final-P1-sol

December 19, 2020

1 Academic Integrity Statement

As a matter of Departmental policy, **we are required to give you a 0** unless you **type your name** after the following statement:

> *I certify on my honor that I have neither given nor received any help, or used any non-permitted resources, while completing this evaluation.*

[TYPE YOUR NAME HERE]

2 Problem 1 (50 points)

Rampant disinformation—often called "fake news"—has emerged as one of the fundamental crises over our time.

There is a growing movement for online platforms to regulate fake news. Doing so at scale requires combing through millions of news items every day, making it very expensive to do by hand. Can an algorithm do it instead?

The following two URLs each contain part of a data set.

- **Fake news items**: `https://raw.githubusercontent.com/PhilChodrow/PIC16A/master/datasets/fake`
- **Real news items**: `https://raw.githubusercontent.com/PhilChodrow/PIC16A/master/datasets/fake`

Use the data at these urls to **construct a fake news classifier.**

1. Your model must be able to **make predictions** about whether or not an unseen news item is fake or real.
2. Because fake news models must be able to make millions of predictions per day, it must be able to make predictions very quickly. More columns mean more computation time. **Your final model should use no more than 50 columns.**

You are free to create any columns that you need, and to use any functions that we have or have not covered in the course. You may also use any machine learning model.

Please use Markdown headers with ## signs to clearly distinguish the different stages of your solution.

2.0.1 Requirements

1. Any operations that you perform multiple times (such processing that you perform on both the training and test sets) must be contained in function with informative docstrings. Comments and explanations are expected throughout. It is especially important to explain how you chose the columns to use in your final model.
2. You should not use for-loops to iterate over the rows of data frames or arrays.
3. You must fit your model on the training data, and not use the test data for fitting.

2.0.2 Hints

- `pd.concat()` is a good way to combine data frames.
- Try fitting a model with as many columns as you want first. See if you can get a representation of which columns are important, and then select your final columns from this list.
- In class, we talked about greedy stagewise feature selection and exhaustive enumeration for determining a good set of columns. Neither of these methods are recommended for this problem.
- If you want to be creative about your model choice, then please go for it. If you want a safe option, try logistic regression.
- If a model takes too long to fit on the full data set, try fitting it on, say, 10% of the data.
- You might find the some of the cheatsheets to be helpful.

2.0.3 Rubric

- (**15 points**): clearly written code that makes economical use of skills from the course to manipulate data.
- (**15 points**): comments, explanatory surrounding text, and docstrings for any functions and classes.
- (**20 points**): computed according to the formula `20 x score`, where `score` is your model's prediction performance on unseen data. Models that use more than 50 columns can receive up to 15 of these points. Scores will be rounded up. For example, if you obtain an 84% predictive performance with 50 columns, then the score of `20 x 0.84 = 16.8` will be rounded up to 17 points.

3 My Solution

Note: *many of you chose very different approaches here. Provided that you explained yourself, these generally received full credit. Some of you noticed that I made a silly mistake: the `subject` column is actually perfectly correlated with whether or not a news item was real, and so by using it you could get 100%. I didn't notice this and should have removed it! However, noticing something like this takes some work and some instincts, so solutions using this column still received full credit.*

First I'll download the data.

Solutions using the older multistep code shown in some lectures were also fine, but if you used it multiple times (rather than wrapping it in a function which you called twice), I made a small *deduction under the first rubric item.*

[1]:

```
import pandas as pd
fake_df = pd.read_csv("https://raw.githubusercontent.com/PhilChodrow/PIC16A/
 ↪master/datasets/fake_news/Fake.csv")
true_df = pd.read_csv("https://raw.githubusercontent.com/PhilChodrow/PIC16A/
 ↪master/datasets/fake_news/true.csv")
```

Next, I'll make indicators in each of the two pieces of the data indicating whether or not they are fake, and then combine them.

Some of you had problems from duplicated indices. Either passing `ignore_index` below or using `reset_index` later can solve these issues.

[2]:

```
fake_df['fake_news'] = 1
true_df['fake_news'] = 0
df = pd.concat((fake_df, true_df), axis = 0, ignore_index=True)
df = df.drop(["title", "date", "subject"], axis = 1)
```

3.0.1 Prepare Data

Now I'll prepare my data. First, I'll extract the term-document matrix using only relatively common words, and add it to my data frame. Then, I'll create predictor and target variables.

[3]:

```
from sklearn.feature_extraction.text import CountVectorizer

def prepare_data(df):
    """
    Prepare the fake news data set, passsed as df, for subsequent machine␣
 ↪learning tasks.
    1. Compute a term document matrix using sane defaults and add it to the df
    2. Prepare predictor X and target y.

    X is obtained by dropping the fake_news and complete text columns from df
        It therefore contains only the term-document matrix
    y is the fake_news column

    return: X, y, the prepared predictor and target data.
    """
    # 1. term-document matrix
    vec       = CountVectorizer(max_df=.5, min_df = 1000, stop_words='english')
    counts    = vec.fit_transform(df['text'])
    counts    = counts.toarray()
    count_df = pd.DataFrame(counts, columns=vec.get_feature_names())

    #    add to data frame
    out       = pd.concat((df, count_df), axis = 1)
    # 2. prepare predictor and target variables
    X        = out.drop(['fake_news', 'text'], axis = 1)
    y        = out['fake_news']

    return X, y
```

[4]:

```
X, y = prepare_data(df)
```

3.0.2 Train-Test Split

[5]:

```
# for reproducibility
import numpy as np
np.random.seed(1234)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5)
```

3.0.3 Identify Good Columns

So, I'm going to use logistic regression on the entire data set first. Then, I'll examine the coefficients, and use these to select the final 50 columns that I'm going to use.

First we'll create and train the model:

[6]:

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(solver = "liblinear")
LR.fit(X_train, y_train)
```

[6]:

```
LogisticRegression(solver='liblinear')
```

Now we'll evaluate. Because the model has so many columns to work with, the training score especially is very high! But the test score also isn't bad.

One could try CV here to improve even more, but I think this is good enough for now.

[7]:

```
LR.score(X_train, y_train), LR.score(X_test, y_test)
```

[7]:

```
(0.9952336406966903, 0.9656554857677402)
```

My approach to figuring out the most relevant words is to grab the top 50 words according to the absolute value of their coefficient.

[8]:

```
# create data frame with the coefficients and words
result_df = pd.DataFrame({"coef" : LR.coef_[0], "word" : X_train.columns})
# compute the absolute value of each coefficient
result_df['abs'] = np.abs(result_df['coef'])
```

```
# sort by the absolute value and grab the best 50
top_words = result_df.sort_values('abs', ascending = False).head(50)
# display
top_words
```

```
[8]:          coef             word       abs
     502   4.283595         featured  4.283595
     570   3.784056              gop  3.784056
     559   3.663105            getty  3.663105
     1051  3.617180             read  3.617180
     1089  3.103784              rep  3.103784
     945   2.957214              pic  2.957214
     628   2.680299            https  2.680299
     1119 -2.644479          reuters  2.644479
     194   2.642317        breitbart  2.642317
     1316 -2.593565         thursday  2.593565
     1402 -2.465560        wednesday  2.465560
     359   2.300493            daily  2.300493
     1439  2.160724        yesterday  2.160724
     1305  2.108230           thanks  2.108230
     27    2.103557           21wire  2.103557
     842   2.062048          morning  2.062048
     637   2.051147            image  2.051147
     26    2.039335             21st  2.039335
     455   2.027216           entire  2.027216
     260  -2.019130           citing  2.019130
     541  -2.015438           friday  2.015438
     838  -1.922789           monday  1.922789
     1298  1.912387           terror  1.912387
     1067 -1.878186        referring  1.878186
     110   1.867583       apparently  1.867583
     683   1.866951              isn  1.866951
     877  -1.864864              nov  1.864864
     122   1.853634             aren  1.853634
     1432  1.824422           wouldn  1.824422
     265   1.822876          claimed  1.822876
     1395  1.807508            watch  1.807508
     1100 -1.799013  representatives  1.799013
     1181 -1.743681             sept  1.743681
     278   1.734637              com  1.734637
     1358 -1.704565            urged  1.704565
     1172  1.703769              sen  1.703769
     848   1.700586               mr  1.700586
     1120  1.689319         revealed  1.689319
     322   1.674222    controversial  1.674222
     1293  1.671209          telling  1.671209
     784   1.639280            lying  1.639280
```

```
1342 -1.639083          tuesday  1.639083
181  -1.636248              bit  1.636248
1357  1.614529           unless  1.614529
770  -1.598257           london  1.598257
402   1.597520             didn  1.597520
1147 -1.560926         saturday  1.560926
114   1.556755          appears  1.556755
614   1.516468             hill  1.516468
1248  1.515818           stated  1.515818
```

Now I'll just get the words on their own:

Typing/copy-pasting the words from your model isn't great practice (not reproducible) and led to a deduction under code quality.

[9]:

```
top_words = top_words['word']
```

3.0.4 Final Model

Now I'll fit another logistic classifier using just the top words, and score it:

possible to do even better with other models!

[10]:

```
LR2 = LogisticRegression()
LR2.fit(X_train[top_words], y_train)
LR2.score(X_test[top_words], y_test)

# score on this problem would be 19/20
```

[10]:

```
0.9306873357387857
```