado de PCA enquanto m´etodo de reduc¸˜ao de dimensionalidade, ele tamb´em pode ser u´til na remoc¸˜ao de ru´ıdo e para feature selection, por exemplo.

Uma desvantagem do PCA ´e que tende a ser bastante afectado por outliers. Podemos contrariar isto usando variantes mais robustas do PCA, como o RandomizedPCA ou SparsePCA do Scikit.

Outro tipo de algoritmos n˜ao supervisionados s˜ao os algoritmos de clustering.

Algoritmos de clustering tentam agrupar os pontos do nosso conjunto de dados atrav´es das propriedades dos mesmos sem recorrer a quaisquer labels predefinidas dos dados.

Existem imensos algoritmos de clustering dispon´ıveis no Scikit. No entanto, um dos mais simples de perceber ser´a o k-means, que est´a dispon´ıvel na classe sklearn.cluster.KMeans.

O k-means procura um nu´mero predeterminado de clusters num conjunto de dados sem labels. Para tal, baseia-se nas seguintes noc¸˜oes de um cluster ´optimo:

O centro do cluster ´e a m´edia aritm´etica de todos os pontos pertencentes ao mesmo.



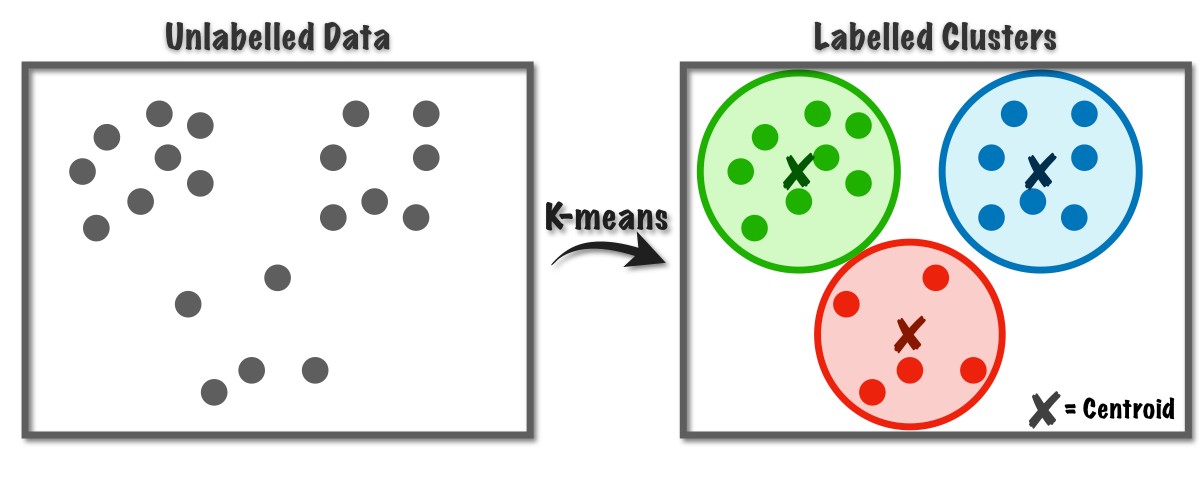
Cada ponto est´a mais pr´oximo do centro do seu cluster do que do centro de outros clusters.

O k-means procura um nu´mero predeterminado de clusters num conjunto de dados sem labels. Para tal, baseia-se nas seguintes noc¸˜oes de um cluster ´optimo:

O centro do cluster ´e a m´edia aritm´etica de todos os pontos pertencentes ao mesmo.



Cada ponto est´a mais pr´oximo do centro do seu cluster do que do centro de outros clusters.



from sklearn.cluster import KMeans kmeans = KMeans(n\_clusters=4) kmeans.fit(X) preds = kmeans.predict(X)

O k-means tem uma forma muito simples de funcionar:

De seguida, enquanto n˜ao convergir:



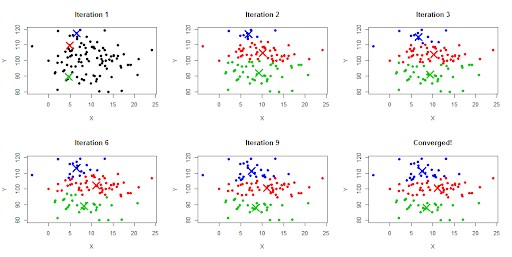
Inicialmente, define ”aleatoriamente” alguns centros de clusters.



Atribui pontos ao cluster com o centro mais pr´oximo.



Actualiza o centro dos clusters com a m´edia dos seus pontos.



Emnbora bastante intuitivo, o uso de k-means pode ter algumas limita¸c˜oes:

O resultado ´optimo pode n˜ao ser conseguido pois pode ficar preso em m´aximos locais e n˜ao chegar a um m´aximo global. Podemos mitigar isto permitindo ao algoritmo ter v´arios palpites iniciais.



O nu´mero de clusters tem de ser definido previamente, n˜ao aprendendo o nu´mero de clusters dos dados. Outros algoritmos j´a aprendem isto como DBSCAN tamb´em dispon´ıvel no Scikit.



O k-means assume que os clusters s˜ao separ´aveis linearmente, n˜ao lidando bem com outras geometrias de clusters. O



SpectralClustering do Scikit faz exactamente isso utilizando kernels.

Pode ser lento para datasets muito grandes, uma vez que visita cada ponto uma vez durante cada itera¸c˜ao. Isto pode ser mitigado atrav´es de agoritmos k-means batch-based como o MiniBatchKMeans do Scikit, onde apenas um subconjunto dos dados ´e utilizado para actualizar os clusters.



# M´etricas de avalia¸c˜ao de classificadores

Os classificadores atribuem labels a cada uma das observa¸c˜oes que lhes forem fornecidas. No entanto, precisamos de arranjar t´ecnicas para medir qu˜ao bem estas atribui¸c˜ao est˜ao a ser feitas.

Uma das medidas mais f´aceis de entender ´e a Accuracy. Para efeitos de simplicidade vamos assumir que temos duas classes: Verdade e Falso.

*TP* + *TN*

*Accuracy* = (16)

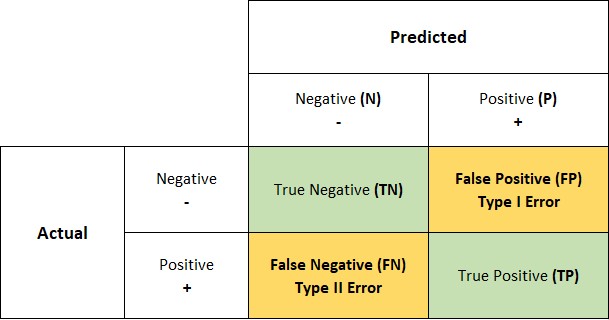
*TP* + *TN* + *FP* + *FN*

Onde TP significa True Positives (valores que o modelo diz que s˜ao verdade e s˜ao de facto), TN significa True Negatives (valores que o modelo descreve como falso e s˜ao de facto), FP significa False Positives (valores que o modelo diz serem verdade, mas s˜ao falsos) e FN significa False Negatives (valores previstos como falsos, mas que s˜ao verdadeiros). Um valor de 99% de Accuracy significa que o nosso modelo acertou em 99% dos casos que lhe foram dados a classificar.

No entanto, a Accuracy pode ser bastante enganadora, dependendo das caracter´ısticas do conjunto de dados.

# M´etricas de avalia¸c˜ao de classificadores

Uma forma muito comum de avaliarmos os nossos classificadores ´e atrav´es da an´alise de uma matriz de confus˜ao. A matriz de confus˜ao combina os valores previstos e os valores verdadeiros das observa¸c˜oes numa tabela.



Atrav´es da an´alise desta tabela, ´e poss´ıvel extrair v´arias m´etricas como Precision, Recall, Accuracy e AUC-ROC, entre outras.

from sklearn.metrics import confusion\_matrix trues = [2, 0, 2, 2, 0, 1] preds = [0, 0, 2, 2, 0, 2]

confusion\_matrix(trues,preds)

# Precision

Calcula quantos dos casos em que o modelo acertou s˜ao positivos. E u´til´ quando ´e prefer´ıvel existirem Falsos Negativos a existirem Falsos Positivos.

*TP*

*Precision* = (17)

*TP* + *FP*

from sklearn.metrics import precision\_score trues = [2, 0, 2, 2, 0, 1] preds = [0, 0, 2, 2, 0, 2]

precision\_score(trues,preds)

# Recall

Calcula quantos positivos fomos capaz de prever correctamente. E u´til´ quando ´e prefer´ıvel existirem Falsos Positivos a existirem Falsos Negativos.

*TP*

*Recall* = (18) *TP* + *FN*

from sklearn.metrics import recall\_score trues = [2, 0, 2, 2, 0, 1] preds = [0, 0, 2, 2, 0, 2]

recall\_score(trues,preds)

# F1 Score

M´edia harm´onica de Precision e Recall. M´axima quando ambas as m´etricas s˜ao iguais. Bastante u´til quando FP e FN s˜ao igualmente maus.

*Precision* × *Recall*

*F*1 = 2 × (19)

*Precision* + *Recall*

from sklearn.metrics import f1\_score trues = [2, 0, 2, 2, 0, 1] preds = [0, 0, 2, 2, 0, 2]

f1\_score(trues,preds)

Tal como acontece na classifica¸c˜ao, ´e u´til saber qu˜ao boas est˜ao a ser as previs˜oes dos nossos regressores.

Para tal, existem algumas m´etricas b´asicas e intuitivas que nos podem ajudar a perceber melhor qu˜ao boa ´e a performance do nosso modelo.

Uma das mais b´asicas ´e o Mean Squared Error (MSE), tamb´em muitas vezes usada como loss function em alguns algoritmos de ML e representa a m´edia do quadrado dos erros do nosso regressor:

*n*

1 X − *TrueYi*)2 (20)

*MSE* = (*Yi*

*N*

*i*=1

O facto de considerarmos o quadrado do erro, ir´a inflacionar erros muito grosseiros.

from sklearn.metrics import mean\_squared\_error

trues = [1,1,0,0,1,0] preds = [0.95,0.85,0.9,0.8,0.7,0.3] error = mean\_squared\_error(trues,preds) Uma extens˜ao desta m´etrica ´e a RMSE (Root Mean Squared Error) que corresponde `a ra´ız quadrada do MSE. Podemos calcular esta func¸˜ao passando um argumento adicional `a MSE do Scikit:

error = mean\_squared\_error(trues,preds,squared=False) Outra m´etrica bastante comum ´e Mean Absolute Error (MAE) que ao contr´ario das suas m´etricas anteriores n˜ao penaliza com magnitudes diferentes erros de ordem diferentes (pois n˜ao faz o quadrado dos erros).

*n*

*MAE* X|*Yi* − *TrueYi*| (21)

*N*

*i*=1

from sklearn.metrics import mean\_absolute\_error

trues = [1,1,0,0,1,0] preds = [0.95,0.85,0.9,0.8,0.7,0.3] error = mean\_absolute\_error(trues,preds)