Kadyrov_Daniel_Assignment4

October 26, 2020

Daniel Kadyrov

Stevens ID: 10455680

CS557 - Natural Language Processing

Group 32 - Daniel Kadyrov

1 Part 1

Apply logit regression to the 2-category data in RFMdataMPJ.xlsx. Preview the document or the Iris data (Links to an external site.) with multiple categories.

Python has logistic regression capability; try the sklearn package. You are welcome to use any other, but in any case, give a reference for it. Or you are welcome to work through the Building A Logistic Regression in Python, Step by Step (Links to an external site.) tutorial and apply it to the data in the RFM and/or Iris file.

The Iris data describes pedal and sepal length and width for 150 Irises classified into three categories. The pedal and sepal length and width are four features to be used to train a classifier. You might want to hold out at least three Irises (one from each category) to be classified by the probability of them being in the categories and see how accurate the classifier is. When applied to NLP classification you can associate the features of some text categories with their classification in a similar manner. For pedagogic purposes the 4-feature Iris dataset is suggested since text features often tend to be much larger, such as many of the most frequent words, proper names, etc.

[20]:
```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

x = iris.data
y = iris.target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=7)

model = LogisticRegression()
```

```
model.fit(x_train, y_train)

model.score(x_test, y_test)
```

[20]:
```
0.8666666666666667
```

2 Part 2

Like in the previous Assignment, train the sentiment classifier using logistic regression, but follow the directions in this paragraph. Use a corpus of tagged reviews similar to the way Amazon does it. Each review is given a tag on a scale, say, from 1 to 5, the higher the tag the more favorable the review, or as categories like favorable, unfavorable, neutral. A 5-point category scale can be very favorable, favorable, neutral, unfavorable, and very unfavorable. In any case, treat the ratings as a set of categories, not numerical values. Find files, one for favorable (Links to an external site.) words and one for unfavorable (Links to an external site.) ones, and count the number of each in each review. Also count the numbers of words that change these into the other category, like “not” and “hardly” as in “hardly amusing”, the “un-“ prefix as in “unhappy”, and the “a-“ prefix as in “amoral”, and any other negators you can think off or find. Regular expressions are useful for this. Lists of these may be more difficult to find so you may have to make some up. These counts become the features (independent variables) used as inputs to the logistic regression model. Each review has three independent variables/features: count of favorable words, count of unfavorable words, and the count of negators, and one dependent variable, the category with the highest probability. Then report a confusion matrix (example in Course Materials) to show how accurate the model matches the training set categories. If you get enough data create the training, validation, and testing partitions and use the matrix functions in Excel, the xla Excel add-in, or a matrix algebra package like MATLAB or TensorFlow. Show your results in a confusion matrix. For 2 categories you can use ClassifyConfusionMatrix.xlsxPreview the document., for more than two categories check here (Links to an external site.).

[21]:
```
positive = []
negative = []

with open("data/opinion-lexicon-English/positive-words.txt", "r",␣
 ↪encoding="ISO-8859-1") as file:
    lines = [l for l in (line.strip() for line in file) if l]
    for line in lines:
        if ";" in line:
            pass
        else:
            positive.append(line)

with open("data/opinion-lexicon-English/negative-words.txt", "r",␣
 ↪encoding="ISO-8859-1") as file:
    lines = [l for l in (line.strip() for line in file) if l]
    for line in lines:
        if ";" in line:
            pass
        else:
            negative.append(line)
```

[22]:
```
import re

with open("data/processed_stars/books/all_balanced.review", "r") as file:
    reviews = []
    labels = []
    for line in file:
        review = []
        liner = line.split(" ")
        for word in liner:
            if "#label#" in word:
                labels.append(int(float(word.split(":")[-1].split()[0])))
            else:
                review.append(word.split(":")[0]+" " * int(word.split(":")[1]))

        reviews.append(" ".join(review).split())
```

[23]:
```
negator = ["un", "not", "hardly", "a"]
```

[24]:
```
X = []
for review in reviews:
    p = 0
    n = 0
    nn = 0
    for word in review:
        if word in positive:
            p += 1
        if word in negative:
            n += 1
        if word in negator:
            nn += 1
        for neg in negator:
            if neg in word:
                nn += 1
        if word.startswith("a"):
            nn += 1

    X.append([p, n, nn])
```

[25]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,␣
 ↪random_state=0)
```

```
[26]: from sklearn.linear_model import LogisticRegression

      LR = LogisticRegression()
      LR.fit(x_train, y_train)

      predict = LR.predict(x_test)

      LR.score(x_test, y_test)
```

```
[26]: 0.3851744186046512
```

```
[27]: from sklearn import metrics

      cm = metrics.confusion_matrix(y_test, predict)

      print(cm)
```

```
[[211  53  34  56]
 [155  68  50  71]
 [ 61  61  50 165]
 [ 64  34  42 201]]
```

3 Part 3

Demonstrate embeddings for a small vocabulary and some phrases/sentences. Check out word2vec_basic.py in the gensim package and similar resources. BKL may not be helpful for this but check out the online Python 3 version anyway. There are demos on line (Links to an external site.). Report the confusion matrix for the application of the model to the test set.

```
[5]: from gensim.test.utils import common_texts, get_tmpfile
     from gensim.models import Word2Vec

     model = Word2Vec(reviews, size=300)
```

```
[6]: w1 = ["dirty"]
     model.wv.most_similar(positive=w1)
```

```
[6]: [('guess_i', 0.9869878888130188),
      ('feeling_that', 0.9866390228271484),
      ('good_writing', 0.9859217405319214),
      ('consider_it', 0.985460638999939),
      ('and_as', 0.985217809677124),
      ('crime_and', 0.9851363897323608),
      ('army', 0.9847644567489624),
      ('are_better', 0.9845976829528809),
      ('and_am', 0.9845890402793884),
 ('too_complex', 0.9843344688415527)]
```

[7]:
```
w2 = ["broken"]
model.wv.most_similar(positive=w2)
```

[7]:
```
[('ago_the', 0.9930318593978882),
 ('he_turned', 0.9921306371688843),
 ('but_unfortunately', 0.9920940399169922),
 ('were_written', 0.9916591644287109),
 ('presented_and', 0.9908245801925659),
 ('then_a', 0.9907580614089966),
 ('possibly', 0.9906696081161499),
 ('cents', 0.9905273914337158),
 ('and_background', 0.9904022216796875),
 ('hopefully', 0.9903094172477722)]
```

[8]:
```
new_positive = []
for pos in positive:
    try:
        new_positive.append(model.most_similar(pos)[0][0])
    except:
        continue
positive += new_positive
```

[9]:
```
new_negative = []
for neg in negative:
    try:
        new_negative.append(model.most_similar(neg)[0][0])
    except:
        continue

negative += new_negative
```

[10]:
```
new_negator= []
for neg in negator:
    try:
        new_negator.append(model.most_similar(neg)[0][0])
    except:
        pass
negator += new_negator
```

[19]:
```
X = []
for review in reviews:
    p = 0
    n = 0
    nn = 0
    for word in review:
        if word in positive:
            p += 1
        if word in negative:
            n += 1
        if word in negator:
            nn += 1
        for neg in negator:
            if neg in word:
                nn += 1
        if word.startswith("a"):
            nn += 1

    X.append([p, n, nn])
```

[15]:
```
x_train, x_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,␣
 ↪random_state=0)

LR = LogisticRegression()
LR.fit(x_train, y_train)

predict = LR.predict(x_test)

LR.score(x_test, y_test)
```

[15]:
```
0.37281976744186046
```

[18]:
```
metrics.confusion_matrix(y_test, predict)
```

[18]:
```
array([[217,  39,  34,  64],
       [175,  44,  51,  74],
       [ 81,  40,  54, 162],
       [ 69,  40,  34, 198]])
```

4 Part 4

Examine the methods, such as document embeddings, in A&S Ch4 from p83 on applied to sentence tokenizers. Compare them to those BKL sections on processing sentences. This is an exploratory assignment which could be open ended and possibly used as a final project. Do not get bogged down.

Sentence tokenization involves segementing a document into sentences. The NLTK package provides the function sent_tokenize() to seperate a document into a list/array of sentences NLTK Sentence Tokenize.