Prof. Amal Zouaq – INF8460 (Automne 2019)
Chargé de laboratoire : Félix Martel

École Polytechnique de Montréal

Département Génie Informatique et Génie Logiciel

INF8460 – Traitement automatique de la langue naturelle

Objectifs d'apprentissage

- Savoir accéder à un corpus, le nettoyer et effectuer divers pré-traitements sur les données
- Apprendre à utiliser divers modèles n-grams pour entrainer un modèle de langue
- Apprendre à comparer des modèles n-grams en utilisant la perplexité
- Utiliser et comparer différentes méthodes pour le calcul des probabilités, incluant le MLE et des méthodes de lissage

Logiciels

Python avec `nltk`, `numpy` et `matplotlib`

Avant de commencer, vous devez télécharger certaines données NLTK :

```
nltk.download("punkt")
nltk.download("wordnet")
```

Lectures / Ressources

http://www.nltk.org/.

https://www.nltk.org/book/ch03.html

https://www.nltk.org/_modules/nltk/lm/api.html#LanguageModel

Modalités de remise du TP

La date de remise est spécifiée sur Moodle. Généralement vous avez deux semaines pour retourner votre TP. Le retour se fait à 11h55 pm le mardi, soit la veille de votre prochain labo.

Vous devez soumettre :

1) Votre code python avec:
   a. Un fichier `tp1_matricule1_matricule2_matricule3.zip` qui contient tous les fichiers .py complétés et la même structure que `tp1_squelette.zip`. En particulier, output doit contenir les fichiers txt générés dans la section 1.1.;
   b. Si besoin, un fichier `ReadMe.txt` pour expliquer certaines fonctions ou fichiers

2) Un rapport **pdf** dont le nom est formé **des numéros de matricules des membres de l'équipe** séparés par -. Vous devez utiliser la page de présentation-TP, disponible sur Moodle, comme page de garde de votre rapport. Le rapport doit contenir :
   a. La page de garde
   b. Des statistiques descriptives sur Shakespeare telles que demandées dans la section 1.1
   c. Les K n-grams les plus fréquents (K=20, n=1,2,3)
   d. Les perplexités des modèles
   e. Le graphe reportant la perplexité en fonction du paramètre gamma, tels que demandé dans la section 1.6 ainsi que la meilleure valeur de gamma.
   f. Les séquences générées sur le corpus Trump

Critères d'évaluation

- La réponse correcte à chaque question (code, sortie)
- La qualité du code incluant les commentaires
- La qualité et le professionnalisme du rapport, incluant sa présentation et son organisation

Corpus et code

Dans le fichier archive `TP1_squelette.zip`, vous trouverez :

- Cinq fichiers Python contenant les squelettes des fonctions à écrire, avec des détails sur les formats d'entrée et de sortie. Vous devrez compléter ces fichiers. Vous pouvez, si besoin est, ajouter des fonctions supplémentaires, mais ça ne devrait pas être nécessaire.
- Un dossier `data` contenant deux corpus :
  - Le corpus Shakespeare, qui contient des pièces de Shakespeare; il est séparé en `shakespeare_train.txt` (pour l'entraînement) et `shakespeare_test.txt` (pour le test)
  - Le corpus Trump, dans `trump.txt`, contient des tweets de Donald Trump
- **Important** : les fichiers Python fournis contiennent des détails sur les valeurs des paramètres à utiliser ou des indications sur la manière de procéder.

Travail à faire

1.1 Exploration des données (20 points)

Lecture et prétraitement

Dans cette section, vous devez compléter le fichier `preprocess_corpus.py`. Lorsqu'on exécute ce fichier, la fonction `test_preprocessing` doit être appelée sur Shakespeare train et test. Les différents fichiers de sortie doivent se retrouver dans le répertoire `output`.

1) Segmentez le corpus en phrases, et stockez-les dans un fichier `<nomcorpus>_phrases.txt` (une phrase par ligne)

2) Segmentez chaque phrase en mots (*tokenization*) et stockez-les dans un fichier `<nomcorpus>_mots.txt` (une phrase par ligne, chaque token séparé par un espace, il n'est pas nécessaire de stocker la phrase non segmentée ici) ;

3) Lemmatisez les mots et stockez les lemmes dans un fichier `<nomcorpus>_lemmes.txt` (une phrase par ligne, les lemmes séparés par un espace) ;

4) Retrouvez la racine des mots (*stemming*) en utilisant `nltk.PorterStemmer()`. Stockez-les dans un fichier `<nomcorpus>_stems.txt` (une phrase par ligne, les racines séparées par une espace) ;

5) Assemblez les fonctions précédentes dans une fonction `test_preprocessing(raw_text, sentence_id=0)` qui prend un corpus brut, effectue une segmentation en phrase puis en mot, une lemmatisation et un stemming, puis affiche le résultat de ces différentes opérations sur la phrase d'indice `sentence_id`.

Lorsqu'on exécute le fichier `preprocess_corpus.py`, cette fonction doit être appelée sur le corpus Shakespeare.

Exploration des données

6) Dans le fichier `explore_corpus.py`, complétez les fonctions retournant les informations suivantes (une fonction par information, chaque fonction prenant en argument un `corpus` composé d'une liste de phrases segmentées en tokens(tokenization)) :

a. Le nombre total de tokens (mots non distincts)
b. Le nombre total de mots distincts (les types qui constituent le vocabulaire)
c. Les N mots les plus fréquents du vocabulaire (N est un paramètre) ainsi que leur fréquence
d. Le ratio token/type
e. Le nombre total de lemmes distincts
f. Le nombre total de racines (stems) distinctes

7) Écrivez la fonction `explore(corpus)` qui calcule et affiche toutes ces informations, précédées d'une légende reprenant l'énoncé de chaque question (a,b, ….g).

8) Dans le bloc final `if __name__ == "__main__"`, appelez la fonction `explore` sur le corpus.

1.2 Modèle de langue n-gramme (20 points)

Vous devez implémenter en Python un modèle de langue n-gramme par l'estimé du maximum de vraisemblance (vu dans le cours). Vous devrez compléter les différentes fonctions et méthodes du fichier `mle_ngram_model.py`.

1) Écrire une fonction `extract_ngrams_from_sentence(sentence, n)` qui énumère les n-grammes d'une phrase tokenizée. Vous pouvez utiliser `nltk.ngrams` ainsi que `nltk.lm.preprocessing.pad_both_ends`.

Pour les numéros 2-6, vous devez implanter les fonctions vous-mêmes et vous ne pouvez pas utiliser l'api `nltk.lm.models`

2) En utilisant la fonction précédente, écrivez une fonction `extract_ngrams(corpus, n)` qui, à partir d'un corpus tokenizé, renvoie la liste des n-grammes de chaque phrase du corpus.

3) Écrire une fonction `count_ngrams(corpus, n)` qui compte les n-grammes du corpus. Cette fonction doit renvoyer un object `counts` tel que, si context $= (w_{i-n+1}, \dots, w_{i-1})$, alors `counts[context][w]` soit égal à Count$(w_{i-n+1}, \dots, w_{i-1}, w)$. Ainsi, dans le cas $n = 1$, on aura `context = ()`; dans le cas $n = 2$, `context = (v, )`; et dans le cas $n = 3$, `context=(u, v)`.

4) Écrivez une fonction `compute_MLE(counts)` qui normalise l'objet `counts` précédent en y stockant l'estimé du maximum de vraisemblance (MLE) tel que vu en cours. La fonction renvoie un objet `mle_counts` tel que `mle_counts[(u,v)][w]` est égal à $P_{MLE}(w \mid u, v)$.

On a encapsulé les fonctions précédentes dans la classe `NgramModel`, ce qui vous permet d'entraîner un modèle n-gramme avec la commande `lm = NgramModel(corpus, n)`.

5) Dans le bloc `__main__`, entraînez des modèles sur le corpus de Shakespeare_train avec n=1, 2, 3.

6) Complétez la méthode `NgramModel.predict_next(self, context)` qui, à partir d'un contexte (c'est-à-dire un tuple de longueur `n-1 où n=1,2,3`), tire un mot w selon la distribution de probabilité $P(w \mid \text{context})$.

7) Toujours dans `__main__`, testez la méthode `predict_next` sur vos trois modèles. La liste des contextes à tester est fournie dans le squelette Python.

1.3. Validation/Comparaison du modèle (10 points)

Pour vérifier que votre implémentation est correcte, on va maintenant comparer ses résultats avec ceux du modèle MLE de NLTK : `nltk.lm.models.MLE`, en vérifiant que les deux modèles attribuent les mêmes probabilités aux n-grammes. La documentation de l'API est disponible ici : https://www.nltk.org/api/nltk.lm.html#nltk.lm.api.LanguageModel

Dans le fichier `mle_model_validation.py` :

8) Complétez la fonction `train_MLE_model(corpus, n)` qui entraîne un modèle MLE de NLTK d'ordre n sur le corpus. Dans le bloc `__main__`, entraînez des modèles sur le corpus de Shakespeare_train avec n=1, 2, 3.

9) Écrivez la fonction `compare_models(your_model, nltk_model, corpus, n)` qui compare les probabilités des modèles `your_model` et `nltk_model` pour tous les n-grammes de `corpus`, et affiche les n-grammes qui diffèrent le cas échéant. Dans le bloc `__main__`, appelez la fonction `compare_models` pour comparer les modèles NLTK avec vos propres modèles (n=1,2,3).

1.4. Méthodes de lissage (20 points)

Le modèle MLE n'est pas très satisfaisant, car il attribue une probabilité nulle aux n-grammes qui n'ont pas été vus à l'entraînement. On va donc essayer deux méthodes de lissage pour pallier ce problème: Lidstone (add-*k*) et Laplace (add-one, variante de la précédente). On les comparera avec le modèle MLE sans lissage de la section précédente, dans son implémentation NLTK.

Pour cela, vous devez compléter le fichier `nltk_models.py`, qui servira aussi pour les deux sections suivantes. Lisez attentivement la section 1.4 et 1.5 car elles sont complémentaires.

1) Définissez une fonction `train_LM_model(corpus, model, n, gamma=None, unk_cutoff=2)` qui entraîne un modèle de langue sur un `corpus` constitué de phrases tokenizées. Cette fonction doit prendre en charge MLE, Lidstone et Laplace. Notez que pour MLE, vous avez déjà implémenté le code dans `train_MLE_model(corpus, n)` de `mle_model_validation.py`. Vous pouvez le reproduire ici.

2) Utilisez la fonction pour entraîner deux des modèles de langue (MLE, Laplace) pour n=1, 2 et 3 sur Shakespeare_train. L'appel à Lidstone se fera dans la section 1.5 avec la fonction `evaluate_gamma`.

1.5 Évaluation des modèles (20 points)

Pour évaluer la qualité d'un modèle de langue, on a fréquemment recours à la **perplexité**. Pour une phrase $S = (w_1, \dots, w_N)$ de longueur N, on a :

$$PP(S) = \left(\prod_{i=1}^{N} P\left(w_i | w_{i-n+1}, \dots, w_{i-1}\right) \right)^{-\frac{1}{N}}$$

Dans le fichier `nltk_models.py` :

1) Définissez une fonction `evaluate(model, corpus)` qui renvoie la perplexité d'un modèle sur un corpus de texte donné. Vous pouvez utiliser NLTK pour le calcul de la perplexité.

2) Pour chaque modèle entraîné à la question 1.4.2, évaluez sa perplexité sur `shakespeare_test`. Reportez les résultats dans votre rapport. Quel est votre meilleur modèle ? Comment expliquez-vous les résultats du modèle MLE ?

3) On vous fournit une fonction `evaluate_gamma(gamma, train, test, n)` qui entraîne un modèle Lidstone avec une valeur de γ donnée sur un corpus `train`, et renvoie sa perplexité sur le corpus `test`. En utilisant cette fonction avec les corpus shakespeare_train et test, vous devez tracer un graphe représentant l'évolution de la perplexité en fonction de γ. Comme valeurs de γ à tester, vous prendrez l'intervalle `numpy.logspace(-5, 0, 10)`.

Que se passe-t-il quand γ tend vers zéro ? Quelle valeur de γ donne les meilleurs résultats ? Indiquez-le dans votre rapport.

1.6 Génération de texte (corpus Trump) (10 points)

Une autre façon d'inspecter la qualité des modèles n-grammes est de générer des textes au moyen de ces modèles et d'évaluer ces textes qualitativement.

Prof. Amal Zouaq – INF8460 (Automne 2019)
Chargé de laboratoire : Félix Martel

Dans le fichier `nltk_models.py` :

1) Écrivez une méthode `generate(model, n_words, text_seed=None, random_seed=None)` qui génère un texte de longueur `n_words` en se basant sur le modèle `model`.

2) Pour n=1, 2, 3, entrainez le modèle n-gramme MLE sur le corpus **Trump** et générez deux segments de 20 mots par n (réglez le paramètre `unk_cutoff` à 1). Reportez les résultats dans votre rapport et indiquez vos observations. Commentez la qualité du résultat obtenu.