TME 8 : Classification multi-classes et hiérarchique Évaluation de l'apprentissage structuré

Travaux sur Machines Encadrés

Exercice 1 Classification multi-classe et évaluation

On rappelle que pour instancier un problème structuré, il est nécessaire de définir la joint feature map $\psi(x, y)$ et la fonction de coût $\Delta(y, y')$. Dans un problème de classification multi-classes, on a :

— $\psi(x, y) = \{0^d ... 0^d, \phi(x), 0^d ... 0^d\}$, où $\phi(x) \in \mathbb{R}^d$ est la représentation vectorelle (*i.e.* BoW) de chaque image.

— $\Delta(y, y')$ est le 0-1 loss : $\Delta(y, y') = \begin{cases} 1 & \text{si } y \neq y' \\ 0 & sinon \end{cases}$.

1. Dans le package `upmc.ri.struct.instantiation`, mettre en place le code d'une classe `MultiClass` qui implémente `IStructInstantiation<double[], String>` pour un problème de classification multi-classes. La sortie étant codée avec des String, on utilisera par exemple une `Map<String,Integer>` pour indicer chaque classe lors de la mise en place de $\psi(x, y)$.
2. Dans le package `upmc.ri.bin`, implémenter une classe `MuticlassClassif` avec un main qui permet d'apprendre un classifieur multi-class à partir des données du TME 7. Les étapes suivantes seront effectuées :
   — Chargement des données en train et en test ⇒ `DataSet<double[], String>`.
   — Instantiation d'un objet de type `MultiClass` (voir ci-dessus), et d'un modèle de prédiction de type `LinearStructModel_Ex` (voir TME 7), dont le type sera fixe à partir de l'instantiation multi-classes précédente.
   — OPT : création d'un évaluateur `Evaluator<double[],String>` auquel on passe les données de train, de test, et le modèle
   — Instantiation d'un objet de type `SGDTrainer<double[], String>` pour apprendre le modèle (TME 7), et appel de la fonction `train`. N.B. : avec les données du TME, l'ordre de grandeur des paramètres est par exemple $\lambda = 10^-6$, $\gamma = 10^-2$ et ~ 100 époques
3. Évaluation : Mesurer les performance de classification sur la base de test.
   — Mettre en place une méthode `void confusionMatrix(List<String> predictions, List<String> gt)` dans la classe `MultiClass` pour calculer une matrice de confusion. Pour la visualiser, vous pouvez utiliser la classe `MatrixVisualization` de la librairie EJML (fournie).
   — Que remarque-t-on à l'issue de l'apprentissage ? Analyser en particulier les résulats de classification : comment se répartissent les erreurs de prédiction ? Visualiser la matrice de confusion et afficher certaines images mal classées.

Exercice 2 Classification hiérarchique et évaluation

On souhaite maintenant utiliser l'information hiérarchique des classes disponible dans l'ontologie d'ImageNet pour modifier le critère d'apprentissage multi-classes. La motivation principale consiste à mettre en place une mesure d'erreur qui prenne en compte la dissimilairté entre les classes de sorties. En effet, dans un système de recherche d'informations, les documents non pertinents retournés n'induisent pas nécessaierement le même déagrément pour l'utilisateur. Par exemple, une image non pertinente pour la requête "wood-frog" sera moins gênante s'il il s'agit d'une image de la classe "tree-frog" que "taxi" ou "harp".

On souhiate donc mettre en place une mesure d'erreur de classification hiérarchique permettant d'avoir une mesure plus sémantique que le 0-1 loss pour évaluer les performances, et de modifier l'algorithme d'apprentissage pour que le modèle appris prenne en compte cette mesure.

1. Dans le package `upmc.ri.struct.instantiation`, mettre en place une classe `MultiClassHier` dérivant de `MultiClass`. La méthode $\psi(x, y)$ ne sera pas surchargée. En revanche, la fonction de coût $\Delta(y, y')$ sera modifiée pour prendre en compte la distance sémantique entre les classe.
   - Munir la classe `MultiClassHier` d'une variable `double[][] distances` pour stocker la distance sémantique entre chaque paire de classes, que la méthode `delta` utilisera directement pour calculer le loss $\Delta(y, y')$. Afin de calculer la matrice de distance, On utilisera pour cela la librairie WS4J fournie, qui calcule similarité sémantique entre deux termes en utlisant la hiérachie word net. En particulier la méthode `calcRelatednessOfWords` de la classe `RelatednessCalculator` (qu'on instanciera par exemple avec `WuPalmer`).
   - Dans le constructeur, on calculera la matrice de distance sémantique, en procédant de la manière suivante :
     - Pour chaque couple de classe différente, *i.e.* $y \neq y'$, calcul de la simialité `similarity(y,y') WuPalmer`
     - Passage à une dissimilarité, *i.e.* $\Delta(y, y') = \begin{cases} 1 - similarity(y, y') & \text{si } y \neq y' \\ 0 & sinon \end{cases}$.
     - Afin d'avoir des mesures de distance grossièrement centrées autour de 1 (pour ne pas trop impacter le paramétrage de l'algorithme d'apprentissage), on normalisera les valeurs de distances pour classes différentes ($\Delta(y, y') \neq 0$) par exemple dans $[0.1; 2]$ (normalisation linéaire).
2. Évaluer les performances du modèle structuré de prédiction ainsi instancié, en utilisant la mesure $\Delta(y, y')$ hiérarchique.
3. Évaluer les performances du classifieur multi-classes appris en utilisant avec $\Delta(y, y') \leftarrow$ 0-1 loss en utilisant la mesure $\Delta(y, y')$ hiérarchique. Que remarque-t-on ? Conclure.
   - N.B. : pour un modèle appris avec un loss Δ donné (*e.g.* 0-1 loss), la classe `Evaluator` permet simplement d'évaluer les performances avec autre loss Δ' (*e.g.* loss hiérarchique) : il suffit de changer de type du modèle (*e.g. MultiClass* $\rightarrow$ *MultiClasshier*), et de relancer la fonction *evaluate()*.