tutorial-sklearn-multinomialnb-exercise-blank_1_

August 6, 2019

1 Tutorial - build MNB with sklearn

This tutorial demonstrates how to use the Sci-kit Learn (sklearn) package to build Multinomial Naive Bayes model, rank features, and use the model for prediction.

The data from the Kaggle Sentiment Analysis on Movie Review Competition are used in this tutorial. Check out the details of the data and the competition on Kaggle. https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews

The tutorial also includes sample code to prepare your prediction result for submission to Kaggle. Although the competition is over, you can still submit your prediction to get an evaluation score.

2 Step 1: Read in data

In [4]:
```
# read in the training data

# the data set includes four columns: PhraseId, SentenceId, Phrase, Sentiment
# In this data set a sentence is further split into phrases
# in order to build a sentiment classification model
# that can not only predict sentiment of sentences but also shorter phrases

# A data example:
# PhraseId SentenceId Phrase Sentiment
# 1 1 A series of escapades demonstrating the adage that what is good for the goose is

# the Phrase column includes the training examples
# the Sentiment column includes the training labels
# "0" for very negative
# "1" for negative
# "2" for neutral
# "3" for positive
# "4" for very positive

import numpy as np
import pandas as p
train=p.read_csv("/Users/byu/Desktop/data/kaggle/train.tsv", delimiter='\t')
y=train['Sentiment'].values
X=train['Phrase'].values
```

3 Step 2: Split train/test data for hold-out test

In [5]:
```
# check the sklearn documentation for train_test_split
# http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_
# "test_size" : float, int, None, optional
# If float, should be between 0.0 and 1.0 and represent the proportion of the dataset
# If int, represents the absolute number of test samples.
# If None, the value is set to the complement of the train size.
# By default, the value is set to 0.25. The default will change in version 0.21. It wi

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=(

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
print(X_train[0])
print(y_train[0])
print(X_test[0])
print(y_test[0])
```

```
(93636,) (93636,) (62424,) (62424,)
almost in a class with that of Wilde
3
escape movie
2
```

Sample output from the code above:
(93636,) (93636,) (62424,) (62424,) almost in a class with that of Wilde 3 escape movie 2

4 Step 2.1 Data Checking

In [6]:
```
# Check how many training examples in each category
# this is important to see whether the data set is balanced or skewed

unique, counts = np.unique(y_train, return_counts=True)
print(np.asarray((unique, counts)))
```

```
[[    0     1     2     3     4]
 [ 4141 16449 47718 19859  5469]]
```

The sample output shows that the data set is skewed with 47718/93636=51% "neutral" examples. All other categories are smaller.
{0, 1, 2, 3, 4} [[0 4141] [1 16449] [2 47718] [3 19859] [4 5469]]

5 Exercise A

In [6]:
```
# Print out the category distribution in the test data set.
#Is the test data set's category distribution similar to the training data set's?
```

```
# Your code starts here

# Your code ends here
```

```
{0, 1, 2, 3, 4}
[[    0  2931]
 [    1 10824]
 [    2 31864]
 [    3 13068]
 [    4  3737]]
```

6 Step 3: Vectorization

In [7]:
```
# sklearn contains two vectorizers

# CountVectorizer can give you Boolean or TF vectors
# http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.Cou

# TfidfVectorizer can give you TF or TFIDF vectors
# http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.Tfi

# Read the sklearn documentation to understand all vectorization options

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

# several commonly used vectorizer setting

#  unigram boolean vectorizer, set minimum document frequency to 5
unigram_bool_vectorizer = CountVectorizer(encoding='latin-1', binary=True, min_df=5, s

#  unigram term frequency vectorizer, set minimum document frequency to 5
unigram_count_vectorizer = CountVectorizer(encoding='latin-1', binary=False, min_df=5,

#  unigram and bigram term frequency vectorizer, set minimum document frequency to 5
gram12_count_vectorizer = CountVectorizer(encoding='latin-1', ngram_range=(1,2), min_d

#  unigram tfidf vectorizer, set minimum document frequency to 5
unigram_tfidf_vectorizer = TfidfVectorizer(encoding='latin-1', use_idf=True, min_df=5,
```

6.1 Step 3.1: Vectorize the training data

In [8]:
```
# The vectorizer can do "fit" and "transform"
# fit is a process to collect unique tokens into the vocabulary
# transform is a process to convert each document to vector based on the vocabulary
# These two processes can be done together using fit_transform(), or used individually
```

```
# fit vocabulary in training documents and transform the training documents into vecto
X_train_vec = unigram_count_vectorizer.fit_transform(X_train)

# check the content of a document vector
print(X_train_vec.shape)
print(X_train_vec[0].toarray())

# check the size of the constructed vocabulary
print(len(unigram_count_vectorizer.vocabulary_))

# print out the first 10 items in the vocabulary
print(list(unigram_count_vectorizer.vocabulary_.items())[:10])

# check word index in vocabulary
print(unigram_count_vectorizer.vocabulary_.get('imaginative'))
```

```
(93636, 11967)
[[0 0 0 ... 0 0 0]]
11967
[('class', 1858), ('wilde', 11742), ('derring', 2802), ('chilling', 1764), ('affecting', 313),
5224
```

Sample output:

(93636, 11967) [[0 0 0 . . . , 0 0 0]] 11967 [('imaginative', 5224), ('tom', 10809), ('smiling', 9708), ('easy', 3310), ('diversity', 3060), ('impossibly', 5279), ('buy', 1458), ('sentiments', 9305), ('households', 5095), ('deteriorates', 2843)] 5224

6.2 Step 3.2: Vectorize the test data

```
In [9]: # use the vocabulary constructed from the training data to vectorize the test data.
        # Therefore, use "transform" only, not "fit_transform",
        # otherwise "fit" would generate a new vocabulary from the test data

        X_test_vec = unigram_count_vectorizer.transform(X_test)

        # print out #examples and #features in the test set
        print(X_test_vec.shape)
```

```
(62424, 11967)
```

Sample output:

(62424, 14324)

7 Exercise B

```
In [10]: # In the above sample code, the term-frequency vectors were generated for training an
```

```
# Some people argue that
# because the MultinomialNB algorithm is based on word frequency,
# we should not use boolean representation for MultinomialNB.
# While in theory it is true, you might see people use boolean representation for Mul
# especially when the chosen tool, e.g. Weka, does not provide the BernoulliNB algori

# sklearn does provide both MultinomialNB and BernoulliNB algorithms.
# http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.ht
# You will practice that later

# In this exercise you will vectorize the training and test data using boolean repres
# You can decide on other options like ngrams, stopwords, etc.

# Your code starts here

# Your code ends here
```

```
(93636, 11967)
(62424, 11967)
```

8 Step 4: Train a MNB classifier

In [10]:
```
# import the MNB module
from sklearn.naive_bayes import MultinomialNB

# initialize the MNB model
nb_clf= MultinomialNB()

# use the training data to train the MNB model
nb_clf.fit(X_train_vec,y_train)
```

Out[10]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

9 Step 4.1 Interpret a trained MNB model

In [12]:
```
## interpreting naive Bayes models
## by consulting the sklearn documentation you can also find out feature_log_prob_,
## which are the conditional probabilities
## http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB

# the code below will print out the conditional prob of the word "worthless" in each
# sample output
# -8.98942647599 -> logP('worthless'|very negative')
# -11.1864401922 -> logP('worthless'|negative')
# -12.3637684625 -> logP('worthless'|neutral')
# -11.9886066961 -> logP('worthless'|positive')
```

```
# -11.0504454621 -> logP('worthless'|very positive')
# the above output means the word feature "worthless" is indicating "very negative"
# because P('worthless'|very negative) is the greatest among all conditional probs

unigram_count_vectorizer.vocabulary_.get('worthless')
for i in range(0,5):
  print(nb_clf.feature_log_prob_[i][unigram_count_vectorizer.vocabulary_.get('worthles
```

```
-8.538982639195012
-10.64363758669141
-11.841984577875767
-11.477837002297735
-10.62975514642929
```

Sample output:
-8.5389826392 -10.6436375867 -11.8419845779 -11.4778370023 -10.6297551464

In [13]:
```
# sort the conditional probability for category 0 "very negative"
# print the words with highest conditional probs
# these can be words popular in the "very negative" category alone, or words popular

feature_ranks = sorted(zip(nb_clf.feature_log_prob_[0], unigram_count_vectorizer.get_
very_negative_features = feature_ranks[-10:]
print(very_negative_features)
```

```
[(-5.941598005980322, 'time'), (-5.931015896649785, 'characters'), (-5.92054459678249, 'minute
```

10 Exercise C

In [21]:
```
# calculate log ratio of conditional probs

# In this exercise you will calculate the log ratio
# between conditional probs in the "very negative" category
# and conditional probs in the "very positive" category,
# and then sort and print out the top and bottom 10 words

# the conditional probs for the "very negative" category is stored in nb_clf.feature_
# the conditional probs for the "very positive" category is stored in nb_clf.feature_

# You can consult with similar code in week 4's sample script on feature weighting
# Note that in sklearn's MultinomialNB the conditional probs have been converted to l

# Your code starts here

# Your code ends here
```

```
[(-4.6685160302515563, 'worst'), (-4.1676048635427438, 'bad'), (-3.9753688496916109, 'stupid')
[(3.5668446135017922, 'rich'), (3.6374621807157457, 'wonderful'), (3.8045162653789113, 'excelle
```

Sample output for print(log_ratios[0])
-0.838009538739

11 Step 5: Test the MNB classifier

```
In [14]: # test the classifier on the test data set, print accuracy score

         nb_clf.score(X_test_vec,y_test)

Out[14]: 0.606401384083045
```

```
In [23]: # print confusion matrix (row: ground truth; col: prediction)

         from sklearn.metrics import confusion_matrix
         y_pred = nb_clf.fit(X_train_vec, y_train).predict(X_test_vec)
         cm=confusion_matrix(y_test, y_pred, labels=[0,1,2,3,4])
         print(cm)
```

```
[[  733  1264   817   106    11]
 [  602  4132  5411   649    30]
 [  246  2397 25756  3226   239]
 [   19   454  5580  6248   767]
 [    1    54   725  1972   985]]
```

```
In [24]: # print classification report

         from sklearn.metrics import precision_score
         from sklearn.metrics import recall_score
         print(precision_score(y_test, y_pred, average=None))
         print(recall_score(y_test, y_pred, average=None))

         from sklearn.metrics import classification_report
         target_names = ['0','1','2','3','4']
         print(classification_report(y_test, y_pred, target_names=target_names))
```

```
[ 0.45783885  0.49777135  0.67267361  0.51208917  0.48474409]
[ 0.2500853   0.38174427  0.80831032  0.47811448  0.26358041]
             precision    recall  f1-score   support

          0       0.46      0.25      0.32      2931
          1       0.50      0.38      0.43     10824
          2       0.67      0.81      0.73     31864
          3       0.51      0.48      0.49     13068
```

```
         4       0.48      0.26      0.34      3737

avg / total       0.59      0.61      0.59     62424
```

12 Step 5.1 Interpret the prediction result

```
In [25]: ## find the calculated posterior probability
         posterior_probs = nb_clf.predict_proba(X_test_vec)

         ## find the posterior probabilities for the first test example
         print(posterior_probs[0])

         # find the category prediction for the first test example
         y_pred = nb_clf.predict(X_test_vec)
         print(y_pred[0])

         # check the actual label for the first test example
         print(y_test[0])
```

```
[ 0.06530109  0.34286038  0.50421851  0.07186049  0.01575954]
2
2
```

sample output array([0.06434628 0.34275846 0.50433091 0.07276319 0.01580115]

Because the posterior probability for category 2 (neutral) is the greatest, 0.50, the prediction should be "2". Because the actual label is also "2", this is a correct prediction

13 Step 5.2 Error Analysis

```
In [26]: # print out specific type of error for further analysis

         # print out the very positive examples that are mistakenly predicted as negative
         # according to the confusion matrix, there should be 53 such examples
         # note if you use a different vectorizer option, your result might be different

         err_cnt = 0
         for i in range(0, len(y_test)):
             if(y_test[i]==4 and y_pred[i]==1):
                 print(X_test[i])
                 err_cnt = err_cnt+1
         print("errors:", err_cnt)
```

```
this might not seem like the proper cup of tea , however it is almost guaranteed that even the
Parents may even find that it goes by quickly , because it has some of the funniest jokes of ar
```

of the best short story writing
are an absolute joy .
Adams , with four scriptwriters , takes care with the characters , who are so believable that y
this might not seem like the proper cup of tea , however it is almost guaranteed that even the
you will probably like it .
the funniest jokes
the film is never dull
, Lawrence 's delivery remains perfect
one of the most high-concept sci fi adventures attempted for the screen
We 've seen the hippie-turned-yuppie plot before , but there 's an enthusiastic charm in Fire t
that it goes by quickly , because it has some of the funniest jokes of any movie this year , in
is a hoot , and is just as good , if not better than much of what 's on Saturday morning TV esp
worldly-wise and very funny script
A smart , provocative drama that does the nearly impossible : It gets under the skin of a man w
very thrilling
excited about on this DVD
an absolute joy
two fine actors , Morgan Freeman and Ashley Judd
stunningly
, its hard to imagine having more fun watching a documentary ...
is just too original to be ignored .
flat-out amusing ,
may even find that it goes by quickly , because it has some of the funniest jokes of any movie
to make the most sincere and artful movie in which Adam Sandler will probably ever appear
The story , once it gets rolling , is nothing short of a great one .
Jackie Chan movies are a guilty pleasure - he 's easy to like and always leaves us laughing
ca n't go wrong .
the best short story writing
deserves a sequel
The film is a hoot , and is just as good , if not better than much of what 's on Saturday morni
wo n't be disappointed .
achieves the near-impossible
succeed merrily at their noble endeavor .
the most high-concept sci fi adventures
first-rate , especially Sorvino
the utter cuteness
simply ca n't recommend it enough .
A terrifically entertaining specimen of Spielbergian sci-fi .
you wo n't be disappointed
Fairy-tale formula , serves as a paper skeleton for some very good acting , dialogue , comedy
that even the stuffiest cinema goers will laugh their *** off for an hour-and-a-half
can do no wrong with Jason X.
Orgasm
The entire cast is first-rate , especially Sorvino .
be spectacularly outrageous
The gags that fly at such a furiously funny pace that the only rip off that we were aware of wa
hard to imagine having more fun watching a documentary ...
he does display an original talent

```
when his story ends or just ca n't tear himself away from the characters
is a seriously intended movie that is not easily forgotten .
ca n't recommend it enough
does n't try to surprise us with plot twists , but rather seems to enjoy its own transparency
errors: 54
```

14 Exercise D

```
In [27]: # Can you find linguistic patterns in the above errors?
         # What kind of very positive examples were mistakenly predicted as negative?

         # Can you write code to print out the errors that very negative examples were mistake
         # Can you find lingustic patterns for this kind of errors?
         # Based on the above error analysis, what suggestions would you give to improve the c

         # Your code starts here

         # Your code ends here
```

```
this is the opposite of a truly magical movie .
achieves the remarkable feat of squandering a topnotch foursome of actors
a deeply unpleasant experience
hugely overwritten
is not Edward Burns ' best film
Once the expectation of laughter has been quashed by whatever obscenity is at hand , even the
is a deeply unpleasant experience .
is hugely overwritten ,
is the opposite of a truly magical movie .
to this shocking testament to anti-Semitism and neo-fascism
is about as humorous as watching your favorite pet get buried alive
errors: 11
```

15 Step 6: write the prediction output to file

```
In [28]: y_pred=nb_clf.predict(X_test_vec)
         output = open('/Users/byu/Desktop/data/prediction_output.csv', 'w')
         for x, value in enumerate(y_pred):
           output.write(str(value) + '\n')
         output.close()
```

16 Step 6.1 Prepare submission to Kaggle sentiment classification competition

In [29]:
```
########## submit to Kaggle submission

# we are still using the model trained on 60% of the training data
# you can re-train the model on the entire data set
#   and use the new model to predict the Kaggle test data
# below is sample code for using a trained model to predict Kaggle test data
#    and format the prediction output for Kaggle submission

# read in the test data
kaggle_test=p.read_csv("/Users/byu/Desktop/data/kaggle/test.tsv", delimiter='\t')

# preserve the id column of the test examples
kaggle_ids=kaggle_test['PhraseId'].values

# read in the text content of the examples
kaggle_X_test=kaggle_test['Phrase'].values

# vectorize the test examples using the vocabulary fitted from the 60% training data
kaggle_X_test_vec=unigram_count_vectorizer.transform(kaggle_X_test)

# predict using the NB classifier that we built
kaggle_pred=nb_clf.fit(X_train_vec, y_train).predict(kaggle_X_test_vec)

# combine the test example ids with their predictions
kaggle_submission=zip(kaggle_ids, kaggle_pred)

# prepare output file
outf=open('/Users/byu/Desktop/data/kaggle/kaggle_submission.csv', 'w')

# write header
outf.write('PhraseId,Sentiment\n')

# write predictions with ids to the output file
for x, value in enumerate(kaggle_submission): outf.write(str(value[0]) + ',' + str(va

# close the output file
outf.close()
```

17 Exercise E

In [ ]:
```
# generate your Kaggle submissions with boolean representation and TF representation
# submit to Kaggle
# report your scores here
# which model gave better performance in the hold-out test
```

```
# which model gave better performance in the Kaggle test
```

Sample output:

(93636, 9968) [[0 0 0 ..., 0 0 0]] 9968 [('disloc', 2484), ('surgeon', 8554), ('camaraderi', 1341), ('sketchiest', 7943), ('dedic', 2244), ('impud', 4376), ('adopt', 245), ('worker', 9850), ('buy', 1298), ('systemat', 8623)] 245

18 BernoulliNB

```
In [30]: from sklearn.naive_bayes import BernoulliNB
         X_train_vec_bool = unigram_bool_vectorizer.fit_transform(X_train)
         bernoulliNB_clf = BernoulliNB(X_train_vec_bool, y_train)
```

19 Cross Validation

```
In [31]: # cross validation

         from sklearn.pipeline import Pipeline
         from sklearn.model_selection import cross_val_score
         nb_clf_pipe = Pipeline([('vect', CountVectorizer(encoding='latin-1', binary=False)),(
         scores = cross_val_score(nb_clf_pipe, X, y, cv=3)
         avg=sum(scores)/len(scores)
         print(avg)
```

```
0.559547456968
```

20 Exercise F

```
In [33]: # run 3-fold cross validation to compare the performance of
         # (1) BernoulliNB (2) MultinomialNB with TF vectors (3) MultinomialNB with boolean ve

         # Your code starts here


         # Your code ends here
```

```
0.55315243657
0.553844611375
0.552306763002
0.560136963721
```

21 Optional: use external linguistic resources such as stemmer

```
In [204]: from sklearn.feature_extraction.text import CountVectorizer
          import nltk.stem
```

```
english_stemmer = nltk.stem.SnowballStemmer('english')
class StemmedCountVectorizer(CountVectorizer):
    def build_analyzer(self):
        analyzer = super(StemmedCountVectorizer, self).build_analyzer()
        return lambda doc: ([english_stemmer.stem(w) for w in analyzer(doc)])

stem_vectorizer = StemmedCountVectorizer(min_df=3, analyzer="word")
X_train_stem_vec = stem_vectorizer.fit_transform(X_train)
```

In [194]:
```
# check the content of a document vector
print(X_train_stem_vec.shape)
print(X_train_stem_vec[0].toarray())

# check the size of the constructed vocabulary
print(len(stem_vectorizer.vocabulary_))

# print out the first 10 items in the vocabulary
print(list(stem_vectorizer.vocabulary_.items())[:10])

# check word index in vocabulary
print(stem_vectorizer.vocabulary_.get('adopt'))
```

```
(93636, 9968)
[[0 0 0 ..., 0 0 0]]
9968
[('disloc', 2484), ('surgeon', 8554), ('camaraderi', 1341), ('sketchiest', 7943), ('dedic', 224
245
```

In [ ]: