oblig_in4080_del3

September 16, 2022

[ ]:
```
import nltk
import random
import numpy as np
import scipy as sp
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
```

[ ]:
```
from nltk.corpus import movie_reviews
nltk.download('movie_reviews')
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data…
[nltk_data]   Unzipping corpora/movie_reviews.zip.
```

[ ]:
```
True
```

[ ]:
```
raw_movie_docs = [
    (movie_reviews.raw(fileid), category) for
    category in movie_reviews.categories() for fileid in
    movie_reviews.fileids(category)
]
```

[ ]:
```
random.seed(2920)
random.shuffle(raw_movie_docs)
movie_test = raw_movie_docs[:200]
movie_dev = raw_movie_docs[200:]
```

[ ]:
```
from sklearn.model_selection import train_test_split
```

[ ]:
```
train_data, dev_test_data = train_test_split(movie_dev, train_size=1600,␣
 ↪random_state=0)
```

[ ]:
```
train_texts = [t[0] for t in train_data]
train_target = [t[1] for t in train_data]
```

[ ]:
```
dev_test_texts = [t[0] for t in dev_test_data]
dev_test_target = [t[1] for t in dev_test_data]
```

[ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

[ ]:
```
v = CountVectorizer()
v.fit(train_texts)
```

[ ]:
```
CountVectorizer()
```

[ ]:
```
train_vectors = v.transform(train_texts)
dev_test_vectors = v.transform(dev_test_texts)
```

[ ]:
```
clf = MultinomialNB()
clf.fit(train_vectors, train_target)
```

[ ]:
```
MultinomialNB()
```

[ ]:
```
dev_test_texts[14]
clf.predict(dev_test_vectors[14])
```

[ ]:
```
array(['neg'], dtype='<U3')
```

[ ]:
```
clf.predict(dev_test_vectors)
```

[ ]:
```
array(['pos', 'pos', 'neg', 'neg', 'neg', 'pos', 'pos', 'neg', 'pos',
       'neg', 'pos', 'neg', 'neg', 'pos', 'neg', 'neg', 'pos', 'neg',
       'pos', 'neg', 'neg', 'neg', 'neg', 'neg', 'pos', 'pos', 'neg',
       'neg', 'neg', 'pos', 'neg', 'pos', 'pos', 'pos', 'neg', 'neg',
       'pos', 'pos', 'neg', 'neg', 'neg', 'neg', 'neg', 'neg', 'neg',
       'pos', 'pos', 'neg', 'neg', 'neg', 'neg', 'pos', 'neg', 'neg',
       'pos', 'pos', 'neg', 'neg', 'pos', 'neg', 'neg', 'pos', 'pos',
       'neg', 'neg', 'pos', 'neg', 'pos', 'pos', 'pos', 'pos', 'pos',
       'neg', 'neg', 'neg', 'pos', 'pos', 'pos', 'pos', 'neg', 'pos',
       'pos', 'pos', 'pos', 'pos', 'neg', 'pos', 'neg', 'neg', 'pos',
       'neg', 'neg', 'pos', 'neg', 'neg', 'pos', 'pos', 'neg', 'pos',
       'pos', 'pos', 'neg', 'pos', 'neg', 'pos', 'pos', 'pos', 'pos',
       'pos', 'pos', 'pos', 'neg', 'neg', 'pos', 'neg', 'pos', 'pos',
       'neg', 'neg', 'pos', 'pos', 'pos', 'pos', 'neg', 'pos', 'pos',
       'pos', 'pos', 'neg', 'pos', 'pos', 'pos', 'neg', 'neg', 'neg',
       'neg', 'pos', 'neg', 'pos', 'pos', 'neg', 'pos', 'pos', 'pos',
       'neg', 'neg', 'pos', 'neg', 'neg', 'neg', 'pos', 'neg', 'pos',
       'pos', 'neg', 'neg', 'neg', 'pos', 'pos', 'pos', 'neg', 'neg',
       'neg', 'neg', 'pos', 'pos', 'neg', 'pos', 'neg', 'pos', 'neg',
       'neg', 'pos', 'neg', 'pos', 'pos', 'pos', 'neg', 'pos', 'neg',
       'pos', 'pos', 'pos', 'pos', 'pos', 'neg', 'neg', 'neg', 'neg',
       'pos', 'pos', 'neg', 'pos', 'neg', 'pos', 'pos', 'neg', 'neg',
       'pos', 'pos'], dtype='<U3')
```

[ ]:
```
clf.score(dev_test_vectors, dev_test_target)
```

[ ]: 0.82

0.0.1 1.1.2 1b) Parameters of the vectorizer

Run experiments where you let binary vary over [False, True] and ngram_range vary over [[1,1], [1,2], [1,3]] and report the accuracy with the 6 different settings in a 2x3 table. Which settings yield the best results?

[ ]:
```
binary_param = [True, False]
ngram_range_param = [[1,1], [1,2], [1,3]]

for bp in binary_param:
  for ngram_param in ngram_range_param:
    v = CountVectorizer(
        binary=bp,
        ngram_range=ngram_param
    )
    train_vec = v.fit_transform(train_texts)
    dev_test_vec = v.transform(dev_test_texts)

    clf = MultinomialNB()
    clf.fit(train_vec, train_target)
    score = clf.score(dev_test_vec, dev_test_target)

    print(f"Binary = {bp}; Ngram_range = {ngram_param} -> score = {score}")
```

```
Binary = True; Ngram_range = [1, 1] -> score = 0.81
Binary = True; Ngram_range = [1, 2] -> score = 0.84
Binary = True; Ngram_range = [1, 3] -> score = 0.855
Binary = False; Ngram_range = [1, 1] -> score = 0.82
Binary = False; Ngram_range = [1, 2] -> score = 0.83
Binary = False; Ngram_range = [1, 3] -> score = 0.8
```

Binary = Ture and n_gram_range = [1,3] yields the best result of 0.855

0.0.2 1.2 Ex 2 n-fold cross-validation (12 points)

1.2.1 2a) Our dev_test_data contains only 200 items. That is a small number for a test set for a binary classifier. The numbers we report may depend to a large degree on the split between training and test data. To get more reliable numbers, we may use n-gram cross-validation. We can use the whole dev_test_data of 1800 items for this. To get round numbers, we decide to use 9-fold cross-validation, which will put 200 items in each test set. Use the best settings from exercise 1 and run a 9-fold cross-validation. Report the accuracy for each run, together with the mean and standard deviation of the 9 runs. In this exercise, you are requested to implement the routine for cross-validation yourself, and not apply the scikit-learn function.

[ ]:
```
cv_train_texts = [t[0] for t in movie_dev]
cv_train_target = [t[1] for t in movie_dev]
```

[ ]:
```
idx = 0
increment = 200
scores = []

v = CountVectorizer(
    binary=True,
    ngram_range=[1,3]
)

for cv in range(9):
  val_txt = cv_train_texts[idx:idx+increment]
  val_target = cv_train_target[idx:idx+increment]

  train_txt = cv_train_texts[:idx] + cv_train_texts[idx+increment:]
  train_target = cv_train_target[:idx] + cv_train_target[idx+increment:]

  idx += increment

  train_vec = v.fit_transform(train_txt)
  val_vec = v.transform(val_txt)

  clf = MultinomialNB()
  clf.fit(train_vec, train_target)
  score = clf.score(val_vec, val_target)
  scores.append(score)
```

[ ]:
```
# scores for run 1 too run 9
scores
```

[ ]:
```
[0.815, 0.855, 0.87, 0.85, 0.82, 0.855, 0.89, 0.84, 0.86]
```

[ ]:
```
# mean score of the 9 runs
np.mean(scores)
```

[ ]:
```
0.8505555555555556
```

[ ]:
```
# standard deviation for the 9 runs
np.std(scores)
```

[ ]:
```
0.022040927138752657
```

1.2.2 2b) The large variation we see between the results, raises a question regarding whether the optimal settings we found in exercise 1, would also be optimal for another split between training

and test. To find out, we combine the 9-fold cross-validation with the various settings for CountVectorizer. For each of the 6 settings, run 9-fold cross-validation and calculate the mean accuracy. Report the results in a 2x3 table. Answer: Do you see the same as when you only used one test set?

[ ]:
```
def cv(iters, val_size, param_binary, param_ngram_range, clf_to_fit):
  idx = 0
  increment = val_size
  scores = []

  v = CountVectorizer(
      binary=param_binary,
      ngram_range=param_ngram_range
  )

  for cv in range(iters):
    val_txt = cv_train_texts[idx:idx+increment]
    val_target = cv_train_target[idx:idx+increment]

    train_txt = cv_train_texts[:idx] + cv_train_texts[idx+increment:]
    train_target = cv_train_target[:idx] + cv_train_target[idx+increment:]

    idx += increment

    train_vec = v.fit_transform(train_txt)
    val_vec = v.transform(val_txt)

    clf = clf_to_fit()
    clf.fit(train_vec, train_target)
    score = clf.score(val_vec, val_target)
    scores.append(score)

  return scores
```

[ ]:
```
binary_param = [True, False]
ngram_range_param = [[1,1], [1,2], [1,3]]

for bp in binary_param:
  for ngram_param in ngram_range_param:
    scores = cv(9, 200, bp, ngram_param, MultinomialNB )

    print(f"""Binary = {bp}; Ngram_range = {ngram_param}
        -> Mean score = {np.mean(scores)}
        -> std = {np.std(scores)}""")
```

```
Binary = True; Ngram_range = [1, 1]
        -> Mean score = 0.8216666666666668
        -> std = 0.021343747458109467
```

```
Binary = True; Ngram_range = [1, 2]
        -> Mean score = 0.8533333333333333
        -> std = 0.02818589087547963
Binary = True; Ngram_range = [1, 3]
        -> Mean score = 0.8505555555555556
        -> std = 0.022040927138752657
Binary = False; Ngram_range = [1, 1]
        -> Mean score = 0.8172222222222223
        -> std = 0.02517837598583255
Binary = False; Ngram_range = [1, 2]
        -> Mean score = 0.833888888888889
        -> std = 0.019547346761954645
Binary = False; Ngram_range = [1, 3]
        -> Mean score = 0.8188888888888889
        -> std = 0.0183753726761569
```

The best parameters has changed from binary = True & n_gram_range = [1,3] too binary = True & n_gram_range = [1,2]. I would say the results are about the same.

0.0.3 1.3 Ex 3 Logistic Regression (8 points)

We know that Logistic Regression may produce better results than Naive Bayes. We will see what happens if we use Logistic Regression instead of Naive Bayes on this task. We start with the same multinomial model for text classification as in exercises (1) and (2) above (i.e. we process the data the same way and use the same vectorizer), but exchange the learner with sciki-learn's LogisticRegression. Since logistic regression is slow to train, we restrict ourselves somewhat with respect to which experiments to run. We consider two settings for the CountVectorizer, the default setting and the setting which gave the best result with naive Bayes when we ran cross-validation. (Though, this does not have to be the best setting for the logistic regression). For each of the two settings, run 9-fold cross-validation and calculate the mean accuracy. Compare the results in a 2x2 table where one axis is Naive Bayes vs. Logistic Regression and the other axis is default settings vs. earlier best settings for CountVectorizer. Write a few sentences where you discuss what you see from the table.

[ ]:
```
# default: binary = False, ngram_range = [1,1]
# best params for multinomial: binary = True, ngram_range=[1,2]
# [(binary, ngram_range)]
params = [
    (False, [1,1]),
    (True, [1,2])
]

for param_combo in params:
    scores = cv(9, 200, param_combo[0], param_combo[1], LogisticRegression)

    print(f"""Binary = {param_combo[0]}; Ngram_range = {param_combo[1]}
        -> Mean score = {np.mean(scores)}
```

```
        -> std = {np.std(scores)}""")
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
Binary = False; Ngram_range = [1, 1]
        -> Mean score = 0.8355555555555555
```

```
        -> std = 0.01770819716723247
Binary = True; Ngram_range = [1, 2]
        -> Mean score = 0.8733333333333333
        -> std = 0.014337208778404392
```

Params	Logit	Naive-bayes
default	0.84	0.82
best params	0.87	0.85

We se that logit does a better job at predicting than naive-bayes, this is true for defaults parameters and the best parameters. This comes at higher computational costs.