Ecole Polytechnique
Promotion X2012
RIOCHET Ronan

RESEARCH INTERNSHIP REPORT

Structural change detection in generalized linear models - AXA GLOBAL DIRECT

Département de Mathématiques appliquées
MAP594: Modélisation probabiliste et statistique
Directeur de stage : Stéphane Gaïffas
Maître de stage : Constance Dumaine
Dates du stage: 16/03/2015 - 24/07/2015
Adresse de l'organisme:
AXA GLOBAL DIRECT
48 rue Carnot
91150 Suresnes

Déclaration d'intégrité relative au plagiat

Je soussigné Ronan Riochet certifie sur l'honneur:

- Que les résultats décrits dans ce rapport sont l'aboutissement de mon travail.
- Que je suis l'auteur de ce rapport.
- Que je n'ai pas utilisé des sources ou résultats tiers sans clairement les citer et les référencer selon les règles bibliographiques préconisées.

mention à recopier

Je déclare que ce travail ne peut être suspecté de plagiat.

Date: Signature:

Contents

Acknowledgement

First of all I would like to thank Axa Global Direct for welcoming me for this internship and giving me a great chance for learning and professional development. I would like to express my particular gratitude to André Weilert and Joanna Chardon for welcoming me in their team.

My deepest thanks go to Constance Dumaine and Aimé Lachapelle. Constance was a great tutor and gave me the necessary advices and guidance to succeed in this internship. She was also very present and careful all along these months. I thank Aimé for encouraging me joining the team for this internship and am sincerely grateful for the precious advices he gave me when I needed.

Finally I would like to thank Clara, Henri, Kevin, Guillaume and Mattia for their help and all the interesting discussions we shared, along with all the members of the team for the great times we spent together.

1 Introduction

In the context of use of time-continuous data, time consistency of models is a burning question. The main objective of this internship was to find a way to detect structural changes of our models in real time. In the first time, the idea was to find a way to monitor a model and to detect when it was obsolete. This would allow one to keep the model up-to-date, refitting it as soon as it is not consistent anymore. The second objective was to analyse the structural change so as to link it to external events and better understand the market.

The second objective was to build a *Competitive Conversion model* and apply the theory of structural change detection to it. This model was a supervised classification model, aimed to infer on prospects' conversion[1] given competition variables. Several methods were tested and the final model was used to test the theory on structural change detection.

This report is split into three sections. The first section gather the theoretical parts dealing with time consistency and structural change detection. The second section deals with the *Competitive Conversion model* and how to apply the previous theory to it. Finally the last one provides an overview of the implementation of all the procedure in Python, followed by two case studies.

[1] When a potential client subscribes to an insurance, hence becoming a client.

2 Theoretical part

2.1 Test for one structural change

Structural change detection has been introduced by Gregory Chow in 1960 for standard linear models, and adapted later for generalized linear models. In both cases, the model is split into two distinct models, before and after the break point. This split model, called alternative model, is then compared to the null model fitted on the entire period. Since the null model is a special case of the alternative, the alternative model always leads to better results[2]. The principle of the following tests is to compare results of the two models and see if this difference is significant.

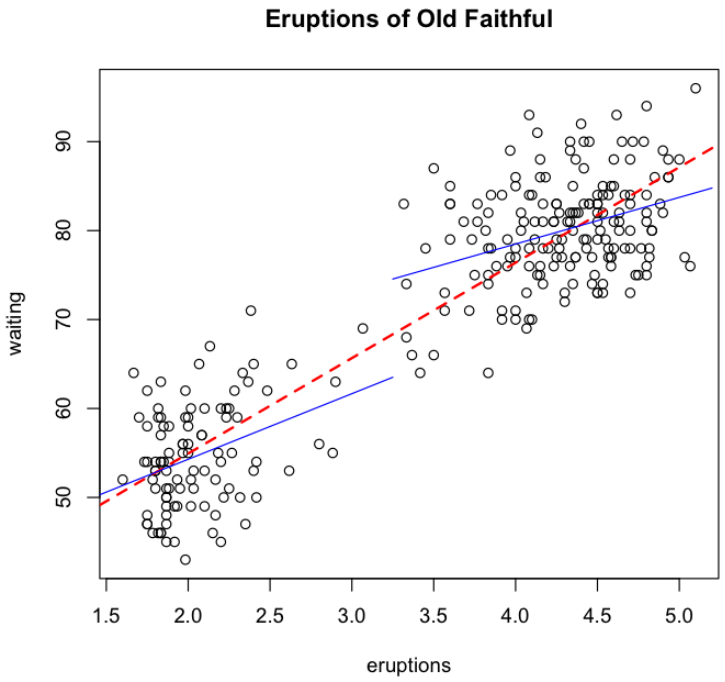

Figure 1: Structural change example: Old Faithful's eruptions. In red the null model, in blue the alternative model (source: https://thetarzan.wordpress.com/2011/06/16/the-chow-test-in-r-a-case-study-of-yellowstones-old-faithful-geyser/

.

[2]In the optimization problems, the variable space of the null model is included in the variable space of the alternative model.

2.1.1 Standard linear regression: the Chow test

Let $(Y_i)_{i=1,...,n}$ be a set of observations and $(X_i)_{i=1,...,n}$ explanatory variables such that:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i,\ i = 1, ..., n$$

The data is split into two subsets $(Y_i^{(1)}, X_i^{(1)})_{i=1,...,n^{(1)}}$ and $(Y_i^{(2)}, X_i^{(2)})_{i=1,...,n^{(1)}}$ such that :

$$\begin{aligned} Y_i^{(1)} &= \beta_0^{(1)} + \beta_1 X_i^{(1)} + \epsilon_i,\ i = 1, ..., n^{(1)} \\ Y_i^{(2)} &= \beta_0^{(2)} + \beta_1 X_i^{(2)} + \epsilon_i,\ i = 1, ..., n^{(2)} \end{aligned}$$

Let d be the number of parameters, S the sum of squared residuals for the combined data, $S^{(1)}$ and $S^{(2)}$ the sum of squared residuals for the subsets 1 and 2, respectively. Under the hypothesis $H_0 : \beta^{(1)} = \beta^{(2)}$, the Chow test statistic:

$$\frac{S - S^{(1)} - S^{(2)}}{S^{(1)} + S^{(2)}} \frac{n - 2d}{d}$$

follows the F-distribution with d and $n - 2d$ degrees of freedom. Hence, this statistic provides a test to detect a structural dissimilarity between the two subsets. To detect a structural change over time we simply take

$$\begin{aligned} (Y_i^{(1)}, X_i^{(1)})_{i=1,...,n^{(1)}} &= (Y_i, X_i)_{i=1,...,k} \\ (Y_i^{(2)}, X_i^{(2)})_{i=1,...,n^{(2)}} &= (Y_i, X_i)_{i=k+1,...,n} \end{aligned}$$

2.1.2 Generalized linear model: the Likelihood-ratio test (or F-test)

Likelihood-ratio test Consider two nested models[3], a null and an alternative model, such that the null model is included in the alternative model. Since the estimates are computed via maximum-likelihood and the null model is included in the alternative model, the alternative model will end up with a greater likelihood. The likelihood ratio test is designed to test if this difference is significant.

The likelihood-ratio test statistic (often denoted by D) is twice the difference in these log-likelihoods:

$$D = -2ln\left(\frac{likelihood\ for\ the\ null\ model}{likelihood\ for\ the\ alternative\ model}\right)$$

D = -2ln(likelihood for the null model) + 2ln(likelihood for the alternative model)

Under the null hypothesis that the two models are equal, D follows a Chi-squared distribution with degrees of freedom ($d_{alternative} - d_{null}$), the difference of numbers of parameters between the two models.

[3]Two statistical models are nested if the first model can be transformed into the second model by imposing constraints on the parameters of the first model.

One can use this statistic to test if a new variable should be included in a model. The null model is the model without the variable, whereas the alternative model takes the new variable into account. If the new variable increases significantly the likelihood, we can reject the null hypothesis H_0: "$\beta_{new\ variable} = 0$"

Application to structural change We can use the same test to test for structural change. Let the null model be the model fitted on the all period, and the alternative model (or split model) be the model fitted independently on the first and the second period.

The D statistic is:

$$D = -2ln\left(\frac{\prod_{i=1}^{n} L(X_i,Y_i,\beta)}{\prod_{i=1}^{k} L(X_i,Y_i,\beta^{(1)}) * \prod_{i=k+1}^{n} L(X_i,Y_i,\beta^{(2)})}\right)$$

where k is the break point, $\beta^{(1)}$ and $\beta^{(2)}$ computed before and after the break point. Under the null hypothesis of no change, D follows Chi-squared distribution with degrees of freedom equal to d, dimension of β.

Hence, in a model with d parameters where the previous likelihood ratio is D at time t, the p-value $cdf^{-1}_{\chi 2(d)}(D)$ is the risk of type-I error, which is detecting a break at t when there is no break.

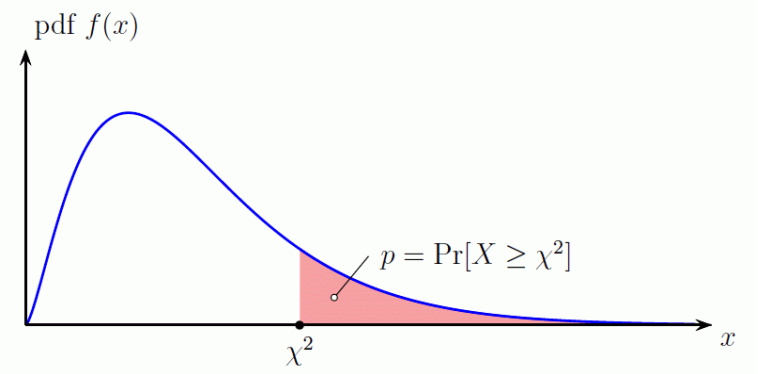

Figure 2: Chi squared distribution for df=10 (source: www.di-mgt.com.au/chisquare-calculator.html)

2.1.3 Dynamic detection

To detect such a change dynamically, we may want to choose a sample size $m \in N$ and each time m new observations arrive, apply F-test to subsets $(Y_i, X_i)_{i=1,\ldots,n}$ and $(Y_i, X_i)_{i=n+1,\ldots,n+m}$. However, this procedure would lead to an increase in risk of type-I error.

Indeed, suppose that H_0 is always true (no change in the model) and that observations are independent enough to admit that events {"type-I error at time k", k>0} are independent. After n periods, the probability of having rejected H_0 at least once will be: $1 - (1 - \alpha)^n \to 1$.

As an example, in *Monitoring Structural Change* (1996), Chu et al. propose the following experiment:

> *In the simulation, we generate data from* $y_t = 0.6x_t + \epsilon_t$ *where* ϵ_t *is niid(0,1) and* x_t *is an AR(1) with the AR coefficient 0.8. The in-sample size is 270; nominal size is 5%. The number of replications is 4,000.*
>
> *[To detect structural change at a given time] we wait for (say) 5 periods, then perform a post-sample F test. If the F test is passed, we update the model by including these 5 new data points and wait for another 5 periods and repeat the post-sample F test; otherwise, we signal a rejection of stability.*
>
> *Simulation results show that thirty periods later,' we have a one-third chance of mistakenly signaling instability. The probability of type one error increases to 70% one hundred periods later. If the data are collected daily, ten months later we will wrongly reject the true hypothesis of stability more than 95% of the time.*

This shows that the likelihood-ratio test is not well-suited for monitoring structural change as new data arrive. To do so we would like to build a continuous-time process describing fluctuations, which can be controlled over time.

Sequential analysis provides techniques for monitoring time series' fluctuations, initially used in quality control theory (Wald's Sequential Probability Ratio Test, 1945). One of these techniques is the CUSUM algorithm. Given a "quality number" θ and observations $(y_t)_{t\geq 0}$ distributed under $p(\theta)$, the CUSUM algorithm proposes a criterion to determine changes in θ. The principle of this algorithm is to control the cumulative sum of the difference between $(y_t)_{t\geq 0)}$ and a target value.

To detect structural change in a linear regression model, an approach provided by Chu et al. is to apply CUSUM algorithm to the linear regression model. In that case, θ is the estimates of the model and the target value is the prediction of the model. Hence, it consists in monitoring the cumulative sum of the empirical errors of the model.

2.2 Dynamic monitoring for linear models

Consider the standard linear regression model:

$$y_i = x_i^T \beta_i + u_i, \text{ for } i = 1, ..., m+1, ...$$

where y_i is the observation, x_i is a $d \times 1$ vector of regressors, with the first component equal to unity, β_i is the $d \times 1$ vector of regression coefficients and u is an i.i.d. and centered error term.

Consider also the two following assumptions on the observations:

ASSUMPTION A.2:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} X_i X_i' = R, \ a.s.$$

where R is a $(d \times d)$ real, non-sigular, symmetric matrix.

This assumption is true if the variables are linearly independent.

ASSUMPTION A.3: The residuals u_i are stationary with

$$\mathbb{E}[u_i|\mathcal{T}_i] = 0 \text{ and } Var(u_i|\mathcal{T}_i) = \sigma^2,$$

where $\mathcal{T}_i$ is the σ-field generated by $\{Y_j, X_j, u_j | s \leq i\}$

Under these assumptions we will be able to construct a fluctuation process which we can control over time. To do so we consider a stable period, in which we know there is no structural change. This is the "non-contamination" assumption:

ASSUMPTION NC : $\beta_i = \beta_0, \forall i = 1, 2, ..., m$

We fit the model on this stable period and start monitoring after the m^{th} observation. Hence, in the following, we keep on testing null hypothesis:

$$H_0 : \beta_i = \beta_0, \ i > m$$

This is a three-step procedure:

- We compute $\{\psi_i\}_{i>m}$, an i.i.d. and centered random variable describing the fluctuations of our model.
- We construct the standardized cumulative sum of $(\psi_i)_i$
- We control this sum as a time depending process

In our case, a good ψ is simply the empirical error term u. The following theorems build a fluctuation process from this ψ.

Theorem 1. ***(Donsker Theorem)*** *If $X_1, X_2, ...$ are independent and identically distributed with mean 0 and variance σ^2, the random function Y_n defined by*

$$S_m(t,\omega) = \frac{1}{\sigma\sqrt{m}}\left\{ \sum_{i=1}^{\lfloor mt \rfloor} \psi_i(\omega) \right\} + (mt - \lfloor mt \rfloor)\frac{1}{\sigma\sqrt{m}}\psi_{\lfloor mt \rfloor + 1}(\omega),\ 0 \le t \le 1$$

converges to the Wiener process W weakly.

Corollary 1. *Under H_0, if $Var(\psi(Y,\beta,X))$ is non-singular:*

$$Var(\psi(Y,\beta^{(m)},X))^{-\frac{1}{2}} m^{-\frac{1}{2}} \sum_{i=1}^{\lfloor mt \rfloor} \psi(Y_i,\beta^{(m)},X_i) \xrightarrow{unif} W(t)$$

where $W(\cdot)$ is a Wiener process.

In the two previous propositions, β is the true parameter which is unknown in our case. However we have the following result:

Theorem 2. *If $\widehat{Var}(\psi(Y,\hat{\beta}^{(m)},X))$ is non-singular:*

$$\widehat{Var}(\psi(Y,\hat{\beta}^{(m)},X))^{-\frac{1}{2}} m^{-\frac{1}{2}} \sum_{i=1}^{\lfloor mt \rfloor} \psi(Y_i,\hat{\beta}^{(m)},X_i) \xrightarrow{unif} W(t) - tW(1)$$

Proof. As said before, the following we have the following assumption

ASSUMPTION A.2:

$$\lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} X_i X_i' = R,\ a.s.$$

where R is a $(d \times d)$ real, non-singular, symmetric matrix.

R being symmetric, it is orthogonally diagonalizable, so we can write

$$\begin{aligned} &\lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} X_i X_i' = P'DP,\ a.s. \\ \Rightarrow \quad &\lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} \frac{PX_i}{\sqrt{D_{(1,1)}}} \left(\frac{PX_i}{\sqrt{D_{(1,1)}}} \right)' = \tilde{D},\ a.s. \end{aligned}$$

where $\tilde{D}$ is diagonal, $\tilde{D}_{(1,1)} = 1$ and $P'P = PP' = I_d$. Hence, we assume that the model has been reparameterized such that:

$$R = \begin{bmatrix} 1 & 0 \\ 0 & R^* \end{bmatrix}$$

which implies that $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{m} X_i = [1, 0, ..., 0]'$.

ASSUMPTION A.3: The residuals u_i are stationary with

$$\mathbb{E}[u_i|\mathcal{T}_i] = 0 \text{ and } Var(u_i|\mathcal{T}_i) = \sigma^2,$$

where $\mathcal{T}_i$ is the σ-field generated by $\{Y_j, X_j, u_j | s \leq i\}$

Rewrite ψ_i as $\psi_i = u_i - X_i'(\hat{\beta} - \beta)$ and $B^m(t) = \frac{1}{\hat{\sigma}\sqrt{m}} \sum_{i=1}^{[mt]} u_i - X_i'(\hat{\beta}^{(m)} - \beta)$.

$$\hat{\sigma} B^{(m)}(t) \quad = \quad \underbrace{m^{-\frac{1}{2}} \sum_{i=1}^{[mt]} u_i}_{(*)} \quad - \quad \underbrace{m^{-\frac{1}{2}} \sum_{i=1}^{[mt]} X_i'(\hat{\beta}^{(m)} - \beta)}_{(**)}$$

First of all, it is a well known fact that for i.i.d. residuals $(u_i)_{i=1,...n}$ we have:

$$(*) \xrightarrow{unif} \sigma W(t)$$

To proof that $(**) \xrightarrow{unif} tW(1)$, consider:

$$n^{-\frac{1}{2}} \sum_{i=1}^{[mt]} X_i'(\hat{\beta}^{(m)} - \beta) \; = \; \left[\frac{1}{m} \sum_{i=1}^{[mt]} X_i'\right] \cdot \left[\sqrt{m}(\hat{\beta}^{(m)} - \beta)\right]$$

From ASSUMPTION A.2 we have $\left[n^{-\frac{1}{2}} \sum_{i=1}^{[mt]} X_i'\right] \to [t, 0, ..., 0]$ as $n \to \infty$. We can also express $\left[\frac{1}{m}(\hat{\beta}^{(m)} - \beta)\right]$ as

$$\begin{aligned}
\left[\sqrt{m}(\hat{\beta}^{(m)} - \beta)\right] &= \sqrt{m}\left[(\sum_{i=1}^{m} X_i X_i')^{-1}(\sum_{i=1}^{m} X_i Y_i) - \beta\right] \\
&= \sqrt{m}(\sum_{i=1}^{m} X_i X_i')^{-1}\left[(\sum_{i=1}^{m} X_i Y_i) - (\sum_{i=1}^{m} X_i X_i')\beta\right] \\
&= m^{-\frac{1}{2}}\left[\frac{1}{m}\sum_{i=1}^{m} X_i X_i'\right]^{-1}\left[\sum_{i=1}^{m} X_i(Y_i - X_i'\beta)\right] \\
&= n^{-\frac{1}{2}}\left[\frac{1}{m}\sum_{i=1}^{m} X_i X_i'\right]^{-1}\begin{bmatrix}\sum_{i=1}^{n} X_i^{(1)} u_i \\ \sum_{i=1}^{n} X_i^{(*)} u_i\end{bmatrix} \\
&= n^{-\frac{1}{2}}\begin{bmatrix}1 & 0 \\ 0 & R^*\end{bmatrix}^{-1}\begin{bmatrix}\sum_{i=1}^{m} u_i \\ \sum_{i=1}^{m} X_i^{(*)} u_i\end{bmatrix} + o_p(1)
\end{aligned}$$

Which leads to $(**) \; = \; t\left\{m^{-\frac{1}{2}} \sum_{i=1}^{m} u_t\right\} + o_p(1)$ and proves the theorem. $\square$

2.2.1 Generalized linear models

In generalized linear models, nothing says that the residuals satisfy the FCLT (in a logistic regression model for example we can't state $\mathbf{E}[\hat{u}_i] = 0$). However, GLMs are computed via maximum likelihood estimation:

$$\cancel{\min_{\beta} \sum_i (Y_i - \beta X_i)^2} \;\rightarrow\; \min_{\beta} -\sum_i^n \ell(X_i, Y_i, \beta)$$

where $\ell(X_i, Y_i, \beta)$ is the log-likelihood of the i^{th} observation for a given β. The idea in the following is to take the derivative of the cost function:

$$\cancel{\psi_i = \hat{u}_i} \rightarrow \;\psi_i = -\nabla_\beta \ell(X_i, Y_i, \beta)$$

Hence, we know that the empirical mean of $(\psi_i)_{i=1\dots n}$ is zero. Indeed the opposite of the log-likelihood is convex and coercive so at the minimum:

$$\nabla_\beta \ell(X, Y, \beta) = 0 \;\Rightarrow\; \sum_i \psi_i = 0 \;\Rightarrow\; \hat{\mathbf{E}}[\psi] = 0$$

The previous theorems apply in this case. In *Tests for Constancy of Model Parameters Over Time* (2002), Hjort and Koning provides a proof of the for M-score processes.

2.2.2 Boundary functions

We would like to build boundary functions such that, under the null hypothesis of no change, the path of S_n crosses with prescribed probability. We will use the following result from *Boundary Crossing of Brownian Motion* by Lerche (1984):

$$P\left\{ |W(\tfrac{t}{t-1})| \geq \sqrt{\tfrac{t}{t-1}(\lambda^2 + ln\tfrac{t}{t-1})} \right\} = 2\big(1 - \Phi(\lambda) + \lambda\phi(\lambda)\big),\; t > 1$$

Suppose that $W(\cdot)$ is a standard Wiener process, then $X(t) = (t-1)W(\frac{t}{t-1})$ is a Brownian bridge on $]1, \infty[$. To prove it we simply show that $X(\cdot)$ is a Gaussian process, with $E(X) = 0$, $Cov(X(t), X(s)) = min(s,t) - st$ and $\lim_{t\to 0} X(t) 0$.

Let $W(\cdot)$ be a standard Wiener process. The following crossing probability holds for W:

$$P\left\{ |W(s)| \geq \sqrt{s(\lambda^2 + ln(s))} \right\} \;=\; 2\big(1 - \Phi(\lambda) + \lambda\phi(\lambda)\big)$$

Replacing s by $\frac{t}{t-1}$, it follows that:

$$\begin{aligned}
P\left\{ |W(\tfrac{t}{t-1})| \geq \sqrt{\tfrac{t}{t-1}(\lambda^2 + ln\tfrac{t}{t-1})} \right\} &= 2\big(1 - \Phi(\lambda) + \lambda\phi(\lambda)\big) \\
P\left\{ (t-1)|W(\tfrac{t}{t-1})| \geq \sqrt{t(t-1)(\lambda^2 + ln\tfrac{t}{t-1})} \right\} &= 2\big(1 - \Phi(\lambda) + \lambda\phi(\lambda)\big) \\
P\left\{ |B(t)| \geq \sqrt{t(t-1)(\lambda^2 + ln\tfrac{t}{t-1})} \right\} &= 2\big(1 - \Phi(\lambda) + \lambda\phi(\lambda)\big)
\end{aligned}$$

where $B(\cdot)$ is a Brownian bridge.

As a consequence, if ψ's mean moves significantly from 0, the algorithm will detect a change after a certain number of observations. In that case, the delay between structural change and detection depends on both the change and the λ. See Figure 3 for an example.

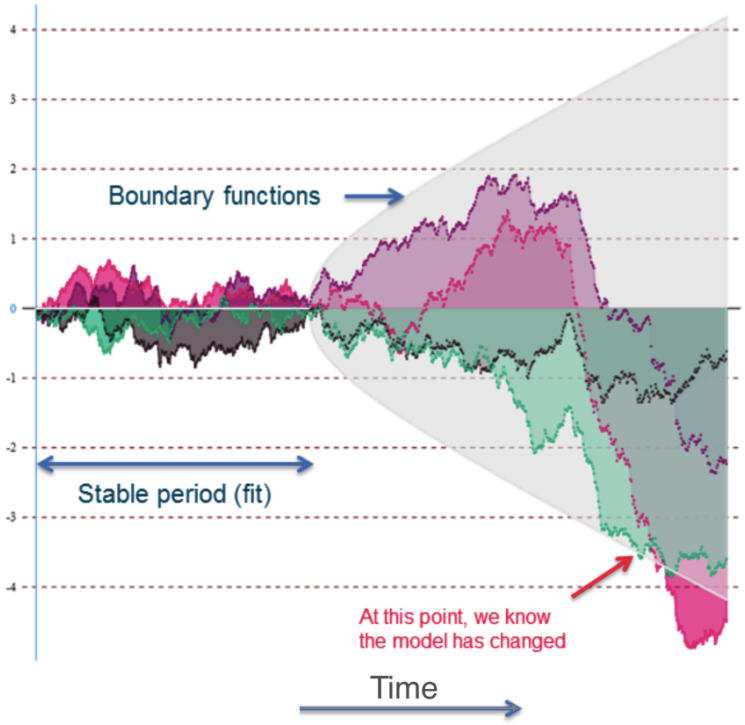

Figure 3: Example of use of empirical fluctuation process, d=4

2.2.3 Limits

After a structural change, the distribution of ψ changes and the so the distribution of its standardized cumulative sum. By controlling the cumulative sum of ψ and not ψ in itself, we lower the risk of type-I error. It is as if we waited for a significant number of 'abnormal' observations before rejecting the null hypothesis of no change. The limit of this method is that there must be a delay between the structural change and its detection.

The second limit is just a consequence of the first one: after a structural change the model's structure may move back to the initial one before the empirical process crosses the boundary function. In that case, the previous procedure wouldn't detect this temporary change.

2.3 Partitioning

We have seen a statistic allowing the user to detect when the model is obsolete. However, this approach makes difficult to infer the precise break point. Indeed, at the time we detect something, the only information we have is that there has been one or several structural change(s) since the end of the fitting period (the non-contamination period). This can be enough in the case where we only want to keep our model updated. However, here we would like to know the precise date when the change occurs since it is likely to be linked to external events.

Now that we have a time interval in which we know there has been at least one break, we can test all the dates as if they were break points and keep the date leading to the lowest p-value. In the case where there has been only one break point this procedure will detect the break point and hence solves our problem.

However, as we saw earlier, when the empirical fluctuation process detects something, there might have already been several structural changes. The F-test still works for several break points. We split the data into n+1 subsets $(X^{(1)}, ..., X^{(n+1)})$ and estimates $(\beta^{(1)}, ..., \beta^{(n+1)})$ on each subsets. The likelihood-ratio statistic is then:

$$D = -2ln\left(\frac{\prod_{i=1}^{n} L(X_i,Y_i,\beta)}{\prod_{i=1}^{b_1} L(X_i,Y_i,\beta^{(1)}) * ... * \prod_{i=b_k}^{n} L(X_i,Y_i,\beta^{(k+1)})}\right)$$

Under the null hypothesis, D follows a chi square distribution with nd degrees of freedom.

If the number of breaks is known, for example n, we just have to compute the likelihood-ratio statistic for all the possible k-uplets of break points. The k-uplet leading to the highest likelihood-ratio is the one we will keep as solution of our problem. This partitioning is the one that improves the most the likelihood of our model. Unfortunately, most of the time it is difficult to infer on the precise number of structural change.

In that case, we have to compare different partitions for different numbers of break points. The likelihood of the optimal partition increases with the number of break points. Indeed, every partition with k break points is a special case of another partition with k+1 break points. Since GLMs are fitted via maximum likelihood estimation, the likelihood of the optimum increases each time a break point is added. However, each time a break point is added, the number of parameters increases and so the degree of freedom of D. As you can see on figure 4, the size of the tail increases with the degrees of freedom. As a consequence, even though the likelihood-ratio statistic always increases with the number of breaks, the p-value of the associated test will decrease until a minimum and re-increase after that.

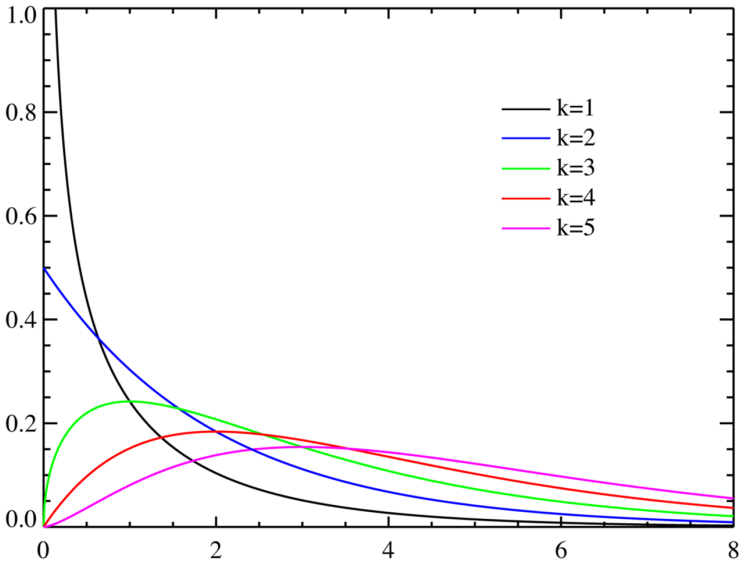

Figure 4: Chi squared pdf for different degrees of freedom (source: en.wikipedia.org/ wiki/Chi-squared_distribution)

Hence brute force, consisting in choosing a maximal number of breaks n and testing all partitions with k breaks, for k up to n, allows us to infer for the most likely break points. If the boundary functions for the empirical fluctuation process are well-chosen, the delay between the break point and its detection should be small, and so the number of break points in the interest period. In the conversion model on which we applied this procedure, we usually found one or two break points at this step.

The main limit of this approach is the complexity of this optimization problem. The number of operations if $o((n+1)^k)$ where n is the number of observations and k is the number of break points. Indeed the number of likelihood-ratio to compute is equivalent to n^k, and the number of operations for each likelihood-ratio is equivalent to n.

To deal with this problem, the first idea is to reduce the number of break points tested. Remember that all our observations are sorted by time and that we test each one to see if it is likely to be a break points. Instead of testing all observations, we can decide to test only N points evenly distributed in the period. In that case the complexity is $o(n * N^k)$. Finally, in their article *Computation and analysis of multiple structural change models (2003)*, Jushan BAI and Pierre PERRON propose to use dynamic programming to keep a complexity as low as $o(n^2)$. The idea is to reuse previous results each time a new likelihood-ratio statistic is computed. During this internship I didn't implement this algorithm because we kept $n = 10^4$, $k < 3$, $N = 30$ which leads to

an approximated number of operations equal to $10^4 \sum_{k=1}^{3} \binom{k}{30} \approx 4 \cdot 10^7$, easily computable by a computer.

2.4 Penalization to describe significant changes

Now that the precise date of the break point(s) is known, we would like to understand the change that occurred. The next step is then to compute the estimates before and after the break point, excluding or not a small period around it. The danger here is to observe insignificant moves of estimates which would make difficult the understanding of the change. To avoid this, we introduce a new penalization that links the estimates of the first and second period. Hence, we minimize the following loss function

$$\min_{\beta_1,\beta_2} \text{likelihood}(X^{(1)}, Y^{(1)}, \beta^{(1)}) + \text{likelihood}(X^{(2)}, Y^{(2)}, \beta^{(2)}) + \frac{r}{2}||\beta^{(2)} - \beta^{(1)}||^2$$

where $X^{(i)}$, $Y^{(i)}$ and $\beta^{(i)}$ are variables, labels and estimates of the i^{th} period.

More precisely, in a given period we suppose $l-1$ break points, such that P can be divided in l sub-periods $\{P_p\}_{p \leq l}$ with distinct coefficients $\beta^{(1)}, ..., \beta^{(l)} \in R^d$. The loss function becomes:

$$loss = \sum_{p=1}^{l} \left(-\frac{1}{n_p} \sum_{i=1}^{n_p} \ell(X_i^{(p)}, Y_i^{(p)}, \beta^{(p)}) + \frac{\alpha}{2}||\beta^{(p)}||^2 \right) + \frac{r}{2} \sum_{p=1}^{l-1} ||\beta^{(p+1)} - \beta^{(p)}||^2$$

where $\ell(X, Y, \beta) = -\big\{Y_i ln(h(\beta X_i)) + (1 - Y_i) ln(1 - h(\beta X_i))\big\}$, l is the number of periods, n_p the number of observations at period p, α the penalization term and r the *rigidity* term.

The *rigidity* term regularizes the model and prevent insignificant fluctuations of estimates. This way, we insure that the changes in estimates are strong enough to beat a given penalization. It is easy to see that setting $r = 0$ is like computing two distinct estimates on the two periods, whereas setting $r >> 0$ is like computing one single estimate on the whole period.

Hence, we are minimizing a function in $\mathbb{R}^{d \times l}$, convex and coercive. The minimum can be found by descent gradient and the Python module `scipy.optimize` provides two such algorithms: `fmin_ncg` and `fmin_l_bfgs_b`. In both cases the loss function and the loss gradient function (callable Python functions) are given as argument and the algorithm return the minimum. The function `fmin_ncg` also takes the Hessian matrix of the loss function as argument.

The computation of the gradient gives:

$$\nabla_\beta \;=\; -\begin{bmatrix} \sum_{i=1}^{n_1} \nabla_{\beta^{(1)}} \ell(X_i^{(1)}, Y_i^{(1)}, \beta^{(1)}) \\ \vdots \\ \sum_{i=1}^{n_l} \nabla_{\beta^{(l)}} \ell(X_i^{(l)}, Y_i^{(l)}, \beta^{(l)}) \end{bmatrix} + \alpha \begin{bmatrix} \beta^{(1)} \\ \vdots \\ \beta^{(l)} \end{bmatrix} + r\ {}^tA \cdot A \begin{bmatrix} \beta^{(1)} \\ \vdots \\ \beta^{(l)} \end{bmatrix}$$

where $A = I_{d\times l}$−`numpy.eye`$(d \times l, d)$, with `numpy.eye(n, k)` being the $d \times l$ identity matrix with diagonal k indices upper (see figure 5).

```
>>> np.eye(3, k=1)
array([[ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  0.]])
```

Figure 5: part of numpy.eye() documentation

The hessian matrix becomes:

$$H_\beta \;=\; -\begin{bmatrix} \sum_{i=1}^{n_1} H_{\beta^{(1)}} \ell(X_i^{(1)}, Y_i^{(1)}, \beta^{(1)}) & & 0 \\ & \ddots & \\ 0 & & \sum_{i=1}^{n_l} H_{\beta^{(l)}} \ell(X_i^{(l)}, Y_i^{(l)}, \beta^{(l)}) \end{bmatrix}$$

$$+ \alpha \cdot I_{d\times l} \;+\; r \cdot\ {}^tA \cdot A$$

Hence, once we have the gradient and the hessian matrix of the log-likelihood, it is easy to compute the loss function of this new model. The main difference here is the number of variables, the optimization is not in $\mathbb{R}^d$ but in $\mathbb{R}^{d\times l}$. In our case, this dimension shouldn't exceed 10^2, which is acceptable for the functions `fmin_ncg` and `fmin_l_bfgs_b` (the main reason is that operations on matrix of this size take a reasonable time).

3 Application to a logistic regression model

AXA Global Direct SA is a subsidiary of AXA Group, founded in 2008 and based in Suresnes. It offers property and casualty insurance solutions. It primarily offers motor and household insurance products online and through call centers in France, Poland, Belgium, Italy, Portugal, Spain, South Korea, and Japan.

The theory described above has been applied to a real model for Direct Seguros, the Spanish entity of AXA for direct[4] insurance. More precisely, the model we studied was the *Competitive Conversion model*, which analyses how prospects (potential client asking for a quote) convert to clients, regarding to our price and the price of competitors.

3.1 Structural change in the *Competitive Conversion model*

Thanks to a rich database, we are able to model demand in the Spanish motor insurance market. Modeling demand has two goals. Firstly, using logistic regression allows us to interpret coefficient as odd probabilities and understand what determines demand. Secondly, monitoring the model helps understanding the market an allows the company to adjust its pricing or its marketing strategy.

Theoretically, a perfect demand model would include an almost infinite number of variables into account. Of course personal information and prices (our prices and competitors' prices) are very important but it would also include some subjective features such as brand value, reputation, reliability, popularity, etc. (for Direct Seguros but also for all other competitors). Obviously we can't measure and integrate all these variables to one single model. Hence, and as in every mathematical model, we create a simpler model with only available and pertinent features. Thus, when one or several hidden variables change, the conversion rate evolution is not explained by our model. In this new situation, estimates changes which lead to what we call a "structural change" of our model. Those structural changes should be detected as soon as possible.

Figure 6: Perfect model (left) and Competitive Conversion model (right)

[4]That is to say on internet only.

Watching conversion rate fluctuation and logistic model fluctuation are two different approaches. Since conversion is strongly linked to profits, conversion rate monitoring is frequent in every online business. Our approach here is to monitor logistic model, in order to consider the underlying structure of demand. For example, all other things being equal, a slight price decreasing of one competitor will directly decrease our conversion rate but, if this move does not imply any structural change of demand, our new estimator will not detect anything. However, some changes in a competitor's pricing policy may cause a structural change which would be detected by our estimator. Such changes may also come from, among other, massive advertising campaigns, re-branding, rumors, etc. Indeed, in these cases, the features' values (e.g. prices and 'biometric' variables) do not change but conversion is modified, which hides structural change of the model.

3.2 Model presentation

There was more than 500 variables in the database but a large part was insignificant, redundant or too sparse to be included in any model. After cleaning the database, there remained about 50 features: 20 dealing with prospects' profile and 30 with competitors (log-ratios between Direct Seguros and each competitor). The main goal of this model is to analyse demand in terms of competition, so I was asked to keep only price variables. The model being a supervised classification model, I firstly tried several famous classification methods like Random Forest, AdaBoost, SVM and logistic regression. The three last methods gave similar results, with a ROC AUC between 70 and 85%, depending on variables included in the model. Even though Random Forest gave slightly better results, it provides very few understanding of the market. Logistic regression results are readable, estimates shows which competitors have a big impact on the market, and as a GLM the previous theory on structural changes can be applied.

The next step was feature engineering. I standardized price variables so that estimates can be compared and interpreted as the impact of the competitor on demand. I also created new variables such as the log-ratio with the min, the 2^{nd} and the 3^{rd} min, the rank of Direct Seguros among all the prices, the log-ratio with the mean, the standard deviation of prices, etc. Finally, "cross-tabulation" was added: products of variables and squared variables, to observe cross-effects of these features. This last step did not give good results so I did not go on with it (note also that prices ratios we introduced are already sort of cross-tabulation).

For feature selection, classic feature selection methods were used, like `sklearn` functions `RandomizedLogisticRegression`, `RFE` and the L1-penalized `LogisticRegression`. After a first selection, I computed a likelihood-ratio test on each variable independently. The principle is the same as described in the first part but it tests the null hypothesis: $H_0 : \ \beta_i = 0$, where β_i is the coefficient of the

variable we want to test. The alternative model, which includes the variable of interest and the null model, which does not, are nested since the null model is a special case of the alternative one. In that case, under H_0, the statistic

$$D = -2ln\left(\frac{likelihood\ without\ the\ variable}{likelihood\ with\ the\ variable}\right)$$

Follows a Chi squared distribution with 1 degree of freedom.

The final model includes five log-ratios with competitors and the log-ratio between Direct Seguros and the average of the three minimum prices (figure 7):

Out[52]:

	Estimate	log(Likelihood-ratio)	p-value
intercept	-0.444489	1842.130912	0.000000e+00
MRK_logratio_avg3min_T2	-0.586416	71.417232	0.000000e+00
MRK_logratio_COMPET1_T2	-0.201363	36.472672	1.548217e-09
MRK_logratio_COMPET2_T2	-0.187417	30.087195	4.130497e-08
MRK_logratio_COMPET3_T2	-0.245457	29.504411	5.578960e-08
MRK_logratio_COMPET4_T2	-0.294086	19.326310	1.101777e-05
MRK_logratio_COMPET5_T2	-0.131494	8.109351	4.403750e-03

Figure 7: Final feature selection, anonymized competitors

The performance of the model is evaluated via several metrics on cross-validation (75%-25%), here are the results (figure 8):

```
In [58]: model.cross_val_score(scoring = 'roc_auc', cv = 4)
Out[58]: array([ 0.73616497,  0.75186862,  0.75537493,  0.7460992 ])


In [71]: model.cross_val_score(scoring = 'accuracy', cv = 4)
Out[71]: array([ 0.60050676,  0.6089527 ,  0.61613176,  0.63540346])


In [69]: model.cross_val_score(scoring = 'recall', cv = 4)
Out[69]: array([ 0.79609544,  0.80694143,  0.80260304,  0.76789588])
```

Figure 8: ROC AUC, accuracy (threshold=0.5) and recall (threshold=0.5)

ROC AUC: Area under the Receiver Operating Characteristic. The ROC plots true positive rates against false positive rates, the closer AUC comes to 1, the better the model is.
Accuracy: Proportion of good predictions.
Recall: positive predicted value: ratio between true positives and all predicted positives.

4 Empirical studies

4.1 Implementation for logistic regression

In this section I will quickly present how the theoretical part described in the first before has been applied to the logistic regression used here.

4.1.1 Empirical fluctuation process

The empirical fluctuation process as described above applies for all generalized linear models computed via maximum likelihood estimation. In our case the logistic loss is:

$$loss \ = \ -\frac{1}{n}\sum_{i=1}^{n} Y_i ln(h(X_i\beta)) + (1-Y_i)ln(1-h(X_i\beta)) + \frac{\alpha}{2}||\beta||^2$$

where $h(z) = \frac{exp(z)}{1+exp(z)}$. The deriving of h is $h'(z) = \frac{exp(z)}{(1+exp(z))^2}$ which leads to:

$$\psi_i = xh'(x\beta)\left\{\frac{h(y\beta)-y_i}{(1-h(x\beta))h(x\beta)}\right\} + \alpha\beta = x_i \cdot (h(x_i'\hat{\beta}) - y_i) + \alpha\beta.$$

Python implementation is given in Annexe.

4.1.2 Partitioning

The likelihood-ratio test, note that the formula:

$$D = -2ln\left(\frac{likelihood \ for \ the \ null \ model}{likelihood \ for \ the \ alternative \ model}\right)$$

is not computable because for large number of observations, the likelihood is too close to zero which leads to a round-off error and indeterminate form $0/0$. The following equation for D solves this problem:

$$D = -2ln(lkhd \ null) + 2ln(lkhd \ alternative)$$
$$\text{with } ln(likelihood) = \sum_{i=1}^{n} Y_i ln(h(X_i\beta)) + (1-Y_i)ln(1-h(X_i\beta))$$

4.1.3 Twisted logistic regression

Computation of the twisted logistic regression derives from `sklearn.linear_model.LogisticRegression`. Object `TwistedLogisticRegression` is initialized with, as attributes, the type of solver to be used, the penalization term α and a rigidity term r. Let $X = [X^{(1)}, ..., X^{(l)}]$ and $Y = [Y^{(1)}, ..., Y^{(l)}]$ be the lists of variables and labels for each periods (hence $X^{(p)} \in \mathbb{R}^{n_p \times d}$ and $Y^{(p)} \in \mathbb{R}^{n_p}$ where n_p is the number of observations at the p^{th} period). When `fit(X, Y)` is called, the following callable functions are computed:

Loss function:

$$loss(\beta) = \sum_{p=1}^{l}\left(-\frac{1}{n_p}\sum_{i=1}^{n_p} \ell(X_i^{(p)}, Y_i^{(p)}, \beta^{(p)}) + \frac{\alpha}{2}||\beta^{(p)}||^2 \right) + \frac{r}{2}\sum_{p=1}^{l-1} ||\beta^{(p+1)} - \beta^{(p)}||^2$$

Gradient function:

$$\nabla_\beta(\beta) \;=\; -\begin{bmatrix} \sum_{i=1}^{n_1} \nabla_{\beta^{(1)}} \ell(X_i^{(1)}, Y_i^{(1)}, \beta^{(1)}) \\ \vdots \\ \sum_{i=1}^{n_l} \nabla_{\beta^{(l)}} \ell(X_i^{(l)}, Y_i^{(l)}, \beta^{(l)}) \end{bmatrix} + \alpha \begin{bmatrix} \beta^{(1)} \\ \vdots \\ \beta^{(l)} \end{bmatrix} + r\ {}^tA \cdot A \begin{bmatrix} \beta^{(1)} \\ \vdots \\ \beta^{(l)} \end{bmatrix}$$

Hessian matrix function:

$$H_\beta(\beta) \;=\; -\begin{bmatrix} \sum_{i=1}^{n_1} H_{\beta^{(1)}} \ell(X_i^{(1)}, Y_i^{(1)}, \beta^{(1)}) & & 0 \\ & \ddots & \\ 0 & & \sum_{i=1}^{n_l} H_{\beta^{(l)}} \ell(X_i^{(l)}, Y_i^{(l)}, \beta^{(l)}) \end{bmatrix}$$

$$+\ \alpha \cdot I_{d\times l} \;+\; r \cdot\ {}^tA \cdot A$$

The computation of $\sum_{i=1}^{n_p} \nabla_{\beta^{(p)}} \ell(X_i^{(p)}, Y_i^{(p)}, \beta^{(p)})$ and $\sum_{i=1}^{n_p} H_{\beta^{(p)}} \ell(X_i^{(p)}, Y_i^{(p)}, \beta^{(p)})$ can be easily derived from the open source of `sklearn` and I just had to rebuild the three functions with it, $\alpha \cdot I_{d\times l}$ and $r \cdot\ {}^tA \cdot A$.

Python implementation is too long to be included in the annexe but can be provided on request.

4.2 Case studies

In this section, two case studies will be presented. For confidentiality purpose, competitors names are replaced by "COMPET1", ..., "COMPET5". These competitors are more or less the five most important competitors in the direct motor insurance market in Spain.

4.2.1 Control case

Before studying special cases, I tested the procedure an artificial stable period. To do so, I fit a logistic regression model on a two month period. Given estimate β, true labels $\{Y_i\}_{i<n}$ are replaced by:

$$\tilde{Y}_i \;=\; h(X_i\beta + \epsilon_i)$$

where $h(z) = \frac{exp(z)}{1+exp(z)}$ and $\epsilon_i \sim \mathcal{N}(0, \sigma)$. The variance σ is chosen such that the area under the ROC with Y and $\tilde{Y}$ are equal.

Empirical fluctuation process At the time when I made this control test, there was only 4 variables in the model, so in the following $\psi_i \in \mathbb{R}^4$. The figure 9 draws the empirical fluctuation process

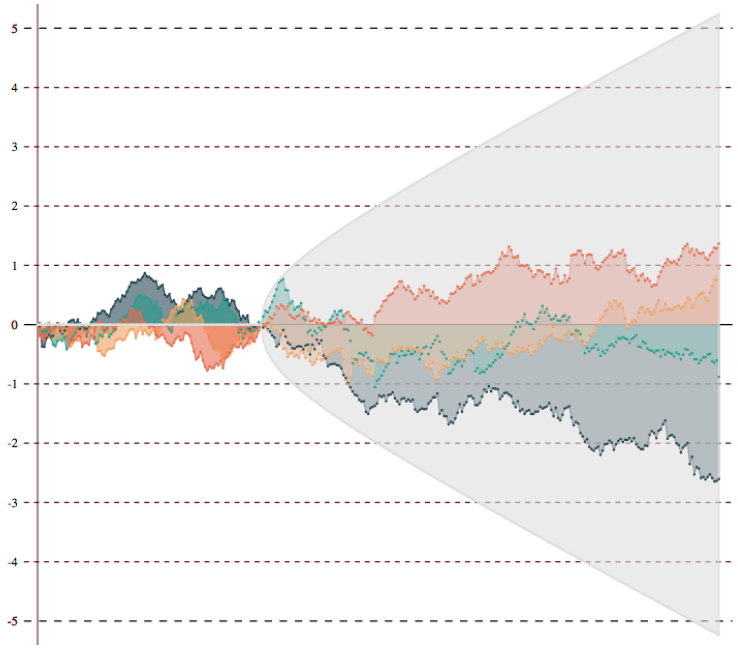

Figure 9: Example of fluctuation process on a stable model

As expected, the null hypothesis of no change is not rejected here. As usual for statistical test, this would not prove that there is no change. Also, we to test the partitioning step on this control dataset.

Partitioning The next step of our procedure consists in partitioning the period of interest. Here twelve likelihood-ratio test are computed, for twelve different dates evenly distributed. Results are in figure 10:

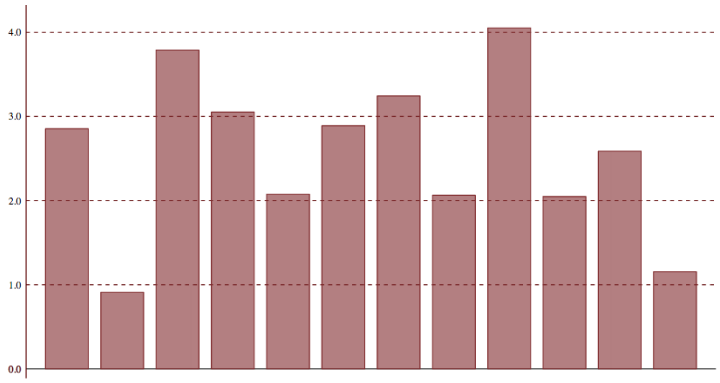

Figure 10: Partitioning the period of interest - stable model

On this example, all the likelihood-ratio statistic are below 4.1. In this case where it follows a Chi squared distribution with 4 degrees of freedom, the p-value associated to 4.1 is about 0.39, which confirms that we can not reject the null hypothesis.

Now that we have an idea of what does the procedure on a unchanged model, we are going to apply it on two case studies.

4.2.2 Case study I: December 2013

In December 2013, Direct Seguros opered a rebranding along with a tariff change. We would like to apply the procedure to see if we can detect these events from the dataset. We fit the model on November 2013 and plot the empirical fluctuation process until february 2014 (figure 11):

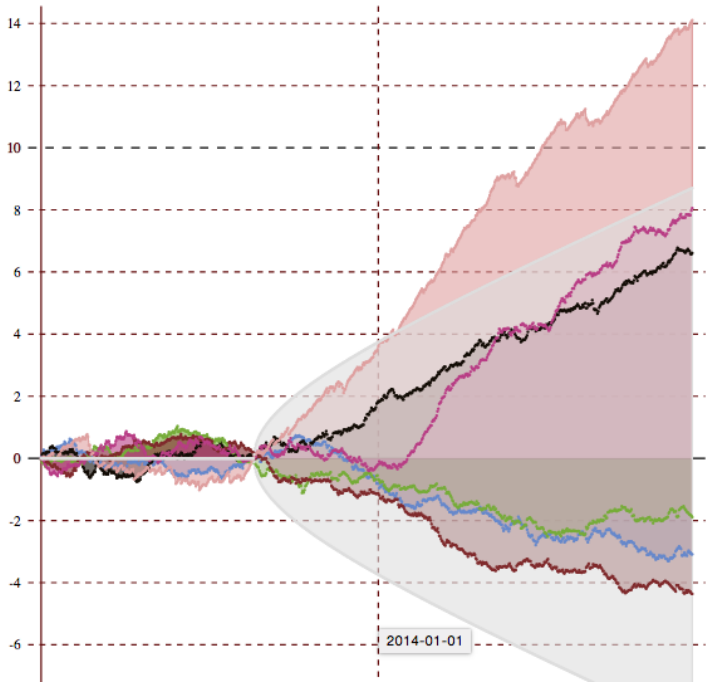

Figure 11: Empirical fluctuation process - late 2013

We can see that at the date 2014-01-01, the fluctuation test rejects the null hypothesis of no change. Hence we know at 95% that there has been a structural change between the end of the fitting (start of December) and the 2014-01-01. We compute several tests on December to infer for the most likely break points.

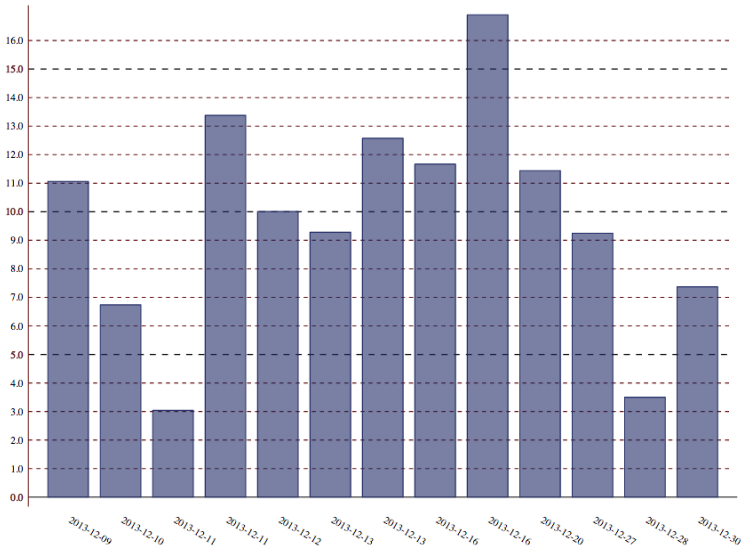

Figure 12: Partioning - December 2013

The maximum is reached on 2013-12-16, which must be linked to the rebranding of the 17th. Note that our algorithm detects a break on the 16th because the split we use does not include the 17th of Decembre (chosen dates are evenly distributed in all observations sorted by date).

Finally, to understand the change we compute the twist logistic regression on this period (figure 13)

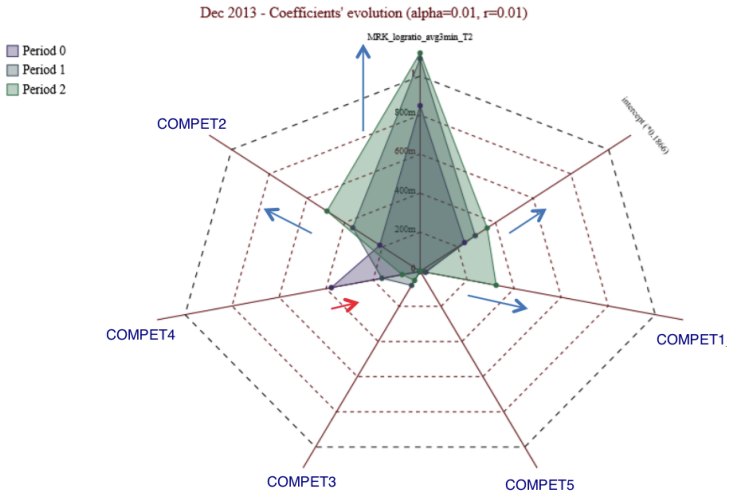

Figure 13: Coefficients' evolution - late 2013

We can see that almost all coefficients of our log-ratios with competitors increase in absolute value. In our model, variables are standardized and hence we consider that a coefficient reflects the impact of a competitor price in comparison with Direct Seguros. For example, if a coefficient is very low, the conversion depends mostly on the intercept, and prices have low impact on conversion. On the other hand, in our case, we can say that price sensitivity to other competitors is dramatically increased after the change (until January 2014 at least).

4.2.3 Case study II: June 2015

In June 2015, COMPET4 and COMPET4bis, which is an important competitor of Direct Seguros but not in our model because of its high correlation with COMPET4[5], change their pricing. Instead of giving two prices to prospect, a "standard" price and "vip" price, they started giving only one price. To assess whether this impacted the conversion in our brand, we monitor the empirical fluctuation process derived from this model.

We fit the model on May 2015 and plot the empirical fluctuation process (figure 14):

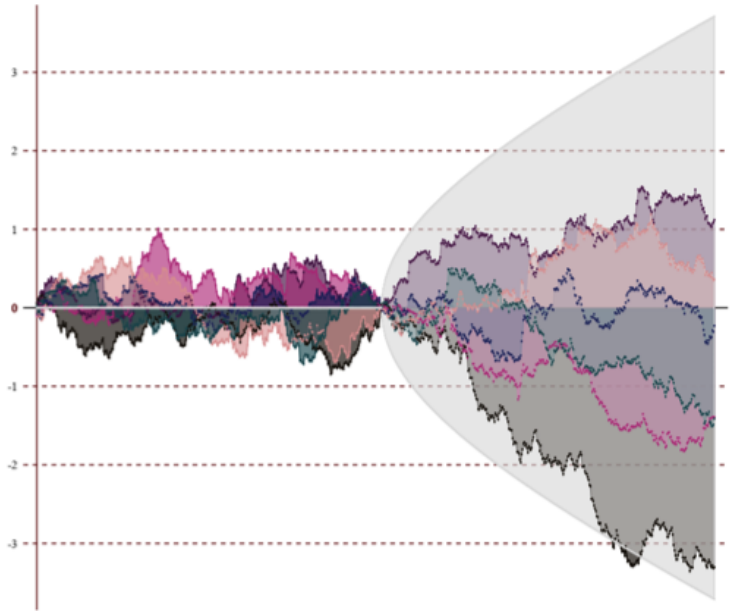

Figure 14: Empirical fluctuation process - late 2013

We can see that at the date 2015-06-24, the fluctuation test rejects the null hypothesis. Hence we know at 95% that there has been a structural change between the start of June and the 2015-06-24. We compute several tests on this period to infer for the most likely break points.

[5]correlation coefficient is ≈ 0.98

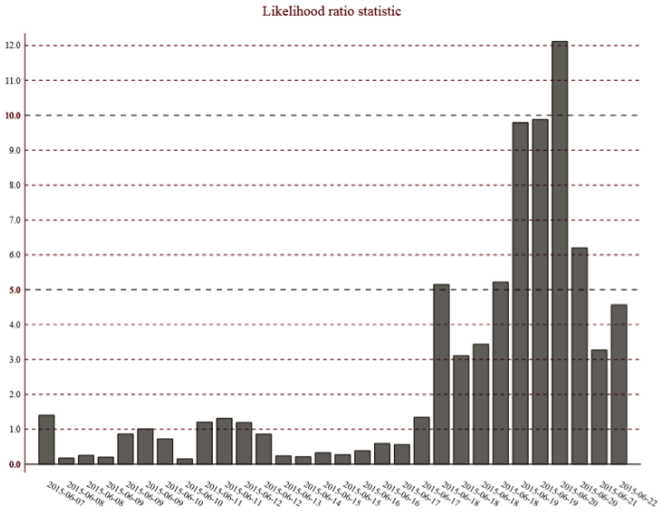

Figure 15: Partioning - June 2015

The break is confirmed on the 2015-06-20, which is exactly the date when COMPET4 and COMPET4bis opered their change.

Finally, to understand the change we compute the twist logistic regression on this period (figure 16)

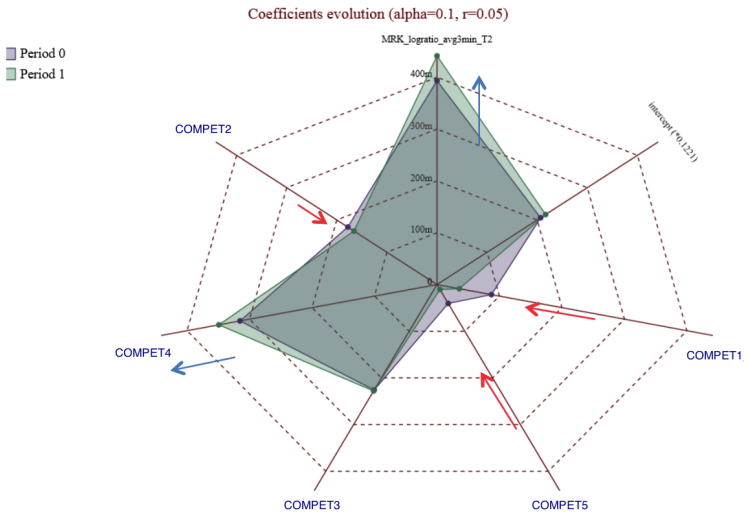

Figure 16: Coefficients' evolution - June 2015

All competitors' coefficients decrease in absolute value, except COMPET4's one which increases. Hence, after the change, our ratio with COMPET4 has more impact on conversion which would mean that we are more compared to COMPET4 after its move.

5 Conclusion

We have seen in the first part that repeating likelihood-ratio test as new observations arrive is not time consistent and would lead to an increase the risk of type-I error. The empirical fluctuation process allows one to monitor a model while keeping this risk constant. The partitioning step returns the precise date of the break. To construct the *Competitive Conversion model* we have also dealt with high dimensionality and correlation problems.

The experiments show that the procedure described above gives meaningful results when trying to detect structural changes in the *Competitive Conversion model*. It also provides precious information on the change in itself, even though a good understand of it would require a good overview of the market and business related experience. The approach presented in this document is expected to be applied in an industrialized manner to pricing demand models, that usually require a thorough model validation and model monitoring process.

From a personal point of view, this internship allowed me to discover research in company. I had the chance to have all my time dedicated to this research, and had a free hand to explore new ideas. I have learnt a lot through this internship and am sure this experience will be of great help for my future professional life.

References

[1] Juchan Bai and Pierre Perron. *Computation and Analysis of Multiple Structural Change Models.* Journal of Applied Econometrics, 2003.

[2] Hjort and Koning. *Tests for Constancy of Model Parameters Over Time.* Nonparametric Statistics, 2002.

[3] Achim Zeileis Friedrich Leisch Christian Kleiber Kurt Hornik. *Monitoring Structural Change in Dynamic Econometric Models.* 2005.

[4] Achim Zeileis; Kurt Hornik. *Generalized M-Fluctuation Tests for Parameter Instability.* 2007.

[5] Lerche. *Boundary Crossing of Brownian Motion.* 1984.

[6] Herbet Robbins. *Statistical Methods Related to the Law of Iterated Logarithm.* The Annals of Mathematical Statistics, 1970.

[7] Leonard A Stefanski and Dennis D Boos. *The Calculus of M-Estimation.* 2001.

[8] Chia-Shang James Chu; Maxwell Stinchcombe; Halbert White. *Monitoring Structural Change.* 1996.

biblio

List of Figures

6 Annexe

Python implementation of empirical fluctuation process for logistic regression:

```
class Efp:
    """Computes empirical fluctuation process for a given model

    Parameters
    ----------

    model: Object of class Model
        The model on which the Efp is computed

    t_0: int, date, string (date-like) or float(in interval [0, 1])
        Time of the beginning of fitting period

    t_m: int, date, string (date-like) or float(in interval [0, 1])
        Time of the end of fitting period, and the beginning of monitoring

    t_k: int, date, string (date-like) of float( in interval [0, 1])
        Last time for monitoring

    l: int, float (default=3)
        parameter for the boundary functions

    C: int, float (default: result of CV optimization)
        parameter for regularization (1/alpha)

    Attributes
    ----------

    model

    i_0, i_m, i_k: int
        Indices corresponding to t_0, t_m, t_k in model.X.index

    C: float
    """
    def _b(self, i, m, l):
        if(i<=m):
            return 0
        else:
            return np.sqrt(i*(i-m)*(l*l+np.log(i/float(i-m))))/m

    def _dh(self, XBI):
        return np.divide(np.exp(XBI),np.power(1+np.exp(XBI),2))

    def _carre(self, x):
        x_=np.matrix(x)
        return np.array(x_.T*x_)

    def _var(self, X, XBI):
        dh_=np.array(self._dh(XBI))
        sum=0
        n=len(X)
        for i in range(n):
            sum=sum+dh_[i]*self._carre(X[i,:])
```

```
    return np.matrix(sum/n)

def _cusum_mosum(self, Y, X, XBI, beta, m):
    var_ = self._var(X, XBI)
    sqrt_var = scipy.linalg.sqrtm(var_)
    inv_sqrt_var = np.linalg.inv(sqrt_var)

    cumsum_phi = np.matrix(np.cumsum(self.pen_phi, axis=0))

    self.mosum_ticks = np.array(map(int,
                                    np.linspace(0,
                                                len(self.pen_phi),
                                                self.n_win+1)))
    split = np.split(self.pen_phi, self.mosum_ticks[1:-1])
    mosum_phi = [np.sum(s, axis=0) for s in split]
    mosum_phi = np.matrix(np.squeeze(mosum_phi))

    return (m**-0.5)*inv_sqrt_var*cumsum_phi.T, inv_sqrt_var*mosum_phi.T


def __init__(self,
             model,
             t_0=None,
             t_m=None,
             t_k=None,
             l=6,
             C = None,
             n_win=50):
    self.model = model

    if t_0 is None:
        t_0 = 0
    t_0 = str(t_0)
    try:
        d = datetime.datetime.strptime(t_0, '%Y-%m-%d')
        t_0 = sum(model.dates<d)
    except ValueError:
        pass
    self.i_0 = int(t_0)

    if t_k is None:
        t_k = len(self.model.Y)
    t_k = str(t_k)
    try:
        d = datetime.datetime.strptime(t_k, '%Y-%m-%d')
        t_k = sum(model.dates<d)
    except ValueError:
        pass
    self.i_k = int(t_k)

    if t_m is None:
        t_m = self.i_k
    t_m = str(t_m)
    try:
        d = datetime.datetime.strptime(t_m, '%Y-%m-%d')
        t_m = sum(model.dates<d)
    except ValueError:
```

```
        pass
    try:
        if ((float(t_m)>0) & (float(t_m)<1)):
            t_m = float(t_m)*self.i_k + (1-float(t_m))*self.i_0
    except ValueError:
        pass
    self.i_m = int(t_m)

    self.l = l
    gauss = scipy.stats.norm()
    self.p_value = 4*(1-gauss.cdf(self.l)+self.l*gauss.pdf(self.l))

    if (self.i_k > self.i_0):
        self.x_axis = range(self.i_0, self.i_k)
        self.b = np.array([self._b(i, self.i_m, l)\
            for i in self.x_axis])
    else:
        print "    ERROR: i_k <= i_0"
        raise NameError("Chronology")

    ind_0_to_k = range(self.i_0, self.i_k)
    Y = np.matrix(self.model.Y.iloc[ind_0_to_k])
    X = np.matrix(self.model.X.iloc[ind_0_to_k])

    ind_0_to_m = range(self.i_0, self.i_m)

    self.model.classifier.class_weight = False

    if C is None:
        self.C = model.C
    else:
        self.C = C

    self.model.classifier.C = float(self.C)/len(ind_0_to_m)
    self.model.classifier.fit(self.model.X.iloc[ind_0_to_m],
                              np.ravel(self.model.Y.iloc[ind_0_to_m]))
    beta = np.transpose(np.matrix(self.model.classifier.coef_[0]))
    if hasattr(self.model.classifier, 'intercept_'):
        intercept = model.classifier.intercept_[0]
    else:
        intercept = 0
    pred = np.matrix(self.model.classifier.predict_proba(X))[:,1]
    self.raw_phi = np.array(X)*np.array(pred-Y)
    m = self.i_m - self.i_0 + 1
    C = self.model.classifier.get_params()["C"]
    self.pen_phi = self.raw_phi+0*(beta/float(m*C)).T
    XBI = X*beta+intercept

    self.n_win = n_win
    self.cusum, self.mosum = self._cusum_mosum(Y, X, XBI, beta, m)
```