Mini projet 1 : You've got mail !

Résumé

Dans ce projet, nous allons étudier plusieurs problématiques d'apprentissage statistique (Machine Learning) en prenant comme application le traitement des emails. Bien qu'une base de données soit fournie, il est préférable d'utiliser vos propres emails, ce sera intéressant. Dans la plupart des plateformes d'emails, il existe une fonction d'export qui permet de récupérer vos emails le plus souvent en sélectionnant les sous-dossiers adéquats. Selon les plateformes, le rendu est soit un fichier texte regroupant l'ensemble des emails, soit un répertoire contenant un fichier par email. Utilisez l'une ou l'autre des méthodes, les emails ainsi exportés sont en format MIME : `https://fr.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions`. Il est conseillé de lire (en diagonal) cette page afin de comprendre comment extraire les informations importantes des emails (au moins le sujet, l'expéditeur, le destinataire et le corps de l'email). Par ailleurs, il existe un paquet python, `email`, qui permet de parser de manière simple les emails. Il n'est pas du tout obligatoire de l'utiliser, vous pouvez très facilement réaliser les opérations nécessaires avec la fonction `split()` et quelques expressions régulières (paquet `re`).

Le projet est à faire en binôme. Le rendu consistera en un petit rapport répondant aux questions, présentant les expériences que vous avez menées avec une petite analyse des résultats ainsi qu'une archive de votre code (aucun code ne doit figurer dans le rapport). Il est à envoyer par e-mail pour la semaine du 26 février avant votre TME à votre chargé de TME. Vous trouverez sur le site de l'ue un petit guide pour la rédaction de votre rapport.

Attention : le sujet du projet sera complété au fur et à mesure, n'oubliez pas de le mettre à jour.

Exercice 1 – Fonctions préliminaires

Q 1.1 Importer à partir de votre messagerie une liste d'emails non spams (*ham* en argot geek) et une liste d'emails spams (sélectionner judicieusement les répertoires afin qu'ils ne contiennent pas des emails persos). Alternativement, télécharger l'archive sur le site de l'ue (avec deux fichiers, `spams.txt` et `ham.txt`).

Q 1.2 Utiliser et adapter le script fourni sur le site de l'ue pour obtenir la liste des textes des emails à partir des fichiers d'archive.

Q 1.3 Faire une fonction `split(liste,x)` qui permet de partager uniformément une liste telle que décrit ci-dessus en deux sous-listes, l'une contenant x% des emails de la liste d'entrée et l'autre les emails restants.

1 Classification d'emails et détection de spam

Dans cette première partie du projet, nous allons étudier le problème de la *classification supervisée*, une thématique centrale de l'apprentissage statistique. L'objectif est de construire automatiquement des fonctions capables de prédire la *classe*[1] d'un objet à partir d'un ensemble d'exemples, une collection d'objets et de leur classe. Lorsqu'il n'y a que deux classes à distinguer, le problème est alors nommé *classification binaire*, une classe étant appelée arbitrairement *positive*, l'autre *négative*. Dans le cas de la détection de spam, il s'agit donc

1. ou *label* ou *étiquette*, ces mots sont équivalents.

de construire une fonction qui prend en entrée un email et renvoie si c'est un spam (classe positive) ou non (classe négative).
Formellement, soit $\mathcal{X}$ un ensemble qui permet de décrire un email, il s'agit de construire une fonction $f : \mathcal{X} \to \{-1, 1\}$ (1 pour spam, -1 pour non spam) à partir d'un ensemble $E_{app} = \{(\mathbf{x}^i, y^i) \in \mathcal{X} \times \{-1, 1\}\}$ d'exemples de emails, $\mathbf{x}^i$ étant la description du $i^{ème}$ email et y^i son étiquette. Un tel ensemble est nommé ensemble d'apprentissage (il va servir à *apprendre* la fonction f).
La méthode d'apprentissage que nous allons étudier est le classifieur bayésien naïf, un algorithme simple qui donne d'excellentes performances sur cette tâche (il est une brique fondamentale de la plupart des algorithmes de détections de spam). Les résultats dépendent fortement de comment est représenter un email, c'est-à-dire le choix de l'ensemble $\mathcal{X}$ et sa représentation. Nous allons étudier principalement deux différentes représentations possibles pour un email : la longueur du texte principal et les mots qu'il contient.
Par ailleurs, cet algorithme est générique : plutôt que de prédire spam ou non spam, vous pouvez prédire également d'autres classes/thèmes.

Exercice 2 – Classification à partir de la longueur d'un email

Dans cet exercice, nous considérons comme espace de description des emails $\mathcal{X}$ la longueur du corps du texte d'un email, c'est-à-dire que chaque email sera décrit dans $\mathcal{X} = \mathbb{N}$ comme le nombre de mots qui le compose. Cette description est très sommaire et il n'est pas certain qu'elle soit suffisante pour résoudre le problème, mais elle a le mérite d'être simple et de pouvoir décrire les emails par une seule caractéristique.
Une solution au problème de classification supervisée est de considérer une approche basée sur les probabilités. Soit Y la variable aléatoire dénotant la classe $(-1, 1)$ et X la variable aléatoire dénotant la description d'un email (dans cet exercice $X \in \mathbb{N}$), soit $P(Y = +1|X = \mathbf{x})$ la probabilité que l'email soit un spam sachant sa description $\mathbf{x}$, par l'application de Bayes nous pouvons calculer que :

$$P(Y = +1|X = \mathbf{x}) = \frac{P(X = \mathbf{x}|Y = +1)P(Y = +1)}{P(X = \mathbf{x})}$$

. Le *classifieur* (la fonction qui classifie) correspondant est donc la fonction suivante qui attribue $+1$ à $\mathbf{x}$ si $P(y = +1|\mathbf{x}) > P(y = -1|\mathbf{x})$, -1 sinon, soit $argmax_y P(Y = y|X = \mathbf{x})$.

Q 2.1 Expliquer succinctement chaque terme de l'équation ci-dessus *en français*. Est-il important de calculer $P(X = \mathbf{x})$? $P(Y = +1)$? Comment estimer la distribution $P(X = \mathbf{x}|Y = +1)$? Est-il important d'estimer $P(X = \mathbf{x}|Y = -1)$?

Q 2.2 Faire une fonction qui permet de calculer pour un email la longueur du texte principal. Faire un histogramme des longueurs des emails. Au vu des résultats, semble-t-il raisonnable de considérer tous les cas possibles ? Que proposer comme regroupement ?

Q 2.3 Faire une fonction `apprend_modele(spam,nonspam)` qui à partir d'un ensemble d'emails spams et non spams permet de renvoyer la distribution $P(X = \mathbf{x}|Y = +1)$. Faire une fonction `predit_email(emails,modele)` qui renvoie le label prédit pour l'ensemble d'emails et le modèle passés en paramètres [2].

Q 2.4 Donner une estimation de la probabilité de l'erreur $P(f(\mathbf{x}) \neq y)$, des erreurs $P(f(\mathbf{x}) = +1|y = -1)$ et $P(f(\mathbf{x}) = -1|y = +1)$. Quelles sont leurs significations ? Ces estimations sont-elles fiables ? En fait, l'estimation de l'erreur ne se fait jamais sur l'ensemble d'apprentissage, mais sur un autre ensemble, appelé ensemble de test, qui ne contient pas les éléments de l'ensemble d'apprentissage. Une façon simple de constituer un tel ensemble est de diviser l'ensemble d'exemples disponibles en deux, l'un servant pour l'apprentissage et l'autre

2. Les signatures de fonction ne sont qu'indicatives, vous pouvez les adapter selon vos besoins.

pour le test. Calculer les différentes erreurs en faisant varier le pourcentage d'exemples servant pour l'apprentissage et le test, commenter les résultats.

Exercice 3 – Classification à partir du contenu d'un email

La méthode précédente est trop grossière pour pouvoir donner des résultats satisfaisants, vu qu'elle ne prend pas en compte la sémantique des emails. Dans cet exercice, nous allons décrire un email en fonction des mots qui le compose : chaque email sera représenté par un vecteur binaire $(x_1, x_2, \ldots, x_d) \in \{0,1\}^d$, où chaque dimension i représente un mot et la valeur x_i indique la présence ou non du mot dans l'email - la signification de chaque dimension reste bien sûr la même pour tous les emails. L'association de chaque dimension à un mot est appelé un dictionnaire.

Q 3.1 Est-il raisonnable d'essayer de calculer la distribution $P(X = \mathbf{x}|Y = +1)$ cette fois ? Si on considère un dictionnaire de 1000 mots uniquement, combien faudrait-il calculer de paramètres pour caractériser la distribution ?
Le classifieur bayésien naïf fait l'hypothèse d'indépendance entre les dimensions, soit $P(X = \mathbf{x}|Y = +1) = \prod_{i=1}^{d} P(X_i = x_i|Y = +1)$. Cette hypothèse est-elle vérifiée en général ? Combien de paramètres sont nécessaires dans ce cas ?

Q 3.2 Faire une fonction qui permet de compter pour chaque mot apparaissant dans une collection d'emails le nombre d'emails dans lesquels il apparait. Sur un sous-ensemble de votre collection d'emails, tracer l'histogramme du nombre d'apparitions. Vous semble-t-il judicieux d'utiliser tous les mots apparaissant dans les emails ? Vaut-il mieux se concentrer sur les mots les plus courants ? Les moins courants ? D'autres mots ? Proposer des heuristiques simples pour réduire le nombre de mots (penser aux accents par exemple, à la ponctuation, aux majuscules . . .).

Q 3.3 Programmer un classifieur bayésien naïf et la fonction de prédiction associée sur un dictionnaire réduit que vous proposerez. Tracer les erreurs de l'exercice précédent en fonction de la taille de votre dictionnaire. Analyser les résultats.
Conseil : vous serez amené à calculer des probabilités très petites. Une astuce classique est de passer au logarithme de la probabilité à calculer, afin d'utiliser des additions et non des multiplications. Pourquoi est-ce possible ? Implémenter vos fonctions avec cette astuce.

2 Visualisation

Nous souhaitons maintenant *visualiser* la répartition de nos emails, représentés comme des vecteurs $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ (on pourra utiliser la même représentation vectorielle qu'à l'exercice précédent) dans un espace de grande dimension (la taille d du vocabulaire). Ceci peut nous permettre d'identifier des *regroupements* cohérants d'emails qui parlent d'un sujet similaire, ou d'attester la qualité de notre représentation d'emails pour discriminer un spam d'un email non-spam, par exemple.
Comme il nous est difficile de visualiser des vecteurs dans un espace de plus de trois dimensions, il est nécéssaire de trouver une nouvelle représentation $\mathbf{y}$ pour nos vecteurs en plus faible dimension (en 2D ou en 3D), qui préserve la distance relative des différents vecteurs $\mathbf{x}$ en grande dimension autant que possible. Plus précisemment, si deux représentations d'emails $\mathbf{x}^i$ et $\mathbf{x}^j$ sont proches dans l'espace par rapport à une distance $d(\mathbf{x}^i, \mathbf{x}^j)$ (la distance euclidienne par exemple), nous souhaitons que la distance $d(\mathbf{y}^i, \mathbf{y}^j)$ soit faible également, et inversement. Ceci est le but d'un algorithme de visualisation en grande dimension : trouver une fonction adéquate qui *projette* la représentation $\mathbf{x} \in \mathbb{R}^d$ de l'email dans l'espace $\mathbb{R}^2$, par exemple.

On se propose d'utiliser l'algorithme *Stochastic Neighbor Embedding* (SNE)[3]. L'algorithme est décrit en détail dans l'article :
`https://papers.nips.cc/paper/2276-stochastic-neighbor-embedding.pdf`. L'algorithme SNE fonctionne de la manière suivante. Nous modélisons d'abord la probabilité que la représentation $\mathbf{x}^j$ soit dans le voisinage de $\mathbf{x}^i$ à l'aide de $P^i(X = \mathbf{x}_j)$, appelée aussi P^i_j :

$$P^i(X = \mathbf{x}^j) = \frac{exp^{-d(\mathbf{x}^i,\mathbf{x}^j)/2\sigma_i}}{\sum_{k\neq i} exp^{-d(\mathbf{x}^k,\mathbf{x}^i)/2\sigma_i}}$$

Intuitivement, plus la distance $d(\mathbf{x}^i, \mathbf{x}^j)$ est grande, plus $P^i(X = \mathbf{x}^j)$ sera faible, et inversement. Nous modélisons ensuite la probabilité que la représentation $\mathbf{y}^j$ dans l'espace de faible dimension soit dans le voisinage de $\mathbf{y}^i$ à l'aide de $Q^i(Y = \mathbf{y}^j)$, appelée aussi Q^i_j :

$$Q^i(Y = \mathbf{y}^j) = \frac{exp^{-d(\mathbf{y}^i,\mathbf{y}^j)}}{\sum_{k\neq i} exp^{-d(\mathbf{y}^k,\mathbf{y}^i)}}$$

Afin de conserver le même voisinage pour les deux représentations $\mathbf{x}_i$ et $\mathbf{y}_i$, nous voulons que la distribution Q^i soit la plus proche possible de P^i : nous souhaitons donc minimiser une *distance* entre les deux distributions de probabilités. Nous utiliserons la divergence de Kullback-Leibler (KL), qui calcule la *distance* entre P^i et Q^i :

$$KL(P^i \| Q^i) = \sum_j P^i_j log(\frac{P^i_j}{Q^i_j})$$

L'objectif devient donc de trouver pour chaque email i sa représentation $\mathbf{y}^i = (y^i_1, y^i_2)$ qui minimise la somme des distances entre P^i et Q^i : $C = \sum_j \sum_{i\neq j} KL(P^i \| Q^j)$. On utilise pour cela l'algorithme de descente de gradient, qui permet de trouver une solution itérative à un problème de minimisation en utilisant les dérivées partielles de la fonction à optimiser par rapport aux paramètres celle-ci. Ci-dessous, les dérivées partielles de C par rapport aux coordonnées (y^i_1, y^i_2) du vecteur $\mathbf{y}^i$, pour la première composante :

$$\frac{\partial C}{\partial y^i_1} = 2\sum_j (y^i_1 - y^j_1)(P^i_j - Q^i_j + P^j_i - Q^j_i)$$

et pour la seconde :

$$\frac{\partial C}{\partial y^i_2} = 2\sum_j (y^i_2 - y^j_2)(P^i_j - Q^i_j + P^j_i - Q^j_i)$$

Algorithm 1: Pseudo code de l'algorithme SNE

1. Pour chaque email i, calculer les probabilités P^i_j.
2. Initialiser les $\mathbf{y}^i$ aléatoirement pour chaque email i, selon la loi $\mathcal{N}(0, \frac{1}{2})$ pour chaque dimension
   ▷ On pourra utiliser `numpy.random.normal(0, 0.5)`, de la libraire `numpy`
3. Sélectionner aléatoirement un email i parmi l'ensemble des emails
4. Mettre à jour $\mathbf{y}^i$ selon : ▷ ϵ est choisie à la main, essayer entre 10^{-1} et 10^{-3}
   $y^i_1 \leftarrow y^i_1 - \epsilon\frac{\partial C}{\partial y^i_1}$
   $y^i_2 \leftarrow y^i_2 - \epsilon\frac{\partial C}{\partial y^i_2}$
5. Retourner à l'étape 3 jusqu'à convergence

3. Une version améliorée de l'algorithme est apparue récemment, t-SNE, qui est devenu une référence dans la visualisation des données.