wsdm-final-submission

March 5, 2025

[1]:
```
import pandas as pd

train = pd.read_parquet('/kaggle/input/wsdm-cup-multilingual-chatbot-arena/test.
 ↪parquet')
```

[2]:
```
import string
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.util import ngrams  # For n-gram overlap
from textblob import TextBlob  # For sentiment analysis
from tqdm import tqdm

tqdm.pandas()  # Enable progress bar for pandas operations

# Define the check list for special characters
check_list = ['*', "'", '-', ',', '|', '\n', '$', '}', '{', ']', '[', '(', ')',␣
 ↪'#', '^', "=", ";", '"',
              '.', '**', ' ', '##', '###', "'s", ':']

# Function to compute text-based column features
def compute_feats(df):
    for col in tqdm(["response_a", "response_b", "prompt"]):
        df[f"{col}_len"] = df[col].str.len()
        df[f"{col}_spaces"] = df[col].str.count("\s")
        df[f"{col}_punct"] = df[col].str.count(",|\.|!")
        df[f"{col}_question_mark"] = df[col].str.count("\?")
        df[f"{col}_quot"] = df[col].str.count("'|\"")
        df[f"{col}_formatting_chars"] = df[col].str.count("\*|\_")
        df[f"{col}_math_chars"] = df[col].str.count("\-|\+|\=")
        df[f"{col}_curly_open"] = df[col].str.count("\{")
        df[f"{col}_curly_close"] = df[col].str.count("}")
        df[f"{col}_round_open"] = df[col].str.count("\(")
        df[f"{col}_round_close"] = df[col].str.count("\)")
        df[f"{col}_special_chars"] = df[col].str.count("\W")
        df[f"{col}_json"] = df[col].str.lower().str.count("json")
        df[f"{col}_yaml"] = df[col].str.lower().str.count("yaml")
```

```
    return df

# Function to calculate overlap features
def overlap_features(text1, text2):
    set1 = set(text1.split())
    set2 = set(text2.split())
    common_words = set1.intersection(set2)
    return {
        "overlap_count": len(common_words),
        "overlap_ratio_1": len(common_words) / len(set1) if set1 else 0,
        "overlap_ratio_2": len(common_words) / len(set2) if set2 else 0,
    }

# Function to calculate n-gram overlap features
def ngram_overlap(text1, text2, n):
    ngrams1 = set(ngrams(text1.split(), n))
    ngrams2 = set(ngrams(text2.split(), n))
    overlap_count = len(ngrams1 & ngrams2)
    jaccard_index = overlap_count / len(ngrams1 | ngrams2) if (ngrams1 |␣
 ↪ngrams2) else 0
    return {
        f"{n}gram_overlap_count": overlap_count,
        f"{n}gram_jaccard_index": jaccard_index,
    }

# Function to calculate sentiment differences
def sentiment_difference(text1, text2):
    sentiment1 = TextBlob(text1).sentiment.polarity
    sentiment2 = TextBlob(text2).sentiment.polarity
    return {
        "sentiment_difference": abs(sentiment1 - sentiment2),
        "sentiment_ratio": sentiment1 / sentiment2 if sentiment2 != 0 else 0,
    }

# Function to compute special character counts
def special_char_count_feature(row, check_list):
    total_count_a = sum(list(row["response_a"]).count(char) for char in␣
 ↪check_list)
    total_count_b = sum(list(row["response_b"]).count(char) for char in␣
 ↪check_list)
    return {
        "special_char_count_a": total_count_a,
        "special_char_count_b": total_count_b,
    }

# Function to calculate cosine similarity
def cosine_similarity_feature(text1, text2):
```

```
    try:
        if not text1.strip() or not text2.strip():  # Check for empty or␣
 ↪invalid text
            return 0.0
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform([text1, text2])
        return cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])[0, 0]
    except ValueError:
        return 0.0  # Handle empty vocabulary cases

# Main feature extraction function for each row
def extract_features_row(row):
    overlap_a = overlap_features(row["prompt"], row["response_a"])
    overlap_b = overlap_features(row["prompt"], row["response_b"])
    response_overlap = overlap_features(row["response_a"], row["response_b"])
    cosine_similarity_a = cosine_similarity_feature(row["prompt"],␣
 ↪row["response_a"])
    cosine_similarity_b = cosine_similarity_feature(row["prompt"],␣
 ↪row["response_b"])
    cosine_similarity_ab = cosine_similarity_feature(row["response_a"],␣
 ↪row["response_b"])

    ngram_features = {
        **ngram_overlap(row["prompt"], row["response_a"], 2),
        **ngram_overlap(row["prompt"], row["response_b"], 2),
    }

    sentiment_features = {
        **sentiment_difference(row["prompt"], row["response_a"]),
        **sentiment_difference(row["prompt"], row["response_b"]),
    }

    special_char_features = special_char_count_feature(row, check_list)

    all_features = {
        **overlap_a,
        **overlap_b,
        **response_overlap,
        "cosine_similarity_a": cosine_similarity_a,
        "cosine_similarity_b": cosine_similarity_b,
        "cosine_similarity_ab": cosine_similarity_ab,
        **ngram_features,
        **sentiment_features,
        **special_char_features,
    }
    return all_features
```

```python
# Function to extract all features
def extract_all_features(data):
    data = compute_feats(data)
    feature_dicts = []
    for _, row in tqdm(data.iterrows(), total=len(data)):
        feature_dicts.append(extract_features_row(row))
    additional_features_df = pd.DataFrame(feature_dicts)
    combined_features_df = pd.concat([data.reset_index(drop=True),␣
 ↪additional_features_df.reset_index(drop=True)], axis=1)
    return combined_features_df

# Create an inverted dataset for augmentation


# Combine original and inverted datasets


# Extract all features
features_df = extract_all_features(train)

# Display extracted features
print(features_df)
```

```
100%|      | 3/3 [00:00<00:00, 128.03it/s]
100%|      | 3/3 [00:00<00:00, 31.21it/s]
```

```
        id                                             prompt  \
0   327228  Caso Clínico: Un hombre de 70 años con anteced…
1  1139415   Peel Company received a cash dividend from a …
2  1235630  Há um grave problema com o relógio da torre da…

                                          response_a  \
0  **Diagnóstico Diferencial de Anemia en Pacient…
1  The correct answer is **(a) No   No**. Here's …
2  Dois problemas interessantes!\n\n**Problema 1:…

                                          response_b  scored  response_a_len  \
0  Basándonos en el caso clínico presentado, pode…   False            1961
1  The correct answer is **(a) No No**. Here's wh…   False             893
2  Vamos resolver os dois problemas em sequência…   False            1997

   response_a_spaces  response_a_punct  response_a_question_mark  \
0                322                28                         0
1                142                13                         0
2                381                38                         0

   response_a_quot  …  overlap_ratio_2  cosine_similarity_a  \
0                0  …         0.209016             0.703378
```

```
1                5  …         0.670886             0.374975
2                2  …         0.442308             0.512557

   cosine_similarity_b  cosine_similarity_ab  2gram_overlap_count  \
0             0.520680              0.765063                   22
1             0.414401              0.876672                    6
2             0.517243              0.688025                   44

   2gram_jaccard_index  sentiment_difference  sentiment_ratio  \
0             0.043478                 0.325         0.000000
1             0.038961                 0.020         0.948276
2             0.127907                 0.150         0.000000

   special_char_count_a  special_char_count_b
0                   406                   666
1                   186                   174
2                   441                   376

[3 rows x 59 columns]
```

[3]:
```
train.columns
```

[3]:
```
Index(['id', 'prompt', 'response_a', 'response_b', 'scored', 'response_a_len',
       'response_a_spaces', 'response_a_punct', 'response_a_question_mark',
       'response_a_quot', 'response_a_formatting_chars',
       'response_a_math_chars', 'response_a_curly_open',
       'response_a_curly_close', 'response_a_round_open',
       'response_a_round_close', 'response_a_special_chars', 'response_a_json',
       'response_a_yaml', 'response_b_len', 'response_b_spaces',
       'response_b_punct', 'response_b_question_mark', 'response_b_quot',
       'response_b_formatting_chars', 'response_b_math_chars',
       'response_b_curly_open', 'response_b_curly_close',
       'response_b_round_open', 'response_b_round_close',
       'response_b_special_chars', 'response_b_json', 'response_b_yaml',
       'prompt_len', 'prompt_spaces', 'prompt_punct', 'prompt_question_mark',
       'prompt_quot', 'prompt_formatting_chars', 'prompt_math_chars',
       'prompt_curly_open', 'prompt_curly_close', 'prompt_round_open',
       'prompt_round_close', 'prompt_special_chars', 'prompt_json',
       'prompt_yaml'],
      dtype='object')
```

[4]:
```
train['scored']
```

[4]:
```
0    False
1    False
2    False
```

```
Name: scored, dtype: bool
```

[5]:
```
features_df.columns
```

[5]:
```
Index(['id', 'prompt', 'response_a', 'response_b', 'scored', 'response_a_len',
       'response_a_spaces', 'response_a_punct', 'response_a_question_mark',
       'response_a_quot', 'response_a_formatting_chars',
       'response_a_math_chars', 'response_a_curly_open',
       'response_a_curly_close', 'response_a_round_open',
       'response_a_round_close', 'response_a_special_chars', 'response_a_json',
       'response_a_yaml', 'response_b_len', 'response_b_spaces',
       'response_b_punct', 'response_b_question_mark', 'response_b_quot',
       'response_b_formatting_chars', 'response_b_math_chars',
       'response_b_curly_open', 'response_b_curly_close',
       'response_b_round_open', 'response_b_round_close',
       'response_b_special_chars', 'response_b_json', 'response_b_yaml',
       'prompt_len', 'prompt_spaces', 'prompt_punct', 'prompt_question_mark',
       'prompt_quot', 'prompt_formatting_chars', 'prompt_math_chars',
       'prompt_curly_open', 'prompt_curly_close', 'prompt_round_open',
       'prompt_round_close', 'prompt_special_chars', 'prompt_json',
       'prompt_yaml', 'overlap_count', 'overlap_ratio_1', 'overlap_ratio_2',
       'cosine_similarity_a', 'cosine_similarity_b', 'cosine_similarity_ab',
       '2gram_overlap_count', '2gram_jaccard_index', 'sentiment_difference',
       'sentiment_ratio', 'special_char_count_a', 'special_char_count_b'],
      dtype='object')
```

[6]:
```
features_df = features_df.drop(columns=['id', 'prompt', 'response_a',␣
↪'response_b', 'scored'])
```

[7]:
```
features_df.shape
```

[7]:
```
(3, 54)
```

[ ]:
```
import lightgbm as lgb
model = lgb.Booster(model_file='/kaggle/input/lgbm_v1/other/default/1/
↪lgbm_model.txt')
```

```
[LightGBM] [Warning] Ignoring unrecognized parameter 'early_stopping_min_delta'
found in model string.
```

[ ]:
```
y_pred_proba = model.predict(features_df)
y_pred =['model_b' if i>=0.5 else 'model_a' for i in  y_pred_proba]
```

[10]:
```
import pandas as pd

sub = pd.read_csv('/kaggle/input/wsdm-cup-multilingual-chatbot-arena/
↪sample_submission.csv')
```

[11]:
```
sub['winner'] = y_pred
```

[12]:
```
sub
```

[12]:

	id	winner
0	327228	model_b
1	1139415	model_b
2	1235630	model_a

[13]:
```
sub.to_csv('submission.csv',index=False)
```

[ ]:
```

```