AML Homework 2

Yue (Billy) Liu (yl992)

Programming Exercise

Question 1

(a)

Download the training and test data from Kaggle

```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
print("Training data have " + str(train.shape[0]) + " data points.")
print("Testing data have " + str(test.shape[0]) + " data points.")

num_fake = train['target'].value_counts(dropna=False)[0]
num_real = train['target'].value_counts(dropna=False)[1]

print("Percentage of real disasters: " + str(round(num_real/(num_real+num_fake),3)))
print("Percentage of not real disasters: " + str(round(num_fake/(num_real+num_fake),3)))
```

How many training and test data points are there?

```
Training data have 7613 data points.
Testing data have 3263 data points.
```

What percentage of the training tweets are of real disasters, and what percentage is not?

```
Percentage of real disasters: 0.43
Percentage of not real disasters: 0.57
```

(b)

Split the training data into training (70%) and development (30%)

```
x = train.iloc[:,:-1]
y = train.iloc[:,-1:]
x_train, x_dev, y_train, y_dev = train_test_split(x, y, test_size=0.30)
```

(c)

Preprocess the data

For reasons for preprocessing steps please refer to comments inside the code block

```
def preprocess(text):
    stopword = stopwords.words('english')
    wordnet_lemmatizer = WordNetLemmatizer()
    #Tokenize text, because this is required for later preprocessing
    word_tokens = nltk.word_tokenize(text)
    #Convert to lowercase, because I want to avoid repeated vocabulary
    word_tokens = [w.lower() for w in word_tokens]
    #Lemmatization and stripping stopwords, because I want to avoid repeated vocabulary
and stopwords are useless
    word_tokens = [wordnet_lemmatizer.lemmatize(w) for w in word_tokens]
    #Strip punctuation,@,and urls, because they are useless
    word_tokens = [w for w in word_tokens if w not in punctuation]
    return word_tokens


tmp = []
for text in train['text']:
    tmp.append(preprocess(text))
train['text'] = tmp

tmp = []
for text in test['text']:
    tmp.append(preprocess(text))
test['text'] = tmp

x = train.iloc[:,:-1]
y = train.iloc[:,-1:]
x_train, x_dev, y_train, y_dev = train_test_split(x, y, test_size=0.30)
```

(d)

Bag of words model

```
def token_to_sentence(df):#Convert the tokenized 'text' field back to sentence in order to
be vectorized
    tmp = []
    for sentence in df['text']:
        str = ''
        for token in sentence:
```

```
            str += token + ' '
        tmp.append(str)
    return tmp

train_text = token_to_sentence(x_train)
dev_text = token_to_sentence(x_dev)
test_text = token_to_sentence(test)


x_train['text'] = train_text
x_dev['text'] = dev_text
test['text'] = test_text

# vectorizer = CountVectorizer(binary=True, min_df=10)
# vectorizer.fit(x_train["text"])

# x_train_bow = vectorizer.transform(x_train['text'])
# x_dev_bow = vectorizer.transform(x_dev['text'])

M = 0
max_score = -9999

for i in range(50):
    vectorizer = CountVectorizer(binary=True, min_df=i)
    vectorizer.fit(x_train["text"])#Build a vocabulary of the words appearing in the
training set

    x_train_bow = vectorizer.transform(x_train['text'])
    x_dev_bow = vectorizer.transform(x_dev['text'])

    model_logistic = LogisticRegression(penalty='none')
    model_logistic.fit(x_train_bow, y_train)

    y_dev_predict = model_logistic.predict(x_dev_bow)

    if f1_score(y_dev, y_dev_predict) > max_score:
        max_score = f1_score(y_dev, y_dev_predict)
        M = i

vectorizer = CountVectorizer(binary=True, min_df=M)
vectorizer.fit(x_train["text"])#Build a vocabulary of the words appearing in the training
set

x_train_bow = vectorizer.transform(x_train['text'])#Bag of words for training data
x_dev_bow = vectorizer.transform(x_dev['text'])#Bag of words for training data
```

```
print('Total number of features of these vectors (They should be equal since vectorizer is
the same): ' + str(x_train_bow.shape[1]))
```

How to find appropriate M:

I used a for loop to find M that result in the highest performance in development set

```
for i in range(50):
    vectorizer = CountVectorizer(binary=True, min_df=i)
    vectorizer.fit(x_train["text"])#Build a vocabulary of the words appearing in the
training set

    x_train_bow = vectorizer.transform(x_train['text'])
    x_dev_bow = vectorizer.transform(x_dev['text'])

    model_logistic = LogisticRegression(penalty='none')
    model_logistic.fit(x_train_bow, y_train)

    y_dev_predict = model_logistic.predict(x_dev_bow)

    if f1_score(y_dev, y_dev_predict) > max_score:
        max_score = f1_score(y_dev, y_dev_predict)
        M = i
```

Report the total number of features in these vectors

```
Total number of features of these vectors (They should be equal since vectorizer is the same): 15988
```

(e)

Logistic regression

i.

Logistic Regression Classifier model without regularization terms

```
model_logistic = LogisticRegression(penalty='none')
model_logistic.fit(x_train_bow, y_train)

y_train_predict = model_logistic.predict(x_train_bow)
print('F1-Score for training set without regularization: ' + str(f1_score(y_train,
y_train_predict)))

y_dev_predict = model_logistic.predict(x_dev_bow)
print('F1-Score for development set without regularization: : ' + str(f1_score(y_dev,
y_dev_predict)))
```

```
F1-Score for training set without regularization: 0.8253895508707608
F1-Score for development set without regularization: 0.7476340694006309
```

ii.

Logistic Regression Classifier model with L1 regularization

```
model_logistic_L1 = LogisticRegression(penalty='l1', solver='liblinear')
model_logistic_L1.fit(x_train_bow, y_train)

y_train_predict = model_logistic_L1.predict(x_train_bow)
print('F1-Score for training set with L1 regularization: ' + str(f1_score(y_train,
y_train_predict)))

y_dev_predict = model_logistic_L1.predict(x_dev_bow)
print('F1-Score for development set with L1 regularization: ' + str(f1_score(y_dev,
y_dev_predict)))
```

```
F1-Score for training set with L1 regularization: 0.8331399396611743
F1-Score for development set with L1 regularization: 0.7572502685284639
```

Comment on whether you observe any issues with overfitting or underfitting.

There are some overfitting occured as training set score is higher than development set score.

iii.

Logistic Regression Classifier model with L2 regularization

```
model_logistic_L2 = LogisticRegression(penalty='l2')
model_logistic_L2.fit(x_train_bow, y_train)

y_train_predict = model_logistic_L2.predict(x_train_bow)
print('F1-Score for training set with L2 regularization: ' + str(f1_score(y_train,
y_train_predict)))

y_dev_predict = model_logistic_L2.predict(x_dev_bow)
print('F1-Score for development set with L2 regularization: ' + str(f1_score(y_dev,
y_dev_predict)))
```

```
F1-Score for training set with L2 regularization: 0.8429752066115701
F1-Score for development set with L2 regularization: 0.7579173376274826
```

iv.

Which one of the three classifiers performed the best on your training and development set?

The logistic classifier without any regularization performed the best on training set.
The logistic classifier with L2 regularization performed the best on development set.

Did you observe any overfitting and did regularization help reduce it?

Yes, there is overfitting across all three logistic classifier since the training set performance is higher than development set performance. However, regularization does apear to reduce overfitting since the gap between performance on training and development is shrinked after regularization

v.

What are the most important words for deciding whether a tweet is about a real disaster or not?

```
#Weights
weights = model_logistic_L1.coef_.flatten()

df_weights = pd.DataFrame({'Label': vectorizer.get_feature_names(), 'Weight': weights})
df_weights = df_weights.sort_values(by=['Weight'], ascending=False)
df_weights.head(5)
```

	Label	Weight
275	debris	3.443252
1131	wildfire	3.441325
862	russia	3.195901
635	massacre	3.081399
1063	typhoon	3.014411

Top 3 words: 'debris', 'wildfire', and 'Russia'.

(f)

Bernoulli Naive Bayes

```
x_train_2 = x_train_bow.toarray()
y_train_arr = y_train["target"].values

n = x_train_2.shape[0]
d = x_train_2.shape[1]
alpha = 1
K = 2
psis = np.zeros([K,d])
```

```
phis = np.zeros([K])

for k in range(K):
    X_k = x_train_2[y_train_arr == k]
    psis[k] = np.mean(X_k, axis=0)
    phis[k] = (X_k.shape[0] + alpha) / (float(n) + alpha * 2)

def nb_predictions(x, psis, phis):
    n, d = x.shape
    x = np.reshape(x, (1, n, d))
    psis = np.reshape(psis, (K, 1, d))

    psis = psis.clip(1e-14, 1-1e-14)

    logpy = np.log(phis).reshape([K,1])
    logpxy = x * np.log(psis) + (1-x) * np.log(1-psis)
    logpyx = logpxy.sum(axis=2) + logpy

    return logpyx.argmax(axis=0).flatten(), logpyx.reshape([K,n])

y_train_predict, logpyx = nb_predictions(x_train_2, psis, phis)
print("F1-score for training set is: " + str(f1_score(y_train, y_train_predict)))
y_dev_predict, logpyx2 = nb_predictions(x_dev_bow.toarray(), psis, phis)
print("F1-score for development set is: " + str(f1_score(y_dev, y_dev_predict)))
```

```
F1-score for training set is: 0.7723726480036714
F1-score for development set is: 0.7446808510638299
```

(g)

Model comparison

Which model performed the best in predicting whether a tweet is of a real disaster or not? Include your performance metric in your response.

Discriminative classifier (Logistic classifier with L2 regularization) performed the best with development set, but not by much compared to generative classifier. In fact, generative classifier performed better than logistic classifier without regularization

Comment on the pros and cons of using generative vs discriminative models.

Generative Model Pros:
Can do more than just prediction: generation, fill-in missing features, etc.
Can include extra prior knowledge: if prior knowledge is correct, model will be more accurate.
Understandability: it is easier to understand the result
Outlier detection: we may detect via $p(x')$ if x' is an outlier.
Scalability: Simple formulas for maximum likelihood parameters.

Generation: we can sample $x \sim p(x \mid y)$ to generate new data (images, audio)

Generative Model Cons:
Built on assumption: Generative model assumes independence which is not always true, causing over or under confident of model
Computationally expensive

Discriminative model Pros:
Most state-of-the-art algorithms for classification are discriminative (including neural nets, boosting, SVMs, etc.)
They are often more accurate because they make fewer modeling assumptions.
Computationally cheaper

Discriminative model Cons:
Cannot generate data
Hard to understand

Think about the assumptions that Naive Bayes makes. How are the assumptions different from logistic regressions?

Naive bayes model assumes words are uncorrelated while logistic regression model does not.

Discuss whether it's valid and efficient to use Bernoulli Naive Bayes classifier for natural language texts.

Intuitively, Naive Bayes assumes words are independent, which is probably not a good assumption for natural language understanding but it still performed quite well in some text classification tasks. I think it really depend on the task. In this case, there are only 2 categories and the texts are relatively short. Therefore, there is no need to capture the context of the sentence. But for longer text such as paragraphs and tasks such as topic classification, naive bayes will struggle.

(h)

N-gram model

How to choose M?

I used the same method in (d) which I used a for loop to find the optimum M for the randomized dataset

```
M = 0
max_score = -9999


for i in range(50):
    vectorizer2 = CountVectorizer(binary=True,min_df=i, ngram_range=(1,2))
    vectorizer2.fit(x_train["text"])

    x_train_bow_2 = vectorizer2.transform(x_train['text'])
    x_dev_bow_2 = vectorizer2.transform(x_dev["text"])

    #Regression with N-gram
```

```
    model_logistic_Ngram = LogisticRegression(penalty='l2')
    model_logistic_Ngram.fit(x_train_bow_2, y_train)

    y_dev_predict_Ngram = model_logistic_Ngram.predict(x_dev_bow_2)

    if f1_score(y_dev, y_dev_predict_Ngram) > max_score:
        max_score = f1_score(y_dev, y_dev_predict_Ngram)
        M = i

#N-Gram models
vectorizer2 = CountVectorizer(binary=True,min_df=M, ngram_range=(1,2))

#Fitting and processing countVectorizer
vectorizer2.fit(x_train["text"])
x_train_bow_2 = vectorizer2.transform(x_train["text"])
x_dev_bow_2 = vectorizer2.transform(x_dev["text"])

dict = vectorizer2.get_feature_names()
count_1gram = 0
count_2gram = 0
for word in dict:
    if ' ' in word:
        count_2gram += 1
    else:
        count_1gram += 1

print("Number of 1 grams: " + str(count_1gram))
print("Number of 2 grams: " + str(count_2gram))

tmp = []
for word in dict[1000:]: #some vocabs in the beginning contains numbers so I skipped those
    if ' ' in word:
        tmp.append(word)
    if len(tmp)==10:
        break

print("Some 2 grams in the vocabulary: ")
print(tmp)



#N-Gram models
vectorizer3 = CountVectorizer(binary=True,min_df=M, ngram_range=(2,2))#2 gram
vectorizer3.fit(x_train["text"])
x_train_bow_3 = vectorizer3.transform(x_train["text"])
x_dev_bow_3 = vectorizer3.transform(x_dev["text"])
```

```
#Regression with N-gram
model_logistic_Ngram = LogisticRegression(penalty='l2')
model_logistic_Ngram.fit(x_train_bow_3, y_train)

#F-1 score with l2 regularization
y_train_predict_Ngram = model_logistic_Ngram.predict(x_train_bow_3)
print("F1-score for training set with L2 Regularization is: " + str(f1_score(y_train,
y_train_predict_Ngram)))
y_dev_predict_Ngram = model_logistic_Ngram.predict(x_dev_bow_3)
print("F1-score for development set with L2 Regularization is: " + str(f1_score(y_dev,
y_dev_predict_Ngram)))


x_train_3 = x_train_bow_3.toarray()

n = x_train_3.shape[0]
d = x_train_3.shape[1]
alpha = 1
K = 2
psis = np.zeros([K,d])
phis = np.zeros([K])

for k in range(K):
    X_k = x_train_3[y_train_arr == k]
    psis[k] = np.mean(X_k, axis=0)
    phis[k] = (X_k.shape[0] + alpha) / (float(n) + alpha * 2)

y_train_predict, logpyx3 = nb_predictions(x_train_3, psis, phis)
print("F1-score for training set is: " + str(f1_score(y_train, y_train_predict)))
y_dev_predict, logpyx4 = nb_predictions(x_dev_bow_3.toarray(), psis, phis)
print("F1-score for development set is: " + str(f1_score(y_dev, y_dev_predict)))
```

Report the total number of 1-grams and 2-grams in your vocabulary

```
Number of 1 grams: 3264
Number of 2 grams: 3698
```

Take 10 2-grams from your vocabulary, and print them out

```
Some 2 grams in the vocabulary:
['but is', 'but it', 'but never', 'but no', 'but not', 'but she', 'but still', 'but that', 'but the', 'but they']
```

Report your results on *training* and *development* set

```
F1-score for training set with L2 Regularization is: 0.818314661471018
F1-score for development set with L2 Regularization is: 0.6417636252296386
```

```
F1 score for training set is: 0.7554291623578077
```

```
F1-score for training set is: 0.7554291623578077
F1-score for development set is: 0.628645495787427
```

Do these results differ significantly from those using the bag of words model? Discuss what this implies about the task.

The model using 2 gram is significantly worse than using bag of words. Intuitively, 2 grams vocabs consists of 2 words, therefore , making the chance of being tokenized into 1 smaller. Additionally, the number of features is reduced compared to bag of words. The result is that the feature is sparser compared to bag of words.

(i)

```
text = token_to_sentence(train)
train['text'] = text
y_train = train.iloc[:,-1:]

test_bow = vectorizer.transform(test['text'])#Bag of words for testing data
vectorizer.fit(train["text"])
x_train_bow_3 = vectorizer.transform(train["text"])
test_bow = vectorizer.transform(test["text"])

model_logistic_L2 = LogisticRegression(penalty='l2')
model_logistic_L2.fit(x_train_bow_3, y_train)

y_train_predict = model_logistic_L2.predict(test_bow)

with open("submission.csv","w") as f:
    writer = csv.writer(f)
    row = ["id", "target"]
    writer.writerow(row)
    for i in range(len(y_train_predict)):
        row = [test['id'][i], y_train_predict[i]]
        writer.writerow(row)
```

Model of choice: L2-regularized logistic classifier with bag of word vectorization

F1-score reported by Kaggle:

Kaggle F1-score: 0.79865

Was this lower or higher than you expected?

This is higher than I expected since the model is performing better on the testing data than the dev data

This is higher than I expected since the model is performing better on the testing data than the dev data. It could be that now the training data is about 43% larger because it includes the original development data, therefore, the model is probably better than before.

Written Exercise

1. Naive Bayes with Binary Features

(a)

Denote M as the event that a patient have COVID-19

Denote F as the event that a patient have fever

Denote C as the evetn that a patient have coughing

$$
\begin{aligned}
P(\neg M|F \cap C) &= \frac{P(F \cap C|\neg M) \cdot P(\neg M)}{P(F \cap C)} \\
&= \frac{P(F \cap C|\neg M) \cdot P(\neg M)}{P(F \cap C|M) \cdot P(M) + P(F \cap C|\neg M) \cdot P(\neg M)} \\
&= \frac{0.04 \cdot (1 - 0.1)}{0.75 \cdot 0.1 + 0.04 \cdot (1 - 0.1)} \\
&=' 0.3243
\end{aligned}
$$

(b)

Naive Bayes Model assumes features are conditionally independet from one another

$$
\begin{aligned}
P(\neg M|F \cap C) &= \frac{P(\neg M) \cdot P(F|\neg M) \cdot P(C|\neg M)}{P(F|M) \cdot P(C|M) \cdot P(M) + P(F|\neg M) \cdot P(C|\neg M) \cdot P(\neg M)} \\
&= \frac{0.9 \cdot 0.05 \cdot 0.05}{(0.75 + 0.05) \cdot (0.75 + 0.05) \cdot 0.1 + (0.04 + 0.01) \cdot (0.04 + 0.01) \cdot 0.9} \\
&=' 0.0340
\end{aligned}
$$

(c)

Yes, the results differ by a magnitude, which is rather significant. I believe the bayes theorm answer is more reliable because in reality, fever and cough are highly correlated, but naive bayes model assumes they are independent. Thus, fitting an naive bayes model by enforcing conditional independence between features will be under-confident.

2. Categorical Naive Bayes

2. Categorical Naive Bayes

(a)

We want to maximize the log-likelihood L(θ)

$$
\begin{aligned}
L(\theta) &= \frac{1}{n}\sum_{i=1}^{n} logP_\theta(x^{(i)}, y^{(i)}) \\
&= \frac{1}{n}\sum_{i=1}^{n}[log(\phi_k) + \sum_{j=1}^{d}\sum_{l=1}^{l} log(\psi_{jkl})] \\
&= \frac{1}{n}\sum_{i=1}^{k} n_k \cdot log(\phi_k) + \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{d}\sum_{l=1}^{l} n_{jkl} \cdot log(\psi_{jkl})
\end{aligned}
$$

We have a hidden known constraint :

$$
\sum_{i=1}^{k} \phi_k = 1
$$

Taking the partial derivative with Larange multiplier:

$$
\begin{aligned}
L'(\theta, \lambda) &= L(\theta) + \lambda \cdot \frac{1}{n} \cdot (1 - \sum_{i=1}^{k} \phi_k) \\
\frac{\partial}{\partial \theta_i} L'(\theta, \lambda) &= \frac{\partial}{\partial \theta_i} L(\theta) + \frac{\partial}{\partial \theta_i} \lambda \cdot (1 - \sum_{i=1}^{k} \phi_k) = 0 \\
\frac{\partial}{\partial \theta_i} \frac{1}{n} \sum_{i=1}^{k} n_k \cdot log(\phi_k) - \lambda \cdot \frac{1}{n} \cdot \frac{\partial}{\partial \theta_i} \sum_{i=1}^{k} \phi_k &= 0 \\
\frac{n_k}{\phi_k} - \lambda &= 0 \\
\phi_k &= \frac{n_k}{\lambda} \\
\sum_{i=1}^{k} \phi_k &= \sum_{i=1}^{k} \frac{n_k}{\lambda} \\
\frac{1}{\lambda} \sum_{i=1}^{k} n_k &= 1 \\
\lambda &= n
\end{aligned}
$$

Subsitute λ with n, we have

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

(b)

(b)

Similar to (a), we want to maximize the log-likelihood L(θ)

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{k} n_k \cdot log(\phi_k) + \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{d}\sum_{l=1}^{l} n_{jkl} \cdot log(\psi_{jkl})$$

We have a hidden known constraint :

$$\sum_{i=1}^{k}\sum_{j=1}^{d}\sum_{l=1}^{l} \psi_{jkl} = 1$$

Taking the partial derivative with Larange multiplier:

$$L'(\theta, \lambda) = L(\theta) + \lambda \cdot \frac{1}{n} \cdot (1 - \sum_{i=1}^{k}\sum_{j=1}^{d}\sum_{l=1}^{l} \psi_{jkl}$$

$$\frac{\partial}{\partial \theta_i} L'(\theta, \lambda) = \frac{\partial}{\partial \theta_i} L(\theta) + \frac{\partial}{\partial \theta_i} \lambda \cdot (1 - \sum_{i=1}^{k}\sum_{j=1}^{d}\sum_{l=1}^{l} \psi_{jkl}) = 0$$

$$\frac{\partial}{\partial \theta_i} \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{d}\sum_{l=1}^{l} n_{jkl} \cdot log(\psi_{jkl}) - \lambda \cdot \frac{\partial}{\partial \theta_i} \frac{1}{n} \cdot \sum_{i=1}^{k}\sum_{j=1}^{d}\sum_{l=1}^{l} \psi_{jkl} = 0$$

$$\frac{n_{jkl}}{\psi_{jkl}} - \lambda = 0$$

$$\psi_{jkl} = \frac{n_{jkl}}{\lambda}$$

$$\sum_{j=1}^{d}\sum_{l=1}^{l} \psi_k = \sum_{j=1}^{d}\sum_{l=1}^{l} \frac{n_{jkl}}{\lambda}$$

$$\sum_{j=1}^{d}\sum_{l=1}^{l} \frac{n_{jkl}}{\lambda} = 1$$

$$\lambda = n_k$$

Substitute λ with nk, we have

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