HAL
open science

Étude de la faisabilité de la classification automatique d'événements sismiques métropolitains sur la base d'attributs paramétriques

Nabil Belahrach

To cite this version:

Nabil Belahrach. Étude de la faisabilité de la classification automatique d'événements sismiques métropolitains sur la base d'attributs paramétriques. Méthodologie [stat.ME]. 2013. dumas-00854752

HAL Id: dumas-00854752

https://dumas.ccsd.cnrs.fr/dumas-00854752v1

Submitted on 28 Aug 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Ecole et Observatoire des Sciences de la Terre

UNIVERSITÉ DE STRASBOURG

Étude de la faisabilité de la classification automatique d'événements sismiques métropolitains sur la base d'attributs paramétriques

RAPPORT DE STAGE

Nabil BELAHRACH

Master 1 Mathématiques et Applications
Spécialité Statistique

Responsable de stage : Alessia MAGGI

Résumé

Chaque année, les réseaux de surveillance et d'alertes sismiques détectent de nombreux événements sismiques. Or , l'analyse manuel de ces enregistrements pour classifier les événement naturels des événements anthropiques reste une tache difficile.

Dans ce contexte on souhaite mettre en place un système d'aide à la décision qui respecte les critères suivantes:

- Classifieur paramétriques: le classifieur va prendre comme données seulement les paramètres de l'évènement (la localisation, la date l'heur, magnitude ….) et sans prendre en compte la forme d'onde.
- La performance: le classifieur doit commettre une erreur maitrisée.
- Le classifieur doit être fondé mathématiquement, de sorte que l'analyste puisse comprendre les démarches suivies pour l'étiquetage.

Dans ce projet nous proposons des modèles paramétriques pour la discrimination entre les événements sismiques naturels et les événements anthropiques (en particulier les tirs de carrière). en utilisant seulement les attributs paramétriques, en s'appuyant sur des méthodes d'apprentissage statistique tels que les SVM, le bayésien naïf et les k-plus proches voisins.
Ces modèles vont être tester sur le catalogue du LDG (Laboratoire de Détection Géophysique), en sachant que ce catalogue peut contenir une proportion d'erreur dans l'étiquetage des événements enregistrés.
Nous détaillerons la performance de ces trois modèles, et nous proposerons des solutions pour optimiser les modèles retenues, pour une éventuelle suite de ce projet.

Table des matières

1 Introduction à la Sismicité 4

1.1 Le dépouillement et la classification des signaux 5

1.2 Objectif du stage 6

1.3. Constructions de la base de données 6

2 Analyse discriminante et les outils statistique adaptés 10

2.1 Introduction à l'analyse discriminante 10

2.2 l'Apprentissage Statistique 11

2.3 Classification bayésienne naïve 11

2.4 Support Vector Machine (SVM): 13

2.4.1 Le cas de données linéairement séparables 13

2.4.2 Le cas de données linéairement non-séparables 14

2.5 Méthode du k plus proches voisin 16

3 Analyse de la base de données 17

3.1 classification avec Support Vector Machine 17

3.2 classification avec le bayésien naïf 27

3.3 classification par la méthode du k plus proches voisins 30

4 Conclusion et propositions 30

Bibliographie 32

Annexe A : code Python de la fonction de construction 33

Annexe B: code Python de la fonction d'analyse 43

Chapitre 1: Introduction à la Sismicité

Un évènement sismique naturel ou d'origine anthropique génère différentes ondes sismiques qui peuvent traverser un milieu en le modifiant selon la magnitude de l'évènement.
On distingue 3 trois types d'ondes : les ondes de volume P, S qui traversent la terre à partir du foyer , et les ondes de surface.
Dans les enregistrements du sismographe, on trouve ces ondes l'une après l'autre .

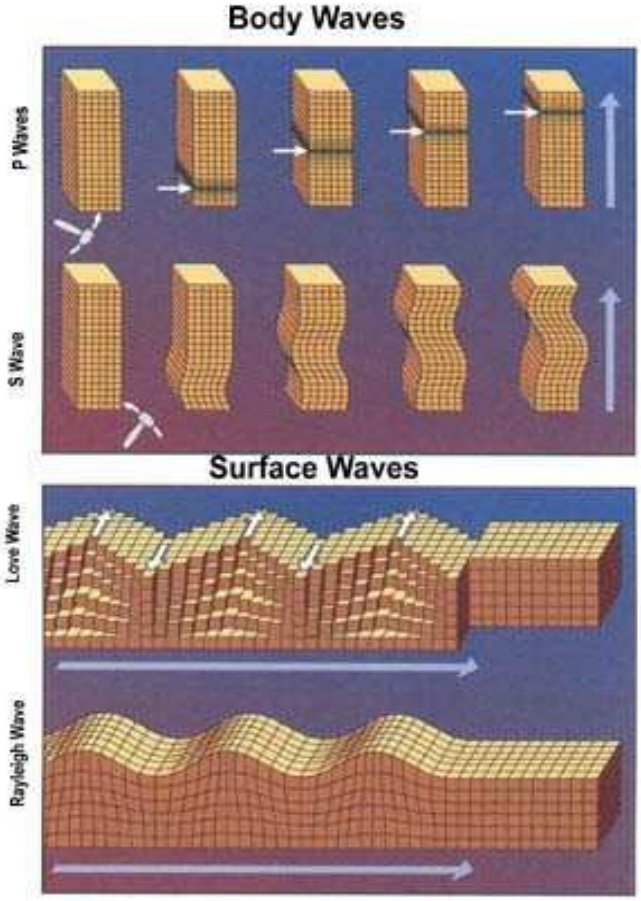


Figure 1: les différentes types d'ondes sismique et leurs effets

●L'onde P: ou de compression-dilatation , est une onde de volume qui secoue le sol d'avant en arrière dans le même sens et la direction opposée de la direction de propagation.
Elle se propage dans tous les types de milieu, par une vitesse de 4 à 8 km/sec

●L'onde S: ou onde de cisaillement une onde de volume, qui secoue le sol et les objet, d'avant et en arrière perpendiculaire à la direction de propagation de l'onde.
Elle de déplace à une vitesse de 2.5-4km/ sec
Ps: elle ne se propage pas dans les fluide à cause de l'absence de résistance de cisaillement

●les onde de surface: ce sont des ondes guidées par la surface de la Terre. Leur effet est comparable aux rides formées à la surface d'un lac. Elle sont moins rapides que les ondes de volume mais leur amplitude est généralement plus forte.

Les sismologues utilisent la différence de temps d'arrivée entre les ondes P et S pour calculer la distance entre la source du tremblement de terre et l'instrument d'enregistrement (sismographe)
Les scientifiques ont besoin des enregistrements provenant d'au moins trois sismographes pour pouvoir localiser avec précision la profondeur et la magnitude d'un séisme. Cette procédure est souvent automatisé en utilisant des techniques numériques de sorte qu'un grand nombre de différentes épisodes sismiques peut être traitée efficacement.

Les sites sismologiques doivent être bien loin de la circulation et d'autres sources de bruit de fond artificiel.

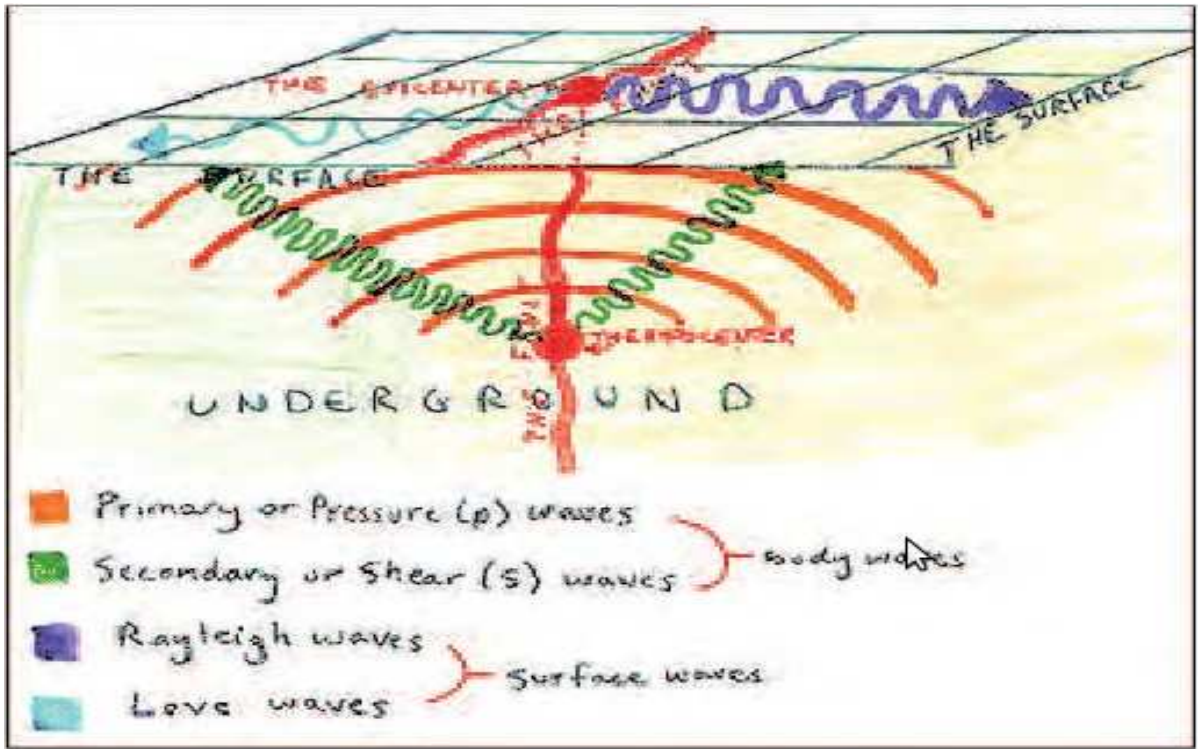


Figure 2: La propagation des trois types d'ondes

1.1 le dépouillement et la classification des signaux :

Les stations sismiques détectent de différentes signaux dont la source peut être un séisme naturel, un tire de carrière, une explosion nucléaire ou un simple exercice de l'armée ou encore un autre événement quelconque, ensuite ils sont traités et dépouillés par les analystes, à l'aide de logiciels conçus pour ce genre de travail. Et après cette phase l'analyste peut affirmer s'il s'agit:

- d'un séisme naturel: EQ
- d'un tire de carrière: blast
- d'un autre type d'évènement

Afin de pouvoir faire la différence entre les signaux d'un séisme naturel et un tir de carrière, on peut remarquer que les tirs de carrières ont des signaux de grande amplitude et de faible fréquence situées

après les ondes S que l'on nomme les ondes Lg . ainsi que par le fait que les tirs de carrière se passent généralement entre 10h et 14h .

1.2 Objectif du stage:

Développer un système d'aide la décision afin de discriminer la nature des événements sismiques enregistrés, en s'appuyant sur des modèles mathématiques de classifications automatiques et en utilisant seulement les informations paramétriques des événements sismiques et sans étudier les formes d'ondes.

1.3. Constructions de la base de données:

Le catalogue LDG (Laboratoire de Détection Géophysique) regroupe un certain nombre de fichiers (figure ci-dessous) qui contiennent à leur tour l'information collectée sur les événements sismiques (naturels et anthropiques) ayant eu lieu principalement en France métropolitaine et dans les zones frontalières.

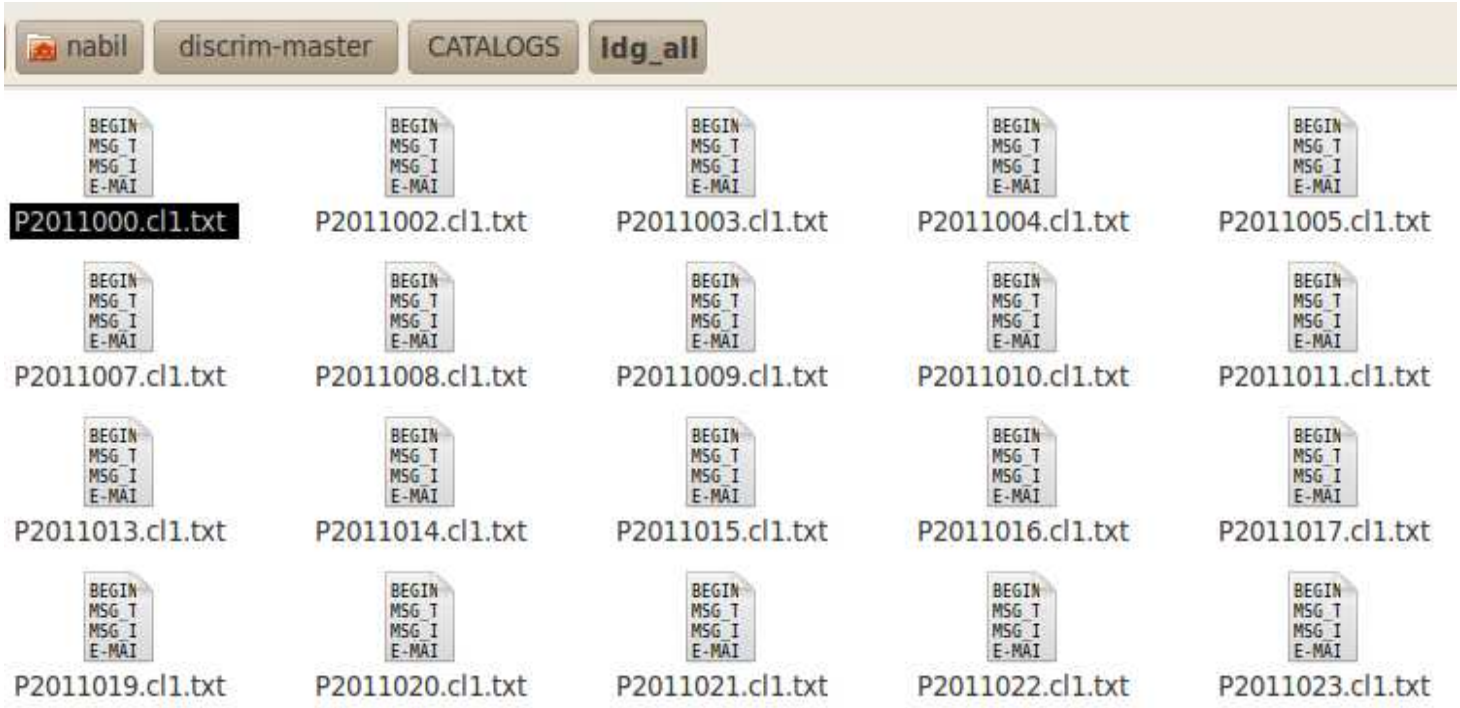


Figure 3: le catalogue LDG dans l'état initial 'brute'

Afin d'extraire l'information qui nous intéresse, à savoir les événement de types: séisme naturel 'ke' (known earthquake) et tire de carrière 'km' (known mining), sous python nous avons construit la fonction 'catalog_io_plot' (voir le code python dans l'annexe) qui parcoure les fichiers du catalogue LDG l'un après l'autre. Dans chaque fichier elle balaye les lignes et dans un premier temps on stocke dans un tableau seulement les lignes d'information qui nous intéresse (Figure 4) et qui correspondent à

des événements type 'ke' ou 'km'.
Ces lignes d'informations comporte la date de détection l'heure , la longitude et la latitude ,la magnitude ainsi que la profondeur et d'autre caractéristiques.
Dans une première étude on va s'intéresser tout d'abord à la date, l'heure et la localisation de l'évènement. On pourra ajouter d'autres attributs dans des études à venir .

```
P2011042.cl1.txt
54 EPF   000.70 069.1 m E Pg      2011/10/19 15:54:27.1   0.1                          T                                    477
55 EPF   000.70 069.1 m E Sg      2011/10/19 15:54:36.0  -0.2                          T                3.7  .26        Md 1.1  477
56
57 .
58 EVENT     265295
59     Date       Time        Latitude Longitude    Depth    Ndef Nsta Gap    Mag1  N    Mag2  N    Mag3  N   Author         ID
60        rms   OT_Error       Smajor Sminor Az        Err   mdist  Mdist     Err        Err        Err      Quality
61
62 2011/10/20 03:53:41.8     44.9974    6.5248      3.0 f    27    9 094  Ml 2.4  1  Md 2.0  4              js067245    265315
63
64       0.50   +-  0.08       1.7    1.4   94.07 +-  0.0    0.20   1.66   +-         +-0.1                m i ke
65
66 FRANCE
67 Sta     Dist  EvAz     Phase      Date       Time     TRes  Azim  AzRes  Slow  SRes Def   SNR       Amp   Per   Mag1   Mag2
68 MBDF  000.32 146.9 m E Pg      2011/10/20 03:53:48.0  -0.3                          T                                    477
69 MBDF  000.32 146.9 m E Sg      2011/10/20 03:53:52.4  -0.2                          T                7.5  .28        Md 1.8  477
70 ORIF  000.46 260.4 m E Pg      2011/10/20 03:53:50.7  -0.1                          T                                    477
```

Figure 4: Exemple d'un fichier du catalogue LDG, et la ligne d'information collecter par «catalog_io_plot»

La figure 5 décrit le premier résultat de la fonction «catalog_io_plot»: on trouve bien les événements de type 'ke' qui sont des séismes naturels et 'km' qui correspondent au tires de carrières, ainsi que les informations associées à chaque événement.

Jour	Heure	Latitude	Longitude	Prof	Magn	Nb	Gap	Type événement
920625	1251 02.4	43-27.59	6-30.59	2.00	0.00	15	226	ke
940213	1236 59.4	43-54.63	7- 8.85	17.30	2.00	13	255	ke
940607	1048 35.9	44- 3.28	6- 0.56	15.00	1.70	10	255	ke
940607	1543 41.8	44- 2.98	5-52.70	15.00	2.10	20	190	ke
020110	0839 18.4	45-20.27	5-39.48	2.00	2.10	21	77	ke
020711	1043 59.5	46-43.76	6-12.35	19.10	2.60	22	153	ke
041109	1031 22.4	45- 9.00	3-49.80	5.00	2.40	12	291	km
070220	1404 01.1	46-43.63	6- 9.76	15.00	2.70	48	131	km
070410	1623 12.7	43-57.98	7-23.20	5.00	2.30	40	62	ke

Figure 5: une partie de donnée extraites du catalogue par la fonction python «catalog_io_plot»

Dans un second temps et pour construire notre modèle mathématique la fonction «catalog_io_plot» crée un fichier 'ldg.hdf5' qui se compose d'une matrice X, et un vecteur de réponse Y.

Les vecteurs colonnes de la matrice X sont les colonnes de la table extraite précédemment (Figure 5) , dans l'ordre suivant: longitude , latitude , le jour, l'heure, 'd' une variable binaire pour différencier le jour et la nuit , et une variable binaire 'w' pour différencier les jours ouvrable de la semaine et les weekends.

Le vecteur Y n'est que la colonne 'type d'événement' (figure 5) avec un petit changement de sort que on affect 0 pour les événement 'ke' et 1 pour les types 'km'.

La variable classe, et les prédictives :

On va s'intéresser tout particulièrement aux variables dont l'intérêt a déjà été démontré dans des études précédentes :

- heure : variable quantitative cyclique comprise dans [0; 24]
- latitude : variable quantitative comprise dans [42; 51]
- longitude : variable quantitative comprise dans [−5; 9]
- magnitude : variable quantitative comprise dans [0, 7; 6, 0]
- date : variable explicative qu'on va traiter avec la fonction python Otime afin de faire apparaître les variables t, d et w.
- classe : variable qualitative ayant deux modalités
  EQ : séismes (EarthQuakes)
  blast : explosions de mines

classe	effectif	pourcentage
1) (EQ / séismes)	8832	87.78 %
2) (blast / explosions de mines)	1229	12.22 %
Totale	10061	100,00%

Tableau 1: la base de donnée étudiée

L'objectif est de construire un modèle statistique capable de prédire la classe d'un évènement sismique grâce à ces variables paramétriques. Ainsi, la variable classe est dite variable d'intérêt (ou de réponse) et toutes les autres sont dites variables aléatoires explicatives ou attributs ou encore descripteurs.
Dans une première étude on considère comme variables explicatives : la longitude, la latitude, l'heure et la date et on ajoutera la variable magnitude dans une seconde étude .

Dans le cas de trois variables explicatives :

En réalité et dans le code python on a six variables explicatives et puis à sept si on ajoute la variable magnitude :

x : la longitude
y : la latitude
t : le jour fractionnaire
h : l'heure locale
d : jour / nuit
w : si pendant le weekend / ou dans les 5 jours de la semaine

Les attributs sont x, y en km , 't' en jours et en heures h. La variable binaire de jour / nuit, d = 1 pour le jour, d= 0 pour la nuit est fixé en prenant en compte que les temps après 18:00 heure locale et avant l'heure locale 06:00 sont pendant la nuit.
Les quatre dernières attribues, on les obtient grâce à la fonction python (otime), cette transformation a énormément d'importance vue que les tirs de carrière se passent en générale dans la journée entre pendant les jours ouvrables de la semaine.
Y : la variable de réponse , elle prend deux valeurs {1, 0} , 0 pour les tremblement de terre et 1 pour les tires.

ainsi notre classifieur peut s'écrire comme :

$$Y = F (x, y, t, h, d, w)$$

Dans le tableau suivant on commence par regarder la corrélation entre les variables explicatives.

	longitude	latitude	La date	heure
longitude	1	0.339910735	-0.052698967	5.269897e-02
latitude	0.339910735	1	-0.055477366	0.009727357
La date	-0.052698967	-0.055477366	1	3.690935e-05
Heure	5.269897e-02	0.009727357	3.690935e-05	1

Tableau 2 : Matrice corrélation des variable quantitatives

D'après la matrice de corrélation on peut affirmer qu'il n'y a pas de liaison forte entre les variables quantitatives. Elle sont donc indépendantes ce qui peut être nécessaire pour nous éventuellement dans des méthodes classifications qui exigent ce critère (les méthode bayésiens par exemple).

Chapitre 2 : Analyse discriminante et les outils statistiques adaptés

2.1 Introduction à l'analyse discriminante:

L'analyse discriminante est utilisée dans des situations ou les classes sont connus a priori . Le but de l'analyse discriminante est de classer une observation ou plusieurs observations, dans ces groupes connus. Par exemple, dans la notation de crédit, une banque sait par expérience qu'il y a de bons clients (qui remboursent leur prêt sans aucun problème) et les mauvais clients (qui ont montré des difficultés à rembourser leur prêt). Quand un nouveau client demande un prêt, la banque doit décider de donner ou non le prêt. Les dossiers antérieurs de la Banque fournit deux ensembles de données: observations multivariées x_i sur les deux catégories de clients (y compris par exemple l'âge, le salaire, le statut marital, le montant de l'emprunt, etc.) Le nouveau client est une nouvelle observation x avec les mêmes variables. La règle de la discrimination doit classer le client dans l'un des deux groupes existants et l'analyse discriminante devrait évaluer le risque d'une éventuelle décision «mauvaise».

Différentes méthodes dans le domaine statistique ont été développées pour résoudre des problèmes de discrimination (des plus anciennes aux plus récentes) :

- l'analyse discriminante décisionnelle
  avec règle de décision géométrique (liée à l'Analyse Factorielle Discriminante),
  avec règle de décision bayésienne,
  la règle de plus proche voisin
- les modèles linéaires généralisés :
  régression logistique (dans le cas à deux classes),
  régression polychotomique (dans le cas à plusieurs classes)
- les modèles connexionnistes.
- les arbres de décision .
- les Support Vector Machine (SVM).

Durant cette études on utilisera les méthodes suivantes: les SVM , le bayésien naïf , et l'algorithme des plus proches voisins

2.2 l'Apprentissage Statistique :

Nous allons nous intéresser essentiellement dans cette étude à l'apprentissage supervisé, pour lequel on dispose d'un ensemble d'apprentissage constitué de données d'observations de type entrée-sortie : $D^n = \{(x_1, y_1),.....,(x_n, y_n)\}$ avec x_i dans X (espace des attribues), et y_i dans Y (l'espace des classes) pour i = 1.....n.

L'objectif est de construire, à partir de cet échantillon d'apprentissage, un modèle qui va nous permettre de prédire la réponse y associé à une nouvelle observation (ou prédicteur) x.

La sortie y peut être quantitative (prix d'un produit, courbe de consommation d'énergie...) comme elle peut être qualitative (attribution d'un prêt à un client, reconnaissance de visages ….) .

On parle d'un problème de classification (ou discrimination) lorsque Y est fini

#Y: est le nombre des classes

Dans le cas de la classification binaire : soit f(x) la fonction apprise par la machine d'apprentissage. On appelle $R_{emp}(f)$ le taux d'erreurs effectuées par la fonction f sur une nouvelle échantillon (échantillon de test):

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i))$$

avec,

$$L = \begin{cases} 1 & \text{si} \quad y_i \neq f(x_i) \\ 0 & \text{sinon} \end{cases}$$

2.3 Classification bayésienne naïve

Méthodes Naïve Bayes sont un ensemble d'algorithmes d'apprentissage supervisé basé sur l'application du théorème de Bayes avec l'hypothèse «naïve» de l'indépendance entre chaque attribut. Étant donné une variable de classe 'y' dépendante à un vecteur de variables explicatives x_1 ..... x_n .

le théorème de Bayes affirme affirme la relation suivante :

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

En utilisant l'hypothèse naïve d'indépendance, cette relation est simplifié à :

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

et donc :

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Les paramètres du modèle (probabilités *a priori* des classes et lois de probabilités associées aux différentes caractéristiques) peuvent faire l'objet d'une approximation par rapport aux fréquences relatives des classes et caractéristiques dans l'ensemble des données d'entraînement.
Il s'agit d'une estimation par maximum de vraisemblance des probabilités. Les probabilités *a priori* des classes peuvent par exemple être calculées en se basant sur l'hypothèse que les classes sont équiprobables, ou bien en estimant chaque probabilité de classe sur la base de l'ensemble des données d'entraînement (la probabilité empirique). Pour estimer les paramètres d'une loi de probabilité relative à une caractéristique précise, il est nécessaire de présupposer le type de la loi en question, sinon, il faut générer des modèles non-paramétriques pour les caractéristiques appartenant à l'ensemble de données d'entraînement.
La règle de décision employée consiste à choisir l'hypothèse la plus probable.

$$\text{classifieur}(x_1 \ldots\ldots x_n) = \underbrace{argmax}_{c} \; p(Y=y) \prod_{i=1}^{n} p(X_i = x_i \mid C=c)$$

Dans notre étude sous python on utilise la méthode 'GaussianNB' qui suppose que les probabilités des caractéristiques sont gaussiennes:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\pi\sigma_y^2}\right)$$

les paramètres μ_y et σ_y sont estimées en utilisant la méthode du maximum de vraisemblance

Sous python on trouve la fonction 'GaussianNB' dans le module 'sklearn.naive_bayes'

2.4 Support Vector Machine (SVM):

2.4.1 Le cas de données linéairement séparables:

la méthode des Support Vectors Machines (les séparateurs à vastes marges) a été développé au début des années 90 par Vladimir Vapnik . Cette méthode consiste à déterminer les hyperplans séparateurs entre des classes au lieu de modéliser les probabilités d'appartenance d'un objet à une classe (comme la méthode bayésienne), cette surface va être engendrer (définit) par quelques éléments de l'échantillon d'entrainement. Ainsi l'idée est d'offrir une représentation compacte des objets à classer . Dans le cadre des SVM, l'étape de l'entrainement détermine l'hyperplan séparateur qui soit le plus optimale possible.

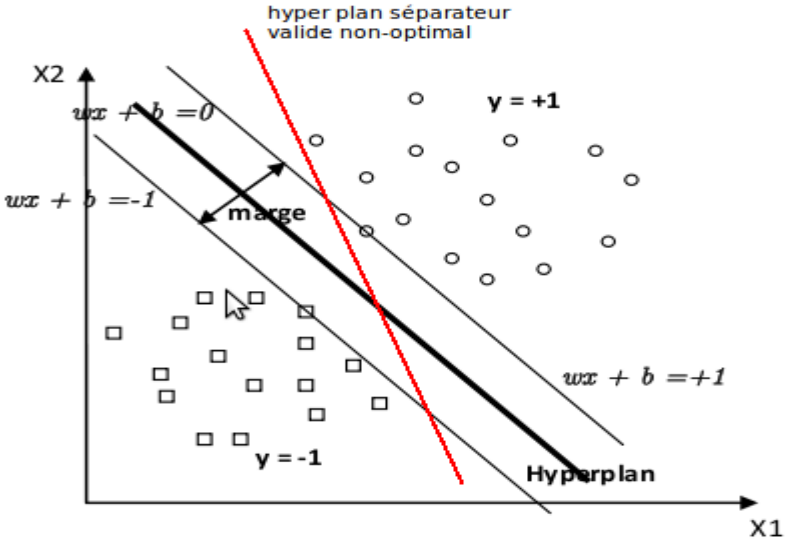


FIGURE 6: SVM binaire dans le cas séparable

Analytiquement On peut démontrer les affirmation suivantes :

l'équation de l'hyperplan optimal est représenté par: $H(x)= w^T x + b = 0$

classe = 1 si $H(x) > 1$

classe = -1 si $H(x) < 1$

$y_i(w^T x_i + b) \geq 1$, pour $i = 1,.....,n$

w est orthogonal au hyperplan séparateur

Donc: pour trouver l'hyperplan séparateur optimal, on doit déterminer w qui a la norme minimal et qui vérifier les affirmation précédente, par conséquent l'hyperplan optimal et obtenue par la résolution du problème d'optimisation linéaire suivant:

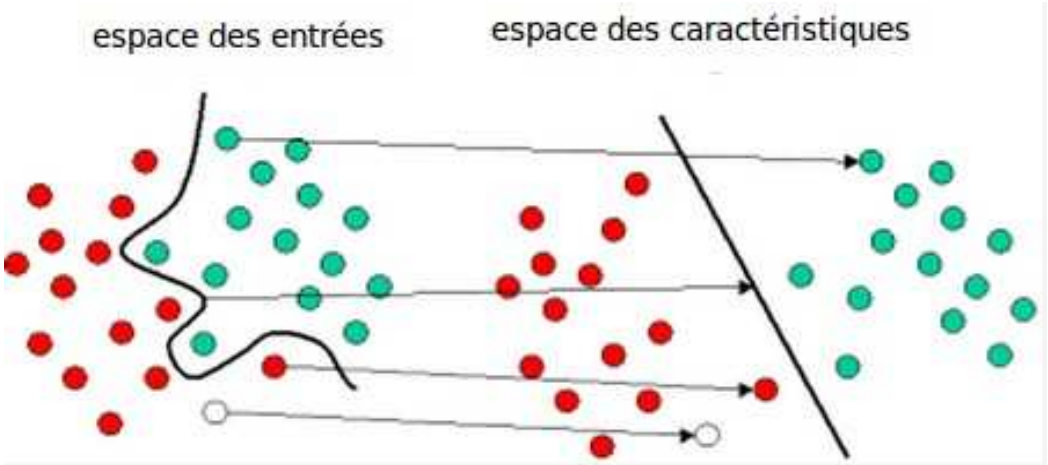
$$\begin{cases} \text{Minimiser} & \frac{1}{2}\|w\|^2 \\ \text{sous contraintes} & \\ y_i(w^T x_i + b) \geq 1 & \forall i = 1..n \end{cases}$$

qui ce résout à l'aide du théorème KKT [7] et par passage au problème dual.

2.4.2 Le cas de données linéairement non-séparables:

La plupart des tâches de classification, ne sont pas si simples que le cas linéaire, et souvent des structures plus complexes sont nécessaires afin de faire une séparation optimale, c'est à dire classifier correctement les nouveaux objets (le l'échantillon de test) sur la base des exemples qui sont disponibles (l'échantillon d'entrainement). Cette situation est représentée dans l'illustration ci-dessous. Par rapport au schéma précédent, il est clair qu'une séparation complète des objets verts et rouges exigerait une courbe (ce qui est plus complexe qu'une droite).

L'idée est de passer de l'espace des entrées à l'espace appelé "espace des caractéristiques" dans lequel on cherche à rendre la discrimination linéaire. Ce passage d'une fonction discriminante non-linéaire à une fonction discriminante linéaire est résolu par la méthode dite du noyau. La solution SVM consiste à choisir l'hyperplan séparateur de stabilité maximale (en relation avec la marge fixée) dans le nouveau espace lorsque le problème est séparable linéairement. Plus le potentiel du perception de sortie est grand, plus on est sûr de la classification. On note que l'on peut donner une interprétation probabiliste de la classification avec les distances des exemples aux surfaces discriminantes. Dans le cas de deux classes, on cherche à déterminer h, fonction continue, proche de 1 pour les objets appartenant à la première classe et de -1 pour l'autre. La surface h(x) = 0 définit la surface séparatrice.



FUGURE 7: SVM binaire, cas linéairement non séparable

Cette transformation d'espace est réalisée souvent à l'aide d'une fonction appelé "Mapping function" et le nouveau espace est appelé espace de caractéristiques $F = \{\varphi(x) | x \in X\}$

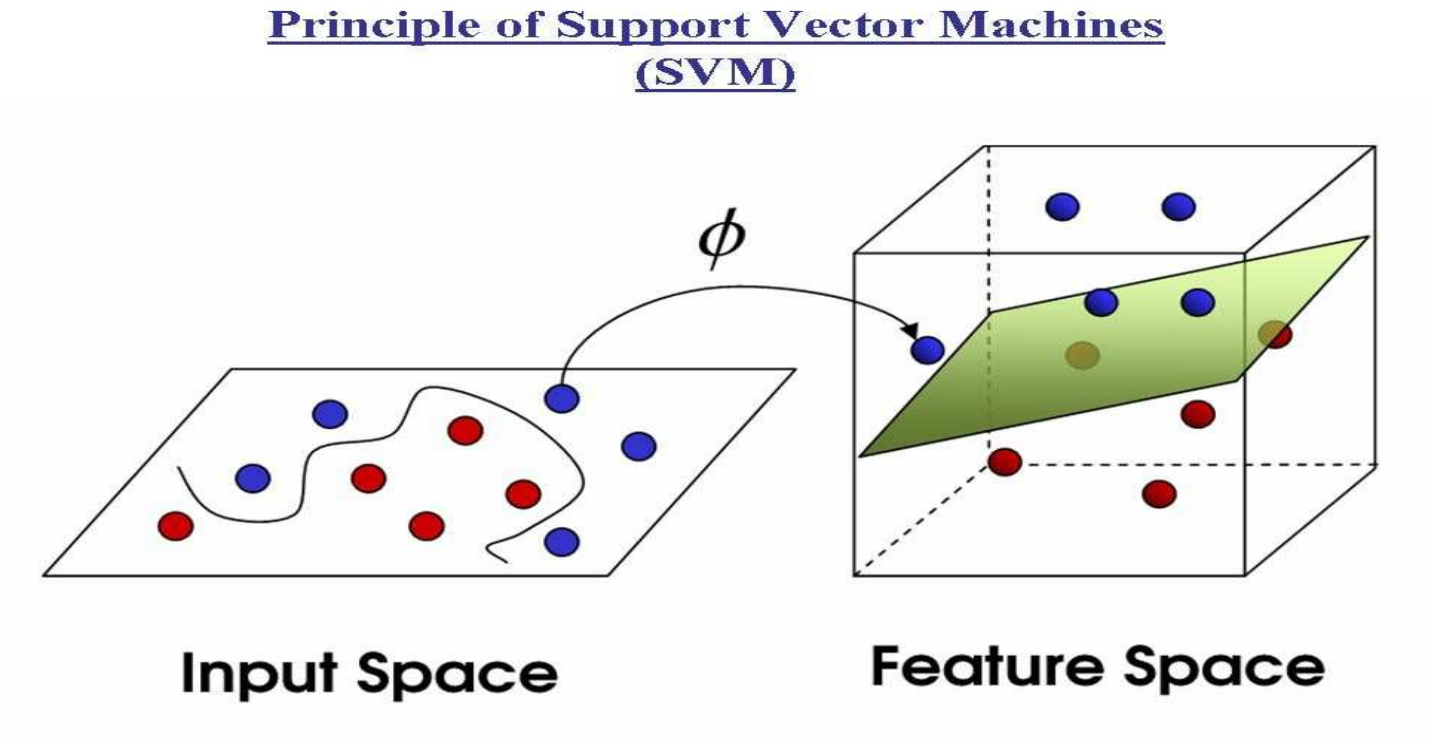


FIGURE 8: la transformation du problème non linéaire au linéaire grâce à la fonction φ

La transformation φ et son inverse (pour revenir à l'espace des données) sont réalisées grâce à une fonction noyau $K(x_i,x_j)$ telle que:

$$K(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle$$

Quelques noyaux connues:

$\mathbf{K(x_i, x_j) = x_i . x_j}$: le noyau linéaire

$\mathbf{K(x_i, x_j) = (1+ x_i . xj)^p}$ noyau polynomial de degré p

$K(x_i, x_j) = \exp^{\frac{-||x_i - x_j||^2}{2\sigma^2}}$: Noyau Gaussien

$K(x_i, x_j) = \tanh(\alpha x_i \cdot x_j + \beta)$: noyau de Perceptron à 2 couches

L'hyperplan séparateur optimal et solution de du problème d'optimisation suivant :

$$\begin{cases} \text{Maximiser} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{sous contraintes} & \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{cases}$$

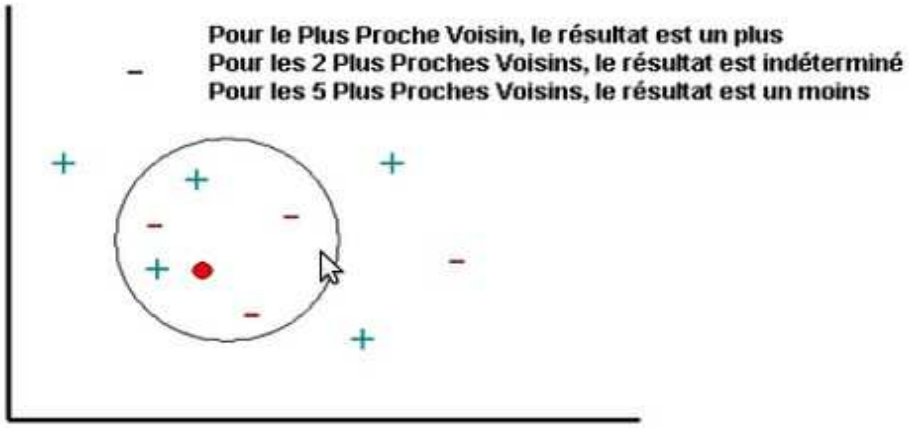
(pour voir l'inégalité de la démonstration voir [7])

Il dépend de deux paramètres à fixer par l'utilisateur (gamma et C) , on va les choisir de sort que le classifieur commit un taux d'erreur minimale. On verra ceci dans l'étape de la programmation et l'analyse

Sous Python on trouve la fonction 'svm' dans le module 'sklearn'

2.5 Méthode du k plus proches voisins:

c'est un algorithme basé sur la mémoire, il appartient à la catégorie des méthodes prototypes. Il n y a aucun modèle à ajuster. Ça fonction avec intuition: les points les plus proches ont plus de chances d'appartenir à la même classe. Ainsi avec la méthode des k plus proches voisins, les prévisions se reposent sur un ensemble d'entrainement prototype qui sert à prévoir de nouvelles données de échantillon test (c'est-à-dire, inconnues) sur la base du vote majoritaire (pour les tâches de classification) ou sur la base de la moyenne (pour les tâches de régression) des K plus proches prototypes (d'où le nom K Plus Proches Voisins).



FUGURE 9: principe de la méthode du k-plus proches voisins

Les points { - , + } représentent des objets de l'échantillon d'entrainement placés dans l'espace des attribues. La petite boule rouge représente un objet de l'échantillon de test auquel l'algorithme cherche à affecter une étiquette.
Dans cet exemple on remarque que le choix du nombre de voisins (nombre de votes) et très influent sur la décision d'étiquetage . En pratique et de façon analogue avec ce qu'on a fait avec les SVM, Il faut choisir le nombre de votants qui nous offre un taux d'erreur minimal.
Sous python on trouve la fonction 'neighbors' dans le module 'sklearn'

Chapitre 3: Analyse de la base de données

De nombreux éléments utilisés dans la fonction objectif d'un algorithme d'apprentissage (tel que le noyau 'rbf' du SVM) supposent que toutes les fonctionnalités sont centrées autour de 0 et ont une variance dans le même ordre. Si une caractéristique a une variance significativement plus grande que les autres, elle pourrait dominer la fonction objectif et rendre le classifieur incapable d'apprendre d'autres fonctionnalités correctement.
Pour sortir de cette problématique on normalise les caractéristiques en enlevant la moyenne en la mise à l'échelle de variance unité (on divise par l'écart-type du caractéristique).
Sous Python on utilise la fonction 'preprocessing.StandarScaler' pour normaliser les vecteurs colonnes de la matrice X.
Pour tous les algorithmes utilisés dans l'analyse on utilisera la matrice X normalisée.

3.1 Support Vector Machine:

3.1.1 Pour C et gamma par défaut :

Notre classifieur peut s'écrire comme : $Y = f_{c,\gamma}(x, y, t, h, d, w)$

Durant toute cette étude, on a pris le choix de validation croisé à trois échantillons , apprentissage (training set), test (test set) et validation (validation set) .
Pour commencer on a choisit d'affecter 30% de données pour l'entrainement, et diviser les 70% restante en deux entre le test et la validation .
On construit le classifieur SVM pour le catalogue LDG, avec des valeurs de C , et gamma par défaut dans un premier temps . (C = 10, gamma = 10)
Implicitement:

- L'étape entrainement: consiste à construire l'hyperplan séparateur.
- L'étape test : se résume à placer et étiqueter les point de l'échantillon de test.
- Le taux l'erreur : on comparer l'étiquetage prédit par l'algorithme SVM et l'étiquetage observé sur les événements test et on calcule ensuite le proportion l'erreur commis par le classifieur, et les probabilités.

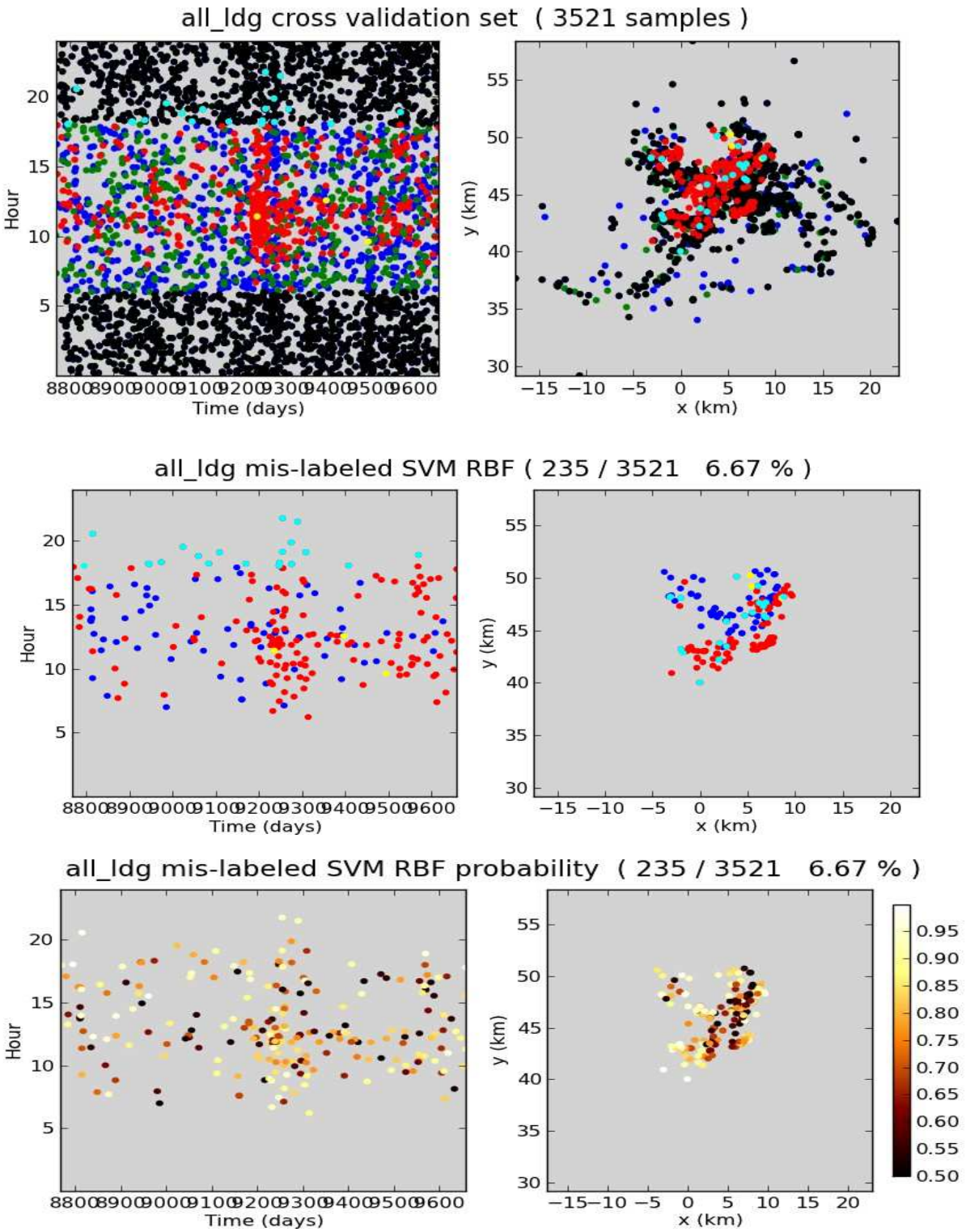


FIGURE 10: (c & gamma par défaut , et test+validation = 70%)

Diagrammes de dispersion du validation set en temps (à gauche) et en espace (à droite) des codes de couleur sont comme suit: tremblements de terre au cours de la semaine (bleu), tremblement de terre week-end (vert), les tremblements de terre au cours de la nuit (noir) ; explosions au cours de la semaine (rouge), les explosions pendant le week-end (jaune), les explosions la nuit (cyan).

Diagramme d'événements mal-classés par l'algorithme SVM. Et le diagramme des probabilités

On remarque que le le choix le la partition est assez influant au sens d'erreur de classification, ainsi pour trouver une partition plus optimale, on évalue l'erreur d'étiquetage de l'algorithme SVM en fonction du pourcentage complémentaire au training c-a-d le pourcentage (test + validation)

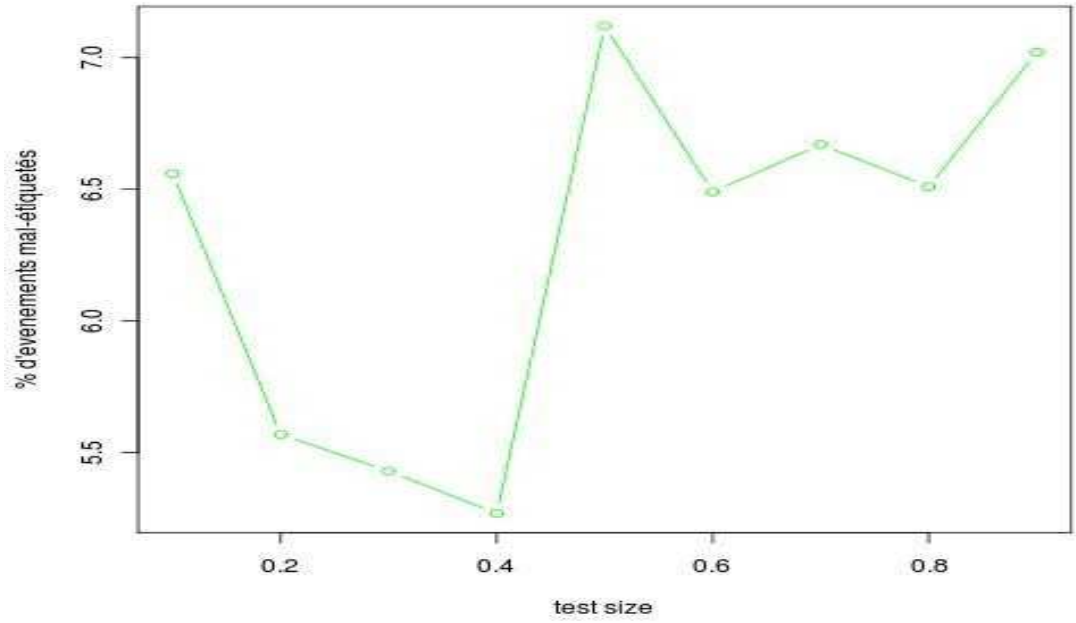


Figure 11 : évaluation de l'erreur de l'algorithme SVM en fonction du choix de pourcentage de donnée affecter à la (test set + validation set)

D'après la courbe de variation de l'erreur par rapport au effectif de l'échantillon, le taux d'erreur minimal est atteint en 0.4 par conséquent, on admet pour le reste du travail que la partition idéale est la suivante : 60% pour le training set , et 20% test , et 20% la validation.

l'échantillon	Le pourcentage	Nombre d'évènement
Entrainement	60%	6037
Test	20%	2012
Validation	20%	2012

On repartitionne notre base de donnée sous cette forme, et relance l'algorithme SVM :

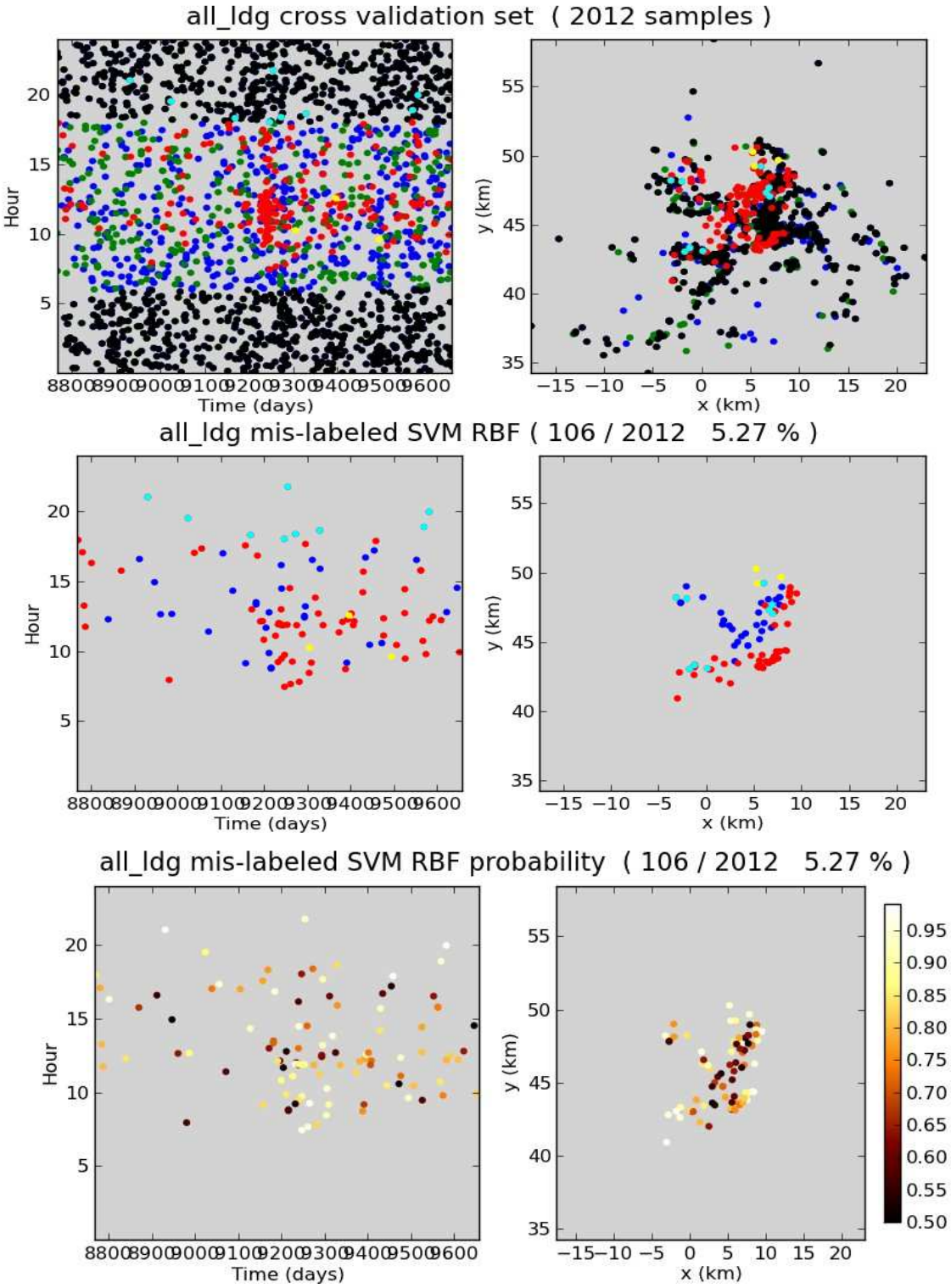


Figure 12: (c & gamma par défaut , et test+validation = 0.4 : le choix optimal)

Diagrammes de dispersion du validation set en temps (à gauche) et en espace (à droite) des codes de couleur les mêmes que les figures précédentes

3.1.2 Recherche des paramètres d'estimation C, et gamma :

Comme ce qu'on a vu dans le cours théorique des séparateurs à vastes marges, l'hyperplan plan optimal dépend de plusieurs paramètres et c'est un point commun avec un certains nombre de classificateurs. Les paramètres qui ne sont pas pris directement dans les estimateurs peuvent être réglés par une recherche dans l'espace des paramètres pour la meilleure validation croisée. Dans le cas du SVM il s'agit des paramètres : C , le noyau , et gamma
De façon générale la recherche des paramètres se compose de :

- Un estimateur (variable explicative ou classificateur comme sklearn.svm.SVC() dans notre cas).
- Un espace des paramètres .
- Une méthode de recherche ou d'échantillonnage des candidats.
- Un système de validation croisé.
- Une fonction de score (dans notre études on va s'intéresse plutôt au taux d'erreur)

Le code python :

```
#do grid search
print "doing grid search"
C_range = 10.0 ** np.arange(-2, 5)
gamma_range = 10.0 ** np.arange(-3,3)
param_grid = dict(gamma=gamma_range, C=C_range)
grid = GridSearchCV(svm.SVC(), param_grid=param_grid )
grid.fit(X_train, y_train)
print("The best classifier is: ", grid.best_estimator_)
```

GridSearchCV() est une fonction qui effectue une recherche de grille sur les paramètres spécifiés. La méthode résumé sur l'objet retourné indique le taux d'erreur de classification pour chaque combinaison de paramètres et le meilleur modèle. Par défaut, le mesure d'erreur est calculée en utilisant une validation croisée 3 fois sur les données fournies.
Dans cet exemple, le meilleur modèle dans la plage de paramètre est obtenue à l'aide de ($C = 1$ et $\gamma = 1$) .

le résultat du code python:

{('The best classifier is: ', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=1.0,kernel='rbf', max_iter=-1, probability=False, shrinking=True, tol=0.001, verbose=False)) }

Et donc les paramètres optimaux: **(C^* ; $gamma^*$) = (1 , 1)**
On règle notre classificateur les nouveaux paramètre optimaux, onrelance l'algorithme SVM sur la même base de données. On remarque (Figure ci-dessous) une baise significative du taux de mal-classés dans l'échantillon de test :

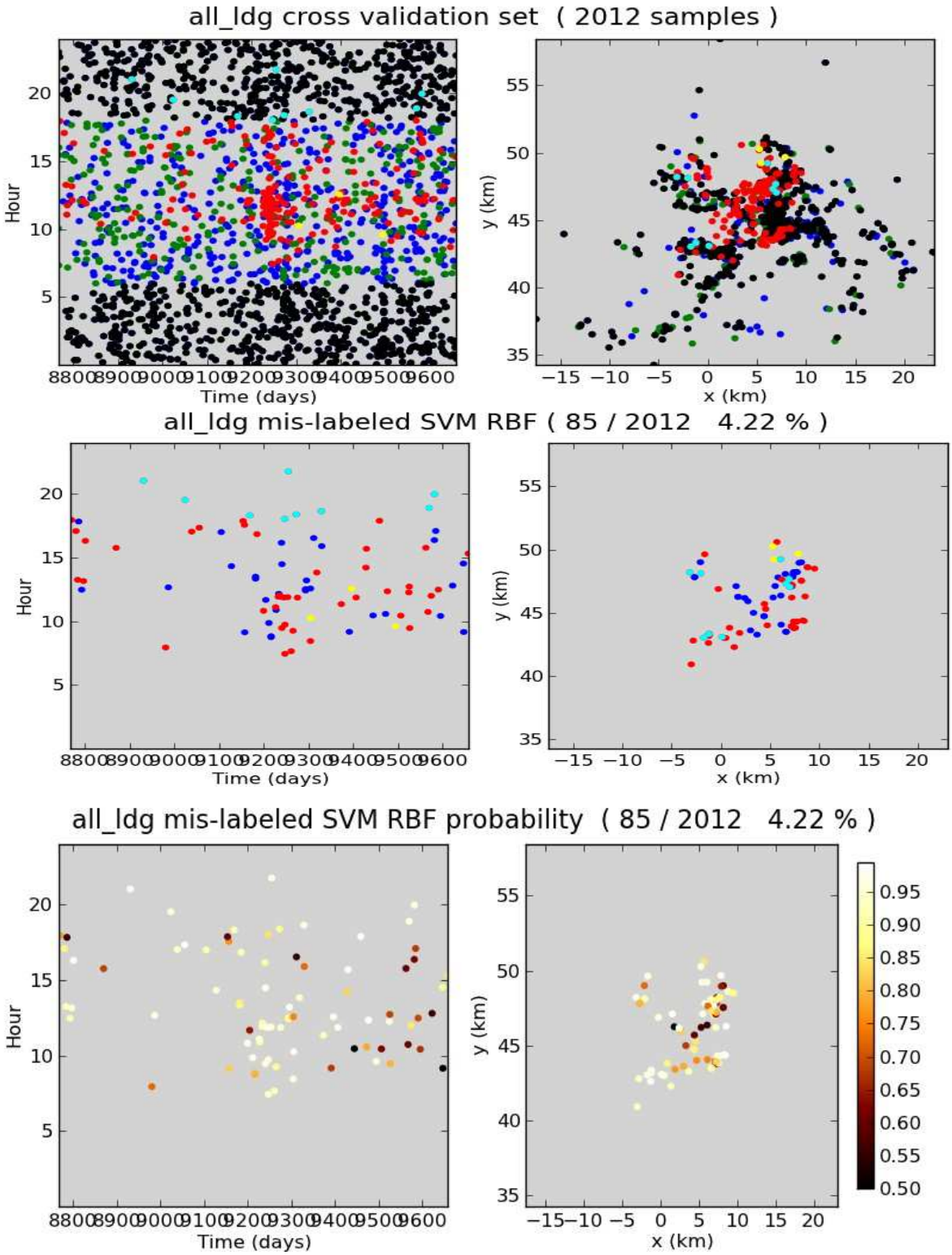


Figure 13: (c & gamma optimales, et test+validation = 0.4)
Diagrammes de dispersion du validation set en temps (à gauche) et en espace (à droite)

Matrice de confusion :

		Classes prédites		
		Classe 'EQ'	**Classe 'blast'**	
Classes Observées	**Classe 'EQ'**	1770	31	1801
	Classe 'blast'	54	157	211
		1824	188	

- le taux générale de bien classés 95.78%
- l'algorithme SVM commit une proportion de 25.6% d'erreur pour les classification des tirs, et 1.75% dans celle des séismes.

On remarque que le taux de bien classés pour les 'EQ' est excellent, par contre il reste à améliorer celui des 'blast' qui représente pour nous« le risque bêta» car on préfère un classifieur qui classe à tort un séisme comme étant une explosion de mine , et qu'on cherche d'avantage à le minimiser.

On peut expliqué l'énorme différence entre les deux erreurs par les hypothèse suivantes:

- le nombre faible les 'blast' dans la base de donnée et par conséquent dans l'échantillon d'apprentissage, la deuxième hypothèse .
- non utilisation de la variable magnitude et la variable profondeur, qui peuvent être très utile dans la reconnaissance des 'blast' en raison de leurs faible magnitude, et leurs profondeur relativement basse.

Remarque : Ces hypothèses feront l'objet de la seconde partie de l'étude.

Question : est ce que la proportion d'erreur 4.22% est uniforme sur l'hexagone ?

Pour répondre à cette question on va parcourir l'hexagone de l'ouest vers l'est en utilisant une boucle sur la longitude, on calcule l'erreur et en incrémente la valeur de la longitude, ainsi pour chaque étape on construit implicitement un sous catalogue d'évènement situés dans les intervalles de longitude et latitude respectivement [-5 , i] , [42; 51] pour i dans {-2, -3, ...,10} on test notre classificateur et on enregistre le taux d'évènement mal classés, on trace la courbe de l'erreur en fonction de l'indice «i», automatiquement pour chaque étape le nombre d'évènement dans le sous-catalogue augmente de façon systématique, les graphes ci-dessous décrits l'opération.

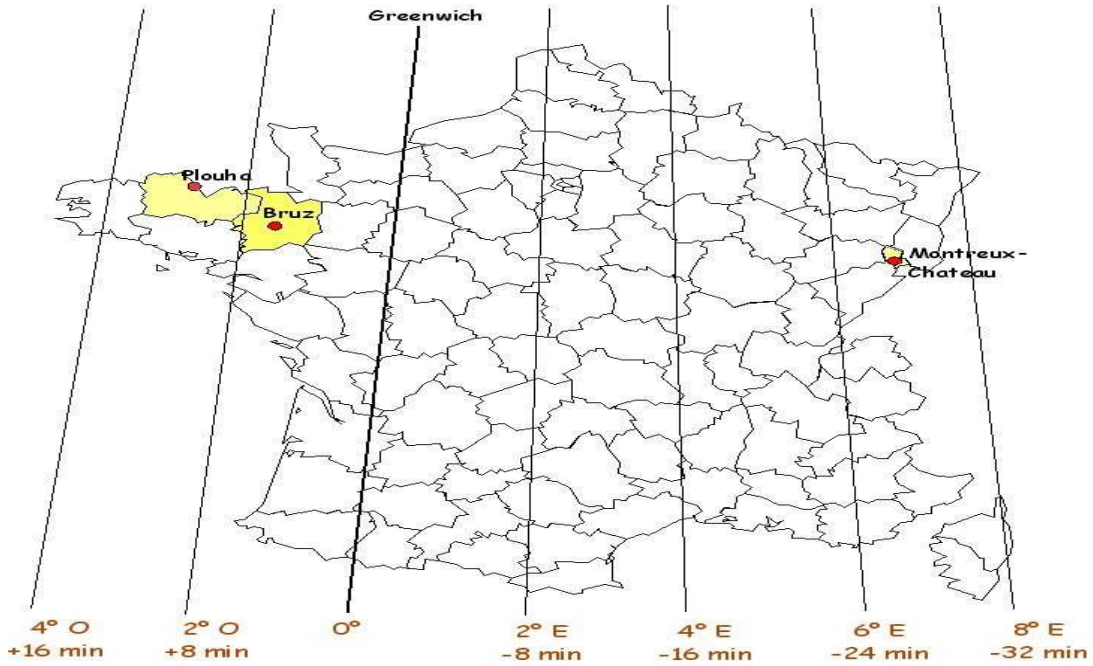


FIGURE 14: les lignes de longitude associé à l'hexagone et les zones frontalières

On trace la courbe du taux d'erreur en fonction de la longitude

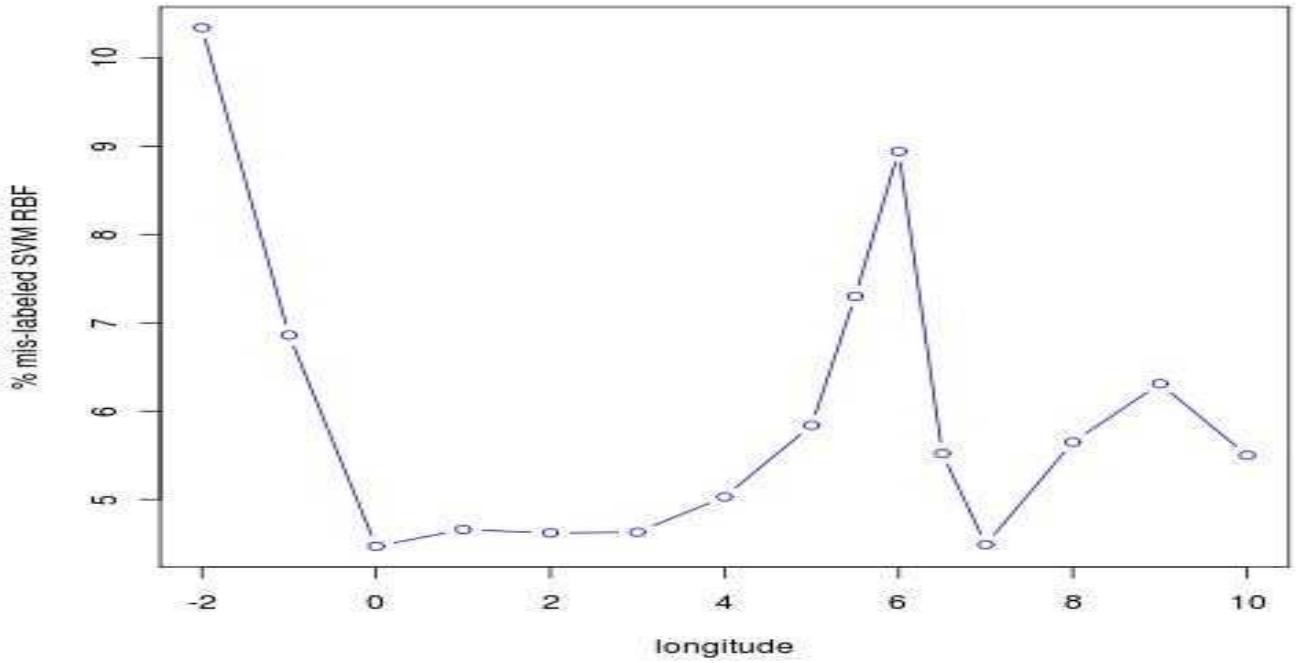


FIGURE 15 : Évaluation de l'erreur en fonction la longitude_max on utilisant l'attitude maximal [42; 51]

On remarque que l'erreur est significativement grand dans l'intervalle [-5, 1] et [5, 6.5]. Pour le premier ceci est explicable par le fait d'insuffisance de données (moins de 1000 évènements), par contre sur le deuxième intervalle, le phénomène mérite une étude plus approfondie.
Dans une seconde étape on fixe l'intervalle de longitude à [5, 6.5], et on parcoure l'hexagone du sud vers le nord mais seulement entre les ligne de longitude E°005 et E°006.5, de façon analogue à l'étape précédente , on trace la courbe de variation de taux l'erreur en fonction de l' attitude

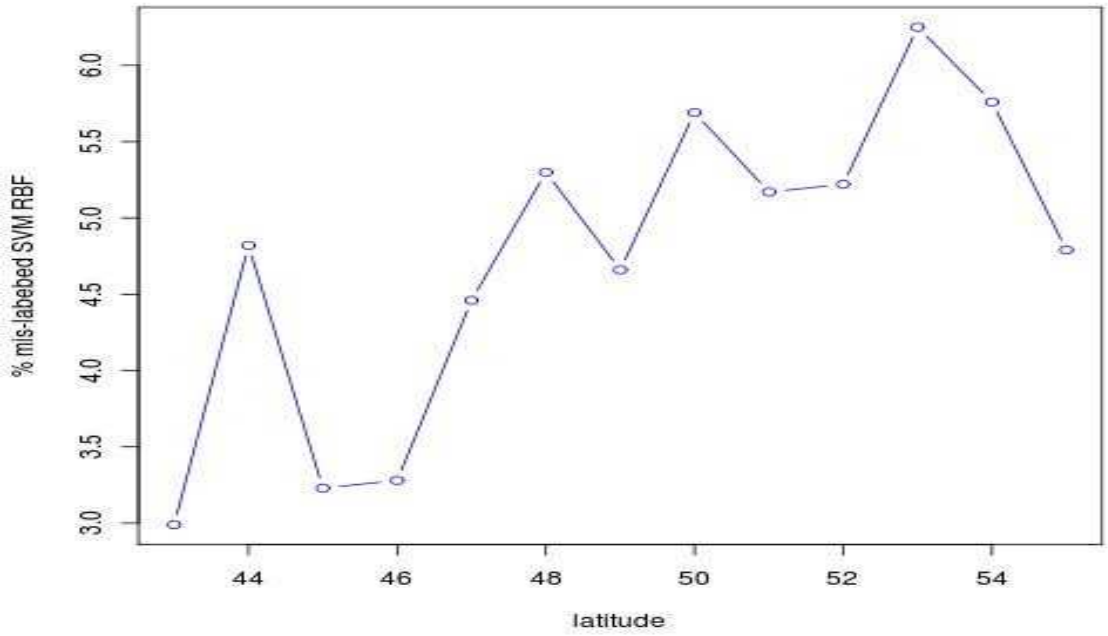


FIGURE 16 : évaluation de l'erreur en fonction latitude_max dans l'intervalle de longitude [4, 7]

En observant la courbe on peut penser que le pavé ou l'algorithme SVM commit le plus d'erreur de classification est définit par les intervalles de longitude et latitude suivants: [4, 7] , [49, 56]
Nous avons choisi d'étudier plus précisément le comportement de notre classificateur dans ce «pavé», pour cela on a construit un nouveau sous-catalogue, comportant les évènement situés dans une fenêtre géographique réduite : entre E004° et E007° en longitude et N49° et N56° en latitude.

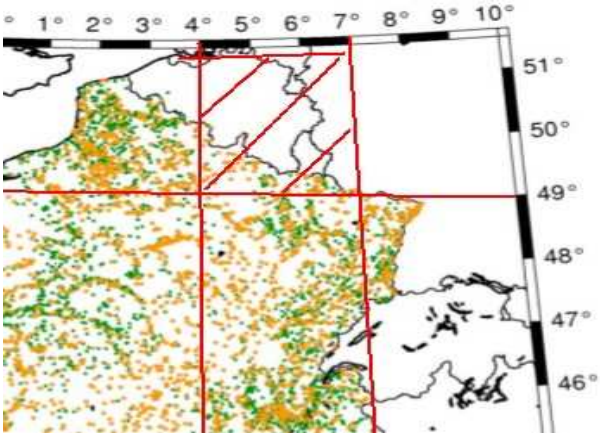


Figure17: Fenêtre géographique E=°[4, 7] * N°=[49, 56]

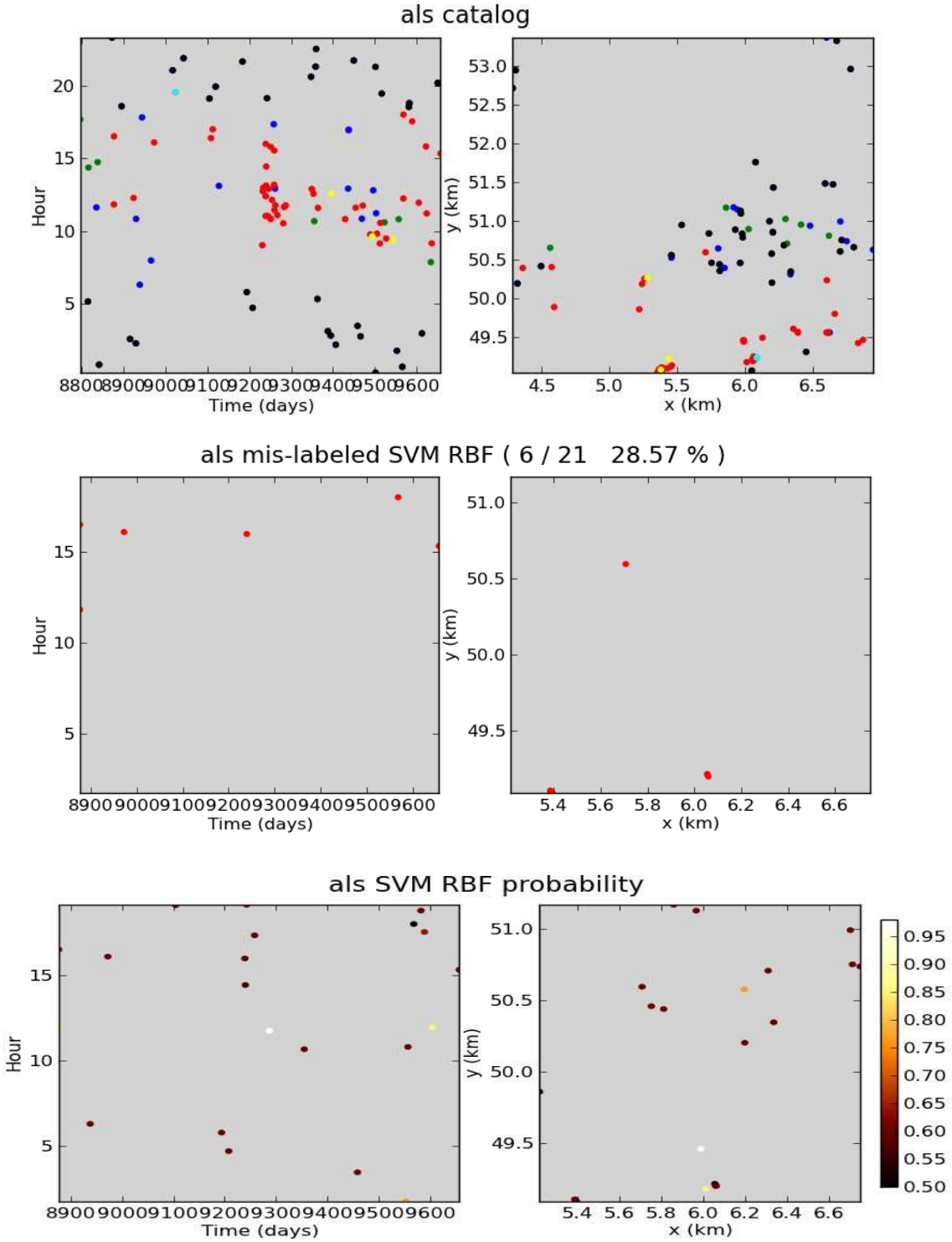


Figure 18: (c & gamma optimales, et test+validation = 0.4 fenêtre géographique long= [4, 7] , lat=[49,56]) Diagrammes de dispersion du validation set en temps (à gauche) et en espace (à droite)

Le résultat de l'algorithme SVM sur cette région nous montre que effectivement le taux des évènements mal classés remonte à 28.57 % qui est énorme par rapport à l'erreur commit par le même l'algorithme sur le catalogue tout entier.
Seules les caractéristiques géographique de cette zone frontalière peuvent apporter une réponse.

3.2 . Classification avec logarithme bayésien naïf:

Dans cette partie on utilisera la méthode bayésienne sur notre base de données, avec le même partitionnement (table ci-dessous):

l'échantillon	Le pourcentage	Nombre d'évènement
Entrainement	60%	6037
Test	20%	2012
Validation	20%	2012

Implicitement:

- L'étape entrainement: consiste à construire deux densités de probabilité (une pour chaque classe)
- L'étape test : pour chaque élément du test, calculer la probabilité d'appartenir à une des deux classes, comparer et étiqueter suivant la probabilité la plus grande
- Le taux l'erreur : Comparer l'étiquetage prédit par l'algorithme et l'étiquetage observé (le vrai)

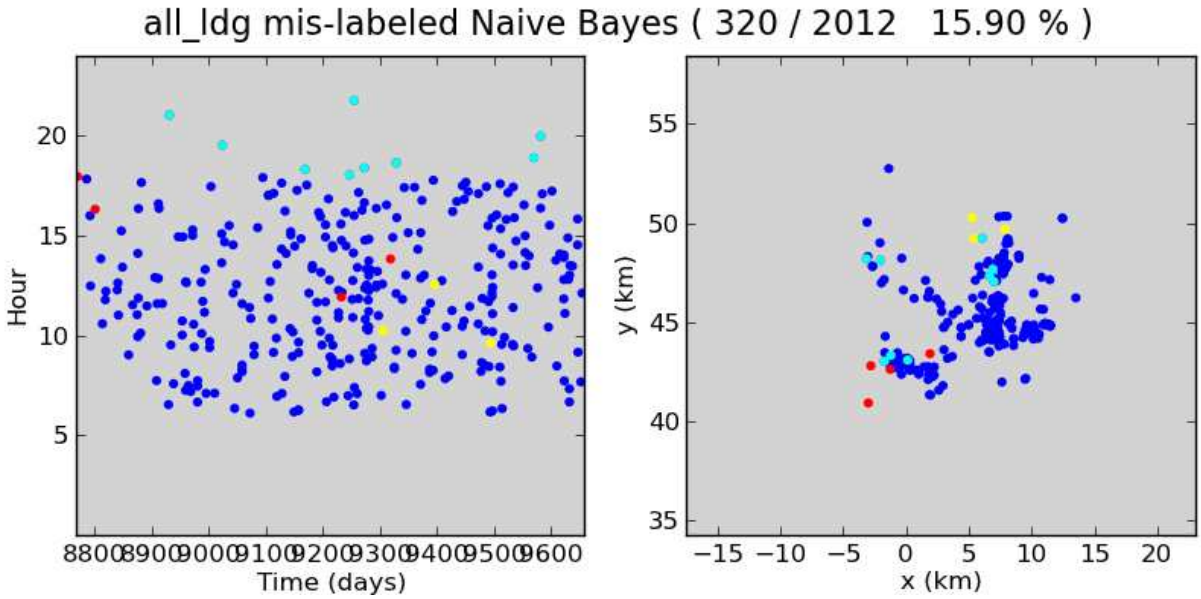


Figure 19 : les évènements mal-classés par le bayésien nait, à gauche en temps, à droite en espace .

Matrice de confusion associe au bayésien naïf :

		Classes prédites		
		Classe 'EQ'	**Classe 'blast'**	
Classes Observées	**Classe 'EQ'**	1497	304	1801
	Classe 'blast'	16	195	211
		1513	499	

- le taux général de bien classés 84.1%

- l'algorithme bayésien naïf commit une proportion de 7.58% d'erreur dans la classification des tirs, et 16.88% pour celle des séismes.

On remarque que le taux générale de bien classées est nettement faible par rapport à la méthode des «support vector machine» par contre, dans la reconnaissance des 'blast' qui représente pour nous « le risque bêta» le bayésien naïf commit moins d'erreur et c'est ce que nous intéresse d'avantage.

Élément de réponse: classifieur "corrompu" ??

On peut ramener cette grande différence entre les taux de bien classée entre les 'blast' et les 'EQ' au fait que les probabilités a priori pour les 'EQ' et 'blast' sont pas égales:

($p(Y=0) = 0.878 \gg P(Y=1) = 0.122$)

Ainsi notre classifieur est encouragé à étiqueter d'avantage les éléments de l'échantillon de test, en tant que 'EQ' .

Pour soigner ce malaise, on est invité dans une prochaine étude à entrainer notre algorithme sur un échantillon d'entrainement dans lequel les classes sont équiprobables

3.3 . classification par la méthode du k plus proches voisins:

Dans cette partie on utilisera la méthode du k plus proches voisins sur notre base de donnée, avec le même partitionnement que l'étude précédente.
Sous Python on a réalisé une boucle sur le nombre de voisins associé au aux d'erreur minimal, le min est atteint pour k=3 voisins.

Implicitement:

- L'étape entrainement: consiste à plonger les éléments de l'échantillon d'entrainement dans espace de dimension 6 (nombre des attribues).

- L'étape test : plonger les éléments de l'échantillon de test dans le même espace, et construire autour de chaque élément du test une boule contenant exactement 3 éléments d'entrainement et ensuite étiqueter le candidat (du test) en fonction d'un 'vote', du sort que le candidat prend la classe majoritaire dans sa boule.

- Le taux l'erreur : Comparer l'étiquetage prédit par l'algorithme et l'étiquetage observé (le vrai)

La figure ci-dessous représente le diagramme du classifieur pour k=3.

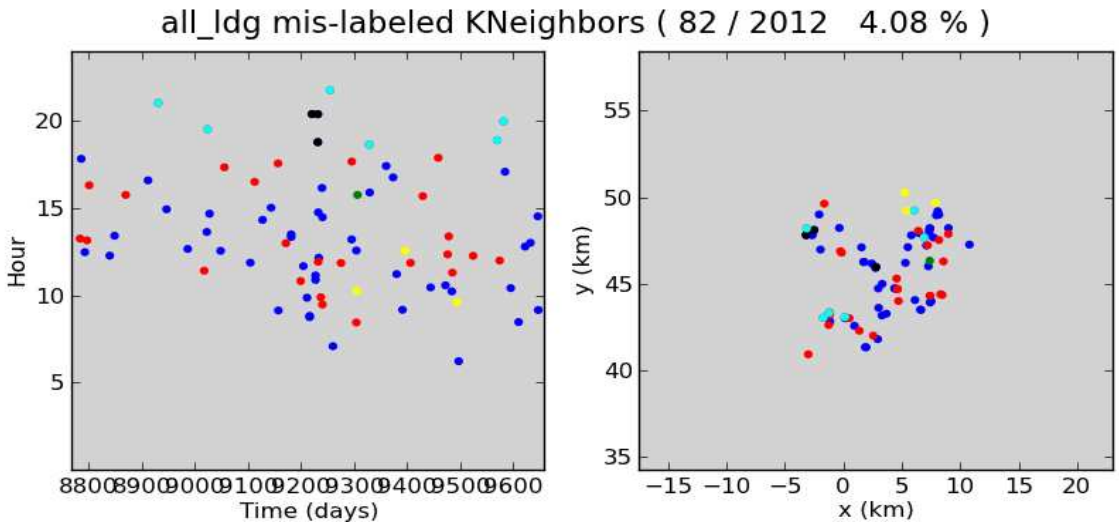


FIGURE 20 : diagramme de l'algorithme du k plus proche voisins

Matrice de confusion :

		Classes prédites		
		Classe 'EQ'	**Classe 'blast'**	
Classes Observées	**Classe 'EQ'**	1752	49	1801
	Classe 'blast'	33	178	211
		1785	227	2012

- le taux général de bien classés 95.92 %

- l'algorithme des k plus proche voisins commit une proportion de 15.64% d'erreur dans la classification des tirs, et 2.72 % pour celle des séismes.

4. Conclusion et propositions:

Le classifieur	Le bayésien naïf	Les SVM	Les k-plus proches voisins
Le taux d'erreur	15.9 %	4.22 %	4.08 %

Dans les résultats de l'algorithme du k-plus proches voisins on remarque que le taux général de bien classées est excellent par rapport aux deux autres classifieurs, par contre en matière de reconnaissance des 'blast' on s'attend à un résultat meilleur vue le caractère récurent des tires, en général ils se passent entre 10h et 18h et dans des endroits géographiques précis et de façon répétitive, donc on avait un pré-sentiment que le classifieur sera plus favorable pour les 'blast'. Ainsi en suivant le mécanisme de décision du k-plus proches voisins, "normalement" un élément de l'échantillon test (un candidat) doit trouver facilement "assez" de 'blast' dans l'entourage qui peuvent voter pour lui. Une explication intuitive de ce problème, serait le fait que les 'blast' sont minoritaires dans la base de données (seulement 12.22%) et par conséquent ils sont minoritaires dans l'échantillon d'entrainement, cette intuition confirme ce qu'on peut voir analytiquement dans les formule du bayésien naïf ($p(Y=0) = 0.878 >> P(Y=1) = 0.122$).
La performance de l'algorithme des SVM en terme de taux d'erreur était assez satisfaisante. Il se rapproche de celui du k-plus proches voisins , mais ce dernier l'emporte par rapport à l'erreur bêta (classer un tire comme étant un séisme).

Durant ce petit projet on a testé 3 classifieurs de natures différentes (probabiliste, linéaire et métrique). Les résultats de ses 3 algorithmes sur la base de donnée convergent vers les conclusions suivantes :

- La domination de la classe 'EQ' sur l'échantillon d'entrainement est un handicape majeur. Ainsi dans les prochaines études on est invités à entrainer le classifieur sur un échantillon dans lequel les classes sont équiprobables

- L'effet du choix de la partition est non négligeable, ainsi il sera préférable de considérer une validation croisée avec une partition 'assez' fine, de sorte qu'on découpe notre base de données en k-échantillons (k assez grand ~ 10) de même poids. On ensuite entraine le classifieur sur un échantillon dans lequel les classes sont équiprobables, on le teste sur les (k-1) échantillons restantes, et on calcule de taux d'erreur, de manière empirique.

- Il faut penser à utiliser les variables explicatives restantes (magnitude, profondeur ….).

Bibliographie :

1. Mathématiques Statistique et Intelligence artificielle

[1] http://wikistat.fr/ Apprentissage Statistique Sous le logiciel R

[2] http://www.math.u-psud.fr/~stafav/IMG/pdf/Appren_stat.pdf, apprentissage statistique

[3] http://scikit-learn.org

[4] http://cs229.stanford.edu/materials.html

[5] coursera.org/course/ml , Andrew NG, Machine Learning

[6] Le choix bayésien: principes et pratique, Christian P.Robert

[7] Apprentissage artificiel : concepts et algorithmes, Antoine Cornuéjols - Laurent Miclet

[8] Chih-Jen Lin , A Guide to Support Vector Machines , 2006

[9] Abdelhamid DJEFFAL , Utilisation des méthodes Support Vector Machine (SVM) dans l'analyse des bases de données , 2012

2.Géophysique:

[9] http://www.usgs.gov/science/

[10] Laurence CORNEZ , Discrimination automatique à base de connaissances expertes d'événements sismiques . 2007

[11] J. Fréchet and F. Thouvenot. Carrières et discrimination sismique. 2012.

[12] Etude des séismes d'origine anthropique dans le Fossé Rhénan, LAMBOTTE Sophie

Annexe A : code python de la fonction 'catalog_io_plot'

```
import os, h5py, string, re, glob
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from dateutil import tz

from_zone = tz.gettz('UTC')
to_zone =   tz.gettz('Europe/Paris')
base_date = datetime(1987,1,1)
base_date = base_date.replace(tzinfo=from_zone)
features = "x (km),y (km),Time (days),Hour,day/night,week"
labels = {'earthquake' : 0, 'blast' : 1}

def write_catalog_hdf5(X, y, filename):

    n_s, n_f =  X.shape

    f=h5py.File(filename,'w')
    X_data=f.create_dataset('X',data=X)
    y_data=f.create_dataset('y',data=y)
    X_data.attrs['features'] = features
    y_data.attrs['earthquake'] = labels['earthquake']
    y_data.attrs['blast'] = labels['blast']
    f.close()

def read_catalog_hdf5(filename):

    f=h5py.File(filename,'r')
    X=np.array(f['X'])
    y=np.array(f['y'])
    features = f['X'].attrs['features'].split(',')
    labels={}
    for key, value in f['y'].attrs.iteritems():
        labels[key] = value
    f.close()

    return X, y, features, labels

def write_catalogs_hdf5(eq_cat, blast_cat, filename):

    X = np.vstack((eq_cat,blast_cat))
    n_eq, f_f = eq_cat.shape
    n_s, n_f =  X.shape
    y = np.ones(n_s)
    y[0:n_eq]=0

    f=h5py.File(filename,'w')
```

```
    X_data=f.create_dataset('X',data=X)
    y_data=f.create_dataset('y',data=y)
    X_data.attrs['features'] = features
    y_data.attrs['earthquake'] = labels['earthquake']
    y_data.attrs['blast'] = labels['blast']
    f.close()


def read_isc_catalog(filename):

    # read file
    f=open(filename,'r')
    lines=f.readlines()
    f.close()

    # set up storage
    ns = len(lines)
    x = np.empty(len(lines), dtype=float)
    y = np.empty(len(lines), dtype=float)
    m = np.empty(len(lines), dtype=float)
    t = np.empty(len(lines), dtype=float)
    h = np.empty(len(lines), dtype=float)
    d = np.empty(len(lines), dtype=int)
    w = np.empty(len(lines), dtype=int)

    i=0
    for line in lines:
        words = line.split()
        #print words

        # get longitude and latitude (interpret as x and y for now)
        # TODO : project onto a local coordinate system
        lon = float(words[2])
        lat = float(words[3])
        x[i]=lon
        y[i]=lat

        # parse date and time strings
        year = int(words[0][0:2])
        if year > 70 :
            year = year + 1900
        else :
            year = year + 2000
        month = int(words[0][2:4])
        day   = int(words[0][4:6])
        hour  = int(words[1][0:2])
        mins  = int(words[1][2:4])
        sec   = int(words[1][4:6])
        try:
            usec  = int(words[1][7:9])*10000
        except ValueError :
            usec = 0
```

```
        # get origin time in UTC
        otime=datetime(year, month, day, hour, mins, sec, usec)
        otime = otime.replace(tzinfo=from_zone)

            # get time-related features
            t[i],h[i],d[i],w[i] = set_time_features_from_otime(otime)

        i=i+1

    X = np.vstack((x,y,m,t,h,d,w))
    return X.T

def read_chooz_catalog(filename):

    # read file
    f=open(filename,'r')
    lines=f.readlines()
    f.close()

    # set up storage
    ns = len(lines)
    x = np.empty(len(lines), dtype=float)
    y = np.empty(len(lines), dtype=float)
    m = np.empty(len(lines), dtype=float)
    t = np.empty(len(lines), dtype=float)
    h = np.empty(len(lines), dtype=float)
    d = np.empty(len(lines), dtype=int)
    w = np.empty(len(lines), dtype=int)

    i=0
    for line in lines:
        words = line.split(',')
        #print words

        # get longitude and latitude (interpret as x and y for now)
        # TODO : project onto a local coordinate system
        lat = float(words[2])
        lon = float(words[3])
        mag1 = float(words[8])
        x[i]=lon
        y[i]=lat
        m[i]=mag1
        # parse date and time strings
        # get origin time in UTC
        otime=datetime.strptime(words[0]+' '+words[1], '%Y-%m-%d %H:%M:%S.%f')
        otime = otime.replace(tzinfo=from_zone)

            # get time-related features
            t[i],h[i],d[i],w[i] = set_time_features_from_otime(otime)

        i=i+1
```

```
    X = np.vstack((x,y,m,t,h,d,w))
    return X.T

def read_gse2_cat(filename):

    # read file
    f=open(filename,'r')
    lines=f.readlines()
    f.close()

    # count the number of events

    n_ev = 0
    for line in lines :
        if line.strip().startswith('EVENT'):
            n_ev += 1

    x = np.empty(n_ev, dtype=float)
    y = np.empty(n_ev, dtype=float)
    m = np.empty(n_ev, dtype=float)
    t = np.empty(n_ev, dtype=float)
    h = np.empty(n_ev, dtype=float)
    d = np.empty(n_ev, dtype=int)
    w = np.empty(n_ev, dtype=int)
    y_class = np.empty(n_ev, dtype=int)

    i=0

    for line in lines :
        # skip empty lines
        if len( line.strip() ) == 0 :
            continue
        # if the first column is not empty
        if len( line[0].strip() ) > 0:
            # check if it is a date
            match_str = r'\d{4}/\d{2}/\d{2}\s+'
            if re.match(match_str, line.strip()):
                    #print line
                # it is a date - this is the origin of the event
                    words = line.split()

            # parse date and time strings
            # get origin time in UTC
            otime=datetime.strptime(words[0]+' '+words[1], '%Y/%m/%d %H:%M:%S.%f')
            otime = otime.replace(tzinfo=from_zone)

                    # get time-related features
                    t1,h1,d1,w1 = set_time_features_from_otime(otime)

                    # lat et lon et magnitude
                    # Note : do not use word decomposition as extraenous characters may be present
```

```
            lat_text = line[25:33]
            lon_text = line[34:43]
        mag1_text = line[75 :77]
            # Some events may be detected but not located - must ignore them completely
            if len(lat_text.strip()) == 0 or len(lon_text.strip()) == 0 or len(mag1_text.strip()) == 0:
                ignore_next_type = True
            else :
                lat = float(line[25:33])
                lon = float(line[34:43])
          mag1 = float(line[75:78])
                ignore_next_type = False

    else :
      if not(line.strip().startswith('Date') or line.strip().startswith('rms') or line.strip().startswith('_ldg')):
        # TODO : Do parsing here
            words = line.split()
            ev_type = words[-1]
            if ignore_next_type :
                continue
        if ev_type == 'ke' or ev_type == 'km' :
                x[i] = lon
                y[i] = lat
          m[i] = mag1
                t[i] = t1
                h[i] = h1
                d[i] = d1
                w[i] = w1
                if ev_type == 'ke' :
            y_class[i] = labels['earthquake']
                else :
                      y_class[i] = labels['blast']
                i += 1
            else :
              continue
# There are now i elements in the vectors. So resize to length i.
x.resize(i)
y.resize(i)
m.resize(i)
t.resize(i)
h.resize(i)
d.resize(i)
w.resize(i)
y_class.resize(i)

X = np.vstack((x,y,m,t,h,d,w))
return X.T, y_class

# TODO : return catalog as for other inputs
```

```
def read_gse2_cat_from_directory(dir):
    X_list = []
    y_list = []
    files = glob.glob(dir + os.sep + '*.txt')
    for filename in files :
            #print filename
            X, y = read_gse2_cat(filename)
            X_list.append(X)
            y_list.append(y)

    y_final = np.hstack((y_list))
    X_final = np.vstack((X_list))

    return X_final, y_final



def read_mine_cat(filename):

    f=open(filename,'r')
    lines=f.readlines()
    f.close()

    n_mines=len(lines[2:-1])

    mines={}
    mine_id=np.empty(n_mines, dtype=int)
    mine_xy=np.empty((n_mines,2),dtype=float)
    mine_cp=[]
    mine_name=[]

    i=0
    for line in lines[2:-1]:
        words = line.split('\t')
        mine_id[i] = int(words[0])
        mine_xy[i,0] = float(words[7])/1000.0
        mine_xy[i,1] = float(words[8])/1000.0
        mine_name.append(words[1])
        mine_cp.append(words[4])
        i=i+1
    mines['ID'] = mine_id
    mines['CP'] = mine_cp
    mines['name'] = mine_name
    mines['xy'] = mine_xy

    return mines

def plot_catalog(X, y, features, labels, title, filename, ranges=None):

    eq_label = labels['earthquake']
    bl_label = labels['blast']
```

```
eq_cat =    X[y == eq_label]
blast_cat = X[y == bl_label]

fig = plt.figure(figsize=(9.5, 4))
ax1=fig.add_subplot(121, axisbg='lightgrey')
ax2=fig.add_subplot(122, axisbg='lightgrey')
#ax3=fig.add_subplot(223, axisbg='lightgrey')
#ax4=fig.add_subplot(224, axisbg='lightgrey')

x_eq, y_eq, m_eq,t_eq, h_eq, d_eq, w_eq = np.hsplit(eq_cat,7)
x_blast, y_blast, m_blast,t_blast, h_blast, d_blast, w_blast = np.hsplit(blast_cat,7)

w_eq = np.around(w_eq.flatten())
d_eq = np.around(d_eq.flatten())
w_blast = np.around(w_blast.flatten())
d_blast = np.around(d_blast.flatten())


ax1.scatter(eq_cat[w_eq==1,2],eq_cat[w_eq==1,3],marker='o', color='b', linewidths=(0,))
ax1.scatter(eq_cat[w_eq==0,2],eq_cat[w_eq==0,3],marker='o', color='green', linewidths=(0,))
ax1.scatter(eq_cat[d_eq==0,2],eq_cat[d_eq==0,3],marker='o', color='black', linewidths=(0,))
ax1.scatter(blast_cat[w_blast==1,2],blast_cat[w_blast==1,3], marker='o',color='red',  linewidths=(0,))
ax1.scatter(blast_cat[w_blast==0,2],blast_cat[w_blast==0,3], marker='o',color='yellow', linewidths=(0,))
ax1.scatter(blast_cat[d_blast==0,2],blast_cat[d_blast==0,3], marker='o',color='cyan', linewidths=(0,))
ax1.set_xlabel(features[2])
ax1.set_ylabel(features[3])
if ranges:
   ax1.set_xlim(ranges[0][2], ranges[1][2])
   ax1.set_ylim(ranges[0][3], ranges[1][3])

#ax2.plot(eq_cat[w_eq==1,0],eq_cat[w_eq==1,3],'b.')
#ax2.plot(eq_cat[w_eq==0,0],eq_cat[w_eq==0,3],'g.')
#ax2.plot(eq_cat[d_eq==0,0],eq_cat[d_eq==0,3],'k.')
#ax2.plot(blast_cat[w_blast==1,0],blast_cat[w_blast==1,3],'r.')
#ax2.plot(blast_cat[w_blast==0,0],blast_cat[w_blast==0,3],'.',color='yellow')
#ax2.plot(blast_cat[d_blast==0,0],blast_cat[d_blast==0,3],'.',color='cyan')
#ax2.set_xlabel(features[0])
#ax2.set_ylabel(features[3])

#ax3.plot(eq_cat[w_eq==1,2],eq_cat[w_eq==1,1],'b.')
#ax3.plot(eq_cat[w_eq==0,2],eq_cat[w_eq==0,1],'g.')
#ax3.plot(eq_cat[d_eq==0,2],eq_cat[d_eq==0,1],'k.')
#ax3.plot(blast_cat[w_blast==1,2],blast_cat[w_blast==1,1],'r.')
#ax3.plot(blast_cat[w_blast==0,2],blast_cat[w_blast==0,1],'.',color='yellow')
#ax3.plot(blast_cat[d_blast==0,2],blast_cat[d_blast==0,1],'.',color='cyan')
#ax3.set_xlabel(features[2])
#ax3.set_ylabel(features[1])

ax2.scatter(eq_cat[w_eq==1,0],eq_cat[w_eq==1,1], marker='o', color='b', linewidths=(0,))
ax2.scatter(eq_cat[w_eq==0,0],eq_cat[w_eq==0,1],marker='o',color='green', linewidths=(0,))
ax2.scatter(eq_cat[d_eq==0,0],eq_cat[d_eq==0,1],marker='o',color='black', linewidths=(0,))
ax2.scatter(blast_cat[w_blast==1,0],blast_cat[w_blast==1,1],marker='o', color='red', linewidths=(0,))
```

```
    ax2.scatter(blast_cat[w_blast==0,0],blast_cat[w_blast==0,1],marker='o',color='yellow', linewidths=(0,))
    ax2.scatter(blast_cat[d_blast==0,0],blast_cat[d_blast==0,1],marker='o',color='cyan', linewidths=(0,))
    ax2.set_xlabel(features[0])
    ax2.set_ylabel(features[1])
    if ranges:
        ax2.set_xlim(ranges[0][0], ranges[1][0])
        ax2.set_ylim(ranges[0][1], ranges[1][1])

    fig.subplots_adjust(hspace=4)

    plt.suptitle(title, fontsize=16)

    #plt.show()
    plt.savefig(filename)
    plt.clf()


def plot_prob(X, X_prob, features, labels, title, filename, ranges=None):

    fig = plt.figure(figsize=(9.5, 4))
    ax1=fig.add_subplot(121, axisbg='lightgrey')
    ax2=fig.add_subplot(122, axisbg='lightgrey')
    #ax3=fig.add_subplot(223, axisbg='lightgrey')
    #ax4=fig.add_subplot(224, axisbg='lightgrey')

    x, y, m, t, h, d, w = np.hsplit(X,6)
    prob=np.max(X_prob, axis=1)

    w = np.around(w.flatten())
    d = np.around(d.flatten())


    ax1.scatter(t.flatten(),h.flatten(),c=prob, marker='o', linewidths=(0,), cmap='afmhot')
    ax1.set_xlabel(features[2])
    ax1.set_ylabel(features[3])
    if ranges:
        ax1.set_xlim(ranges[0][2], ranges[1][2])
        ax1.set_ylim(ranges[0][3], ranges[1][3])

    cs=ax2.scatter(x.flatten(),y.flatten(),c=prob, marker='o', linewidths=(0,), cmap='afmhot')
    ax2.set_xlabel(features[0])
    ax2.set_ylabel(features[1])
    if ranges:
        ax2.set_xlim(ranges[0][0], ranges[1][0])
        ax2.set_ylim(ranges[0][1], ranges[1][1])

    fig.colorbar(cs, ax=ax2, shrink=0.9)
    fig.subplots_adjust(hspace=4)

    plt.suptitle(title, fontsize=16)

    #plt.show()
```

```
    plt.savefig(filename)
    plt.clf()

def dot2(u, v):
    return u[0]*v[0] + u[1]*v[1]

def cross2(u, v, w):
    """u x (v x w)"""
    return dot2(u, w)*v - dot2(u, v)*w

def ncross2(u, v):
    """|| u x v ||^2"""
    return sq2(u)*sq2(v) - dot2(u, v)**2

def sq2(u):
    return dot2(u, u)


def plot_mines(mines_dict, title, filename):
    from scipy.spatial import Delaunay
    from matplotlib.collections import LineCollection

    fig = plt.figure(figsize=(9.5,4))
    ax1=fig.add_subplot(121, axisbg='lightgrey')
    ax2=fig.add_subplot(122, axisbg='lightgrey')

    # plot scatterplot of mines
    x=mines_dict['xy'][:,0]
    y=mines_dict['xy'][:,1]
    ax1.scatter(x,y,marker='.',color='red', linewidths=(0,))
    ax1.set_xlim(np.min(x), np.max(x))
    ax1.set_ylim(np.min(y), np.max(y))

    # plot voronoi cells
    tri=Delaunay(mines_dict['xy'])
    p = tri.points[tri.vertices]

    # Triangle vertices
    A = p[:,0,:].T
    B = p[:,1,:].T
    C = p[:,2,:].T

    # See http://en.wikipedia.org/wiki/Circumscribed_circle#Circumscribed_circles_of_triangles
    # The following is just a direct transcription of the formula there
    a = A - C
    b = B - C

    cc = cross2(sq2(a) * b - sq2(b) * a, a, b) / (2*ncross2(a, b)) + C

    # Grab the Voronoi edges
    vc = cc[:,tri.neighbors]
    vc[:,tri.neighbors == -1] = np.nan # edges at infinity, plotting those would need more work...
```

```
    lines = []
    lines.extend(zip(cc.T, vc[:,:,0].T))
    lines.extend(zip(cc.T, vc[:,:,1].T))
    lines.extend(zip(cc.T, vc[:,:,2].T))

    # Plot it

    lines = LineCollection(lines, edgecolor='k', linewidth=0.2)

    #ax2.plot(points[:,0], points[:,1], '.')
    #ax2.plot(cc[0], cc[1], '.')
    ax2.set_xlim(np.min(x), np.max(x))
    ax2.set_ylim(np.min(y), np.max(y))
    plt.gca().add_collection(lines)


    plt.suptitle(title, fontsize=16)
    plt.savefig(filename)
    plt.clf()

def set_time_features_from_otime(otime):
    """
    Requires origin time in UTC. Returns t, h, d, w features.
    """

    # get origin time in local time
    otime_local = otime.astimezone(to_zone)
    #print otime.isoformat(' '), otime_local.isoformat(' ')

    # get t, fractional days since base_date
    t=((otime - base_date).days*86400 + (otime - base_date).seconds) / float(86400)

    # use local time to get h
    h=(otime_local.hour*3600+otime_local.minute*60+otime_local.second)/float(3600)


    # set day/night flag
    # for now cheat, and use 6-18h as daytime
    # TODO : use ephemerides to set day or night flag
    if h > 6.0 and h < 18.0 :
        d = 1
    else :
        d = 0

    # set weekday flag
    wday = otime_local.weekday()
    if wday < 5 :
        w = 1
    else :
        w = 0
```

```
    return t, h, d, w


if __name__ == '__main__' :

    fname = '../CATALOGS/fiches_new_act.csv'
    mines = read_mine_cat(fname)
    plot_mines(mines, 'Mine short-list after J Frechet', 'Frechet_mines.png')
```

Annexe B : code python de la fonction 'run_ldg_discrim'

```
import os
import numpy as np
from sklearn import preprocessing, cross_validation, svm
from sklearn.decomposition import PCA
from sklearn.metrics import precision_recall_fscore_support
from sklearn.naive_bayes import GaussianNB
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import KFold
from syn_catalog import *
from catalog_io_plot import *


############################
# read and pre-process data
############################

# eq = 0
# blast = 1
# sinn : filename = "/home/nabil/discrim-master/CATALOGS/ldg_all.hdf5"
filenames=['ldg.hdf5']
base_titles=['all_ldg']
C_values=[10000, 10, 10000]
gamma_values=[0.01, 0.1, 0.01]

for filename, base_title, C, gamma in zip(filenames, base_titles, C_values, gamma_values) :
    print filename, base_title

    basename,ext=os.path.splitext(filename)
    X, y, features, labels = read_catalog_hdf5(filename)

    # do feature scaling
    scaler = preprocessing.StandardScaler()
    scaler.fit(X)
    X_scaled = scaler.transform(X)
```

```
    # split out a training set
    X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_scaled, y,
test_size=0.4,random_state=0)  #div en 2 test et train
    X_cv, X_test, y_cv, y_test = cross_validation.train_test_split(X_test, y_test, test_size=0.5,
random_state=0)  # div le test en 2 xtest et xcv
    n_cv=len(y_cv)
    n_train=len(y_train)
    min_cv=np.min(scaler.inverse_transform(X_cv), axis=0)
    max_cv=np.max(scaler.inverse_transform(X_cv), axis=0)

    plot_fname=basename + '_train.png'
    title=base_title + ' training set' + '  ( %d samples )'%n_train
    plot_catalog(scaler.inverse_transform(X_train), y_train, features, labels, title, plot_fname,
ranges=(min_cv, max_cv))

    plot_fname=basename + '_cv.png'
    title=base_title + ' cross validation set' + '  ( %d samples )'%n_cv
    plot_catalog(scaler.inverse_transform(X_cv), y_cv, features, labels, title, plot_fname,
ranges=(min_cv, max_cv))

    ###########################
    # do a naive bayes analysis
    ###########################
    gnb = GaussianNB()
    y_pred = gnb.fit(X_train, y_train).predict(X_cv)

    p,r,f,s = precision_recall_fscore_support(y_cv, y_pred)
    #print "Precision for Naive Bayes : ", p
    #print "Recall for Naive Bayes    : ", r
    print "F1 for Naive Bayes        : ", f
    n_false=np.sum(y_cv != y_pred)
    frac_false=n_false/float(n_cv)
    print "Number of mis-labeled points : %d"% n_false

    X_false = X_cv[y_cv != y_pred]
    y_false = y_cv[y_cv != y_pred]

    plot_fname=basename + '_NB.png'
    title = base_title+' mis-labeled Naive Bayes'+' ( %d / %d   %.2f %% )'%(n_false,n_cv,frac_false*100)
    plot_catalog(scaler.inverse_transform(X_false), y_false, features, labels, title, plot_fname,
ranges=(min_cv, max_cv))
```

```
    ############################
    # do a SVM
    ############################
    C=1.0
    gamma=1.0
    svc = svm.SVC(kernel='rbf',C=C, gamma=gamma, probability=True)
    svc.fit(X_train, y_train)
    y_pred=svc.predict(X_cv)
    X_prob=svc.predict_proba(X_cv)

    p,r,f,s = precision_recall_fscore_support(y_cv, y_pred)
    #print "Precision for SVM : ", p
    #print "Recall for SVM    : ", r
    print "F1 for SVM        : ", f
    n_false=np.sum(y_cv != y_pred)
    frac_false=n_false/float(n_cv)
    print "Number of mis-labeled points : %d"% n_false
    print X.shape
    X_false = X_cv[y_cv != y_pred]
    y_false = y_cv[y_cv != y_pred]

    X_prob_false=svc.predict_proba(X_false)

    plot_fname=basename + '_svm_rbf.png'
    title = base_title+' mis-labeled SVM RBF'+' ( %d / %d   %.2f %% )'%(n_false,n_cv,frac_false*100)
    plot_catalog(scaler.inverse_transform(X_false), y_false, features, labels, title, plot_fname,
ranges=(min_cv, max_cv))

    plot_fname=basename + '_svm_rbf_prob.png'
    title = base_title+' SVM RBF'+' probability'
    plot_prob(scaler.inverse_transform(X_cv), X_prob, features, labels, title, plot_fname,
ranges=(min_cv, max_cv))

    plot_fname=basename + '_svm_rbf_false_prob.png'
    title = base_title+' mis-labeled SVM RBF'+' probability  ( %d / %d   %.2f %% )'%
(n_false,n_cv,frac_false*100)
    plot_prob(scaler.inverse_transform(X_false), X_prob_false, features, labels, title, plot_fname,
ranges=(min_cv, max_cv))


# do grid search
#print "doing grid search"
#C_range = 10.0 ** np.arange(-2, 5)
```

```
#gamma_range = 10.0 ** np.arange(-3,3)
#param_grid = dict(gamma=gamma_range, C=C_range)
#grid = GridSearchCV(svm.SVC(), param_grid=param_grid )
#grid.fit(X_train, y_train)
#print("The best classifier is: ", grid.best_estimator_)

#('The best classifier is: ', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
gamma=1.0,kernel='rbf', max_iter=-1, probability=False, shrinking=True, tol=0.001, verbose=False))

#('The best classifier is: ', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
gamma=1.0,kernel='rbf', max_iter=-1, probability=False, shrinking=True, tol=0.001,verbose=False))
```