Chapitre 4
Les bases du langage

1. Introduction

Comme tous les langages de programmation, C# impose certaines règles au développeur. Ces règles se manifestent au travers de la syntaxe du langage mais elles couvrent le large spectre fonctionnel proposé par C#. Avant d'explorer en profondeur les fonctionnalités du langage au chapitre suivant, nous étudierons donc les notions essentielles et fondamentales de C# : la création de données utilisables par une application et le traitement de ces données.

2. Les variables

Les données utilisables dans un programme C# sont représentées par des variables. Une variable est un espace mémoire réservé auquel on assigne arbitrairement un nom et dont le contenu est une valeur dont le type est fixé. On peut manipuler ce contenu dans le code en utilisant le nom de la variable.

2.1 Nommage des variables

La spécification du langage C# établit quelques règles à prendre en compte lorsque l'on nomme une variable :

– Le nom d'une variable ne peut comporter que des chiffres, des caractères de l'alphabet latin accentués ou non, le caractère ç ou les caractères spéciaux _ et μ.

– Le nom d'une variable ne peut en aucun cas commencer par un chiffre. Il peut en revanche en comporter un ou plusieurs à toute autre position.

– Le langage est sensible à la casse, c'est-à-dire qu'il fait la distinction entre majuscules et minuscules : les variables `unevariable` et `uneVariable` sont donc différentes.

– La longueur d'un nom de variable est virtuellement illimitée : le compilateur ne remonte pas d'erreur lorsqu'il est composé de 30 000 caractères ! Il n'est évidemment pas recommandé d'avoir des noms de variables aussi longs, le maximum en pratique étant plus souvent de l'ordre de la trentaine de caractères.

– Le nom d'une variable ne peut pas être un mot-clé du langage. Il est toutefois possible de préfixer un mot-clé par un caractère autorisé ou par le caractère @ pour utiliser un nom de variable similaire.

Les noms de variables suivants sont acceptés par le compilateur C# :

– `maVariable`

– `maVariableNumero1`

– `@void` (`void` est un mot-clé de C#)

– `µn3_VàRïãbl3`

De manière générale, il est préférable d'utiliser des noms de variables explicites, c'est-à-dire permettant de savoir à quoi correspond la valeur stockée dans la variable, comme `nomClient`, `montantAchat` ou `ageDuCapitaine`.

2.2 Type des variables

Une des caractéristiques de C# est la notion de typage statique : chaque variable correspond à un type de données et ne peut en changer. De plus, ce type doit être déterminable au moment de la compilation.

2.2.1 Types valeurs et types références

Les différents types utilisables en C# peuvent être décomposés en deux familles : les types valeurs et les types références. Cette notion peut être déconcertante au premier abord, puisqu'une variable représente justement une donnée et donc une valeur. Ce concept est en fait lié à la manière dont est stockée l'information en mémoire.

Lorsque l'on utilise une variable de type valeur, on accède directement à la zone mémoire stockant la donnée. Au moment de la création d'une variable de type valeur, une zone mémoire de la taille correspondant au type est allouée. Chaque octet de cette zone est automatiquement initialisé avec la valeur binaire 00000000. Notre variable aura donc une suite de 0 pour valeur binaire.

Dans le cas d'une variable de type référence, le comportement est différent. La zone mémoire allouée à notre variable contient une adresse mémoire à laquelle est stockée la donnée. On passe donc par un intermédiaire pour accéder à notre donnée. L'adresse mémoire est initialisée avec la valeur spéciale `null`, qui ne pointe sur rien, tandis que la zone mémoire contenant les données n'est pas initialisée. Elle sera initialisée lorsque la variable sera instanciée. Dans le même temps, l'adresse mémoire stockée dans notre variable sera mise à jour.

Une variable de type référence pourra donc ne contenir aucune donnée, tandis qu'une variable de type valeur aura forcément une valeur correspondant à une suite de 0 binaires.

Cette différence de fonctionnement a une conséquence importante : la copie de variable se fait par valeur ou par référence, ce qui signifie qu'une variable de type valeur sera effectivement copiée, tandis que pour un type référence, c'est l'adresse que contient la variable qui sera copiée, et il sera donc possible d'agir sur la donnée réelle indifféremment à partir de chacune des variables pointant sur ladite donnée.

2.2.2 Types intégrés

La plateforme .NET embarque plusieurs milliers de types différents utilisables par les développeurs. Parmi ces types, nous en avons une quinzaine que l'on peut considérer comme fondamentaux : ce sont les types intégrés (aussi nommés types primitifs). Ce sont les types de base à partir desquels sont construits les autres types de la BCL ainsi que ceux que le développeur crée dans son propre code. Ils permettent de définir des variables contenant des données très simples.

Ces types ont comme particularité d'avoir chacun un alias intégré à C#.

Types numériques

Ces types permettent de définir des variables numériques entières ou décimales. Ils couvrent des plages de valeurs différentes et ont chacun une précision spécifique. Certains types seront donc plus adaptés pour les calculs entiers, d'autres pour les calculs dans lesquels la précision décimale est très importante, comme les calculs financiers.

Les différents types numériques sont énumérés ci-dessous avec leurs alias ainsi que les plages de valeurs qu'ils couvrent.

Type	Alias C#	Plage de valeurs couverte	Taille en mémoire
`System.Byte`	`byte`	0 à 255	1 octet
`System.SByte`	`sbyte`	-128 à 127	1 octet
`System.Int16`	`short`	-32768 à 32767	2 octets
`System.UInt16`	`ushort`	0 à 65535	2 octets
`System.Int32`	`int`	-2147483648 à 2147483647	4 octets
`System.UInt32`	`uint`	0 à 4294967295	4 octets
`System.Int64`	`long`	-9223372036854775808 à 9223372036854775807	8 octets
`System.UInt64`	`ulong`	0 à 18446744073709551615	8 octets
`System.Single`	`float`	±1,5e-45 à ±3,4e38	4 octets

Type	Alias C#	Plage de valeurs couverte	Taille en mémoire
`System.Double`	`double`	±5,0e-324 à ±1,7e308	8 octets
`System.Decimal`	`decimal`	±1,0e-28 à ±7,9e28	16 octets

Les types numériques primitifs sont tous des types valeurs. Une variable de type numérique et non initialisée par le développeur aura pour valeur par défaut 0.

Remarque

.NET 4 a apporté le type `System.Numerics.BigInteger` afin de manipuler des entiers d'une taille arbitraire. Ce type est aussi un type valeur, mais il ne fait pas partie des types intégrés.

Les valeurs numériques utilisées au sein du code peuvent être définies à l'aide de trois bases numériques :

– **Décimale** (base 10) : c'est la base numérique utilisée par défaut.

  Exemple : 280514

– **Héxadécimale** (base 16) : elle est très utilisée dans le monde du Web, pour la définition de couleurs, et plus généralement pour encoder des valeurs numériques dans un format plus condensé. **Une valeur héxadécimale est écrite en la préfixant par 0x.**

  Exemple : 0x0447C2

– **Binaire** (base 2) : cette base est celle qui permet de s'approcher au plus près de la manière dont la machine interprète le code compilé. L'utilisation du binaire en C# peut par conséquent se révéler précieuse dans le cadre d'optimisations (décalages de bits au lieu de multiplications par 2, stockage de multiples données au sein d'une même variable, etc.). **Les valeurs binaires sont préfixées par 0b.**

  Exemple : 0b01000100011111000010

La lecture de valeurs numériques peut s'avérer difficile dans certains cas, notamment lorsqu'elles sont représentées sous forme binaire. L'équipe en charge des évolutions de C# a intégré, dans la septième version du langage, la possibilité d'ajouter des séparateurs dans les valeurs numériques afin d'améliorer la lisibilité globale. Le caractère utilisé pour cela est _ :

– 280_514
– 0x04_47_C2
– 0b0100_0100_0111_1100_0010

Types textuels

Il existe dans la BCL deux types permettant de manipuler des caractères Unicode et des chaînes de caractères Unicode : `System.Char` et `System.String`. Ces types ont respectivement pour alias `char` et `string`.

Le type `char` est un type valeur encapsulant les mécanismes nécessaires au traitement d'un caractère Unicode. Par conséquent, une variable de type char est stockée en mémoire sur deux octets.

Les valeurs de type `char` doivent être encadrées par les caractères ' ' : 'a'.

Certains caractères ayant une signification particulière pour le langage doivent être utilisés avec le caractère d'échappement \ afin d'être correctement interprétés. D'autres caractères n'ayant pas de représentation graphique doivent être aussi déclarés avec des séquences spécifiques. Le tableau suivant résume les séquences qui peuvent être utilisées.

Séquence d'échappement	Caractère associé
\'	Simple quote '
\"	Double quote "
\\	Backslash \
\a	Alerte sonore
\b	Retour arrière
\f	Saut de page
\n	Saut de ligne

Chapitre 5
Quand la machine se met à apprendre

1. Introduction

Que l'on ne se méprenne pas : l'apprentissage machine est un métier, un sacerdoce mathématique et informatique, et un formidable jeu intellectuel. C'est une des facettes les plus fascinantes de l'informatique, une des plus porteuses d'espoirs, de craintes, mais surtout d'incompréhension totale.

On fait tout dire à l'IA, et indirectement à l'apprentissage machine, et on lui fait porter tous les maux futurs de l'humanité. Maintenant, vous avez vu dans les pages précédentes que les machines sont loin d'être parfaites. Les services existants s'améliorent constamment, mais on est encore loin d'une intelligence artificielle générale. Cependant, on ne peut plus ignorer les services d'IA. Nous savons qu'appeler Rekognition, Translate ou Transcribe est une affaire de quelques lignes de code. Il en est de même avec les services de SageMaker ou d'Amazon Machine Image (AMI).

Ce chapitre leur est consacré. Vous allez y apprendre, avec un peu plus de détails qu'au chapitre Mettre en œuvre des projets d'IA avec AWS qui n'était qu'une introduction, comment construire un modèle, le former et y lancer des inférences. Que vous souhaitiez faire de la détection de défectuosités sur une chaîne de montage, de l'analyse de sentiments ou des prévisions de chiffre d'affaires, ce chapitre est là pour ça.

Avant tout, quelques mots de précautions. Nous avons parcouru un long chemin depuis la première page de ce livre. Cependant, dans ce chapitre, nous changeons de braquet, car la pente est ardue. Pour pouvoir suivre les notions qui seront abordées, il vous faut les connaissances suivantes :

– Docker et les conteneurs.
– Python.
– Les blocs-notes Jupyter.
– Le CLI AWS.
– S3.

Et pour couronner le tout, vous devez avoir une curiosité et une résistance au stress hors norme. Curiosité, car il va falloir en faire preuve pour aller creuser la signification de tel ou tel hyperparamètre, ou comprendre pourquoi on choisit sept plutôt que six couches de convolution ou pourquoi tel framework et pas tel autre. Il est impossible d'expliquer tous ces choix dans les pages suivantes, d'autant que la littérature en ligne vous permettra facilement de trouver la réponse à toutes vos questions. Je donne les pointeurs pour vous y aider, comme je l'ai déjà fait dans les pages précédentes.

Enfin, vous devez résister au stress parce que la machine learning :

– ne marche jamais tout à fait comme on veut et c'est souvent décevant, en revanche, quand on tient un modèle correctement formé qui fait des prévisions aux petits oignons, c'est le pied !
– réclame beaucoup de rigueur. Un paramètre manquant ou avec une valeur incorrecte et c'est tout un modèle qui ne peut pas se former. Parfois, ce sont des heures, voire des jours, à chercher le pourquoi du comment. Patience et longueur de temps font plus que force ni que rage. Cette maxime de La Fontaine s'applique bien au pratiquant de l'apprentissage machine.

Vous voilà prévenu. Bon voyage !

2. Machine et Deep Learning

Nous avons étudié au chapitre Intelligence artificielle, mythes et réalités que le machine learning (apprentissage machine) était un sous-ensemble de l'intelligence artificielle. Le *deep learning* (apprentissage profond) est un sous-ensemble de l'apprentissage machine comme le montre l'image ci-dessous :

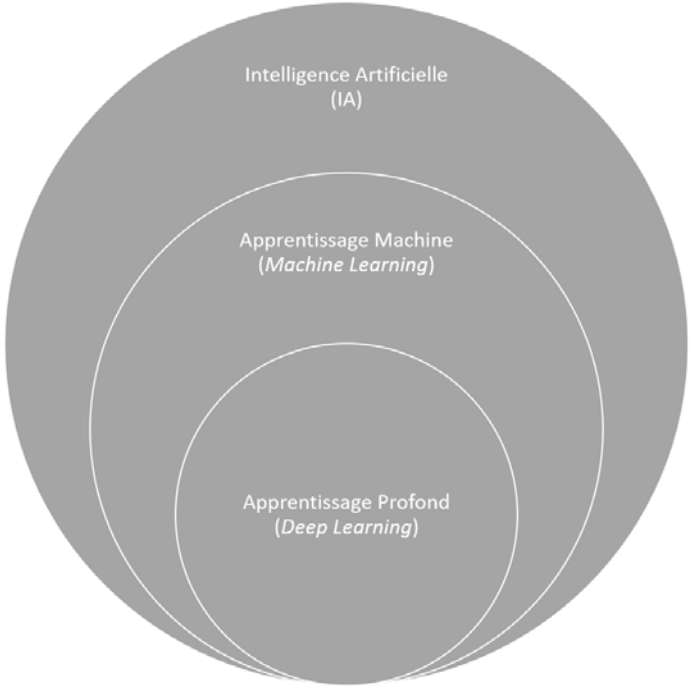

Figure 1 - Intelligence artificielle

On peut donc travailler avec de l'intelligence artificielle sans faire de l'apprentissage machine, mais pas l'inverse, bien que tout cela soit avant tout affaire de sémantique. Cependant, tout ceci n'a que peu d'importance, l'idée étant d'obtenir un résultat à partir d'algorithmes plus ou moins sophistiqués. La discipline qui a le vent en poupe en ce moment est l'apprentissage, supervisé ou non. Elle consiste à faire en sorte que la machine apprenne soit à partir de règles prédéfinies (les heuristiques), soit à partir de jeux de données, étiquetées ou non (les jeux d'apprentissage). Le résultat est un modèle sur lequel on va pouvoir faire des inférences, c'est-à-dire tester des hypothèses et obtenir des prévisions ou de simples réponses.

L'intelligence artificielle a envahi nos vies numériques sans que l'on s'en aperçoive. Nos ordinateurs, téléphones portables et tablettes en sont remplis. Les logiciels de tout acabit s'y mettent aussi afin d'en améliorer la sécurité, la simplicité d'utilisation et la puissance. Les réseaux sociaux ne pourraient pas continuer à exister sans elle, ses prévisions, ses recommandations et ses influences pour nous rendre toujours plus captifs. Les places de marchés et magasins en ligne abreuvent leurs modèles d'IA de nos données, de nos comportements et de nos clics. En bref, l'IA est partout autour de nous sans qu'on ne la voie, la sente ou se rende compte de son omniprésence.

La question étant, et comment faire pour en tirer parti ? L'apprentissage machine n'est pas une discipline simple. Elle s'est construite à l'intersection des mathématiques et de l'informatique. Elle fait appel à de nombreuses autres disciplines comme les probabilités, les statistiques, l'algèbre matricielle et vectorielle, les bases de données, le développement logiciel, pour n'en citer que quelques-unes, parcourues au pas de course au chapitre Intelligence artificielle, mythes et réalités. L'objectif de cette section est de rentrer dans le détail de la construction d'un modèle d'apprentissage machine afin de pouvoir utiliser SageMaker ou les AMI relativement confortablement.

Nous allons donc voir dans les pages suivantes ce qu'est un modèle prédictif, comment préparer les données pour en construire un et l'entraîner. Ensuite, nous verrons comment le déployer et pouvoir y faire des inférences. Nous terminerons par l'analyse des résultats obtenus et comment modifier nos modèles afin de les améliorer. L'illustration ci-après vous donne un aperçu des différentes étapes nécessaires à l'utilisation d'un modèle d'apprentissage machine.

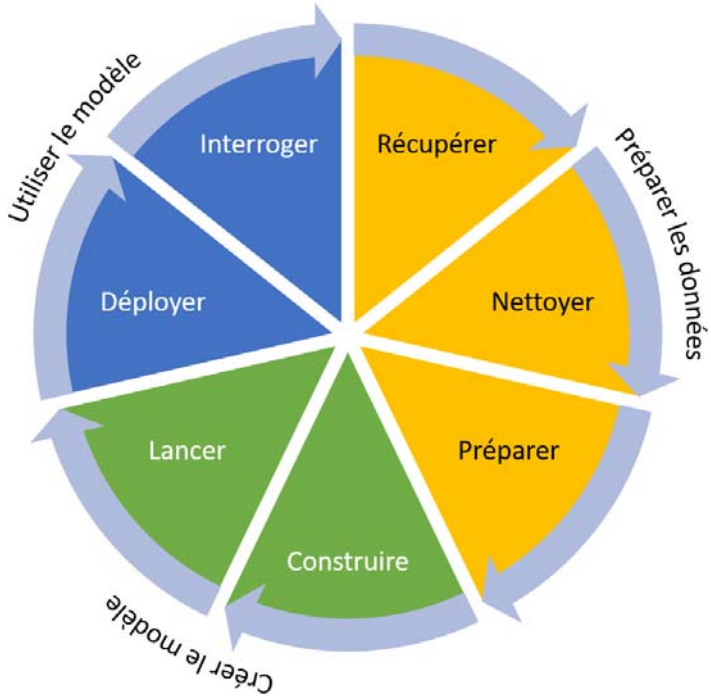

Figure 2 - Cycle de création d'un modèle d'apprentissage machine

Un dernier mot avant de rentrer dans le vif du sujet. L'apprentissage machine n'est pas la panacée à tous les maux. Il est parfois décevant, les résultats n'étant pas toujours à la hauteur des attentes. C'est une des raisons, sans doute, pour lesquelles on parle plus souvent de prédiction que de prévision. Prédire est un peu surnaturel (artificiel peut-être), prévoir a une connotation plus scientifique, probabiliste, pourtant, il s'agit bien de prévision et non de prédiction, lorsqu'on souhaite tirer profit des résultats obtenus. Peu importe le terme employé, de toute façon les résultats obtenus seront toujours numériques et probabilistes. Cela signifie qu'il y aura des erreurs et des incertitudes, mais comme l'a dit Henri Poincaré, le célèbre mathématicien :

« Le nom seul de calcul des probabilités est un paradoxe : la probabilité, opposée à la certitude, c'est ce qu'on ne sait pas, et comment peut-on calculer ce que l'on ne connaît pas ? »

C'est sans doute un des grands défis de l'intelligence artificielle : tenter de prévoir l'inconnu, c'est s'exposer à la critique.

2.1 Données, algorithmes et apprentissages

Dans le chapitre Intelligence artificielle, mythes et réalités, nous avons vu les différents types d'apprentissages : supervisé, non supervisé, semi-supervisé et par renforcement. Nous avons aussi abordé les différents modèles existant : la détection d'anomalie, le partitionnement (clustering), la régression (prédiction et prévision) et la classification. Faisons le lien entre ces notions et SageMaker : le tableau suivant en donne une synthèse, du besoin exprimé à l'algorithme SageMaker :

Besoin	Apprentissage	Algorithme SageMaker
Prévision et prédiction	Supervisé	Apprentissage linéaire
		Machine de factorisation
		K plus proches voisins
		Objetc2Vec
		Prévisions DeepAR
		Seq2Seq
		XGBoost
	Non supervisé	IP Insights
Classification	Supervisé	Apprentissage linéaire
		Blazing Text (Classification)
		Classification d'images
		Détection d'objets
		Machine de factorisation
		K plus proches voisins
		Objetc2Vec (classification)
		Segmentation sémantique