TP5 – Perceptron moyenné – Étiquetage morphosyntaxique

- Implémenter un étiqueteur en parties-du-discours (*PoS tagger*)
- Données *in-domain* : UD 1.3, français
- Données *out-domain* : Tweet parlant de foot

On propose d'implémenter un étiqueteur morphosyntaxique basé sur une fenêtre de mots, c'est-à-dire que pour étiqueter un mot, on s'appuie sur la forme du mot (y compris son suffixe) et celles des n mots précédents et suivants (généralement, $n = 2$). Pour cela, on utilise un perceptron moyenné multiclasses, dont l'input sera la représentation d'une fenêtre de mots et dont l'output sera un tag (comme `"NOUN"`) parmi un ensemble fixé de tags possibles. On classifie donc chaque mot à l'aide de la formule:

$$\hat{y} = \arg\max_{y \in C} \mathbf{w}_y \cdot \mathbf{x}$$

où $\mathbf{w}_y$ est le vecteur de paramètres pour la classe y, à apprendre à l'aide du corpus d'entraînement, $\mathbf{x}$ est une représentation de la fenêtre de mots sous la forme d'un vecteur, et C est l'ensemble des classes possibles.

1. Représentation des vecteurs

Une fenêtre de mots sera représentée par un vecteur de booléens valués par des fonctions de traits du type *le mot précédent est* `"le"` On utilisera les traits suivants (notations: w_i est le mot courant, w_{i-1} le mot qui précède, etc ...).

- Le premier caractère de w_i est en majuscule
- w_i est entièrement en majuscules
- w_{i-2} est le mot X (pour tout X du vocabulaire)
- w_{i-1} est le mot X (pour tout X du vocabulaire)
- w_i est le mot X (pour tout X du vocabulaire)
- w_{i+1} est le mot X (pour tout X du vocabulaire)
- w_{i+2} est le mot X (pour tout X du vocabulaire)
- w_{i-1} a pour suffixe X (pour tout suffixe X possible)[1]
- w_i a pour suffixe X (pour tout suffixe X possible)
- w_{i+1} a pour suffixe X (pour tout suffixe X possible)

1. En supposant qu'il y a N mots dans le vocabulaire et S suffixes possibles, quelle sera la taille du vecteur $\mathbf{x}$?
2. Quel sera le nombre maximum de valeurs non nulles de $\mathbf{x}$?
3. Conclure: quelle structure de données utiliser pour représenter $\mathbf{x}$ de manière compacte tout en permettant de calculer facilement le produit scalaire $\mathbf{w}_y \cdot \mathbf{x}$.

[1] On utilise *suffixe* au sens informatique, par exemple les 3 dernières lettres de la chaîne de caractères.

2. Perceptron simple

1. Implémentez la fonction `get_features` qui renvoie une représentation de $\mathbf{x}$ pour une fenêtre donnée.
2. Implémentez la fonction `classify` qui réalise une prédiction pour $\mathbf{x}$ à l'aide des vecteurs de poids $\mathbf{w}_y$ en utilisant la formule ci-dessus.
3. Implémentez la fonction `evaluate` qui taggue le corpus donné en argument et renvoie l'exactitude des prédicitons.
4. Implémentez la méthode `update` qui réalise une mise à jour des poids du perceptron.

Vous pouvez maintenant tester votre programme. Comme l'entraînement peut être long, on conseille de commencer les tests sur des données de taille réduite (par exemple les 10 premières phrases du corpus d'entraînement). Si tout se passe bien, le perceptron devrait atteindre 100% d'exactitude sur ces quelques phrases en quelques itérations.

Voici un exemple d'entraînement sur la totalité du corpus d'entraînement:

```
Epoch 0 : train acc = 91.55 dev acc = 90.05 updates = 45250
Epoch 1 : train acc = 93.45 dev acc = 90.99 updates = 25781
Epoch 2 : train acc = 95.0 dev acc = 92.03 updates = 20266
Epoch 3 : train acc = 95.36 dev acc = 92.04 updates = 16452
Epoch 4 : train acc = 96.14 dev acc = 92.74 updates = 14023
Epoch 5 : train acc = 96.12 dev acc = 92.47 updates = 12190
Epoch 6 : train acc = 96.25 dev acc = 92.44 updates = 10770
Epoch 7 : train acc = 96.84 dev acc = 92.94 updates = 9760
Epoch 8 : train acc = 96.94 dev acc = 92.89 updates = 8989
Epoch 9 : train acc = 97.14 dev acc = 92.9 updates = 8088
Epoch 10 : train acc = 97.69 dev acc = 93.46 updates = 7522
Epoch 11 : train acc = 97.05 dev acc = 92.74 updates = 7122
Epoch 12 : train acc = 97.46 dev acc = 93.2 updates = 6554
Epoch 13 : train acc = 97.68 dev acc = 93.05 updates = 6255
Epoch 14 : train acc = 97.68 dev acc = 93.41 updates = 5961
Epoch 15 : train acc = 97.73 dev acc = 93.41 updates = 5745
Epoch 16 : train acc = 97.71 dev acc = 93.4 updates = 5308
Epoch 17 : train acc = 98.09 dev acc = 93.37 updates = 5111
Epoch 18 : train acc = 97.92 dev acc = 93.16 updates = 4850
Epoch 19 : train acc = 98.02 dev acc = 93.06 updates = 4778
```

- Observez comment évoluent l'exactitude sur le corpus d'entraînement, celle sur le corpus de développement, et le nombre de mises à jour des poids lors de chaque itération sur le corpus. Comment choisir le nombre d'itération optimal ?

3. Perceptron moyenné.

Au lieu de renvoyer les vecteurs de poids obtenus à la fin de l'entraînement, le perceptron moyenné renvoie la moyenne des vecteurs de poids obtenus successivement pendant l'apprentissage. Cette moyenne peut se calculer efficacement (sans stocker tous les vecteurs de poids successifs), à l'aide de l'algorithme donné en pseudocode à la fin de l'énoncé.

1. Implémentez la méthode `update_avg` qui réalise l'update des poids du perceptron dans le cas du perceptron moyenné.
2. Implémentez la méthode `average_weights` qui réalise le moyennage
3. Modifiez la fonction `learn` pour passer au perceptron moyenné.

Vous devriez obtenir de meilleurs résultats. Remarquez que le nombre de mises à jour à chaque itération ne change pas par rapport à la version non moyennée (il y a de très petites variations liées au shuffle du corpus):

```
Epoch 0 : train acc = 95.52 dev acc = 93.31 updates = 45018
Epoch 1 : train acc = 96.59 dev acc = 94.13 updates = 25999
Epoch 2 : train acc = 97.22 dev acc = 94.45 updates = 20355
Epoch 3 : train acc = 97.63 dev acc = 94.65 updates = 16592
Epoch 4 : train acc = 97.88 dev acc = 94.78 updates = 14076
Epoch 5 : train acc = 98.1 dev acc = 94.88 updates = 12339
Epoch 6 : train acc = 98.28 dev acc = 94.89 updates = 10968
Epoch 7 : train acc = 98.43 dev acc = 94.92 updates = 9783
Epoch 8 : train acc = 98.56 dev acc = 94.94 updates = 8859
Epoch 9 : train acc = 98.67 dev acc = 94.97 updates = 8235
Epoch 10 : train acc = 98.78 dev acc = 94.97 updates = 7478
Epoch 11 : train acc = 98.88 dev acc = 94.94 updates = 6941
```

4. Analyse d'erreurs

- Sur le corpus de test, observez des erreurs obtenues sur des mots ambigus et sur des mots qui ne sont pas dans le corpus d'entraînement (c'est-à-dire les mots inconnus ou *OOV* pour *out of vocabulary*).

5. Expériences hors domaine.

1. Pourquoi ne peut-on pas utiliser le corpus hors domaine pour apprendre un étiqueteur morphosyntaxique ?
2. Faire des séparations train/test aléatoires (90% / 10%) et étudier comment varie l'exactitude.
3. Entraîner un tagger sur le corpus *in-domain* et évaluez-le sur le corpus *out-domain*. Comment évoluent les erreurs ? Proposez des explications (on pourra regarder par exemple les différences de distribution des pos-tags d'un corpus à l'autre).

Annexe: algorithme du perceptron moyenné

Algorithm 1: Perceptron moyenné

Input: N: nombre d'itérations
Input: $\{(y^{(i)}, \mathbf{x}^{(i)})\}_{i \in \{1,\dots,M\}}$: exemples d'entraînement
Output: $\mathbf{w}$: vecteurs de poids pour chacune des classes

$\mathbf{w}_y \leftarrow (0, \dots, 0), \forall y$
$\mathbf{c}_y \leftarrow (0, \dots, 0), \forall y$
$T \leftarrow 1$
for $n \leftarrow 1$ **to** N **do**
    for $i \leftarrow 1$ **to** M **do**
        $\hat{y} \leftarrow \arg\max_y \{\mathbf{w}_y \cdot \mathbf{x}^{(i)}\}$
        if $\hat{y} \neq y^{(i)}$ **then**
            $\mathbf{w}_{y^{(i)}} \leftarrow \mathbf{w}_{y^{(i)}} + \mathbf{x}^{(i)}$
            $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \mathbf{x}^{(i)}$

            $\mathbf{c}_{y^{(i)}} \leftarrow \mathbf{c}_{y^{(i)}} + T \cdot \mathbf{x}^{(i)}$
            $\mathbf{c}_{\hat{y}} \leftarrow \mathbf{c}_{\hat{y}} - T \cdot \mathbf{x}^{(i)}$
        $T \leftarrow T + 1$
return $\mathbf{w} - \frac{1}{T}\mathbf{c}$