CS 585 – Fall 2023 – Homework 2

In [1]:
```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
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
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import precision_score, recall_score
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
#Given datasets
Positive_examples = 'https://github.com/pfrcks/clickbait-detection/raw/master/clickbait'
Negative_examples = 'https://github.com/pfrcks/clickbait-detection/raw/master/not-clickbait'
```

PROBLEM 1 – Reading the data

In [3]:
```
pos_clickbait = pd.read_csv(Positive_examples, sep='\t', names=['text'])
pos_clickbait['label'] = 1
#here for positive clickbait data added labels 1's
neg_clickbait = pd.read_csv(Negative_examples, sep='\t', names=['text'])
neg_clickbait['label'] = 0
#And also for negative clickbait  added labels 0's .
# neg_clickbait
# pos_clickbait
```

In [4]:
```
#combning both datasets(positive and negative clickbaits)
combined_pos_neg_dataset = pd.concat([pos_clickbait, neg_clickbait], ignore_index=True)
#shuffle the combined datset
#first converting dataframe into numpy array
np_array = combined_pos_neg_dataset.values
np.random.shuffle(np_array)   #shuffling numpy array
shuffled_dataset = pd.DataFrame(np_array, columns=combined_pos_neg_dataset.columns)#converting back to dataframe from num
shuffled_dataset
```

Out[4]:

	text	label
0	'Selfie' of Brunel looking fed up on a train g...	1
1	George Clooney seeks to expose those who fund ...	0
2	Health Care Fraud Takedown	0
3	This Behavior Is The #1 Predictor Of Divorce, ...	1
4	Plans to stop collecting data on wealthiest 1%...	0
...	...	...
2383	Goldman Sachs Finally Admits it Defrauded Inve...	0
2384	Anthony Bourdain Says This Kitchen Staple Is a...	1
2385	Lenovo caught installing adware on new computers	0
2386	Australia to penalize parents who don't vaccin...	0
2387	Doubts rise over TTIP as France threatens to b...	0

2388 rows × 2 columns

In [5]:

```
#splitting the train and test data into 80% train and 20% test
train_clickbait_dataset, test_clickbait_dataset = train_test_split(shuffled_dataset, test_size=0.20, random_state=18)
# now further split the training data into 90% train and 10% validation sets
train_clickbait_dataset, validation_clickbait_dataset = train_test_split(train_clickbait_dataset, test_size=0.10, random_
# test_clickbait_dataset
# train_clickbait_dataset.to_csv('train_dataset.csv', index=False)
# validation_clickbait_dataset.to_csv('validation_dataset.csv', index=False)
# test_clickbait_dataset.to_csv('test_dataset.csv', index=False)
```

In [6]:

```
#here calculating the  "target rate" of these three datasets in percentage.

train_target_rate = (train_clickbait_dataset['label'] == 1).mean() * 100
validation_target_rate = (validation_clickbait_dataset['label'] == 1).mean() * 100
test_target_rate = (test_clickbait_dataset['label'] == 1).mean() * 100

# printing the "Target Rate" for three datasets
print(f"The training dataset Target Rate: {train_target_rate:.3f}%")
```

```
print(f"The validation dataset Target Rate: {validation_target_rate:.3f}%")
print(f"The testing dataset Target Rate: {test_target_rate:.3f}%")
```

```
The training dataset Target Rate: 33.624%
The validation dataset Target Rate: 39.791%
The testing dataset Target Rate: 33.473%
```

PROBLEM 2 – Baseline Performance

For the trivial baseline classifier that marks all texts as clickbait:

Precision is determined by dividing the number of genuine clickbait samples by the total number of samples in the test dataset. The precision is determined as follows because the model classifies everything as clickbait: Precision is defined as True Positives (TP) / True Positives (TP) + False Positives (FP) and is equal to 38 / (38 + 54) 0.413.

Recall: The proportion of authentic clickbait samples to all clickbait samples is known as recall. As all actual clickbait samples were accurately classified by the classifier in this instance, the recall is perfect: True Positives (TP) / (TP + FN) = 38 / (38 + 0) = 1 is the formula for recall.

F1 Score: The F1 score is a balanced measurement of recall and precision and is calculated as the harmonic mean of both: F1 Score = 2 *((Precision* Recall) / (Precision + Recall)) = 2 *((0.413* 1) / (0.413 + 1))) = 2 *((0.413* 1) / (0.413 + 1)))=0.584

Therefore, the precision, recall, and F1 score for the basic baseline classifier that classifies all texts as clickbait are around 0.413, 1, and 0.584, respectively.

PROBLEM 3 – Training a single Bag-of-Words (BOW) Text Classifier

In [11]:

```
# Creating a pipeline to train a model (MultinomialNB)
label_encoder = LabelEncoder()
label_train_encoded = label_encoder.fit_transform(train_clickbait_dataset['label'])  # Modified this line
label_validation_encoded = label_encoder.transform(validation_clickbait_dataset['label'])

pipeline = Pipeline([
    ('countvectorizer', CountVectorizer(ngram_range=(1, 2))),
    ('classifier', MultinomialNB())
])
text_train = train_clickbait_dataset['text']
text_validation = validation_clickbait_dataset['text']

# Fitting the classifier on the training data
pipeline.fit(text_train, label_train_encoded)
```

```
label_train_pred = pipeline.predict(text_train)
label_validation_pred = pipeline.predict(text_validation)

# Now compute precision, recall, F1-score
train_precision = precision_score(label_train_encoded, label_train_pred)
validation_precision = precision_score(label_validation_encoded, label_validation_pred)
train_recall = recall_score(label_train_encoded, label_train_pred)
validation_recall = recall_score(label_validation_encoded, label_validation_pred)
train_f1_score = f1_score(label_train_encoded, label_train_pred)
validation_f1 = f1_score(label_validation_encoded, label_validation_pred)
```

In [12]:

```
import pandas as pd
#Results for training and validation datasets
train_val_metrics = pd.DataFrame({
    'Metric': ['Precision', 'Recall', 'F1-score'],
    'Training data': [train_precision, train_recall, train_f1_score],
    'Validation data': [validation_precision, validation_recall, validation_f1]
})

print("These are Results(Metrics) for Training and Validation Sets:")
print(train_val_metrics)
```

```
These are Results(Metrics) for Training and Validation Sets:
      Metric  Training data  Validation data
0  Precision       0.989708         0.929577
1     Recall       0.998270         0.868421
2   F1-score       0.993971         0.897959
```

PROBLEM 4

In [16]:

```
#Hyperparameter grid
hp_grid = {'ngram_range': [(1,1), (1,2)],'alpha_clasifier':[1.0,1.0,5.0],'max_df_countvectorizer':[0.9,0.9,0.9]}
```

In [18]:

```
score_precision_metrics, score_recall_metrics, score_f1_metrics = [], [], []
label_encoder = LabelEncoder()
label_train_encoded = label_encoder.fit_transform(train_clickbait_dataset['label'])
label_validation_encoded = label_encoder.transform(validation_clickbait_dataset['label'])

for p in ParameterGrid(hp_grid):
    pipeline = Pipeline([
        ('countvectorizer', CountVectorizer(ngram_range=p['ngram_range'], max_df=p['max_df_countvectorizer'])),
        ('classifier', MultinomialNB(alpha=p['alpha_clasifier']))])
```

```
    pipeline.fit(text_train, label_train_encoded)
    val_label_pred = pipeline.predict(text_validation)
    # Metrics calculation on validation data
    precision_metrics = precision_score(label_validation_encoded, val_label_pred)
    recall_metrics = recall_score(label_validation_encoded, val_label_pred)
    f1_val_metrics = f1_score(label_validation_encoded, val_label_pred)
    # Scores appending into lists
    score_precision_metrics.append(precision_metrics)
    score_recall_metrics.append(recall_metrics)
    score_f1_metrics.append(f1_val_metrics)
# Creating a DataFrame to display results
grid_search_results = pd.DataFrame({
    'Ngram_Range': [params['ngram_range'] for params in ParameterGrid(hp_grid)],
    'Alpha': [params['alpha_clasifier'] for params in ParameterGrid(hp_grid)],
    'Max_DF': [params['max_df_countvectorizer'] for params in ParameterGrid(hp_grid)],
    # Add more columns for your additional hyperparameters here
    'Precision': score_precision_metrics,
    'Recall': score_recall_metrics,
    'F1-score': score_f1_metrics })
# Sort the results by F1-score in descending order
grid_search_results.sort_values(by='F1-score', ascending=False, inplace=True)
# Display the top and bottom results
good_grid_search_results = grid_search_results.head(5)  # only taking top 5 good results
poor_grid_search_results = grid_search_results.tail(5)  # only taking last 5 poor results
# Printing the results
# The lowest results are the results having poor f1_score.
print("\nLowest Results:")
print(poor_grid_search_results)
# The highest results are the results having high f1_score and can be considered as a better model.
print("Highest Results:")
print(good_grid_search_results)
```

```
Lowest Results:
   Ngram_Range  Alpha  Max_DF  Precision    Recall  F1-score
7       (1, 2)    1.0     0.9   0.929577  0.868421  0.897959
11      (1, 2)    1.0     0.9   0.929577  0.868421  0.897959
5       (1, 2)    1.0     0.9   0.929577  0.868421  0.897959
3       (1, 2)    1.0     0.9   0.929577  0.868421  0.897959
9       (1, 2)    1.0     0.9   0.929577  0.868421  0.897959
Highest Results:
   Ngram_Range  Alpha  Max_DF  Precision    Recall  F1-score
17      (1, 2)    5.0     0.9   0.969697  0.842105  0.901408
16      (1, 1)    5.0     0.9   0.969697  0.842105  0.901408
15      (1, 2)    5.0     0.9   0.969697  0.842105  0.901408
```

```
14      (1, 1)    5.0     0.9    0.969697  0.842105  0.901408
13      (1, 2)    5.0     0.9    0.969697  0.842105  0.901408
```

PROBLEM 5 – Model selection

In [22]:

```
print(train_clickbait_dataset['label'].values)
```

```
[0 0 0 ... 0 0 0]
```

In [24]:

```
# Testing new model on the test set
text_test = test_clickbait_dataset['text']
label_test = test_clickbait_dataset['label']
# using the best f1 score hyperparameters for using in selected model
new_model_ngram = model_params_selected['Ngram_Range']
new_model_alpha = model_params_selected['Alpha']
new_model_max_df = model_params_selected['Max_DF']
# Now creating new model's pipeline
model_new_pipeline = Pipeline([
    ('countvectorizer', CountVectorizer(ngram_range=new_model_ngram, max_df=new_model_max_df)),
    ('classifier', MultinomialNB(alpha=new_model_alpha))
])
# fitting training data on the new model
model_new_pipeline.fit(text_train, train_clickbait_dataset['label'].values)  # Ensure 'label' is a 1D array
# predict (pipeline on test data)
test_pred_label = model_new_pipeline.predict(text_test)
# Extract labels from the test_clickbait_dataset as a 1D array
label_test = test_clickbait_dataset['label'].astype(int).values
# Now, you can calculate precision, recall, and F1-score
precision_test_data = precision_score(label_test, test_pred_label)
recall_test_data = recall_score(label_test, test_pred_label)
f1_score_test_data = f1_score(label_test, test_pred_label)
# These are the resultant metrics after applying the best model on test data
print("New model Metrics")
print(f"Precison: {precision_test_data:.3f}")
print(f"Recall: {recall_test_data:.3f}")
print(f"F1_score: {f1_score_test_data:.3f}")
```

```
New model Metrics
Precison: 0.933
Recall: 0.787
F1_score: 0.854
```

PROBLEM 6 – Key Indicators

In [25]:

```
# the clickbait_log_probs contains the log probabulities of clickbait
clickbait_log_probs = model_new_pipeline.named_steps['classifier'].feature_log_prob_[1]
# the not_clickbait_log_probs contains the not_clickbait probabulities.
not_clickbait_log_probs = model_new_pipeline.named_steps['classifier'].feature_log_prob_[0]
#Here we are doing difference between both positive and negative calsses of clickbait
log_probs_diff = clickbait_log_probs - not_clickbait_log_probs
# Gatheing feature names from the vectorizer
feature_names = model_new_pipeline.named_steps['countvectorizer'].get_feature_names()
# Storing log probs difference
word_log_probs_diff = dict(zip(feature_names, log_probs_diff))
# Sorting the words by log-probability (desecending order)
words_sorted = sorted(word_log_probs_diff.items(), key=lambda x: x[1], reverse=True)
# Select the top 5 words with the highest log-probability differences
strong_clickbait_indicators = [word for word, _ in words_sorted[:5]]
# Printing the words has highest difference
print("Strong 5 Clickbait Indicators are:")
for k_words in strong_clickbait_indicators:
    print(k_words)
```

```
Strong 5 Clickbait Indicators are:
you
this
believe
won believe
you won
```

PROBLEM 7 – Regular expressions

In [26]:

```
# Regular expression
tp, fp, fn = 0, 0, 0
pattern = r'\b(?:' + '|'.join(re.escape(w) for w in strong_clickbait_indicators) + r')\b'
def having_clickbait(indi_words):
    return bool(re.search(pattern, indi_words, re.IGNORECASE))
pred_labels = [1 if having_clickbait(z) else 0 for z in text_test]
# True Positive, False Positive, False Negative Calculation
tp = sum((pred == 1 and G_truth == 1) for pred, G_truth in zip(pred_labels, label_test))
fn = sum((pred == 0 and G_truth == 1) for pred, G_truth in zip(pred_labels, label_test))
fp = sum((pred == 1 and G_truth == 0) for pred, G_truth in zip(pred_labels, label_test))
# Computed tp, fn, fp; now using them to calculate Precision and Recall
re_precision = tp / (tp + fp)
re_recall = tp / (tp + fn)
print("Precision:", re_precision)
print("Recall:", re_recall)
```

```
Precision: 0.9178082191780822
Recall: 0.41875
```

PROBLEM 8 – Comparing results

a)The precision and recall scores for the rule-based classifier were 0.91 and 0.41, respectively, whereas the precision and recall scores for the machine learning model utilizing grid search were somewhat higher at 0.93 and 0.78. The chance of a text falling into each class is estimated by naive bayes. In order to determine the posterior probability of each class, the Bayes theorem is used to determine the likelihood of observing words that belong to a certain class. For the rule-based classifier, we use log probabilities to identify the most effective keywords. These are merely probability in a logarithmic scale, once more. Whether or not these words are present in the input text determines whether the classification is correct. Regular expressions are not capable of generalizing to fresh data.
Random guessing plays a significant role in the performance of the trivial baseline classifier.

b)Use an ensemble model to classify the text: Ensemble models are effective classifiers that combine the results of multiple models. Using neural networks, we can model sequential dependencies in text data using RNNs that are good for text classification.We may be able to convert the words to numerical format and use feature extraction techniques like TF-IDF and bag of words to enable ML models to make predictions based on the frequencies. Word embeddings are yet another technique that can be used to capture context and semantic relationships. Text processing techniques like tokenization, stemming, and lemmatization can help us decrease the dimensions and noise and transform them to structured format that ML models can understand. For instance, word2Vec, GloVe, and so on.

In [ ]: