CHALMERS

Applied Machine Learning

DAT341 / DIT867

2023-02-17

Project

Assignment 3 - Group PA 3 1

By

Mirco S. Ghadri

Sameer Jathavedan

Tobias Filmberg

Table of content

1. Introduction

The purpose of this assignment is to develop a supervised machine learning algorithm that can determine whether a given text message represents a positive or negative opinion about Covid-19 vaccination. To achieve this, the model will be trained with labeled data that contains messages with positive opinion about Covid-19 vaccination and messages with negative opinion about Covid-19 vaccination. The goal is to create a machine learning model with as high accuracy as possible in predicting whether a message is pro covid 19-vaccination or anti covid-19 vaccination.

2. Method

The method could be divided into 4 steps. The first step was getting the training and testing data. The second step was preprocessing the training data. The third step was transforming the features, in this case messages, to vectors so that our machine learning algorithm could use them. The 4th step was to train the machine learning algorithm with the vectorized features and labels.

Step 1 - Getting the training and testing data

In order to load the training and testing data into dataframes we first had to import pandas.

```
import pandas as pd
```

We then downloaded the training and testing data and put them in the same folder in our computer as the jupyter notebook where we ran our code. We loaded the training data using the follow piece of code:

```
training_data_final = pd.read_csv("a3_train_final.tsv", sep="\t",
header=None, names = ["Prediction","Message"])
```

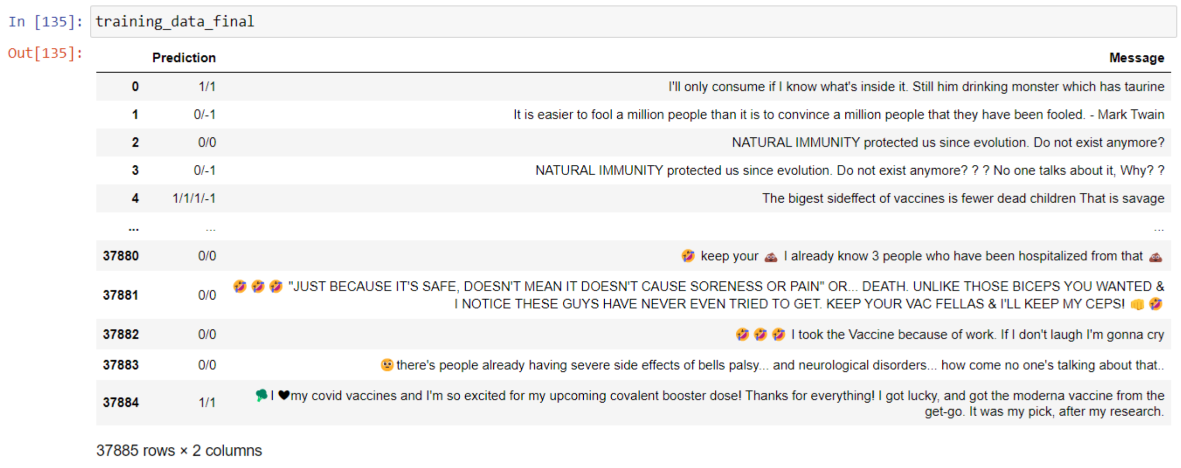
Since the training data was stored in a tsv file (tab-separated values), we had to use the parameter **sep=”\t”** in the read_csv() function. The **header=None** specifies that the tsv file does not contain any header, so the first row of data in the file should be treated as a row of data and not as a header. The names parameter sets the column names. The first column holds the prediction labels(0 for a message that is anti-vaccination and 1 for a message that is pro vaccination or -1 for a message that is neither anti nor pro vaccination). The second column holds the actual message that was classified as a 1,0 or a -1. So for example, the message could be something like “The only way forward for us in this pandemic is the vaccine”.

In order to load the testing data, we did the same procedure as for the training set.

```
test_set = pd.read_csv("a3_test.tsv", sep="\t", header=None, names =
["Prediction", "Message"])
```

Step 2 - Data preprocessing

It was not enough to load the training data into a variable. After inspecting the first few rows of the training data, we could notice that there were multiple labels for each message.



```
In [135]: training_data_final
Out[135]:
```

	Prediction	Message
0	1/1	I'll only consume if I know what's inside it. Still him drinking monster which has taurine
1	0/-1	It is easier to fool a million people than it is to convince a million people that they have been fooled. - Mark Twain
2	0/0	NATURAL IMMUNITY protected us since evolution. Do not exist anymore?
3	0/-1	NATURAL IMMUNITY protected us since evolution. Do not exist anymore? ? ? No one talks about it, Why? ?
4	1/1/1/-1	The bigest sideffect of vaccines is fewer dead children That is savage
...	...	...
37880	0/0	🤣 keep your 💩 I already know 3 people who have been hospitalized from that 💩
37881	0/0	🤣 🤣 🤣 "JUST BECAUSE IT'S SAFE, DOESN'T MEAN IT DOESN'T CAUSE SORENESS OR PAIN" OR... DEATH. UNLIKE THOSE BICEPS YOU WANTED & I NOTICE THESE GUYS HAVE NEVER EVEN TRIED TO GET. KEEP YOUR VAC FELLAS & I'LL KEEP MY CEPS! 👊 🤣
37882	0/0	🤣 🤣 🤣 I took the Vaccine because of work. If I don't laugh I'm gonna cry
37883	0/0	😟 there's people already having severe side effects of bells palsy... and neurological disorders... how come no one's talking about that..
37884	1/1	🥦 I 🖤my covid vaccines and I'm so excited for my upcoming covalent booster dose! Thanks for everything! I got lucky, and got the moderna vaccine from the get-go. It was my pick, after my research.

37885 rows × 2 columns

The reason that the messages had multiple labels was that the messages had been classified by multiple people and some people might have disagreed or agreed on the classification of the message. Each label is the classification of one person and the labels of different people are separated by a "/". In the first row for example, the label is "1/1". This means that 1 person has classified the message as 1(pro vaccine) and another person has also classified it as a 1(pro vaccine). The second row shows the label "0/-1". This means that the first person classified the message as anti covid-19 vaccine and the second person classified the message as neither anti nor pro covid-19 vaccination.
In order to train our model, we could not have ambiguous labels. Therefore we had to find a way to make the labels with multiple classification into a single classification. The way we did this was to use the most common label in the prediction. So for example if a prediction was "1/1/0", we would classify it as a 1 since that is the most common prediction. This was achieved in code by writing:

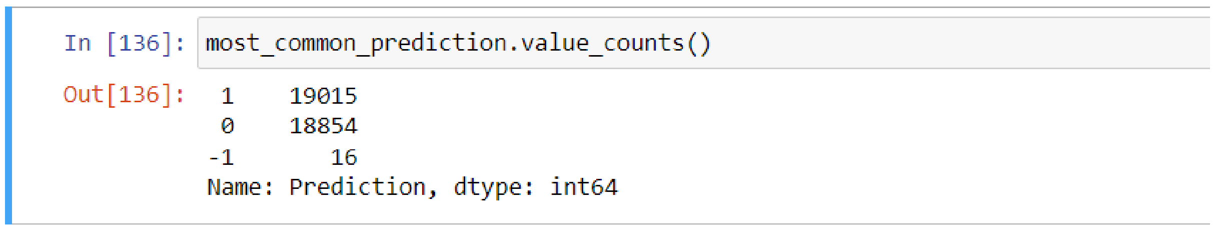
```
from statistics import mode

most_common_prediction = training_data_final["Prediction"].apply(lambda
x: int(mode(x.split("/"))))
```

We apply a lambda function to each row in the Prediction column. The x parameter in this case is the label in that column, such as "1/1/0". The function works by first splitting the label into a list of its individual classifications so "1/1/0" would become ["1","1","0"]. It then uses the mode function from the statistics module to find the most frequent element in the list. Lastly it calls the int() function on the most frequent element in the list to turn it from a string to an integer. We store the column that was created from applying the lambda function to the Prediction column in a new series variable called most_common_prediction.

However, we are not yet done. We need to check if there are any rows where -1 was the majority label and thus became the final classification for that row. We can check the value counts of all rows using:

```
most_common_prediction.value_counts()
```

The result showed us that 16 rows had -1 as their final classification label.



```
In [136]: most_common_prediction.value_counts()

Out[136]:  1    19015
           0    18854
          -1       16
          Name: Prediction, dtype: int64
```

We want to delete these rows as we only want to train our classifier to make a prediction of whether a message is pro-vaccine or anti-vaccine. In order to remove these rows, we need to know the index of the rows where the majority label was -1. We can get those indices using:

```
negative_one_prediction_indices =
list(most_common_prediction[most_common_prediction==-1].index)
```

We can then simply remove these rows from the original dataframe:

```
training_data_final.drop(negative_one_prediction_indices, inplace=True)
```

We also call the same lambda function that we used to get most_common_prediction, but this time we store the result in the same dataframe(inplace).

```
training_data_final["Prediction"] =
training_data_final["Prediction"].apply(lambda x:
int(mode(x.split("/"))))
```

Now we have a single classification for each row and we have deleted all the rows in the dataframe that had a majority classification of -1. We can now move on to the training of our algorithm.

Step 3 - Transforming messages to Vectors

A machine learning algorithm can not understand text messages without first representing the data(in this case the messages) in some numerical form. We can represent text messages as numerical features using 2 classes: sklearn.feature_extraction.text.CountVectorizer and sklearn.feature_extraction.text.TfidfVectorizer.

The CountVectorizer works as such that it first learns the vocabulary of all of the messages in the training dataset, that is, all of the unique words in all of the messages in the training dataset. This is done when you call its **.fit()** function of the CountVectorizer. It then creates a 2d matrix of bag-of-words encoding for all of the messages where each column in each row represents the word-count of a specific word in that specific message. This is done when you call the **.transform()** function of the CountVectorizer. The number of columns in the 2d matrix will be the number of unique messages in the training data and the number of rows in the 2d-matrix will be the total number of messages, where each row represents corresponding message in the training dataset.

The TfidfVectorizer works similarly to CountVectorizer except that it multiples the term frequency(which CountVectorizer uses) by the inverse document frequency. The result is that words that occur in many documents get a smaller weight, since they are considered common words that are uninformative for the prediction task.
We create and train the 2 vectorizers using:

```
cv = CountVectorizer()
```

```
tv = TfidfVectorizer()
word_counts_final = cv.fit_transform(training_data_final["Message"])
word_weights_final = tv.fit_transform(training_data_final["Message"])
```

We now have a numerical representation of our features which we can feed to whichever machine learning algorithm that we decide to use.
We also separate the labels from the dataset as:

```
training_labels_final = training_data_final["Prediction"]
```

Step 4 - Training the Machine learning algorithm

There are many different machine learning algorithms that could be used for text classification. However, one of the most common ones is Naive Bayes Classifier, since it is designed particularly with text classification in mind. It can be used for classifying whether a message is spam or ham, classify whether a review is positive or negative or in our case, classify whether a message is pro-vaccination or anti-vaccination.

We will use Multinomial Naive Bayes classifier. We create 2 MNB models, one to classify the text data that was encoded with CountVectorizer() and one to classify the text data encoded with TfidfVectorizer().

```
mnb_cv_final = MultinomialNB()
mnb_tv_final = MultinomialNB()
```

We then train both of the models using the .fit() method and pass it the 2 vectors. The mnb_cv_final model will receive the CountVectorizer() matrix and the mnb_tv_final will receive the TfidfVectorizer matrix.

```
mnb_cv_final.fit(word_counts_final, training_labels_final)
mnb_tv_final.fit(word_weights_final, training_labels_final)
```

After we have trained both of the models, it is time to test the accuracy of the model. We do this using the testing set, since we want to see how well the model performs on unknown data that it has not been trained on. We load the testing data the same way we loaded the training data.

```
test_set = pd.read_csv("a3_test.tsv", sep="\t", header=None, names =
["Prediction", "Message"])
```

However, the testing data, just like the training data, stores the messages in text format. We need to transform the messages into a numerical representation. Otherwise the machine learning algorithm will not be able to work with the test data.

```
word_counts_test_final = cv.transform(test_set["Message"])
word_weights_test_final = tv.transform(test_set["Message"])
```

It is important that we only call the transform function of the CountVectorizer and TfidfVectorizer and not fit_transform(). The reason is that we do not want the Vectorizers to learn a new vocabulary from the test set. Instead we only want the Vectorizers to transform the test set into a matrix based on the existing vocabulary it learned from the training set. Otherwise, this will lead to an error when we try to use the word_counts_test_final and word_weights_test_final with the MNB models that have been trained on a different vocabulary with different dimensions.

After we have transformed the test set messages into a numerical representation, we can finally test our data and see the accuracy score.

3. Results

The accuracy of the Multinomial Bayes classifier was different depending on if you trained it with a CountVectorizer() matrix or if you trained it with a TfIdfVectorizer() matrix.
For the MNB model trained with a CountVectorizer() matrix, we got the accuracy using:

```
mnb_cv_final.score(word_counts_test_final, test_set["Prediction"])
```

The accuracy was 0.833. This means that it got 83% of the predictions on the test set correct.

For the MNB model trained with a TfidfVectorizer() matrix, we got the accuracy using:

```
mnb_tv_final.score(word_weights_test_final, test_set["Prediction"])
```

The accuracy was **0.853.** This means that it got 85% of the predictions on the test set correct.

Stop-words in CountVectorizer()

We also tried to feed the MNB model a CountVectorizer() matrix where we used stop-words.

```
cv = CountVectorizer(stop_words="english")
```

Stop-words are common words that will not be considered/counted in the resulting matrix. This could be words such as “and”, “this”, “or”. We thought that this would improve the model since these words are uninformative in making a prediction. Similar to how TfidfVectorizer() improved the accuracy, we thought that using stop-words would improve the accuracy. However, it was not true. The accuracy after using stop words had decreased to **0.826.**

We then tried using a different set of stop-words instead of the built-in stop-words you get when you set **stop_words=”english”**. Such a list was found in the nltk module of python.

```
import nltk
from nltk.corpus import stopwords
```

We fed this to the CountVectorizer.

```
cv = CountVectorizer(stop_words=stopwords.words("english"))
```

We fitted and transformed the training data using the CountVectorizer that uses stop words and trained the MNB model with the resulting encoded matrix. The accuracy this time was exactly the same as the accuracy of not using any stop words. The accuracy we got was **0.833.**

Other parameters in CountVectorizer

Stopwords is not the only hyperparameter for the CountVectorizer. There are many parameters which can be used to make it better. One of the other parameters we tried was **ngram_range**. This parameter determines if the CountVectorizer should only count single words(unigrams) or consecutive pairs of words(bigrams) or any other n-gram of words. The default behavior is to use unigrams, where each word is counted individually. If set to ngram_range = (2,2) it will count the frequency occurrence of all consecutive pairs of words(bigrams). We will set ngram_range = (1,2). This will count the frequency of all words individually and also the frequency of all consecutive pairs of words.

```
cv = CountVectorizer(ngram_range=(1, 2))
```

The accuracy after using ngram_range=(1,2) increased drastically from **0.833** to **0.875**.

Different Models - Perceptron and LinearSVC

The multinomial naive bayes is one of the best models to use for text classification. However, there are many other machine learning algorithms that could be used to classify a message as pro covid-19 vaccination or anti covid-19 vaccination. Some alternatives are Perceptron, LinearSVC and neural network. We created and tested a perceptron model as:

```
from sklearn.linear_model import Perceptron
perceptron = Perceptron()
perceptron.fit(word_counts_final, training_labels_final)
perceptron.score(word_counts_test_final, test_set["Prediction"])
```

The accuracy we got was **0.853** when using CountVectorizer with **ngram_range = (1,2).** When feeding it a matrix from a CountVectorizer with a default ngram_range(unigrams only), then the accuracy score dropped to **0.807.**

We also tried LinearSVC. The accuracy score we got when training it with a matrix from CountVectorizer with ngram_range=(1,2) was **0.858**. When feeding it a matrix with only unigram count, the accuracy dropped to **0.832**. The accuracy of the LinearSVC was oddly close to the accuracy of the Naive Bayes classifier.

4. Discussion

Trivial Baseline - 0.5

A good way to measure how well your machine learning model performs is to compare it to a trivial baseline. For the baseline classifier, we used the dummy classifier in sci-kit learn.

```
from sklearn.dummy import DummyClassifier
```

The default behavior of the dummy classifier is to classify any unknown data as the most common class that it was trained on. So for example, if the dummy classifier was trained on 50 pro covid-19

vaccination messages and 70 anti-covid 19 vaccination messages, it will always predict any unknown message as anti covid-19 vaccination.
The accuracy of our dummy classifier landed at 0.5. This is because 50% of the messages in the test set are pro covid-19 vaccination and 50% of the messages in the test set are anti covid-19 vaccination. The dummy classifier always predicts the same label, and therefore the accuracy landed at 50%.

```
dc = DummyClassifier()
dc.fit(training_data["Message"], training_labels)
dc.score(test_set["Message"], test_set["Prediction"])
```

Any classifier with an accuracy score below or close to that of the dummy classifier(0.5) should be considered very poor.

Important Features

Important features are words which have a high predicative value in a sentence. Such a word could either predict that a message is pro covid-19 vaccination or anti covid-19 vaccination when found in a sentence. A way to identify these words is to see which words occur most frequently in pro covid-19 vaccination messages and which words occur most frequently in anti covid-19 vaccination messages. This can be done using the code:

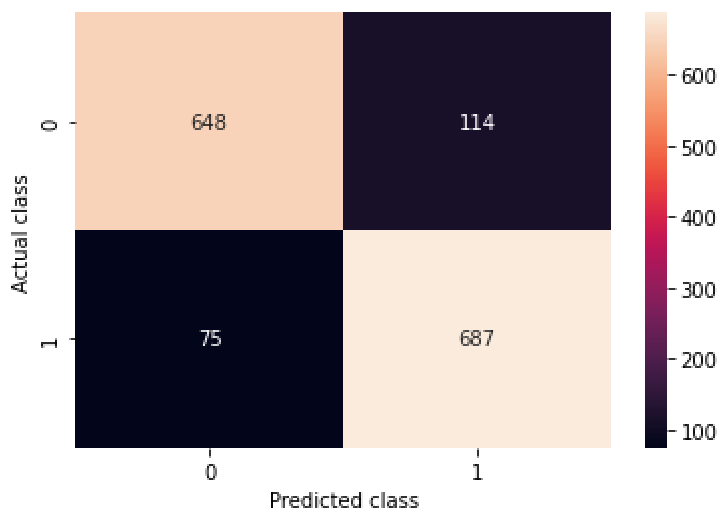
```
print(sorted(list(zip(cv.get_feature_names(),
mnb_cv.feature_log_prob_[0])), key = lambda x : x[1], reverse=True))
```

The mnb_cv.feature_log_prob[0] gives you the log probability of finding that word given that the message is anti covid 19 vaccination. The higher the value, the more likely you are to find this word in an anti covid-19 vaccination message. However, this is not enough. **A word that is common to find in an anti covid-19 vaccination message could also be common in a pro covid-19 vaccination message and is therefore alone not a good indicator if a message is pro or anti covid-19 vaccination.** A word which would be common to find in anti covid-19 vaccination message but very uncommon to find in pro vaccination message would however be a good indicator for feature importance of that word. However, there are few such words when using unigrams, since a lot of the words rely on context. Words such as “die” could have positive or negative meaning depending on the context. It could be someone saying: “No one I know did die from covid-19 vaccine” or “I know 5 people that did die from the vaccine”. Therefore, feature importance for this task is a complicated issue.

Errors

Even the best model(the MNB with TfidfVectorizer and ngram_range=(1,2)) was not perfect. To better visualize and understand the errors, we created a confusion matrix that we plotted as a heatmap

```
cm = confusion_matrix(test_set["Prediction"],
mnb_tv_final.predict(word_weights_test_final))
sns.heatmap(cm, annot= True, fmt="d")
plt.ylabel("Actual class")
plt.xlabel("Predicted class")
```



The heat map shows the true negative rates(0 is the actual class of the message and 0 was predicted) and the true positives rates(1 is the actual class of the message and 1 was predicted) as well as the false positive and false negative rates. We wanted to investigate some of the false positive and false negative messages to see what they looked like.

False Positive messages	**False Negatives messages**
About 4 months ago, I had COVID-19. I just got my first vaccine and I became very ill with fever, chills, body aches and dizzy spells. Now I'm afraid to get the second dose. Any suggestions?	And a vaccine literally introduces the virus to your immune system in a non harmful way... Good food doesn't produce antibodies, a vaccine does.
that vaccines are preventing millions of deaths is the real misinformation	vaccines can resist the harm caused by the new coronavirus
never vaccinate your children	the vaccine is developed by a trusted company

From investigating these messages, we see no clear pattern or explanation why the algorithm classified them as false positive and false negative. We think it has to do with the fact that Naive Bayes is Naive and does not understand the context of the words. Understanding the context of the words is just as important as counting the words that occur, which is not the strong side of Naive Bayes.

5. Conclusion

The best model we found for classifying text messages as pro covid-19 vaccination or anti covid-19 vaccination was the Multinomial Naive Bayes classifier. In order to maximize the accuracy of the MNB, we gave it a matrix from a TfidfVectorizer with the additional hyperparameter ngram_range set to (1,2). We landed on a final, best accuracy of **0.875**.