Note de machine learning

DJEBALI Wissam

4 novembre 2018

1 Cross Validation

(à compléter)

2 Loss Functions

2.1 Loss Functions

$\boldsymbol{y}$ est la vraie valeur et $\boldsymbol{y'}$ est la valeur prédite.

- $\boldsymbol{L_{\text{0-1}}(y, y') = \mathbb{I}_{yy' \leq 0}}$
- $\boldsymbol{L_{\text{hinge}}(y, y') = (1 - yy')_+}$
- $\boldsymbol{L_{\text{logistic}}(y, h(x)) = Log_{\text{loss}}(y, h(x)) = log(1+e^{-y \times h(x)}) = \sum_{(x,y) \in D} -y \log(h(x)) - (1-y) \log(1-h(x))}$
- $\boldsymbol{L_{\text{Quadratic Hingle Loss}}(y, y') = \frac{1}{2}(1 - yy')^2_+}$
- $\boldsymbol{L_{\text{Huber loss}}(y, y') = -4yy' \mathbb{I}_{yy' \leq -1} + (1 - yy')^2_+ \mathbb{I}_{yy' \geq -1}}$
- $\boldsymbol{L_{\text{least-squares}}(y, y') = \frac{1}{2}(y - y')^2}$ (régression linéaire)

2.2 Loss Functions with Regularization

Au lieu de seulement chercher à minimiser la perte (minimisation du risque empirique), comme suit :

$$\boldsymbol{minimiser(Perte(Données|Modèle))}$$

Pour éviter le **surapprentissage(overfitting)**, nous allons minimiser à la fois la perte et la complexité, ce qui s'appelle la **minimisation du risque structurel** :

$$\boldsymbol{minimiser(Perte(Données|Modèle)) + complexité(Modèle)}$$

Notre algorithme d'optimisation de l'apprentissage est désormais une fonction à deux facteurs : le **facteur de perte**, qui mesure l'efficacité d'apprentissage du modèle, et le **facteur de régularisation**, qui mesure la complexité du modèle.

Il y a deux façon courantes(et apparentées) d'aborder la complexité d'un modèle :

- La complexité du modèle en tant que fonction des pondérations de toutes les caractéristiques du modèle.

- a complexité du modèle en tant que fonction du nombre total de caractéristiques dont les pondérations sont différentes de zéro

Si la complexité du modèle est fonction des pondérations, la pondération d'une caractéristique est plus ou moins complexe selon que sa valeur absolue est plus ou moins élevée.

- **Régression Ridge(Régression avec pénalisation(régularisation) norme L_2**

  Nous pouvons quantifier la complexité à l'aide de la formule de $\boldsymbol{L_2}$, qui définit le facteur de régularisation comme étant la somme des carrés de toutes les pondérations des caractéristiques :

  $$L_2 \textbf{ facteur de régularisation} = ||w||_2^2 = w_1^2 + w_2^2 + w_3^2 + ... + w_n^2$$

  Dans cette formule, les pondérations proches de zéro ont peu d'effet sur la complexité du modèle, tandis que celles qui correspondent à des anomalies peuvent avoir un effet considérable.

  Les développeurs de modèles ajustent l'effet global du facteur de régularisation en multipliant sa valeur par une grandeur scalaire appelé **lambda**(ou **taux de régularisation**). Ils cherchent à obtenir le résultat suivant :

  $$\boldsymbol{minimiser(Perte(Données|Modèle) + \lambda complexité(Modèle))}$$

  Avec la régularisation L_2 on veut :

  $$\boldsymbol{min(func_{Loss} + \lambda||w||_2^2)}$$

  La régularisation L_2 a les effets suivants sur un modèle :
  - Les valeurs des pondérations se rapprochent de (mais ne sont pas exactement égales à) zéro.
  - a moyenne des pondérations tend vers zéro, avec une distribution normale (en cloche ou gaussienne).

  L'augmentation du lambda renforce les effets de la régularisation.

  Le lambda choisi doit permettre d'obtenir un juste équilibre entre simplicité et efficacité d'apprentissage :
  - Si le lambda est trop élevé, le modèle sera simple, mais il présentera un risque de sous-apprentissage des données. Le modèle n'en apprendra pas suffisamment sur les données d'apprentissage pour réaliser des prédictions utiles.
  - Si le lambda est trop faible, le modèle sera plus complexe, et il présentera un risque de surapprentissage des données. Le modèle apprendra trop sur les spécificités des données d'apprentissage, et il ne pourra pas généraliser celles-ci à de nouvelles données.

  Remarque : La définition du lambda sur zéro supprime complètement la régularisation. Dans ce cas, l'apprentissage vise exclusivement à minimiser la perte, et le risque de surapprentissage est maximal.

  Le lambda idéal produit un modèle qui effectue une généralisation efficace en cas de nouvelles données qui n'étaient pas visibles précédemment. Le lambda idéal dépend des données, on doit donc malheureusement procéder à des réglages.

- **Régression LASSO(Régression avec pénalisation(régularisation) norme L_1**
  Même principe que la **Régression Ridge** mais en utilisant la **norme L_1**. C'est une pénalisation plus sévère que **Ridge**
  .
  Avec la régularisation L_1 on veut :

$$\boldsymbol{min(func_{Loss} + \lambda||w||_1)}$$

  La régularisation L_1 a les effets suivants sur un modèle :
  - Fixe les valeurs de certaines pondérations à zéro.
  - Supprime le poids de certaines variables dans le modèle.

  Pour le reste on retrouve les mêmes interprétations.

  A noter que les normes L_1 et L_2 permettent de garder un problème convexe et donc résolvable.

2.2.1 Remarque :

⋆ La dérivée de $\boldsymbol{L_2}$ est comparable à une force qui retire x % de la pondération à chaque fois. Comme le savait Zénon, même si vous retirez x % d'un nombre plusieurs milliards de fois, le nombre ainsi réduit ne sera jamais égal à 0. (Zénon n'était pas conscient de la précision limitée des calculs en virgule flottante, qui peuvent produire un résultat exactement égal à 0). Quoi qu'il en soit, la régularisation $\boldsymbol{L_2}$ ne ramène généralement pas les pondérations à 0.

⋆ La dérivée de $\boldsymbol{L_1}$ est comparable à une force qui soustrait une constante de la pondération à chaque fois. Toutefois, grâce aux valeurs absolues, $\boldsymbol{L_1}$ présente une discontinuité à 0, si bien que les résultats de la soustraction qui sont inférieurs à 0 sont mis à 0. Par exemple, si la soustraction fait passer une pondération de +0,1 à -0,2, $\boldsymbol{L_1}$ définit cette pondération sur exactement 0. $\boldsymbol{L_1}$ met donc la pondération à 0, ce qui correspond exactement au but recherché.

En pénalisant la valeur absolue de toutes les pondérations, la régularisation $\boldsymbol{L_1}$ s'avère particulièrement efficace pour les modèles larges.

- **ElasticNet**
  Combinaison des fonction de **la Régression Ridge** et de la **Régression LASSO**.

3 Kernel(Utilisés dans SVM)

- **The RBF Kernel**(Radial Basis Function)
  Pour $\gamma > 0$

$$\boldsymbol{K(x, x') = \exp\left(-\gamma \left\|x - x\right\|_2^2\right)}$$

- **The Tanh Kernel** aussi appelé **The Sigmoid Kernel**

$$\boldsymbol{K'(x, x') = tanh(a\langle x, x'\rangle + c) = \frac{e^{a\langle x,x'\rangle+c} - e^{a\langle x,x'\rangle-c}}{e^{a\langle x,x'\rangle+c} + e^{a\langle x,x'\rangle-c}}} \text{ pour } \boldsymbol{a > 0, c > 0}$$

4 Fonctions Distances :

- **Distance Euclidienne**

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^{n} (x_{1j} - x_{2j})^2}$$

- **Distance chi-deux**

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^{n} \frac{1}{f_n} (f_{1j} - f_{2j})^2}$$

avec f_{ij} proportion par individu et f_j proportion moyenne pour la variable j.

- **Distance de Manhattan(ou distance City-Block ou Taxi-Distance)**

$$d(x_1, x_2) = \sum_{j=1}^{n} |x_{1j} - x_{2j}|$$

Par rapport à la distance euclidienne usuelle, qui utilise le carré des écarts, on utilisera notamment cette mesure pour minimiser l'influence des grands écarts.

5 Fonctions Sigmoïde(utilisées dans Régression Logistique,...)

- **Courbe de Gompertz :**

$$y(t) = a \exp(-be^{-ct})$$

La courbe de Gompertz est utilisée pour modéliser certaines séries temporelles, dont la croissance est lente au départ et s'arrête à un moment. Cette fonction est employée pour décrire une population dans un espace confiné, dont la reproduction augmente au début puis ralenti une fois que les ressources à disposition deviennent rares 1.

- **tangente hyperbolique :**

$$tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$$

Bien que la tangente hyperbolique soit à la base une fonction complexe, on s'intéresse ici à sa restriction à , qui est une bijection strictement croissante de dans [-1,1].

- **Fonction de répartition de la loi normale centrée réduite :**

$$\phi(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-\frac{1}{2}x^2} dx$$

Les fonctions de répartition de la loi de normale centrée réduite sont de magnifiques candidates pour la fonction hypothèse que l'on recherche. Dans les faits, on préférera la fonction de répartition de la loi logistique.

- **Fonction de répartition de la loi logistique :**

$$g(t, \mu, s) = \frac{1}{1 + e^{-\frac{t-\mu}{s}}}$$

La loi logistique ressemble beaucoup à une loi normale, mais elle présente un plus grand kurtosis (indicateur qui mesure le coefficient d'applatisssement d'une distribution).

Cas particulier : Fonction logistique
Un cas particulier de la fonction de répartition de la loi logistique est celle de paramètre (0,1). On l'appelle fonction logistique et elle se définit comme suit :

$$\boldsymbol{g(t) = \frac{1}{1 + e^{-t}}}$$

On nommera fonction sigmoïde cette fonction logistique. Elle est effectivement, dans la littérature et la pratique, la fonction sigmoïde la plus utilisée.

6 Régression

Soient un ensemble de $\boldsymbol{m}$ variables $\boldsymbol{X_1, X_2, ..., X_m}$ pour lesquelles on possède $\boldsymbol{n}$ observations, le tout représenté dans la matrice $\boldsymbol{X}$. On souhaite à partir des variables $\boldsymbol{X_1, X_2, ..., X_m}$ et leurs observations prédire la variable $\boldsymbol{Y}$. On cherche donc une fonction de la forme :

$$\boldsymbol{\hat{Y} = f(X) = wX + b = w_1X_1 + w_2X_2 + ... + w_mX_m + b}$$

tel que $\boldsymbol{\hat{Y}}$ qui est le vecteur des valeurs prédites se rapproche le plus possible du vecteur des valeurs cibles $\boldsymbol{Y}$

- $\boldsymbol{w}$ représente le vecteur des poids
- **le biais** est défini par $\boldsymbol{Y - wX - b}$.
- $\boldsymbol{b}$ est l'ordonnée à l'origine appelé aussi **intercept**.
- l'erreur est donnée par : $\boldsymbol{Y - \hat{Y} = Y - f(X) = Y - wX - b}$.
  On utilise jamais l'erreur tel quelle, on préfère considérait la somme du carrés erreurs :

$$\sum_{i=1}^{n} (f(X^i) - Y^i)^2$$

7 Classification

7.0.1 Justesse

$$\boldsymbol{Justesse} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}}$$

7.0.2 Précision, Rappel

$$\boldsymbol{Precision = Precision = \frac{VP}{VP + FP}}$$

où **VP = Vrais Positifs** et **FP = Faux Positifs**

$$\boldsymbol{Rappel = Recall = \frac{VP}{VP + FN}}$$

où **FN = Faux Négatifs**

7.0.3 Courbe ROC(Receiver Operating Characteristic)

$$\textbf{Taux de vrais positifs} = \frac{\boldsymbol{VP}}{\boldsymbol{VP + FN}} = \boldsymbol{Recall}$$

$$\textbf{Taux de faux positifs} = \frac{\boldsymbol{FP}}{\boldsymbol{FP + VN}}$$

$$\textbf{Courbe ROC : \{Taux de VP=Recall} = \boldsymbol{func}\textbf{(Taux de FP)\}}$$

7.0.4 Biais de Prédiction

biais de prédiction = valeur moyenne des prédictions - valeur moyenne des étiquettes dans l'ensemble des données

Un biais de prédiction très différent de zéro vous informe que votre modèle ne fonctionne pas correctement, puisqu'il n'identifie pas la fréquence réelle des étiquettes positives.

Admettons, par exemple, que nous savons que 1 % de tous les e-mails sont du spam. Si nous ignorons tout d'un e-mail donné, nous devrions prédire que la probabilité pour qu'il s'agisse de spam est de 1 %. Selon cette logique, un bon modèle de détection de spam doit prédire en moyenne que les e-mails ont 1 % de chances d'être du spam. En d'autres termes, si nous calculons la moyenne des prédictions de probabilité pour que chaque e-mail soit du spam, le résultat devrait être 1 %. Mais si la prédiction moyenne du modèle évalue la probabilité du spam à 20 %, nous pouvons en conclure que le modèle est affecté d'un biais de prédiction.

Plusieurs causes peuvent être à l'origine d'un biais de prédiction :

- Un ensemble de caractéristiques incomplet
- La présence de bruit dans l'ensemble de données
- Un pipeline ne fonctionnant pas correctement
- Un échantillon d'apprentissage non représentatif
- Une régularisation excessive

Vous pourriez être tenté de corriger un **biais de prédiction** en ajustant le modèle après apprentissage, c'est-à-dire en ajoutant un niveau de calibration afin d'adapter les résultats de votre modèle et de diminuer le biais de prédiction.
Par exemple, si votre modèle a un biais de prédiction de +3 %, vous pouvez ajouter un nouveau de calibration réduisant la prédiction moyenne de 3 %.
Toutefois, l'ajout d'un niveau de calibration est déconseillé pour les raisons suivantes :

- Vous intervenez sur le symptôme et non sur la cause du biais.
- Votre système devient moins flexible et requiert désormais d'être tenu à jour.

Dans la mesure du possible, évitez les niveaux de calibration. Les projets qui ont recours aux niveaux de calibration ont tendance à en devenir dépendants, car ils les utilisent pour résoudre tous les défauts de leur modèle. La gestion de tous ces niveaux de calibration peut finir par tourner au cauchemar.

Remarque : Un bon modèle aura généralement un **biais de prédiction** voisin de zéro. Cela étant, un **biais de prédiction** très bas ne suffit pas à garantir la qualité de votre modèle. Un très mauvais modèle peut avoir un **biais de prédiction** négligeable. Par exemple, un modèle prédisant simplement la valeur moyenne pour tous les exemples serait un mauvais modèle, malgré un **biais de prédiction** égal à zéro.

7.0.5 Régression logistique

- **Fonction de coût global** pour n individus et m variables :

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{n} j(h(x_i, y_i))$$

où

$$j(h(x), y) = \begin{cases} -log(h(x)), & y = 1 \\ -log(1 - h(x)), & y = 0 \end{cases}$$

8 Réseaux de Neurones

8.1 Fonction d'activation

La valeur de chaque nœud de la couche cachée 1 est transformée par une fonction non linéaire avant d'être transférée aux sommes pondérées de la couche suivante et ainsi de suite. Cette fonction non linéaire est appelée ***fonction d'activation***.
Utilisé dans les réseaux de neuronnes, elle **permet de crée des modèles non linéaire**.

- **ReLu(Rectification Linear Unit)Unité de Rectification Linéaire**

$$\boldsymbol{f(x) = max(0, x)}$$

ReLu est souvent utilisé comme fonction d'activation car c'est une fonction simple qui permet de résoudre un large nombre de problèmes.

La fonction d'activation d'unité de rectification linéaire (ou ReLU) est souvent un peu plus efficace qu'une fonction lisse de type sigmoïde, tout en étant bien plus simple à calculer.
La supériorité de la fonction ReLU repose sur des conclusions empiriques, sans doute du fait que la fonction ReLU présente une plage de réponse plus utile. La réponse de la fonction sigmoïde est rapidement défaillante de chaque côté.

- **Fonctions Sigmoïdes**

On peut aussi utiliser pour la fonction d'activation des fonctions sigmoïdes :

⋆ **Fonction Tanh** :

$$\boldsymbol{Tanh(wX + b) = \frac{e^{wX+b} - e^{-wX-b}}{e^{wX+b} + e^{-wX-b}}}$$

⋆ **Fonction logistique**

$$\boldsymbol{g(t) = \frac{1}{1 + e^{-t}}}$$

Elle est effectivement, dans la littérature et la pratique, la fonction sigmoïde la plus utilisée.

- **Fonctions en générales**

  En fait, toute fonction mathématique peut être utilisée comme fonction d'activation. Supposons que $\boldsymbol{\sigma}$ représente notre fonction d'activation (ReLU, sigmoïde ou autre). La valeur d'un nœud dans le réseau est alors donnée par la formule suivante :

$$\boldsymbol{\sigma(wX + b)}$$

8.2 Backpropagation(Rétropropagation)

Algorithme principal utilisé pour exécuter la descente de gradient sur des réseaux de neurones. Les valeurs de sortie de chaque nœud sont d'abord calculées (et mises en cache) dans une propagation avant, puis la dérivée partielle de l'erreur pour chaque paramètre est calculée dans une rétropropagation via le graphe.

8.2.1 Problèmes possibles

La rétropropagation peut être à l'origine de plusieurs problèmes courants.

⋆ ***Disparition des gradients***
Les gradients des couches inférieures (qui sont les plus proches de l'entrée) peuvent devenir extrêmement petits. Dans les réseaux profonds, le calcul de ces gradients peut impliquer le produit de nombreux petits termes.
Lorsque les gradients se rapprochent de 0 pour les couches inférieures, ces dernières sont entraînées très lentement, voire pas du tout.
La fonction d'activation des unités ReLU permet d'empêcher la disparition des gradients.

⋆ ***Explosion des gradients***
Si les pondérations d'un réseau sont très importantes, les gradients des couches inférieures impliquent le produit de nombreux termes de grande taille. Dans ce cas, les gradients peuvent exploser. Autrement dit, ils sont trop grands pour que la convergence fonctionne.
La normalisation de lot, tout comme la réduction du taux d'apprentissage, peut empêcher l'explosion des gradients.

⋆ ***Unités ReLU inactives***
Lorsque la somme pondérée d'une unité ReLU descend en dessous de 0, l'unité peut se figer. Elle génère alors 0 activation et ne contribue donc pas à la sortie du réseau, tandis que les gradients ne peuvent plus y passer lors de la rétropropagation.
En cas d'élimination d'une source de gradients, il se peut même que l'entrée effectuée dans l'unité ReLU ne puisse plus jamais changer suffisamment pour que la somme pondérée repasse au-dessus de 0.
La réduction du taux d'apprentissage peut empêcher les unités ReLU de devenir inactives.

8.2.2 Régularisation par abandon

Une autre forme de régularisation, appelée abandon, est utile pour les réseaux de neurones. Cette méthode "abandonne" de manière aléatoire des activations d'unités dans un réseau pour un pas de gradient unique. Plus il y a d'abandons, plus la régularisation est poussée :

- ⋆ 0.0 = pas de régularisation par abandon.
- ⋆ 1.0 = abandon total (le modèle n'apprend rien).
- ⋆ Les valeurs comprises entre 0.0 et 1.0 sont plus efficaces.

8.2.3 régularisation par abandon (dropout regularization)

Forme de **régularisation** utile dans l'apprentissage des **réseaux de neurones**. La régularisation par abandon consiste à supprimer de manière aléatoire un nombre fixe d'unités dans une couche du réseau pour un pas de gradient unique. Plus le nombre d'unités abandonnées est élevé, plus la régularisation est solide. Cette méthode est analogue à l'entraînement du modèle pour émuler un groupe exponentiellement large de réseaux plus petits.

8.3 Réseaux de neurones à classes multiples : un contre tous

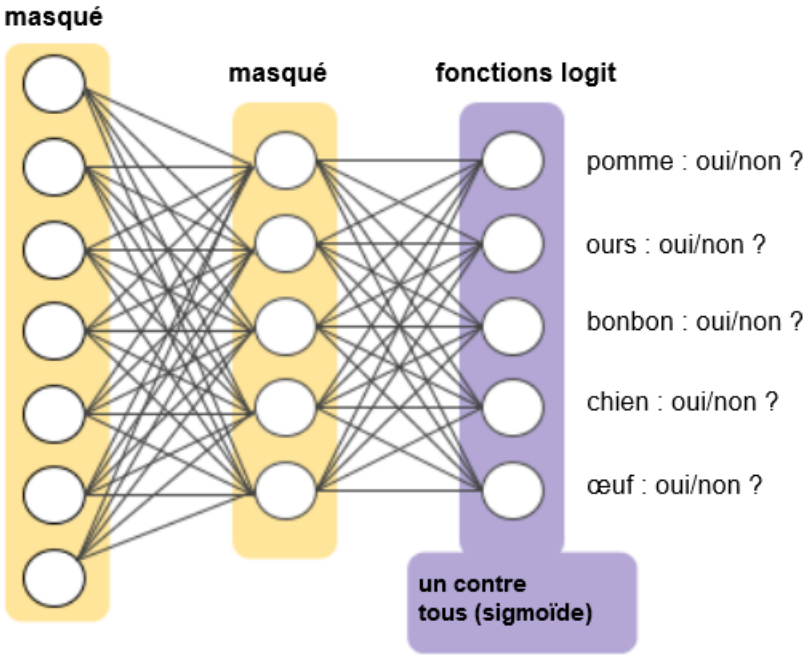
Un contre tous permet d'utiliser la classification binaire. Étant donné un problème de classification avec **N solutions possibles**, une solution un contre tous consiste en **N classifieurs binaires distincts** : un classifieur binaire pour chaque résultat possible.
Au cours de l'apprentissage, le modèle parcourt une séquence de classifieurs binaires, formé chacun pour répondre à une question de classification distincte.

Par exemple, prenons une image représentant un chien. Cinq reconnaissances différentes pourront être formées, dont quatre verront l'image comme un exemple négatif (pas un chien) et une verra l'image comme un exemple positif (un chien). Par exemple :

1. Cette image représente-t-elle une pomme ? Non.
2. Cette image représente-t-elle un ours ? Non.
3. Cette image représente-t-elle des friandises ? Non.
4. Cette image représente-t-elle un chien ? Oui.

Cette approche est relativement raisonnable lorsque nombre total de classes est réduit, mais devient de plus en plus inefficace à mesure que le nombre de classes augmente.

Nous pouvons créer un modèle un contre tous considérablement plus efficace avec un réseau de neurones profond, dans lequel chaque nœud de résultat représente une classe différente. La figure suivante suggère cette approche :



8.4 Réseaux de neurones à classes multiples : Softmax

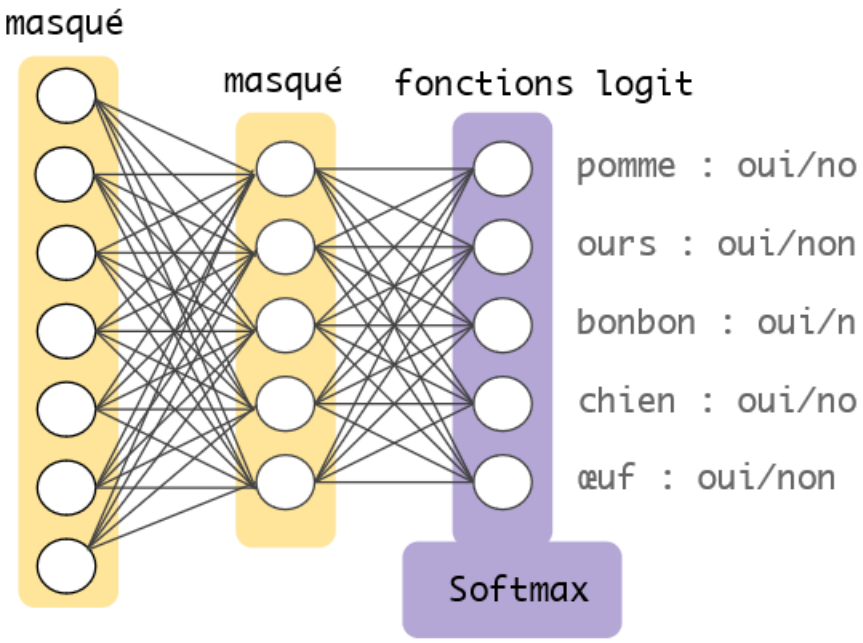
Rappelez-vous que la régression logistique produit une décimale entre 0 et 1. Par exemple, un résultat de régression logistique de 0,8 pour un classificateur d'e-mails suggère que les chances qu'un e-mail soit indésirable sont de 80 %, et les chances qu'il ne soit pas indésirable de 20 %. De façon évidente, la somme des probabilités qu'un e-mail soit indésirable ou non est égale à 1.

Softmax étend cette idée à un monde à plusieurs classes. C'est-à-dire que **Softmax** attribue des probabilités décimales à chaque classe d'un problème à plusieurs classes. La somme de ces probabilités décimales doit être égale à 1. Cette contrainte supplémentaire permet de faire converger l'apprentissage plus rapidement qu'il ne le ferait autrement.

Par exemple, revenons à l'analyse de l'image de la Figure 1. Softmax pourra produire les probabilités suivantes qu'une image appartienne à une classe spécifique :

Classe	Probabilité
pomme	0,001
ours	0,04
friandises	0,008
chien	0,95
oeuf	0,001

Softmax est mis en œuvre via une couche de réseau de neurones juste avant la couche du résultat. La couche Softmax doit comporter le même nombre de nœuds que la couche du résultat.



L'équation **Softmax** est la suivante :

$$\mathbb{P}(y = j|x) = \frac{\exp(w_j^T x + b_j)}{\sum_{k \in K} \exp(w_k^T x + b_k)}$$

Notons que cette formule développe la formule de la régression logistique sur plusieurs classes.

8.4.1 Options de Softmax

Prenons les variantes de Softmax suivantes :

◊ **Softmax complet** est le **Softmax** dont nous avons parlé, c'est-à-dire le **Softmax** qui calcule une probabilité pour chaque classe possible.

◊ **L'échantillonnage de candidats** signifie que Softmax calcule une probabilité pour toutes les étiquettes positives, mais seulement pour un échantillon aléatoire d'étiquettes négatives. Par exemple, si nous souhaitons déterminer si une image d'entrée est un beagle ou un limier, il est inutile de fournir des probabilités pour chaque exemple "non chien".

Softmax complet est relativement économique lorsque le nombre de classes est petit, mais devient extrêmement coûteux lorsque le nombre de classes augmente. L'échantillonnage de candidats peut améliorer l'efficacité des problèmes qui comportent un grand nombre de classes.

8.4.2 Une étiquette ou plusieurs étiquettes

Softmax suppose que chaque exemple appartient exactement à une classe. Cependant, certains exemples peuvent appartenir simultanément à plusieurs classes.
Pour ces exemples :

- Vous ne pouvez pas utiliser Softmax.
- Vous devez vous appuyer sur les régressions logistiques.

Par exemple, supposons que vos exemples sont des images contenant exactement un élément : un fruit. **Softmax** peut déterminer la probabilité que cet élément soit une poire, une orange, une pomme, etc. Si vos exemples sont des images contenant toute sorte de choses (des saladiers contenant plusieurs types de fruits), alors vous devrez utiliser plusieurs régressions logistiques.

9 Mesures de performances

9.1 Régression

- **RMSE(Root Mean Square Error)**

$$\boldsymbol{RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}}$$

- $\boldsymbol{m}$ est la taille de l'échantillon pour lequel on mesure la RMSE.

  Par exemple,pour le jeu de donnée California Housing(Prédiction du prix des maisons en Californie par district), si on évalue la RMSE pour un échantillon de validation avec 2 000 districts donc 2 000 observations, alors $\boldsymbol{m = 2000}$.
- $\boldsymbol{x^{(i)}}$ est un vecteur de toutes les valeurs des features(variables) pour l'observation **ième** dans le jeu de donnée. $\boldsymbol{y^{(i)}}$ est le label de l'observation **ième**(c'est la valeur qu'on souhaite prédire).

  Par exemple, si le premier district dans le jeu de donnée localisé à -118.29° en longitude et 33.91° en latitude et qui compte 1 146 habitants, un revenu median de $38 372 et un prix médian des maisons à $156 400 alors on a :

$$\boldsymbol{x^{(1)}} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1.416 \\ 38.372 \end{pmatrix}$$

  et

$$\boldsymbol{y^{(1)}} = 156,400$$

- **X** est la matrice qui représente le jeu de données entier en excluant la variable à prédire y
- $\boldsymbol{h}$ est la fonction du modèle utilisé pour la prédiction, aussi appelé **hypothesis**. Quand on fournit au modèle l'observation $\boldsymbol{x^{(i)}}$, il prédit la valeur

$$\boldsymbol{\hat{y}^{(i)}} = h(x^{(i)})$$

et la vraie valeur est $\boldsymbol{y^i}$, l'error de prédiction est donc

$$\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^i$$

- RMSE($\mathbf{X}$,h) est la fonction de coût liée à la fonction $\boldsymbol{h}$ du modèle.

Une mesure de performance typique pour les cas de problèmes de régression est la Root Mean Square Error (RMSE). Elle mesure la déviation standard de l'erreur que le modèle commet lors de sa prédiction. Par exemple, pour le jeu de donnée California Housing(Prédiction du prix des maisons en Californie), la RMSE vaut 50 000 ce qui veut dire qu'environ 68% des prédictions du modèle auront un écart \$50,000 avec la valeur actuelle de la maison, et environ 95% des prédictions auront un écart \$100,000 avec la valeur actuelle.

- **MAE(Mean Absolute Error)**

$$\boldsymbol{MAE(X,h)} = \frac{\mathbf{1}}{\boldsymbol{m}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} |\boldsymbol{h}(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)}|$$

RMSE et **MAE** sont tous les deux des façons de mesurerla distance entre deux vecteurs : le vecteur de prediction et le vecteur des vraies valeurs.

Plus on utilise des normes grandes, plus ces mesures de distance se concentre sur les grandes valeurs et négligent les petites valeurs.
C'est ainsi que **RMSE (norme L_2)** est plus sensible aux valeurs extrêmes (outliers) que **MAE (norme L_1)**. Par contre lorsque ces valeurs extrêmes(extrêma) sont exponentiellement rares(comme dans le cas de courbe en forme de cloche), la RMSE est une excellente mesure et est généralement préférée.

9.2 Classification

- **F_β score** :

$$\boldsymbol{F_\beta} = \mathbf{(1 + \beta^2)} \times \frac{\boldsymbol{Precision} \times \boldsymbol{Rappel}}{\boldsymbol{Precision} + \boldsymbol{Rappel}}$$

En général, on utilise le **F_1 score** :

$$\boldsymbol{F_1} = \mathbf{2} \times \frac{\boldsymbol{Precision} \times \boldsymbol{Rappel}}{\boldsymbol{Precision} + \boldsymbol{Rappel}}$$

- **AUC(Area Under the ROC Curve)**

Cette valeur mesure l'intégralité de l'aire à deux dimensions située sous l'ensemble de la courbe ROC (par calculs d'intégrales) de (0,0) à (1,1).
L'AUC fournit une mesure agrégée des performances pour tous les seuils de classification possibles. On peut interpréter l'AUC comme une mesure de la probabilité pour que le

modèle classe un exemple positif aléatoire au-dessus d'un exemple négatif aléatoire. Les valeurs d'AUC sont comprises dans une plage de 0 à 1. Un modèle dont 100 % des prédictions sont erronées a un AUC de 0,0. Si toutes ses prédictions sont correctes, son AUC est de 1,0.

L'AUC présente les avantages suivants :

- L'AUC est **invariante d'échelle**. Elle mesure la qualité du classement des prédictions, plutôt que leurs valeurs absolues.
- L'AUC est **indépendante des seuils de classification**. Elle mesure la qualité des précisions du modèle quel que soit le seuil de classification sélectionné.

L'AUC présente les limites suivantes :

- **L'invariance d'échelle n'est pas toujours souhaitable**. Par exemple, nous avons parfois besoin d'obtenir des probabilités précisément calibrées, ce que l'AUC ne permet pas de déterminer.
- **L'indépendance vis-à-vis des seuils de classification n'est pas toujours souhaitable**. Lorsque des disparités importantes de coût existent entre les faux négatifs et les faux positifs, il peut être essentiel de minimiser l'un des types d'erreur de classification. Par exemple, dans un contexte de détection de spam il sera probablement préférable de minimiser en priorité les faux positifs (même si cela entraîne une augmentation significative des faux négatifs). L'AUC n'est pas un critère à retenir pour ce type d'optimisation.

10 Algorithme de résolution de problème min et max de la fonction de coût et la pénalisation

- **GD(Gradient Descent)**

  (à compléter)

- **SGD(Stochastic Gradient Descent)**

  (à compléter)

- **FTRL(Follow The Regularized Leader)**

  Algorithme basé sur l'algorithme de **Gradient Descent**, les modèles linéaires de grande dimension tirent parti de l'utilisation d'une variante de l'optimisation basée sur le gradient, appelée FTRL.
  L'avantage de cet algorithme est de permettre une mise à l'échelle du taux d'apprentissage qui diffère en fonction du coefficient, ce qui peut se révéler particulièrement utile si certaines caractéristiques utilisent rarement des valeurs non nulles (il est aussi parfaitement adapté à la régularisation L1).