Yuqi Cheng
yc2233

Homework 2

CS 5785
October 5, 2021

Programming Exercises

1. Binary Classification on Text Data

(a). Download the data

We first download the dataset from Kaggle and examine the size of the train set and the test set.

```
In [590]: train.shape
Out[590]: (7613, 5)

In [591]: test.shape
Out[591]: (3263, 4)
```

Figure 1: Train and test size

Here we get **7613 datapoints in train.csv, and 3263 datapoints in test.csv.**Meanwhile, we also check the proportion of the positive and negative labels.

```
In [593]: len(train[train['target'] == 1]) / len(train)
Out[593]: 0.4296597924602653

In [594]: len(train[train['target'] == 0]) / len(train)
Out[594]: 0.5703402075397347
```

Figure 2: Label proportion

The percentage of real disaster is approximately 43%, and **the percentage of not being a real distaer is about 57%**.

(b). Split the training data.

```
In [595]: training = train.sample(frac = 0.7)
          develop = train.drop(training.index)

In [596]: training = training.sort_values(by = 'id')
          #sort the training sequence based on the value of 'id', which is chaotic after sampling
          training = training.reset_index(drop=True)
          #set a new index for traning dataset
```

Figure 3: Label proportion

After exploring the dataset, we then split it into a training set and a development set. The percentage we set as **70% to 30%**.

(c). Preprocess the data.

The key to a successful ML model is high-quality data. Here, we combine all the following steps into a function.

Because there are a lot of NaN values, we first replace the NaN. To split the sentence and construct a successful word bag, we need to turn all the words into a uniform format. Thus we also need to convert all upper case into lower, can delete other useless words like "and", "or" which do not contribute to the model training. URL information are useless as well, so we drop all these words.

We put what we have done as follows.

1. Replace NaN float values with Nan string values
2. Convert words into lower case
3. Lemmatize the words using nltk.stem
4. Strip punctuation (including @)
5. Strip the stop words, the, or, and
6. Strip url (http, www)

The code can be found in the ipynb file.

(d). Bag of words model.

In this section, we try to construct the word bag. We choose M=5, which means only include words that appear in at least 5 tweets. While taking 0.01, k = 50, only 63 words qualifies. Taking 0.005, k = 25, we get 277 words. The number of feature vector is too small for further training. For our choice, M = 5, the total number of the features in vector of development dataset is around 900 as we specified. Taking the max_features = 900 will automatically pick the most commonly occured **900 words for all qualified words** into the vocabulary.

The size of training and developing set are shown as follows

```
In [771]: x_train.shape
Out[771]: (5329, 900)
```

Figure 4: Training set size

```
In [773]: x_predict.shape
Out[773]: (2284, 900)
```

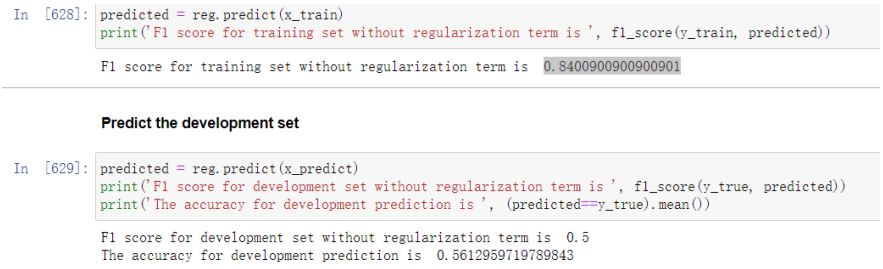
Figure 5: Developing set size

(e). Logistic regression.

i. Train the logistic regression model **without regularization term**

We first train the model without regularization term. We use sklearn package here. We **drop the column of id, keyword and location**. These column are irrelative for our further training.

We calculate two different metrics, training error and generalization error. **The training error is 0.84 in F1-score and the generalization error is 0.5 in F1.**

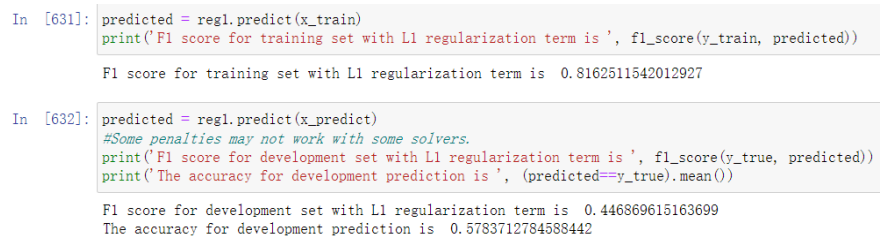


```
In [628]: predicted = reg.predict(x_train)
          print('F1 score for training set without regularization term is ', f1_score(y_train, predicted))

          F1 score for training set without regularization term is  0.8400900900900901

          Predict the development set

In [629]: predicted = reg.predict(x_predict)
          print('F1 score for development set without regularization term is ', f1_score(y_true, predicted))
          print('The accuracy for development prediction is ', (predicted==y_true).mean())

          F1 score for development set without regularization term is  0.5
          The accuracy for development prediction is  0.5612959719789843
```

Figure 6: Result without regularization term

ii. Logistic Regression Model with L1 regularization

We then add L1 regularization term to the model.**The training error is 0.816 in F1-score and the generalization error is 0.447 in F1.**

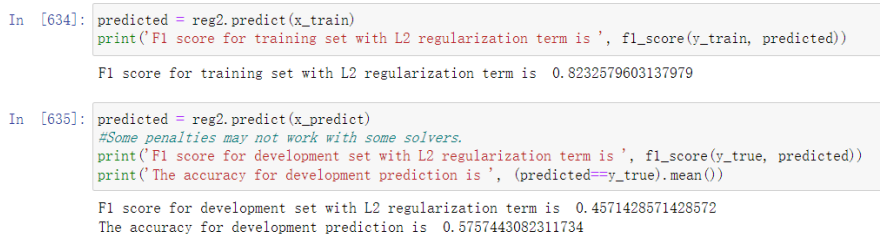


```
In [631]: predicted = reg1.predict(x_train)
          print('F1 score for training set with L1 regularization term is ', f1_score(y_train, predicted))

          F1 score for training set with L1 regularization term is  0.8162511542012927

In [632]: predicted = reg1.predict(x_predict)
          #Some penalties may not work with some solvers.
          print('F1 score for development set with L1 regularization term is ', f1_score(y_true, predicted))
          print('The accuracy for development prediction is ', (predicted==y_true).mean())

          F1 score for development set with L1 regularization term is  0.446869615163699
          The accuracy for development prediction is  0.5783712784588442
```

Figure 7: Result with L1 regularization term

iii. Logistic Regression Model with L2 regularization

We then add L2 regularization term to the model. **The training error is 0.824 in F1-score and the generalization error is 0.457 in F1.**



```
In [634]: predicted = reg2.predict(x_train)
          print('F1 score for training set with L2 regularization term is ', f1_score(y_train, predicted))

          F1 score for training set with L2 regularization term is  0.8232579603137979

In [635]: predicted = reg2.predict(x_predict)
          #Some penalties may not work with some solvers.
          print('F1 score for development set with L2 regularization term is ', f1_score(y_true, predicted))
          print('The accuracy for development prediction is ', (predicted==y_true).mean())

          F1 score for development set with L2 regularization term is  0.4571428571428572
          The accuracy for development prediction is  0.5757443082311734
```

Figure 8: Result with L2 regularization term

iv. The first classifier (Logistic model without regularization) that the generalization error in F1 score is 0.5 and the F1 score for training set is 0.84009, presents the best performance.

Also, **no overfitting or underfitting occurs obviously**, but the regularization did help reduce the possibility of overfitting slightly, as the F1 score for training set decreases after adding regularization.

v. Use coef_ to acess the weight vector in L1 regularization Logistic Regression classifier.

We get the top 10 weighted vector in L1 regularization classifier.

```
In [801]: #get the top 10 weighted vector in L1 regularization classifier
          for i in range(10):
              maximum = sorted_coef[i]
              index = list(lg_coef).index(maximum)
              print(vocab_train[index])

tv
death
hollywood
whirlwind
sound
dust
sue
old
mass
middl

In [802]: for i in range(10):
              maximum = sorted_coef[i]
              index = list(lg_coef).index(maximum)
              print(vocab_predict[index])

typhoondevast
damn
heat
win
south
doubl
structur
nw
low
mass
```

Figure 9: Top 10 weighted vector

(f). Bernoulli Naive Bayes.

In this section, we implement Naive Bayes classifier by ourselves. k = 2 hereh since we are predicting p(y = 1) or p(y = 0), whether is a disaster.

F1 score for development set using Bernoulli Classifier is 0.528

```
print('F1 score for development set using Bernoulli Classifier is ', f1_score(y_true, predicted))
print('The accuracy for development prediction is ', (predicted==y_true).mean())

F1 score for development set using Bernoulli Classifier is  0.5284780578898226
```

Figure 10: F1 score for development set

(g). Model comparison

From our results, we can easily find that Naive Bayes Model performs better than the Logistic Regression Model. For Logstic Regression, the best f1_score is 0.5, while for Bayes's, the score is 0.53.

While the generative model make predictions based on joint distribution and model posterior probability $P(y, x)$, the discriminative model make predictions based on joint probability and use Bayes' rule to calculate conditional probability $P(y|x)$.

Thus, one advantage of the generative model is the capability to generate new data similar to the exsiting data using $P(y, x)$, requiring less training data. On the other hand, for the discriminative model, one advantage is that it could take all data into consideration, giving a more comprehensive prediction, which also lead to a disadvantage that the running time becomes much longer.

The Naive Bayes assumes that all features are strongly independent, while the logistic regression does not assume anything related to independence of features but requires no multilinearity among the features. Therefore, for natural language processing problem, bayes model is a great choice because words in sentences could be considered as independent.

(h). N-gram model

In this section, we take N=1 and N=2 gram models to compare with the performance of bags of words, using Logistic Regression and the Bernoulli classifier. We choose the treshold M = 6, meaning that the words at least occur in 6 tweets. In total there are 857 words qualified in the developement set. Then we set max_features = 850 for future training, which requires the size to be the same. We choose this M by checking the different performances with different M. It turns out M = 6 performs the best in most cases.

i. Logistic Regression

Applying the logistic regression on the N-gram model, we get the result **F1 score = 0.448** on the generalization error and **F1 score = 0.824** on the training error.

```
In [653]: predicted = reg.predict(x_train_gram)
          print('F1 score for training set with L1 regularization term is ', f1_score(y_train, predicted))

          F1 score for training set with L1 regularization term is  0.82400906002265

In [654]: predicted = reg.predict(x_predict_gram)
          print('F1 score for development set without regularization term is ', f1_score(y_true, predicted))
          print('The accuracy for development prediction is ', (predicted==y_true).mean())

          F1 score for development set without regularization term is  0.44764957264957267
```

Figure 11: Logistic regression on N-gram model

ii. Bernoulli Classifier

F1 score for development set using Bernoulli Classifier is **0.546** and **0.76** for training set.

```
[657]: idx_gram, logpyx = nb_predictions(x_train_gram, psis, phis)
       print(idx_gram[:10])
       print('F1 score for training set using Bernoulli Classifier term is ', f1_score(y_train, idx_gram))
       print('The accuracy for training prediction is ', (idx_gram==y_train).mean())

       [1 1 1 1 1 1 1 1 1 1]
       F1 score for training set using Bernoulli Classifier term is  0.7605359317904994
       The accuracy for training prediction is  0.8155376243197598
```

Predict the development set using Bernoulli Classifier with N-gram Model

```
[658]: predicted, logpyx_new = nb_predictions(x_predict_gram, psis, phis)
       print('F1 score for development set using Bernoulli Classifier is ', f1_score(y_true, predicted))
       print('The accuracy for development prediction is ', (predicted==y_true).mean())

       F1 score for development set using Bernoulli Classifier is  0.5462920313219716
       The accuracy for development prediction is  0.568739054290718
```

Figure 12: Bernoulli classifier on N-gram model

(i). Determine performance with the test set

Finally, we re-train our classifier using the entire Kaggle training data and use the new model to predict the test set. In this section, we choose **Bernoulli Classifier using the N-gram model** which yield the best performance based on the former results.

We preprocess the training set and test set following the method we present in section (c). Then the processed training dataset is used to fit the Bernoulli classifier. In the end we submit our result to Kaggle.

The Kaggle score we get is **0.57799**

The submission result is lower than we expected. After analyzing, **we think that the one of the possible reasons might be that in preprocessing step, some important words are lemmatized and stemmed into the short word, which might mix with other useless words.** Thus the model is not fully trained.

Another reason could be the size of vocabulary. In our training process, we choose 600, a relatively small size, to train our model. And this might lead to a lose of information. We do not try to use a larger size, but based on our analysis, a larger size of vocabulary would improve our model.

Homework 2

Written Exercises

1. Naive Bayes with Binary Features.

(a). According to the given data, we have

$$P(COVID) = 0.1 \ \ and \ \ P(normal) = 0.9$$

and

$$P(fc|COVID) = 0.75, \ \ P(f|COVID) = 0.05, \ \ P(c|COVID) = 0.05, \ \ P(none|COVID) = 0.15$$

$$P(fc|normal) = 0.04, \ \ P(f|normal) = 0.01, \ \ P(c|normal) = 0.01, \ \ P(none|normal) = 0.94$$

where, fc is fever and coughing, f represents only fever and c represents only coughing.
So we have

$$P(normal|fc) = \frac{P(fc|normal)\,P(normal)}{P(fc)} = \frac{0.04 * 0.9}{0.1 * 0.75 + 0.9 * 0.04} = \frac{0.036}{0.111} = 32.4\%$$

(b). In this section, we also need to calculate $P(no\,COVID|x_1 = fever and x_2 = coughing)$, but the basic probabilty is changed.
Here , we have
$P(x_1 = fever|no\ COVID) = 0.01 + 0.04 = 0.05$, and $P(x_1 = coughing|no\ COVID) = 0.01 + 0.04 = 0.05$

$$\begin{aligned} P(no\,COVID|fc) &= P(no\,COVID)\frac{P(fever|no\,COVID)P(coughing|no\,COVID)}{P(fever)P(coughing)} \\ &= 0.9\frac{0.05 \times 0.05}{(0.05 \times 0.9 + 0.8 \times 0.1) \times (0.05 \times 0.9 + 0.8 \times 0.1)} \\ &= 0.144 \\ &= 14.4\% \end{aligned}$$

(c). Yes, the approach in part (b) does give a significantly different result compared with part (a). In this situation, I would say that the approach we used in part (a) (Bayes theorem) gives a more reasonable result. Because the difference in the assumptions between Naive Bayes and Bayes theorem is that Naive Bayes assumes strong independence between features, which also be reflected in the calculation process. In this case, however, fever and coughing are not strongly independent events. Thus Naive Bayes may underestimate the result.

2. Categorical Naive Bayes.

(a). According to the question 2, we have the maximun likelihood

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^{n} logP_\theta(x^{(i)}, y^{(i)})$$

where $P_\theta(x^{(i)}, y^{(i)}) = P_\theta(y|x)\, P_\theta(x) = P_\theta(x|y)\, P_\theta(y)$, and $\sum_{i=1}^{n} logP_\theta(x^{(i)}, y^{(i)}) = log \prod_{i=1}^{n} P_\theta(x^{(i)}, y^{(i)})$ thus we have

$$\begin{aligned}
l(\theta) = log \prod_{i=1}^{n} P_\theta(x^{(i)}, y^{(i)}) &= log \prod_{i=1}^{n} P_\theta(y)\, P_\theta(x|y) \\
&= log \prod_{i=1}^{n} (\prod_{j=1}^{d} P_\theta(x_j|y)) P_\theta(y) \\
&= \sum_{i=1}^{n} (logP_\theta(y) + \sum_{j=1}^{d} logP_\theta(x_j|y)) \\
&= \sum_{i=1}^{n} [\sum_{i=1}^{K} logP(y=k)^{I(y_i=k)} + \sum_{j=1}^{d} \sum_{l=1}^{S_i} logP(x=a_{jl}|y=k)^{I(x=a_{jl}, y=k)}] \\
&= \sum_{i=1}^{n} [\sum_{k=1}^{K} I(y_i=k) log\phi_k + \sum_{j=1}^{d} \sum_{l=1}^{S_i} I(x=a_{jl}, y=k) log\psi_{jkl}]
\end{aligned}$$

in which, $I(y_i = k)$ is indicator function which represents the number of samples (x_j, y_j) in class k. $I(x = a_{jl}, y = k)$ is indicator function as well. We have,

$$\begin{aligned}
l(\theta) &= \sum_{i=1}^{n} [\sum_{k=1}^{K} I(y_i=k) log\phi^* + \sum_{j=1}^{d} \sum_{l=1}^{S_i} I(x=a_{jl}, y=k) log\psi^*_{jkl}] \\
&= \sum_{k=1}^{K} n_k log\phi_k + \sum_{j=1}^{d} \sum_{l=1}^{S_i} n_{jkl} log\psi_{jkl}
\end{aligned}$$

To optimize ϕ^*, where $\phi^* = (\phi_1, \phi_2, ..., \phi_k)$, we can easily find that only the **first item** in $l(\theta)$ has ϕ^*. To solve the problem, we construct a new function,

$$J(\theta) = \sum_{k=1}^{K} n_k log\phi_k + M(1 - \sum_{k=1}^{K} \phi_k) \tag{1}$$

Because $\sum_{k=1}^{K} \phi_k = 1$, so that the equation (1) works. Then we have

$$\frac{\partial J(\theta)}{\partial \phi^*} = \frac{n_k}{\phi_k} - M = 0$$

$$\sum n_k = M \sum \phi_k$$

$$M = n$$

Thus we have

$$\frac{\partial J(\theta)}{\partial \phi^*} = \frac{n_k}{\phi_k} - n = 0$$

$$\phi^* = \frac{n_k}{n}$$

(b). Same as what we have done in part (a), we have

$$l(\theta) = \sum_{k=1}^{K} n_k log \phi_k + \sum_{j=1}^{d} \sum_{l=1}^{S_i} n_{jkl} log \psi_{jkl}$$

To optimize ψ_{jkl}, we can easily find that only the **second item** in $l(\theta)$ has ψ_{jkl}. To solve the problem, we construct a new function,

$$J(\psi) = \sum_{j=1}^{d} \sum_{l=1}^{S_i} n_{jkl} log \psi_{jkl} + M(1 - \sum_{l=1}^{S_i} \psi_{jkl}) \tag{2}$$

Because $\sum_{l=1}^{S_i} \psi_{jkl} = 1$, so that the equation (2) works. Then we have

$$\frac{\partial J(\theta)}{\partial \psi^*} = \frac{n_{jkl}}{\psi_{jkl}} - M = 0$$

$$M = n_k$$

Thus we have

$$\frac{\partial J(\theta)}{\partial \psi^*} = \frac{n_{jkl}}{\psi_{jkl}} - n_k = 0$$

$$\psi^* = \frac{n_{jkl}}{n_k}$$