Team T13

20_Newsgroup

AG_News

Data Preperation

Clean dataset

sklearn: fetch_20newsgroups(remove=('headers', 'footers', 'quotes')) --> remove revealing meta data about given doc

remove strings with '@' in them (emails)

Lemmatization

(e.g. "running" --> "run")
from nltk.corpus import stopwords
from pywsd.utils import lemmatize_sentence
--> standardize/prune words to their stem using english dictionary; also remove english stopwords without (much) meaning (e.g. "the", "a")

libraries:
sklearn
nltk
pywsd

LLM Summary

prompt:
"Summarize the following text in 3-4 sentences. Disregard previously seen texts and only consider this given document"
==> our goal is to avoid having the LLM collect "domain knowledge" of the texts

Vectorization

TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer

TfidfVectorizer(lowercase=True, stop_words="english")

Doc2Vec

from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize

tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()), tags=[str(i)]) for i, _d in enumerate(data)]
--> tokenize each doc to update doc vector based on unique doc vector (pre-step for Doc2Vec)

model = gensim.models.doc2vec.Doc2Vec(vector_size=[...], epochs=, window=[...]) # init of Doc2Vec
model.build_vocab(tagged_data)
model.train(tagged_data, total_examples=[...], epochs=[...])

S-BERT

from sentence-transformers import SentenceTransormer

model = SentenceTransformer("all-MiniLM-L6-v2") # no final model choice, but is lightweight and shows good benchmark performance

sbert_vectors = model.encode(documents) # output should be matrix of shape [n_docs, col] which can be used as classifier input in next step

libraries:
sklearn
nltk
gensim
sentence-transformers

Classification

SVM

sklearn: SVC(C=[0.001, 0.01,..., 10], kernel=['linear', 'poly', 'rbf'], gamma=[1, 10, 100])

MLPClassifier

sklearn.neural_network

hidden_layer_sizes:
activation function: 'relu' 'tanh' or 'logistic'
learning_rate: 'constant', 'invscaling', or 'adaptive'
early_stopping:
solver: 'adam', 'sgd', 'lbfgs'

Decision Tree

sklearn: tree.DecisionTreeClassifier():

criterion: 'gini', 'entropy'
max_depth: 1-30
max_features: 0.1 - 1.0 (fraction of samples) or 'sqrt', 'auto'

libraries:
sklearn

We adopt a hybrid approach that combines domain knowledge with informed trial-and-error to define key model parameters, while using GridSearchCV to systematically explore combinations (e.g. hidden layer configurations) to optimize performance.

XAI & Reasoning

LIME

LimeTextExplainer(): initialize with class names

.explain_instance(text_instance, predict_proba_fn): generate explanation for a single text example

.save_to_file(): visualize explanation in html or png file

LLM

requests.post(url, json=payload): interact with API endpoint from Ollama (http://localhost:11434/api/generate)

Evaluation

Compare LIME top words with LLM extracted key Tokens
-> **direct comparison**

Evaluation Metrics:
Stability- Measure outcome for ten iterations to see how stable response is

Robustness- Make very small changes to intput texts and measure how much the explanations vary

Understandability-
Provide both explanations to another model with the prompt to Rank Understandability from 1 to 10

libraries:
requests
lime