Aprendizagem Automática

Validação Cruzada
+
Métricas de Desempenho

𝔾. Marques

Motivação:

- Para poder avaliar o desempenho de um classificador é necessário usar um conjunto de teste. O objetivo é medir o como o classificador se comporta com novos dados
  - O propósito não é saber se o modelo está bem adaptado ao conjunto de treino.
  - É aferir qual a capacidade de fazer predições acertadas (ou a sua *capacidade de generalização*)
- Ao dividir os dados em dois conjuntos, um de treino e o outro de teste, garantimos que o desempenho é estimado com base em dados nunca vistos pelo modelo.
  (reduzimos o risco de *sobre-aprendizagem*)

Desvantagens:

- Dividir os dados em 2 conjuntos (treino e teste) para medir o desempenho pode não ser uma boa estratégia.
  - Podemos “ter sorte” e só apanhar exemplos fáceis no conjunto de teste.
  - Podemos “ter azar” e só apanhar exemplos difíceis no conjunto de teste.

Validação Cruzada

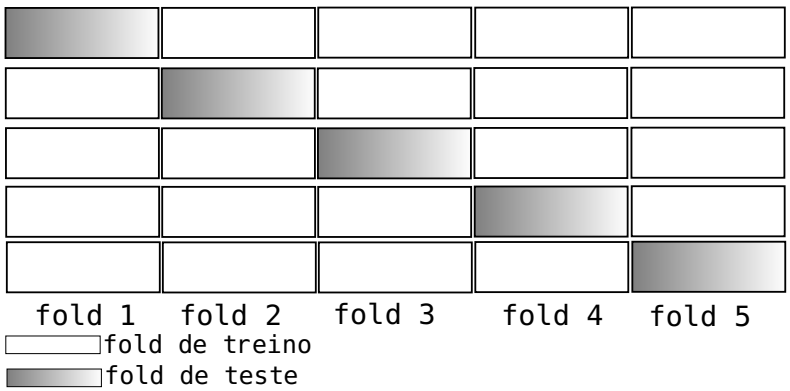
- Validação cruzada é um método estatístico de avaliação da capacidade de generalização, de uma forma mais estável e sistemático do que dividir os dados num conjunto de treino e outro de teste.
- Na validação cruzada o conjunto de dados é dividido em vários sub-conjuntos, e são treinados múltiplos modelos.

Validação Cruzada

K-fold Cross Validation:

- Uma das formas mais utilizada de validação cruzada é a K-fold cross validation (validação cruzada com K “dobras” ou “folds”).
  1. Os dados são divididos em K sub-conjuntos ou folds, cada fold contém o (aproximadamente) mesmo número de exemplos.
  2. Um dos folds é usa para teste e os restantes para treino.
     O modelo é treinado com os folds de treino e avaliado no fold de teste.
  3. Repetir o processo K vezes para todos os folds

Exemplo: validação cruzada com 5 folds:



Validação Cruzada

K-fold Cross Validation:

- Vantagens:
  - Todos os dados são testados. Não há risco de só apanharmos exemplos fáceis ou difíceis no teste. Um bom modelo necessita de ter uma boa capacidade de generalização para todos os exemplos para obter um bom desempenho em todos os folds de testes.
  - Repetir várias vezes a estimação de medidas de desempenho (tantas quanto folds) também fornece alguma informação sobre a variação do desempenho sobre novos dados.
  - Pode-se utilizar mais dados para treino. Por exemplo, ao fazer uma divisão de 10 folds estamos a usar 90% dos dados para treino, o que geralmente resulta em modelos mais precisos.
- Desvantagem:
  - A principal desvantagem é o custo computacional. Agora é necessário treinar K modelos e assim o processo é K vezes mais lento do que dividir os dados em treino e teste.

Validação Cruzada

K-fold Cross Validation:

- IMPORTANTE:
  - A validação cruzada não é uma maneira de projectar um modelo - esta estratégia de avaliação não retorna nenhum modelo.
  - A validação cruzada é uma forma de estimar qual o desempenho de um dado classificador (ou outro algoritmo).
  - Tendo só disponível uma quantidade limitada de exemplos, primeiro pode-se estimar o desempenho de um modelo recorrendo à validação cruzada K fold, mas para a classificação de novos dados tem que se treinar de novo o modelo com todos os exemplos.

Validação Cruzada `scikit-learn`

Exemplo

Divisão em treino e teste – base de dados `iris`:

Carregar dados

```
>>> from sklearn.datasets import load_iris
>>> iris=load_iris()
```

Carregar função separadora dos dados em treino e teste:

```
>>> from sklearn.model_selection import train_test_split
>>> X1,X2,t1,t2=train_test_split(iris.data,iris.target,test_size=0.3)
```

Os dados de treino e respectivas classes estão nas variáveis `X1` e `t1`. Os dados de teste estão nas variáveis `X2` e `t2`. O parâmetro `test_size=0.3` atribui 30% dos exemplos ao conjunto de teste. Para os conjuntos de treino e de teste terem a mesma distribuição de classes que todos os dados, chamar função com parâmetro: `stratify=iris.target`

Carregar classificador (discriminante logístico), treinar e classificar:

```
>>> from sklearn.linear_model import LogisticRegression
>>> logreg=LogisticRegression().fit(X1,t1)
>>> print('Prob Acertos:  %.1f'%(logreg.score(X2,t2)*100))
```

Probabilidade de acertos varia entre 80% e 100% – depende dos exemplos escolhidos para o teste pela função `train_test_split()` - inicializada com o parâmetro `random_state`.

Validação Cruzada `scikit-learn`

Exemplo

Validação cruzada K fold– base de dados `iris`:

Carregar dados

```
>>> from sklearn.datasets import load_iris
>>> iris=load_iris()
```

A maneira mais directa usar validação cruzada é chamar a função `cross_val_score()`. Esta recebe como parâmetros de entrada o classificador, os dados, as respectivas classes, e o número de folds.

```
>>> from sklearn.model_selection import cross_val_score
>>> from sklearn.linear_model import LogisticRegression
>>> logreg=LogisticRegression()
>>> scores=cross_val_score(logreg,iris.data,iris.target,cv=5)
array([1.  0.967 0.933 0.9 1.])
```

A função devolve a probabilidade de acertos em cada fold de teste. Porém não se sabe as predições individuais. Para saber as classificações atribuídas usar a função `cross_val_predict()`. Esta retorna num `numpy array` os resultados das classes preditas para os elementos de todo o conjunto.

```
>>> from sklearn.model_selection import cross_val_predict
>>> results=cross_val_predict(logreg,iris.data,iris.target,cv=5)
```

Validação Cruzada `scikit-learn`

Funções de Validação Cruzada

As funções `cross_val_score()` e `cross_val_predict()` são uma forma fácil de implementar a validação cruzada. No entanto, a divisão dos dados em folds e o treino e teste dos modelos é realizado internamente pelas funções, e não se tem directo controlo sobre estes processos. Para ter um maior controlo sobre o processo de divisão dos dados em folds, pode-se usar outras funções do sub-módulo `model_sellection` do `scikit-learn`

- `KFold`
- `StratifiedKFold`
- `ShuffeSplit`
- `StratifiedShuffeSplit`

Ver igualmente as funções:

- `LeaveOneOut`
  Conjunto de teste composto por apenas 1 único ponto
- `LeavePOut`
  Conjunto de teste composto por P pontos

Validação Cruzada `scikit-learn`

Funções de Validação Cruzada

- `KFold`

Para ter mais controlo de como os dados são divididos em folds, pode-se usar o *"K-fold splitter"*.

```
>>> from sklearn.model_selection import KFold
>>> kfold=KFold()
```

A instância `kfold` é passada às funções `cross_val_score()` e `cross_val_predict()` (parâmetro `cv=kfold`)

ATENÇÃO: se não especificado, a divisão é feita sequencialmente
⟶ pode ser problemático para dados que estejam ordenados por classe:

```
>>> kfold=KFold(n_splits=4)
>>> cross_val_score(logreg, iris.data, iris.target, cv=kfold)
 array([ 1.  , 0.63157895, 0.75675676, 0.10810811])
>>> kfold=KFold(n_splits=3)
>>> cross_val_score(logreg, iris.data, iris.target, cv=kfold)
array([ 0., 0., 0.])
```

Pode-se contornar este problema especificando para baralhar os dados antes de dividir (parâmetro `shuffle=True`). Neste caso também se pode inicializar o gerador de números aleatórios para obter sempre a mesma divisão:

```
>>> kfold=KFold(n_splits=3,shuffle=True,random_state=0)
>>> cross_val_score(logreg, iris.data, iris.target, cv=kfold)
array([ 0.9, 0.96, 0.96])
```

Validação Cruzada `scikit-learn`

Funções de Validação Cruzada

- `StratifiedKFold`

  Este é outro *"K-fold splitter"* que divide os dados em K folds.

  - A proporção do número de exemplos por classe na base de dados é mantida (aproximadamente) em cada fold.
  - Muito útil para dados que com diferentes percentagens de exemplos por classe, como é geralmente o caso em situações de detecção onde o número de exemplos negativos é significativamente superior aos exemplos positivos.

```
>>> from sklearn.model_selection import StratifiedKFold
>>> kfold=StratifiedKFold(n_splits=3)
>>> cross_val_score(logreg, iris.data, iris.target, cv=kfold)
array([ 0.96078431, 0.92156863, 0.95833333])
```

Validação Cruzada `scikit-learn`

Funções de Validação Cruzada

- `ShuffleSplit`

`ShuffleSplit` é outra forma de aplicar a validação cruzada. Os dados são divididos de forma aleatória em dois, parte para o treino e outra para o teste, e este processo é repetido K vezes. Note que pode-se só usar parte dos dados para treino e teste (as percentagens relativas aos dados de treino e teste não têm que somar 100%).

```
>>> from sklearn.model_selection import ShuffleSplit
>>> kfold=ShuffleSplit(n_splits=5,train_size=0.5,test_size=0.3)
>>> cross_val_score(logreg, iris.data, iris.target, cv=kfold)
array([ 0.82222222, 0.95555556, 0.82222222, 0.97777778, 0.95555556])
```

- `StratifiedShuffleSplit`

`StratifiedShuffleSplit` é em tudo semelhante à função `ShuffleSplit`, excepto que as probabilidades a priori das classes são mantidas nos conjuntos de treino e teste.

```
>>> from sklearn.model_selection import StratifiedShuffleSplit
>>>
kfold=StratifiedShuffleSplit(n_splits=5,train_size=0.5,test_size=0.3)
>>> cross_val_score(logreg, iris.data, iris.target, cv=kfold)
array([ 0.93333333, 0.95555556, 0.95555556, 0.97777778, 0.97777778])
```

Conjunto de Validação

Nos exemplos anteriores, falou-se de dois tipos de conjuntos: o conjunto de treino que serve para treinar o modelo, e o conjunto de teste que serve para avaliar o modelo. Porém a maioria dos modelos de classificação ou regressão são treinados ajustando uma série de **hiper-parâmetros** cujo o valor também é preciso achar (*e.g.* por tentativa e erro) durante a fase de treino. Ajustar os hiper-parâmetros que resultem no melhor desempenho no conjunto de teste não é uma boa maneira de aferir o desempenho, visto que o conjunto de teste já foi usado.

A solução passo por definir mais um conjunto: **o conjunto de validação**. Assim, o conjunto de treino é usado para construir o modelo, o conjunto de validação para selecionar os hiper-parâmetros, e o de teste para avaliação.

NOTAS:

- Depois de achar os melhores hiper-parâmetros, deve-se re-treinar o modelo com os conjuntos de treino e validação
- Atenção que a nomenclatura é um pouco enganadora e muitas vezes os termos “conjunto de validação” e “conjunto de testes” são usados como sinónimos.

Métricas de Desempenho

- Em problemas de classificação multi-classe ou binários, as matrizes de confusão não normalizadas contêm toda a informação necessária para avaliar o desempenho de classificadores.
- A avaliação deve ser feita com exemplos não usados para treino. Caso se use estratégias de validação cruzada, obtém-se vários conjuntos de teste e consequentemente várias matrizes de confusão - uma para cada fold ou conjunto de teste.
  - Para o caso da validação cruzada K-Fold, pode-se concatenar os resultados de cada fold, e obter uma única matriz confusão para todo o conjunto de exemplos.
  - Pode também ser importante saber os resultado de cada fold individualmente para obter medidas como desempenhos mínimos, máximos, etc.
- Em problemas de classificação binária existem várias métricas de desempenho além da probabilidade total erro. Estas também podem ser usadas em problemas de multi-classe, adotando uma estratégia de escolher cada classe versos as restantes. Assim um problema de c classes é decomposto em c problemas de classificação binária.
- Em problemas de classificação binária é frequente ajustar o processo de classificação, podendo-se visualizar os resultados em gráficos como as curvas ROC ou as curvas de precision/recall.

Métricas de Desempenho `scikit-learn`

Matriz de Confusão

A matriz de confusão é obtida com a função `confusion_matrix()` do sub-módulo `sklearn.metrics`.

Exemplo: Base de dados `iris`
Dividir conjunto em treino e teste e avaliar o classificador Naïve Bayes com funções de densidade gaussiana.

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import confusion_matrix
>>> from sklearn.naive_bayes import GaussianNB
>>> iris=load_iris()
>>> X1,X2,t1,t2=train_test_split(iris.data,iris.target,test_size=1./3,stratify=iris.target)
>>> NBgau=GaussianNB(prior=None).fit(X1,t1)
>>> results=NBgau.predict(X2)
>>> print('Matriz de confusão:  ',confusion_matrix(t2,results))
>>> print('Proabilidade de acertos:  '%NBgau.score(X2,t2))
```

Matriz de confusão: $\mathbf{P} = \begin{bmatrix} 17 & 0 & 0 \\ 0 & 16 & 1 \\ 0 & 1 & 15 \end{bmatrix}$

Prob. de acertos: 96%

Métricas de Desempenho `scikit-learn`

Matriz de Confusão

A matriz de confusão é obtida com a função `confusion_matrix()` do sub-módulo `sklearn.metrics`.

Exemplo: Base de dados `iris`
Dividir conjunto em K-Folds e teste e avaliar o classificador Naïve Bayes com funções de densidade gaussiana.

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import StratifiedKFold,cross_val_score,cross_val_predict
>>> from sklearn.metrics import confusion_matrix
>>> from sklearn.naive_bayes import GaussianNB
>>> iris=load_iris()
>>> kfold=StratifiedKFold(n_splits=3)
>>> NBgau=GaussianNB(prior=None)
>>> scores=cross_val_score(NBgau,iris.data,iris.target,cv=kfold)
>>> results=cross_val_predict(NBgau,iris.data,iris.target,cv=kfold)
>>> print('Matriz de confusão:  ',confusion_matrix(iris.target,results))
>>> print('Proabilidade de acertos:  '%np.mean(scores))
```

Matriz de confusão: $\mathbf{P} = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 44 & 6 \\ 0 & 4 & 46 \end{bmatrix}$

Acertos por fold: [94.4%,94.1%,100%]
Prob. de acertos: 96.2%

Métricas de Desempenho `scikit-learn`

Matriz de Confusão

A matriz de confusão é obtida com a função `confusion_matrix()` do sub-módulo `sklearn.metrics`.

Exemplo: Base de dados `iris`
Por vezes é necessário o desempenho do classificador em cada fold (matriz de confusão + probabilidade de acertos). Os iteradores `KFold` e `StratifiedKFold` têm associado a função `split()` que permite obter os índices dos exemplos de treino e de teste para cada fold. Os seguintes comandos permitem estimar o desempenho em cada fold:

```
>>> from sklearn.metrics import accuracy_score
>>> iris=load_iris()
>>> NBgau=GaussianNB(prior=None)
>>> kfold=StratifiedKFold(n_splits=3)
>>> for iTrain,iTest in kfold.split(iris.data,iris.target,None):
>>>     NBgau.fit(iris.data[iTrain,:],iris.target[iTrain])
>>>     results=NBgau.predict(iris.data[iTest,:])
>>>     print(confusion_matrix(iris.target[iTest],results))
>>>     print('Acertos:  %f'%accuracy_score(iris.target[iTest],results))
```

Matrize de confusão: $\mathbf{P}_1 = \begin{bmatrix} 17 & 0 & 0 \\ 0 & 14 & 3 \\ 0 & 1 & 16 \end{bmatrix}$ $\mathbf{P}_2 = \begin{bmatrix} 17 & 0 & 0 \\ 0 & 14 & 3 \\ 0 & 2 & 15 \end{bmatrix}$ $\mathbf{P}_3 = \begin{bmatrix} 16 & 0 & 0 \\ 0 & 16 & 0 \\ 0 & 1 & 15 \end{bmatrix}$

Acertos por fold: [92.2%,90.2%,97.9%]

Métricas de Desempenho `scikit-learn`

Matriz de Confusão

Exemplo: Base de dados `digits` do `sklearn`

```
>>> from sklearn.datasets import load_digits
>>> D=load_digits()
>>> print(D.DESCR)
Optical Recognition of Handwritten Digits Data Set
==================================================
Data Set Characteristics:
    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998
This is a copy of the test set of the UCI ML hand-written digits datasets
http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.
Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.
```

Métricas de Desempenho `scikit-learn`

Matriz de Confusão

Exemplo: Base de dados `digits` do `sklearn`

Dividir conjunto em K-folds e avaliar o classificador Naïve Bayes com funções de densidade gaussiana.

```
>>> from sklearn.datasets import load_digits
>>> from sklearn.model_selection import StratifiedKFold,cross_val_predict,cross_val_score
>>> from sklearn.metrics import confusion_matrix
>>> from sklearn.naive_bayes import GaussianNB
>>> D=load_digits()
>>> NBgau=GaussianNB(prior=None)
>>> kfold=StratifiedKFold(n_splits=10)
>>> scores=cross_val_score(NBgau,D.data,D.target,cv=kfold)
>>> results=cross_val_predict(NBgau,D.data,D.target,cv=kfold)
```

Matriz de confusão:

$$\begin{bmatrix} 174 & 0 & 0 & 0 & 2 & 1 & 0 & 1 & 0 & 0 \\ 0 & 147 & 1 & 0 & 0 & 0 & 4 & 5 & 19 & 6 \\ 0 & 14 & 112 & 1 & 1 & 1 & 1 & 0 & 47 & 0 \\ 0 & 2 & 3 & 134 & 0 & 8 & 0 & 8 & 24 & 4 \\ 0 & 1 & 1 & 0 & 150 & 2 & 3 & 21 & 3 & 0 \\ 0 & 1 & 0 & 3 & 1 & 165 & 1 & 6 & 4 & 1 \\ 0 & 0 & 1 & 0 & 2 & 2 & 176 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 175 & 0 & 1 \\ 0 & 12 & 0 & 2 & 0 & 3 & 0 & 10 & 147 & 0 \\ 1 & 7 & 0 & 5 & 3 & 3 & 1 & 18 & 19 & 123 \end{bmatrix}$$

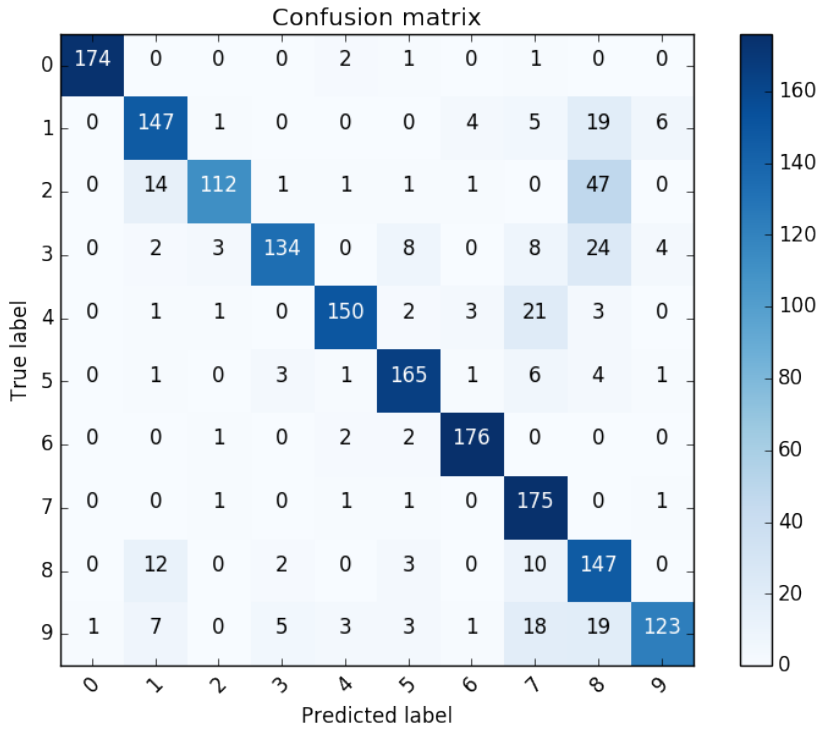
Acertos por fold: [0.9, 0.85, 0.85, 0.65, 0.70, 0.75, 0.60, 0.70, 0.90, 1.00]
Prob. total de acertos: 79.0%

Métricas de Desempenho `scikit-learn`

Matriz de Confusão

Exemplo: Base de dados `digits` do `sklearn`

Pode ser útil visualizar a matriz de confusão como uma imagem, nomeadamente quando o número de classes é elevado. A imagem da matriz de confusão foi obtida com a função `ConfusionMatrixDisplay` (ver a página do manual do `scikit-learn` sobre model selection/confusion matrix).



Métricas de Desempenho `scikit-learn`

Métricas de Classificação Binária

- Existem várias métricas de desempenho para problemas de classificação binária. Em diversos contextos, como em sistemas de deteção, identificação, ou recolha de informação, é comum usar a *precision* e o *recall* para avaliar o desempenho.

precision Percentagem de classificações positivas corretas $\left(\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}\right)$

recall Percentagem dos positivos classificados corretamente $\left(\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}\right)$

	$\hat{\omega}_\mathrm{p}$	$\hat{\omega}_\mathrm{n}$
ω_p	**T**rue **P**ositives	**F**alse **N**egatives
ω_n	**F**alse **P**ositives	**T**rue **N**egatives

A *precision* é uma mediada de avaliação usada quando o objetivo é reduzir o número de falsos positivos. Por outro lado, o *recall* é usado quando o objetivo é reduzir os falsos negativos. É fácil de projetar classificadores que obtenham bons resultados numa destas duas métricas, mas já não é tão simples obter bons resultados em ambas as métricas. O *f-score* sumariza estas duas métricas.

f-score Média harmónica entre a *precision* e *recall* $\left(2\frac{\mathrm{precision}\times\mathrm{recall}}{\mathrm{precision}+\mathrm{recall}}\right)$

Métricas de Desempenho `scikit-learn`

Métricas de Classificação Binária

Exemplo: Base de dados `breast_cancer`

```
>>> from sklearn.datasets import load_breast_cancer

Breast Cancer Wisconsin (Diagnostic) Database
=============================================Data Set Characteristics:
    :Number of Instances: 569
    :Number of Attributes: 30 numeric, predictive attributes and the class
    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign
    :Missing Attribute Values: None
    :Class Distribution: 212 - Malignant, 357 - Benign
```

Métricas de Desempenho `scikit-learn`

Métricas de Classificação Binária

Exemplo: Base de dados `breast_cancer`
Esta base de dados (Breast Cancer Wisconsin Diagnostic Dataset) é relativa ao problema de deteção de cancro da mama. Baseado num conjunto de 30 características extraídas de imagens de tecidos obtidos por punção aspirativa, o objetivo é prever se o tecido é ou não maligno. De notar que se considerarmos a classe dos positivos como "maligno", o que se quer evitar são erros de deteção (pretende-se evitar os falsos negativos).

Os seguintes comandos treinam um discriminante logístico e obtêm as previsões para o conjunto teste.

```
>>> BC=load_breast_cancer()
>>> X1,X2,t1,t2=train_test_split(BC.data,BC.target,test_size=.5,stratify=BC.target)
>>> logR=LogisticRegression().fit(X1,t1)
>>> results=logR.predict(X2)
>>> scores=logR.score(X2,t2)
```

Métricas de Desempenho `scikit-learn`

Métricas de Classificação Binária

Exemplo: Base de dados `breast_cancer`
Esta base de dados (Breast Cancer Wisconsin Diagnostic Dataset) é relativa ao problema de deteção de cancro da mama. Baseado num conjunto de 30 características extraídas de imagens de tecidos obtidos por punção aspirativa, o objetivo é prever se o tecido é ou não maligno. De notar que se considerarmos a classe dos positivos como "maligno", o que se quer evitar são erros de deteção (pretende-se evitar os falsos negativos).

Resultados:

Matriz de confusão: $\begin{bmatrix} 96 & 10 \\ 7 & 172 \end{bmatrix}$

Probabilidade total de acertos: 0.9404

Métricas Binárias:

$$\textit{precision} = \frac{96}{96+7} = 0.9320$$

$$\textit{recall} = \frac{96}{96+10} = 0.9056$$

$$\textit{f-score} = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = 0.9187$$

Métricas de Desempenho `scikit-learn`

Métricas de Classificação Binária

Exemplo: Base de dados `breast_cancer`
Esta base de dados (Breast Cancer Wisconsin Diagnostic Dataset) é relativa ao problema de deteção de cancro da mama. Baseado num conjunto de 30 características extraídas de imagens de tecidos obtidos por punção aspirativa, o objetivo é prever se o tecido é ou não maligno. De notar que se considerarmos a classe dos positivos como "maligno", o que se quer evitar são erros de deteção (pretende-se evitar os falsos negativos).

Resultados:

Matriz de confusão: $\begin{bmatrix} 96 & 10 \\ 7 & 172 \end{bmatrix}$

Probabilidade total de acertos: 0.9404

Métricas Binárias:

Para obter o sumário das medidas de avaliação binária, pode-se usar a função `classification_report()` do módulo `sklearn.metrics`

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(BC.target,results,target_names=['maligno','benigno']))
```

```
             precision    recall  f1-score   support

    maligno       0.93      0.91      0.92       106
    benigno       0.95      0.96      0.95       179

avg / total       0.94      0.94      0.94       285
```

Métricas de Desempenho `scikit-learn`

Métricas de Classificação Binária

Exemplo: Base de dados `breast_cancer`
Esta base de dados (Breast Cancer Wisconsin Diagnostic Dataset) é relativa ao problema de deteção de cancro da mama. Baseado num conjunto de 30 características extraídas de imagens de tecidos obtidos por punção aspirativa, o objetivo é prever se o tecido é ou não maligno. De notar que se considerarmos a classe dos positivos como "maligno", o que se quer evitar são erros de deteção (pretende-se evitar os falsos negativos).

Resultados:

Matriz de confusão: $\begin{bmatrix} 96 & 10 \\ 7 & 172 \end{bmatrix}$

Probabilidade total de acertos: 0.9404

Métricas Binárias:

Para obter o sumário das medidas de avaliação binária, pode-se usar a função `classification_report()` do módulo `sklearn.metrics`

- Atenção: a função `classification_report()` calcula as métricas binárias para todas a classes, e pode igualmente ser usada para problemas de classificação multi-classe. Esta função assume que os exemplos de uma das classes são os exemplos positivos e os exemplos das restantes são os negativos. Este processo é repetido para todas as classes.

Métricas de Desempenho `scikit-learn`

Calibração de Modelos e Curvas de Desempenhos

- No exemplo anterior sobre cancro da mama, o discriminante logístico obteve 10 falhas de deteção (false negatives) e gerou 7 falsos alarmes (false positives). Neste exemplo é preferível reduzir o número de falhas de deteção, em detrimento de um aumento dos falsos alarmes. Para tal, pode-se ajustar os limiares de decisão de muitos modelos de classificação, incluindo o discriminante logístico, para obter os resultados desejados.
- A maioria dos modelos de classificação do `scikit-learn` têm associado uma destas duas funções de decisão: `decision_function()` ou `predict_proba()`. O processo de classificação é obtido através da aplicação um limiar fixo à saída da função de decisão. Este limiar é 0 para o caso de `decision_function()` e 0.5 para as saídas de `predict_proba()`.
- Para obter diferentes resultados de classificação, pode-se variar o limiar de decisão destas funções.
- Ao variar o limiar, obtém-se curvas de desempenho que podem ajudar no processo de calibração dos modelos de classificação.

Métricas de Desempenho `scikit-learn`

Calibração de Modelos e Curvas de Desempenhos

Exemplo: Base de dados `breast_cancer`
Os seguintes comandos treinam um discriminante logístico e obtêm as previsões para o conjunto teste.

```
>>> BC=load_breast_cancer()
>>> X1,X2,t1,t2=train_test_split(BC.data,BC.target,test_size=.5,stratify=BC.target)
>>> logR=LogisticRegression().fit(X1,t1)
```

1. Para este classificador pode-se variar o limiar de decisão através da função `decision_function()`. Para o caso de duas classes, a função retorna um array de dimensão igual ao número de exemplos do conjunto de teste. O processo de classificação equivale a comparar os valores do array a um dado limiar de decisão. O limiar de omissão é 0; para obter menos falsos negativos, deve-se aumentar o valor do limiar.

```
>>> df=logR.decision_function(X2)
>>> print(confusion_matrix(t2,df>1.0))
```

Matriz de confusão: $\begin{bmatrix} 101 & 5 \\ 11 & 168 \end{bmatrix}$

Probabilidade total de acertos: $\frac{269}{285} = 0.9439$

Métricas de Desempenho `scikit-learn`

Calibração de Modelos e Curvas de Desempenhos

Exemplo: Base de dados `breast_cancer`
Os seguintes comandos treinam um discriminante logístico e obtêm as previsões para o conjunto teste.

```
>>> BC=load_breast_cancer()
>>> X1,X2,t1,t2=train_test_split(BC.data,BC.target,test_size=.5,stratify=BC.target)
>>> logR=LogisticRegression().fit(X1,t1)
```

2. Para este classificador pode-se variar o limiar de decisão através da função `predict_proba()`. A função retorna um array de $N \times c$, onde N é o número de exemplos do conjunto de teste, e c o número de classes (2 neste caso). Para cada exemplo (linha), a soma dos valores é igual a 1.0. Através de um limiar, o valor das probabilidades podem ser usado para classificar. O limiar de omissão é 0.5; para obter menos falsos negativos, deve-se aumentar o valor do limiar.

```
>>> pp=logR.predict_proba(X2)[:,1] # 2ª coluna - classe dos "1s"
>>> print(confusion_matrix(t2,pp>0.74))
```

Matriz de confusão: $\begin{bmatrix} 101 & 5 \\ 11 & 168 \end{bmatrix}$

Probabilidade total de acertos: $\frac{269}{285} = 0.9439$

Métricas de Desempenho `scikit-learn`

Curvas *precision*/*recall* e Curvas ROC

- Mudar o limiar de decisão é uma forma de ajustar a *precision* e o *recall* para um dado classificador.
- O limiar deve ser adequado a cada problema. Por exemplo, pode-se querer uma taxa de falsos negativos menor que um dado valor (*recall* alto), ou reduzir o número de falsos alarmes (*precision* alto).
- Ao projetar um modelo de classificação, nem sempre é fácil estimar qual o limiar ótimo.
- Para ter uma melhor perceção do problema, é desejável ver os resultados para todos os limiares possíveis.
- Isto é possível através da visualização de curvas *precision*/*recall* e de curvas ROC.

Métricas de Desempenho `scikit-learn`

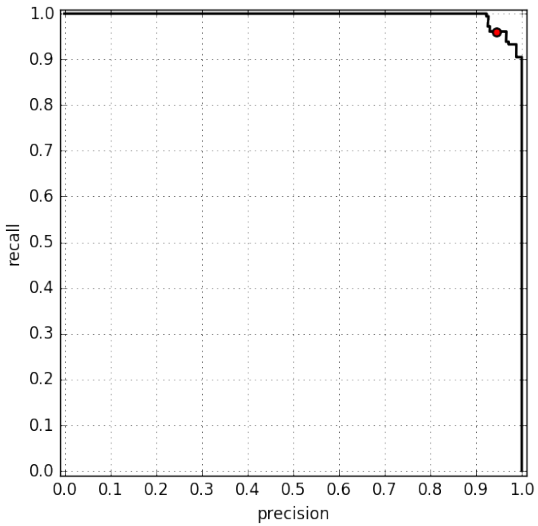
Curvas *precision*/*recall*

Exemplo: Base de dados `breast_cancer`
Os seguintes comandos treinam um discriminante logístico.

```
>>> BC=load_breast_cancer()
>>> X1,X2,t1,t2=train_test_split(BC.data,BC.target,test_size=.5,stratify=BC.target)
>>> logR=LogisticRegression().fit(X1,t1)
```

A função para estimar a curva de *precision*/*recall* está no módulo `sklearn.metrics`.

```
>>> from sklearn.metrics import precision_recall_curve
>>> t2logR=logR.decision_function(X2)
>>> prec,rec,limiar=precision_recall_curve(t2,t2logR)
>>> plt.plot(prec,rec)
```



- Cada ponto nesta curva corresponde aos resultados de classificação para um dado limiar.
- O ponto a vermelho é o limiar de omissão (zero).
- Quanto mais próximo estiver a curva do canto superior direito, melhor é o resultado da classificação.
- Diferentes classificadores podem ter um bom funcionamento em diferentes partes da curva.

Métricas de Desempenho `scikit-learn`

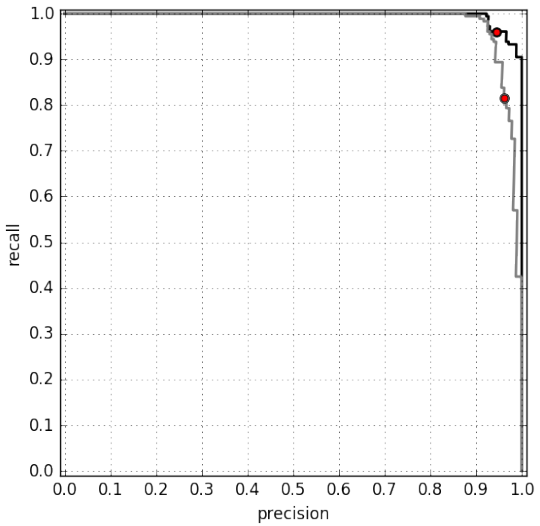
Curvas *precision/recall*

Exemplo: Base de dados `breast_cancer`
Os seguintes comandos treinam um discriminante logístico.

```
>>> BC=load_breast_cancer()
>>> X1,X2,t1,t2=train_test_split(BC.data,BC.target,test_size=.5,stratify=BC.target)
>>> logR=LogisticRegression().fit(X1,t1)
```

A função para estimar a curva de *precision/recall* está no módulo `sklearn.metrics`.

```
>>> from sklearn.metrics import precision_recall_curve
>>> t2logR=logR.decision_function(X2)
>>> prec,rec,limiar=precision_recall_curve(t2,t2logR)
>>> plt.plot(prec,rec)
```



- Resultados usando outro classificador (máquina de suporte vetorial linear) - curva a cinzento

```
>>> from sklearn.svm import LinearSVC
>>> lsvm=LinearSVC().fit(X1,t1)
>>> t2svm=lsvm.decision_function(X2)
>>> p,r,l=precision_recall_curve(t2,t2svm)
```

Métricas de Desempenho `scikit-learn`

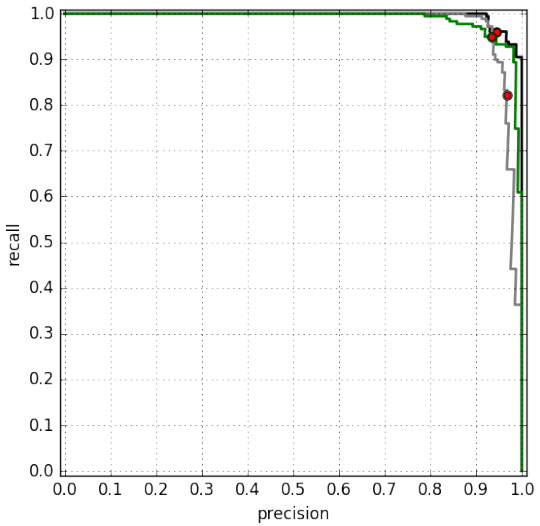
Curvas *precision/recall*

Exemplo: Base de dados `breast_cancer`
Os seguintes comandos treinam um discriminante logístico.

```
>>> BC=load_breast_cancer()
>>> X1,X2,t1,t2=train_test_split(BC.data,BC.target,test_size=.5,stratify=BC.target)
>>> logR=LogisticRegression().fit(X1,t1)
```

A função para estimar a curva de *precision/recall* está no módulo `sklearn.metrics`.

```
>>> from sklearn.metrics import precision_recall_curve
>>> t2logR=logR.decision_function(X2)
>>> prec,rec,limiar=precision_recall_curve(t2,t2logR)
>>> plt.plot(prec,rec)
```



- Resultados usando outro classificador (naïve Bayes) - curva a verde

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gNB=GaussianNB().fit(X1,t1)
>>> t2gNB=gNB.predict_proba(X2)[:,1]
>>> p,r,l=precision_recall_curve(t2,t2gNB)
```

Métricas de Desempenho `scikit-learn`

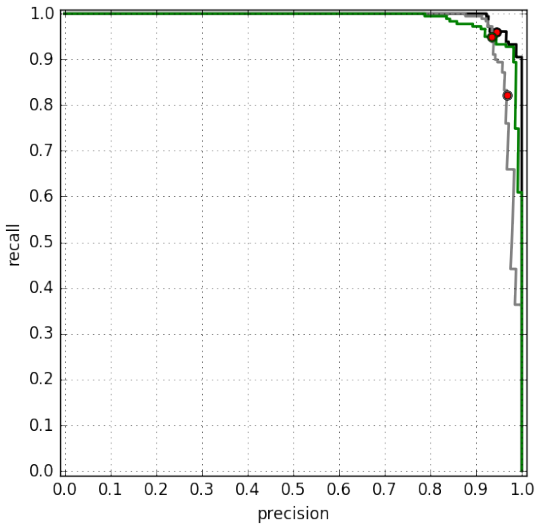
Curvas *precision/recall*

Exemplo: Base de dados `breast_cancer`
Os seguintes comandos treinam um discriminante logístico.

```
>>> BC=load_breast_cancer()
>>> X1,X2,t1,t2=train_test_split(BC.data,BC.target,test_size=.5,stratify=BC.target)
>>> logR=LogisticRegression().fit(X1,t1)
```

A função para estimar a curva de *precision/recall* está no módulo `sklearn.metrics`.

```
>>> from sklearn.metrics import precision_recall_curve
>>> t2logR=logR.decision_function(X2)
>>> prec,rec,limiar=precision_recall_curve(t2,t2logR)
>>> plt.plot(prec,rec)
```



- Uma maneira de resumir estas curvas num só valor é calcular a área debaixo da curva. A área da curva pode ser obtida usando a função `average_precision_score()`.

```
>>> from sklearn.metrics import average_precision_score
>>> ap_logR=average_precision_score(t2,t2logR)
>>> ap_svm=average_precision_score(t2,t2svm)
>>> ap_gNB=average_precision_score(t2,t2gNB)
```

Áreas das curvas: 0.9959 - discriminante logístico; 0.9793 - SVM; 0.9890 Naïve Bayes

Métricas de Desempenho `scikit-learn`

Curvas ROC

- As curvas ROC são outra maneira de visualizar os resultados de classificação para diferentes limiares de decsisão.
- A curva considera todos os possíveis limiares de decisão para um dado classificador, e mostra a taxa de falsos positivos (falsos alarmes) versus a taxa de verdadeiros positivos (recall).
- O ponto operacional ótimo da curva ROC é o canto superior esquerdo.
- Tal como as curvas *precision*/*recall*, também se pode calcular a área da curva ROC. Este valor é conhecido com AUC (Area Under the ROC Curve).

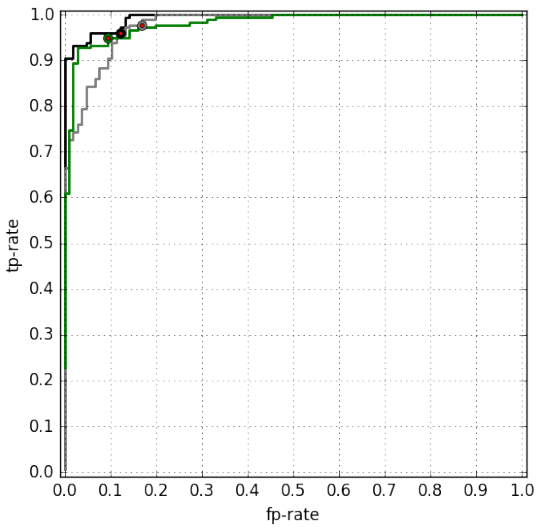
Métricas de Desempenho `scikit-learn`

Curvas ROC

Exemplo: Base de dados `breast_cancer`

- As curvas ROC são obtidas com a função `roc_curve()` do módulo `sklearn.metrics`.
- A área da curva ROC (AUC) é calculada com a função `roc_auc_score()`.

```
>>> from sklearn.metrics import roc_curve,roc_auc_score
>>> fpr,tpr,limiar=roc_curve(t2,t2logR)
>>> auc_logR=roc_auc_score(t2,t2logR)
```



- Cada ponto nesta curva corresponde aos resultados de classificação para um dado limiar.
- O ponto a vermelho é o limiar de omissão (zero).
- Quanto mais próximo estiver a curva do canto superior esquerdo, melhor é o resultado da classificação.

Áreas das curvas: 0.9928 discriminante logístico; 0.9776 SVM; 0.9811 Naïve Bayes.