Catégorisez automatiquement des questions

par Ana Bernal

Mars 2023

OpenClassrooms

Mentor: Samir Tanfous

Programme

1. Rappel **mission**
2. Pre-processing + exploration
3. **Modélisation** + évaluation + choix de modèle
4. Développement **API**
5. **Conclusions**

1

Mission

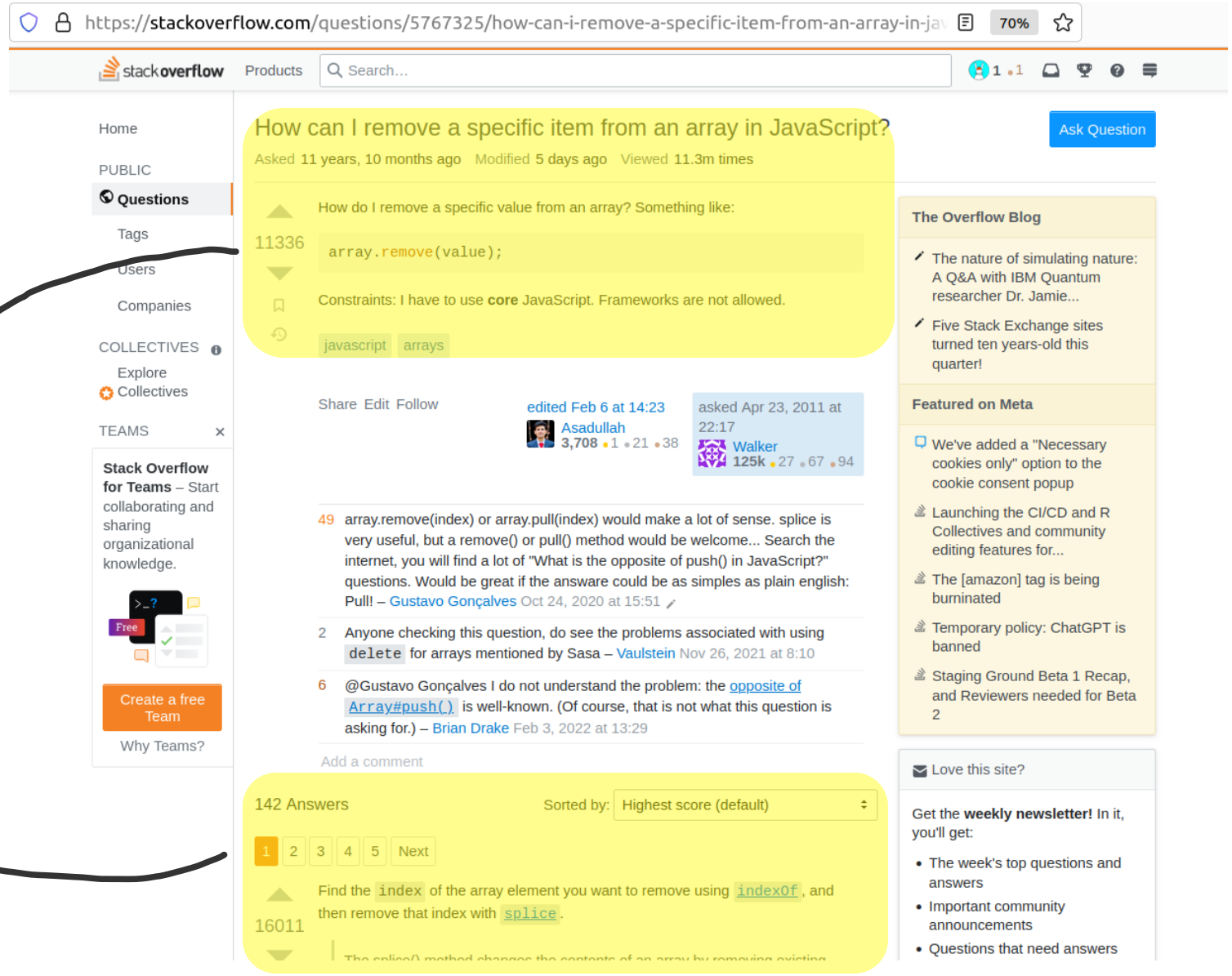
Mission

À propos de

stackoverflow

question

réponses



Mission

À propos de

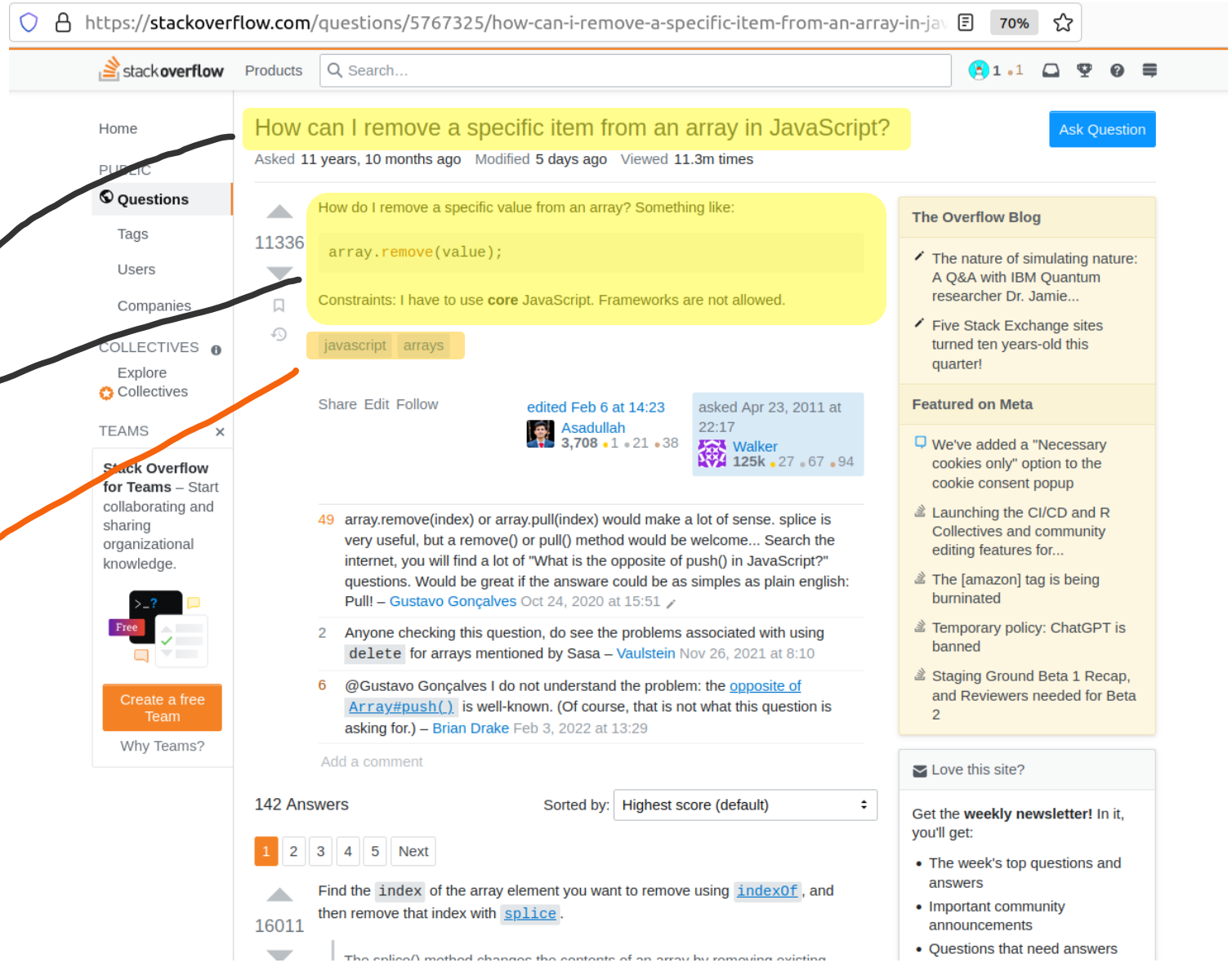
stackoverflow

titre

corps

étiquettes

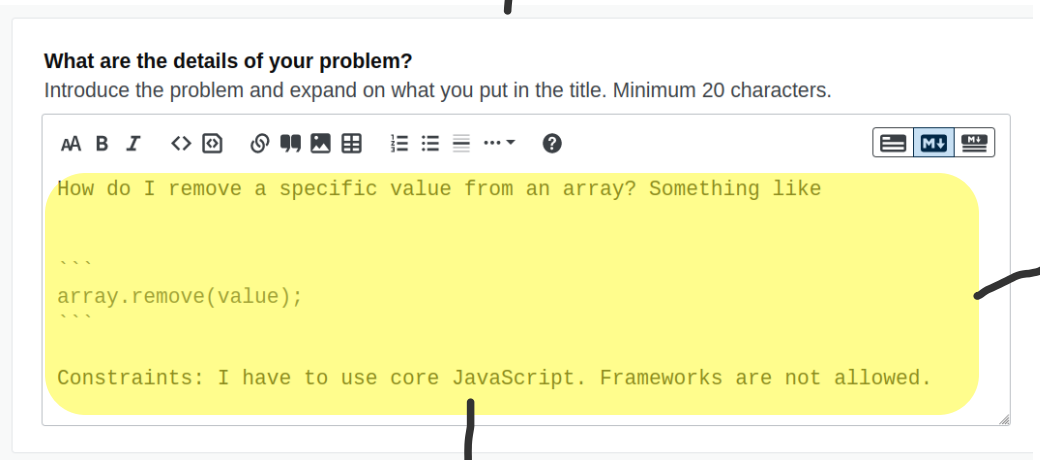
(tags)



Mission

Notre mission:

Input



Texte : question sur programmation (en anglais)

Algorithme de ML

Output

Liste de tags

javascript

arrays

* Avec API

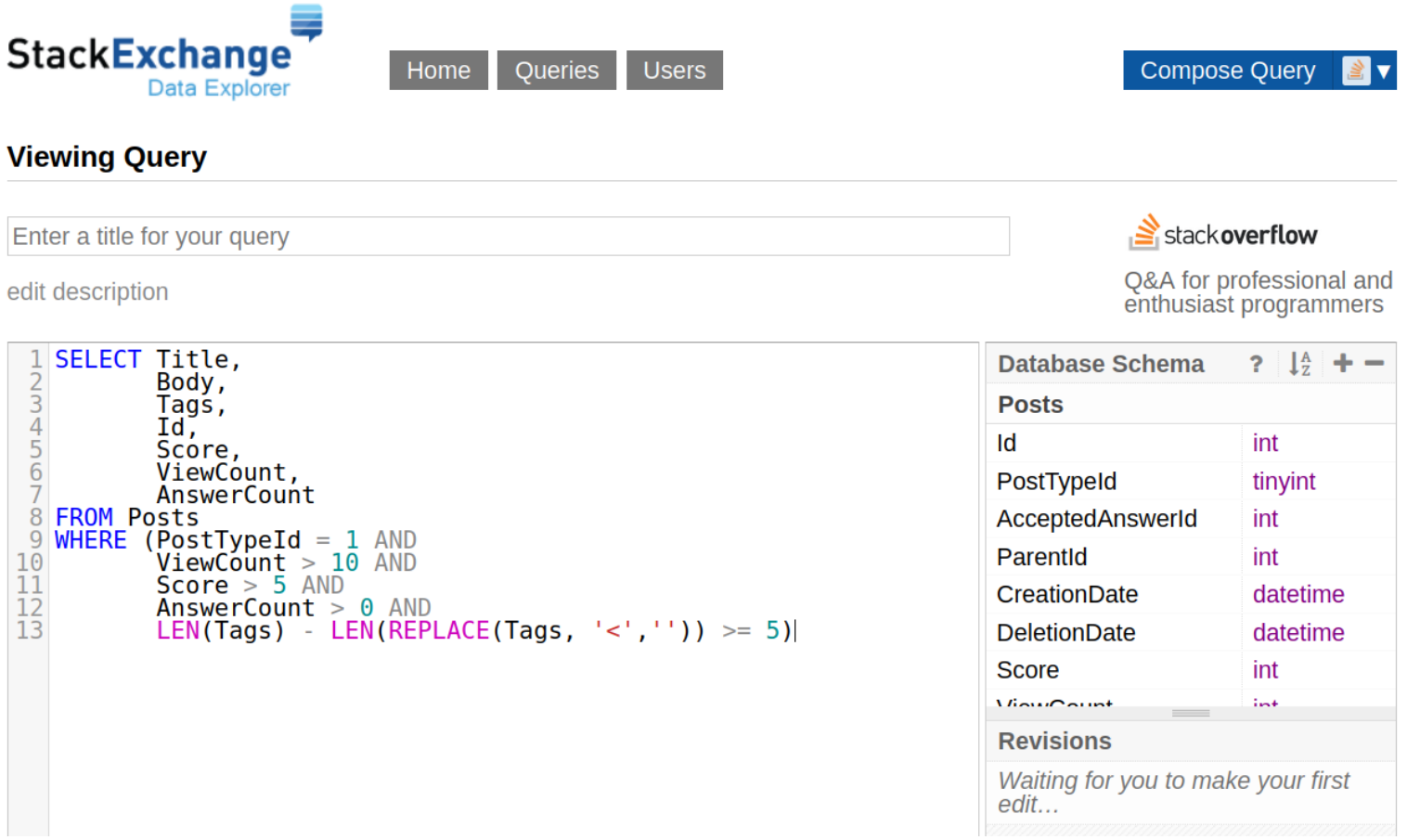
* système gestion version

2

Pre-processing

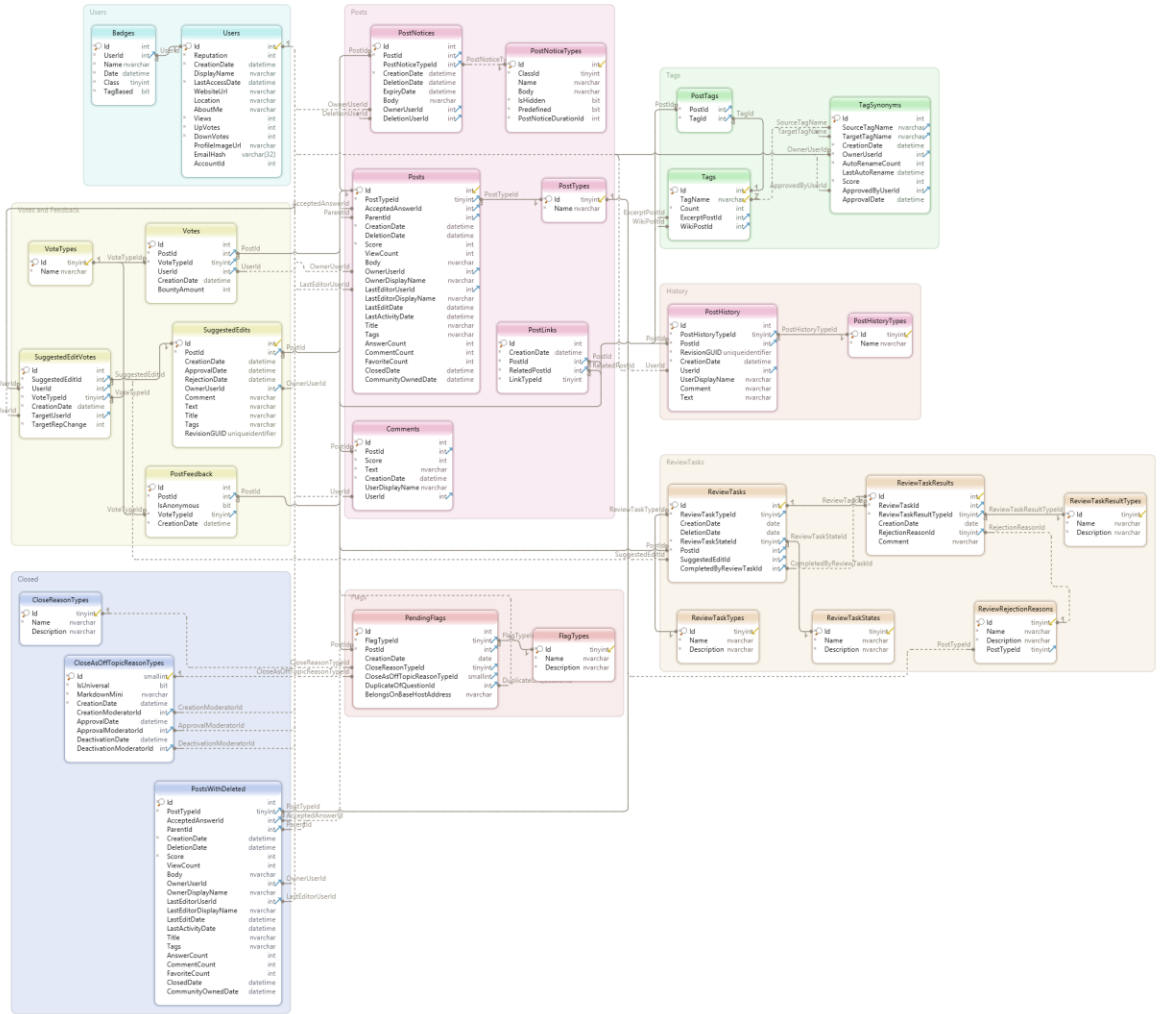
Pre-processing

Obtention des données avec StackExchange Data Explorer



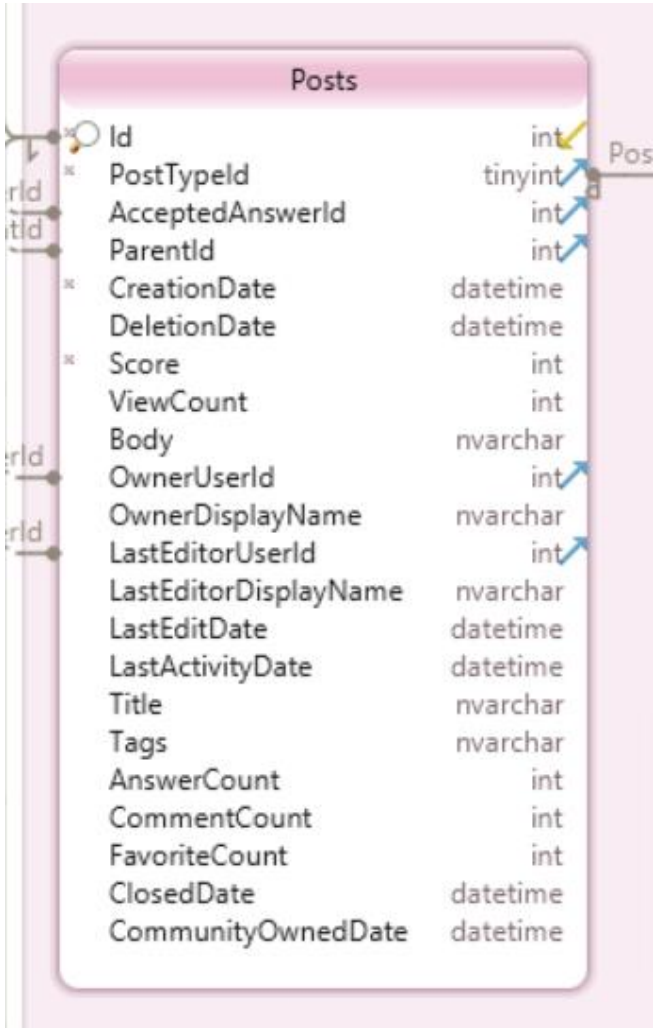
Pre-processing

Obtention des données avec StackExchange Data Explorer



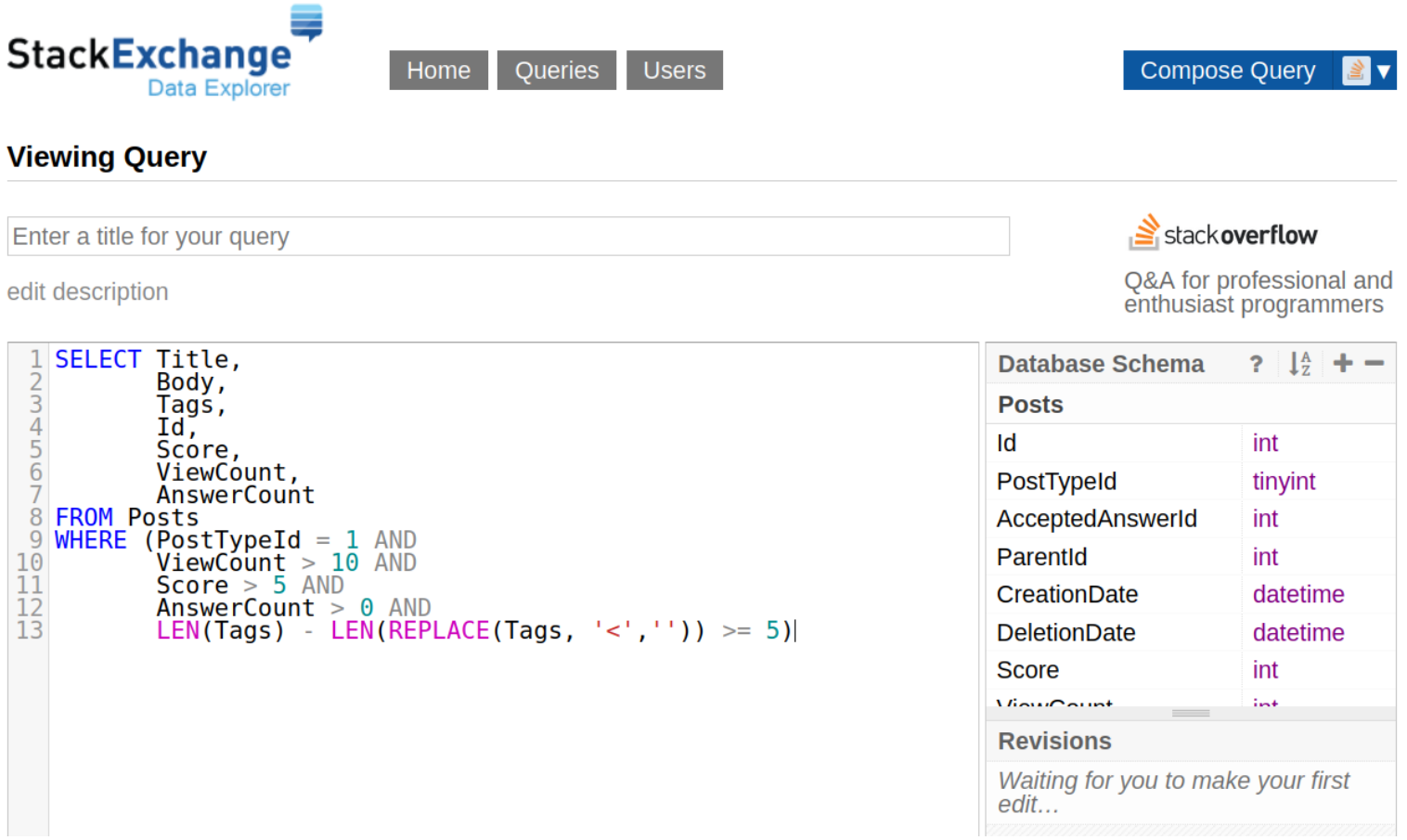
Pre-processing

Obtention des données avec StackExchange Data Explorer



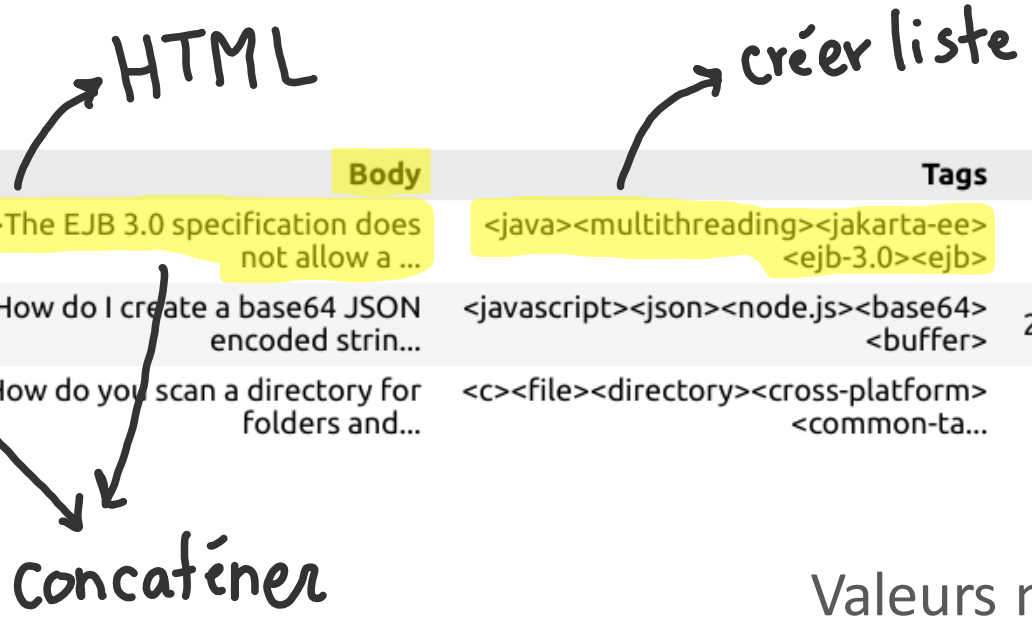
Pre-processing

Obtention des données avec StackExchange Data Explorer



Pre-processing

Aperçu des données



	Title	Body	Tags	Id	Score	ViewCount	AnswerCount
0	Multithreading in a stateless session bean?	<p>The EJB 3.0 specification does not allow a ...	<java><multithreading><jakarta-ee><ejb-3.0><ejb>	3816286	9	13503	4
1	base64 JSON encoded strings in nodejs	<p>How do I create a base64 JSON encoded strin...	<javascript><json><node.js><base64><buffer>	22515180	14	23710	3
2	How do you get a directory listing in C?	<p>How do you scan a directory for folders and...	<c><file><directory><cross-platform><common-ta...	12489	69	142856	9

Taille

Nombre de lignes	50 000
Nombre de colonnes	7

Valeurs manquantes et dtypes

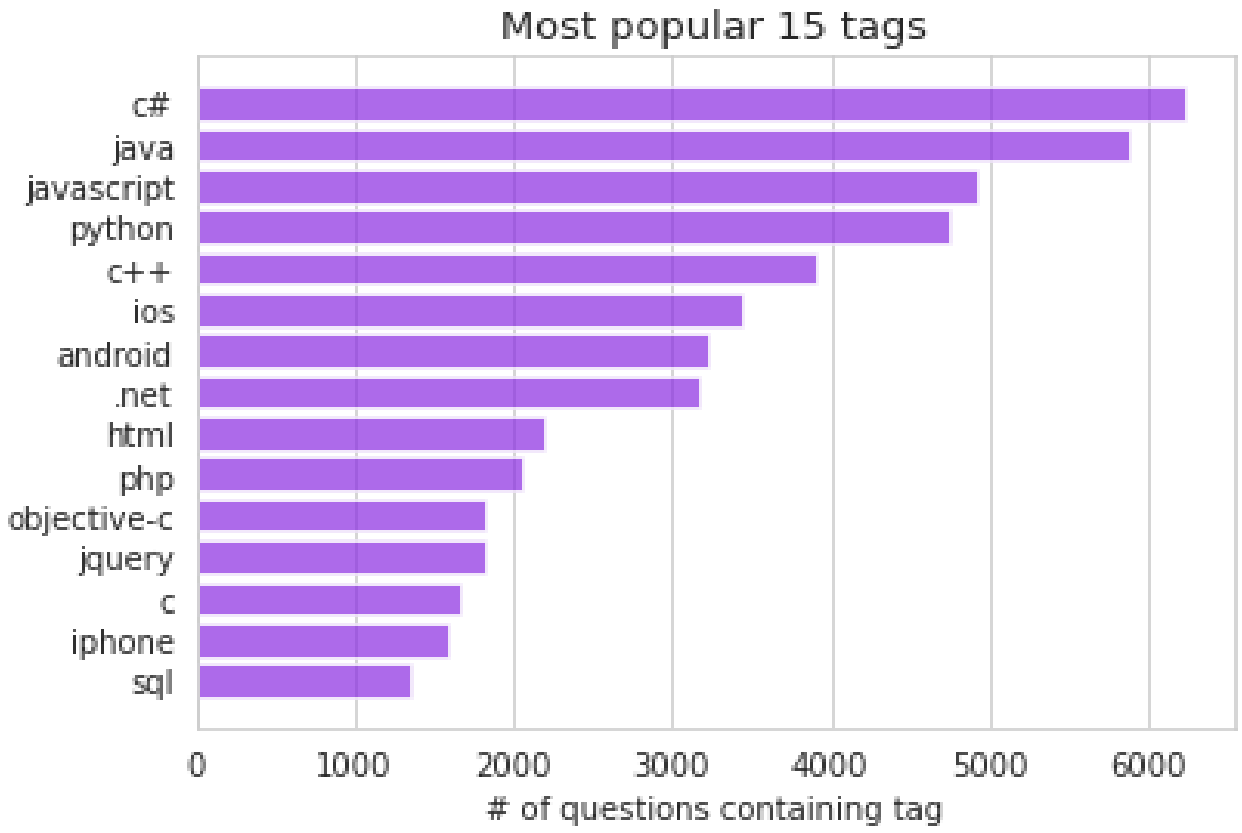
```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Title        50000 non-null  object
 1   Body         50000 non-null  object
 2   Tags         50000 non-null  object
 3   Id           50000 non-null  int64
 4   Score        50000 non-null  int64
 5   ViewCount    50000 non-null  int64
 6   AnswerCount  50000 non-null  int64
```

Pre-processing

Filtrer ensemble de tags: + garder docs. contenant au moins 1 de ces tags

The total # of popular tags is 38
(appearing in more than 700 questions)

Tag	# of quest
c#	6226.0
java	5880.0
javascript	4920.0
python	4747.0
c++	3904.0
ios	3449.0
android	3223.0
.net	3164.0
html	2197.0
php	2050.0
objective-c	1816.0
jquery	1810.0
c	1655.0
iphone	1592.0
sql	1349.0
asp.net	1339.0
css	1331.0
linux	1286.0
node.js	1253.0
performance	1076.0



Pre-processing

Création de target avec ces 38 tags:

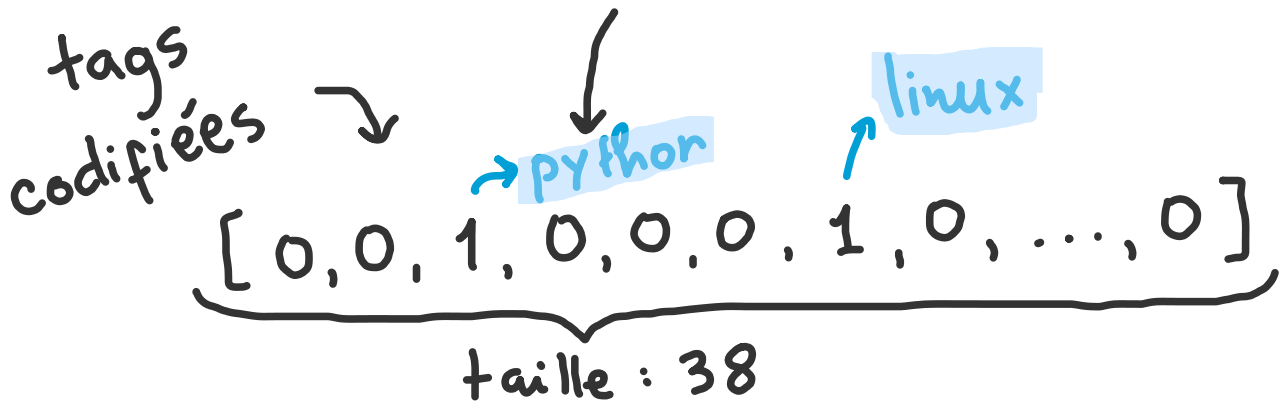
```
The total # of popular tags is 38
(appearing in more than 700 questions)
----------------------------------
Tag              # of quest
----------------------------------
        c#       6226.0
      java       5880.0
javascript       4920.0
    python       4747.0
       c++       3904.0
       ios       3449.0
   android       3223.0
      .net       3164.0
      html       2197.0
       php       2050.0
objective-c      1816.0
    jquery       1810.0
         c       1655.0
    iphone       1592.0
       sql       1349.0
   asp.net       1339.0
       css       1331.0
     linux       1286.0
   node.js       1253.0
performance      1076.0
```

Creating target

```
1 mlb = MultiLabelBinarizer(classes=popular_tags)
2 mlb.fit([popular_tags])
```

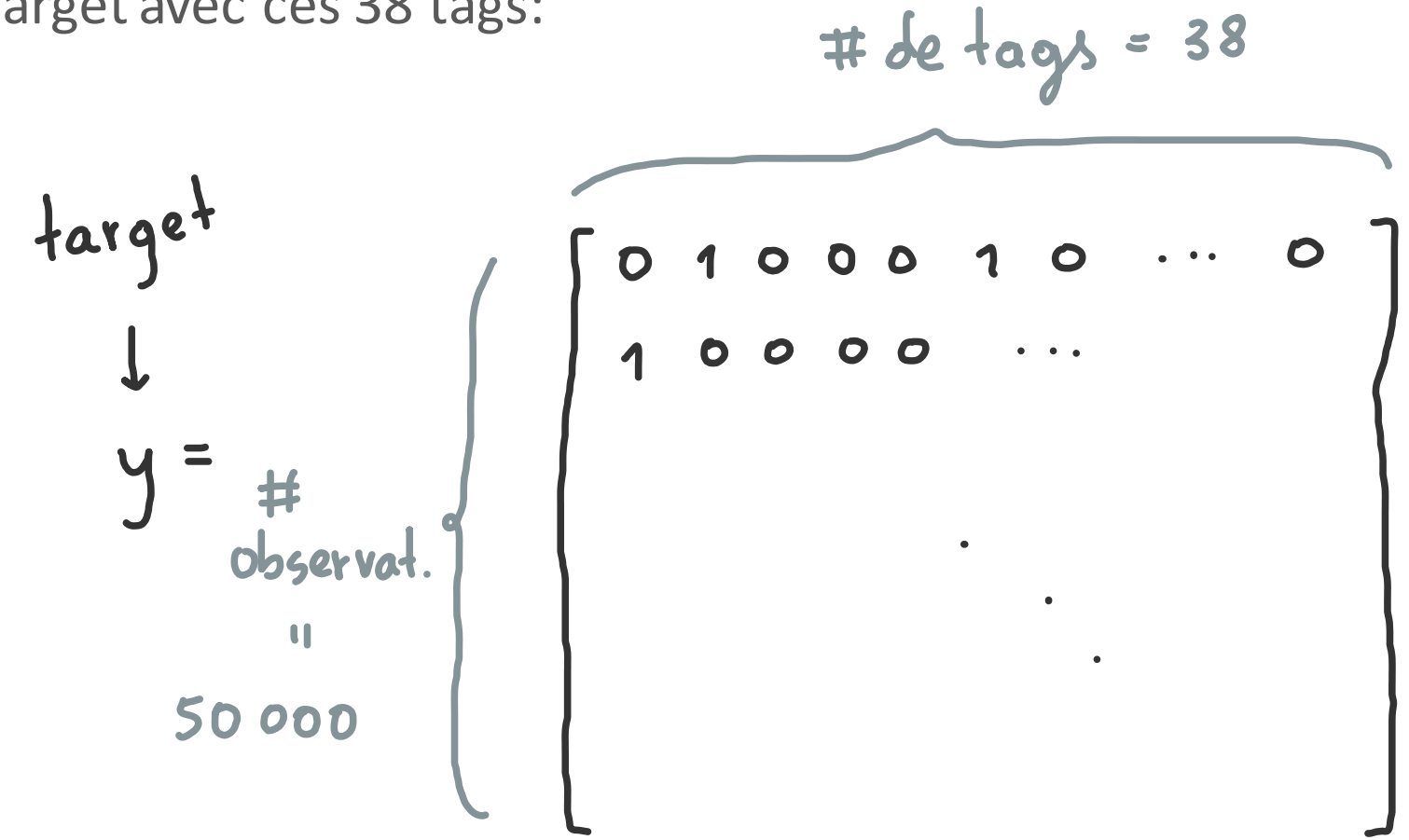
MultiLabelBinarizer

```
MultiLabelBinarizer(classes=['c#', 'java', 'javascript', 'python', 'c++', 'ios',
                             'android', '.net', 'html', 'php', 'objective-c',
                             'jquery', 'c', 'iphone', 'sql', 'asp.net', 'css',
                             'linux', 'node.js', 'performance', 'spring',
                             'windows', 'swift', 'xcode', 'ruby-on-rails',
                             'mysql', 'json', 'sql-server', 'multithreading',
                             'asp.net-mvc', ...])
```



Pre-processing

Création de target avec ces 38 tags:



Pre-processing

Création de fonction de nettoyage/tokenisation de texte

```python
def clean(text,tokenize=False,strict=False, **kwargs):
    """
    Returns a dictionnary with keys 'text' or 'tokens', where
    'tokens' corresponds tothe list of lemmatized tokens from
    the string text. Ommiting stopwords and punctuation, and the text is
    the joint text.

    Parameters:
        - text: str
        - tokenize: bool
            If True returns list of tokens, if False returns string.
        - strict: bool
            If true only keeps nouns
    """


    # Removing <code>some code</code>
    clean_txt = remove_code(text)

    # Removing HTML tags
    soup = BeautifulSoup(clean_txt, features='html.parser')
    clean_txt = soup.get_text()

    # Removing new line character: \n
    clean_txt = clean_txt.replace('\n', ' ')

    # Removing unicode characters
    clean_txt = clean_txt.encode("ascii", "ignore").decode()

    # Removing digits
    clean_txt = ''.join(char for char in clean_txt if not char.isdigit())

```

Pre-processing

Création de fonction de nettoyage/tokenisation de texte

```
32
33      # Replacing 'c ++' and 'c #' for 'c++' and 'c#' and others
34      clean_txt = clean_txt.replace('c ++', 'c++')
35      clean_txt = clean_txt.replace('c #', 'c#')
36      clean_txt = clean_txt.replace('C ++', 'c++')
37      clean_txt = clean_txt.replace('C #', 'c#')
38      clean_txt = clean_txt.replace('C#', 'c#')
39      clean_txt = clean_txt.replace('C ++', 'c++')
40
41      # Adding special case rule
42      special_case = [{ORTH: "c#"}]
43      nlp.tokenizer.add_special_case("c#", special_case)
44      special_case = [{ORTH: ".net"}]
45      nlp.tokenizer.add_special_case(".net", special_case)
46      special_case = [{ORTH: "objective-c"}]
47      nlp.tokenizer.add_special_case("objective-c", special_case)
48      special_case = [{ORTH: "asp.net"}]
49      nlp.tokenizer.add_special_case("asp.net", special_case)
50      special_case = [{ORTH: "node.js"}]
51      nlp.tokenizer.add_special_case("node.js", special_case)
52      special_case = [{ORTH: "ruby-on-rails"}]
53      nlp.tokenizer.add_special_case("ruby-on-rails", special_case)
54      special_case = [{ORTH: "sql-server"}]
55      nlp.tokenizer.add_special_case("sql-server", special_case)
56      special_case = [{ORTH: "unit-testing"}]
57      nlp.tokenizer.add_special_case("unit-testing", special_case)
```

3 Attention aux caractères propres aux noms des langages

C + + → C++

Pre-processing

Création de fonction de nettoyage/tokenisation de texte

```
    # Tokenize with spacy
    doc = nlp(clean_txt)

    # Tokenize properties
    if strict == True:
        tokens = [token.lemma_.lower() for token in doc
                  if token.pos_ in ['NOUN', 'PROPN', 'VERB'] and
                      (not (token.is_stop or
                            token.is_punct or
                            token.is_space
                            )
                       )
                  ]
    else:
        tokens = [token.lemma_.lower() for token in doc
                  if not (token.is_stop or
                          token.is_punct or
                          token.is_space
                          )
                  ]

    clean_txt = ' '.join(tokens)

    # Ask if return text or tokens
    if tokenize == True:
        result = tokens
    else:
        result = clean_txt

    # Option for list of entities in output
    if 'ent' in kwargs:
        result = {'output':result, 'ents': doc.ents}

    return result
```

4 Tokenisation avec spaCy

5 Filtrer : noms
noms propres
verbes

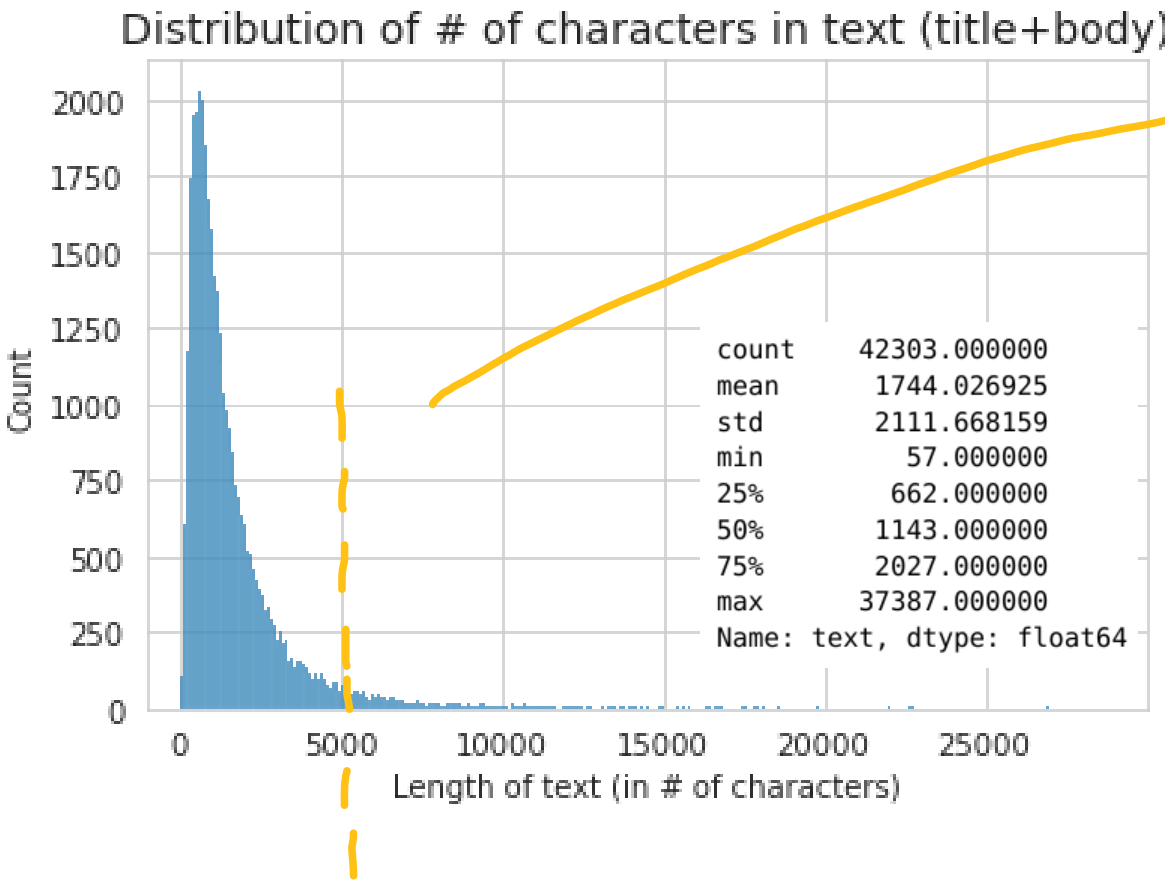
6 Résultat → texte ou tokens

Pre-processing

Voir example dans notebook

Pre-processing

Derniers détails préparation texte



1. Concaténer
2. filtrer (≤ 5000 car.)

```
Shape of dataset before filter: (42303, 45)
Shape of dataset after filter: (40166, 45)
Number of deleted rows: 2137
```

3. Appliquer ma fonction nettoyage

```
Total time to process text: 1233.7 seconds. (= 20.56 minutes)
```

↓

EXPORTER DONNÉES !

3

Modélisation + choix de modèle

Modélisation

Notre problème :

texte

(raw HTML)

associer

/prédire

étiquettes : tags

C# python linux

sql pandas

↓

plusieurs sont possibles

pour 1 seul input

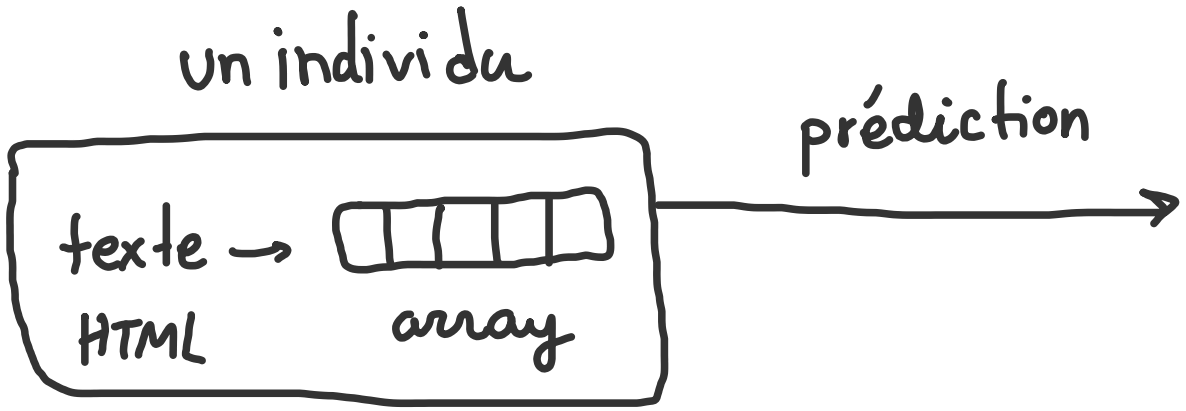
CLASSIFICATION

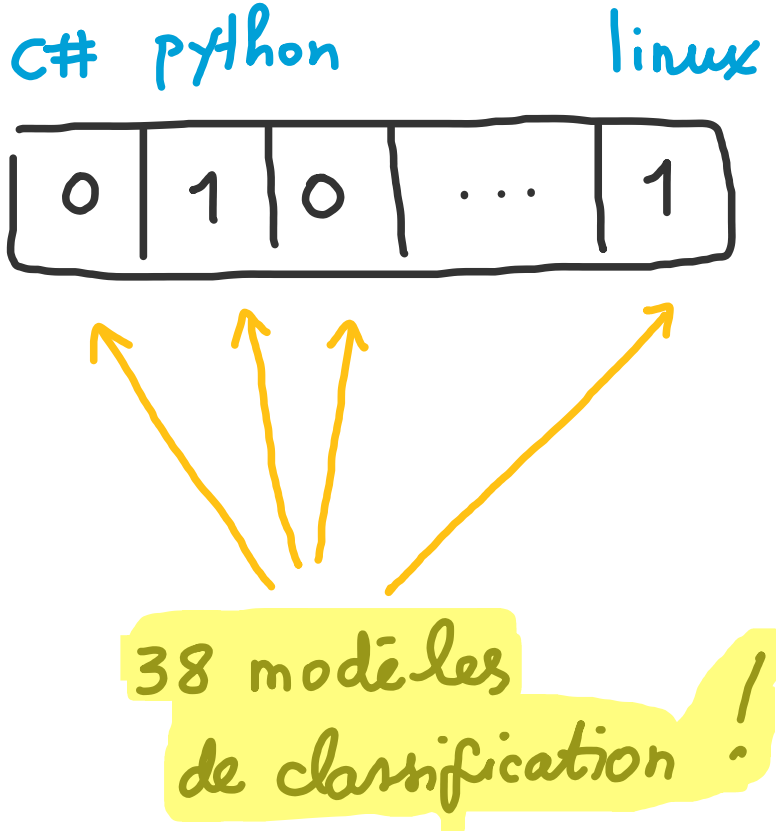
MULTILABEL

⇐

Modélisation

Notre stratégie : scikit learn OneVsRestClassifier

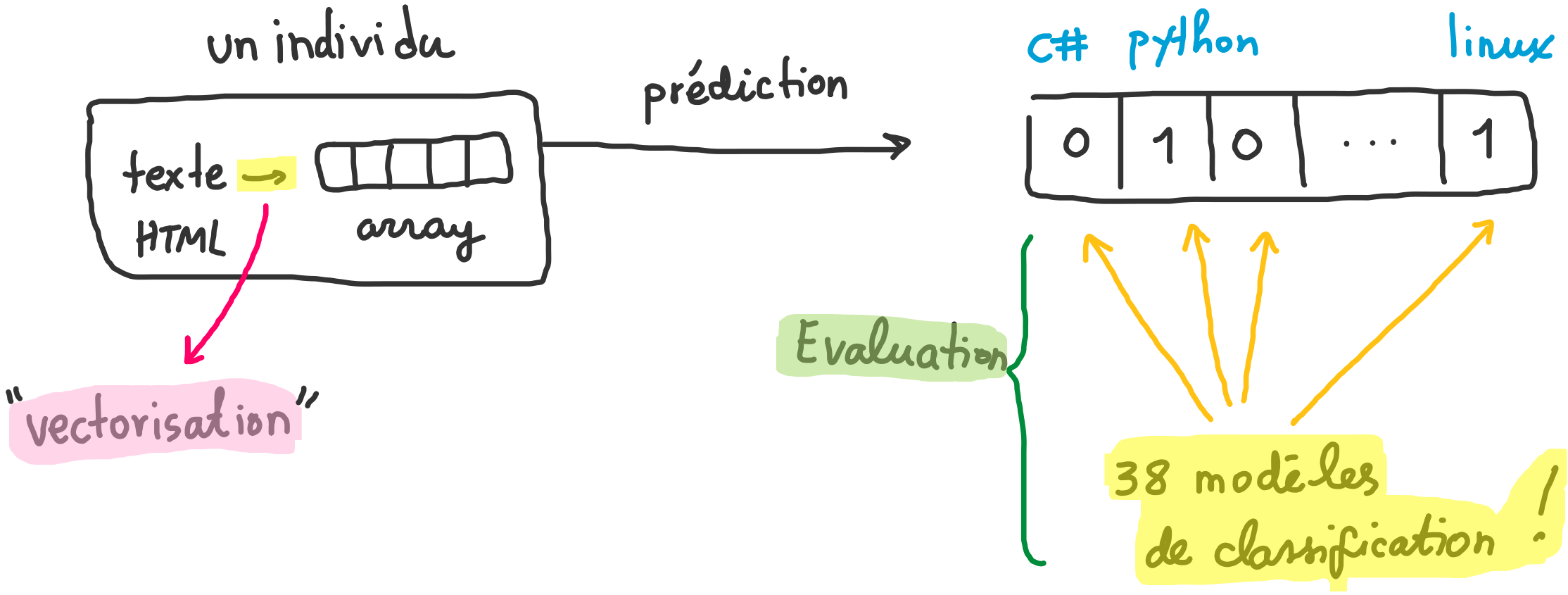




CLASSIFICATION MULTILABEL

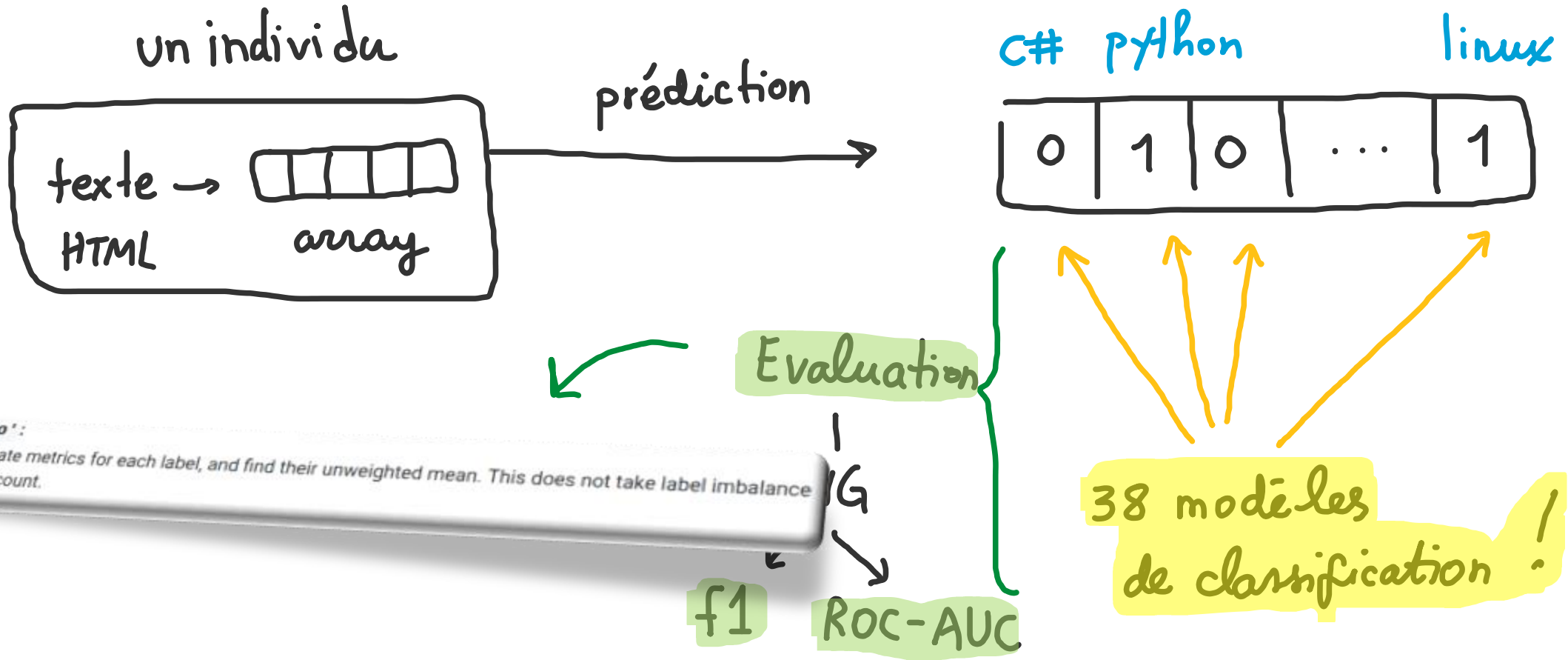
Modélisation

Notre stratégie : scikit learn OneVsRestClassifier

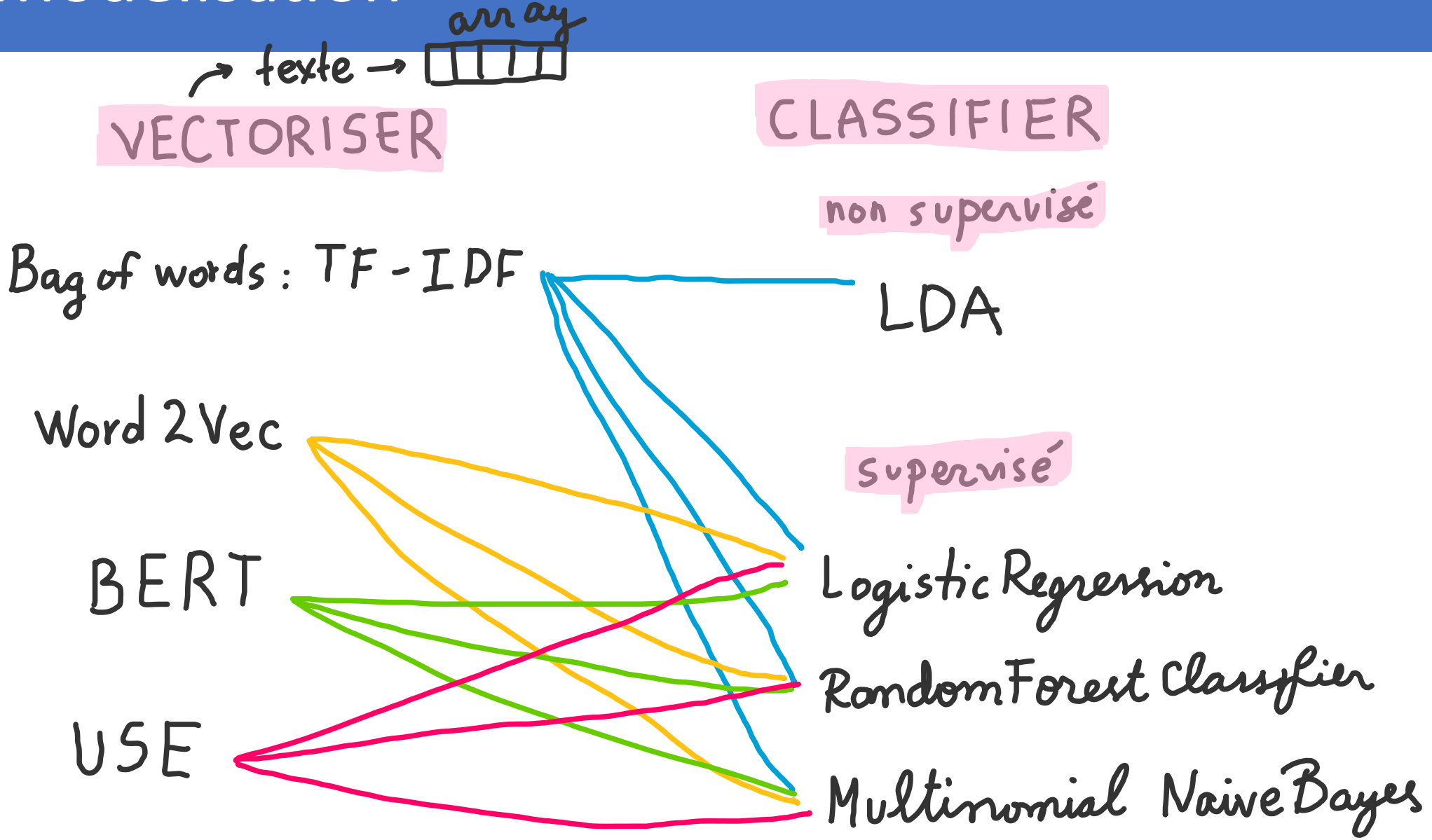


Modélisation

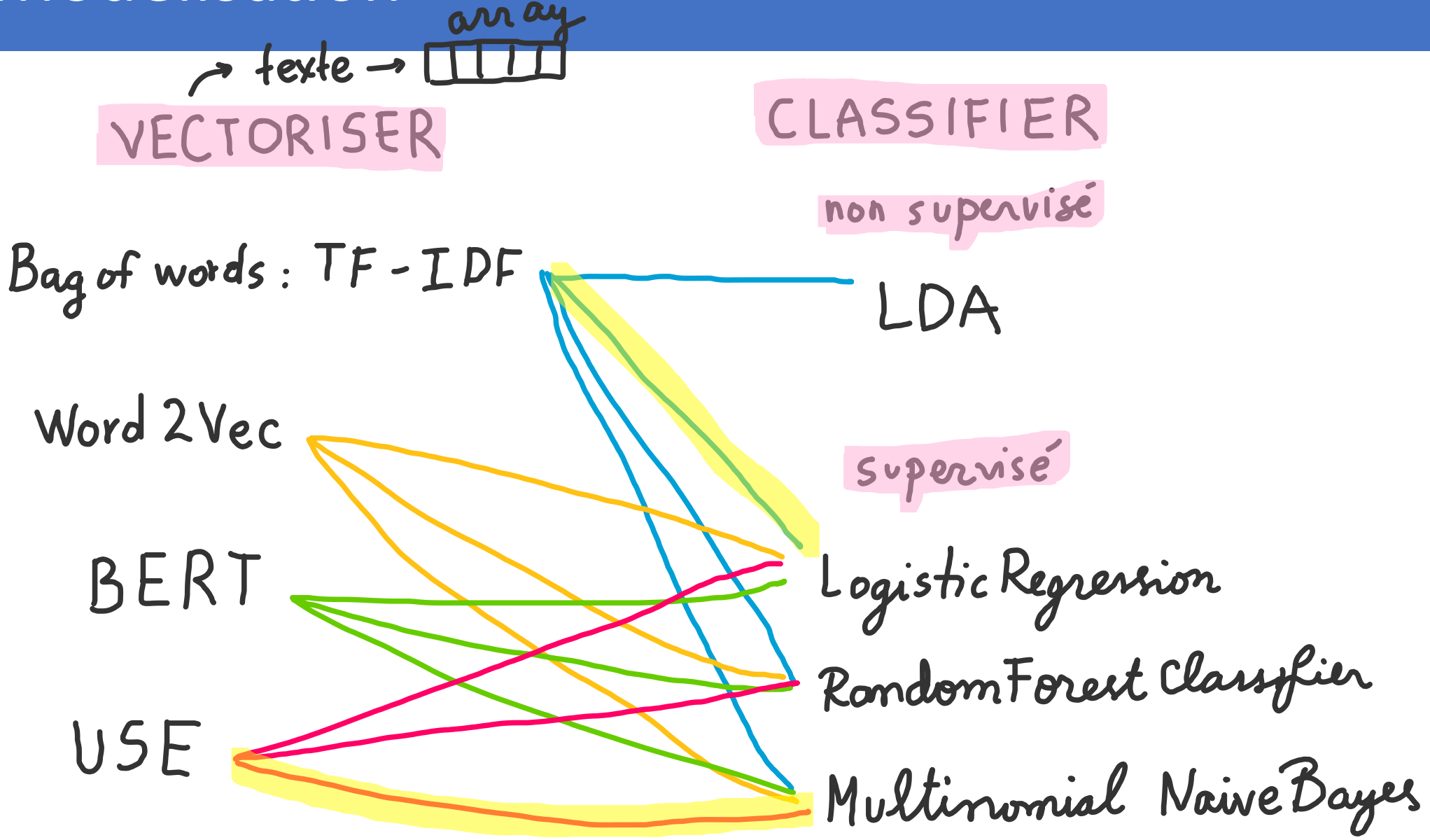
Notre stratégie : scikit learn OneVsRestClassifier



Modélisation



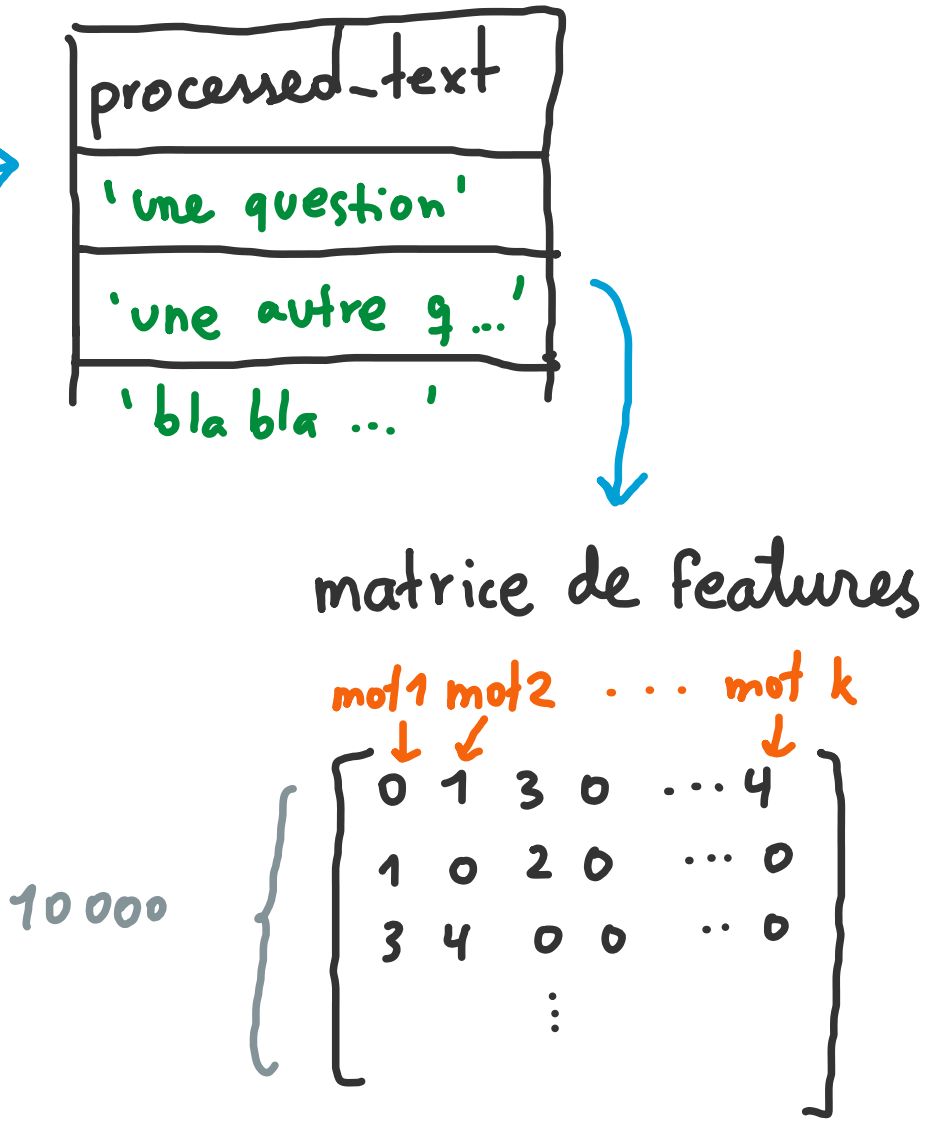
Modélisation



Modélisation (supervisée)

Bag of words: TF-IDF

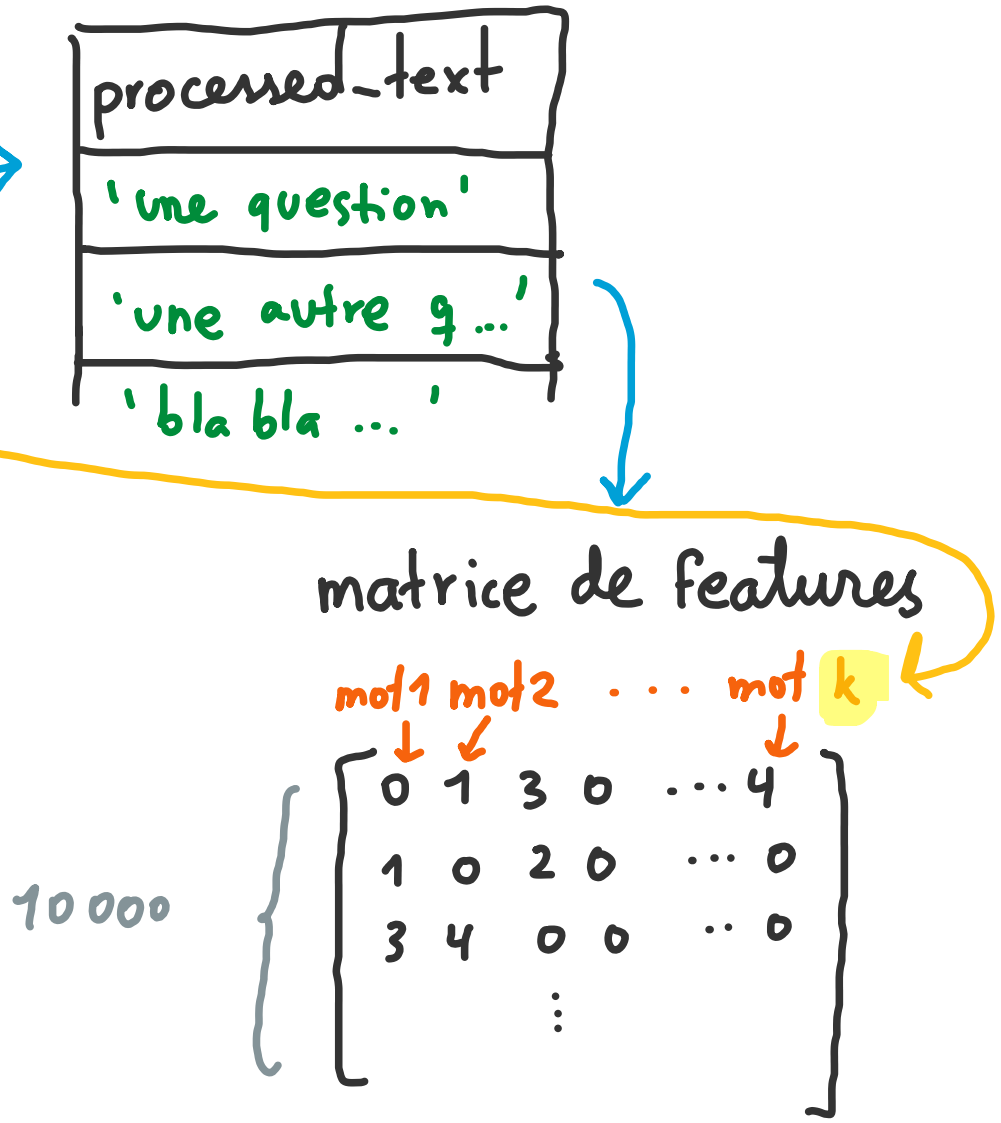
```python
# Instantiate the vectorizer
tfidf = TfidfVectorizer(max_df=0.75,
                        min_df=0.0015
                        )

# Vectorize the processed text
X = tfidf.fit_transform(df.processed_text)

# Defining target vector
columns_tags = ['tag_'+tag for tag in tag_list]
y = df[columns_tags]

# Splitting into train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.33, random_state=5
)
```



Modélisation (supervisée)

Bag of words: TF-IDF

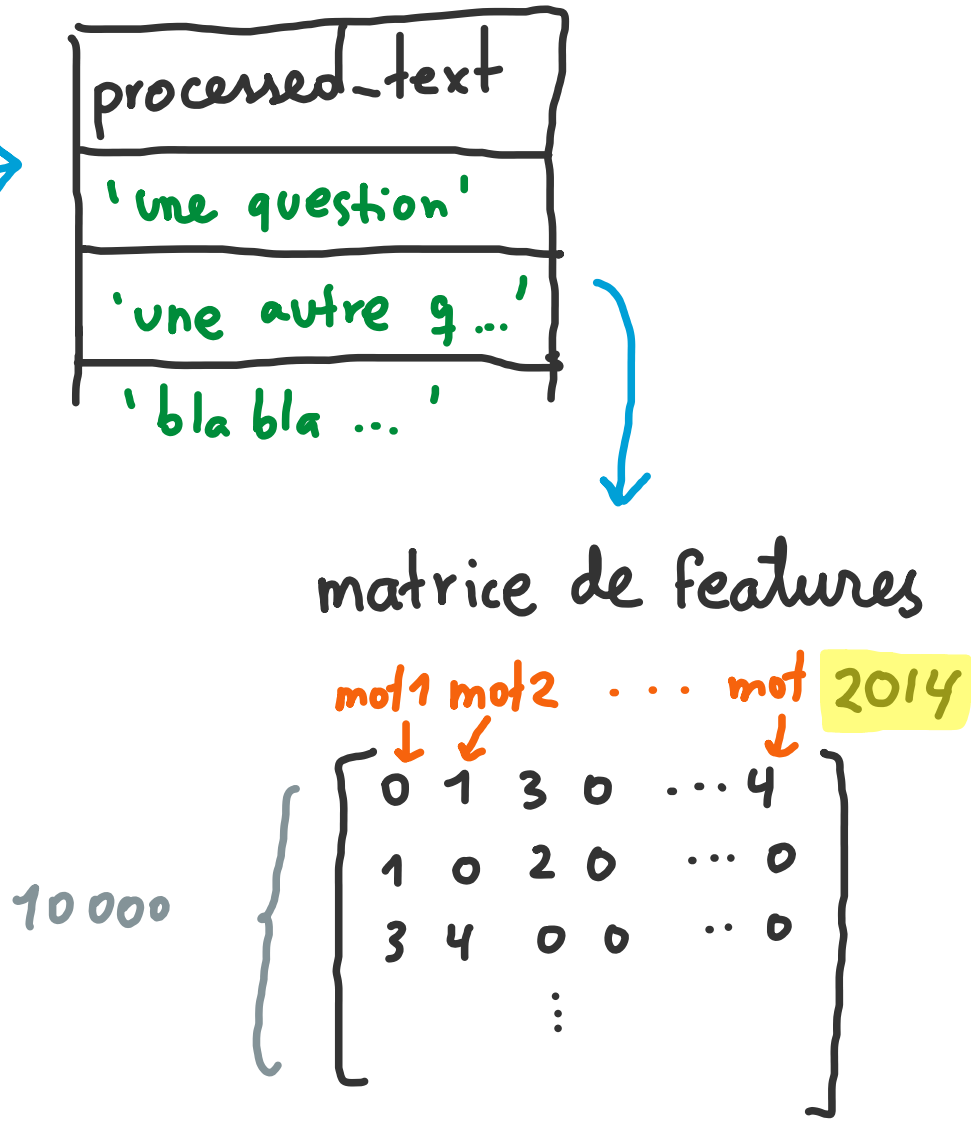
```python
# Instantiate the vectorizer
tfidf = TfidfVectorizer(max_df=0.75,
                        min_df=0.0015
                        )

# Vectorize the processed text
X = tfidf.fit_transform(df.processed_text)

# Defining target vector
columns_tags = ['tag_'+tag for tag in tag_list]
y = df[columns_tags]

# Splitting into train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.33, random_state=5
)
```



Modélisation (supervisée)

Bag of words: TF-IDF

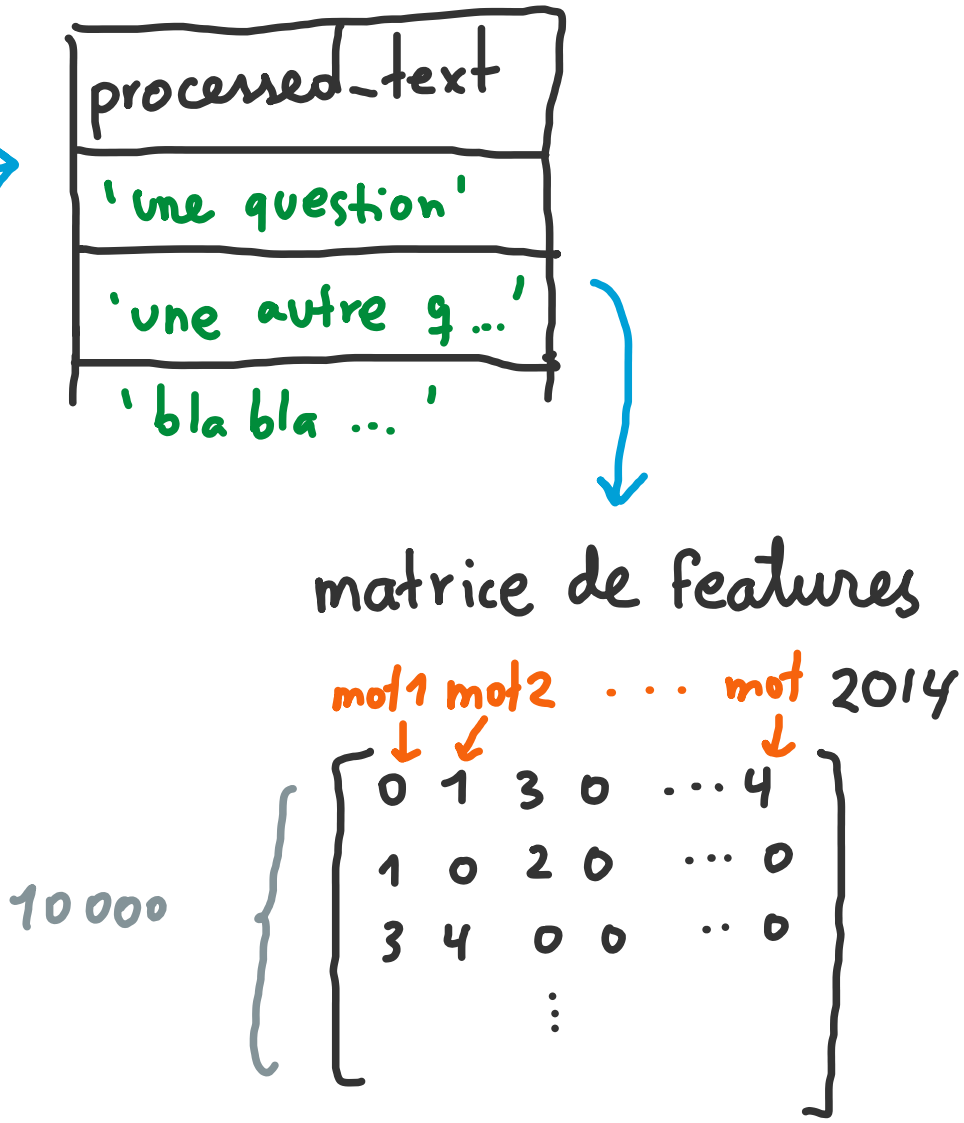
```python
# Instantiate the vectorizer
tfidf = TfidfVectorizer(max_df=0.75,
                        min_df=0.0015
                        )

# Vectorize the processed text
X = tfidf.fit_transform(df.processed_text)

# Defining target vector
columns_tags = ['tag_'+tag for tag in tag_list]
y = df[columns_tags]

# Splitting into train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.33, random_state=5
)
```



Modélisation (supervisée)

Bag of words: TF-IDF → term frequency

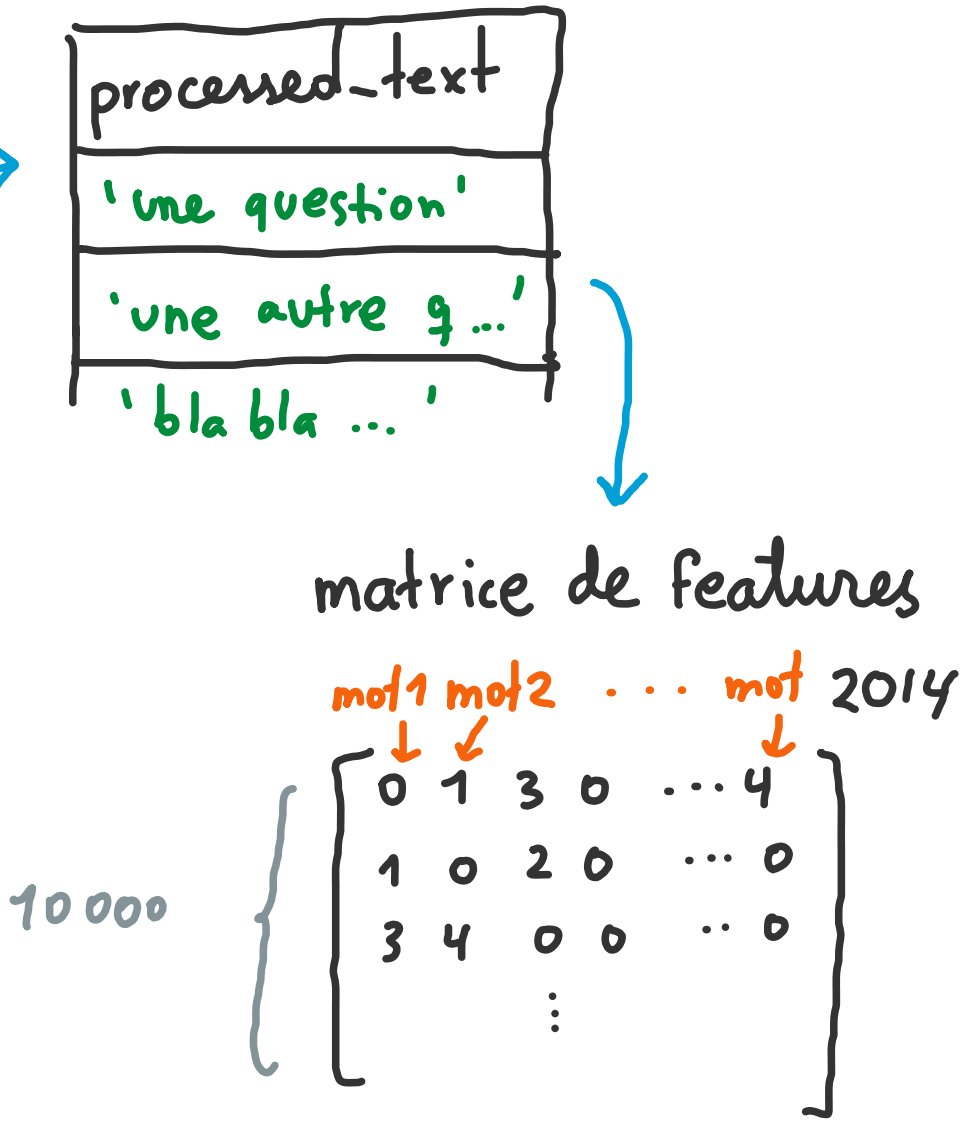
```python
# Instantiate the vectorizer
tfidf = TfidfVectorizer(max_df=0.75,
                        min_df=0.0015
                        )

# Vectorize the processed text
X = tfidf.fit_transform(df.processed_text)

# Defining target vector
columns_tags = ['tag_'+tag for tag in tag_list]
y = df[columns_tags]

# Splitting into train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.33, random_state=5
)
```



Modélisation (supervisée)

Bag of words: TF-IDF → term frequency

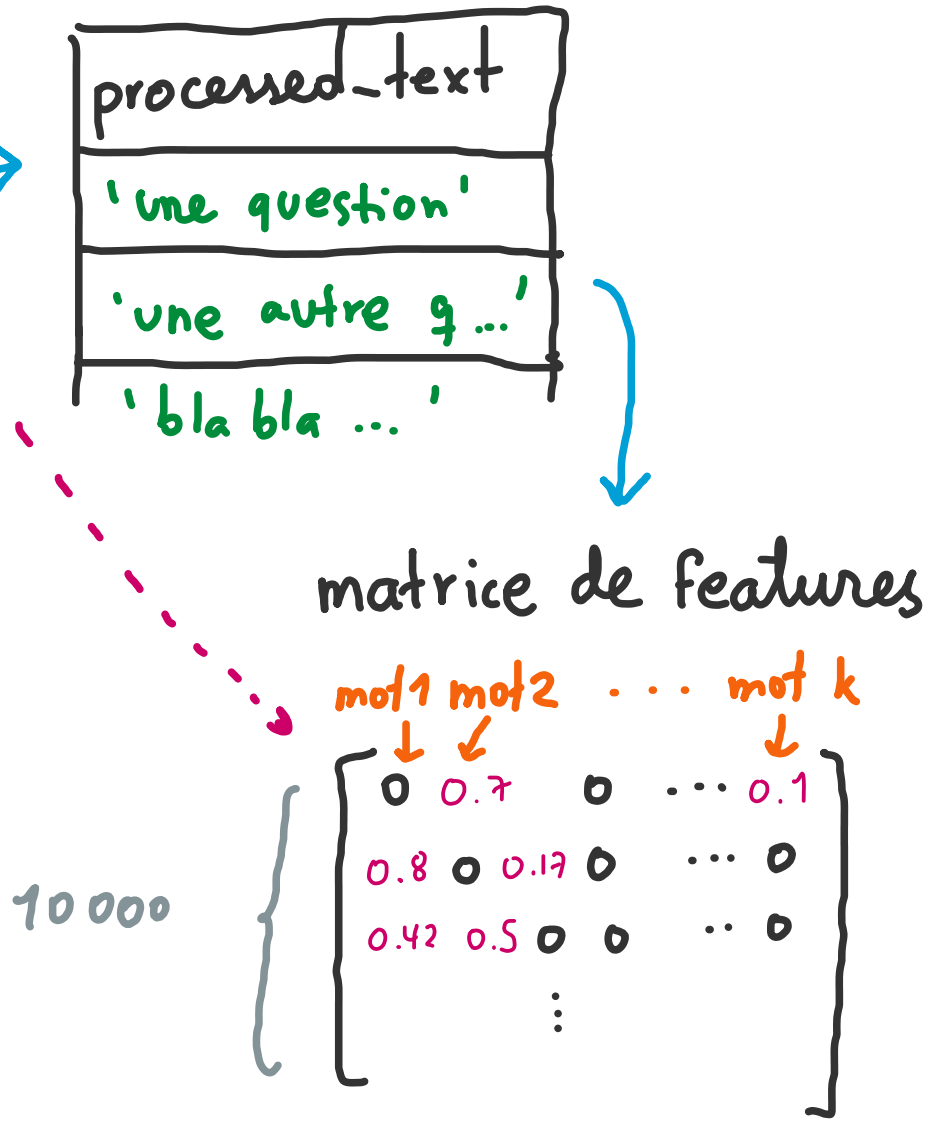
```python
# Instantiate the vectorizer
tfidf = TfidfVectorizer(max_df=0.75,
                        min_df=0.0015
                        )

# Vectorize the processed text
X = tfidf.fit_transform(df.processed_text)

# Defining target vector
columns_tags = ['tag_'+tag for tag in tag_list]
y = df[columns_tags]

# Splitting into train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.33, random_state=5
)
```



Modélisation (supervisée)

Bag of words: TF-IDF → term frequency X Inverse doc. frequency

```python
# Instantiate the vectorizer
tfidf = TfidfVectorizer(max_df=0.75,
                        min_df=0.0015
                        )

# Vectorize the processed text
X = tfidf.fit_transform(df.processed_text)

# Defining target vector
columns_tags = ['tag_'+tag for tag in tag_list]
y = df[columns_tags]

# Splitting into train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.33, random_state=5
)
```



Modélisation (supervisée)

Bag of words: TF-IDF + **LogisticRegression**

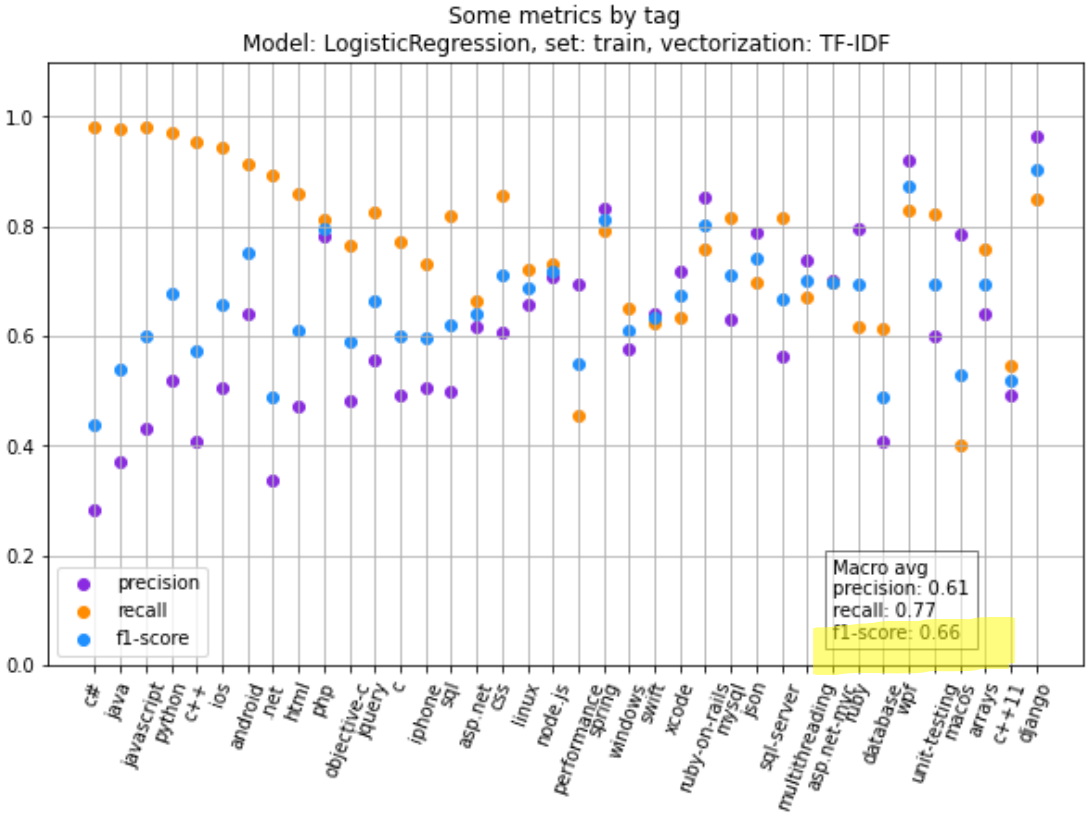
```python
# Hyperparameters for logistic regression
parameters = {
    'estimator__penalty' : ['l1', 'l2'],
    'estimator__tol' : [1e-6, 1e-5, 1e-4, 1e-3],
    'estimator__C' : [0.01,0.05 ,0.1, 0.5, 0.7, 1],
    'estimator__fit_intercept' : [True, False],
    'estimator__solver' : ['liblinear','sag', 'saga']
}

# Instantiating OneVsRest Classifier
cl = OneVsRestClassifier(LogisticRegression())

# Random search for best hyperparameters
random_search = RandomizedSearchCV(
    estimator = cl,
    param_distributions= parameters,
    n_iter=40,
    scoring='roc_auc_ovr',
    random_state=5,
    n_jobs=-1
)

# Perform random search
random_search.fit(X_train, y_train)

# Instantiating BEST OneVsRest Classifier
cl = random_search.best_estimator_

# Fitting classifier
start = time.time()
cl.fit(X_train,y_train)
finish = time.time()
fittime = finish - start
print(f'fit time: {fittime} secs.')

```

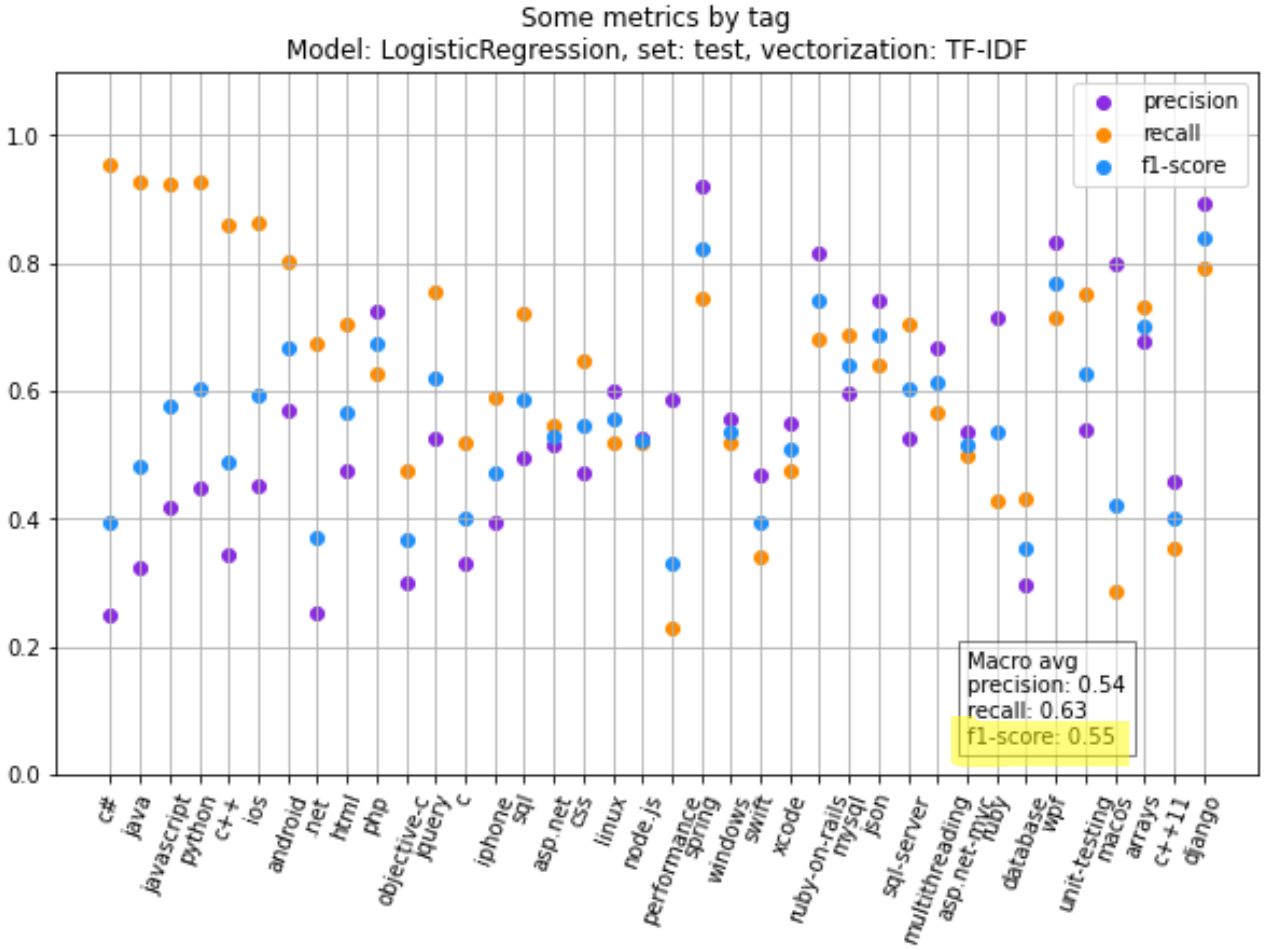
validation croisée avec Randomized Search

évaluation du temps aussi

Modélisation (supervisée)

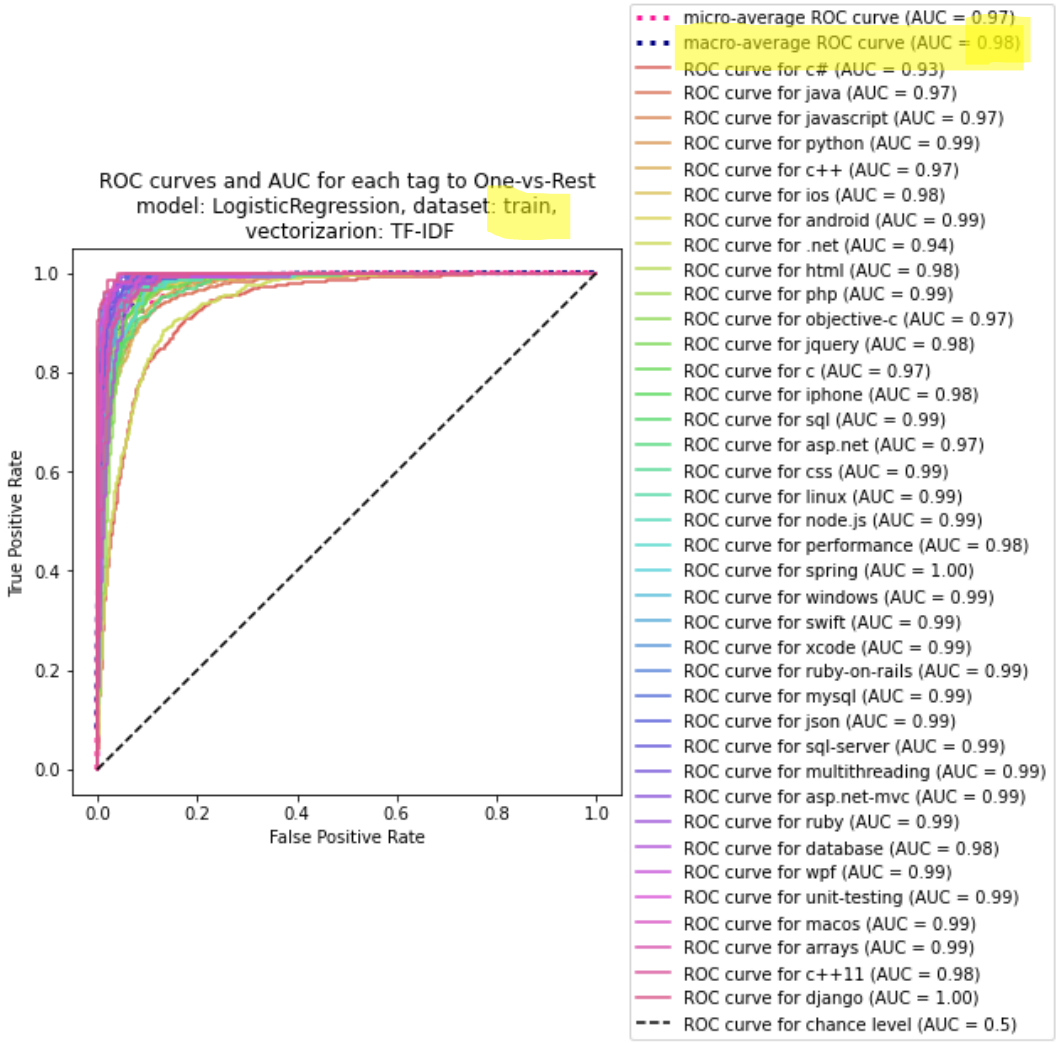
Bag of words: TF-IDF + LogisticRegression **--> Évaluation**





Modélisation (supervisée)

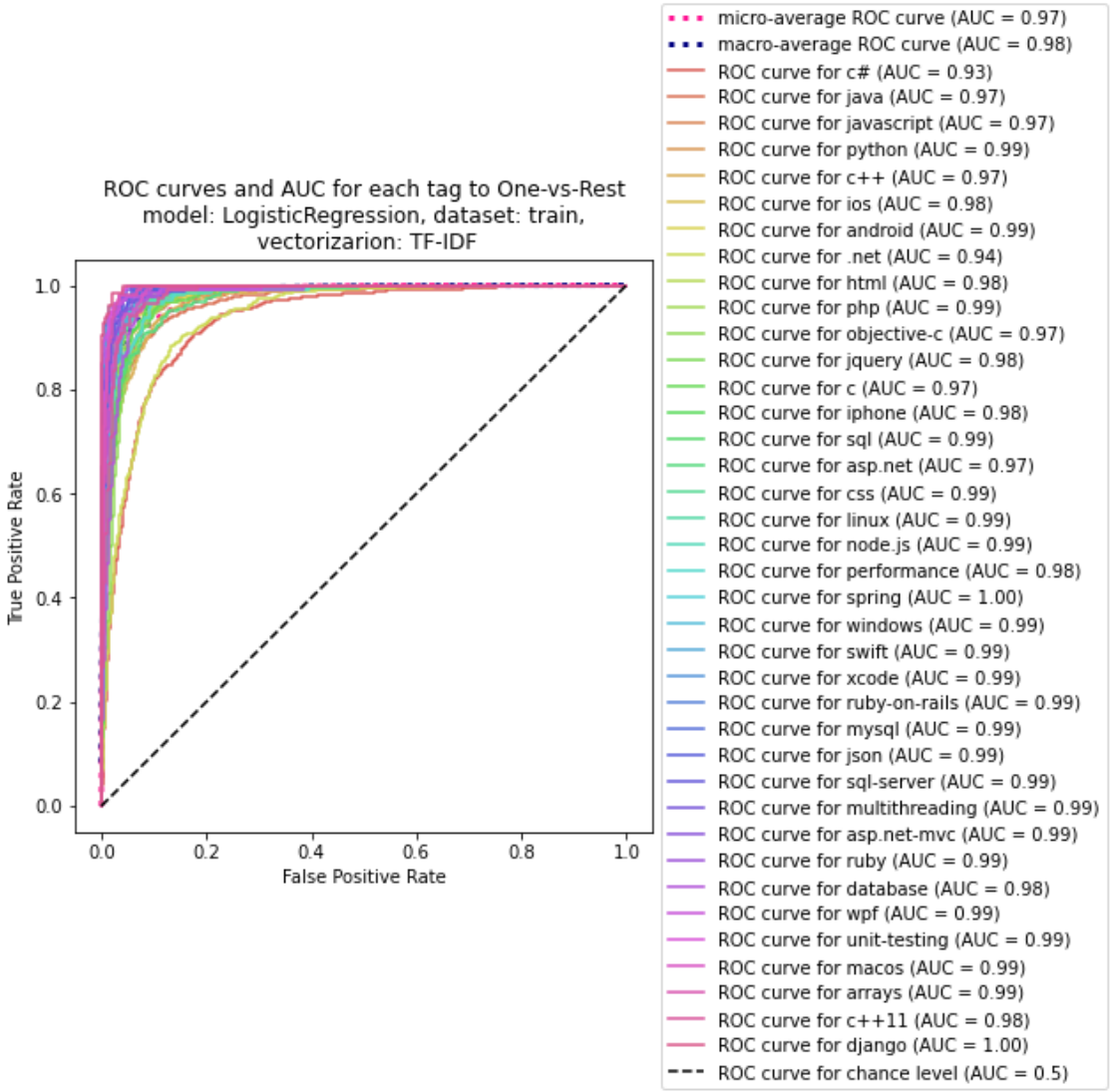
Bag of words: TF-IDF + LogisticRegression **--> Évaluation**

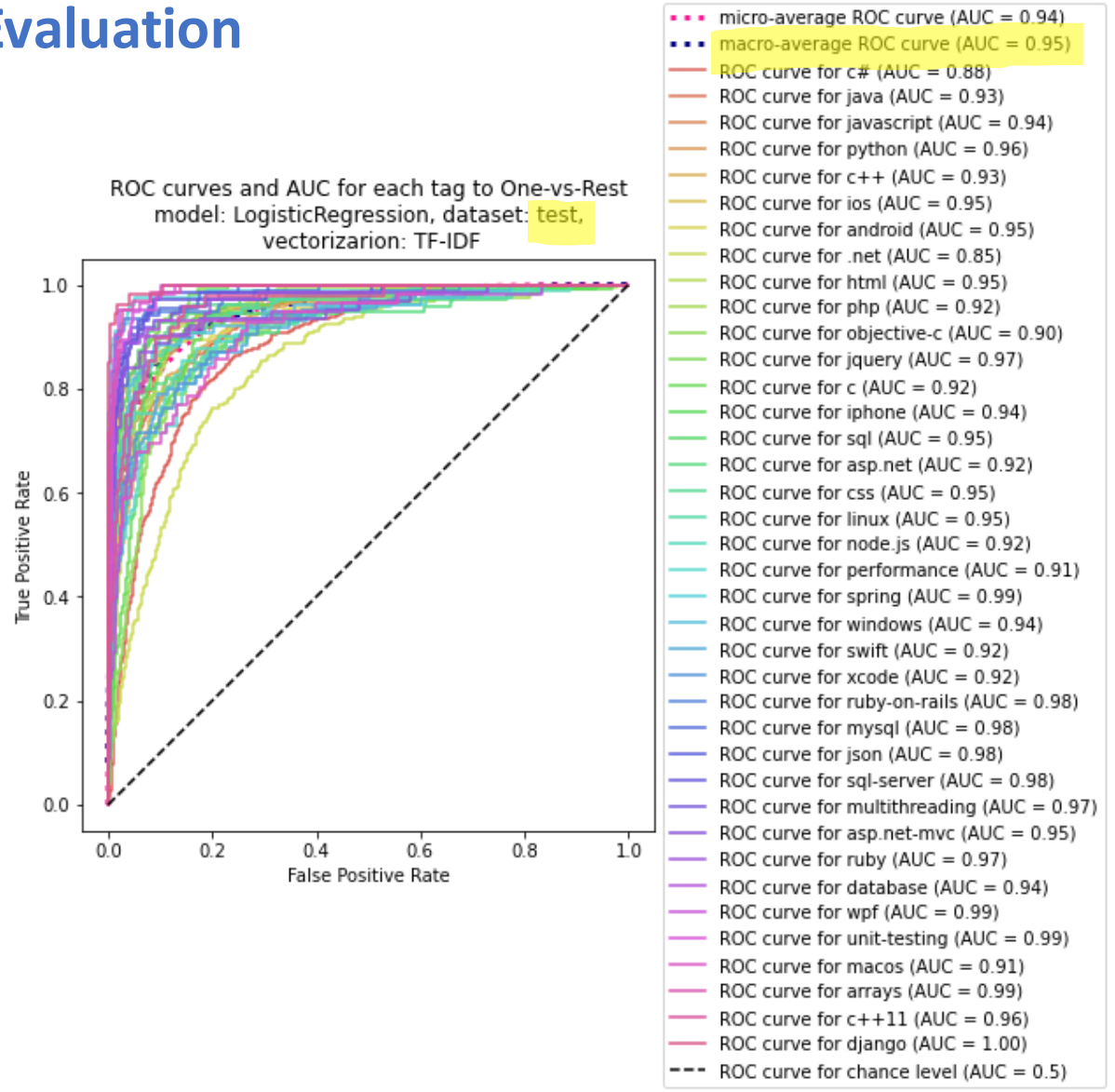


ROC pour chacun des classifieurs

Modélisation (supervisée) 3

Bag of words: TF-IDF + LogisticRegression --> Évaluation





Modélisation (supervisée)

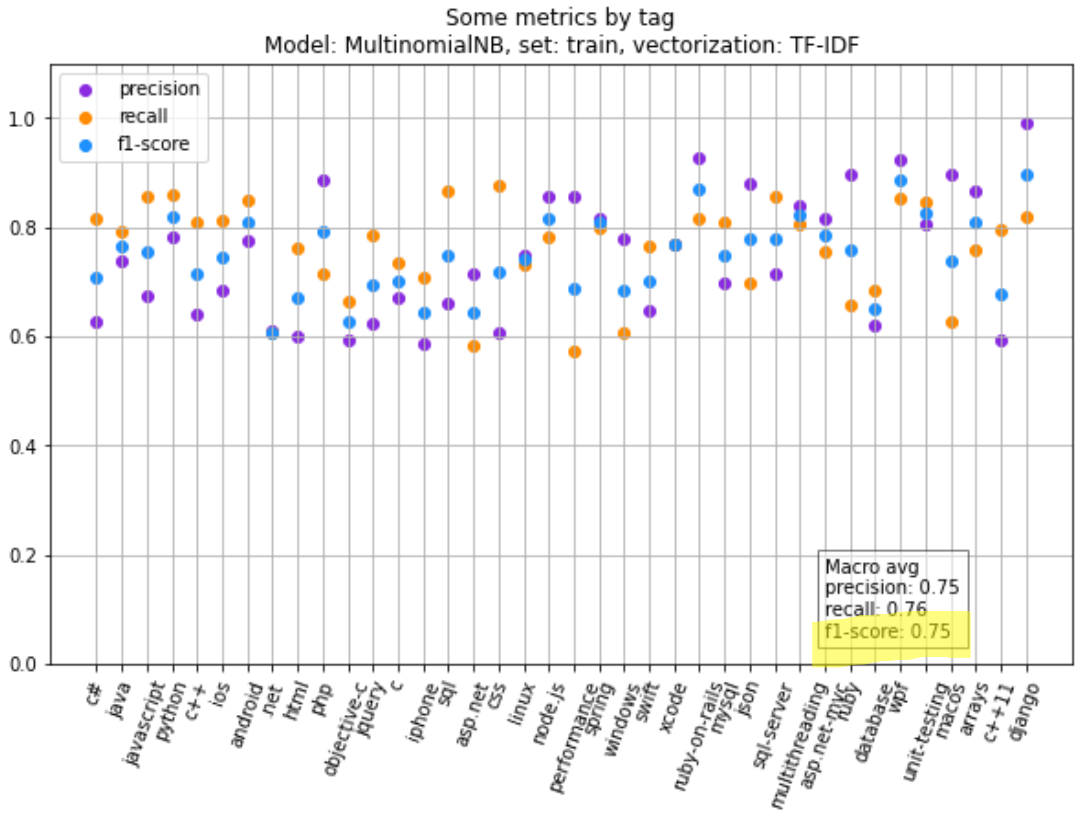
Bag of words: TF-IDF + **Naïve-Bayes classifier**

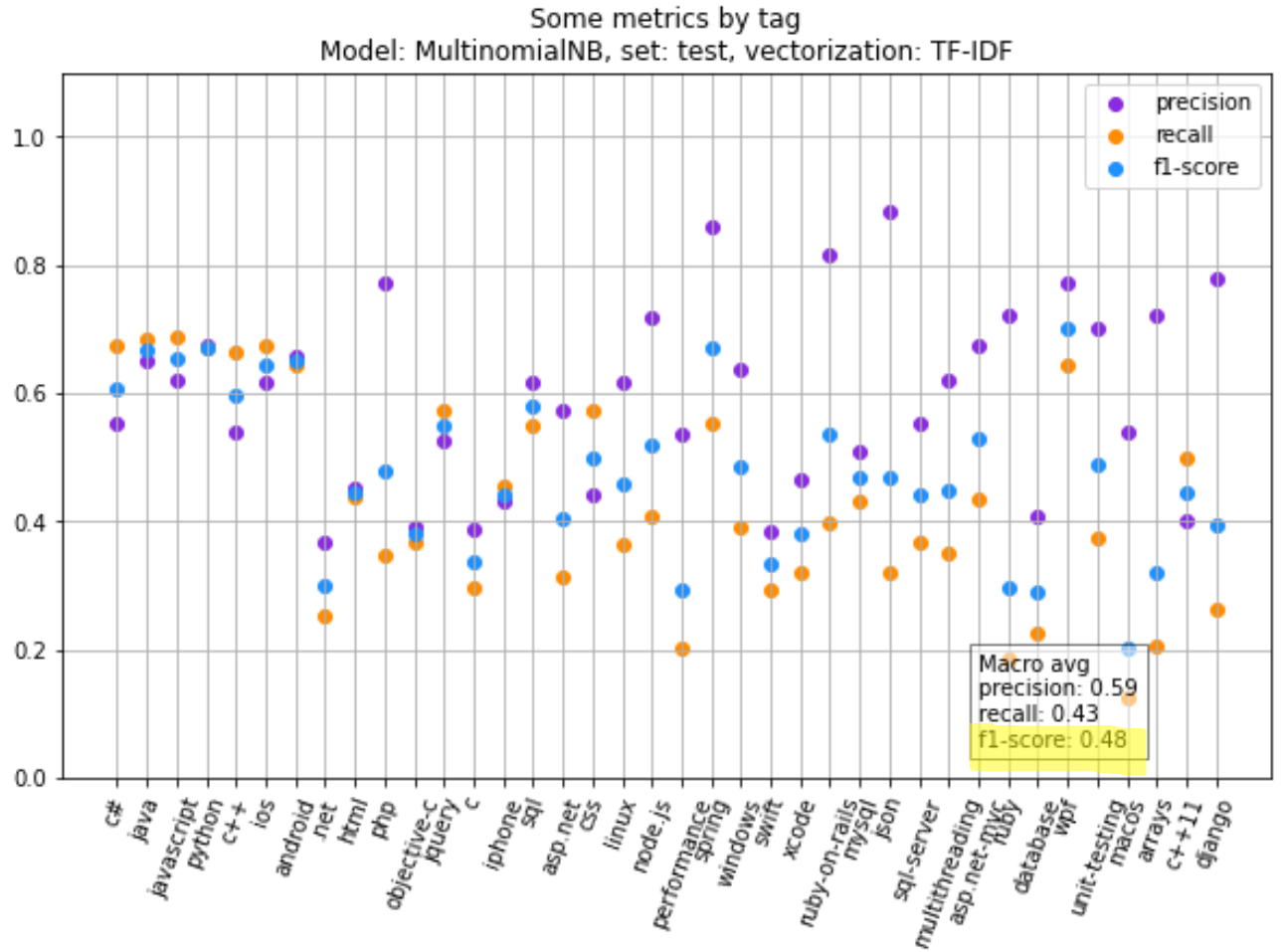
```python
parameters = {
    'estimator__alpha' : [0.05, 0.2, 0.5, 1],
    'estimator__fit_prior' : [True, False]
}

# Instantiating OneVsRest Classifier
cl = OneVsRestClassifier(MultinomialNB())

# Grid search for best hyperparameters
grid_search = GridSearchCV(
    estimator = cl,
    param_grid= parameters,
    scoring='roc_auc_ovr',
    verbose=2,
    n_jobs=-1
)
```

Validation croisée pour trouver hyperparamètres adaptés

Modélisation (supervisée)

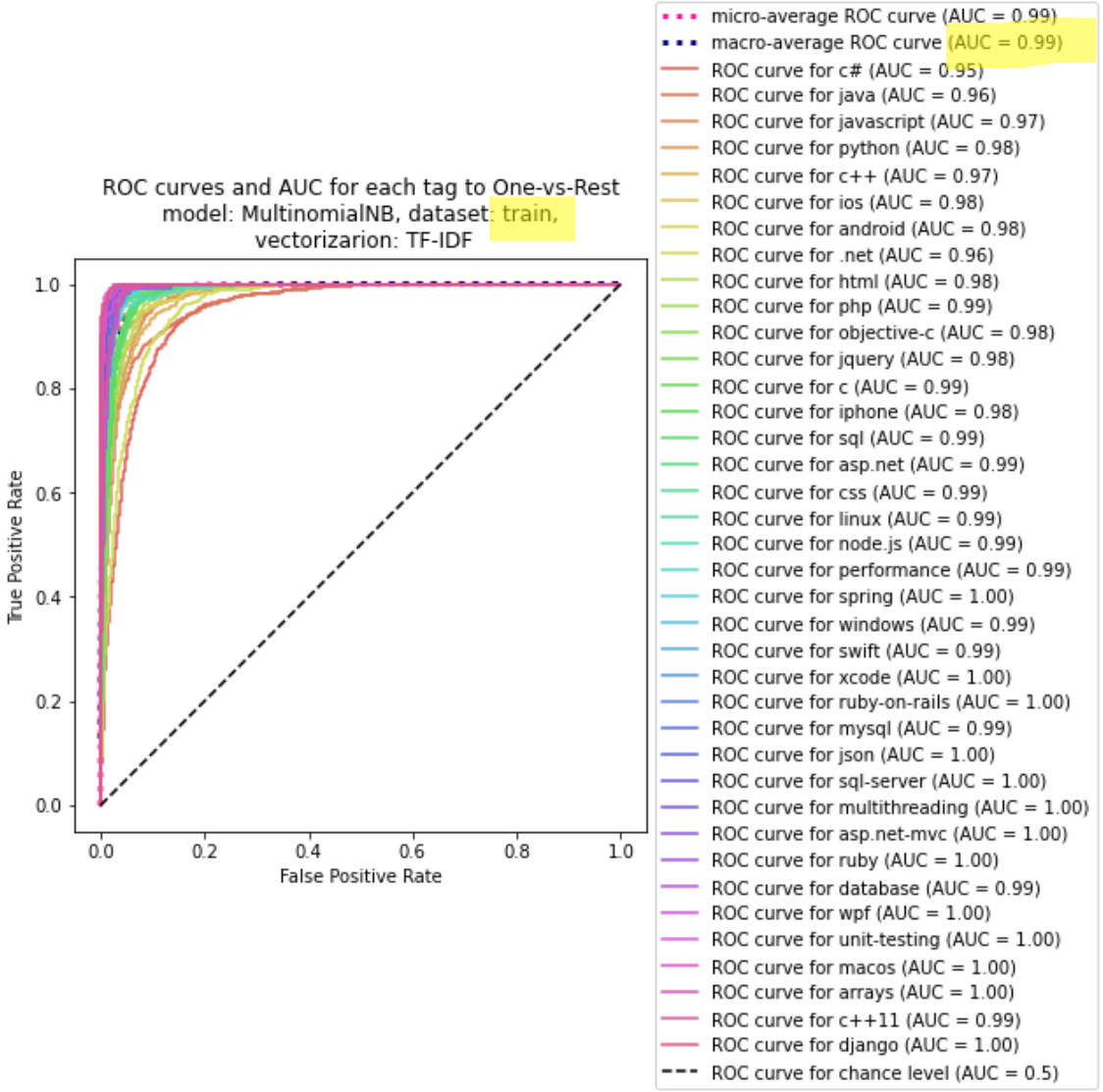
Bag of words: TF-IDF + **Naïve-Bayes classifier --> Évaluation**

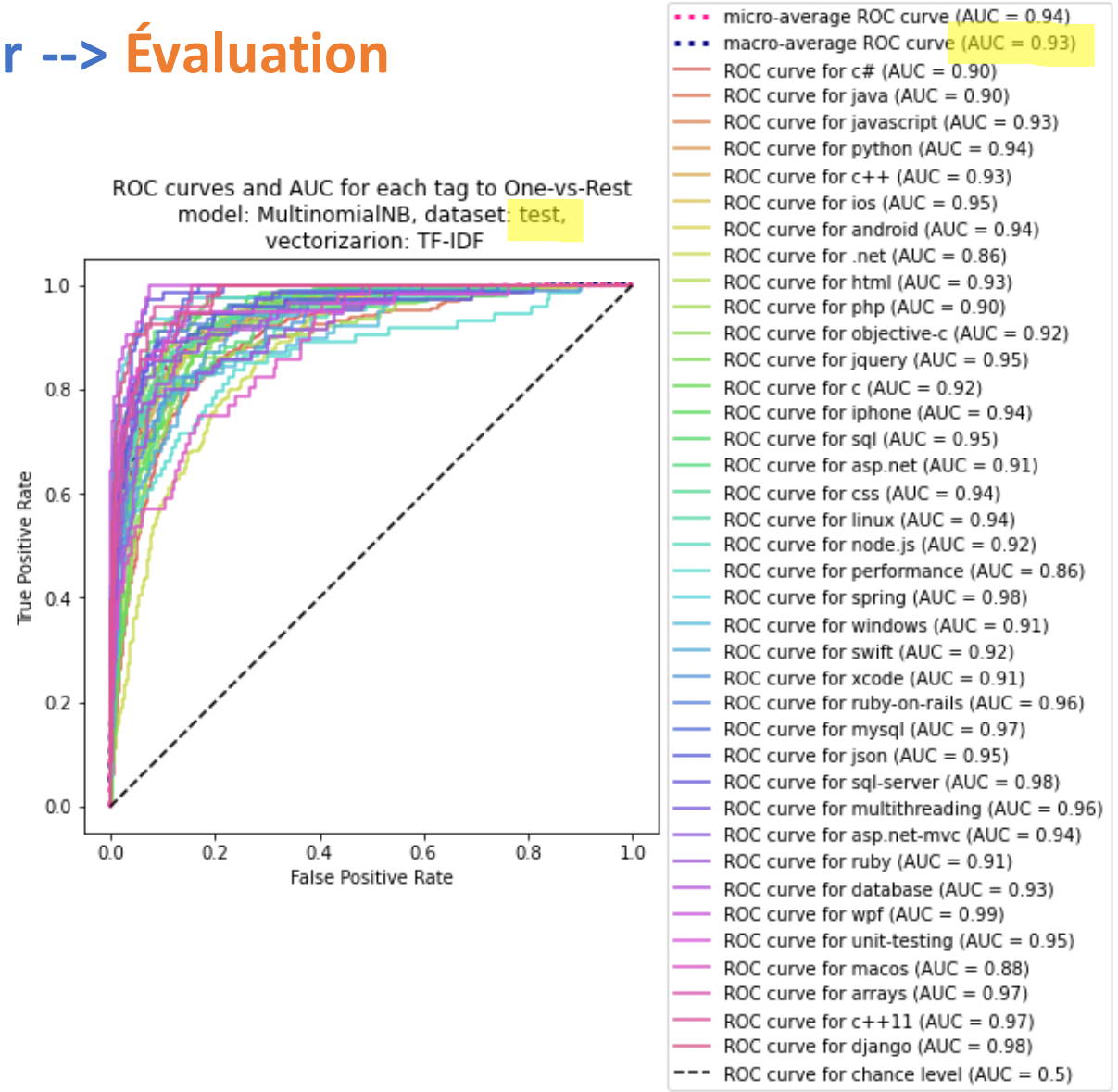




Modélisation (supervisée)

Bag of words: TF-IDF + **Naïve-Bayes classifier** --> **Évaluation**





Modélisation (supervisée)

Bag of words: TF-IDF + **RandomForestClassifier**

```
# Hyperparameter tuning for RandomForest

# parameters = {
#     'estimator__n_estimators' : [50, 100, 150, 200, 250],
#     'estimator__max_depth' : [int(x) for x in np.linspace(10, 110, num = 11)],
#     'estimator__min_samples_split' : [2,5,10],
#     'estimator__min_samples_leaf' : [1,2,4],
#     'estimator__bootstrap' : [True, False],
#     'estimator__max_features' : ['sqrt', 'log2', None]
# }

# # Instantiating OneVsRest Classifier
# cl = OneVsRestClassifier(RandomForestClassifier())

# # Random search for best hyperparameters
# random_search = RandomizedSearchCV(
#     estimator = cl,
#     param_distributions= parameters,
#     n_iter=30,
#     scoring='roc_auc_ovr',
#     verbose=2,
#     random_state=5,
#     n_jobs=-1
# )
```

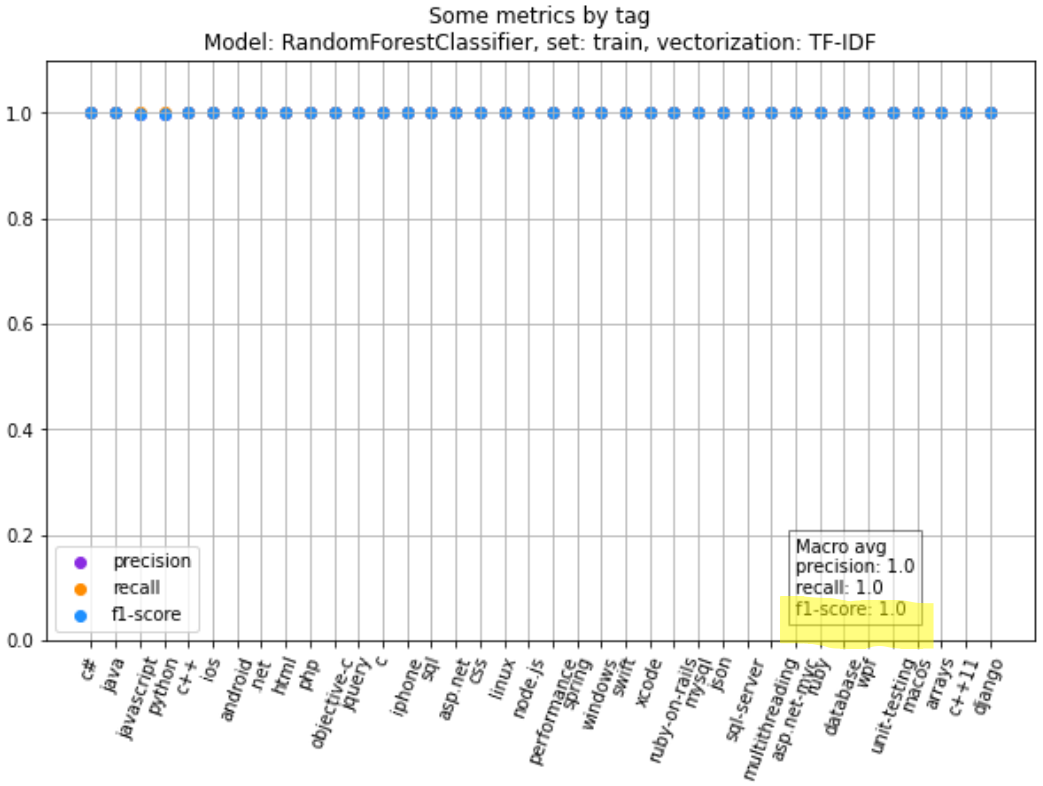
100
None
2
2
True
sqrt

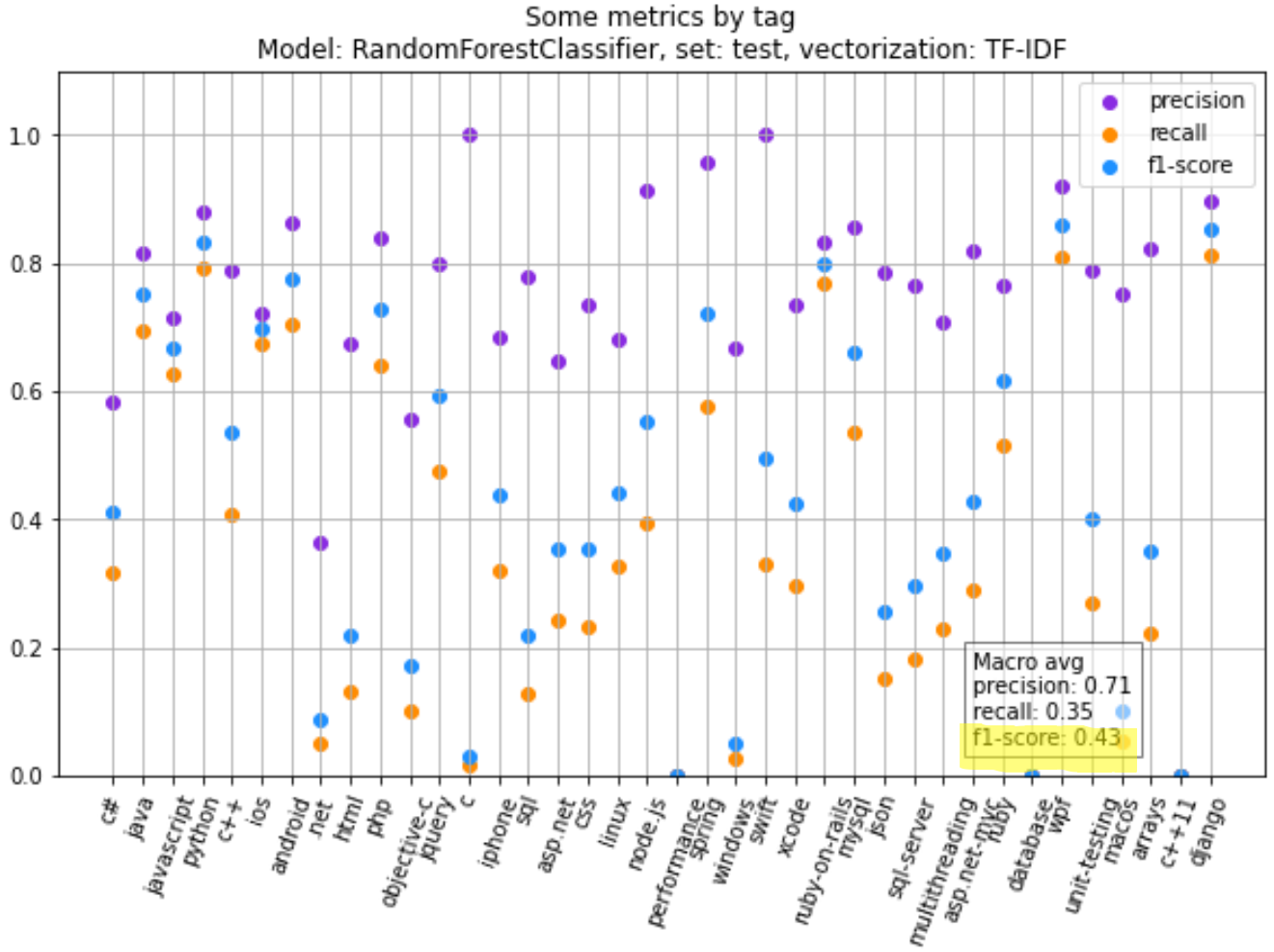
Nous ressources de calcul **n'ont pas permis** de faire une validation croisée pour RandomForest.

Solution: entraînement (très long) avec paramètres par défaut.

Modélisation (supervisée) 3

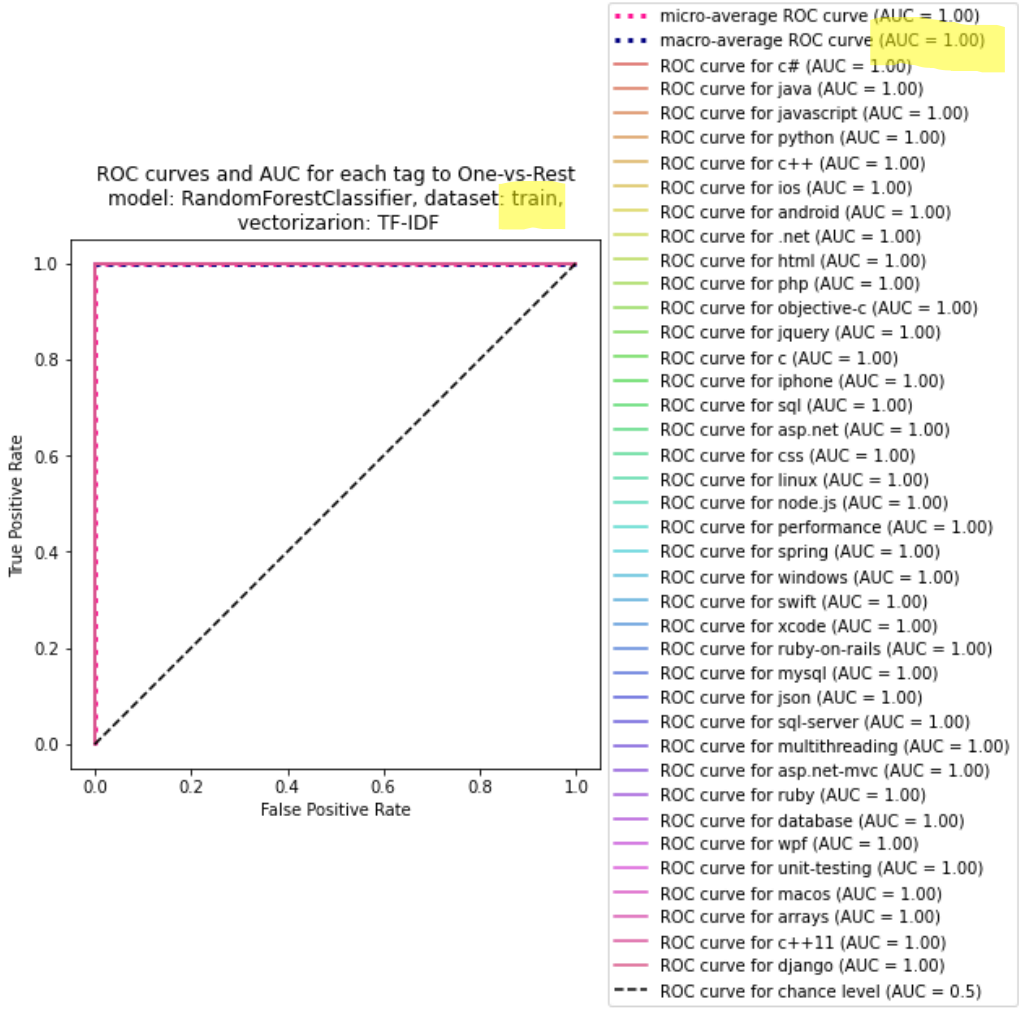
Bag of words: TF-IDF + **RandomForestClassifier --> Évaluation**

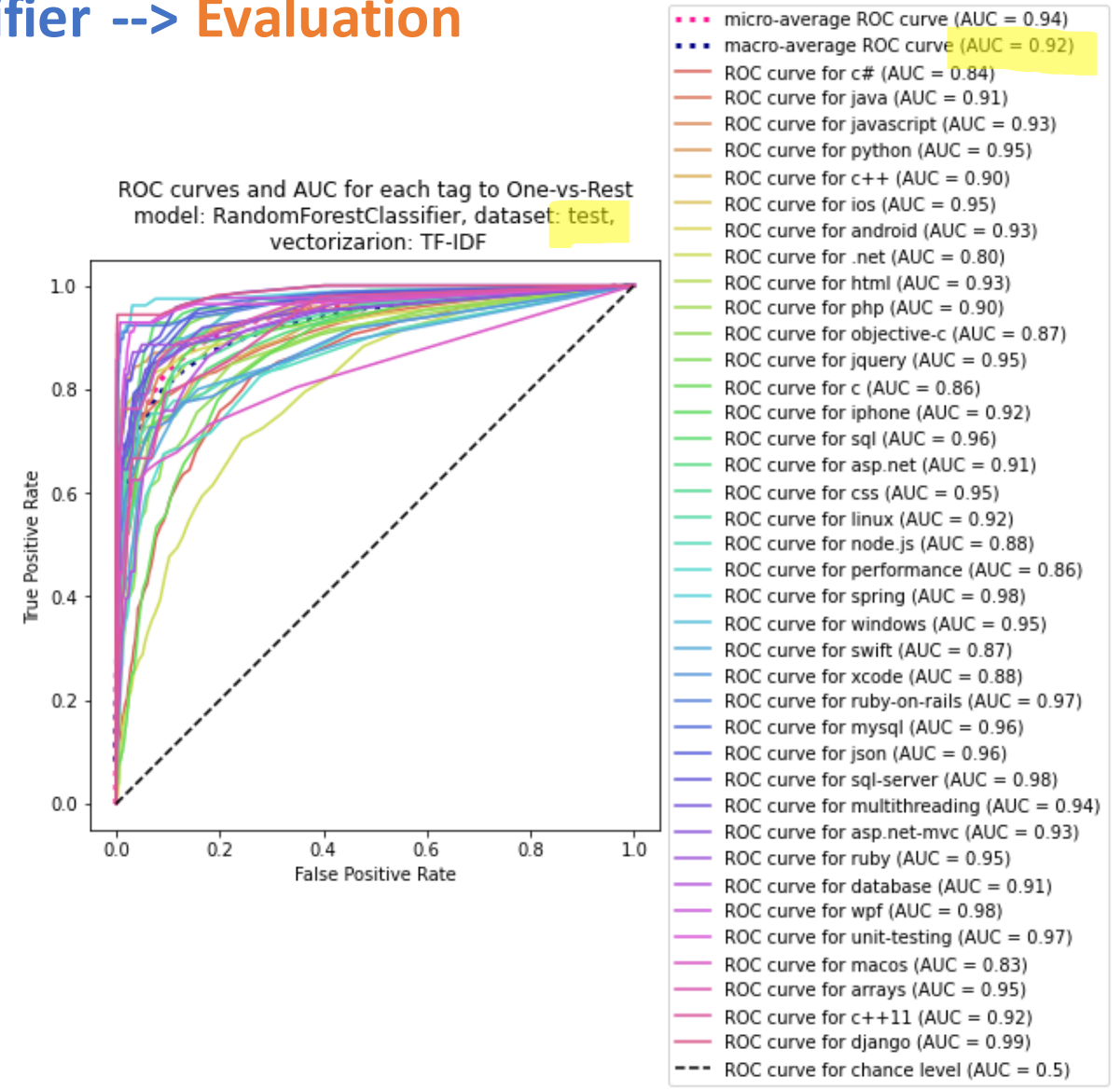




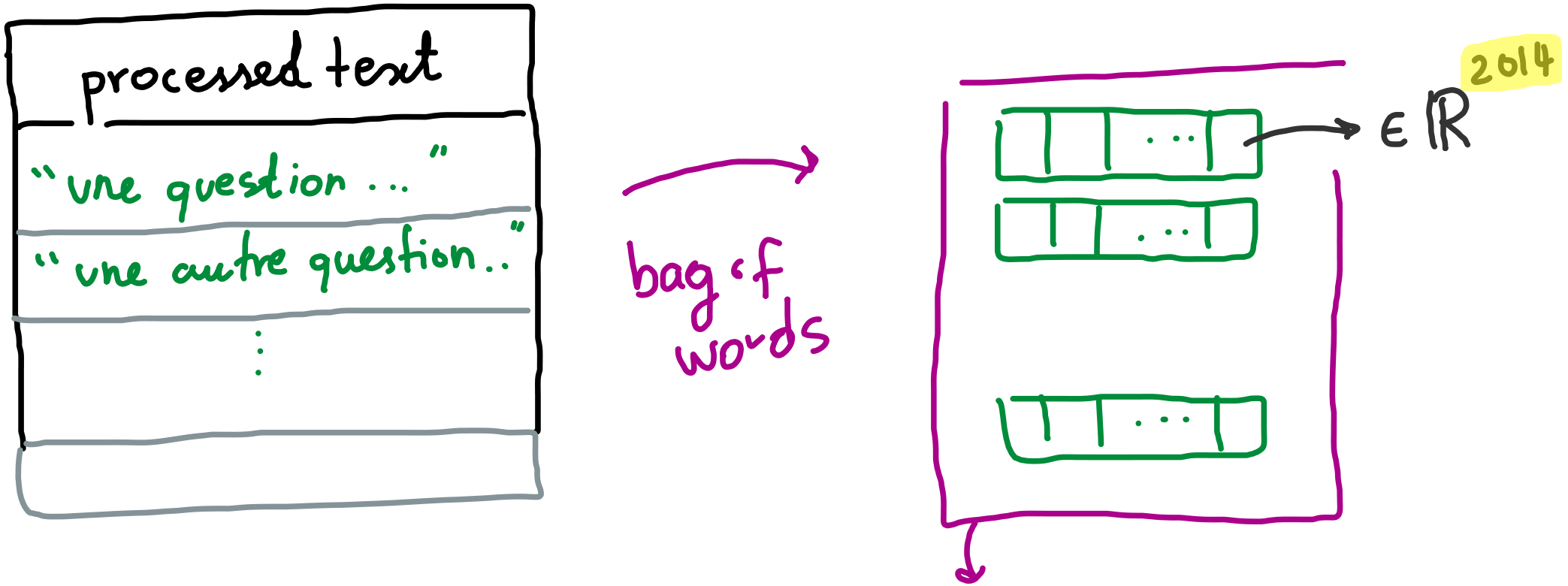
Modélisation (supervisée)

Bag of words: TF-IDF + **RandomForestClassifier** --> **Évaluation**

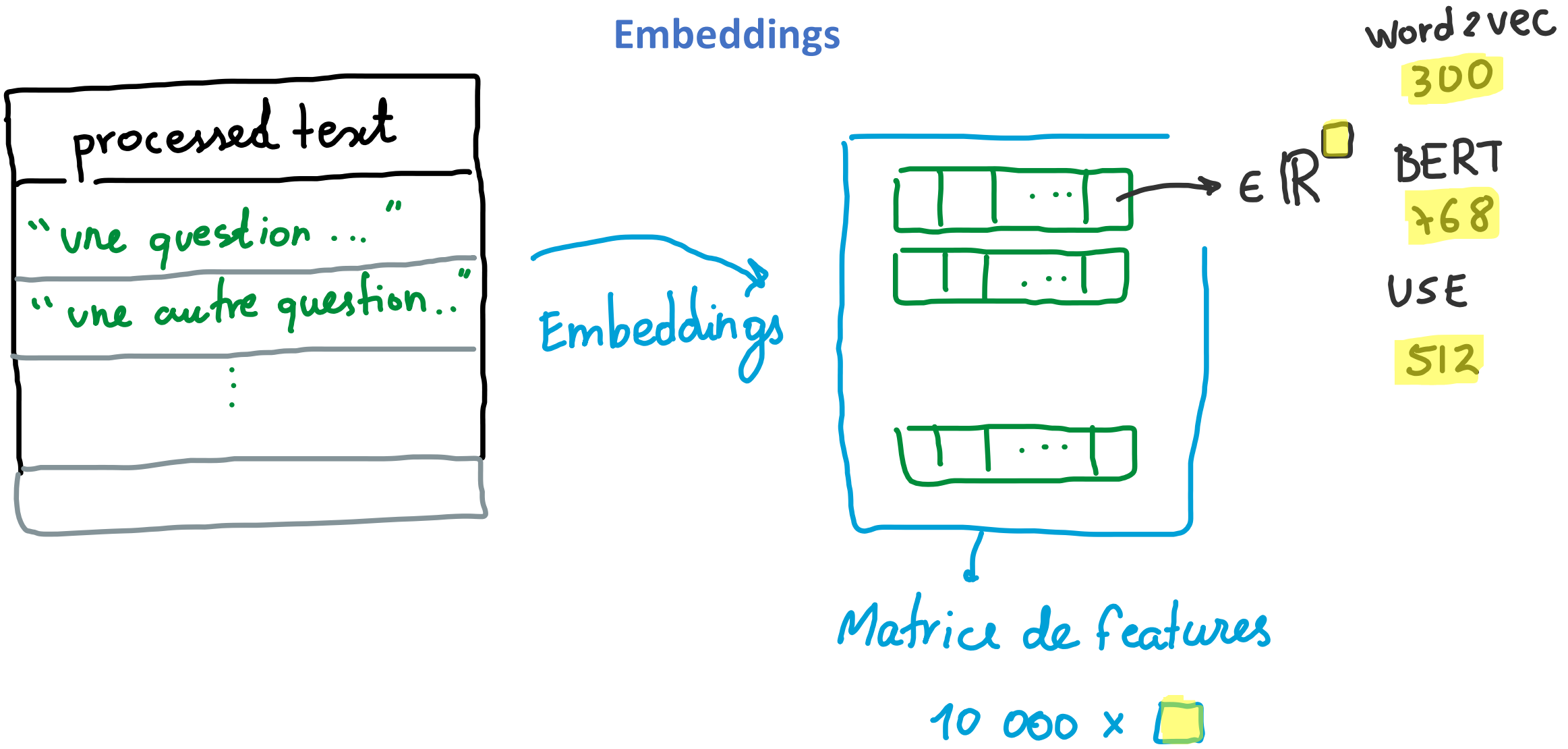




Modélisation (supervisée)



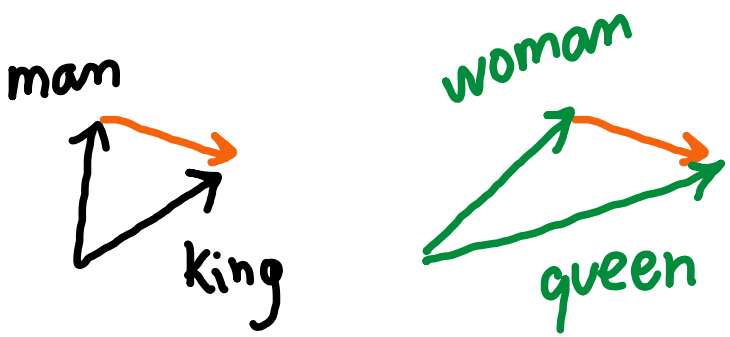
Modélisation (supervisée)



Modélisation (supervisée)

Embeddings

Word2Vec



- basée : réseaux de neurones
- CBOW, skipgram

BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

basée : Transformers

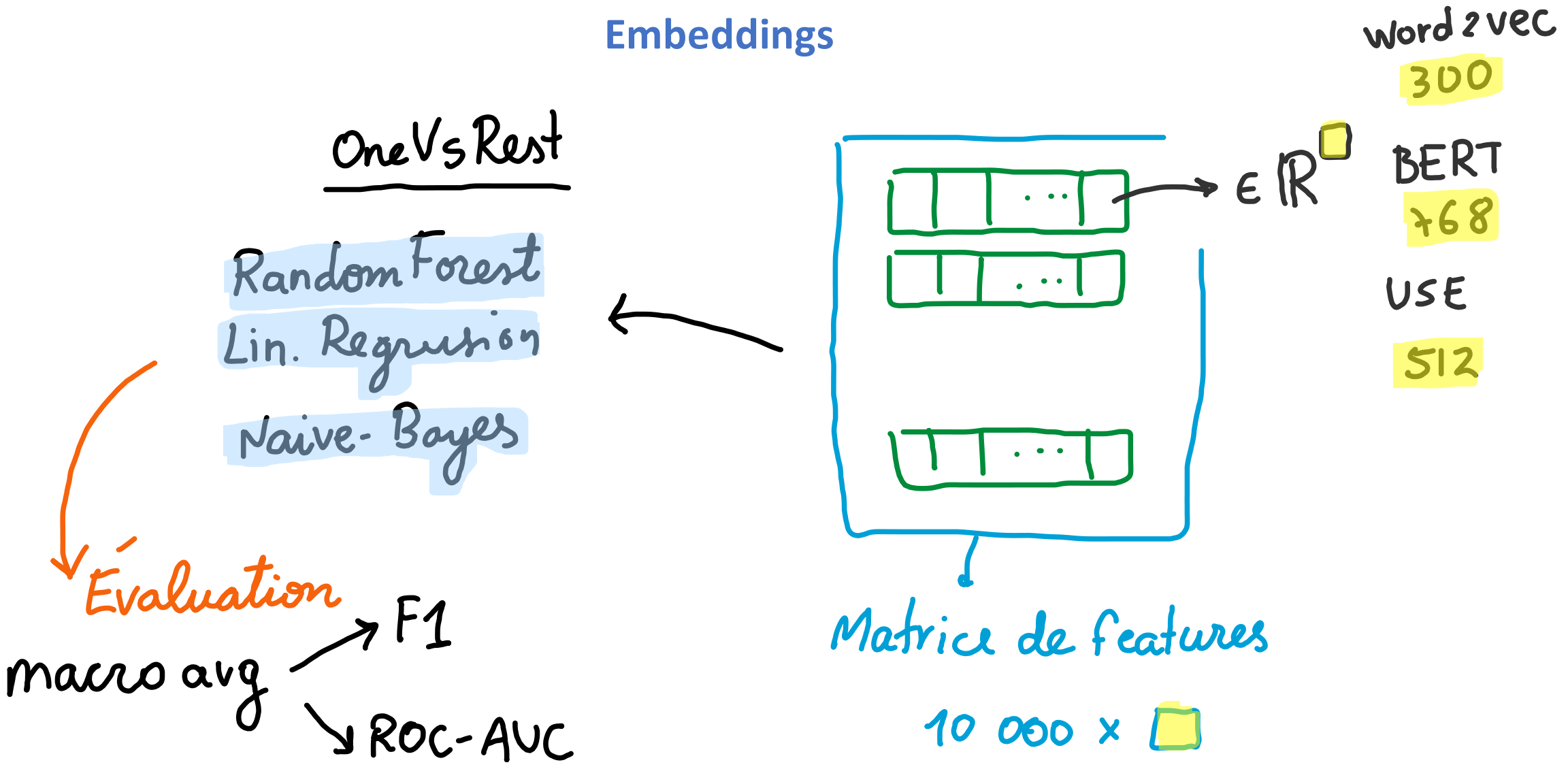
- pré-entrainement en masquant des mots

USE

Universal **S**entence **E**ncoder

phrase entière → embedding

Modélisation (supervisée)



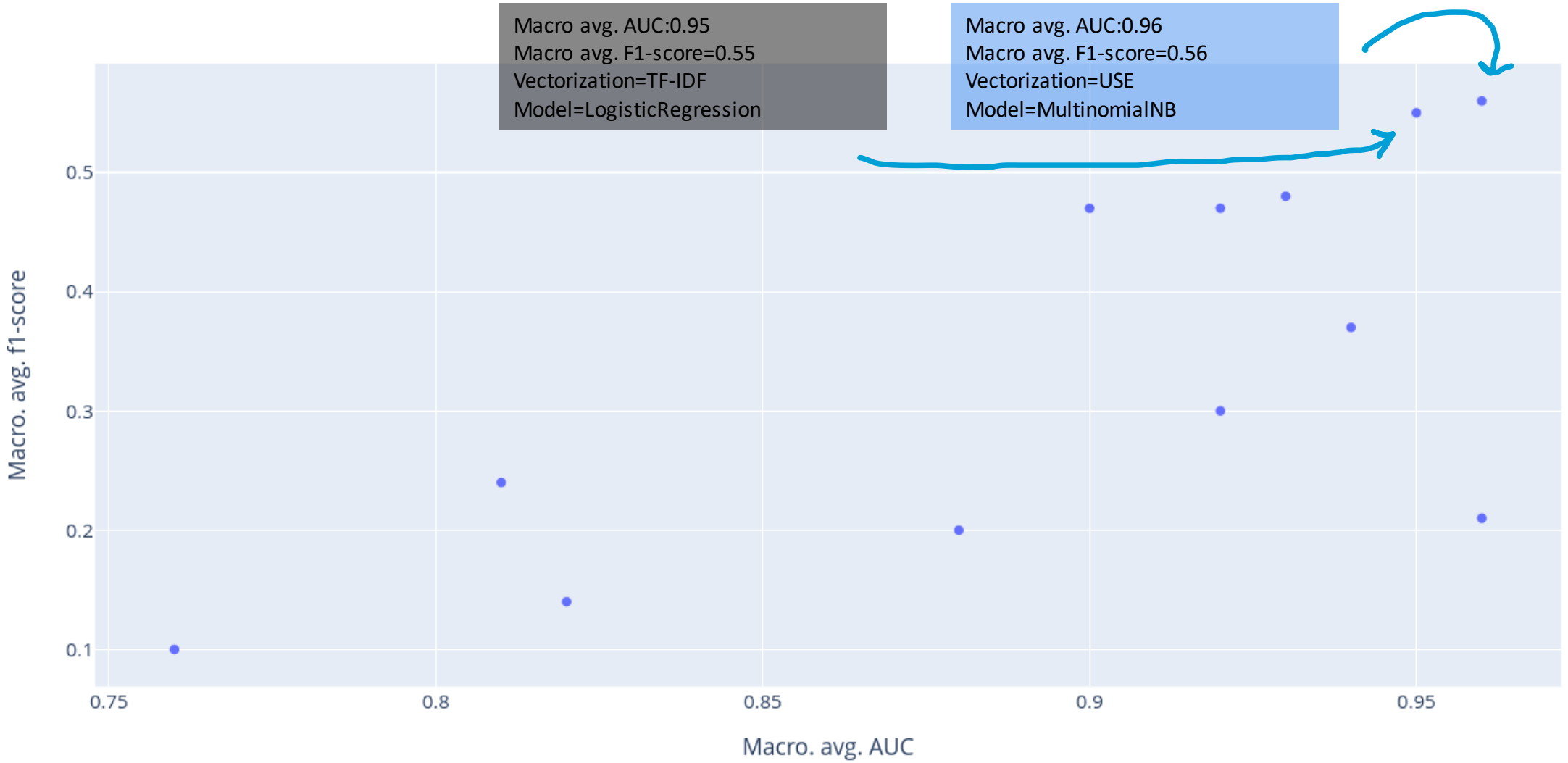
Modélisation (supervisée)

Comparaison performances tous les modèles supervisés

	Vectorization	Model	Macro. avg. f1-score	Macro. avg. AUC	Fit time
0	TF-IDF	LogisticRegression	0.55	0.95	3.87
1	TF-IDF	MultinomialNB	0.48	0.93	3.87
2	TF-IDF	RandomForestCl	0.43	0.92	86.42
3	Word2Vec	LogisticRegression	0.47	0.92	25.08
4	Word2Vec	RandomForest	0.15	0.82	401.76
5	Word2Vec	MultinomialNB	0.20	0.88	401.76
6	BERT	LogisticRegression	0.47	0.90	259.71
7	BERT	RandomForest	0.10	0.76	984.06
8	BERT	MultinomialNB	0.24	0.81	1.91
9	USE	LogisticRegression	0.21	0.96	19.21
10	USE	RandomForest	0.37	0.94	443.52
11	USE	MultinomialNB	0.56	0.96	0.62

Modélisation (supervisée)

Comparaison performances tous les modèles supervisés



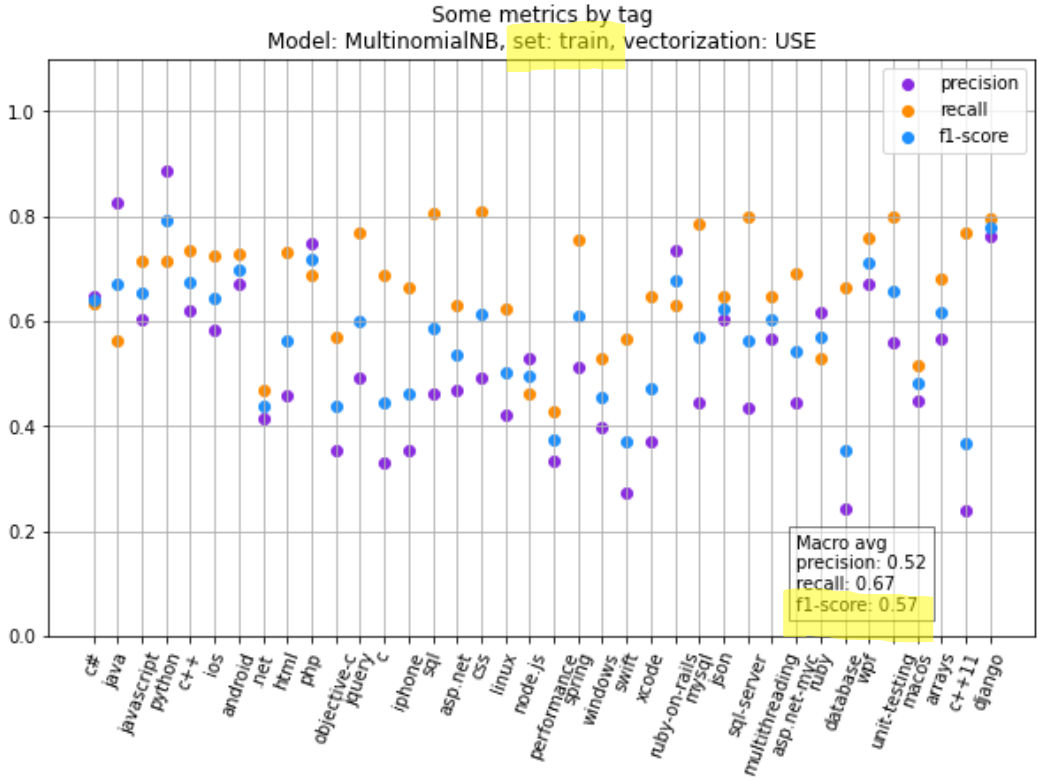
Modélisation (supervisée)

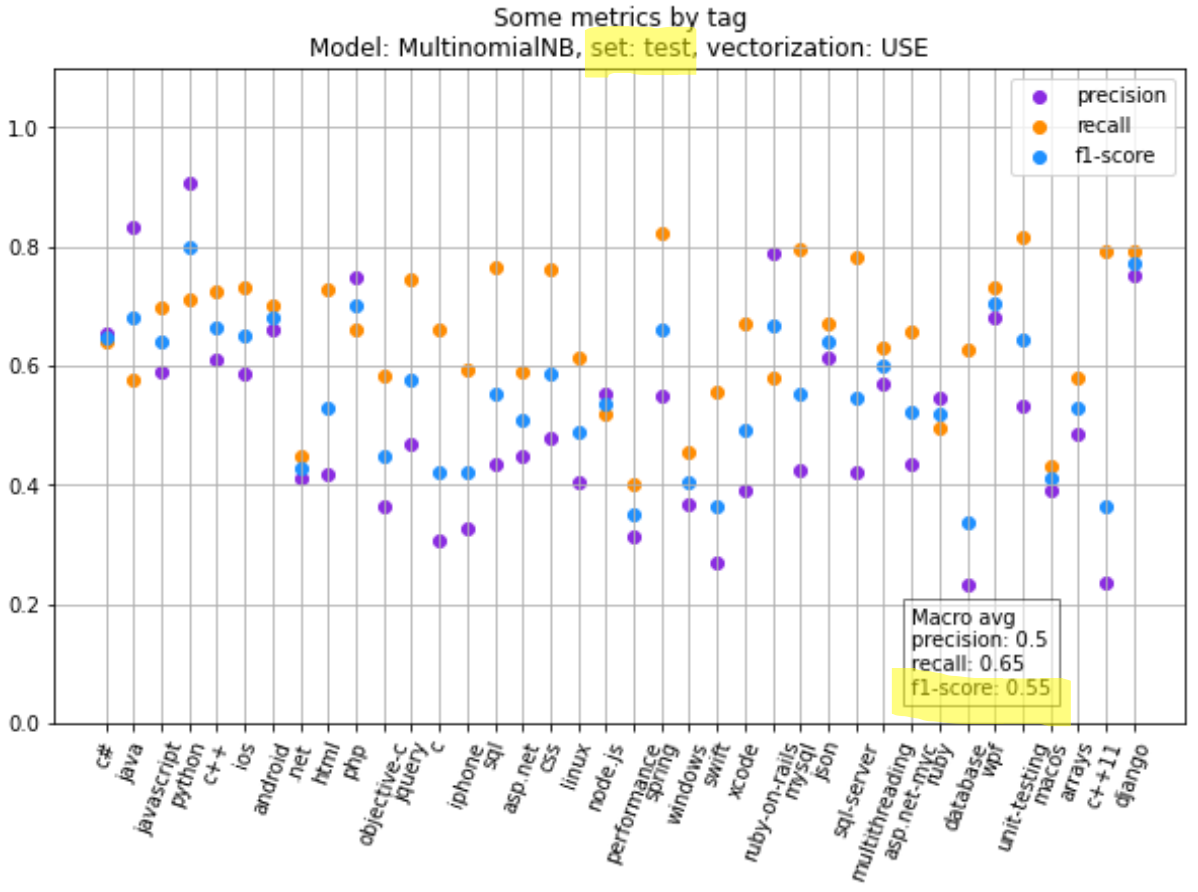
Comparaison performances tous les modèles supervisés

	Vectorization	Model	Macro. avg. f1-score	Macro. avg. AUC	Fit time
0	TF-IDF	LogisticRegression	0.55	0.95	3.87
1	TF-IDF	MultinomialNB	0.48	0.93	3.87
2	TF-IDF	RandomForestCl	0.43	0.92	86.42
3	Word2Vec	LogisticRegression	0.47	0.92	25.08
4	Word2Vec	RandomForest	0.15	0.82	401.76
5	Word2Vec	MultinomialNB	0.20	0.88	401.76
6	BERT	LogisticRegression	0.47	0.90	259.71
7	BERT	RandomForest	0.10	0.76	984.06
8	BERT	MultinomialNB	0.24	0.81	1.91
9	USE	LogisticRegression	0.21	0.96	19.21
10	USE	RandomForest	0.37	0.94	443.52
11	USE	MultinomialNB	0.56	0.96	0.62

Modélisation (supervisée)

Entraînement OneVsRest(Naïve Bayes Multinomial) avec toutes les données

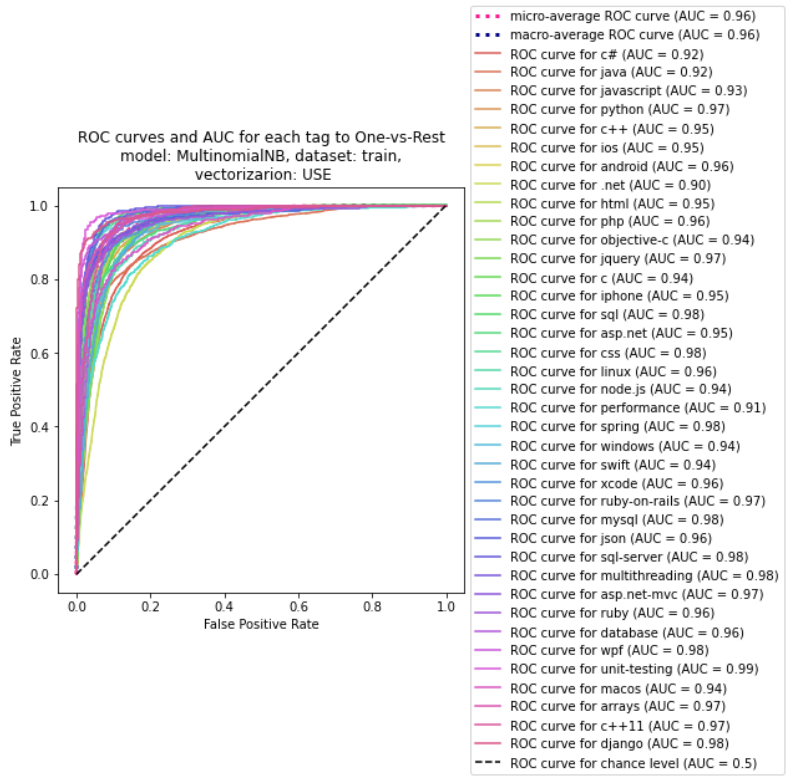


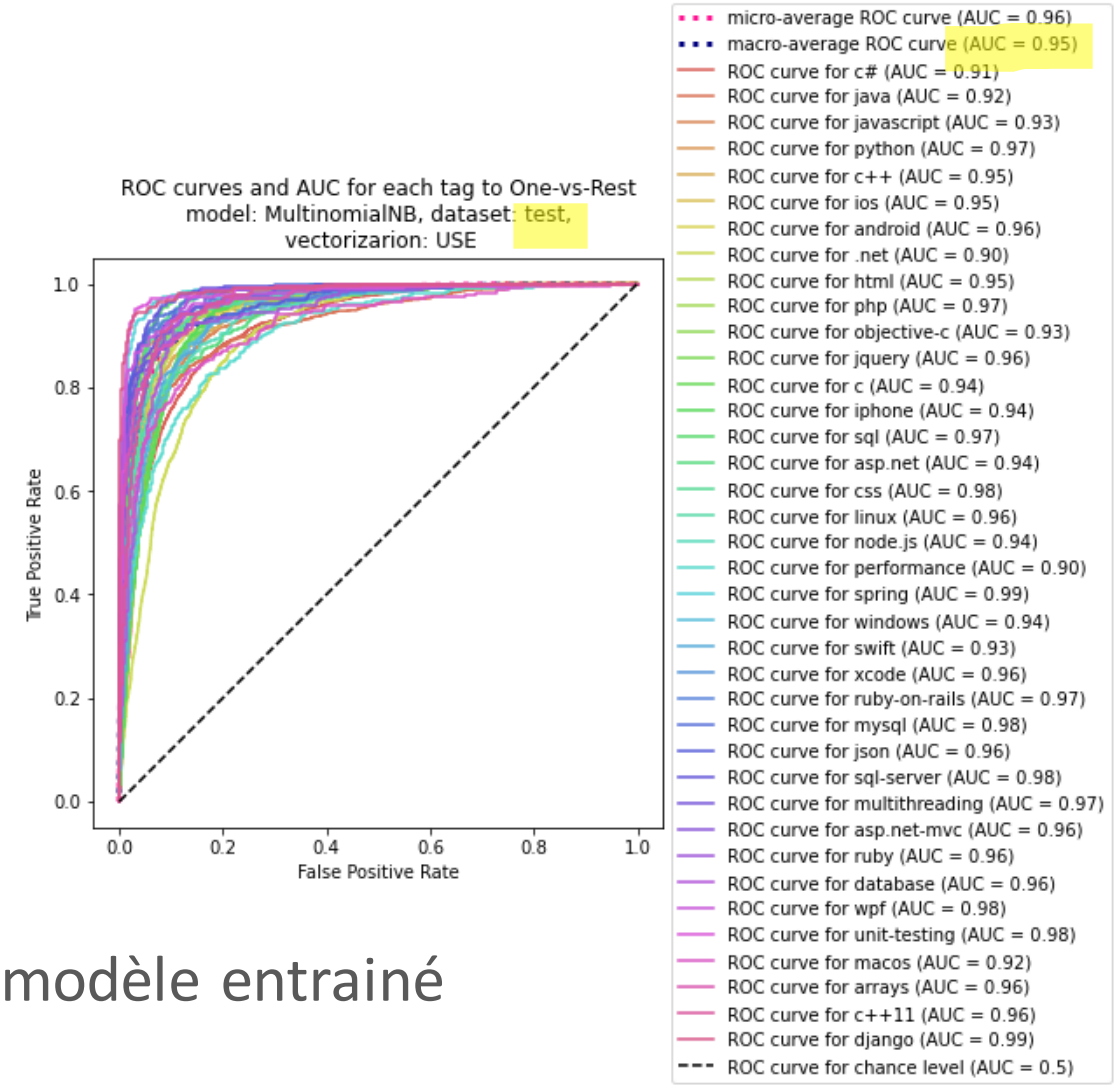


Exporter modèle entrainé

Modélisation (supervisée)

Entraînement OneVsRest(Naïve Bayes Multinomial) avec toutes les données

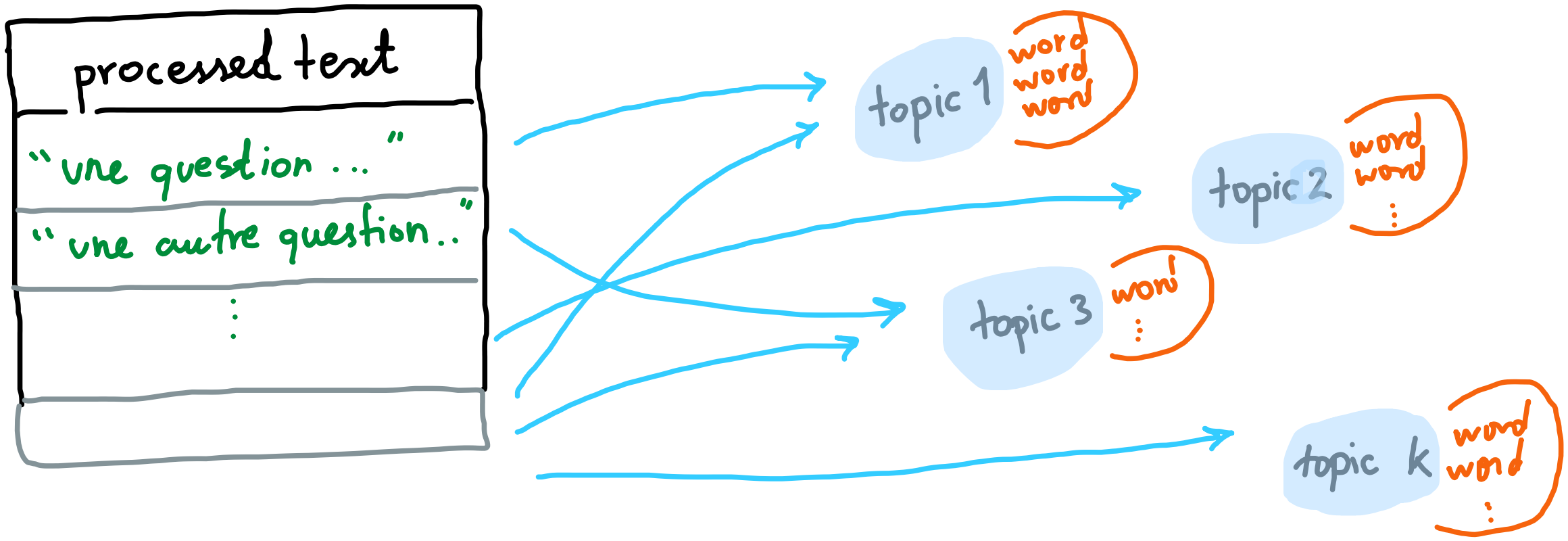




Exporter modèle entrainé

Modélisation (**non**-supervisée)

LDA : Latent Dirichlet Allocation



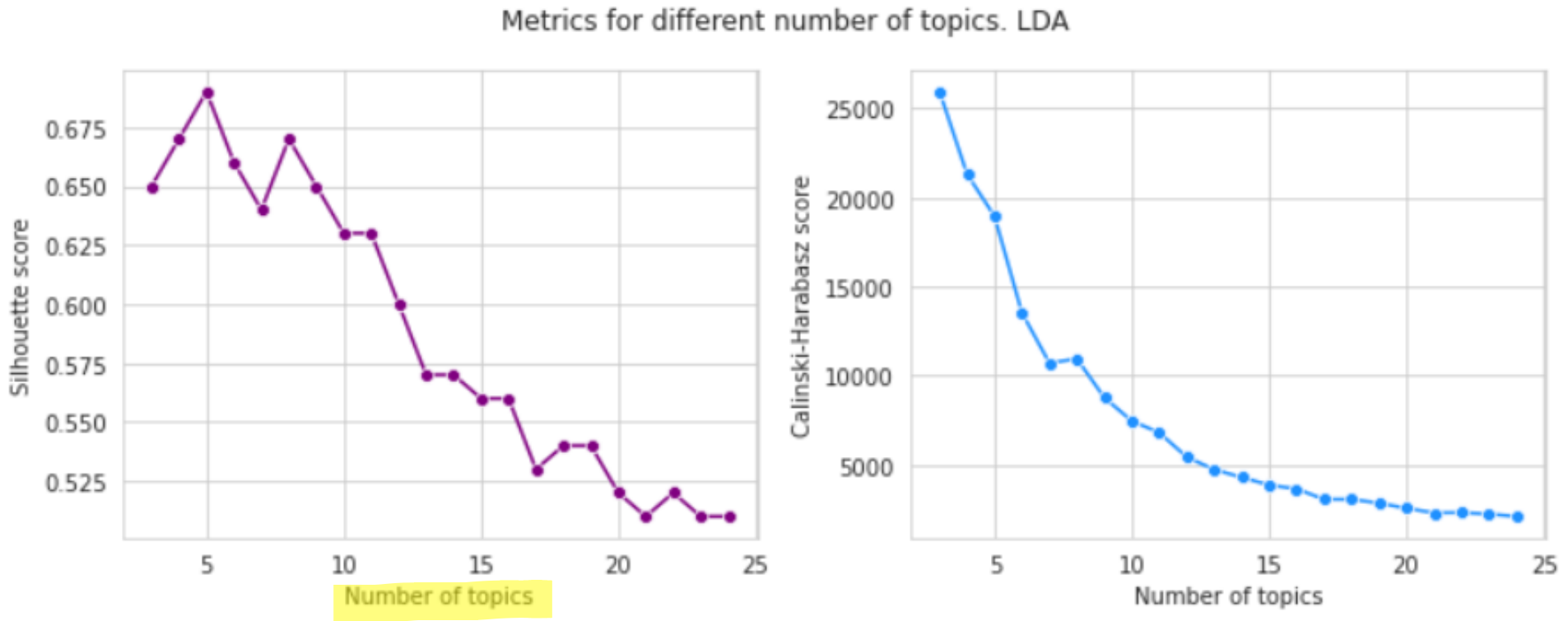
Modélisation (**non**-supervisée)

LDA : Latent Dirichlet Allocation

Modélisation (**non**-supervisée)

LDA : Latent Dirichlet Allocation. --> **Pas très adapté à notre but !**

Mes approches :

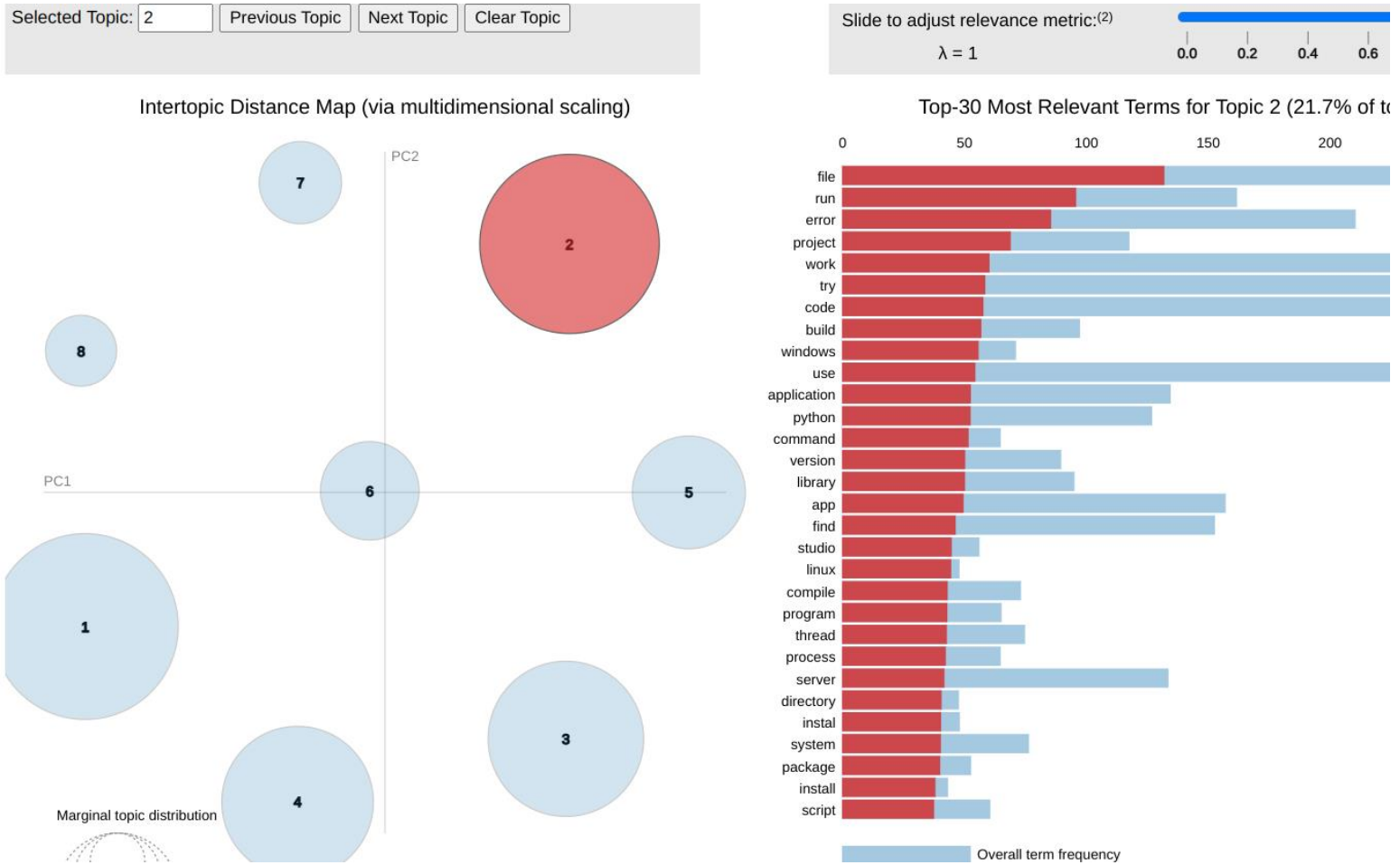


hyperparamètre

Modélisation (**non**-supervisée)

LDA : Latent Dirichlet Allocation. --> **Pas très adapté à notre but !**

Mes approches :



Distribution de mots par sujet

↓

idée

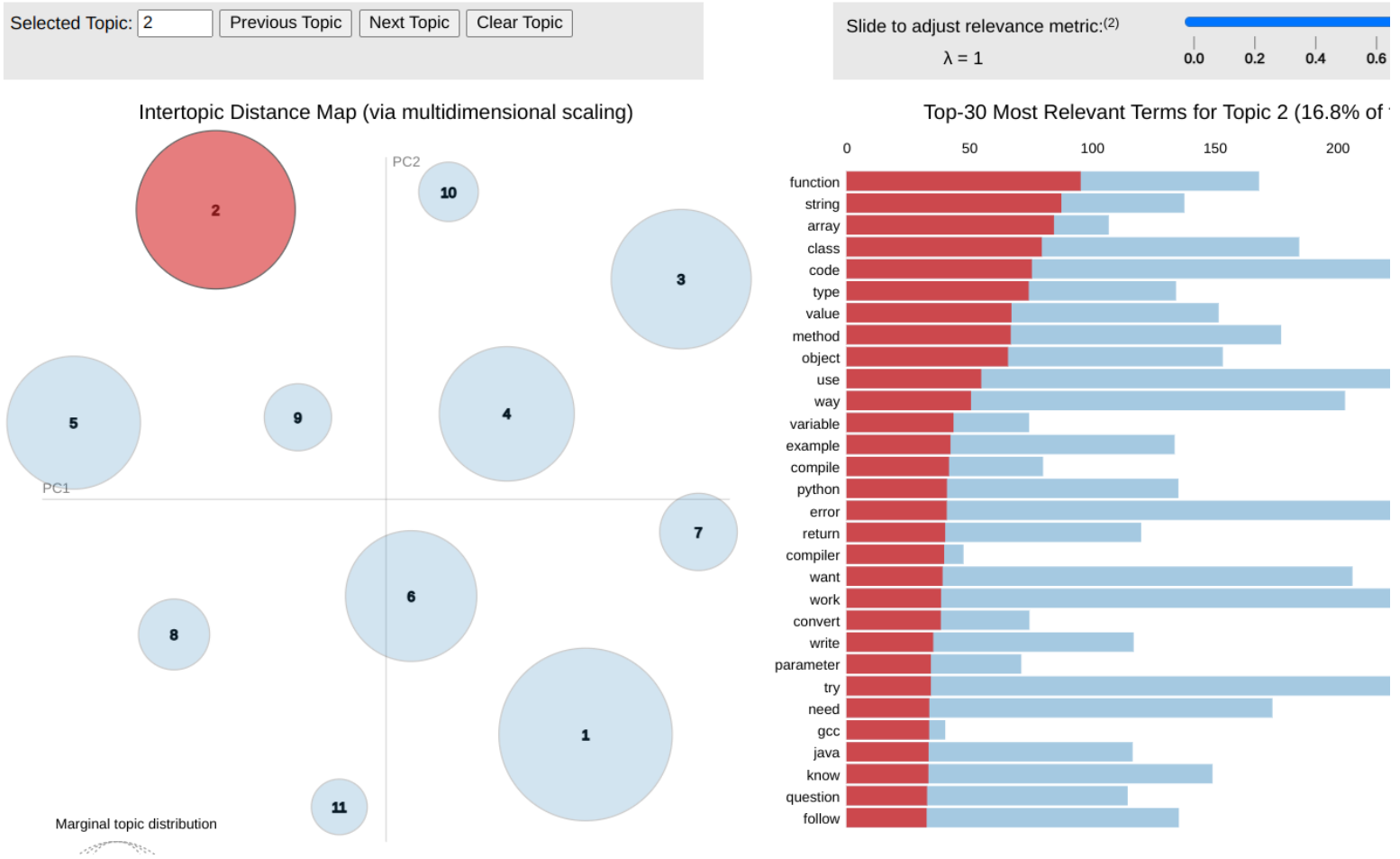
↓

intersection avec mes tags

Modélisation (**non**-supervisée)

LDA : Latent Dirichlet Allocation. --> **Pas très adapté à notre but !**

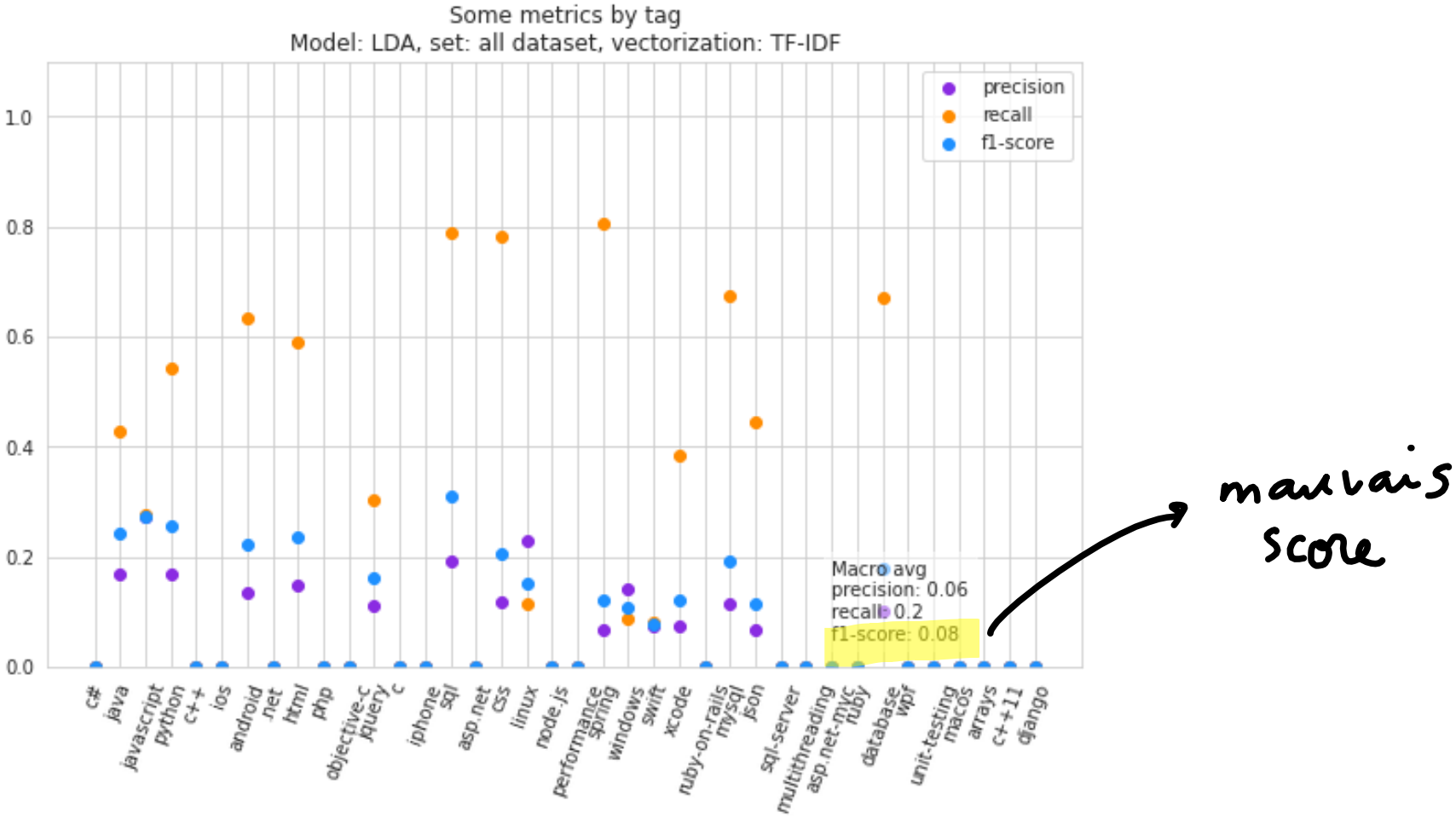
Mes approches :



Modélisation (**non**-supervisée) 3

LDA : Latent Dirichlet Allocation. --> **Pas très adapté à notre but !**

Mes approches :



4

API

API

Avec gradio

```python
def tag_suggestion(raw_text):
    """
    Returns a list of tags suggested for the question raw_text.
    """
    # Clean text first
    clean_text = clean(raw_text)
    document = [clean_text]

    # Find an embedding of the text with USE
    X = embed(document)

    # Predict a tag set with our classification model
    pred = my_pred(X)

    return binary_to_tag_list(pred)


# API ##

demo = gr.Interface(fn=tag_suggestion,
                    inputs="text",
                    outputs=["text"],
                    examples=examples)


if __name__ == "__main__":
    demo.launch()
```

1 Nettoyer texte avec notre fonction

↓ embed avec USE

2

↓ prediction OneVsRest Naive Bayes

3

↓ traduction

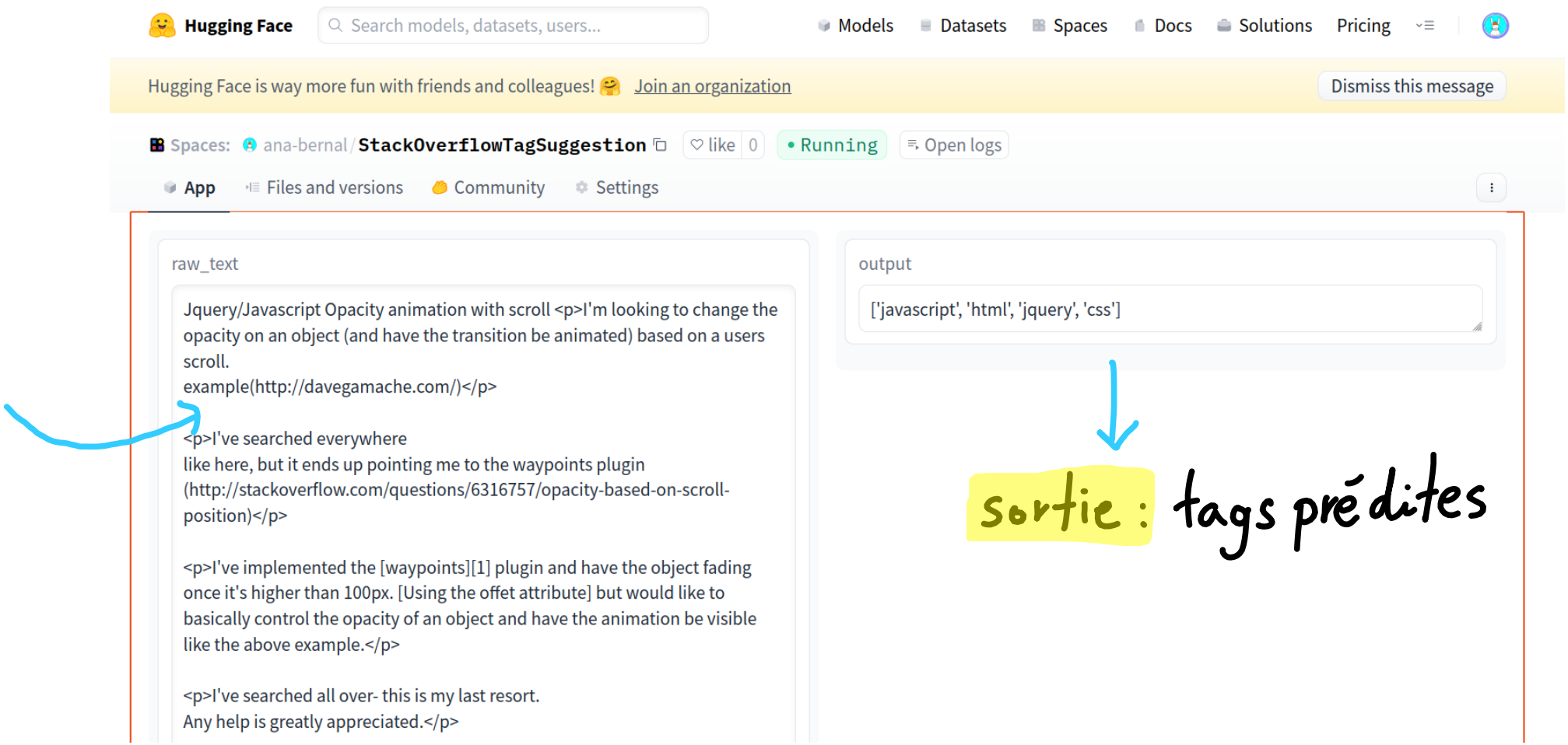
4

javascript php

API

Hébergé sur spaces de Hugging Face

aller à l'API



5

Conclusions

Conclusions

Suggestions des tags + Projet OC

- Fonctionnement **satisfaisant** avec le meilleur modèle.
- Beaucoup de **nouveaux apprentissages**: NLP, version control, API avec Gradio et HuggingFace.
- Traitement de texte : cas **complexe** pour un premier approche à NLP --> du code dans le texte, HTML, etc.

Pour aller + loin

- Inclure plus de tags: il faut plus de ressources de calcul. (nous: 40 sur 18 000)
- Inclure plus de questions (nous: 50 000 sur ∞)
- Améliorer visualisation API, rajouter des options.
- Prédictions avec des réseaux de neurones.
- Analyse exploratoire + profond.