Probabilités - Statistiques : Rapport

Groupe :

- BIARD David
- PENUCHOT Jules
- PICAN Gaëtan

README.txt

Les exercices sont déclinés dans les sources `ex_*.py` . Le code du classifieur Bayesien naïf est dans `bayesien_naif.py` , il a été encapsulé dans une classe pour pouvoir l'utiliser comme un objet générique.

`MultinomialBN.py` contient le code du Bayesien avec implémentation du lissage.

`covvoisins.py` contient le code permettant de considérer l'observation des pixels par rapport à leurs voisins dans le modèle (exercice 1). Les fonctions sont appelées dans `ex_3_1.py` qui permet d'observer les résultats des différentes implémentations du classifieur.

`observations_ex_1.py` permet d'effectuer les observations mentionnées dans l'exercice 1.

Nous ne sommes parvenus qu'à un début d'implémentation pour l'exercice 2 par manque de temps (ndlr: d'organisation).

Exercice 1

Dans l'exercice 3 on considère les variables comme indépendantes. Si cette hypothèse est vraie alors les covariances devraient être identiques d'une classe à l'autre.

Pour vérifier cette hypothèse, il suffit d'afficher les covariances de plusieurs classes pour vérifier qu'elles soient identiques.

Or, on observe qu'elles ne le sont pas et pourraient nous permettre de mieux classer nos images.

On peut donc se demander également si les activations relatives entre les voisins pourraient nous permettre de nous aider à classer nos images.

Pour ce faire il suffit d'afficher les covariances entre voisins d'une classe à l'autre et observer une fois de plus les résultats.

Une telle observation nous permet de faire une observation ciblée sur les bordures des chiffres, qui dans notre cas n'a permis de gagner qu'environ 0.15% sur le pourcentage d'erreur (voir `ex_3_1.py`).

Cette observation peut paraître logique puisque les voisins sont par principe relativement dépendants les uns des autres dans les zones où les pixels sont activés, ce qui ne nous renseigne pas beaucoup plus sur l'image.

Exercice 2

Ici on repart du classifieur Bayésien Naïf Gaussien appliqué aux images MNIST. Ce classifieur implémenté dans l'exercice 3 considère la réalisation d'un pixel conditionnellement à une classe comme une variable aléatoire gaussienne.

Une première analyse que l'on peut effectuer est de regarder, à partir d'un pixel possédant une forte variance, la répartition des valeurs de ce pixel sur toutes nos images. Si l'on représente la répartition de ces valeurs à l'aide d'un histogramme, on peut facilement se rendre compte qu'une courbe représentative telle qu'une gaussienne n'est en fait pas vraiment appropriée pour modéliser cette répartition. En effet, l'hypothèse d'assimiler cela à une seule gaussienne a tendance à fausser la représentation que nous nous étions faite des données.

Une meilleur approche serait de "combiner ou mélanger" deux gaussiennes. Il faudrait donc considérer que la valeur d'un pixel conditiellement à sa classe soit une variable aléatoire dont la distribution est donnée par le mélange de deux gaussiennes.

Nous avons donc programmé un début d'implémentation de l'algorithme EM détaillé en cours dont le principe est assez simple :

```
- Initialiser des poids au hasard pour les deux gaussiennes (avec somme éga

- Evaluer les paramètres de la première gaussienne
- Evaluer les paramètres de la deuxième gaussienne

// Une fois les deux gaussiennes intialisées il suffit de passer
// à la phase itérative qui consiste à évaluer les poids des gaussiennes
// et ajuster les paramètres des gaussiennes jusqu'à trouver des valeurs
// (à peu près) stables

change := faux

while(!change)
{
  change := false
  for c in classes
  {
    a0, a1 = alpha0[c], alpha1[c]

    alpha0[c] := contrib(sousEnsemble[c], gaussienne0[c])
    alpha1[c] := 1. - alpha0[c]

    //  Si les alpha ont changé
    if(alpha0[c], alpha1[c] != a0, a1) change := true

    g0, g1 = gaussienne0[c], gaussienne1[c]

    gaussienne0[c] = evalParams(sousEnsemble[c], gaussienne0[c])
    gaussienne1[c] = evalParams(sousEnsemble[c], gaussienne1[c])

    //  Si les gaussiennes ont changé
    if(g0, g1 != gaussienne0[c], gaussienne1[c]) change := true
  }
}
```

Exercice 3

Dans cette exercice, le but est d’obtenir un classifieur naïf bayésien gaussien capable de prédire la classe des images fournies en entré. Dans un premier temps on implémente une version basée sur des images dont les composantes sont représentées par des valeurs réelles. Pour la phase de prédiction on utilise la formule de bayèse afin d’obtenir la probabilité pour chaque image d’appartenir aux 10 classes possible.

Estimation : regrouper les $\boldsymbol{x}$ par classe, puis calculer moyenne et variance par composante

Inférence :

$$P(Y = y|\boldsymbol{X} = \boldsymbol{x}) \propto P(Y = y) \prod_{i=1}^{d} \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma_{i,y}^2}} \times e^{-\frac{1}{2}\frac{(x_i - \mu_{i,y})^2}{\sigma_{i,y}^2}}$$

$$\log P(Y = y|\boldsymbol{X} = \boldsymbol{x}) \propto \log P(Y = y) - \frac{1}{2} \sum_{i=1}^{d} \left(\log(2 \cdot \pi \cdot \sigma_{i,y}^2) + \frac{(x_i - \mu_{i,y})^2}{\sigma_{i,y}^2} \right)$$

La probabilité maximum étant choisie. Avec cette méthode on obtient donc un taux de précision d'environ 77 %.

Pour la seconde approche on binarise les images afin que celles-ci soient représentées par des composantes binaires. Les résultats obtenus grâce à cette méthode sont relativement meilleurs avec un taux de précision d'environ 79 %.

Enfin, il est possible d'effecteur une approche légèrement différentes avec un modèle bayesien naïf binomial. Les formules d'estimation et d'inférences sont légèrement différentes :

Estimation :

$$\pi_{i,y} = \frac{\text{nombre de doc dans la classe y contenant le mot i}}{\text{nombre de doc de la classe y}} = \frac{n(i,y)}{n(y)}$$

.

Inférence en log :

$$\begin{aligned} \log(P(Y = y|\boldsymbol{X} = \boldsymbol{x})) &= \log(P(\boldsymbol{X} = \boldsymbol{x}|Y = y)) + \log(P(Y = y)) \\ &= \log(P(Y = y)) + \sum_{i=1}^{d} x_i \log(\pi_{i,y}) + (1 - x_i) \log(1 - \pi_{i,y}) \end{aligned}$$

De plus il est possible d'introduire la notion de lissage ce qui permet de ne pas attribuer une valeur nulle sur les pixels qui n'apparaissent jamais. On peut remarquer qu'en "jouant" avec la constante de lissage "alpha" les résultats différent légèrement. En implémentant la méthodes avec l'outils "sklearn" on obtient au mieux un taux d'erreurs d'environ 16 % pour un alpha égal à 0.01. Plus l'on augmente cette constante, la précision du modèle diminue.

Etonnament, en implémentant ce même procédé à la main, on obtient un taux d’erreurs d’envrion 15,5 % avec une constante de lissage égale à 1e-5, ce qui est légèrement mieux qu’avec sklearn.