HW2_A20439949_Soutonglang

November 12, 2023

1 CS 585 - Homework 2

Tania Soutonglang A20439949

[1]:
```
# imports
import pandas as pd
import re

from sklearn.model_selection import train_test_split, ParameterGrid
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import precision_score, recall_score, f1_score
```

1.1 Problem 1 - Reading the Data

- 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)

[2]:
```
def readin(file, label):
    # read in file
    file_text = open(file, 'r', encoding='utf-8')
    file_list = file_text.readlines()
    file_list = [(x.strip('\n')) for x in file_list]
    # cleaned_text = text.rstrip('\n')

    # add labels
    file_list = [(x.lower(), label) for x in file_list]

    # turn into dataframe
    file_df = pd.DataFrame(file_list, columns = ["sentence", "label"])

    return file_df
```

[3]:
```
clickbait_df = readin('clickbait.txt', 1)
print("clickbait")
print(clickbait_df.head())
```

```
print()

notclickbait_df = readin('not-clickbait.txt', 0)
print("not clickbait")
print(notclickbait_df.head())
```

```
clickbait
                                            sentence  label
0  man repairs fence to contain dog, hilarity ens…      1
1  long-term marijuana use has one crazy side eff…      1
2  the water from his ear trickles into the bucke…      1
3  you'll never guess what nick jonas does in the…      1
4  how cruise liners fill all their unsold cruise…      1

not clickbait
                                            sentence  label
0  congress slips cisa into a budget bill that's …      0
1                       dui arrest sparks controversy      0
2  it’s unconstitutional to ban the homeless from…      0
3  a government error just revealed snowden was t…      0
4  a toddler got meningitis. his anti-vac parents…      0
```

[4]:
```
all_df = pd.concat([clickbait_df, notclickbait_df])
all_df = all_df.sample(frac = 1, random_state = 42).reset_index(drop = True)
all_df.head()
```

[4]:

	sentence	label
0	18 celebrities who might be time travelers	1
1	in chhattisgarh, pm modi touches the feet of 1…	0
2	n.j. woman jailed for tossing neighbor's dog i…	0
3	us releases guantánamo prisoner after 14 years…	0
4	the best buzzer-beater of the weekend miiiight…	1

- 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).
  - If you prefer, you may save each split as an index (list of row numbers) rather than creating 3 separate datasets.

[5]:
```
n_texts = len(all_df)

train_df, test_df = train_test_split(all_df, test_size = 0.2)
train_df, val_df = train_test_split(train_df, test_size = 0.1)

print(f"Train size:  {len(train_df)} -> {len(train_df)/n_texts:0.1%}")
print(f"Test size:   {len(test_df)} -> {len(test_df)/n_texts:0.1%}")
print(f"Validation size: {len(val_df)} -> {len(val_df)/n_texts:0.1%}")
```

```
Train size:  1719 -> 72.0%
```

```
Test size:   478 -> 20.0%
Validation size: 191 -> 8.0%
```

- 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.

[6]:
```
print(f"Train size: {len(train_df)}, clickbait: {sum(train_df['label'] == 1)}␣
↪-> {sum(train_df['label'] == 1)/len(train_df):0.1%} target rate")
print(f"Test size: {len(test_df)}, clickbait: {sum(test_df['label'] == 1)} ->␣
↪{sum(test_df['label'] == 1)/len(test_df):0.1%} target rate")
print(f"Validation size: {len(val_df)}, clickbait: {sum(val_df['label'] == 1)}␣
↪-> {sum(val_df['label'] == 1)/len(val_df):0.1%} target rate")
```

```
Train size: 1719, clickbait: 593 -> 34.5% target rate
Test size: 478, clickbait: 145 -> 30.3% target rate
Validation size: 191, clickbait: 76 -> 39.8% target rate
```

1.2 Problem 2 - Baseline Performance

- Assume you have a trivial baseline classifier that flags every text presented to it as clickbait. What is the precision, recall, and F1-score of such a classifier on your test set? Do you think there is another good baseline classifier that would give you higher F-1 score?

Test size: 478, clickbait: 159

True positive: 159 False positive: 319 False negative: 0

There probably is another good baseline classifier that would give a higher F-1 score. This one flags all items as clickbait, making all the items that are not clickbait a false positive and then making the precision score low.

[7]:
```
precision = 159/(159 + 319)
recall = 159/(159 + 0)
f1 = 2 * ((precision * recall)/(precision + recall))

print("Precision: ", precision)
print("Recall: ", recall)
print("F1-Score: ", f1)
```

```
Precision:  0.33263598326359833
Recall:  1.0
F1-Score:  0.49921507064364207
```

1.3 Problem 3 - Training a Single Bag-of-Words (BOW) Text Classifier

- 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)

[8]:
```
def fit_pipeline(*, texts, labels, min_df = 1, max_df = 0.1, ngram_range =␣
↪(1,1), alpha = 1.0):
```

```python
    """ Train a text classifier model given input hyperparameters:
      - CountVectorizer: min_df, max_df, ngram_range
      - NaiveBayes:      alpha
    """

    # Pipeline Step 1: texts -> BOW vectors
    vectorizer = CountVectorizer(min_df = min_df,
                                 max_df = max_df,
                                 stop_words = "english",
                                 ngram_range = ngram_range)

    # Pipeline Step 2: document vectors -> model score
    model = MultinomialNB(alpha = alpha)

    pipeline = Pipeline(steps = [
        ("vectorizer",vectorizer),
        ("classifier",model)
    ])

    pipeline.fit(texts,labels)

    return pipeline
```

- Fit your classifier on your training set

[9]:
```python
X_train = train_df.sentence.values
y_train = train_df.label.values

pipeline = fit_pipeline(texts = X_train, labels = y_train)

y_pred_train = pipeline.predict(X_train)

pipeline
```

[9]:
```
Pipeline(steps=[('vectorizer',
                 CountVectorizer(max_df=0.1, stop_words='english')),
                ('classifier', MultinomialNB())])
```

[10]:
```python
X_val = val_df.sentence.values
y_val = val_df.label.values

y_pred_val = pipeline.predict(X_val)
```

- 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)

[11]:

```
train_precision = precision_score(y_train, y_pred_train)
train_recall = recall_score(y_train, y_pred_train)
train_f1 = f1_score(y_train, y_pred_train)

print("training set")
print(f"Precision: {train_precision:.2f}")
print(f"Recall:    {train_recall:.2f}")
print(f"F1:        {train_f1:.2f}")

val_precision = precision_score(y_val, y_pred_val)
val_recall = recall_score(y_val, y_pred_val)
val_f1 = f1_score(y_val, y_pred_val)

print("\nvalidation set")
print(f"Precision: {val_precision:.2f}")
print(f"Recall:    {val_recall:.2f}")
print(f"F1:        {val_f1:.2f}")
```

```
training set
Precision: 0.98
Recall:    0.96
F1:        0.97

validation set
Precision: 0.89
Recall:    0.74
F1:        0.81
```

1.4 Problem 4 - Hyperparameter Tuning

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)

[12]:

```
param_grid = ParameterGrid({
    "max_df":       [0.01, 0.1],
    'alpha':        [0.1, 0.5, 1.0, 2.0],
    "ngram_range":  [(1,1),(1,2),(1,3)],
})

def get_metrics(pipeline, texts, labels):
    preds = pipeline.predict(texts)

    pr = precision_score(labels, preds)
    re = recall_score(labels, preds)
    f1 = f1_score(labels, preds)
```

```python
    return {
        "precision": pr,
        "recall": re,
        "f1": f1
    }
```

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.

[13]:
```python
print(f"# of grid points: {len(param_grid)}")

results_arr = []

for gridpt in param_grid:
    print(gridpt)
    trained = fit_pipeline(texts=X_train, labels=y_train, **gridpt)
    metrics = get_metrics(trained, X_val, y_val)

    # save hyperparams and results
    combined_data = {**gridpt, **metrics, "trained":trained}
    # check for overfitting
    combined_data['f1_train'] = f1_score(y_train, trained.predict(X_train))

    # vocab size
    combined_data['K'] = len(trained[0].vocabulary_)

    results_arr.append(combined_data)

results_df = pd.DataFrame(results_arr).sort_values("f1", ascending=False)
results_df.drop("trained",axis=1)
```

```
# of grid points: 24
{'alpha': 0.1, 'max_df': 0.01, 'ngram_range': (1, 1)}
{'alpha': 0.1, 'max_df': 0.01, 'ngram_range': (1, 2)}
{'alpha': 0.1, 'max_df': 0.01, 'ngram_range': (1, 3)}
{'alpha': 0.1, 'max_df': 0.1, 'ngram_range': (1, 1)}
{'alpha': 0.1, 'max_df': 0.1, 'ngram_range': (1, 2)}
{'alpha': 0.1, 'max_df': 0.1, 'ngram_range': (1, 3)}
{'alpha': 0.5, 'max_df': 0.01, 'ngram_range': (1, 1)}
{'alpha': 0.5, 'max_df': 0.01, 'ngram_range': (1, 2)}
{'alpha': 0.5, 'max_df': 0.01, 'ngram_range': (1, 3)}
{'alpha': 0.5, 'max_df': 0.1, 'ngram_range': (1, 1)}
{'alpha': 0.5, 'max_df': 0.1, 'ngram_range': (1, 2)}
{'alpha': 0.5, 'max_df': 0.1, 'ngram_range': (1, 3)}
{'alpha': 1.0, 'max_df': 0.01, 'ngram_range': (1, 1)}
{'alpha': 1.0, 'max_df': 0.01, 'ngram_range': (1, 2)}
{'alpha': 1.0, 'max_df': 0.01, 'ngram_range': (1, 3)}
```

```
{'alpha': 1.0, 'max_df': 0.1, 'ngram_range': (1, 1)}
{'alpha': 1.0, 'max_df': 0.1, 'ngram_range': (1, 2)}
{'alpha': 1.0, 'max_df': 0.1, 'ngram_range': (1, 3)}
{'alpha': 2.0, 'max_df': 0.01, 'ngram_range': (1, 1)}
{'alpha': 2.0, 'max_df': 0.01, 'ngram_range': (1, 2)}
{'alpha': 2.0, 'max_df': 0.01, 'ngram_range': (1, 3)}
{'alpha': 2.0, 'max_df': 0.1, 'ngram_range': (1, 1)}
{'alpha': 2.0, 'max_df': 0.1, 'ngram_range': (1, 2)}
{'alpha': 2.0, 'max_df': 0.1, 'ngram_range': (1, 3)}
```

[13]:

	alpha	max_df	ngram_range	precision	recall	f1	f1_train	K
9	0.5	0.10	(1, 1)	0.876923	0.750000	0.808511	0.983051	5253
15	1.0	0.10	(1, 1)	0.888889	0.736842	0.805755	0.970213	5253
3	0.1	0.10	(1, 1)	0.888889	0.736842	0.805755	0.990733	5253
10	0.5	0.10	(1, 2)	0.863636	0.750000	0.802817	0.997473	15635
11	0.5	0.10	(1, 3)	0.875000	0.736842	0.800000	0.999156	24840
4	0.1	0.10	(1, 2)	0.848485	0.736842	0.788732	1.000000	15635
5	0.1	0.10	(1, 3)	0.835821	0.736842	0.783217	1.000000	24840
7	0.5	0.01	(1, 2)	0.868852	0.697368	0.773723	0.998314	15577
16	1.0	0.10	(1, 2)	0.866667	0.684211	0.764706	0.997473	15635
8	0.5	0.01	(1, 3)	0.852459	0.684211	0.759124	0.999156	24782
17	1.0	0.10	(1, 3)	0.877193	0.657895	0.751880	0.997473	24840
21	2.0	0.10	(1, 1)	0.890909	0.644737	0.748092	0.957649	5253
13	1.0	0.01	(1, 2)	0.862069	0.657895	0.746269	0.997473	15577
6	0.5	0.01	(1, 1)	0.836066	0.671053	0.744526	0.981356	5197
14	1.0	0.01	(1, 3)	0.875000	0.644737	0.742424	0.998314	24782
23	2.0	0.10	(1, 3)	0.957447	0.592105	0.731707	0.997473	24840
1	0.1	0.01	(1, 2)	0.796875	0.671053	0.728571	1.000000	15577
12	1.0	0.01	(1, 1)	0.830508	0.644737	0.725926	0.970162	5197
2	0.1	0.01	(1, 3)	0.784615	0.671053	0.723404	1.000000	24782
20	2.0	0.01	(1, 3)	0.918367	0.592105	0.720000	0.998314	24782
22	2.0	0.10	(1, 2)	0.936170	0.578947	0.715447	0.994941	15635
0	0.1	0.01	(1, 1)	0.803279	0.644737	0.715328	0.987342	5197
19	2.0	0.01	(1, 2)	0.882353	0.592105	0.708661	0.996627	15577
18	2.0	0.01	(1, 1)	0.849057	0.592105	0.697674	0.951389	5197

1.5 Problem 5 - Model Selection

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.

[14]:
```
selected_pipeline = results_df.loc[15,"trained"]
selected_pipeline
```

[14]:
```
Pipeline(steps=[('vectorizer',
                 CountVectorizer(max_df=0.1, stop_words='english')),
                ('classifier', MultinomialNB())])
```

[15]:
```
print("Train metrics", get_metrics(
    selected_pipeline,
    texts=X_train,
    labels=y_train)
)

print("Valid metrics", get_metrics(
    selected_pipeline,
    texts=X_val,
    labels=y_val)
)
```

```
Train metrics {'precision': 0.979381443298969, 'recall': 0.9612141652613828,
'f1': 0.9702127659574468}
Valid metrics {'precision': 0.8888888888888888, 'recall': 0.7368421052631579,
'f1': 0.8057553956834532}
```

Next apply your selected model to your test set and compute precision, recall and F1. Show results in your notebook

[16]:
```
X_test = test_df.sentence.values
y_test = test_df.label.values

y_pred_test = selected_pipeline.predict(X_test)

test_precision = precision_score(y_test, y_pred_test)
test_recall = recall_score(y_test, y_pred_test)
test_f1 = f1_score(y_test, y_pred_test)

print(f"Precision: {test_precision:.2f}")
print(f"Recall:    {test_recall:.2f}")
print(f"F1:        {test_f1:.2f}")
```

```
Precision: 0.80
Recall:    0.75
F1:        0.77
```

1.6 Problem 6 - Key Indicators

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.

[17]:
```
selected_vect = selected_pipeline[0]
```

```
vocab = sorted([v for v in selected_vect.vocabulary_.keys()], key=selected_vect.
    ↪vocabulary_.get)
print(len(vocab))

selected_model = selected_pipeline[1]
log_probs = selected_model.feature_log_prob_[1:].reshape(-1)
v_to_logprob = {v:score for v,score in zip(vocab,log_probs)}

top_vocab = sorted(vocab, key=v_to_logprob.get, reverse=True)[:5]
print(top_vocab)
```

```
5253
['believe', 'won', 'll', 'new', 'did']
```

1.7 Problem 7 - Regular Expressions

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.

[18]:
```
pattern = r'\b(?:' + '|'.join(re.escape(keyword) for keyword in top_vocab) +␣
    ↪r')\b'

# apply regex to the test set
matched = [i for i, text in enumerate(X_test) if re.search(pattern, text)]

tp = sum(y_test[i] == 1 for i in matched)
fp = len(matched) - tp
fn = sum(y_test == 1) - tp

# calculate precision and recall
precision = tp / (tp + fp)
recall = tp / (tp + fn)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

```
Precision: 0.86
Recall: 0.17
```

1.8 Problem 8 - Comparing Results

- Compare your rules-based classifier from the previous problem with your machine-learning solution. Which classifier showed the best model metrics? Why do you think it performed the best? How did both compare to your trivial baseline (Problem 2)?

Problem 5: Precision: 0.83 Recall: 0.75 Problem 8: Precision: 0.73 Recall: 0.20

- If you had more time to try to improve this clickbait detection solution, what would you explore? (There is no single correct answer to this question; review your results and come up with your own ideas)

The machine learning model from problem 5 had better precision and recall scores compared to the classifier from problem 8. I think that performed better because it was able to pick up patterns that the rules classifier may have missed. If I had more time I would try to see if preprocessing the headlines even further would improve anything. I only did lowercasing, but I want to see if stemming, lemmatization, and/or removing stop words would help improve it.

[ ]:

```

```