Soutonglang_CS585_Homework02

September 27, 2023

1 CS 585 - Homework 2

Tania Soutonglang A20439949

[ ]:
```
# 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

read in clickbait.txt file

[ ]:
```
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
```

[ ]:
```
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
```

concatenating and shuffling the datasets

[ ]:
```
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()
```

[ ]:
```
                                             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
```

split into train, test, and validation datasets 72% train, 8% validation, 20% test

[ ]:
```
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%
```

finding the “target rate”

[ ]:
```
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: 597 -> 34.7% target rate
Test size: 478, clickbait: 159 -> 33.3% target rate
Validation size: 191, clickbait: 58 -> 30.4% target rate
```

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

(taken from Scikit-learn Intro.ipynb)

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

train model

[ ]:
```
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
```

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

predict using the trained model

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

y_pred_val = pipeline.predict(X_val)
```

the precision, recall, and f1-score on the training and validation datasets

[ ]:
```
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: 0.91
Recall:    0.71
F1:        0.80
```

1.3 Problem 4 - Hyperparameter Tuning

(taken from Scikit-learn Intro.ipynb)

[ ]:
```
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)

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

[ ]:
```
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)}
```

[ ]:

	alpha	max_df	ngram_range	precision	recall	f1	f1_train	K
15	1.0	0.10	(1, 1)	0.911111	0.706897	0.796117	0.967797	5213
10	0.5	0.10	(1, 2)	0.930233	0.689655	0.792079	0.996644	15527
11	0.5	0.10	(1, 3)	0.930233	0.689655	0.792079	0.996644	24690
4	0.1	0.10	(1, 2)	0.891304	0.706897	0.788462	0.998325	15527
5	0.1	0.10	(1, 3)	0.891304	0.706897	0.788462	0.998325	24690
9	0.5	0.10	(1, 1)	0.909091	0.689655	0.784314	0.984020	5213
16	1.0	0.10	(1, 2)	0.928571	0.672414	0.780000	0.995809	15527
17	1.0	0.10	(1, 3)	0.928571	0.672414	0.780000	0.996644	24690
3	0.1	0.10	(1, 1)	0.888889	0.689655	0.776699	0.990764	5213
21	2.0	0.10	(1, 1)	0.950000	0.655172	0.775510	0.956971	5213
1	0.1	0.01	(1, 2)	0.906977	0.672414	0.772277	0.998325	15470
2	0.1	0.01	(1, 3)	0.906977	0.672414	0.772277	0.998325	24633
14	1.0	0.01	(1, 3)	0.973684	0.637931	0.770833	0.996644	24633
8	0.5	0.01	(1, 3)	0.948718	0.637931	0.762887	0.997485	24633
7	0.5	0.01	(1, 2)	0.948718	0.637931	0.762887	0.997485	15470
13	1.0	0.01	(1, 2)	0.948718	0.637931	0.762887	0.995809	15470
12	1.0	0.01	(1, 1)	0.904762	0.655172	0.760000	0.964407	5159
22	2.0	0.10	(1, 2)	0.972973	0.620690	0.757895	0.993277	15527
23	2.0	0.10	(1, 3)	0.972973	0.620690	0.757895	0.996644	24690
0	0.1	0.01	(1, 1)	0.883721	0.655172	0.752475	0.988215	5159
6	0.5	0.01	(1, 1)	0.902439	0.637931	0.747475	0.980688	5159
18	2.0	0.01	(1, 1)	0.972222	0.603448	0.744681	0.953608	5159
19	2.0	0.01	(1, 2)	1.000000	0.551724	0.711111	0.995809	15470
20	2.0	0.01	(1, 3)	1.000000	0.551724	0.711111	0.996644	24633

1.4 Problem 5 - Model Selection

(taken from Scikit-learn Intro.ipynb)

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

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

[ ]:
```
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': 0.9794168096054888, 'recall': 0.9564489112227805,
'f1': 0.9677966101694915}
Valid metrics {'precision': 0.9111111111111111, 'recall': 0.7068965517241379,
'f1': 0.7961165048543689}
```

use the selected model on the test dataset

[ ]:
```
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: 0.83
Recall:    0.75
F1:        0.79
```

1.5 Problem 6 - Key Indicators

(taken from Scikit-learn Intro.ipynb)

[ ]:
```

```

```
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)
```

```
5213
['believe', 'won', 'll', 'new', 'guess']
```

1.6 Problem 7 - Regular Expressions

[ ]:
```
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: 0.73
Recall: 0.20
```

[ ]:
```

```