Programming Exercises

0.1 Binary Classification on Text Data

0.1.1 Download the Data

We download the train and the test dataset as shown.

```
[3] train_df = pd.read_csv("train.csv", low_memory=False)
    test_df = pd.read_csv("test.csv", low_memory=False)
    train_df
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1
...	...	...	...	...	...
7608	10869	NaN	NaN	Two giant cranes holding a bridge collapse int...	1
7609	10870	NaN	NaN	@aria_ahrary @TheTawniest The out of control w...	1
7610	10871	NaN	NaN	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	1
7611	10872	NaN	NaN	Police investigating after an e-bike collided ...	1
7612	10873	NaN	NaN	The Latest: More Homes Razed by Northern Calif...	1

7613 rows × 5 columns

1. In total, there are 7613 training data points and 3263 test data points. In the training dataset, 43% of the data points represent a real disasters.

```
[ ] # Percentage of real disastes

    train_df['target'].value_counts()
```

```
0    4342
1    3271
Name: target, dtype: int64
```

```
[ ] 3271/(3271+4342)
```

```
0.4296597924602653
```

0.1.2 Split the Training Data

We split the training dataset using sklearn's train_test_split. We choose the train size as 0.7 as we want to randomly pick 70% of the dataset for our training set, and the rest as the development set.

```
[ ] from sklearn.model_selection import train_test_split

    X = train_df.drop(columns=['target']).copy()
    y = train_df['target']

    X_train, X_dev, y_train, y_dev = train_test_split(X, y, train_size=0.7)
```

```
X_train
```

	id	keyword	location	text
1902	2733	crushed	Sunny South florida	WRAPUP 2-U.S. cable TV companies' shares crush...
2900	4166	drown	NaN	@Lwilliams_13 I'll drown you in the river walk
5591	7978	razed	NaN	The Latest: More homes razed by Northern Calif...
4320	6134	hellfire	Riyadh ')	Hellfire! We don‰Ûªt even want to think about ...
3997	5676	floods	Global-NoLocation	#flood #disaster Bengal floods: CM Mamata Bane...
...	...	...	...	...
3522	5035	eyewitness	india	Read a Schoolboy‰Ûªs Eyewitness Account of Hir...
2206	3161	deluge	Los Angeles, CA	RT @NLM_DIMRC: A deluge of resources on #flood...
767	1110	blew%20up	california mermaid ?	Some guy whistled at me in the parking lot &am...
2410	3469	derailed	Washington, DC	[UPDATE] No-Passenger Metro Train Derails Caus...
573	829	bioterror	Washington D.C.	News: FedEx no longer to transport bioterror g...

5329 rows × 4 columns

0.1.3 Preprocessing

1. Convert to lowercase: Because we want to standardise the data, we get rid of the uppercase/lowercase inconsistencies by converting all texts to lowercase.

```
[ ] def make_lowercase(df, colname):
      df[colname] = df[colname].str.lower()

      return df

    X_train = make_lowercase(X_train, 'text')
    X_dev = make_lowercase(X_dev, 'text')
    X_test = make_lowercase(test_df, 'text')
    train_whole = make_lowercase(train_df, 'text')

    X_train
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

	id	keyword	location	text
1902	2733	crushed	Sunny South florida	wrapup 2us cable tv companies shares crushed a...
2900	4166	drown	NaN	lwilliams_13 ill drown you river walk
5591	7978	razed	NaN	latest more homes razed by northern california...
4320	6134	hellfire	Riyadh ')	hellfire we donûªt even want to think about me...
3997	5676	floods	Global-NoLocation	flood disaster bengal floods cm mamata banerje...
...	...	...	...	...
3522	5035	eyewitness	india	read schoolboyûªs eyewitness account of hirosh...
2206	3161	deluge	Los Angeles, CA	rt nlm_dimrc deluge of resources on floods for...
767	1110	blew%20up	california mermaid ?	some guy whistled at me parking lot amp did no...
2410	3469	derailed	Washington, DC	update nopassenger metro train derails causing...
573	829	bioterror	Washington D.C.	news fedex no longer to transport bioterror ge...

5329 rows × 4 columns

2. Remove punctuation and special characters: Due to the nature of the tweets containing many special characters such as @, and _, we remove the punctuation and the special characters using regular expressions.

```
def remove_punc(df, column):
  df[column] = df[column].str.replace('[^\w\s]','')
  return df


X_train = remove_punc(X_train, 'text')
X_dev = remove_punc(X_dev, 'text')
X_test = remove_punc(X_test, 'text')
train_whole = remove_punc(train_whole, 'text')

X_train
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

	id	keyword	location	text
1902	2733	crushed	Sunny South florida	wrapup 2us cable tv companies shares crushed a...
2900	4166	drown	NaN	lwilliams_13 ill drown you river walk
5591	7978	razed	NaN	latest more homes razed by northern california...
4320	6134	hellfire	Riyadh ')	hellfire we donû³t even want to think about me...
3997	5676	floods	Global-NoLocation	flood disaster bengal floods cm mamata banerje...
...	...	...	...	...
3522	5035	eyewitness	india	read schoolboyû³s eyewitness account of hirosh...
2206	3161	deluge	Los Angeles, CA	rt nlm_dimrc deluge of resources on floods for...
767	1110	blew%20up	california mermaid ?	some guy whistled at me parking lot amp did no...
2410	3469	derailed	Washington, DC	update nopassenger metro train derails causing...
573	829	bioterror	Washington D.C.	news fedex no longer to transport bioterror ge...

5329 rows × 4 columns

3. Strip stop words: We make a list of the most common stop words used, and delete them from the text data.

```
def strip_stop(df, column):
  stop_words = ["and", "or", "the", "just", "my", "a", "an", "mine", "also", "any", "are", "is", "be", "but", "each", "else", "if", "in", "it", "your",
  df[column] = [' '.join([item for item in x.split()
                 if item not in stop_words])
                 for x in df[column]]

  return df

X_train = strip_stop(X_train, 'text')
X_dev = strip_stop(X_dev, 'text')
X_test = strip_stop(X_test, 'text')
train_whole = strip_stop(train_whole, 'text')

X_train
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

	id	keyword	location	text
1902	2733	crushed	Sunny South florida	wrapup 2us cable tv companies shares crushed a...
2900	4166	drown	NaN	lwilliams_13 ill drown you river walk
5591	7978	razed	NaN	latest more homes razed by northern california...
4320	6134	hellfire	Riyadh ')	hellfire we donûªt even want to think about me...
3997	5676	floods	Global-NoLocation	flood disaster bengal floods cm mamata banerje...
...	...	...	...	...
3522	5035	eyewitness	india	read schoolboyûªs eyewitness account of hirosh...
2206	3161	deluge	Los Angeles, CA	rt nlm_dimrc deluge of resources on floods for...
767	1110	blew%20up	california mermaid ?	some guy whistled at me parking lot amp did no...
2410	3469	derailed	Washington, DC	update nopassenger metro train derails causing...
573	829	bioterror	Washington D.C.	news fedex no longer to transport bioterror ge...

5329 rows × 4 columns

4. Lemmatise the tweets: Using the Natural Language Processing Toolkit, we lemmatise the text points to get a list of words for each data point. In order to do this, we use the Whitespace Tokeniser and the WordNetLemmatiser from the NLP Toolkit. This will make the process of picking the threshold M easier in the next part.

```
[ ] !pip install -q wordcloud
    import wordcloud

    import nltk
    nltk.download('wordnet')

    w_tokeniser = nltk.tokenize.WhitespaceTokenizer()
    lemmatiser = nltk.stem.WordNetLemmatizer()

    def lemmatize_text(text):
        return [lemmatiser.lemmatize(w) for w in w_tokeniser.tokenize(text)]

    X_train_lemmatised = X_train.copy()

    X_train_lemmatised['text'] = X_train_lemmatised.text.apply(lemmatize_text)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
X_train_lemmatised
```

	id	keyword	location	text
1902	2733	crushed	Sunny South florida	[wrapup, 2us, cable, tv, company, share, crush...
2900	4166	drown	NaN	[lwilliams_13, ill, drown, you, river, walk]
5591	7978	razed	NaN	[latest, more, home, razed, by, northern, cali...
4320	6134	hellfire	Riyadh ')	[hellfire, we, donûªt, even, want, to, think, ...
3997	5676	floods	Global-NoLocation	[flood, disaster, bengal, flood, cm, mamata, b...
...	...	...	...	...
3522	5035	eyewitness	india	[read, schoolboyûªs, eyewitness, account, of, ...
2206	3161	deluge	Los Angeles, CA	[rt, nlm_dimrc, deluge, of, resource, on, floo...
767	1110	blew%20up	california mermaid ?	[some, guy, whistled, at, me, parking, lot, am...
2410	3469	derailed	Washington, DC	[update, nopassenger, metro, train, derails, c...
573	829	bioterror	Washington D.C.	[news, fedex, no, longer, to, transport, biote...

0.1.4 Bag of Words

Using the CountVectoriser, we create a bag of words model. In order to make a sensible decision for the threshold M, in other words, picking words that occur in at least k tweets, we make use of the lemmatised version of the training set. We build a dictionary 'occurrences' to calculate how many tweets each word occurs in.

```
[16] import collections

     # Lemmatising helps us build a giant list of all the stripped words after preprocessing, and we can u
     # each word to make a sensible decision for M. (bag of words)
     word_list = []

     for element in X_train_lemmatised['text'].tolist():
       word_list = word_list + element


     occurrences = collections.Counter(word_list)
     occurrences
```

```
Counter({'experienced': 2,
         'urogyn': 1,
         'trying': 18,
         'to': 1349,
         'help': 59,
         'mesh': 1,
         'injured': 33,
         'woman': 66,
         'talk': 14,
         'worst': 15,
         'offender': 1,
         'httptconpoqlkqup9': 1,
         'meshnewsdesk': 1,
         'find': 22,
         'out': 197,
```

We sort this dictionary by commonality, to see how many times on average words appear on different tweets. As we can observe, most of the tweets occur in only 1 tweet. As the number of word that occur at least in 2 tweets is also relatively high, we pick $M = 3$ as a mathematically derived decision.

```
[19] from collections import Counter

     count = Counter(occurrences.values())
     count
```

```
Counter({1: 11567,
         2: 1681,
         3: 776,
         4: 460,
         5: 305,
         6: 200,
         7: 155,
         8: 130,
         9: 104,
         10: 87,
         11: 74,
         12: 73,
         13: 65,
         14: 64,
         15: 54,
         16: 41,
         17: 37,
         18: 38,
         19: 41,
         20: 38,
         21: 32,
         22: 28,
         23: 27,
         24: 33,
         25: 23,
```

[21]
```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer(binary=True, min_df=3)
X_train_counts = count_vect.fit_transform(X_train.text)
X_test_counts = count_vect.transform(X_test.text)

X_train_counts
```

```
<5329x3376 sparse matrix of type '<class 'numpy.int64'>'
	with 48283 stored elements in Compressed Sparse Row format>
```

[22]
```
X_train_counts.shape
```

```
(5329, 3376)
```

[23]
```
X_test_counts.shape
```

```
(3263, 3376)
```

[24]
```
count_vect.vocabulary_.get("this")
```

```
2941
```

[25]
```
y_train.shape
```

```
(5329,)
```

0.1.5 Logistic Regression

Training Set

i. Logistic Regression with No Regularisation

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

     # Create an instance of Softmax and fit the data.

     logreg = LogisticRegression(penalty='none', C=1e5, multi_class='multinomial', verbose=True)
     logreg.fit(X_train_counts, y_train)


     #predict on the training set
     X_train_predicted = logreg.predict(X_train_counts)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting pe
  "Setting penalty='none' will ignore the C and l1_ratio "
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s finished
```

```
[28] X_train_predicted
```

```
array([1, 1, 0, ..., 0, 0, 0])
```

```
[29] # Calculating the F1 score

     from sklearn.metrics import f1_score

     f1_score(y_train, X_train_predicted, average='weighted')
```

```
0.9853538634598548
```

```
[30] max(logreg.coef_[0])
```

```
47.37046912997119
```

As shown above, the F1 score is quite high, which indicates that there are no signs of overfitting or underfitting, and our model is performing well.

ii. Logistic Regression with L1 Regularisation

```
[31] logreg = LogisticRegression(penalty='l1', solver='liblinear',
                                max_iter=int(1e6),
                                warm_start=True,
                                intercept_scaling=10000.)
     logreg.fit(X_train_counts, y_train)

     #predict on test set
     X_train_predicted_l1 = logreg.predict(X_train_counts)
     X_train_predicted_l1

     array([1, 1, 0, ..., 0, 0, 0])
```

```
f1_score(y_train, X_train_predicted_l1, average='weighted')

0.8809911595373776
```

iii. Logistic Regression with L2 Regularisation

```
[33] logreg = LogisticRegression(penalty='l2', C=1e5, multi_class='multinomial', verbose=True)
     logreg.fit(X_train_counts, y_train)

     #predict on the training set
     X_train_predicted_l2 = logreg.predict(X_train_counts)

     [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
     /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfg
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as shown in:
         https://scikit-learn.org/stable/modules/preprocessing.html
     Please also refer to the documentation for alternative solver options:
         https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
       extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
     [Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s finished
```

```
f1_score(y_train, X_train_predicted_l2, average='weighted')

0.9853440846977901
```

As shown above, the F1 score is quite high for all models on the training set, which indicates that there are no signs of overfitting or underfitting, and our models are performing well.

Development Set

i. Logistic Regression with No Regularisation

```
[35] # First, creating bag of words on the development set.
     count_vect = CountVectorizer(binary=True, min_df=3)
     X_dev_counts = count_vect.fit_transform(X_dev.text)
     X_test_counts = count_vect.transform(X_test.text)

     # Create an instance of Softmax and fit the data.
     logreg = LogisticRegression(penalty='none', C=1e5, multi_class='multinomial', verbose=True)
     logreg.fit(X_dev_counts, y_dev)


     #predict on the training set
     X_dev_predicted = logreg.predict(X_dev_counts)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting pe
  "Setting penalty='none' will ignore the C and l1_ratio "
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s finished
```

```
[43] f1_score(y_dev, X_dev_predicted, average='weighted')
```

```
0.9855491360487559
```

ii. Logistic Regression with L1 Regularisation

```
[44] logreg = LogisticRegression(penalty='l1', solver='liblinear',
                                 max_iter=int(1e6),
                                 warm_start=True,
                                 intercept_scaling=10000.)
     logreg.fit(X_dev_counts, y_dev)

     #predict on test set
     X_dev_predicted_l1 = logreg.predict(X_dev_counts)
     X_dev_predicted_l1
```

```
array([0, 1, 0, ..., 0, 1, 1])
```

```
[45] f1_score(y_dev, X_dev_predicted_l1, average='weighted')
```

```
0.8887577216499724
```

iii. Logistic Regression with L2 Regularisation

```
[46] logreg = LogisticRegression(penalty='l2', C=1e5, multi_class='multinomial', verbose=True)
     logreg.fit(X_dev_counts, y_dev)

     #predict on the training set
     X_dev_predicted_l2 = logreg.predict(X_dev_counts)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s finished
```

```
[47] f1_score(y_dev, X_dev_predicted_l2, average='weighted')
```

```
0.9855421385321214
```

As shown above, the F1 score is quite high for all models on the development set, which indicates that there are no signs of overfitting or underfitting, and our models are performing well.

iv. For the models trained with both the training and the development sets, the performance of no regularisation and L2 regularisation were quite similar as their F1 scores were same until the fifth decimal. L2 regularisation performed better (F1 = 0.985344) on the model trained with the training dataset (by a very small margin), and no regularisation (F1 = 0.985549) performed better on the model trained with the development dataset (also by a very small margin). L1 regularisation's F1 scores were significantly lower (10%) compared to the other models for both sets. Nevertheless, since the F1 scores for all deployed models were still very high and in the good range, no underfitting or overfitting was observed.

v. Using the coef_ attribute of the sklearn's logistic regression function, we inspected the model weights. In order to find the features (in this case, words) with most importance on deciding whether a tweet is a disaster tweet or not, we first printed the vocabulary obtained after applying CountVectorizer. Then, we got the maximum weight from the coef array, and then matched the index of the word with the highest weight value. For the training set, the most important word in prediction was "hiroshima". For the development set on the other hand, it was "set".

[57]
```
count_vect.vocabulary_
```

```
{'trying': 3059,
 'to': 2978,
 'help': 1367,
 'injured': 1507,
 'women': 3288,
 'worst': 3307,
 'find': 1123,
 'out': 2106,
 'how': 1433,
 'fund': 1212,
 'was': 3200,
 'used': 3122,
 'for': 1165,
 'typhoon': 3084,
 'philippines': 2177,
 'see': 2562,
 'relief': 2396,
 'funds': 1213,
 'report': 2409,
 'wreck': 3317,
 'happy': 1326,
 'no': 2013,
```

```
weights = logreg.coef_[0]
max_coef = max(weights)
max_index = np.where(weights == max_coef)
max_index
```

```
(array([1391]),)
```

[69]
```
vocab = count_vect.vocabulary_

keys = [k for k, v in vocab.items() if v == 1391]
print(keys)
```

```
['hiroshima']
```

```
[79] weights = logreg.coef_[0]
     max_coef = max(weights)
     max_index = np.where(weights == max_coef)
     max_index

     (array([1739]),)
```

```
[80] vocab = count_vect.vocabulary_

     keys = [k for k, v in vocab.items() if v == 1391]
     print(keys)

     ['set']
```

0.2 Bernoulli Naive Bayes

Below is the code we used for the Bernoulli Naive Bayes implementation, adapted from the Lecture 6 slides. In order to test the training set on the development set, we added Laplace smoothing in the computation step of parameters.

```
[81] count_vect = CountVectorizer(binary=True, min_df=3)
     X_train_counts = count_vect.fit_transform(X_train.text).toarray()

     X_train_counts.shape

     (5329, 3376)
```

```
[82] X_dev_counts = count_vect.transform(X_dev.text).toarray()
     X_dev_counts

     array([[0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            ...,
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0]])
```

```
[83] n = X_train_counts.shape[0] # size of the dataset
     d = X_train_counts.shape[1] # number of features in our dataset
     K = 2 # number of clases

     # these are the shapes of the parameters
     psis = np.zeros([K,d])
     phis = np.zeros([K])

     # we now compute the parameters
     for k in range(K):
         X_k = X_train_counts[y_train.to_numpy() == k]
         psis[k] = np.mean(X_k, axis=0)
         phis[k] = (X_k.shape[0] + 1) / (float(n) +2)

     # print out the class proportions
     print(phis)
```

```
[0.57268805 0.42731195]
```

```
def nb_predictions(x, psis, phis):
    """This returns class assignments and scores under the NB model.

    We compute \arg\max_y p(y|x) as \arg\max_y p(x|y)p(y)
    """
    # adjust shapes
    n, d = x.shape
    x = np.reshape(x, (1, n, d))
    psis = np.reshape(psis, (K, 1, d))

    # clip probabilities to avoid log(0)
    psis = psis.clip(1e-14, 1-1e-14)

    # compute log-probabilities
    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])

idx, logpyx = nb_predictions(X_dev_counts, psis, phis)
print(idx[:10])
```

```
[0 1 0 1 1 1 1 0 1 0]
```

```
[88] f1_score(y_dev, idx, average='weighted')
```

```
0.7865501258877472
```

```
[86] psis.shape
```

```
(2, 3376)
```

```
[87] X_dev_counts.shape
```

```
(2284, 3376)
```

Looking at the F1 score, we can see that it is 0.7866. This seems to be much lower compared to the F1 scores we got using the logistic regression models.

0.2.1 Model Comparison

- Looking at the above data for F1 scores obtained, it can be clearly seen that logistic regression performed much better than Bernoulli Naive Bayes. Discriminative models are often more accurate as they draw a decision boundary rather than assigning each data point to a group. Intuitively, logistic regression should result in a higher accuracy compared to Bernoulli Naive Bayes at it is a discriminative model, whereas Naive Bayes is a generative model. This can be supported by the F1 results we got from the previous sections. The accuracy between these two algorithms differed up to 20%, which is a very big percentage when considering model precision. Since our data is quite large and we are dealing with words for text classification, using discriminative models make more sense.

  In cases where there are missing values, generative models come in handy as they can be used to make fit the missing values using the distributions. If the dataset is smaller, it can also be beneficial to use generative models depending on the machine learning task.

- Naive Bayes' assumption is that we assume each feature (in this case, each word) to be independent of each other. This is called the independence of predictors. On the other hand, logistic regression assumes that the observations are independent and there is no multicollinearity among variables.

  Since our task is text classification, Naive Bayes' assumption fails at times as some words are often used alongside other words, which establishes dependence between features. Another important thing to note is that if we do not lemmatise the words in the dataset, words derived from the same root (such as drink, drinking and drinker) will be dependent. For these reasons, it is not ideal to use Bernoulli Naive Bayes for text classification tasks.

0.2.2 N-gram Model

We set a new count vectoriser by setting the N-grams to include 1-grams and 2-grams.

```
[89] # N = 2

     count_vect = CountVectorizer(binary=True, min_df=3, ngram_range=(1,2))
     X_train_counts = count_vect.fit_transform(X_train.text).toarray()
     X_dev_counts = count_vect.fit_transform(X_dev.text).toarray()

     X_train_counts.shape

     (5329, 5932)
```

We train the new set with Logistic regression using L2 regularisation.

```
logreg = LogisticRegression(penalty='l2', C=1e5, multi_class='multinomial', verbose=True)
logreg.fit(X_train_counts, y_train)

#predict on the training set
X_train_predicted_l2 = logreg.predict(X_train_counts)

#predict on the development set
logreg.fit(X_dev_counts, y_dev)
X_dev_predicted_l2 = logreg.predict(X_dev_counts)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   18.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.9s finished
```

```
[91] X_train_predicted_l2
```

```
array([1, 1, 0, ..., 0, 0, 0])
```

```
[92] X_dev_predicted_l2
```

```
array([0, 1, 0, ..., 0, 1, 1])
```

We train the new set using Bernoulli Naive Bayes.

```
[93] n = X_train_counts.shape[0] # size of the dataset
     d = X_train_counts.shape[1] # number of features in our dataset
     K = 2 # number of clases

     # these are the shapes of the parameters
     psis = np.zeros([K,d])
     phis = np.zeros([K])

     # we now compute the parameters
     for k in range(K):
         X_k = X_train_counts[y_train.to_numpy() == k]
         psis[k] = np.mean(X_k, axis=0)
         phis[k] = (X_k.shape[0] + 1) / (float(n) +2)


     idx, logpyx = nb_predictions(X_train_counts, psis, phis)
     print(idx[:10])

     [0 1 0 1 0 1 0 0 1 0]
```

```
n = X_dev_counts.shape[0] # size of the dataset
d = X_dev_counts.shape[1] # number of features in our dataset
K = 2 # number of clases

# these are the shapes of the parameters
psis = np.zeros([K,d])
phis = np.zeros([K])

# we now compute the parameters
for k in range(K):
    X_k = X_dev_counts[y_dev.to_numpy() == k]
    psis[k] = np.mean(X_k, axis=0)
    phis[k] = (X_k.shape[0] + 1) / (float(n) +2)


idx, logpyx = nb_predictions(X_dev_counts, psis, phis)
print(idx[:10])

[0 1 0 1 1 1 0 1 1 0]
```

We get the new dataset including 1 and 2-grams. We randomly print 10 samples gathered from the array.

```
[95] features = count_vect.get_feature_names()
     features
```

```
      'floods ur',
      'florida',
      'fog',
      'follow',
      'following',
      'food',
      'food crematoria',
      'football',
      'for',
      'for changes',
      'for first',
```

```
[96] two_grams = [x for x in features if len(x.split()) == 2]
     two_grams[:10]
```

```
     ['12000 nigerian',
      '15 saudi',
      '16yr old',
      '2us cable',
      '30 fires',
      '31 md',
      '3g this',
      '40 families',
      '5km of',
      '70 years']
```

The results of the logistic regression are as follows, 0.9859 and 0.9855 for the training and the development sets respectively.

```
[102] print(f1_score(y_train, X_train_predicted_l2, average='weighted'))
      print(f1_score(y_dev, X_dev_predicted_l2, average='weighted'))
```

```
      0.9859133921702686
      0.9855456840724779
```

The results of the Bernoulli Naive Bayes are as follows, 0.8771 and 0.8756 for the training and the development sets respectively.

Bernoulli Naive Bayes

```
[105] # Training Set

n = X_train_counts.shape[0] # size of the dataset
d = X_train_counts.shape[1] # number of features in our dataset
K = 2 # number of clases

# these are the shapes of the parameters
psis = np.zeros([K,d])
phis = np.zeros([K])

# we now compute the parameters
for k in range(K):
    X_k = X_train_counts[y_train.to_numpy() == k]
    psis[k] = np.mean(X_k, axis=0)
    phis[k] = (X_k.shape[0] + 1) / (float(n) +2)


idx, logpyx = nb_predictions(X_train_counts, psis, phis)
print(idx[:10])
```

```
[0 1 0 1 0 1 0 0 1 0]
```

```
[106] f1_score(y_train, idx, average='weighted')
```

```
0.8770717213823563
```

```
# Development Set


n = X_dev_counts.shape[0] # size of the dataset
d = X_dev_counts.shape[1] # number of features in our dataset
K = 2 # number of clases

# these are the shapes of the parameters
psis = np.zeros([K,d])
phis = np.zeros([K])

# we now compute the parameters
for k in range(K):
    X_k = X_dev_counts[y_dev.to_numpy() == k]
    psis[k] = np.mean(X_k, axis=0)
    phis[k] = (X_k.shape[0] + 1) / (float(n) +2)


idx, logpyx = nb_predictions(X_dev_counts, psis, phis)
print(idx[:10])
```

```
[0 1 0 1 1 1 0 1 1 0]
```

```
f1_score(y_dev, idx, average='weighted')
```

```
0.8756319030079861
```

We can see clearly that using the bag of words model significantly improved Bernoulli Naive Bayes. This was due to the contingency we mentioned in the previous question. When we used bigrams, we eliminated the dependency of commonly together used words in some cases, which improved the accuracy of our model by more than 10%.

0.2.3 Determine the performance of the test set

Retraining the model as follows:

```
[109] count_vect = CountVectorizer(binary=True, min_df=3)
      train_whole_counts = count_vect.fit_transform(train_whole.text)
      X_test_counts = count_vect.transform(X_test.text)
```

```
[110] logreg = LogisticRegression(penalty='l2', C=1e5, multi_class='multinomial', verbose=True)
      logreg.fit(train_whole_counts, y)

      #predict on the test set
      test_whole_predicted_l2 = logreg.predict(X_test_counts)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s finished
```

```
[111] test_whole_predicted_l2
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
id = test_df['id']
results = pd.DataFrame({"id":id, "target": test_whole_predicted_l2})
results.to_csv("NLP_results.csv", index=False)
```

We got 0.735 as our Kaggle score.

Name	Submitted	Wait time	Execution time	Score
NLP_results.csv	a day ago	1 seconds	0 seconds	0.73490

This seems to be lower than expected since the F1 scores calculated above were much higher. This indicated that there is much more tuning to be done to the models, or more accurate models to be deployed for NLP tasks and text classification. Overall, the score is still a good one, showing that Logistic Regression is not a bad alternative for text classification.

Written Exercises

1. a) We have the following probabilities gathered from the data given:

$$P(COVID) = \frac{10}{100}$$
$$P(NoCOVID) = \frac{10}{100}$$
$$P(F, C \mid COVID) = \frac{75}{100}$$
$$P(F, noC \mid COVID) = \frac{5}{100}$$
$$P(noF, C \mid COVID) = \frac{5}{100}$$
$$P(noF, noC \mid COVID) = \frac{15}{100}$$

$$P(F, C \mid noCOVID) = \frac{4}{100}$$
$$P(F, noC \mid noCOVID) = \frac{1}{100}$$
$$P(noF, C \mid noCOVID) = \frac{1}{100}$$
$$P(noF, noC \mid noCOVID) = \frac{94}{100}$$

We write the formula for the given patient's probability of not having COVID given they have fever and cough,

$$P(noCOVID \mid F, C)$$

, using Bayes' rule:

$$P(noCOVID \mid F, C) = \frac{P(F, C \mid noCOVID) \cdot P(noCOVID)}{P(noCOVID) \cdot P(F, C \mid noCOVID) + P(COVID) \cdot P(F, C \mid COVID)}$$

$$P(noCOVID \mid F, C) = \frac{0.04 \cdot 0.9}{0.04 \cdot 0.9 + 0.1 \cdot 0.75}$$

$$= \frac{0.036}{0.036 + 0.075}$$

$$= 0.324$$

b) Considering that the dataset is IID, we adapt the Naive Bayes approach to calculate

$$P(noCOVID \mid F, C)$$

by making the variables (fever and cough) independent of each other, as this is the Naive Bayes assumption.

We then add up the above probabilities by each feature.

$$P(F \mid noCOVID) = P(F, C \mid noCOVID) + P(F, noC \mid noCOVID) = 0.01 + 0.04 = 0.05$$

$$P(C \mid noCOVID) = P(F, C \mid noCOVID) + P(noF, C \mid noCOVID) = 0.01 + 0.04 = 0.05$$

$$P(F \mid COVID) = P(F, C \mid COVID) + P(F, noC \mid COVID) = 0.75 + 0.05 = 0.80$$

$$P(C \mid COVID) = P(F, C \mid COVID) + P(noF, C \mid COVID) = 0.75 + 0.05 = 0.80$$

Then we have as our formula:

$$P(noCOVID \mid F, C) = \frac{P(F \mid noCOVID) \cdot P(C \mid noCOVID) \cdot P(noCOVID)}{P(Observed)}$$

where

$$P(Observed) = P(F \mid noCOVID) \cdot P(C \mid noCOVID) \cdot P(noCOVID) + \\ P(F \mid COVID) \cdot P(C \mid COVID) \cdot P(COVID)$$

$$P(noCOVID \mid F, C) = \frac{0.05 \cdot 0.05 \cdot 0.9}{0.05 \cdot 0.05 \cdot 0.05 + 0.80 \cdot 0.80 \cdot 0.10}$$

$$= 0.034$$

c) As we can see from the above results, the probability in (a) was much higher than the probability in (b). This is due to the fact that we lose information when we adapt the Naive Bayes assumption: that the variables are independent of each other. The Naive Bayes classifier simplifies the relationship between the variables, sampling a smaller chunk of the data when making predictions and calculations. Therefore the difference between the two models is quite high, where the Bayes' rule probability is around 10 times the probability obtained using the Naive Bayes classifier. The first approach seems to be more accurate and applicable.

2. a) In order to find the maximum likelihod estimate, we need to look at the objective function. Setting our objective function to 0 will give us the right MLE. We have, as per the lecture slides;

$$
\begin{aligned}
J(\vec{\phi}) &= \textstyle\sum_{i=1}^{n} \log P_\theta(y^{(i)}; \vec{\phi}) \\
&= \textstyle\sum_{i=1}^{n} \log \phi_{y^{(i)}} - n \cdot \log \sum_{k=1}^{K} \phi_k \\
&= \textstyle\sum_{k=1}^{K} \sum_{i:y^{(i)}=k} \log \phi_k - n \cdot \log \sum_{k=1}^{K} \phi_k
\end{aligned}
$$

We know that

$$\sum_{i:y^(i)=k} = n_k \tag{1}$$

Taking the derivative of

$$J(\phi_k) = \sum_{k=1}^{K} n_k \log \phi_k - n \cdot log \sum_{k=1}^{K} \phi_k \tag{2}$$

we get

$$\frac{dJ(\phi_k)}{d\phi} = n_k \cdot \frac{1}{\phi_k} - \frac{n}{\sum_{k=1}^{K} \phi_k} \tag{3}$$

Setting this to 0,

$$\frac{n_k}{\phi_k} = \frac{n}{\sum_{k=1}^{K} \phi_k} \tag{4}$$

Where

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

. Therefore we get

$$\phi_k = \frac{n_k}{n} \tag{5}$$

b)

$$\ell = \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)}) = \sum_{i=1}^{n}\sum_{j=1}^{d} \log P_\theta(x_j^{(i)}|y^{(i)}) + \sum_{i=1}^{n} \log P_\theta(y^{(i)})$$
$$= \sum_{k=1}^{K}\sum_{j=1}^{d} \underbrace{\sum_{i:y^{(i)}=k} \log P(x_j^{(i)}|y^{(i)};\psi_{jk})}_{\text{all the terms that involve } \psi_{jk}} + \underbrace{\sum_{i=1}^{n} \log P(y^{(i)};\vec{\phi})}_{\text{all the terms that involve } \vec{\phi}} .$$

Taking the derivative of our objective function with respect to

$$\psi_{jk}$$

, we get:

$$\frac{dJ}{d\psi} = K \cdot d \cdot \sum_{i:y^{(i)}=k} log(\psi_{jk}) \tag{6}$$

Each of the log likelihood terms with

$$\psi_{jkl}$$

is equivalent to an instance of an instance of categorical distribution, derived in part (a). Therefore, it follows logically from the derivation in (a) that the MLE for the Bernoulli distribution is the number of data points with class k for which the jth feature equals l over the number of data points with class k.

$$\psi_{jkl} = \frac{n_{jkl}}{n_k} \tag{7}$$