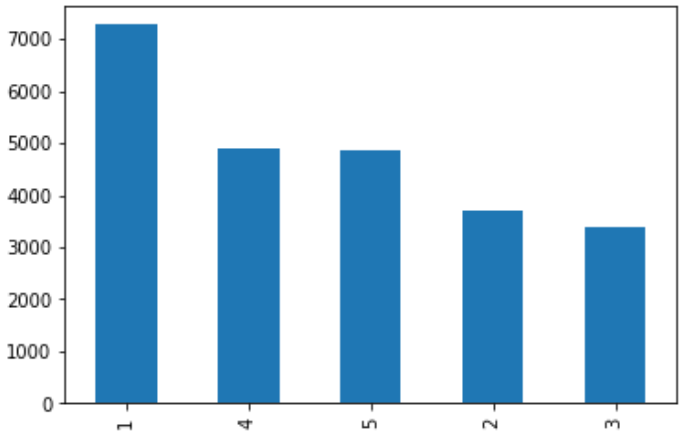
Advanced Machine Learning for NLP

Project 2 : Insurance Reviews

Maxime Louward, Aladin Homsy

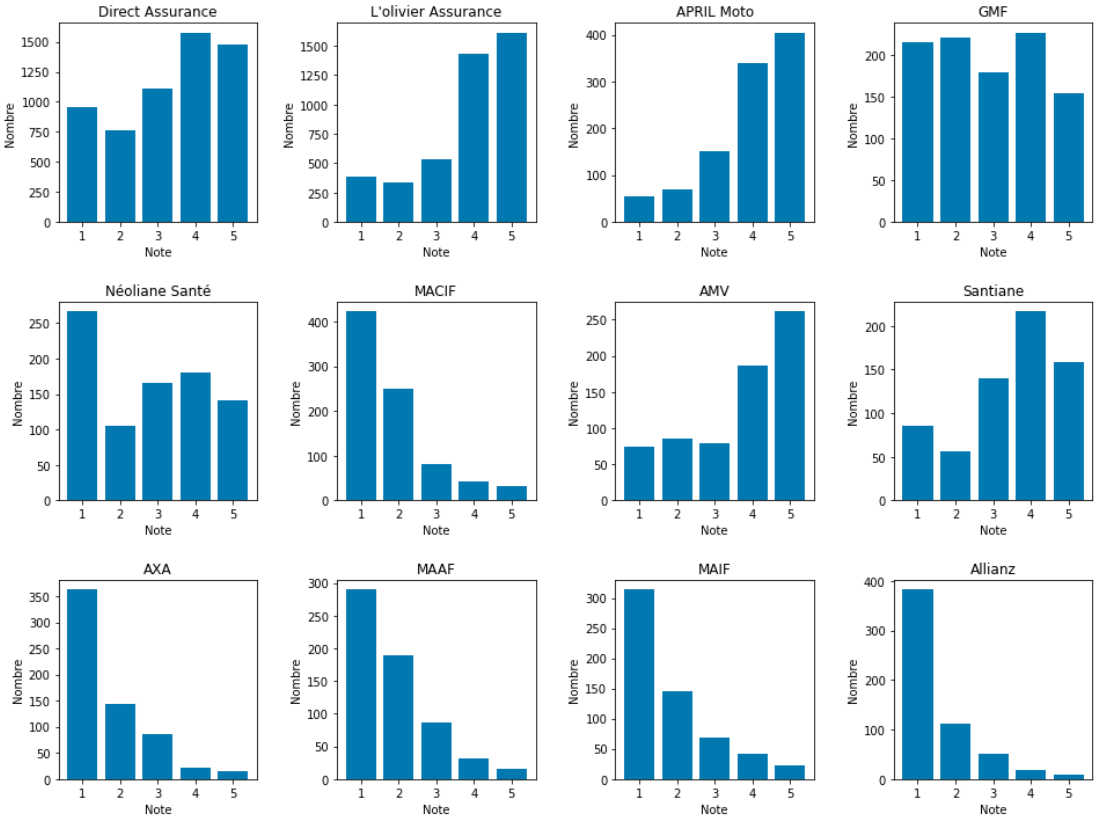
I. Exploration des données

Nous avons chargé le dataset de train dans pandas, convertit les colonnes dans leur type et exploré plus en détails les différentes données disponibles. Tout d'abord, voici la répartition des notes dans le dataset :



On voit que la note la plus fréquente est 1/5, ce qui veut dire que les gens mettent en général un avis lorsqu'ils sont mécontents du service.

On peut ensuite s'intéresser à la répartition des notes pour les 12 assureurs le plus courants :



On voit que cette répartition varie grandement. Pour les assureurs le plus connus (Allianz, AXA, MAAF, …) les notes semblent très basses, ce qui pourrait être biaisé, tandis que pour d'autres assurances comme pour l'Olivier et APRIL Moto, qui sont parmi les assureurs les plus présents dans le dataset, les notes sont relativement hautes.

II. Apprentissage non supervisé

II.I Nettoyage, tokenization

Pour le nettoyage, nous avons choisi de supprimer toute la ponctuation, les accents, ainsi que les stopwords. Ensuite, nous avons tokenizé avec nltk et supprimé les mots de moins de 3 lettres :

```python
fr_stop = set(stopwords.words("french"))
fr_stop.update({"a", "les"})


def preprocess_texts(texts: List[str]) → List[List[str]]:
    """
    Preprocess texts and remove stopwords.
    Remove punctuation and numbers.
    Keeps only nouns, adjectives, verbs and adverbs.
    Args:
      texts: List of texts to preprocess.

    Returns:
      tokens: List of tokens for each text.
    """
    tokens: List[List[str]] = []
    punctuation = r'!"#$%&\''*+-.,/:;⟺?()[]{}@\^_`|~0123456789' + "\r\n"
    t = str.maketrans(dict.fromkeys(punctuation, " "))
    # remove punctuation
    texts = [doc.translate(t) for doc in texts]
    # remove accents
    texts = [unidecode.unidecode(doc) for doc in texts]
    # set to lowercase
    texts = [doc.lower() for doc in texts]
    # tokenize with nltk
    tokens = [word_tokenize(doc) for doc in texts]
    # remove tokens with length < 3
    tokens = [[word for word in doc if len(word) ⩾ 3] for doc in tokens]
    # remove stopwords
    tokens = [[word for word in doc if word not in fr_stop] for doc in tokens]
    return tokens


tokenized_reviews = preprocess_texts(reviews)

[11]
```

Voici un exemple d'avis tokenizé : ['prix', 'semblent', 'tres', 'corrects', 'rapport', 'ancienne', 'assurance', 'demarche', 'tres', 'simple', 'reste', 'maintenant', 'voir', 'pratique', 'instant', 'plutot', 'satisfaite']

II.II Lemmatization, Bigrammes et Trigrammes

Nous avons cette fois utilisé les librairies spacy et gensim pour faire la lemmatization ainsi que pour créer des bigrammes et des trigrammes :

Lemmatization

```python
# Initialize spacy 'fr_core_news_lg' model, keeping only tagger component (for efficiency)
nlp = spacy.load("fr_core_news_lg", disable=["parser", "ner"])


def lemmatize_bigrams_trigrams(
    tokenized_texts,
    do_lemmatization: bool = True,
    allowed_postags: List[str] = ["NOUN", "ADJ", "VERB", "ADV"],
):
    """
    Create bigrams and trigrams from a list of tokenized texts.
    Lemmatizes and removes stopwords from the result.
    Args:
      tokenized_texts: List of tokenized texts.
      allowed_postags: List of allowed parts of speech.

    Returns:
      texts_out: List of lemmatized texts with bigrams/trigrams.
    """
    bigrams = [bigram_mod[doc] for doc in tokenized_texts]
    trigrams = [trigram_mod[bigram_mod[doc]] for doc in bigrams]
    if do_lemmatization:
        texts_out = []
        for sent in trigrams:
            doc = nlp(" ".join(sent))
            texts_out.append(
                [token.lemma_ for token in doc if token.pos_ in allowed_postags]
            )
        return texts_out
    return trigrams

```

Voici quelques exemples de trigrammes dans notre dataset : 'personne_bout_fil', 'lettre_mise_demeure', 'regard_politique_acceptation', 'pare_choc_arriere', ‘assuree_tous_risques’.

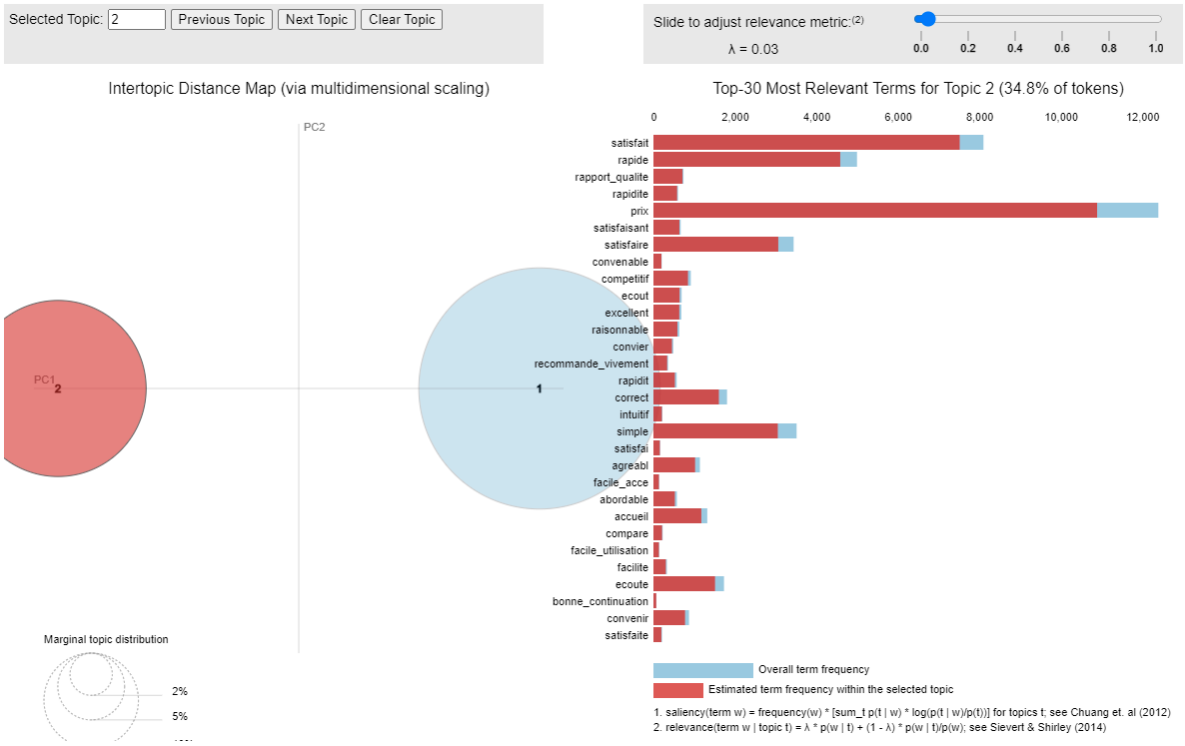
II.III LDA

Nous avons utilisé la librairie gensim pour effectuer une LDA sur nos tokens. Après avoir essayé plusieurs combinaisons de paramètres, nous avons trouvé les meilleurs résultats avec ceux-ci :

```python
ldamodel = models.ldamulticore.LdaMulticore(
    corpus,
    num_topics=NUM_TOPICS,
    id2word=id2word,
    chunksize=2200,
    passes=5,
    alpha="asymmetric",
    per_word_topics=True,
    workers=10,
)
```

Les scores obtenus sont -7.35 pour la perplexity et 0.46 pour la cohérence.

Pour la visualisation, nous avons choisi 2 clusters. En regardant les mots de chaque cluster, on réalise que la séparation s'est faite sur l'aspect positif ou négatif des mots. Ainsi, le premier cluster contient des mots tels que « satisfait », « rapide », « simple », … et le second cluster «



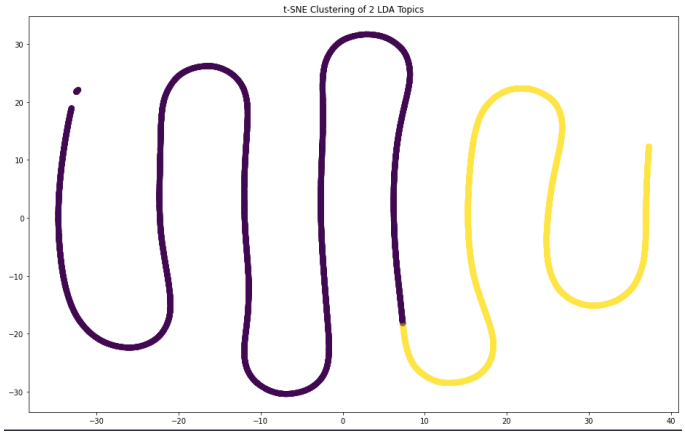
Ensuite, on a pu catégoriser les avis en fonction du cluster auquel ils appartiennent, ainsi que leur contribution à ce cluster :

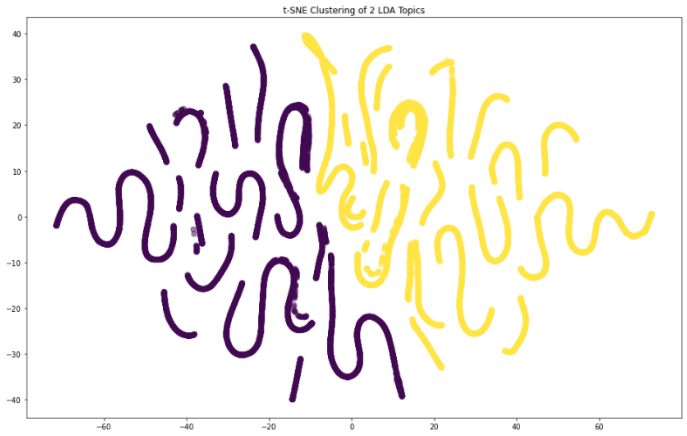
	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	Text
0	0	1	0.9655	service, prix, assurance, tres, satisfait, tre...	[prix, sembler, tres, correct, rapport, ancien...
1	1	0	0.7922	assurance, plus, faire, contrat, mois, dire, s...	[fuir, competer, remboursement, parler, choix,...
2	2	1	0.9617	service, prix, assurance, tres, satisfait, tre...	[assez, satisfait, ensemble, prix, correct, pe...
3	3	1	0.9541	service, prix, assurance, tres, satisfait, tre...	[satisfait, rapport, exigence, prix, correct, ...
4	4	1	0.9777	service, prix, assurance, tres, satisfait, tre...	[bon, prix, simple, pratique, satisfait, servi...
5	5	1	0.9548	service, prix, assurance, tres, satisfait, tre...	[recu, emelin, jour, accueil, point, parfait, ...
6	6	0	0.8244	assurance, plus, faire, contrat, mois, dire, s...	[harmonie, mutuel, refuse, faire, photocopie, ...
7	7	1	0.9525	service, prix, assurance, tres, satisfait, tre...	[satisfait, prix, devis, beaucoup, moins, cher...
8	8	1	0.7950	service, prix, assurance, tres, satisfait, tre...	[satisfait, assurance, auto, direct, assurance...
9	9	1	0.5991	service, prix, assurance, tres, satisfait, tre...	[assure, an, reste, satisfait, service, voitur...

II.IV t-SNE LDA et Word2Vec

I. LDA

En changeant la valeur de la perplexité et du nombre d'itérations, on obtient des formes très différentes. Dans le graphe de gauche, la perplexité vaut 300, et dans celui de droite, elle vaut 100. Nous pensons que le deuxième graphe représente mieux la séparation des données entre les 2 topics (avis plutôt positifs et avis plutôt négatifs)

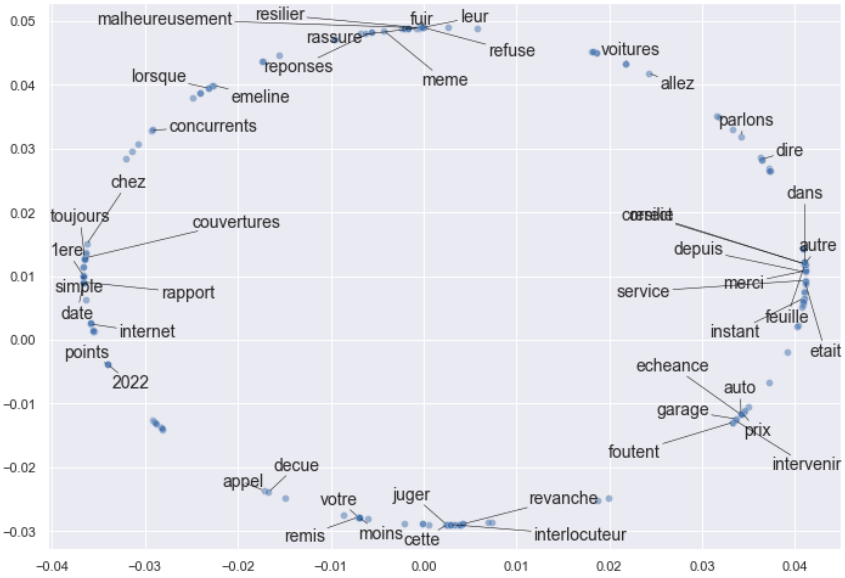




II. Word2Vec

Nous avons aussi entraîné un modèle Word2Vec sur nos textes, et la t-SNE permet de découvrir quelques clusters de mots ressemblants :

Ainsi, on trouve
« malheureusement »,
« résilier », « fuir », « refuse »
au même endroit, ainsi que
« garage » et « auto »,
« appel » et
« interlocuteur »…



III. Apprentissage supervisé

Pour générer les notes à partir du texte de l'avis, nous avons tout d'abord utilisé un modèle multilingual HuggingFace (nlptown/bert-base-multilingual-uncased-sentiment) qui permet de donner une note entre 1 et 5 à partir de texte. Ensuite, nous exécutons ce modèle sur nos textes pour créer une nouvelle colonne de notes, que l'on utilise comme feature supplémentaire dans un modèle XGBoost pour prédire la note en utilisant aussi les autres colonnes du dataset.

III.I Modèle HuggingFace

Nous avons testé le modèle trouvé sur une centaine de textes, et nous obtenions une RMSE d'environ 0.9. Nous avons donc décidé de le fine-tuner sur une partie de notre dataset et l'évaluer sur l'autre partie. Une fois le modèle fine-tuné, la RMSE était passée à 0.65 sur une centaine de lignes du dataset.

L'idée est donc ensuite d'utiliser ces prédictions comme nouvelle feature dans notre dataset de base.

```python
training_args = TrainingArguments(
    output_dir="./results",
    learning_rate=2e-5,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    num_train_epochs=3,
    weight_decay=0.01,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()
```

III.II XGBoost

Nous avons donc ajouté cette colonne au dataframe pandas, puis créé quelques features à partir de la date (jour, mois, année et jour de la semaine), ainsi que fait le one hot encoding pour les colonnes *assureur* et *produit*. Enfin, nous avons pu drop les colonnes date, auteurs (ils sont quasiment tous distincts donc apportent peu d'information) et avis (déjà pris en compte dans la colonne *prediction*). Notre dataset comporte désormais 74 colonnes ainsi que la target, *note*.

Voici les paramètres utilisés pour XGBoost après en avoir essayé plusieurs :

```python
model = XGBClassifier(
    reg_alpha=20,
    reg_lambda=20,
    tree_method="hist",
    max_bin=512,
    subsample=0.8,
)
```

Avec cette méthode, la RMSE passe de 0.71 pour HuggingFace seul à 0.69 pour le dataset de base + la prédiction de HuggingFace.

Enfin, nous avons répété ces étapes sur le dataset de test : feature engineering, prédiction de la note avec HuggingFace et enfin prédiction de la note finale avec XGBoost.

Nos prédictions se trouvent dans le fichier *predictions_final.csv*