cs585-hw2-ankan-mazumdar

September 28, 2023

1 CS 585 - HW 2

First, we will import packages and modules and load both the datasets

#PROBLEM 1 – Reading the data (5 pts) • Using Python, read in the 2 clickbait datasets (See section DATA), and combine both into a single, shuffled dataset. (One function to shuffle data is numpy.random.shuffle) • Next, split your dataset into train, test, and validation datasets. Use a split of 72% train, 8% validation, and 20% test. (Which is equivalent to a 20% test set, and the remainer split 90%/10% for train and validation). o If you prefer, you may save each split as an index (list of row numbers) rather than creating 3 separate datasets. • What is the "target rate" of each of these three datasets? That is, what % of the test dataset is labeled as clickbait? Show your result in your notebook.

[25]:
```
import pandas as pd
import numpy as np

def read_and_format_text_file(filename):
    # Read the text file with a newline delimiter
    with open(filename, 'r') as file:
        lines = file.readlines()

    # Remove newline characters ('\n') from each line
    lines = [line.strip() for line in lines]

    # Create a DataFrame with the specified column name
    dataset = pd.DataFrame(lines)

    return dataset


formatted_dataset1 = read_and_format_text_file('/content/clickbait.txt')
formatted_dataset2 = read_and_format_text_file('/content/not-clickbait.txt')

formatted_dataset1['label'] = 1

formatted_dataset2['label'] = 0
# print(formatted_dataset1.head(3))
# print(formatted_dataset2.head(3))
```

```
# print(formatted_dataset2.shape)
combined_dataset = pd.concat([formatted_dataset1, formatted_dataset2])

# print(combined_dataset.head(3))
# Shuffle the combined dataset
# shuffled_dataset = combined_dataset.sample(frac=1, random_state=42).
 ↪reset_index(drop=True)
# print (shuffled_dataset)

combined_array = combined_dataset.values

# Shuffle the NumPy array using numpy.random.shuffle
np.random.shuffle(combined_array)

# Convert the shuffled array back to a DataFrame
shuffled_dataset = pd.DataFrame(combined_array)
# print('type(shuffled_dataset)',type(shuffled_dataset))

# print(shuffled_dataset.tail(100))
# shuffled_dataset.rename(columns={0: 'text', 1: 'label'}, inplace=True)
shuffled_dataset = shuffled_dataset.rename(columns={0: 'text', 1: 'label'})

# print(shuffled_dataset.head(3))
count_rows_with_label_1 = shuffled_dataset['label'].sum()
# print('shuffled_dataset with count',shuffled_dataset)
# Define the split percentages
train_split = 0.72
validation_split = 0.08
test_split = 0.20
# test_size = len(shuffled_dataset) - train_split - validation_split

# Calculate the number of samples for each split
total_samples = len(shuffled_dataset)
train_samples = int(train_split * total_samples)
validation_samples = int(validation_split * total_samples)

# Split the dataset into train, validation, and test sets


train_data = shuffled_dataset.iloc[:train_samples].copy()
validation_data = shuffled_dataset.iloc[train_samples:
 ↪train_samples+validation_samples].copy()
test_data = shuffled_dataset.iloc[train_samples+validation_samples:].copy()


# print('train_data'train_data.shape)
# print(validation_data.shape)
```

```
# print(test_data.shape)
# test_data

#clickbait precentage calculator
def calculate_clickbait_percentage(dataset):
    # Clean the 'label' column by stripping whitespace and converting to string.
    dataset['label'] = dataset['label'].astype(str).str.strip()

    # Calculate the count of rows with the label '1'.
    count_rows_with_label_1 = (dataset['label'] == '1').sum()

    # Calculate the total number of rows in the dataset.
    total_num_rows = len(dataset)

    # Calculate the percentage of rows labeled as '1'.
    percentage = (count_rows_with_label_1 / total_num_rows) * 100

    return percentage

# Usage example:
# Assuming your DataFrame is named 'test_data'
train_clickbait_percentage = calculate_clickbait_percentage(train_data)
validation_clickbait_percentage =␣
 ↪calculate_clickbait_percentage(validation_data)
test_clickbait_percentage = calculate_clickbait_percentage(test_data)
print(f"Percentage of clickbait in the train dataset:␣
 ↪{train_clickbait_percentage:.2f}%")
print(f"Percentage of clickbait in the validation  dataset:␣
 ↪{validation_clickbait_percentage:.2f}%")
print(f"Percentage of clickbait in the test dataset: {test_clickbait_percentage:
 ↪.2f}%")
```

```
Percentage of clickbait in the train dataset: 34.50%
Percentage of clickbait in the validation  dataset: 37.70%
Percentage of clickbait in the test dataset: 31.17%
```

#PROBLEM 3 – Training a single Bag-of-Words (BOW) Text Classifier (20 pts) • Using scikit-learn pipelines module, create a Pipeline to train a BOW naïve bayes model. We suggest the classes CountVectorizer and MultinomialNB. Include both unigrams and bigrams in your model in your vectorizer vocabulary (see parameter: ngram_range) • Fit your classifier on your training set • Compute the precision, recall, and F1-score on both your training and validation datasets using functions in sklearn.metrics. Show results in your notebook. Use "clickbait" is your target class (I.e., y=1 for clickbait and y=0 for non-clickbait) ALTERNATIVE: If you are already well-versed in Naïve Bayes, you may select another bag-of-words classifier for this problem. Your method should have some way to select top features or key indicators, mapped to words or n-grams in your vocabulary, so that you can complete the remaining problems

[26]:
```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import precision_score, recall_score, f1_score

# Create a pipeline with CountVectorizer and MultinomialNB
text_classification_pipeline = Pipeline([
    ('text_vectorizer', CountVectorizer(ngram_range=(1, 2))),  # Include␣
 ↪unigrams and bigrams
    ('classifier', MultinomialNB())
])

# Fit the classifier on your training set
train_data = train_data.copy()
validation_data = validation_data.copy()
train_data['label'] = train_data['label'].astype(int)
validation_data['label'] = validation_data['label'].astype(int)

text_classification_pipeline.fit(train_data['text'], train_data['label'])

# Predictions on training and validation sets
train_predictions = text_classification_pipeline.predict(train_data['text'])
valid_predictions = text_classification_pipeline.
 ↪predict(validation_data['text'])

# Compute precision, recall, and F1-score on the training set
train_precision = precision_score(train_data['label'], train_predictions,␣
 ↪average='binary', pos_label=1)
train_recall = recall_score(train_data['label'], train_predictions,␣
 ↪average='binary', pos_label=1)
train_f1_score = f1_score(train_data['label'], train_predictions,␣
 ↪average='binary', pos_label=1)

# Compute precision, recall, and F1-score on the validation set
valid_precision = precision_score(validation_data['label'], valid_predictions,␣
 ↪average='binary', pos_label=1)
valid_recall = recall_score(validation_data['label'], valid_predictions,␣
 ↪average='binary', pos_label=1)
valid_f1_score = f1_score(validation_data['label'], valid_predictions,␣
 ↪average='binary', pos_label=1)

# Display results
print(f"Train data Precision: {train_precision:.4f}")
print(f"Train data Recall: {train_recall:.4f}")
print(f"Train data F1-Score: {train_f1_score:.4f}")
print(f"Validation data Precision: {valid_precision:.4f}")
```

```
print(f"Validation data Recall: {valid_recall:.4f}")
print(f"Validation data F1-Score: {valid_f1_score:.4f}")

# Check if we can exclude stop words
```

```
Train data Precision: 0.9900
Train data Recall: 0.9983
Train data F1-Score: 0.9941
Validation data Precision: 0.9104
Validation data Recall: 0.8472
Validation data F1-Score: 0.8777
```

#PROBLEM 4 – Hyperparameter Tuning (20 pts) Using the ParameterGrid class, run a small grid search where you vary at least 3 parameters of your model • max_df for your count vectorizer (threshold to filter document frequency) • alpha or smoothing of your NaïveBayes model • One other parameter of your choice. This can be non-numeric; for example, you can consider a model with and without bigrams (see parameter "ngram" in class CountVectorizer) Show metrics on your validation set for precision, recall, and F1-score. If your grid search is very large (>50 rows) you may limit output to the highest and lowest results.

ALTERNATIVE – If you used a method other than Naïve Bayes in Problem 3, then be sure it is clear what metrics you tuned in Problem 4.

Solution for Paramater GRID-

[27]:
```
import pandas as pd
from sklearn.model_selection import ParameterGrid
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import precision_score, recall_score, f1_score
from nltk.corpus import stopwords
import nltk

# Download NLTK stopwords (if not already downloaded)
nltk.download('stopwords')

# Define a function to remove stopwords using NLTK
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Assuming 'train_data' and 'validation_data' are your datasets
train_data = train_data.copy()
validation_data = validation_data.copy()
train_data['cleaned_text'] = train_data['text'].apply(remove_stopwords)
validation_data['cleaned_text'] = validation_data['text'].
 ↪apply(remove_stopwords)
```

```
# Define the parameter grid
param_grid = {
    'vectorizer__max_df': [0.2, 0.4, 0.6],
    'classifier__alpha': [0.1, 1.0, 3.0 , 5.0],
    'vectorizer__ngram_range': [(1, 1), (1, 2)]  # Varying ngram range
}

# Initialize an empty list to store results
results_list = []

# Create a pipeline with CountVectorizer and MultinomialNB
text_classification_pipeline = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('classifier', MultinomialNB())
])

# Perform grid search and compute metrics for each parameter combination
for params in ParameterGrid(param_grid):
    # Set parameters for the pipeline
    text_classification_pipeline.set_params(**params)

    # Fit the pipeline on the training data
    text_classification_pipeline.fit(train_data['cleaned_text'],␣
 ↪train_data['label'])

    # Make predictions on the validation data
    validation_predictions = text_classification_pipeline.
 ↪predict(validation_data['cleaned_text'])

    # Compute precision, recall, and F1-score
    precision = precision_score(validation_data['label'],␣
 ↪validation_predictions)
    recall = recall_score(validation_data['label'], validation_predictions)
    f1 = f1_score(validation_data['label'], validation_predictions)

    # Append results to the list
    results_list.append({
        'max_df': params['vectorizer__max_df'],
        'alpha': params['classifier__alpha'],
        'ngram_range': params['vectorizer__ngram_range'],
        'precision': precision,
        'recall': recall,
        'f1_score': f1
    })

# Create a DataFrame by concatenating the results list
```

```
results_df = pd.concat([pd.DataFrame(result) for result in results_list],␣
 ↪ignore_index=True)


print(results_df)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

	max_df	alpha	ngram_range	precision	recall	f1_score
0	0.2	0.1	1	0.805970	0.750000	0.776978
1	0.2	0.1	1	0.805970	0.750000	0.776978
2	0.2	0.1	1	0.833333	0.763889	0.797101
3	0.2	0.1	2	0.833333	0.763889	0.797101
4	0.4	0.1	1	0.805970	0.750000	0.776978
5	0.4	0.1	1	0.805970	0.750000	0.776978
6	0.4	0.1	1	0.833333	0.763889	0.797101
7	0.4	0.1	2	0.833333	0.763889	0.797101
8	0.6	0.1	1	0.805970	0.750000	0.776978
9	0.6	0.1	1	0.805970	0.750000	0.776978
10	0.6	0.1	1	0.833333	0.763889	0.797101
11	0.6	0.1	2	0.833333	0.763889	0.797101
12	0.2	1.0	1	0.830769	0.750000	0.788321
13	0.2	1.0	1	0.830769	0.750000	0.788321
14	0.2	1.0	1	0.836066	0.708333	0.766917
15	0.2	1.0	2	0.836066	0.708333	0.766917
16	0.4	1.0	1	0.830769	0.750000	0.788321
17	0.4	1.0	1	0.830769	0.750000	0.788321
18	0.4	1.0	1	0.836066	0.708333	0.766917
19	0.4	1.0	2	0.836066	0.708333	0.766917
20	0.6	1.0	1	0.830769	0.750000	0.788321
21	0.6	1.0	1	0.830769	0.750000	0.788321
22	0.6	1.0	1	0.836066	0.708333	0.766917
23	0.6	1.0	2	0.836066	0.708333	0.766917
24	0.2	3.0	1	0.859649	0.680556	0.759690
25	0.2	3.0	1	0.859649	0.680556	0.759690
26	0.2	3.0	1	0.865385	0.625000	0.725806
27	0.2	3.0	2	0.865385	0.625000	0.725806
28	0.4	3.0	1	0.859649	0.680556	0.759690
29	0.4	3.0	1	0.859649	0.680556	0.759690
30	0.4	3.0	1	0.865385	0.625000	0.725806
31	0.4	3.0	2	0.865385	0.625000	0.725806
32	0.6	3.0	1	0.859649	0.680556	0.759690
33	0.6	3.0	1	0.859649	0.680556	0.759690
34	0.6	3.0	1	0.865385	0.625000	0.725806
35	0.6	3.0	2	0.865385	0.625000	0.725806
36	0.2	5.0	1	0.900000	0.625000	0.737705
37	0.2	5.0	1	0.900000	0.625000	0.737705

38	0.2	5.0	1	0.956522	0.611111	0.745763
39	0.2	5.0	2	0.956522	0.611111	0.745763
40	0.4	5.0	1	0.900000	0.625000	0.737705
41	0.4	5.0	1	0.900000	0.625000	0.737705
42	0.4	5.0	1	0.956522	0.611111	0.745763
43	0.4	5.0	2	0.956522	0.611111	0.745763
44	0.6	5.0	1	0.900000	0.625000	0.737705
45	0.6	5.0	1	0.900000	0.625000	0.737705
46	0.6	5.0	1	0.956522	0.611111	0.745763
47	0.6	5.0	2	0.956522	0.611111	0.745763

Solution using Grid CV-

[28]:
```
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import precision_score, recall_score, f1_score
from nltk.corpus import stopwords

# Download NLTK stopwords (if not already downloaded)
import nltk
nltk.download('stopwords')

# Define a function to remove stopwords using NLTK
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Assuming 'train_data' is your dataset
train_data = train_data.copy()
train_data['text'] = train_data['text'].apply(remove_stopwords)

# Define the parameter grid
param_grid = {
    'vectorizer__max_df': [0.2, 0.4, 0.6],
    'classifier__alpha': [0.1, 1.0, 3.0 , 5.0],
    'vectorizer__ngram_range': [(1, 1), (1, 2)]  # Varying ngram range
}

# Create a pipeline with CountVectorizer and MultinomialNB
pipeline = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('classifier', MultinomialNB())
])
```

```
# Create a GridSearchCV object
grid_search_cv = GridSearchCV(
    pipeline, param_grid, cv=5, scoring='f1_macro', verbose=1, n_jobs=-1)

# Fit the GridSearchCV object on the training data
grid_search_cv.fit(train_data['text'], train_data['label'])

# Print the best parameters and corresponding metrics
print("Best Parameters: ", grid_search_cv.best_params_)
print("Best F1 Score: ", grid_search_cv.best_score_)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters:  {'classifier__alpha': 1.0, 'vectorizer__max_df': 0.2,
'vectorizer__ngram_range': (1, 2)}
Best F1 Score:  0.8366997663191809
```

As F1 score from Grid CV is better than parameter grid, we will go ahead with GRID CV results

#PROBLEM 5 – Model selection (10pts) Using these validation-set metrics from the previous problem, choose one model as your selected model. It is up to you how to choose this model; one approach is to choose the model that shows the highest F1-score on your training set. Next apply your selected model to your test set and compute precision, recall and F1. Show results in your notebook

[29]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# GridSearchCV and obtained the best model
selected_model = grid_search_cv.best_estimator_

# Apply the selected model to the test set
test_data = test_data.copy()
test_data['label'] = test_data['label'].astype(int)
test_predictions = selected_model.predict(test_data['text'])

# Calculate precision, recall, and F1-score on the test set
test_precision = precision_score(test_data['label'], test_predictions,␣
 ↪average='binary', pos_label=1)
test_recall = recall_score(test_data['label'], test_predictions,␣
 ↪average='binary')
test_f1_score = f1_score(test_data['label'], test_predictions, average='binary')

# Print the test metrics
print(f"Test data Precision: {test_precision:.4f}")
print(f"Test data Recall: {test_recall:.4f}")
print(f"Test F1 Score: {test_f1_score:.4f}")
```

```
Test data Precision: 0.8344
Test data Recall: 0.8792
Test F1 Score: 0.8562
```

#PROBLEM 6 – Key Indicators (10pts) Using the log-probabilities of the model you selected in the previous problem, select 5 words that are strong Clickbait indicators. That is, if you needed to filter headlines based on a single word, without a machine learning model, then these words would be good options. Show this list of keywords in your notebook. You can choose how to handle bigrams (e.g., "winbig"); you may choose to ignore them and only select unigram vocabulary words as key indicators.

[30]:
```
# Access feature log probabilities from the MultinomialNB model
feature_log_probs = selected_model.named_steps['classifier'].feature_log_prob_

#  CountVectorizer named as 'countvectorizer'
feature_names = selected_model.named_steps['vectorizer'].get_feature_names_out()

# Create a dictionary to store feature names and their associated log␣
 ↪probabilities, feature_log_probs[1][i] will access elements indexed as label␣
 ↪1 that is clickbait.
feature_probs_dict = {feature_names[i]: feature_log_probs[1][i] for i in␣
 ↪range(len(feature_names))}

# Sort the dictionary by log probabilities to get the most informative features
sorted_features_items = sorted(feature_probs_dict.items(), key=lambda x: x[1],␣
 ↪reverse=True)

# Display the top 5 informative features
top_features = sorted_features_items[:5]
print('top_features along with their log probabilities',top_features)

top_words = [item[0] for item in sorted_features_items[:5]]
print('5 top Clickbait indicating words -',top_words)
```

```
top_features along with their log probabilities [('believe',
-6.213260251709267), ('one', -6.276439153330799), ('never', -6.578720025203732),
('here', -6.608572988353414), ('new', -6.846984011798412)]
5 top Clickbait indicating words - ['believe', 'one', 'never', 'here', 'new']
```

#PROBLEM 7 – Regular expressions (10pts) Your IT department has reached out to you because they heard you can help them find clickbait. They are interested in your machine learning model, but they need a solution today. • Write a regular expression that checks if any of the keywords from the previous problem are found in the text. You should write one regular expression that detects any of your top 5 keywords. Your regular expression should be aware of word boundaries in some way. That is, the keyword "win" should not be detected in the text "Gas prices up in winter months". • Using the python re library – apply your function to your test set. (See function re.search). What is the precision and recall of this classifier? Show your results in your notebook

[31]:
```
import re

# Define the regular expression pattern
# pattern = r'\b(?:' + '|'.join(top_words) + r')\b'

pattern = r'\b(?:' + '|'.join(re.escape(keyword) for keyword in top_words) +␣
 ↪r')\b'

# function to classify the text is based on  pattern
def clickbait_dataset_with_regex_classifier(text):
    return bool(re.search(pattern, text, flags=re.IGNORECASE))

#  function to the test dataset
test_data = test_data.copy()
test_data.loc[:, 'clickbait_predict'] = test_data['text'].
 ↪apply(clickbait_dataset_with_regex_classifier)

# Calculat the precision and recall for test dataset
tp = len(test_data[(test_data['clickbait_predict'] == 1) & (test_data['label']␣
 ↪== 1)])
fp = len(test_data[(test_data['clickbait_predict'] == 1) & (test_data['label']␣
 ↪== 0)])
fn = len(test_data[(test_data['clickbait_predict'] == 0) & (test_data['label']␣
 ↪== 1)])

precision = tp / (tp + fp)
recall = tp / (tp + fn)


print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

```
Precision: 0.7333
Recall: 0.2953
```