Étude sur les AVC

Iruomachi IRUOMAH, Yann BROCHET

Sommaire

1 Introduction

1.1 Présentation du sujet

Chaque année, environ **15 millions de personnes** dans le monde sont victimes d'un accident vasculaire cérébral (AVC). Il s'agit de la **première cause** de handicap acquis chez l'adulte et de la deuxième cause de mortalité à l'échelle mondiale, juste après les maladies cardiaques.

En France, un **AVC** survient toutes les quatre minutes, ce qui témoigne de l'ampleur de ce problème de santé publique.

Plusieurs facteurs de risque augmentent la probabilité de survenue d'un **AVC**. Parmi les principaux, on retrouve :

- l'hypertension artérielle
- le tabagisme et la consommation excessive d'alcool,
- le diabète et l'obésité,
- ainsi que le manque d'activité physique.

1.2 Présentation des données

Nous disposons d'un **ensemble de données** portant sur les **caractéristiques** d'individus ayant ou non été victimes d'un **accident vasculaire cérébral** (AVC).

Initialement, la base comptait **5 110 observations** réparties sur **11 variables**. Après nettoyage et transformation des données, le nombre d'observations a été ramené à **4 908.**

Parmi les variables, **3** sont quantitatives et **8** sont qualitatives.

L'objectif principal de cette étude est de **développer un modèle** permettant de **prédire** la survenue d'un AVC à partir des **caractéristiques individuelles**.

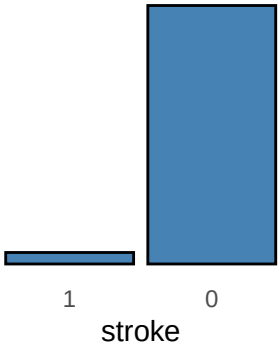
Voici un aperçu ressemble la base de donnée :

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
1	Male	67	0	1	Yes	Private	Urban
3	Male	80	0	1	Yes	Private	Rural
4	Female	49	0	0	Yes	Private	Urban
5	Female	79	1	0	Yes	Self-employed	Rural
6	Male	81	0	0	Yes	Private	Urban
7	Male	74	1	1	Yes	Private	Rural

1.3 Analyse descriptive

1.3.1 Distribution de la variable à prédire

La variable que nous cherchons à prédire est **stroke** : si sa valeur est 1, cela signifie que l'individu a été victime d'un AVC ; sinon, il ne l'a pas été.



Comme on peut le voir, nous observons un très grand **déséquilibre** dans la variable stroke, ce qui est problématique mais un **ajustement** sera fait par la suite.

1.3.2 Variables explicatives

Variable	Type	Description
gender	factor	Sexe de l'individu
age	numeric	Âge de l'individu
hypertension	integer	Hypertension (0 = non, 1 = oui)
heart_disease	integer	Maladie cardiaque (0 = non, 1 = oui)
ever_married	factor	Marié(e) ou non
work_type	factor	Type de travail
Residence_type	factor	Zone rural ou urbain
avg_glucose_level	numeric	Niveau moyen de glucose
bmi	integer	Indice de masse corporelle (BMI)
smoking_status	factor	Statut de fumeur

Voici nos variables explicatives plus en détail.

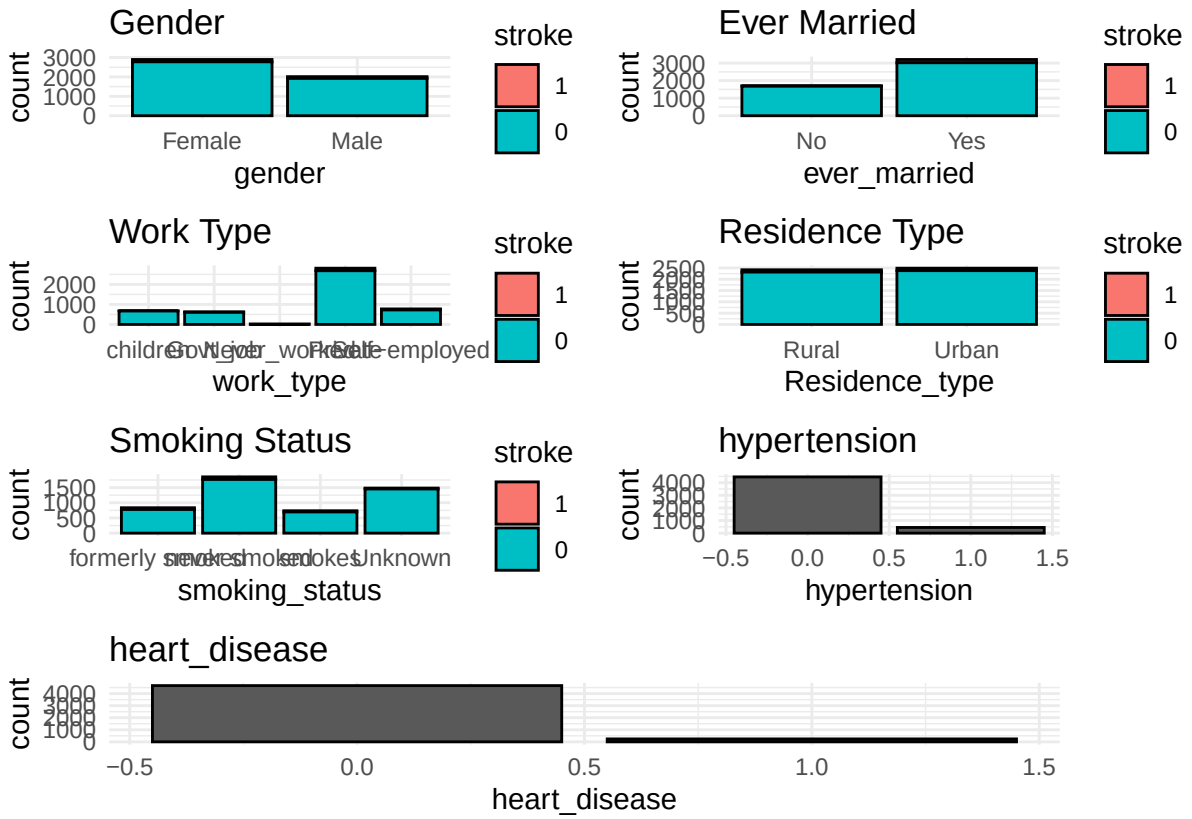
1.3.3 Variables quantitatives

Nous avons réalisé un **test de Student** sur nos variables quantitatives afin de voir si leur moyenne différait entre les deux classes. Les p-values sont toutes **très faibles**, ce qui indique que l'âge, le taux de glucose et l'IMC varient de manière significative entre les groupes. Ces variables semblent donc **pertinentes pour la classification**.

Variable	p_value
Age	0.0000000
Avg Glucose Level	0.0000000
BMI	0.0001631

1.3.4 Variables qualitatives

Regardons les **proportions** que prennent les modalités dans nos variables afin d'analyser les liaisons qu'elles auraient avec la variable à prédire.



La variable Residence Type ne semble **pas discriminante**, car les proportions de cas avec et sans AVC y sont très similaires. À l'inverse, d'autres variables comme ever_married, work_type, ou smoking_status montrent une certaine variation entre les deux groupes. On note également, encore une fois, un **fort déséquilibre** entre les classes de la variable cible (stroke), qu'il sera nécessaire de corriger avant d'entraîner un modèle.

Nous allons effectuer un test du χ^2 :

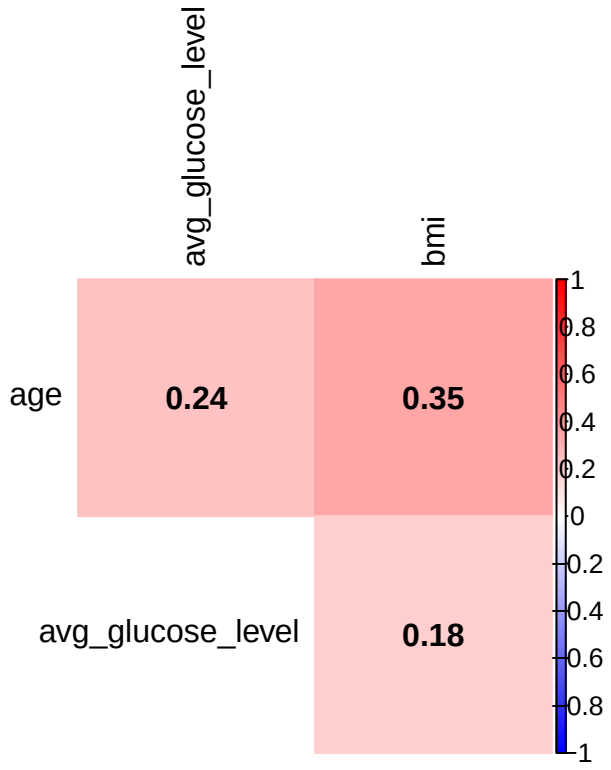
Variable	p_value
Gender	0.6805109
Ever Married	0.0000000
Work Type	0.0000000
Residence Type	0.7272126
Smoking Status	0.0000001
Hypertension	0.0000000
Heart Disease	0.0000000

Nous pouvons observer que deux de nos variables ne sont **pas significatives** : le genre est donc probablement moins utile pour prédire un AVC, tout comme le lieu d'habitation (rural ou urbain), ce que nous avions déjà pu constater à travers sa distribution précédente.

1.4 Préparation des données

1.4.1 Colinéarité

Avant la préparation de nos données, portons un regard sur la **colinéarité** de nos variables quantitatives.



Nos variables sont **peu corrélées** entre elles, ce qui signifie qu'il n'est pas nécessaire d'agir sur la colinéarité à ce stade. Nous aurions pu, si besoin, traiter automatiquement via un **step_corr()** dans les recettes de nos modèles.

1.4.2 Échantillonage

- Comme dit auparavant, nous avons un **gros déséquilibre** sur la variable à prédire **stroke**.
- Nous allons donc utiliser **l'algorithme SMOTE** pour équilibrer nos données, via **step_smote()** de la librairie themis. Cette méthode permet de **créer de nouvelles observations** pour la classe minoritaire, en s'inspirant de celles déjà présentes. L'idée est **d'équilibrer les données** sans simplement copier les anciennes, ce qui aide le modèle à mieux apprendre.
- Et afin d'optimiser nos hyperparamètres, nous allons effectuer une **validation croisée à 10 plis**.

1.4.3 Découpage des données

- **3/4** des données seront utilisées pour l'entraînement et **1/4** pour les données test.
- Pour que les classes soient **équitablement réparties** dans ce découpage, nous utiliserons l'argument **strata**. Cet argument permet de **stratifier** les données (s'assurer que chaque pli de la validation croisée contient à peu près la même proportion de chaque classe cible).
- Pour nos variables qualitatives, on fera un **encodage One-Hot**, c'est-à-dire que chaque modalité sera transformée en une colonne supplémentaire contenant 0 ou 1, grâce à la fonction **step_dummy()**. (Par exemple, pour la variable WorkType, une colonne sera créée par modalité, comme WorkType_Private, qui prendra la valeur 1 ou 0 selon le cas.)

1.5 Métriques

- Dans le cadre de cette étude, nous privilégierons la métrique **recall** plutôt que la simple accuracy. En effet, dans le domaine médical, et plus particulièrement dans **la détection d'AVC**, les cas positifs sont

rares par rapport aux cas négatifs. Une **forte accuracy** pourrait donc être **trompeuse**, car le modèle pourrait prédire majoritairement l'absence d'AVC sans réellement identifier les cas à risque.

- Le **recall** mesure la capacité du modèle à **détecter correctement** les cas positifs. Un faible recall signifierait que des patients ayant fait un AVC ne sont pas détectés, ce qui peut avoir de **lourdes conséquences**. Dans ce contexte, il est préférable de générer des faux positifs (alerter à tort) que de manquer de vrais cas (faux négatifs).

2 Construction des modèles

2.1 Linear Discriminant Analysis

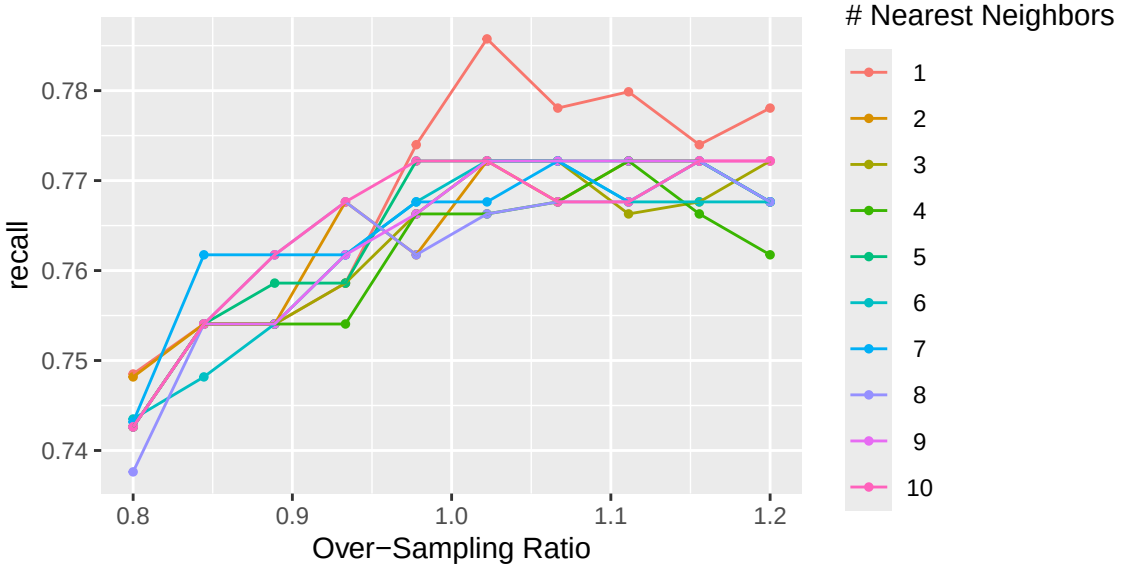
2.1.1 Spécification et recette du modèle

```
lda_mod <- discrim_linear() |>
  set_engine("MASS") |>
  set_mode("classification")

rec_lda <- recipe(stroke ~ ., data = data_train) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors()) |>
  step_smote(stroke, over_ratio = tune(), neighbors = tune())
```

- Tout d'abord, nous avons **centré et réduit** toutes nos variables numériques grâce à **step_center()** et **step_scale()**, afin d'uniformiser leurs échelles.
- Nous utilisons ensuite **l'algorithme SMOTE** pour corriger le **déséquilibre** de la variable cible. Le paramètre **over_ratio** détermine le degré de suréchantillonnage de la classe minoritaire, tandis que **neighbors** définit le nombre de voisins pris en compte pour générer les nouvelles observations synthétiques. Nous leur avons affecté le paramètre **tune()** afin de pouvoir les **optimiser** par la suite.

2.1.2 Optimisation des hyperparamètres



over_ratio	neighbors
1.022222	1

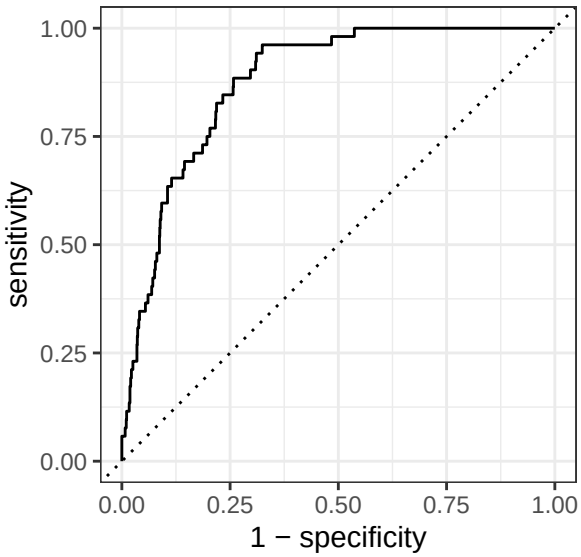
- Comme nous pouvons le constater, l'optimisation des paramètres a conduit à une valeur de **neighbors** égale à 1 et un **over_ratio** proche de 1 ce qui signifie qu'il y a presque autant d'observations synthétiques que d'observations de la classe majoritaire.

2.1.3 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	46	334
0 (prédit)	6	841

- Le modèle **LDA** est très bon par rapport au **recall** (seulement 6 faux négatifs), ce qui est essentiel dans la détection d'AVC. Malgré un taux de faux positifs élevé, il reste **adapté au contexte**, car mieux vaut alerter à tort que rater un vrai cas.

2.1.4 Courbe ROC



L'aire sous **la courbe ROC** semble **satisfaisante**. Par la suite, nous ne représenterons plus cette courbe, car elle est pratiquement identique pour l'ensemble de nos modèles.

2.1.5 Performance globale

.metric	.estimate
accuracy	72.3
f_meas	21.3
recall	88.5
precision	12.1
specificity	71.6
roc_auc	87.7
error	27.7

- Avec un **recall de 88,5 %**, une **accuracy de 72,3 %** et une aire sous la courbe ROC de 87,7 %, notre modèle **LDA** se positionne déjà comme **l'un des meilleurs** candidats pour la détection des AVC.

2.2 Quadratic Discriminant Analysis

Nous n'avons pas pu exploiter **le modèle QDA**, en raison de **potentielles limitations** techniques dans nos données:

- Les matrices de covariance pourraient être **mal conditionnées**, rendant l'estimation instable.
- La QDA est **sensible à la colinéarité** entre variables. Malgré le fait qu'a priori nous ne rencontrions pas de problème à ce niveau, nous avons tout de même tenté d'appliquer un step_corr() sur nos données, ce qui n'a toutefois pas permis de stabiliser le modèle.
- Le rééquilibrage par downsampling n'a pas permis de résoudre les **erreurs persistantes**.
- Et enfin, la taille très réduite de la classe minoritaire (AVC) peut empêcher une estimation fiable des paramètres du modèle.

2.3 K-Nearest Neighbors

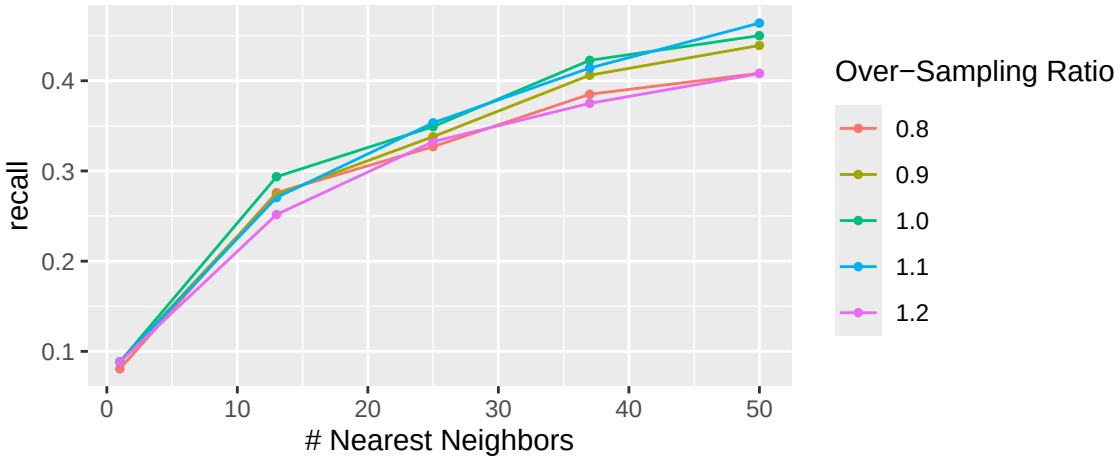
2.3.1 Spécification et recette du modèle

```
knn_mod <- nearest_neighbor() |>
  set_mode("classification") |>
  set_engine("kknn") |>
  set_args(neighbors = tune())

rec_knn <- recipe(stroke ~ ., data = data_train) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors()) |>
  step_smote(stroke, over_ratio = tune())
```

- Le modèle **k-Nearest Neighbors** est un algorithme de classification basé sur la **proximité** : pour prédire la classe d'une nouvelle observation, il regarde les **k observations** les plus proches dans l'espace des variables.
- Le paramètre **neighbors** correspond ici à **k**, c'est-à-dire le nombre de voisins à considérer pour prendre la décision.

2.3.2 Optimisation des hyperparamètres



over_ratio	neighbors
1.1	50

- L'optimisation a conduit à un **over_ratio** de 1.1, ce qui signifie que la classe minoritaire a été légèrement **sur-échantillonnée** pour rééquilibrer le jeu de données. Le paramètre **neighbors** est ici égal à 50, ce qui correspond au nombre de voisins utilisés par le modèle KNN pour faire ses prédictions.

2.3.3 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	28	260
0 (prédit)	24	915

- Le modèle **KNN** présente un **recall** assez faible (24 faux négatifs sur 52 cas réels), ce qui est problématique dans un contexte comme la détection d'AVC. Bien qu'il détecte bien les non-AVC, son efficacité sur les cas critiques reste **limitée**.

2.3.4 Performance globale

.metric	.estimate
accuracy	76.9
f_meas	16.5
recall	53.8
precision	9.7
specificity	77.9
roc_auc	75.9
error	23.1

- Comme nous l'avons dit auparavant, le modèle présente un **recall faible** de **53,8 %**, mais une **accuracy** relativement bonne de **76,9 %**, principalement due à une bonne détection des vrais négatifs. L'aire sous la courbe ROC est correcte, mais reste inférieure à celle obtenue dans le modèle précédent. Dans l'ensemble, ce modèle ne semble **pas adapté** à notre objectif, qui repose sur une détection fiable des cas d'AVC.

2.4 Naive Bayes

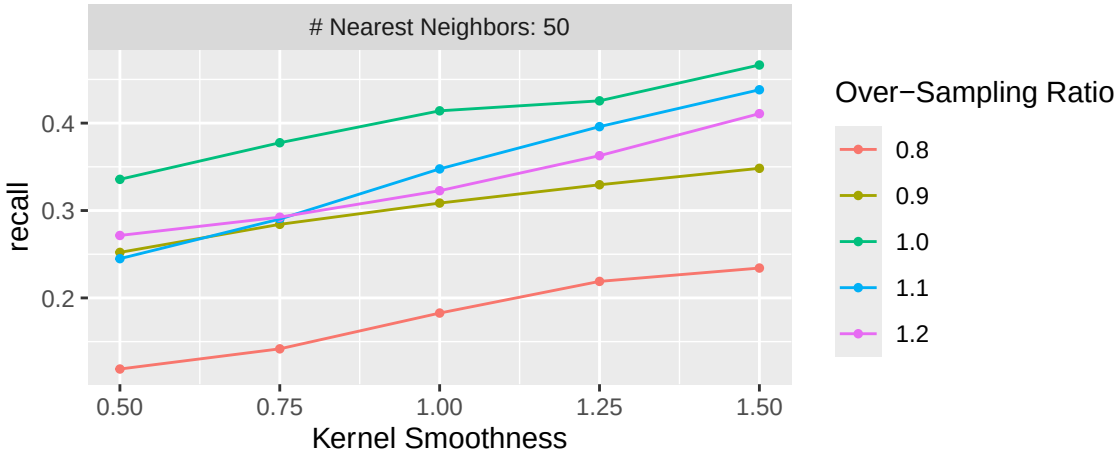
2.4.1 Spécification et recette du modèle

```
bayes_mod <- naive_Bayes(smoothness = tune(), Laplace = 1) |>
  set_engine("naivebayes") |>
  set_mode("classification")

rec_bayes <- recipe(stroke ~ ., data = data_train) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors()) |>
  step_smote(stroke, over_ratio = tune(), neighbors = tune())
```

- Le **Naive Bayes** est un modèle simple et rapide basé sur les **probabilités**. Il suppose que les variables sont **indépendantes** entre elles et prédit la classe la plus probable. Malgré cette hypothèse forte, il donne souvent de bons résultats, surtout sur des **données déséquilibrées**. Nous allons voir ce que cela donne sur nos données.
- **Laplace** permet d'éviter les probabilités nulles en ajoutant une petite constante aux effectifs, c'est utile quand une modalité n'apparaît pas dans une classe.
- Le paramètre **smoothness** sert à rendre les prédictions plus **stables** pour les variables numériques. Il contrôle le niveau de lissage des distributions : plus la valeur est élevée, plus le modèle "adoucit" les différences entre les observations.

2.4.2 Optimisation des hyperparamètres



over_ratio	neighbors	smoothness
1	50	1.5

- L'optimisation a conduit à un **smoothness de 1.5**, ce qui permet un lissage modéré des distributions numériques. Cela rend le modèle plus souple et moins sensible aux valeurs extrêmes, tout en conservant une bonne capacité de discrimination.
- Et nous obtenons avec ceci un **over_ratio** de **1** et un **neighbors** de **50**.

2.4.3 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	20	98
0 (prédit)	32	1077

Le modèle **naive bayes** présente un **recall très faible**. Il ne détecte que 20 cas positifs sur 52, laissant passer 32 vrais AVC, ce qui est problématique dans un contexte où la détection est prioritaire. En revanche, il affiche une **très bonne capacité à identifier les non-AVC**, avec 1 077 vrais négatifs sur 1 175. Cela se traduit par une **bonne accuracy** , mais un recall trop faible pour être adapté à notre objectif.

2.4.4 Performance globale

.metric	.estimate
accuracy	89.4
f_meas	23.5
recall	38.5
precision	16.9
specificity	91.7
roc_auc	82.1
error	10.6

- Comme nous l'avons observé, ce modèle affiche **le recall le plus faible** obtenu jusqu'à présent (38,5 %). En contrepartie, il atteint une **accuracy élevée** de 89,4 %, mais cela ne correspond pas à notre objectif. Il est donc pas adapté à nos données.

2.5 Support Vecteur Machine Linear

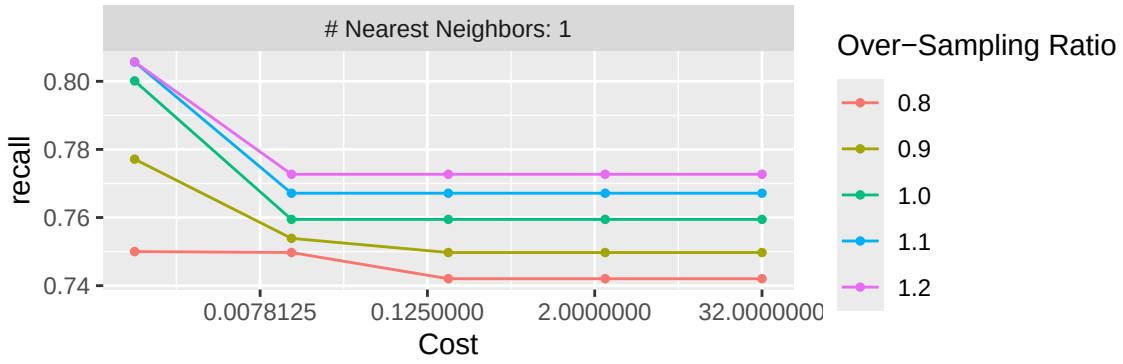
2.5.1 Spécification et recette du modèle

```
svm_linear_mod <- svm_linear() |>
  set_mode("classification") |>
  set_engine("kernlab") |>
  set_args(cost = tune())

rec_svml <- recipe(stroke ~ ., data = data_train) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors())  |>
  step_smote(stroke, over_ratio = tune(), neighbors = tune())
```

- Le **SVM linéaire** est un modèle de classification supervisée qui cherche à séparer les classes à l'aide d'un **hyperplan optimal**. Il repose sur le principe de **maximisation de la marge**, c'est-à-dire la distance entre l'hyperplan et les points les plus proches de chaque classe (appelés vecteurs de support). L'objectif est de trouver une **séparation la plus nette possible** entre les classes, ce qui améliore la capacité du modèle à généraliser sur de nouvelles données. Ce type de modèle est particulièrement efficace dans les cas où les classes sont **séparables de manière approximativement linéaire.**
- Le paramètre **cost** détermine **l'importance donnée** aux erreurs de classification. Une valeur élevée cherche à éviter les erreurs à tout prix, quitte à créer un modèle plus complexe. Une valeur faible autorise quelques erreurs pour obtenir un modèle plus simple et plus souple.

2.5.2 Optimisation des hyperparamètres



over_ratio	neighbors	cost
1.1	1	0.0009766

- Ici, à l'aide du graphique, on voit que le paramètre **cost** est plutôt **faible**, ce qui autorise quelques erreurs pour obtenir un modèle plus souple.

2.5.3 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	47	388
0 (prédit)	5	787

- Le modèle **identifie correctement** la majorité des vrais positifs et **commet très peu d'erreurs** de type faux négatif, ce qui est essentiel dans un contexte sensible comme la détection d'un AVC. Bien qu'il génère un grand nombre de faux positifs, cela reste acceptable dans notre cas : il vaut mieux déclencher une alerte à tort que de passer à côté d'un cas réel.

2.5.4 Performance globale

.metric	.estimate
accuracy	68.0
f_meas	19.3
recall	90.4
precision	10.8
specificity	67.0
roc_auc	87.3
error	32.0

- Pour rappel, la métrique principale visée est le **recall**, qui atteint ici **90,4 %**, un excellent résultat et **le meilleur obtenu jusqu'à présent**. **L'accuracy** est d'environ **70 %**, et la courbe ROC affiche une AUC autour de **87 %**, ce qui confirme la bonne performance globale du modèle. À ce stade, il s'agit donc de notre **meilleur modèle** pour prédire un AVC.

2.6 Suppport Vecteur Machine Radial

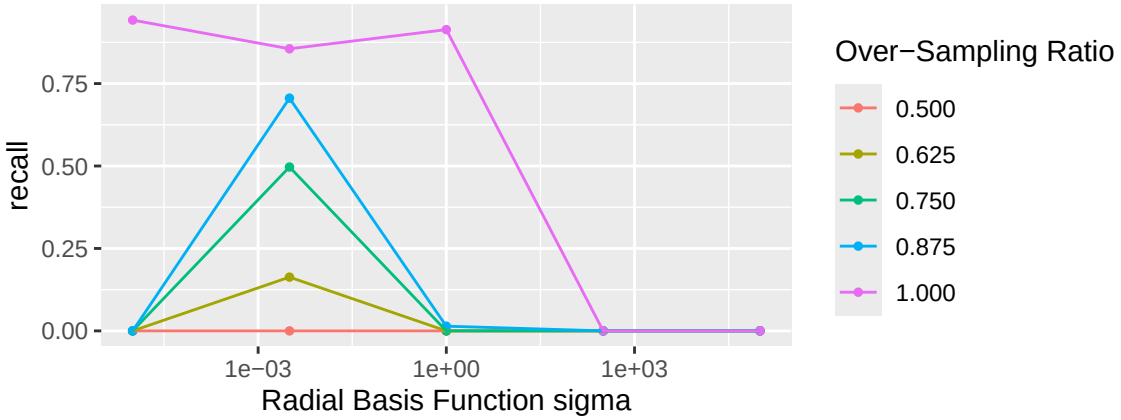
2.6.1 Spécification et recette du modèle

```
svmr_mod <- svm_rbf() |>
  set_mode("classification") |>
  set_engine("kernlab") |>
  set_args(cost = tune(), rbf_sigma = tune())

rec_svmr <- recipe(stroke ~ ., data = data_train) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors()) |>
  step_smote(stroke, over_ratio = tune(), neighbors = 5)
```

- Le **SVM à noyau radial** est une version **non linéaire du SVM** qui permet de séparer les classes de façon plus flexible, en traçant des **frontières de décision non linéaires**. Cela permet au modèle de mieux s'adapter aux données complexes ou non séparables linéairement.
- le paramètre **rbf_sigma** règle la **finesse du modèle** dans la séparation des classes.

2.6.2 Optimisation des hyperparamètres



cost	rbf_sigma
0.03125	1e-05

- L'optimisation a abouti à une valeur de **cost relativement faible** (0.03125), ce qui rend le modèle **plus souple** en acceptant certaines erreurs. Le **rbf_sigma**, quant à lui, est très petit (1e-05), ce qui signifie que le modèle se concentre fortement sur les points très proches (attention au surajustement).

2.6.3 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	51	698
0 (prédit)	1	477

- Le modèle **détecte presque tous les AVC**, avec seulement 1 faux négatif, ce qui donne un **excellent rappel**. En revanche, il produit **beaucoup de fausses alertes**, ce qui peut poser problème, même si cela reste acceptable dans un contexte où il vaut mieux alerter à tort que rater un cas réel.
- c'est notre **meilleur modèle** pour le moment **en terme de recall**.

2.6.4 Performance globale

.metric	.estimate
accuracy	43.0
f_meas	12.7
recall	98.1
precision	6.8
specificity	40.6
roc_auc	83.3
error	57.0

- Le modèle **SVM radial** affiche un **recall presque parfait de 98,1 %**, ce qui signifie qu'il détecte quasiment tous les cas d'AVC. En revanche, son **accuracy est faible** (43 %), en raison d'un grand nombre de fausses alertes. Il s'impose donc comme le **meilleur modèle** si l'on se concentre **uniquement sur le recall**, mais d'un point de vue global, un autre modèle pourrait offrir **un meilleur compromis**, ce que nous verrons en conclusion.

2.7 Arbre de décisions

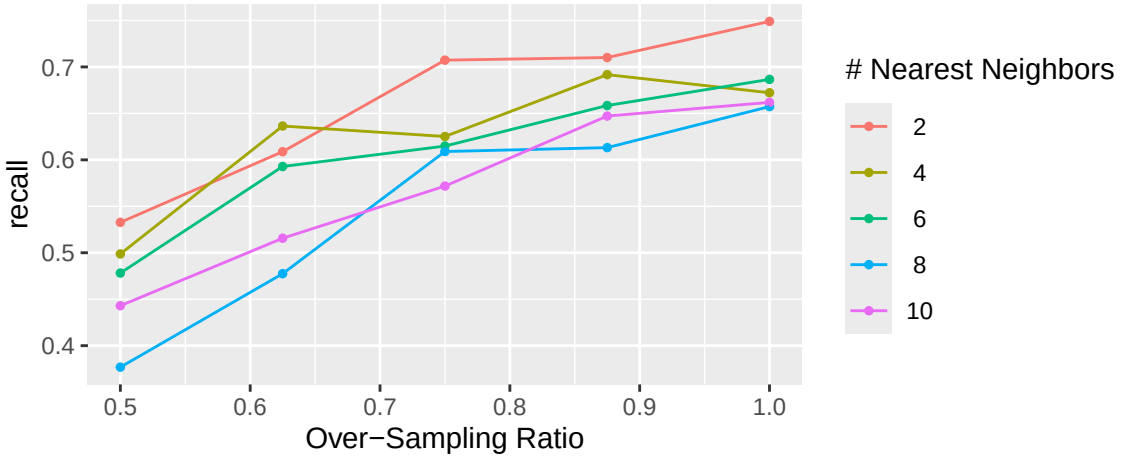
2.7.1 Spécification et recette du modèle

```
dec_tree_mod <- decision_tree(tree_depth = 4, cost_complexity = 0.00001) |>
  set_mode("classification") |>
  set_engine("rpart")

rec_dec_tree <- data_train |>
  recipe(stroke ~ ., strata = stroke) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_smote(stroke, over_ratio = tune(), neighbors = tune())
```

- **L'arbre de décision** est un modèle de classification qui repose sur une série de règles de type "si la condition est respectée alors…" permettant de **segmenter progressivement les données**. À chaque étape, il choisit la variable **qui sépare le mieux les classes**. C'est un modèle simple à comprendre et interpréter mais il peut facilement **surajuster les données** s'il est trop complexe.
- **tree_depht** est un paramètre qui gère **la profondeur de l'arbre**.
- Lorsqu'un arbre est **trop profond**, il risque de **surajuster les données** (overfitting). Le paramètre **cost complexity** consiste à supprimer certaines branches de l'arbre pour le rendre plus simple, et donc **plus général**.

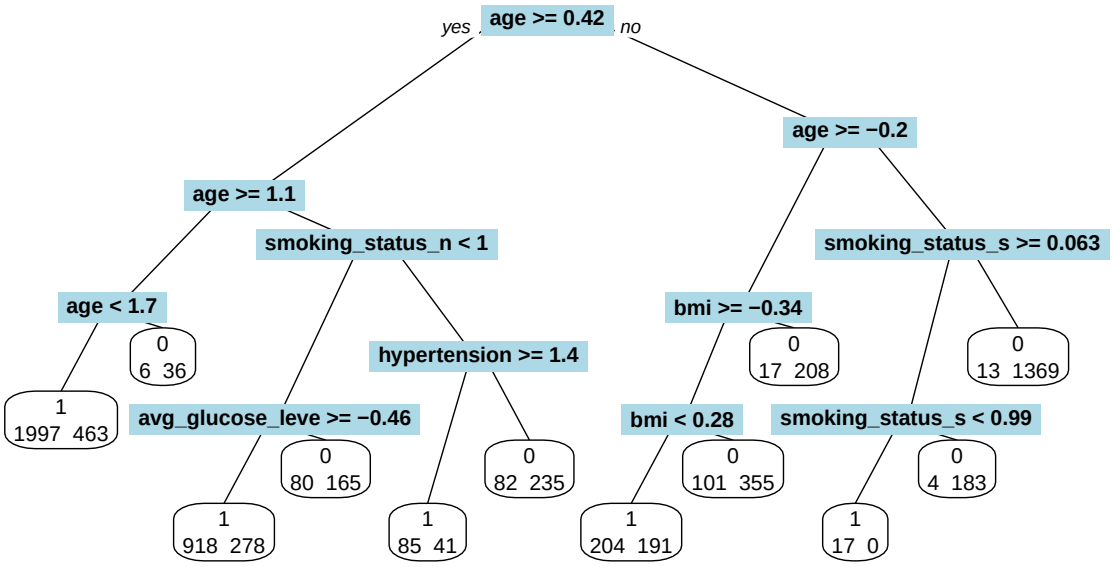
2.7.2 Optimisation des hyperparamètres



over_ratio	neighbors
1	2

- Ici, nous avons uniquement **optimisé les paramètres les moins sensibles**, car l'optimisation de **cost_complexity** et **tree_depth** conduisait soit à un **arbre trop profond**, soit à un arbre **trop simpliste**. En fixant ces paramètres, nous avons obtenu un arbre **plus équilibré** et interprétable.

2.7.3 L'arbre



2.7.4 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	42	361
0 (prédit)	10	814

- Le modèle présente un **recall élevé** (il détecte la plupart des AVC), ce qui est essentiel dans ce contexte. Il rate peu de cas positifs (faux négatifs = 10), ce qui en fait un **modèle pertinent** pour ce type de problème, malgré un grand nombre de faux positifs, acceptables ici. C'est un **bon modèle** mais **pas le meilleur** comme nous l'avons vu auparavant.

2.7.5 Performance globale

.metric	.estimate
accuracy	69.8
f_meas	18.5
recall	80.8
precision	10.4
specificity	69.3
roc_auc	83.2
error	30.2

Ici, nous observons un modèle **globalement performant** dans notre contexte : un recall de **80,8 %**, une accuracy proche de **70 %**, et une aire sous la courbe ROC de **83,2 %**, indiquant une bonne capacité de discrimination.

2.8 Forêt aléatoire

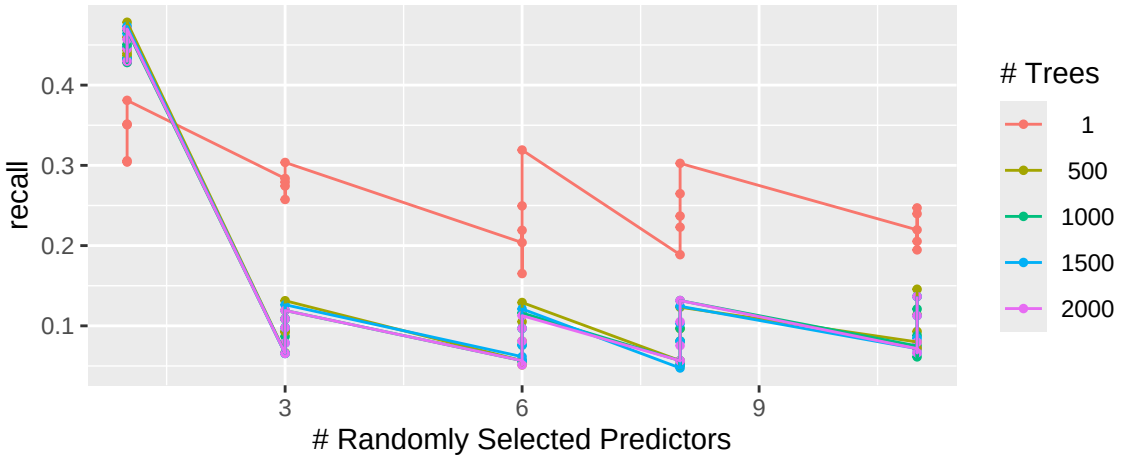
2.8.1 Spécification et recette du modèle

```
rf_mod <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) |>
  set_engine("ranger", importance = "impurity") |>
  set_mode("classification")

rf_rec <- recipe(stroke ~ ., data = data_train) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_smote(stroke, over_ratio = 1, neighbors = 5, skip = TRUE)
```

- La **forêt aléatoire** est un modèle qui combine **plusieurs arbres de décision**. Chaque arbre est construit sur un **échantillon aléatoir**e des données et, à chaque nœud, sur un sous-ensemble aléatoire de variables.
- Le paramètre importance = **"impurity"** permet de mesurer **l'importance des variables** selon la réduction de **l'impureté (indice de Gini)** qu'elles apportent dans les arbres : plus une variable permet de bien séparer les classes, plus elle est jugée importante.
- **min_n** : c'est le nombre minimum d'observations qu'un nœud doit contenir pour pouvoir être divisé.
- **mtry** : c'est le nombre de variables que l'on teste à chaque division d'un arbre.

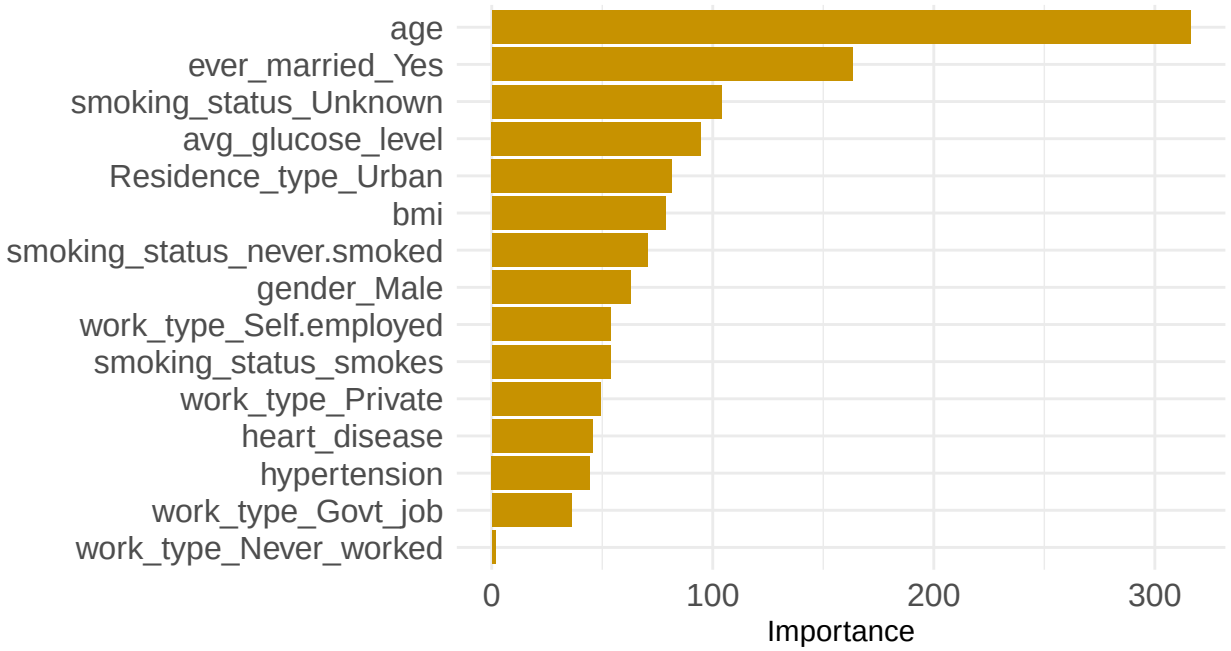
2.8.2 Optimisation des hyperparamètres



mtry	trees	min_n
1	500	40

- L'optimisation a conduit à une valeur de **mtry = 1**, ce qui signifie que chaque arbre teste une seule variable à chaque division. Avec **min_n = 40**, les arbres sont empêchés de trop se complexifier. Le modèle s'appuie sur **500 arbres**, un nombre suffisant pour assurer une bonne stabilité des résultats.

2.8.3 Importance des variables



- Ici, **l'importance des variables est plus équilibrée**, bien qu'elle soit toujours fortement dominée par la variable âge.

2.8.4 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	28	186
0 (prédit)	24	989

- Le modèle a l'air d'afficher une **bonne accuracy** au vu d'une bonne détection des non AVC. En revanche, le **recall reste trop faible** (seulement 28 vrais positifs sur 52), ce qui est problématique dans notre contexte, où l'objectif principal est de ne pas rater les cas d'AVC. **Ce modèle ne répond donc pas suffisamment à nos attentes**.

2.8.5 Performance globale

.metric	.estimate
accuracy	82.9
f_meas	21.1
recall	53.8
precision	13.1
specificity	84.2
roc_auc	80.3
error	17.1

Malgré une accuracy de **82,9%** et une aire sous la courbe ROC de **80,3%**, qui traduisent une performance globale correcte, **ce modèle n'est pas adapté** à notre objectif de détection d'AVC. En effet, son **recall** de seulement **53,8%** signifie qu'il échoue à identifier près de la moitié des cas positifs, ce qui est **inacceptable dans un contexte médical** où l'enjeu principal est d'éviter les faux négatifs.

2.9 Boosting

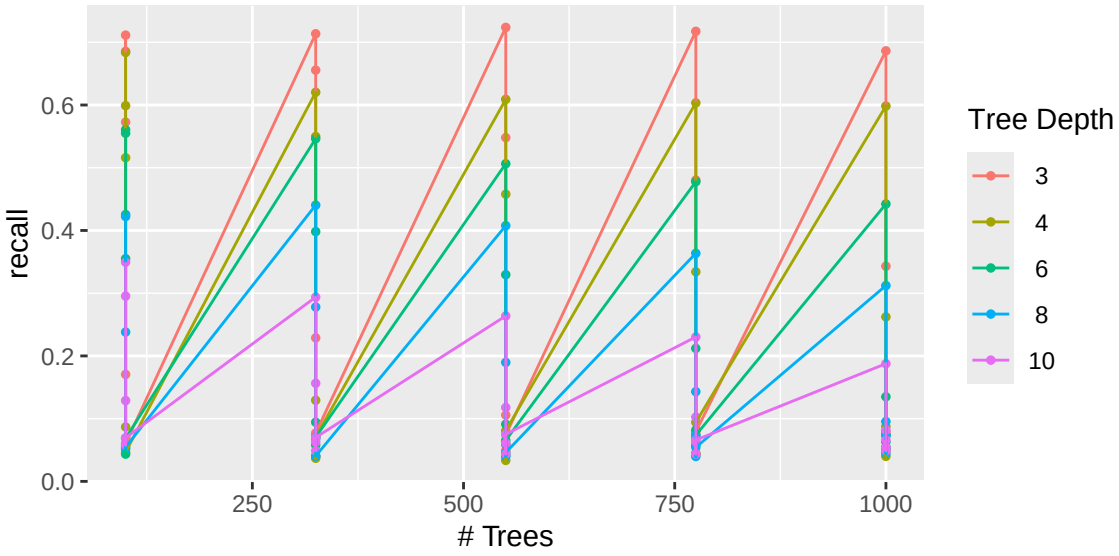
2.9.1 Spécification et recette du modèle

```
boost_mod <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune()
) |>
  set_mode("classification") |>
  set_engine("xgboost")

boost_rec <- recipe(stroke ~ ., data = data_train) |>
  step_center(all_numeric_predictors()) |>
  step_scale(all_numeric_predictors()) |>
  step_smote(stroke, over_ratio = 0.8, neighbors = 5)
```

- Le **boosting** est une **méthode d'apprentissage supervisé** qui consiste à combiner plusieurs modèles simples pour former un modèle final puissant. Chaque nouveau modèle est entraîné pour corriger les erreurs commises par les précédents, en accordant plus d'importance aux observations mal classées. Cette approche permet **d'obtenir un modèle très performant**, souvent plus précis que les méthodes classiques.
- le paramètre **trees** correspond aux nombres d'arbes.
- Le paramètre **learn_rate** contrôle la vitesse d'apprentissage du modèle de boosting. Il **détermine l'impact de chaque nouvel arbre ajouté** à l'ensemble final.

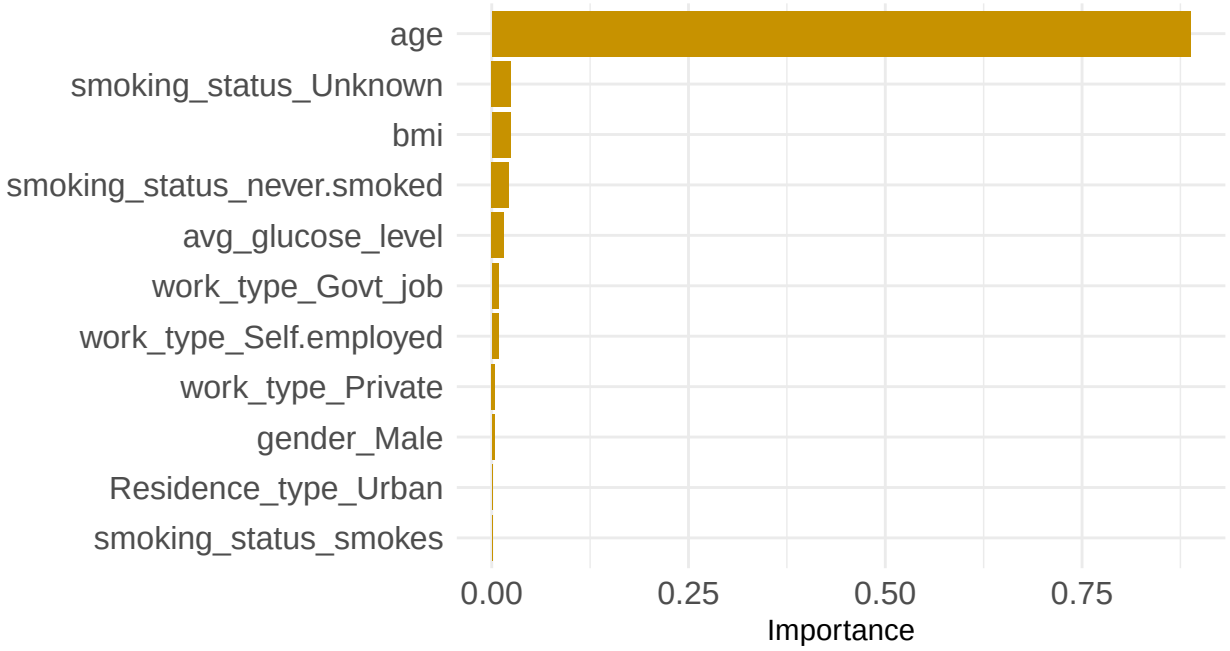
2.9.2 Optimisation des hyperparamètres



tree_depth	trees	learn_rate
3	550	0.001

- Le **modèle final** utilise une **profondeur d'arbre de 3**, avec **550 arbres construits**, et un **learn_rate** très faible (0.001). Ce faible taux d'apprentissage permet une construction progressive, rendant le **modèle plus stable** et performant à long terme.

2.9.3 Importance des variables



- Le **graphique d'importance** basé sur le **critère de Gini** montre que **l'âge domine largement** les autres variables, concentrant presque toute l'importance du modèle. Cela signifie que le modèle s'appuie essentiellement sur cette seule variable pour prédire les AVC.
- Un modèle plus performant **intégrerait plusieurs variables discriminantes**, pas uniquement l'âge.

2.9.4 Matrice de confusion

	1 (réel)	**0 (réel)**
1 (prédit)	43	375
0 (prédit)	9	800

- Le modèle présente un **bon recall**, ce qui est essentiel dans notre contexte. Bien qu'il génère un nombre important de faux positifs, cela reste acceptable. **Le modèle à l'air globalement bon.**

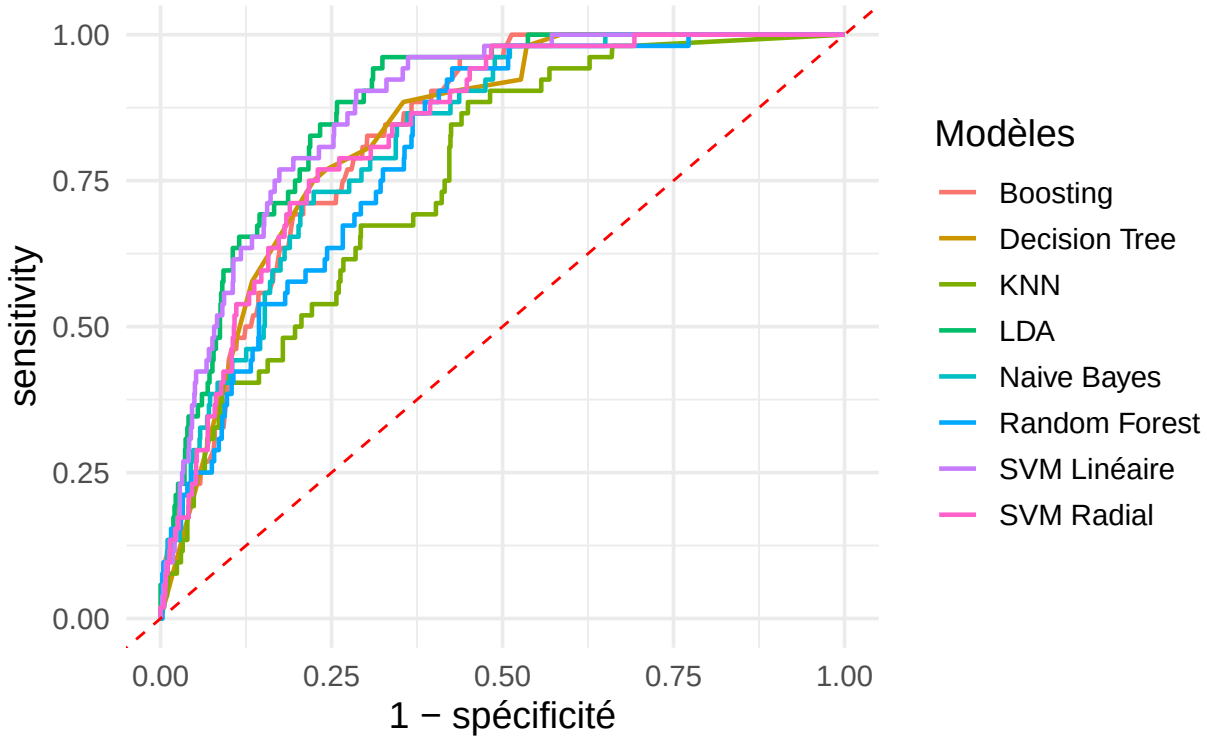
2.9.5 Performance globale

.metric	.estimate
accuracy	68.7
f_meas	18.3
recall	82.7
precision	10.3
specificity	68.1
roc_auc	83.2
error	31.3

- Avec un **recall** de **82,7 %**, une **accuracy** proche **70 %** et une aire sous la **courbe ROC** de **83,2 %**, le modèle montre des **performances globalement bonnes**. Toutefois, pour un modèle de **boosting**, on aurait pu s'attendre à de meilleurs résultats. Nous reviendrons sur ces limites dans la conclusion.

3 Choix du modèle

3.1 Les courbes ROC



Ici, on observe que les modèles ayant la plus grande aire sous la courbe ROC sont **la LDA et le SVM linéaire**, c'est-à-dire nos **deux modèles linéaires**.

3.2 Les performances globales

model	accuracy	f_meas	recall	precision	specificity	roc_auc	error
SVM Radial	43.0	12.7	98.1	6.8	40.6	83.3	57.0
SVM Linéaire	68.0	19.3	90.4	10.8	67.0	87.3	32.0
LDA	72.3	21.3	88.5	12.1	71.6	87.7	27.7
Boosting	68.7	18.3	82.7	10.3	68.1	83.2	31.3
Decision Tree	69.8	18.5	80.8	10.4	69.3	83.2	30.2
KNN	76.9	16.5	53.8	9.7	77.9	75.9	23.1
Random Forest	82.9	21.1	53.8	13.1	84.2	80.3	17.1
Naive Bayes	89.4	23.5	38.5	16.9	91.7	82.1	10.6

3.3 Conclusion

- L'ensemble des modèles testés présente des **performances variables** en fonction des métriques observées. Dans le cadre spécifique de la **détection des AVC**, où le recall est prioritaire afin de ne **manquer aucun cas positif**, les modèles les plus performants sont **la LDA et le SVM linéaire**, avec des scores de recall respectifs de **88,5%** et **90,4%**.

- Bien qu'ils ne soient pas **les modèles les plus complexes**, ces deux méthodes linéaires montrent des **résultats solides et cohérents**. Cela montre que la séparation entre les classes dans nos données est **probablement linéaire**, ce qui rend ces approches particulièrement bien adaptées à notre cas. Contrairement à certaines attentes, des modèles plus sophistiqués comme le **Random Forest** ou le **Boosting** n'offrent pas ici de **gain significatif** sur le critère principal (recall), et présentent même un **recall inférieur**, ce qui les rend **moins pertinents** dans notre contexte (ils restent tout de même de bons modèles).

- En résumé, **la LDA et le SVM linéaire** apparaissent comme des **choix efficaces**, à la fois pour leur **simplicité**, leur **interprétabilité** et leur **capacité à détecter** correctement **les cas d'AVC** dans notre jeu de données.