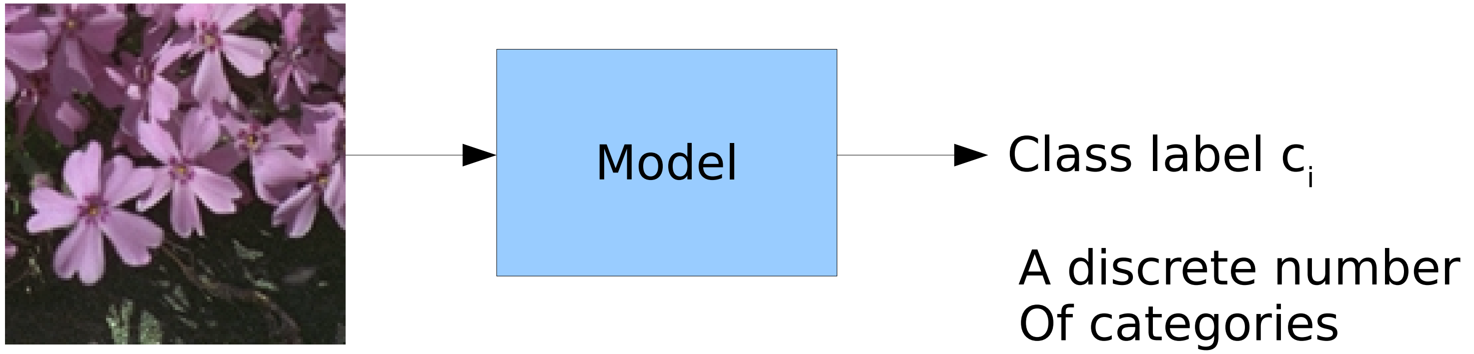
International Master in Computer Vision

Fundamentals of machine learning for computer vision

Eva Cernadas

Contents

Supervised classification



- The model is trained with an examples or images (a set of input patterns and desired outputs).
- How to select the best model?
- How to evaluate the model quality?

Nomenclature

- Training pattern is a vector : $\mathbf{x}_i=(x_{i1}...x_{in})$: n inputs
- Test pattern: $\mathbf{x}$ (not included as a training pattern)
- Desired output (prediction), y_i , for the training pattern $\mathbf{x}_i$
- Classification: C classes: $y_i \in \{1...C\}$: the classifier assigns the class y_i to the pattern $\mathbf{x}_i$

Evaluation methodology (I)

- Training set: examples and outputs: $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$, N is the number of training patterns.
- $\mathbf{x}_i$: n-dimensional pattern; y_i : output value.
- z_i : output predicted by the classification algorithm.
- Training: provides a trained model, calculating the trainable parameters (different for each model type).
- Validation or test: the trained model is used to predict the class on a data set different to the training set.

Evaluation methodology (II)

- The prediction quality must be evaluated. There are different performance measurements for classification.
- Important issues in the evaluation:

1) The trained model is optimized to predict the class of the training patterns.
2) Do not evaluate the prediction quality using training patterns, because it will be very optimistic, not realistic.
3) The training set should be big and representative of the problem under consideration.

K-fold cross-validation

- K is the number of folds or partitions (K=4, 5, 10 usually).
- Divide the available data into K disjoint partitions.
- Train the model with K-1 partitions.
- Validate with the excluded partition.
- Repeat the process K times, each time excluding a different fold to evaluate the prediction quality of the model.

	Trial 1	Trial 2	Trial 3	Trial 4
Training	1 2 3	2 3 4	3 4 1	4 1 2
Test	4	1	2	3

K-fold cross-validation

- The partitions should be ramdonly generated:

1) Shuffle the N patterns indices.
2) Divide the shuffled patterns into K partitions:
    a) All partitions must have patterns of all classes.
    b) Keep the relative class populations.
3) Assign each partition to training or test in the different trials.

K-fold cross-validation

- **Each trial**: training + test.
- Higher K increases the number of trials to repeat the training+test loop and time raises.
- For **large-scale problems** (many patterns), K should be low to avoid large times.
- For **small-sample problems** (very few patterns), use K=N (LOOCV, leave-one-out cross-validation). We need N trials and, in each trial, exclude one pattern and train with the remaining patterns.

Data pre-processing

- Standardization: zero mean and standard desviation one:

$$x_{ij}'=\frac{x_{ij}-m_j}{d_j} \qquad m_j=\frac{1}{N}\sum_{i=1}^{N}x_{ij} \qquad d_j=\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_{ij}-m_j\right)^2}$$

- Equalizes all the inputs with equal range (about ±2 around 0).
- Means and standard desviation for each input must be computed using only the trained patterns.
- The test patterns must be processed using the previous calculated means and standard desviations.
- If the input x_i is discrete with *M* values: convert it into *M dummy variables* y_j: if x_i takes the *k*-th value, then $y_k=1$ and $y_l=0$ for $l \neq k$.

Quality measures for classification (I)

- Confusion matrix: C_{ij}= number of patterns of class *i* assigned to class *j*.

Class label		Predicted	
		Class 1	Class 2
True	Class 1	**C11**	C12
	Class 2	C21	**C22**

- Classification errors: outside the main diagonal.
- **Accuracy:** $Ac(\%)=\frac{100\sum_{i=1}^{C} C_{ii}}{\sum_{i=1}^{C}\sum_{j=1}^{C} C_{ij}}$

Acc∈[0,100]: very sensitive to imbalance between classes

Kappa∈[-100,100]

- **Kappa:** $\kappa(\%)=100\frac{a-e}{s-e}, a=\sum_{i=1}^{C} C_{ii}, e=\frac{1}{s}\sum_{i=1}^{C}\left(\sum_{j=1}^{C} C_{ij}\right)\left(\sum_{k=1}^{C} C_{ki}\right), s=\sum_{i=1}^{C}\sum_{j=1}^{C} C_{ij}$
- Both accuracy and Cohen kappa can be applied to multiclass problems.

Quality measures for classification (II)

- Best measurement: kappa (%). The values can be understood as:

1) Kappa≤20%: poor agreement between true and predicted class labels

2) 21%≤Kappa<40%: weak agreement

3) 41%≤Kappa≤60%: moderated

4) 61%≤Kappa≤80%: good

5) 81%≤Kappa≤100%: very good

Source: "The Measurement of Observer Agreement for Categorical Data", J. Landis and H. G. Koch, *Biometrics*, No. 1, pp. 159-174 (1977)

ROC curves for binary problems

- In a two-class detection problem, we assume that: class 1 is negative (N) and class 2 is positive (P).

  T=True, F=False

	C1	C2
C1	**TN**	FP
C2	FN	**TP**

- **Sensitivity** or **recall**: probability of pattern of class 2 will be classified as class 2.

$$Se = Rc = \frac{TP}{FN + TP}$$

- **Specificity**: probability of pattern of class 1 will be classified as class 1.

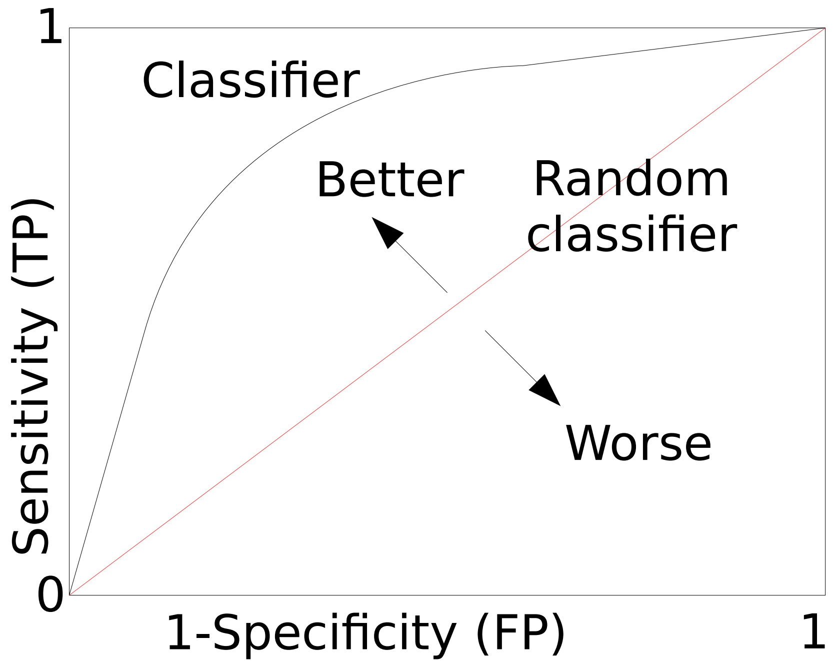
$$Sp = \frac{TN}{TN + FP}$$

- ROC (Receiver Operating Characteristic) curve: represents the sensitivity (or TP) in relation with the 1 – specificity (or FP).

ROC curves for binary problems

- The points of black line are obtained running the classifier with various threshold values to choose one of the two classes.
- The more to the left and up the black curve, the better classification.
- The lower left and upper right points are the extreme values of threshold, where all patterns are asigned to class 1 (FP=0, left) or 2 (TN=0, FP=1).
- With more than 2 classes, a ROC curve for each class (positive class) in relation to the remaining classes (negative class).

	C1	C2
C1	**TN**	FP
C2	FN	**TP**



Other quality measures for binary problems (I)

	C1	C2
C1	**TN**	FP
C2	FN	**TP**

- The Area Under the ROC Curve (AUC) also measures the classifier quality for two-class problems.
- **Positive predictivity** or **precision**: $PP = Pr = \frac{TP}{FP + TP}$
- F-score (o F1-measure): $\beta \in [0, +\infty)$ is a weighting factor of precision (Pr) and recall (Rc): $\beta = 0$ (only weights Pr), $\beta = \infty$ (only weights Rc)

$$F = F1 = \left(1+\beta^2\right) \frac{Pr \cdot Rc}{\beta^2 Pr + Rc} = \frac{\left(1+\beta^2\right) TP}{\left(1+\beta^2\right) TP + \beta^2 FN + FP}$$

- $\beta=1$ weights equally Pr and Rc; $\beta>1$: weights more Rc, $\beta<1$ weights more Pr.

Other quality measures for binary problems (II)

- Fowlkes-Mallows index:

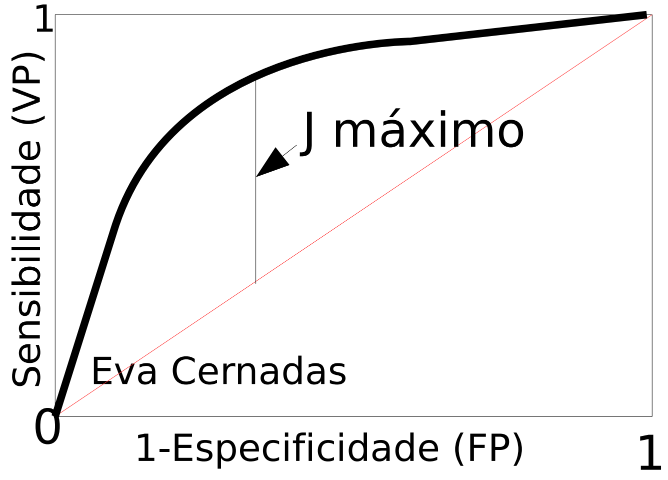
$$FM=\sqrt{Se\cdot PP}=\frac{TP}{\sqrt{(TP+FP)(TP+FN)}}$$

- Balanced accuracy: $Bacc=\frac{Se+Sp}{2}=\frac{1}{2}\left(\frac{TP}{TP+FN}+\frac{TN}{TN+FP}\right)$
- Matthews correlation coefficient (ϕ):

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

- Youden index: J=Se+Sp-1

$$J=\frac{TP}{TP+FP}+\frac{TN}{TN+FP}-1$$



Other quality measures for binary problems (III)

- False positive rate (FPR): $FPR = \frac{FP}{TN + FP}$
- False negative rate (FNR): $FNR = \frac{FN}{FN + TP}$
- True positive rate (TPR): $TPR = \frac{TP}{FN + TP}$

The simplest classifier: KNN

- The class of a pattern is predicted by voting among the closest training patterns (**x**: test pattern): $y(\mathbf{x}) = y_j,\ j = \underset{i=1\ldots C}{argmax}\{v_i\}$
- If K>1: voting, more robust than *K*=1: v_i=number of patterns of class *i*: predicts the most voted class among the K nearest neighbours
- 1NN: nearest neighbor classifier: $y(\mathbf{x}) = y_j,\ j = \underset{i=1\ldots N}{argmin}\{\lvert\mathbf{x} - \mathbf{x}_i\rvert\}$
- Using the Euclidean distance or others.
- There is no training nor trainable parameters. The whole training set must be stored (!).

Model selection (I)

- What number K of neighbors is the most suitable?
- K: hyper-parameter, not calculated in the training.
- Solution: tuning. Test with various values of K (odd for binary classification, in order to avoid ties) and choose the value which provides the highest performance.
- To evaluate the classifier performance without optimistically biasing, you need a separate test set, not used for tuning.
- K: tunable hyper-parameter. It exists in almost all the machine learning models (classifiers).

Model selection (II)

Cross Validation with 3 different sets:

1) **Training set**: used to calculate the trainable parameters using each hyper-parameter (hp) value.

2) **Validation set**: used to evaluate the model quality with each hp value.

   The training-validation loop is repeated for all the hp values. The value that provides the highest quality on the validation set is selected.

3) **Test set**: used to evaluate the quality of the model trained with the selected hp value on the training and validation sets.

Model selection (III)

- **K-fold Cross Validation with 3 sets**

  1) *K*=number of folds. Divide the available data into *K* disjoint partitions. Training the model with *K*-2 partitions.

  2) Validate with 1 of 2 left folds for each hp value.

  3) Repeat the process K times. Select the best hp with highest avg. perf.

  4) Train with K-1 folds and the best hp. Test on the remaining fold. Repeat *K* times. The model performance is the average test value.

K=4	Trial 1	Trial 2	Trial 3	Trial 4
Training	1 2	2 3	3 4	4 1
Validation	3	4	1	2
Test	4	1	2	3

Bias-variance dilemma (I)

- Let z(**x**$_i$) the model output for **x**$_i$ and y$_i$ the true output; let y(**x**) the true output for a test pattern **x**; let $\bar{z}$ the mean of z(**x**$_i$) over the training set:

$$\bar{z}=\frac{1}{N}\sum_{i=1}^{N} z(\boldsymbol{x}_i)$$

- **Bias** on **x**: $B=[\bar{z}-y(\boldsymbol{x})]^2=\left[\frac{1}{N}\sum_{i=1}^{N} z(\boldsymbol{x}_i)-y(\boldsymbol{x})\right]^2$

  Difference between the mean of the predicted outputs over the training set and true output for **x**.

- **Variance** over the training set:

$$V=\overline{z^2}-\bar{z}^2=\frac{1}{N}\sum_{i=1}^{N} z(\boldsymbol{x}_i)^2-\left[\frac{1}{N}\sum_{i=1}^{N} z(\boldsymbol{x}_i)\right]^2$$

  Difference between the mean of squared predicted output z(**x**$_i$)^2 and the square of the mean predicted output z(**x**$_i$)

Bias-variance dilemma (II)

- **High bias** on the training set means that model did not learn the training data correctly.
- **Low bias** on the training set means that model learnt the training data correctly.
- **High variance** means that predicted output varies very much with respect to its mean value, so it is over fitted to training data.
- **Low variance** means that predicted value does not change too much compared to its mean.
- Since bias and variance are errors, both should be low.
- However, bias and variance are constrained: to achieve low bias on the training set leads to high variance (over fitting).

Bias-variance dilemma (III)

- A bad training happens when bias on training set is high.
- Over fitting happens when variance is high.
- Bias and variance should be kept low simultaneously: better higher bias if lower values lead to higher variance.
- Model must work well on:

1) The training set: this requires low bias on it.
2) New data (validation or test) sets: this requires low variance.

- The presence of noise leads to a trade-off or dilemma between high training performance (variance) and validation / test performance (bias).

Over fitting and generalization ability

- If the model learns very well the training set: low bias but high variance.
- This can produce **over fitting**: good prediction for the training set, but bad prediction for new data sets.
- This normally happens when the number of trainable parameters is high in relation with the number of training patterns.

Curse of dimensionality

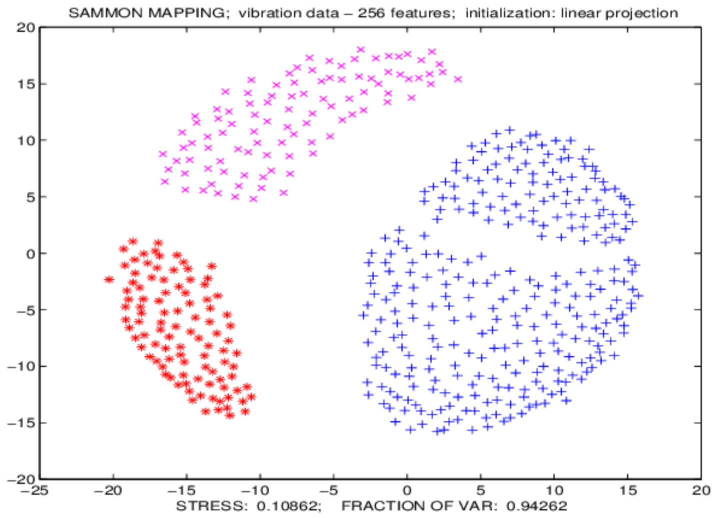
- When the dimensionality n raises, the volume of the input space raises very fastly
- Many data are required (high N) to cover the input space: data become sparse
- It was proven that in order to keep the data density required to learn a problem, the dataset size must raise **exponentially** with n
- Classifiers perform poorly due to low data density: the information is very low for such a high dimensionality

Ghaphical representation of a classification problem

- Plane containing projected patterns to 2D (color represents different classes).
- Shannon mapping: method to dimensionality reduction: visualize classification problems.
- From random patterns, it updates them satisfying that the distances between $\mathbf{x}_i$ in $\mathbb{R}^n$ and $\mathbf{y}_i$ $\mathbb{R}^2$ are similar.
- Minimizes the φ Kruskall ou Sammon stress:

$$\phi = \frac{\sum_{i<j} \frac{\left(d\,\mathbf{x}_{ij} - d\,\mathbf{y}_{ij}\right)^2}{d\,\mathbf{x}_{ij}}}{\sum_{i<j} d\,\mathbf{x}_{ij}}, \quad d\,\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j, \quad d\,\mathbf{y}_{ij} = \mathbf{y}_i - \mathbf{y}_j$$

- Shows the class overlap.



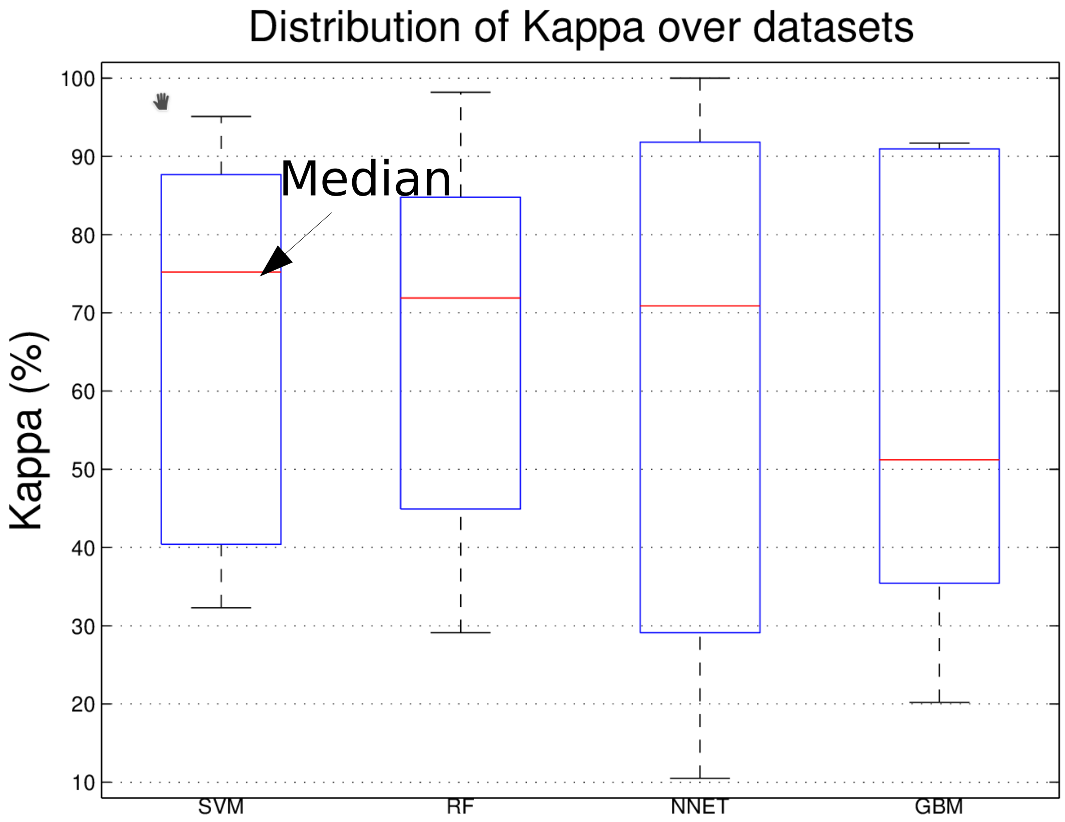
https://www.researchgate.net/figure/Sammon-mapping-of-vibration-data_fig3_2704140

Classifier comparison (I)

- For a classification problem: higher performance measurement (e.g. kappa) means better classifier
- If classifier A outperforms classifier B on a dataset, it does not mean that A outperforms B in all datasets
- Some classifiers perform better on certain datasets, while other classifiers perform better on other datasets
- It is not possible that the best classifier is the same in all the problems: no-free-lunch theorem (Wolpert & Macready, 1997): “every optimization algorithm is equivalent when it is averaged over all the possible datasets”
- In order to compare classifiers, we average kappa over a wide collection of datasets

Classifier comparison (II)

- The wider collection, the more reliable comparison. The size of the collection is very important
- The average kappa weights more datasets where kappa is higher
- We can compare graphically the kappa distributions: boxplot
- We can also use statistical tests to evaluate the significance of the difference between classifiers



Classifier comparison (III)

- Comparison between 2 classifiers X and Y: **Wilcoxon** ranksum test (Mann–Whitney U-test), among other tests
- ranksum(**x**,**y**) function in Octave/Matlab, where **x** and **y** are vectors with kappa of both classifiers over all datasets
- It tests the **null hipothesis** that both classifier performances over the dataset collection belong to statistical distributions with the same mean
- Tests whether classifiers X and Y are equally good. The test may say YES or NO
- The ranksum function returns a **p-value** (p): a high value means YES (accepts the null hypothesis), a low value means NO (rejects the null hypothesis)

Classifier comparison (IV)

1) When $p<0.05$ (o 5%), the null hypothesis is rejected:

- The difference is statistically significant in favour of the classifier with the highest kappa
- The difference between classifiers is high enough to consider one classifier as better than the other

2) When $p \geq 0.05$, the difference is not statistically significant (it is not high enough): we can not consider that a classifier outperforms the other on the current dataset collection

- We can extend the collection: the more datasets, the less difference is required to achieve $p<0.05$
- The difference between two classifiers normally reduces when the number of datasets increases

Classifier comparison (V)

- The Wilcoxon raksum test is useful for two classifiers
- To create a classifier ranking, where classifiers are ranked by decreasing kappa, you should use the **Friedman ranking**
- For each dataset in the collection, sort classifiers by descending kappa
- The rank of each classifier is its average position over all the datasets in the collection
- Let us consider the kappa achieved by the following classifiers and datasets:

Classifier	D1	D2	D3	D4	D5
SVM	95.1	32.3	85.2	75.2	43.1
RF	98.2	29.1	80.3	71.9	50.2
NNET	100	35.3	89.1	70.9	10.5
GBM	91.7	40.5	90.7	20.2	51.2

Classifier comparison (VI)

- How to create the Friedman ranking:

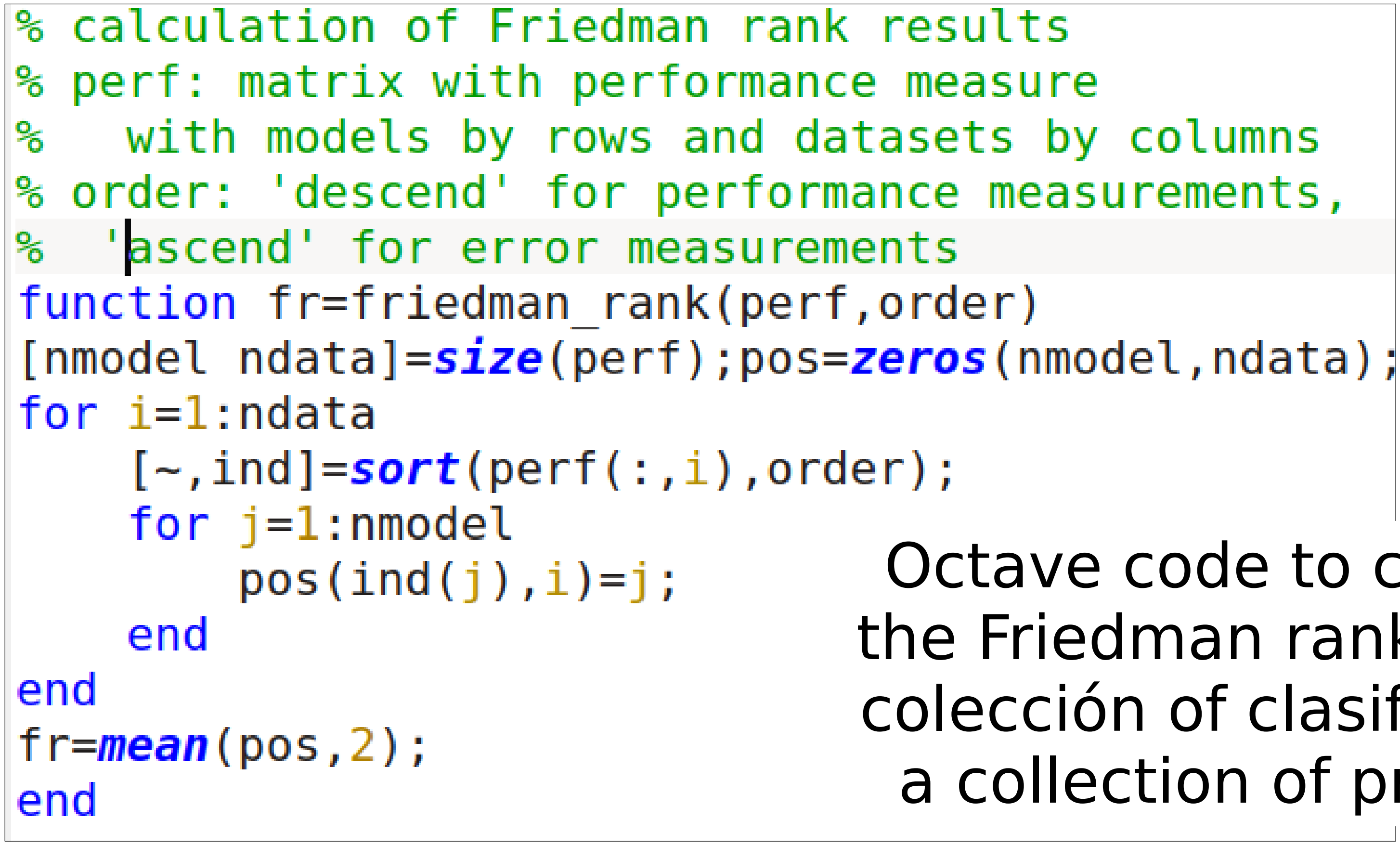
Sorting --->	1st	2nd	3rd	4th
D1	NNET	RF	SVM	GBM
D2	GBM	NNET	SVM	RF
D3	GBM	NNET	SVM	RF
D4	SVM	RF	NNET	GBM
D5	GBM	RF	SVM	NNET

The difference between the ranks of two classifiers reports the "distance" between them

Position	D1	D2	D3	D4	D5	Mean
SVM	3	3	3	1	3	2.6
RF	2	4	4	2	2	2.8
NNET	1	2	2	3	4	2.4
GBM	4	1	1	4	1	2.2

Position	Pos.	Rank
GBM	1ª	2.2
NNET	2ª	2.4
SVM	3ª	2.6
RF	4ª	2.8

Classifier comparison (VII)

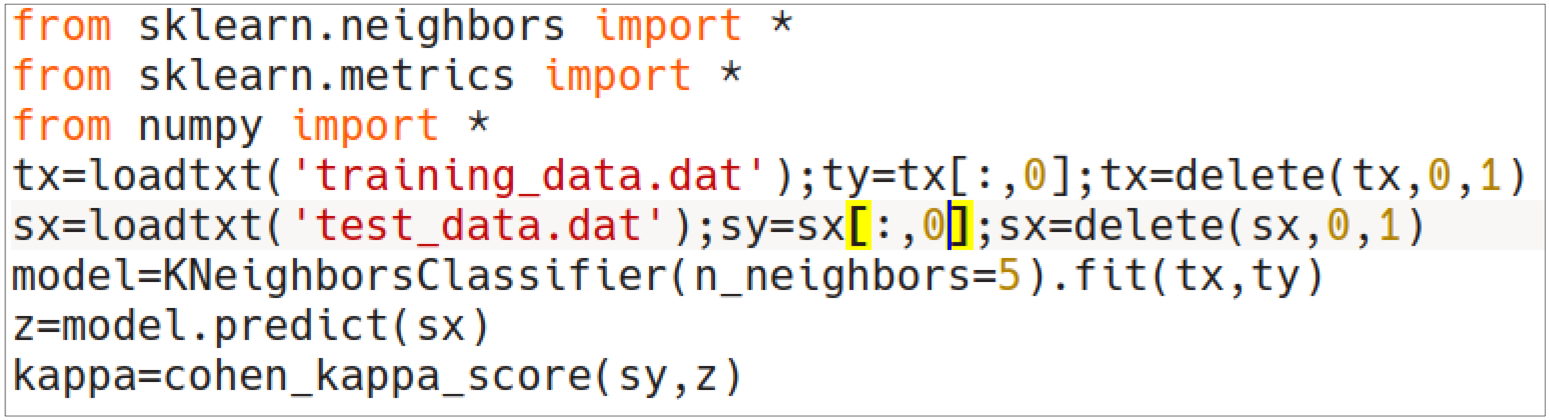


```
% calculation of Friedman rank results
% perf: matrix with performance measure
%   with models by rows and datasets by columns
% order: 'descend' for performance measurements,
%  'ascend' for error measurements
function fr=friedman_rank(perf,order)
[nmodel ndata]=size(perf);pos=zeros(nmodel,ndata);
for i=1:ndata
    [~,ind]=sort(perf(:,i),order);
    for j=1:nmodel
        pos(ind(j),i)=j;
    end
end
fr=mean(pos,2);
end
```

Octave code to calculate the Friedman ranking for a colección of clasifiers over a collection of problems

KNN classifier in Python

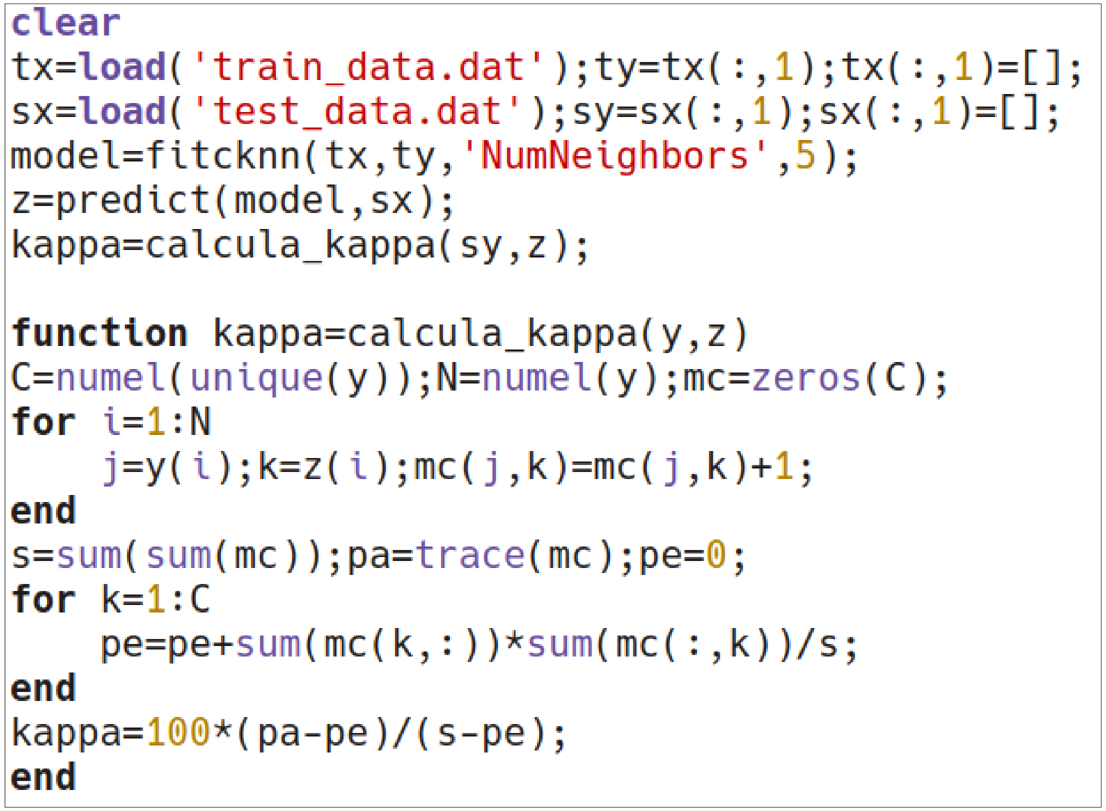
- Use the **scikit-learn** module.
- **Sklearn.neighbors.KNeighborsClassifier** object.
- **fit**() method for training.
- **predict**() method for testing.
- **sklearn.metrics.cohen_kappa_score**() for kappa calculation



```
from sklearn.neighbors import *
from sklearn.metrics import *
from numpy import *
tx=loadtxt('training_data.dat');ty=tx[:,0];tx=delete(tx,0,1)
sx=loadtxt('test_data.dat');sy=sx[:,0];sx=delete(sx,0,1)
model=KNeighborsClassifier(n_neighbors=5).fit(tx,ty)
z=model.predict(sx)
kappa=cohen_kappa_score(sy,z)
```

KNN classifier in Matlab

- Function **fitcknn**() for training.
- **predict**() for testing.



```
clear
tx=load('train_data.dat');ty=tx(:,1);tx(:,1)=[];
sx=load('test_data.dat');sy=sx(:,1);sx(:,1)=[];
model=fitcknn(tx,ty,'NumNeighbors',5);
z=predict(model,sx);
kappa=calcula_kappa(sy,z);

function kappa=calcula_kappa(y,z)
C=numel(unique(y));N=numel(y);mc=zeros(C);
for i=1:N
    j=y(i);k=z(i);mc(j,k)=mc(j,k)+1;
end
s=sum(sum(mc));pa=trace(mc);pe=0;
for k=1:C
    pe=pe+sum(mc(k,:))*sum(mc(:,k))/s;
end
kappa=100*(pa-pe)/(s-pe);
end
```