Aprendizagem máquina

Conceitos e arquitetura computacional

Modelos de aprendizagem supervisionada

Aprendizagem Automática / *Machine Learning*

Definição:

Um programa de computador aprende com a experiência E, numa classe de tarefas T, dada uma medida de desempenho P, se o seu desempenho nas tarefas contidas em T, medido por P, melhora com a experiência E.

(Mitchell, 1997)

Este campo tem já algumas décadas e fornece muita da base técnica para o desenvolvimento de modelos e algoritmos de Mineração de Dados

Paradigmas da Aprendizagem Automática

Não supervisionada:

não é fornecida nenhuma indicação externa, sendo a aprendizagem realizada pela descoberta de regularidades nos dados de entrada (e.g. clustering).

Supervisionada

é fornecida uma resposta correcta para cada situação; aprendizagem é realizada a partir de exemplos compostos por um vector de entradas e por um vector de saídas desejadas. Correspondem às tarefas de **predição** da Mineração de Dados.

Por reforço

apenas se fornecem "recompensas ou "punições" mediante as respostas dadas pelo modelo em determinadas situações

Aplicações

Atribuição de crédito por parte de uma instituição bancária baseada na informação disponibilizada pelo cliente;

Diagnóstico médico a partir de sintomas/sinais;

Previsão das necessidades ao nível do fornecimento de energia numa dada área geográfica;

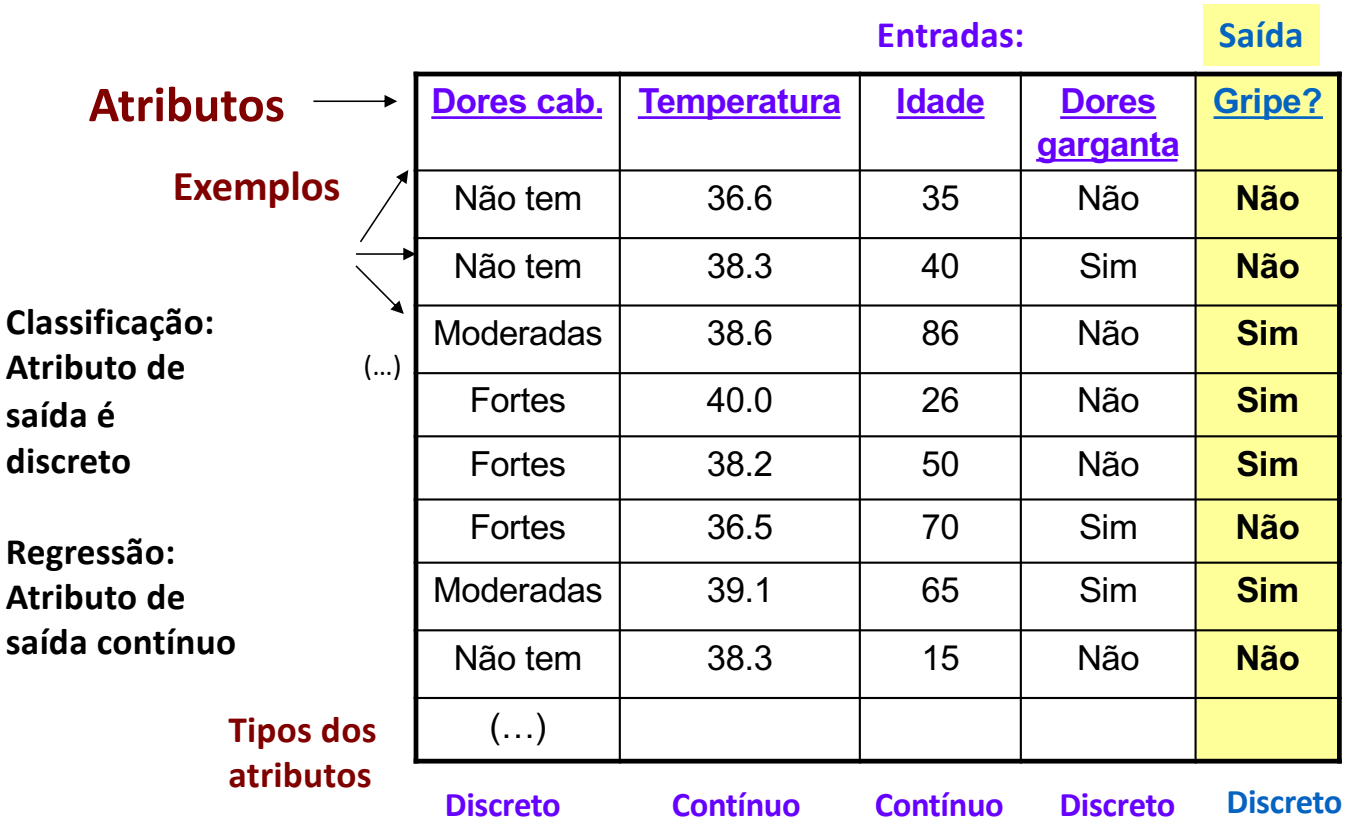
Detecção de manchas de petróleo nos oceanos a partir de imagens recolhidas por satélite;

Detecção precoce de falhas ou comportamentos atípicos em equipamentos electromecânicos;

Análise do comportamento de clientes de diversos serviços ...

entre muitas outras ...

Conjuntos de dados – aprendizagem supervisionada



Atributos e exemplos

Atributo: Par ***(a,D)***, em que ***a*** é o nome e ***D*** o domínio do atributo (conjunto de valores possíveis).

Se D = {d1, d2, ..., dn} – atributo **discreto** ou **enumerado**

Se D ⊆ R – atributo **numérico** ou **contínuo**

Exemplo: Conjunto de pares ***(a,v)***, em que ***(a,D_a)*** é um atributo e ***v*** $\in D_a$

Conjunto de dados: Tuplo ***(A,S,E)*** em que

A – conjunto de ***M*** atributos de entrada: ***A = {(a_i, D_i), i=1, …, M}***

S – atributo de saída: ***S = (s,D_s)***

E – conjunto de ***N*** exemplos: ***E = { Ej | j=1, …, N}, Ej={(a_i, V_{ij}), i=1, …, M, $V_{ij} \in D_i$} ∪{(s,V_{sj}), $V_{sj} \in D_s$ }***

Aprendizagem supervisionada – modelos e algoritmos

Um **modelo**

é uma função que permite obter o valor de variáveis de saída a partir dos valores de variáveis de entrada; **generaliza** a partir dos dados existentes

Para um conjunto de dados ***X = (A,S,E)***, um modelo é uma função

$F: D_1, D_2, \ldots, D_M \rightarrow D_s$

Os **algoritmos de aprendizagem**

permitem obter um modelo a partir de um conjunto de **exemplos de treino** (onde se conhecem entradas e saídas);

efetuam um processo de procura do melhor modelo definido por uma dada **estrutura de representação** tipicamente a partir de um processo de otimização que otimiza uma função objetivo (e.g. minimizar função de erro)

Aprendizagem supervisionada: modelos e algoritmos

Duas fases distintas:

Construção (ou treino) do modelo a partir dos dados conhecidos (entradas e saídas) – normalmente passa por, dada uma estrutura de representação, determinar o valor de alguns parâmetros de forma a minimizar uma função objetivo – **problema de optimização**

Predição - Aplicação do modelo a dados onde apenas se conhece o valor dos atributos de entrada e pretende prever-se o valor do atributo de saída

Estruturas de representação para modelos

Árvores de decisão

Regras de classificação / regressão

Redes Neuronais Artificiais

Métodos estatísticos (bayesianos)

Métodos baseados em instâncias (por ex. método dos vizinhos mais próximos)

Modelos funcionais lineares

(…)

Avaliação de modelos: medidas de erro

A avaliação da **qualidade** de um modelo para uma dada tarefa faz-se calculando medidas de erro sobre um determinado **conjunto de exemplos** (preferencialmente não usados na construção do modelo)

Estas medidas de erro dependem do tipo de problema: classificação ou regressão

Medidas de erro: classificação

- **Percentagem de exemplos corretamente classificados** (aka **accuracy**)

  PECC = (VN+VP) / (VN+VP+FP+FN) (x100%)

  Para mais de duas classes

  PECC = Nº exemplos corretamente classificados /
  Nº total de exemplos do conjunto de dados

- **Sensibilidade** (erro tipo II): (aka **recall**)

  sensibilidade = VP / (FN + VP)
  E(tipo II) = FN / (FN + VP) = 1 - sensibilidade

- **Especificidade** (erro tipo I)

  especificidade = VN / (VN + FP)
  E(tipo I) = FP / (FP + VN) = 1 – especificidade

Medidas de erro: classificação

- Positive predictive value (aka **precision**)

  PPV = VP / (VP + FP)

- Negative predictive value

  NPV = VN / (VN + FN)

- Medida F

  F1 = 2 x (precision x recall) / (precision + recall)

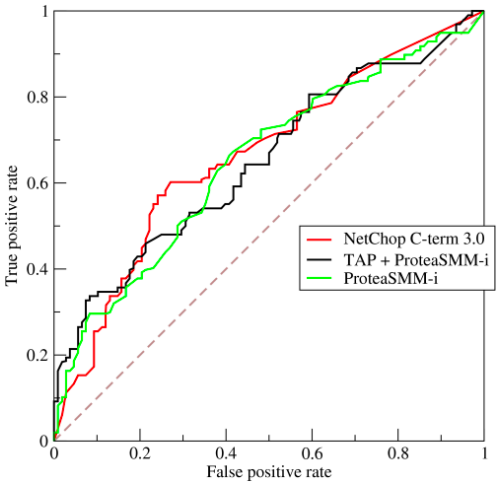
- Matthews Correlation Coefficient

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

NOTA: No caso de existirem mais do que 2 classes, os valores são calculados como se existisse uma **matriz de confusão** 2x2 para cada classe, onde os valores "negativos" são os elementos de todas as outras classes

Curvas ROC (Receiver Operating Curve)

- Gráfico que determina a capacidade de discriminação de um classificador à medida que um threshold de classificação se altera
- Mostra a taxa de Verdadeiros Positivos (eixo yy) em função da taxa de Falsos Positivos
- A área abaixo da curva ROC (AUC) é um bom indicador da qualidade de um classificador (sendo 0,5 um valor para um classificador aleatório e 1 um valor para um classificador perfeito)
- Também podem ser criados gráficos de Precision-Recall que são mais adequados quando há desbalanceamento de dados (com a classe negativa maioritária)



Medidas de erro - regressão

- Neste caso, calculam-se medidas com base no **erro** cometido para cada exemplo. Este é a diferença entre o valor previsto (y^) e o valor real (y)
- Para N exemplos podem usar-se várias medidas de erro:
  - Soma quadrado dos erros (SQE);
  - Raíz quadrada da média do quadrado dos erros (RMQE)
  - Média dos desvios absolutos (MDA)
- Estes valores podem igualmente ser normalizados para poderem ser comparados entre diversos problemas (e.g. RRQE).
- Podem usar-se ainda:
  - Medidas de correlação (e.g. Pearson, Spearman)
  - O coeficiente de determinação R^2 – mede a proporção da variabilidade na variável de saída explicada pelo modelo (valor entre 0 e 1)

Métodos de estimação do erro

Objectivo: avaliar **credivelmente** o desempenho de um algoritmo

Não é possível fazê-lo aplicando uma medida de erro ao mesmo conjunto de dados que deu origem ao modelo – necessário definir um conjunto de **exemplos de teste**

Exemplos de teste devem ser retirados da mesma população de forma **independente** dos exemplos de treino

Divisão da amostra

Método mais popular: divisão do conjunto de exemplos inicial em 2 grupos – **treino + teste**

Este método toma a designação de **divisão da amostra**.

É simples e rápido, mas muito sensível à forma como a amostra é dividida.

Validação cruzada

Melhoramento do método de divisão da amostra que permite usar todos os dados disponíveis.

Validação cruzada k-desdobrável *(k-fold cross validation*) –

Exemplos são divididos em k sub-conjuntos **mutuamente exclusivos** (de tamanhos idênticos);

algoritmo aplicado e testado k vezes;

em cada iteração *i* (*i*=1,...,k): modelo criado a partir de todos os exemplos excepto os do grupo *i* e testado apenas nos exemplos do grupo *i.*

erro final calculado pela **média** dos erros em cada iteração.

Exemplo para k=3

Estratificação da amostra e leave-one-out

- Em problemas de classificação: nos métodos de partição de conjuntos para estimação (i.e. nos dois anteriores) poderá haver o cuidado de manter as proporções relativas de cada classe nos diversos conjuntos
- Processo de **estratificação** da amostra deverá garantir o melhor esforço possível para este resultado (i.e. nº de exemplos de cada classe nos diversos grupos não varia mais do que 1)
- Caso particular validação cruzada estratificada: **leave-one-out**:
  - k é igual ao nº de exemplos -> uma partição por cada exemplo
  - Em cada iteração modelo é criado com todos os exemplos excepto um; este será o exemplo a prever pelo modelo.
  - Método mais pesado computacionalmente; resultados mais fiáveis

Tendência de indução

Dado um conjunto de exemplos existem sempre inúmeros modelos possíveis que explicam estes exemplos satisfatoriamente.

Para se conseguir generalizar, temos que impor uma "tendência" ou restrições ao processo de construção do modelo – **tendência de indução**.

A tendência de indução pode reflectir-se a vários níveis, dos quais se destacam:

- Estrutura de representação (qual a estrutura do modelo);
- Processo de procura pelo melhor modelo (qual o algoritmo?)

Sobreajustamento

Um dos maiores problemas no processo de aprendizagem: o modelo aprende "demasiado bem" os exemplos e **perde capacidade de generalizar**, i.e. o erro nos exemplos de treino diminui mas no universo de interesse aumenta.

Como se pode evitar ?

- Tendo mais dados;
- Evitando modelos demasiado complexos em relação aos dados disponíveis;
- Evitando prolongar o processo de aprendizagem em demasia ...

Na prática, o problema é difícil de resolver ... serão discutidas diversas abordagens ao longo deste curso !

Aprendizagem baseada em instâncias

Nesta classe de algoritmos não é criado explicitamente um modelo que generaliza os dados de treino, mas estes são apenas **armazenados**.

Os dados armazenados são apenas usados quando se pretende fazer a classificação de um novo exemplo (daí o nome de ***lazy learning***).

Métodos mais lentos na fase de previsão, mas rápidos na fase de aprendizagem.

Algoritmo dos k-vizinhos mais próximos

Método mais popular da classe anterior

Como funciona ?

- na fase de **treino**: armazenar os exemplos disponíveis;
- quando se pretende **classificar um novo exemplo**:
  - calcular os **k exemplos mais semelhantes** ao exemplo a classificar (usando atributos de entrada);
  - calcular a classe mais comum nos k exemplos – essa será a classe prevista
  - para problemas de **regressão** faz-se a média dos valores do atributo de saída nos k exemplos

Necessário definir uma **função de similaridade** entre dois exemplos, desejavelmente normalizada (e.g. [0,1]) – nem sempre simples !

Modelos clássicos de regressão

Representam relação entre variáveis de **entrada** $X_1,...,X_p$ (variáveis independentes), e uma variável de **saída** Y (variável dependente).

Previsão do modelo dado por:

$$\hat{y} = f(x_1, \ldots, x_p; \theta)$$

p – nº de entradas

ϑ - parâmetros do modelo

Estimação dos parâmetros

- Sabendo a estrutura do modelo -> problema de **optimização numérica** – minimização de uma função de erro

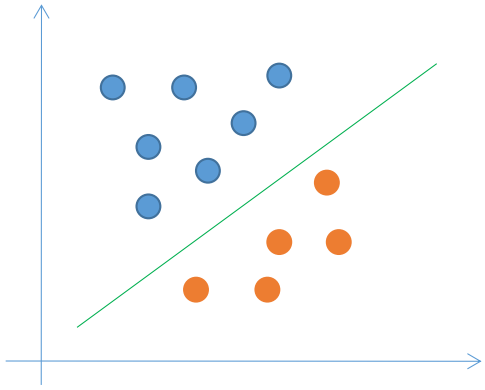
- Nos modelos lineares, pode usar-se o método dos **mínimos quadrados**, que minimiza a SQE.

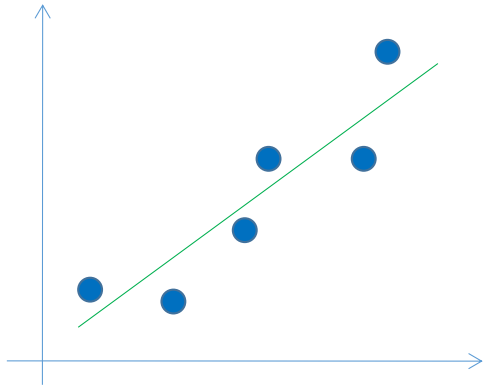
- Algoritmos de **gradiente descendente** são usados em muitos casos onde a função de erro é diferenciável, embora possam convergir para mínimos locais.

Modelos lineares

Atractivos, dada a simplicidade do cálculo e da análise

Podem ser usados para **classificação** (separação entre classes) ou **regressão**.





Modelos de regressão linear

Caso geral: modelos de **regressão**:

$$\hat{y} = \beta_0 + \sum_{i=1}^{p} \beta_i x_i$$

Se p = 1 : **regressão linear**

Se p >= 2: **regressão linear múltipla**

β_i - parâmetros do modelo

Regressão logística

Variável dependente discreta: problema de **classificação**

Regressão **logística**: usa modelos de regressão para classificação binária interpretando a saída do modelo de forma a extrair uma classe

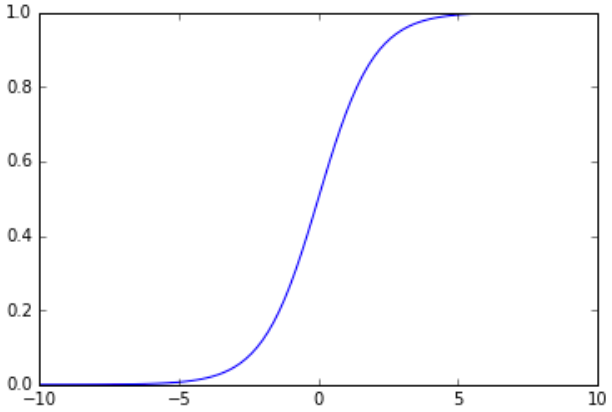
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$0 \leq h_\theta(x) \leq 1$$

Interpretação: h estima a **probabilidade** de y (saída) ser igual a 1 para o exemplo *x*

$$\frac{1}{1 + e^{-z}}$$

Função **sigmoid** (logística)



Support Vector Machines

Máquinas de vetor de suporte

As Máquinas de Vector de Suporte/ *Support Vector Machines* (SVM) foram desenvolvidas por Vapnik no início da década de 90

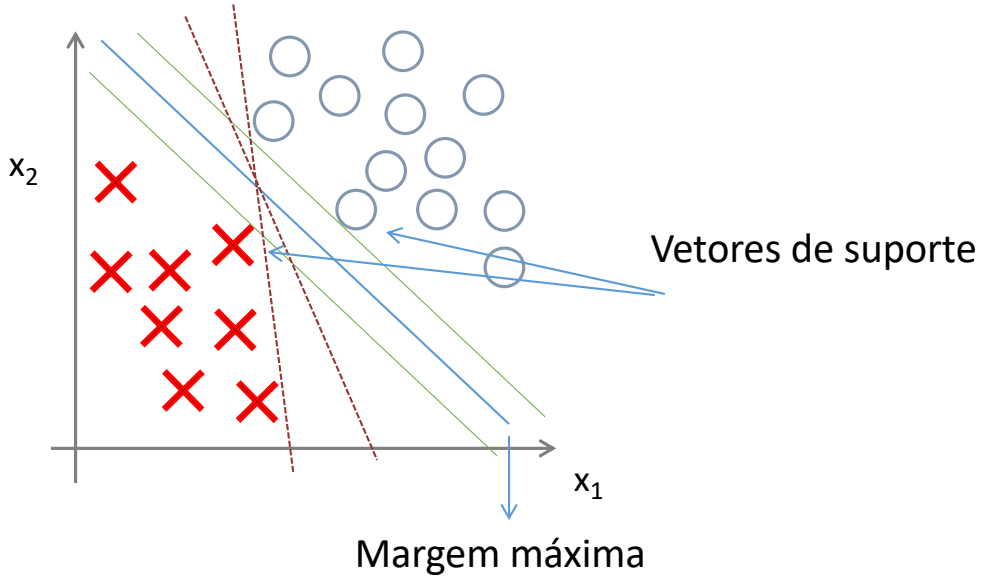
Têm vindo a ganhar popularidade como ferramentas de classificação e regressão, devido a uma série de vantagens teóricas e resultados empíricos

Baseiam-se em 2 ideias:

1 - utilizar somente alguns exemplos de cada classe (vetores de suporte) definindo planos de corte entre classes com margem máxima

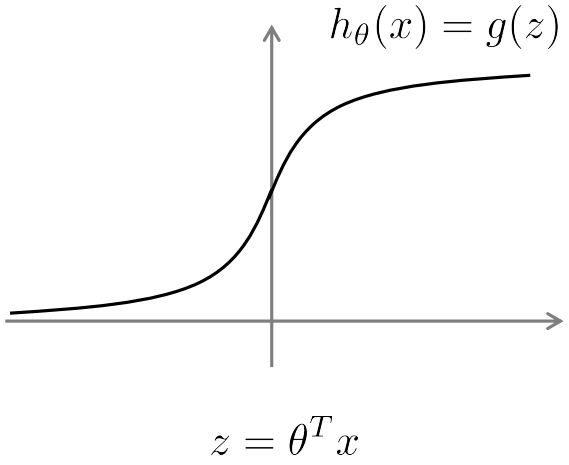
2 - utilizar uma transformação (pode ser não linear) às entradas para um espaço de características lineares, via uma função de Kernel

Classificador com margem máxima



Visão alternativa da regressão logística

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



$y = 1$ queremos $h_\theta(x) \approx 1$ $\theta^T x \gg 0$

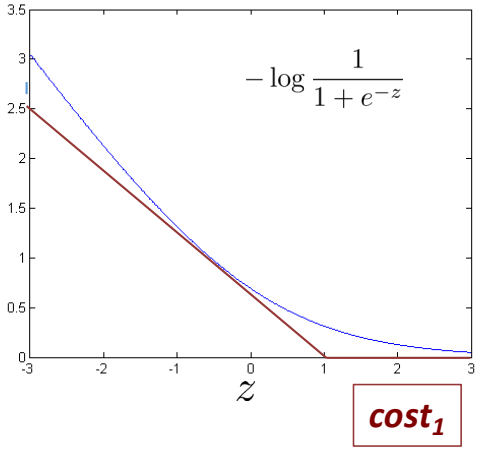
$y = 0$ queremos $h_\theta(x) \approx 0$ $\theta^T x \ll 0$

Visão alternativa da regressão logística

$$-(y \log h_\theta(x) + (1-y) \log(1 - h_\theta(x)))$$

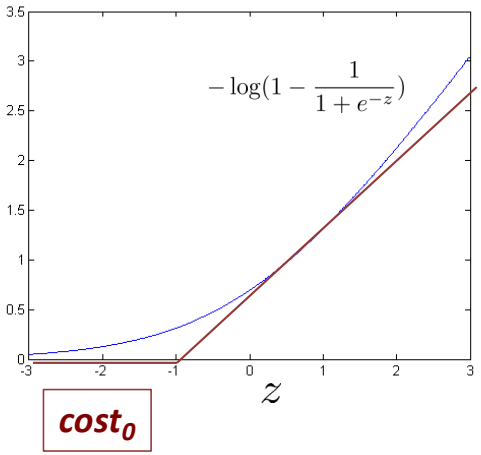
$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y) \log(1 - \frac{1}{1+e^{-\theta^T x}})$$

SE $y = 1$ (queremos $\theta^T x \gg 0$):



cost$_1$

SE $y = 0$ (queremos $\theta^T x \ll 0$):



cost$_0$

Função de custo SVMs

Soft margin SVM: usado nos casos reais onde a separação perfeita não é possível

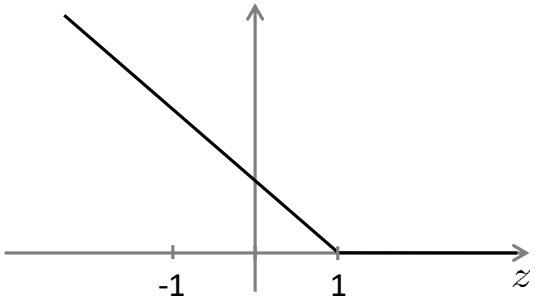
$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

Parâmetro C:
Controlo trade-off
entre erro e complexidade

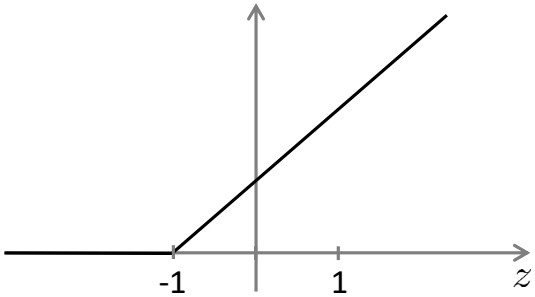
Erro

Regularização

Se y = 1 queremos: $\theta^T x \geq 1$
não apenas >= 0



Se y = 0 queremos: $\theta^T x \leq -1$
não apenas < 0



Kernels

Definem mapeamentos, possivelmente não lineares, para as variáveis (atributos) originais criando “novos” atributos

Dadas variáveis originais, calcular novos atributos usando funções de similaridade com pontos pré-definidos

Em SVMs, estes pontos pré-definidos são os próprios exemplos

Exemplo: kernel gaussiano

$$f_i = \text{similarity}(x, l^{(i)})$$

$$= \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$

SVMs em classificação

O algoritmo de treino SMO garante que é atingido sempre o hiperplano óptimo de separação entre classes.

Existem diversos tipos de kernels (linear, polinomial, gaussiano, splines), sendo o gaussiano (RBF) o mais popular;

Dois parâmetros de configuração: C > 0 (complexidade vs erro) e gamma > 0 (o parâmetro do kernel gaussiano)

- C grande e gamma pequeno favorecem overfitting
- C pequeno e gamma grande podem conduzir a underfitting

Seleção de atributos e otimização de modelos

Seleção de atributos

Em muitos casos práticos há necessidade/ diversas vantagens em **reduzir o nº de atributos de entrada** de um modelo:

- **complexidade** dos modelos aumenta com o nº atributos de entrada o que pode provocar **sobre-ajustamento**;
- dados e modelos podem ser mais facilmente **analisados** e **compreendidos**;
- eliminação de atributos redundantes ou contraditórios pode melhorar o próprio processo de aprendizagem reduzindo o **ruído**

Processo de escolha do melhor sub-conjunto de atributos de um dado conjunto é um **problema de optimização** que pode tornar-se complexo, dado o espaço alargado de procura

Seleção de atributos: estratégia

Algoritmos de **filtragem**:

- Seleção de atributos realizada antes do processo de aprendizagem, de forma independente dos classificadores
- Avaliação dos subconjuntos de atributos realizada com medidas estatísticas (e.g. Entropia/ ganho, etc)

Algoritmos **envolventes** (*wrappers*):

- Seleção dos atributos realizada em paralelo com a construção do modelo
- Avaliação dos subconjuntos de atributos realizada treinando um modelo e estimando o seu erro (usando os métodos estudados)

Algoritmos **embebidos** (*embedded*):

- Seleção dos atributos realizada junto com o processo de aprendizagem (e.g. regressão linear com regularização)

Filters

- **Não supervisionados**:
  - Calculam uma métrica a partir dos valores do atributo apenas (e.g. baseado na sua variabilidade)
  - Seleção pode ser feita por ranking (e.g. percentil) ou por valor absoluto mantendo todos os atributos abaixo/acima de um dado limite
- **Supervisionados**:
  - Baseiam-se numa métrica calculada a partir dos valores do atributo cruzados com o atributo de entrada (e.g. informação mútua)
  - Podem ser baseados em testes estatísticos univariados (e.g. t-test, ANOVA, chi^2, etc)

Wrappers: algoritmos de optimização

- Heurísticas:
  - **Forward selection**: inicia com poucos atributos (e.g. 1) e vai adicionando atributos até atingir um comportamento satisfatório
  - **Backward selection**: inicia com todos os atributos e vai removendo

- Meta-heurísticas:
  - **Algoritmos Evolucionários**
  - **Simulated Annealing**

Otimização de modelos

- De forma a optimizar o processo de construção, em muitos casos, é realizado um processo de seleção de modelos / parametrização
  - Processo de optimização minimizando-se uma medida de erro sobre um conjunto de exemplos de validação (não usados para o processo de treino) podendo usar-se processos de validação cruzada
  - Um dos objetivos é procurar modelos que minimizem função de erro
  - Podem ser usadas técnicas semelhantes à seleção de atributos (heurísticas e meta-heurísticas)
- Exemplo de um hiperparâmetro: C nos SVMs

Otimização de hiperparâmetros - algoritmos

- ***Procura em grelha (grid search)*** – considera para cada hiperparâmetro um conjunto de possíveis alternativas e explora todas as possíveis combinações de valores dos vários parâmetros escolhendo a que tiver melhor resultado
- ***Procura Aleatória (randomized search)*** – semelhante ao anterior mas apenas explora uma parte das possíveis combinações escolhidas de forma aleatória
- ***Bayesian optimization*** – métodos mais elaborados de procura estocástica que procuram explorar zonas mais promissoras do espaço de procura
- Outras metaheurísticas como computação evolucionária têm também sido usadas
- Em todos os casos, cada combinação de hiperparâmetros é avaliada treinado o modelo com esses valores e calculando uma estimativa de erro, tipicamente usando uma partição de exemplos de validação ou validação cruzada

scikit-learn

Implementação de pipelines de aprendizagem: *scikit-learn*

Biblioteca **open-source** escrita em **Python** que permite a implementação de scripts para aplicação de métodos de Aprendizagem em problemas diversos

Código eficiente; escrita usando bibliotecas *numpy*, *scipy*, *matplotib* (incluídas nas distribuição do ***anaconda***)

Inclui vários modelos e algoritmos para classificação, regressão, pré-processamento, clustering, redução de dimensionalidade, seleção de modelos e atributos

Documentação em: ***http://scikit-learn.org***

Conjuntos de dados

Para serem usados nas funções do package, conjuntos de dados devem ter uma matriz com os valores de entrada (array *numpy* com dimensões *nexs* x *nats*, onde *nexs* é nº exemplos e *nats* é nº de atributos de entrada) e valores de saída (array *numpy* com 1 dimensão de tamanho nexs)

Existem alguns conjunto de dados internos que podem ser carregados para as primeiras experiências (e.g. *iris*, *digits*, *boston-housing, diabetes*)

Datasets: carregamento

IMPLEMENTAÇÃO EM PYTHON

```
from sklearn import datasets
import numpy as np

iris = datasets.load_iris()
print(iris.data )
print(iris.target)
print(iris.data.shape)
print(np.unique(iris.target))
```

Dataset de regressão

```
from sklearn import datasets
diabetes = datasets.load_diabetes()
```

```
from sklearn import datasets

digits = datasets.load_digits()
print(digits.data)
print(digits.target)
print(digits.data.shape)

import matplotlib.pyplot as plt

plt.figure(1, figsize=(3, 3))
plt.imshow(digits.images[0], cmap=plt.cm.gray_r,
 interpolation='nearest')
plt.show()

print(digits.target[0])
```

IMPLEMENTAÇÃO EM PYTHON

Modelos de classificação: KNN

O modelo dos k-vizinhos mais próximos pode ser construído a partir da classe K*NeighborsClassifier*

```
indices = np.random.permutation(len(iris.data))
train_in = iris.data[indices[:-10]]
train_out = iris.target[indices[:-10]]
test_in = iris.data[indices[-10:]]
test_out = iris.target[indices[-10:]]

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
print(knn.fit(train_in, train_out))
print("Valores previstos: ” , knn.predict(test_in))
print("Valores reais: ” , test_out)
```

Pode definir-se o algoritmo de procura

Pode também usar-se para regressão

User guide
Section 1.6

IMPLEMENTAÇÃO EM PYTHON

Modelos de classificação: regressão logística

O modelo de regressão logística pode ser construído a partir da classe *LogisticRegression*

```
from sklearn import linear_model

logistic = linear_model.LogisticRegression(solver = "lbfgs",
                                           multi_class = "auto")
logistic = logistic.fit(train_in, train_out)
print(logistic)

print("Valores previstos: ” , logistic.predict(test_in))
print("Valores reais: ” , test_out)
```

User guide
Section 1.1

Estimação de erro

IMPLEMENTAÇÃO EM PYTHON

Divisão da amostra

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size= 0.3)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

log_model = linear_model.LogisticRegression(solver = "lbfgs",
                                            multi_class = "auto", max_iter = 1000)
log_model = log_model.fit(X_train, y_train)
print(log_model.score(X_test, y_test))
```

Estimação de erro

IMPLEMENTAÇÃO EM PYTHON

Validação cruzada

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(log_model, iris.data,
                         iris.target, cv = 5)
print(scores)
print(scores.mean())
```

Usa função de scoring definido por omissão no estimador: PECC (accuracy)

```
print("Funcao scoring: F1”)
scores_f1 = cross_val_score(log_model,
                         iris.data, y = iris.target, scoring = "f1_weighted", cv = 5)
print(scores_f1)
print(scores_f1.mean())
```

Define a função de scoring

IMPLEMENTAÇÃO EM PYTHON

Modelos de regressão linear

- O modelo de RL pode ser construído a partir da classe *LinearRegression*

User guide
Section 1.1

```
from sklearn import linear_model

Xd_train, Xd_test, yd_train, yd_test =
        train_test_split(diabetes.data, diabetes.target, test_size= 0.3)

regr_model = linear_model.LinearRegression()
regr_model = regr_model.fit(Xd_train, yd_train)
print(regr_model)
print("Valores previstos: ” , regr_model.predict(Xd_test))
print("Valores reais: ” , yd_test)
```

IMPLEMENTAÇÃO EM PYTHON

Estimação de erro - regressão

```
from sklearn.model_selection import cross_val_score
from sklearn import linear_model
regr_model = linear_model.LinearRegression()
scores_r2 = cross_val_score(regr_model,
    diabetes.data, y = diabetes.target, scoring = "r2", cv = 5)
print(scores_r2)
print(scores_r2.mean())
```

Métrica: R^2

```
scores_mse = cross_val_score(regr_model, diabetes.data, diabetes.target,
                             scoring= ”neg_mean_squared_error", cv= 5)
print (scores_mse)
scores_mad = cross_val_score(regr_model, diabetes.data, diabetes.target,
                             scoring = ”neg_mean_absolute_error", cv= 5)
print (scores_mad)
```

Métricas: MAD e MSE

IMPLEMENTAÇÃO EM PYTHON

SVMs

```
from sklearn import svm

svm_model = svm.SVC(kernel='linear', C=1)
svm_model.fit(X_train, y_train)
print(svm_model.score(X_test, y_test))
```

Classificação

Regressão

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

svm_reg = SVR(gamma = "auto")
svm_reg = svm_reg.fit(Xd_train, yd_train)
pred_svm = svm_reg.predict(Xd_test)
print("Valores previstos: " , pred_svm)
mse = mean_squared_error(yd_test, pred_svm)
print("MSE: %.1f" % mse)
```

Estimação erro - SVMs

IMPLEMENTAÇÃO EM PYTHON

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(svm_model, iris.data, iris.target, cv = 5)
print(scores)
print(scores.mean())
```

Exemplos com cross validation

```
scores_f1 = cross_val_score(svm_model,
                iris.data, y = iris.target, scoring = "f1_weighted", cv = 5)
print(scores_f1)
print(scores_f1.mean())
```

Exemplo com Leave One Out

```
from sklearn.model_selection import LeaveOneOut
loo_cv = LeaveOneOut()
scores_loo = cross_val_score(svm_model, iris.data, iris.target, cv=loo_cv)

print(scores_loo.mean())
```

Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Filtros por variabilidade – remover atributos que variam “pouco” (abaixo de um threshold definido)

```
from sklearn import svm
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import cross_val_score

sel = VarianceThreshold(threshold=20)
filt = sel.fit_transform(digits.data)
print (filt.shape)
svm_model = svm.SVC(gamma=0.001, C=100.)
scores= cross_val_score(svm_model, digits.data, digits.target, cv= 10)
print (scores.mean())
scores_vt= cross_val_score(svm_model, filt, digits.target, cv= 10)
print (scores_vt.mean())
```

Experimente variar o threshold !

Ver secção 1.13.1 do *User Guide*

Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Filtros por análise univariada (testes estatísticos) – manter atributos com valores melhores de p-value

```
from sklearn.feature_selection import SelectKBest, chi2, f_classif

filt_kb = SelectKBest(chi2, k=32).fit_transform(digits.data, digits.target)
print (filt_kb.shape)
scores_kb = cross_val_score(svm_model, filt_kb, digits.target, cv = 10)
print (scores_kb.mean())

filt_kb2 = SelectKBest(f_classif, k=32).fit_transform(digits.data, digits.target)
scores_kb2 = cross_val_score(svm_model, filt_kb2, digits.target, cv = 10)
print (scores_kb2.mean())
```

Ver secção 1.13.2 do *User Guide*

Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Wrapper: *recursive feature elimination (RFE)*

```
from sklearn.feature_selection import RFE

svm_model = svm.SVC(kernel = "linear", C=100.)

rfe = RFE(estimator=svm_model, n_features_to_select=32, step=1)

scores_rfe = cross_val_score(rfe, digits.data, digits.target, cv = 10)

print (scores_rfe.mean())
```

Ver secção 1.13.3 do *User Guide*

Seleção de modelos

IMPLEMENTAÇÃO EM PYTHON

Procura em grelha de parâmetros de SVMs (com validação cruzada na estimação do erro)

```
from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score, GridSearchCV

parameters = {'kernel':('linear', 'rbf'), 'C':[1, 3, 10, 100],
                      'gamma':[0.01, 0.001]}
svm_model_d = svm.SVC( )

opt_model_d = GridSearchCV(svm_model_d, parameters)
opt_model_d.fit(digits.data, digits.target)
print (opt_model_d.best_estimator_)
scores_gs = cross_val_score(opt_model_d, digits.data, digits.target, cv = 5)
print (scores_gs.mean())
```

Ver secção 3.2 do *User Guide*