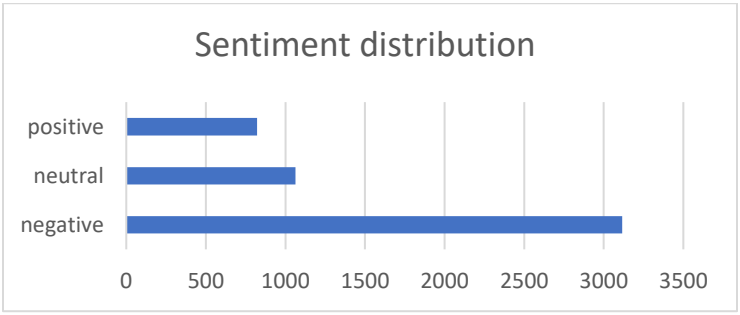
9414 Artificial Intelligence, 20T2

Assignment 2 Report,
Vishal Bondwal, 5278101

1. Give simple descriptive statistics showing the frequency distribution for the sentiment classes for the whole dataset of 5000 tweets. What do you notice about the distribution?

Sentiment classes are categorical variables, with the values 'positive', 'negative', and 'neutral'. Out of the total 5000 tweets in the given dataset, the frequency distribution of these 3 classes is:

Class	Value Count	Percentage
positive	822	16.4%
neutral	1063	21.3%
negative	3115	62.3%
Total	5000	100.0%



We notice that sentiments in this set of tweets are mostly negative, with neutral sentiments a distant second. Positive sentiments are only approximately one-sixth of the total number of tweets. Thus there is a strong negative bias in this dataset. However we can't comment on how it reflects the real world scenario because this sample may or may not be reflective of the population.

2. Develop BNB and MNB models from the training set using (a) the whole vocabulary, and (b) the most frequent 1000 words from the vocabulary (as defined using CountVectorizer, after preprocessing by removing "junk" characters). Show all metrics on the test set comparing the two approaches for each method. Explain any similarities and differences in results.

(Note: lowercase=False was specified in CountVectorizer( ) arguments in all models, as in forum posts @278 and @338, the professor has asked not to convert to lower case in the standard models.)

The metrics for both models in both conditions are given below. Accuracy cell is marked in bold.

(On my installation, classification_report() writes "micro avg" as "accuracy", and gives only one value, while on the CSE machine, it's written as "micro avg", and gives the same value, repeated across the three cells (since here we have only one test set and precision and recall are the same, so their harmonic mean, i.e. f1 score is also the same). I am using both terms here, the values remain the same)

BNB Model	**With Full Vocabulary**			**With 1000 word vocabulary**			
	precision	recall	f1-score	precision	recall	f1-score	support
negative	0.68	0.99	0.8	0.87	0.83	0.85	628
neutral	0.77	0.21	0.33	0.6	0.67	0.63	210
positive	0.91	0.12	0.22	0.61	0.65	0.63	162
accuracy (micro avg)	**0.69**	**0.69**	**0.69**	**0.76**	**0.76**	**0.76**	1000
macro avg	0.79	0.44	0.45	0.69	0.71	0.7	1000
weighted avg	0.73	0.69	0.61	0.77	0.76	0.77	1000

MNB Model	**With Full Vocabulary**			**With 1000 word vocabulary**			
	precision	recall	f1-score	precision	recall	f1-score	support
negative	0.72	0.99	0.84	0.84	0.89	0.86	628
neutral	0.79	0.26	0.39	0.63	0.54	0.58	210
positive	0.83	0.39	0.53	0.67	0.62	0.65	162
accuracy (micro avg)	**0.74**	**0.74**	**0.74**	**0.77**	**0.77**	**0.77**	1000
macro avg	0.78	0.54	0.58	0.71	0.69	0.7	1000
weighted avg	0.76	0.74	0.69	0.77	0.77	0.77	1000

We notice that for both models, on limiting the feature set to 1000 words, there is an increase in accuracy and F1 scores (indicating a healthier balance between precision and recall). In both models, earlier there is very high recall (.99) on the negative class, indicating that the classifiers are classing many examples as negative, but they are not accurate (indicated by the much lower precision). Reducing the feature set improves precision and reduces spurious recall, improving the f1 score.

The total number of features in this dataset, after junk character/URL removal and taking min word length as two characters, is approximately 7448 words. Since we have only 4000 training examples in our train set, we actually have much fewer examples than features. This causes noisy data and overfitting. Reducing the number of words in the vocabulary reduces noise and overfitting to the training set, thus allowing better adaptation to the test set.

3. Evaluate the three standard models with respect to the VADER baseline. Show all metrics on the test set and comment on the performance of the baseline and of the models relative to the baseline.

(Note: As per professor's direction in @208, junk characters and URLs are not removed for VADER.)

(Note: as noted in the previous question, accuracy and micro avg refer to the same thing, and the same value is repeated in all cells of the row. So using the term 'accuracy' from now, and also not repeating the values to increase clarity. This is the output given by classification_report() by default on my system.)

The metrics for VADER on the test set are as follows (the given thresholds of >=.05 positive and <=-.05 negative, neutral in between, have been used.)

VADER	**precision**	**recall**	**f1-score**	**support**
negative	0.91	0.48	0.63	628
neutral	0.36	0.43	0.39	210
positive	0.34	0.89	0.49	162
accuracy (micro avg)			**0.54**	1000
macro avg	0.54	0.6	0.51	1000
weighted avg	0.7	0.54	0.56	1000

The metrics for the standard models are as follows:

	BNB model			**MNB model**			**DT**		
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score
negative	0.68	0.99	0.8	0.72	0.99	0.84	0.73	0.9	0.81
neutral	0.77	0.21	0.33	0.79	0.26	0.39	0.46	0.25	0.33
positive	0.91	0.12	0.22	0.83	0.39	0.53	0.68	0.48	0.56
accuracy (micro avg)			**0.69**			**0.74**			**0.7**
macro avg	0.79	0.44	0.45	0.78	0.54	0.58	0.62	0.54	0.56
weighted avg	0.73	0.69	0.61	0.76	0.74	0.69	0.67	0.7	0.67

We can see that VADER baseline has poorer accuracy than all the three standard models. It is also biased towards giving positive classifications, with poor precision and high recall on the positive class (It classified 420 test cases as positive, when only 162 are actually positive). The BNB model, the MNB model, and the decision tree, all have much better F1 scores on the negative class than VADER, because they are trained on this set and reflect its negative bias more accurately. On the other hand, there's not as much difference in F1 score for the positive class, and VADER's score is actually better than BNB.

VADER is trained on a crowdsourced lexicon, which has words from a variety of contexts, not just airlines as the tweets here. So models trained on this flight dataset itself perform more accurately here. On investigating the VADER lexicon, we observe that it uses emojis, but they are text emojis, like :-O or :), which haven't been used much in our tweets. VADER also has junk characters in its dataset, which would not contribute to meaningful information.

Additional note on VADER: Since the tweets are mostly negative, we can change the comparison threshold of VADER to match this dataset more accurately. The current threshold interval of [-.05, +.05] for the *score['compound']* parameter is giving results biased towards positivity. Through trial and error, it was determined that shifting this threshold significantly to the right, to [0.19, 0.57] gives the prediction counts (617, 215, 168) for negative, neutral and positive respectively, which are very close to the actual class values of (628, 210, 162). However accuracy and precision are still poor.

4. Evaluate the effect of preprocessing the input features by applying NLTK English stop word removal then NLTK Porter stemming on classifier performance for the three standard models. Show all metrics with and without preprocessing on the test set and explain the results.

Results after removing standard English NLTK stopwords, and then Porter stemming, compared to original results. The support column is not shown because it is the same as in all previous metrics.

BNB model	**Original**			**With stopwords and stemming**		
	precision	recall	f1-score	precision	recall	f1-score
negative	0.68	0.99	0.8	0.69	0.98	0.81
neutral	0.77	0.21	0.33	0.73	0.23	0.35
positive	0.91	0.12	0.22	0.88	0.23	0.36
accuracy (micro avg)			**0.69**			**0.7**
macro avg	0.79	0.44	0.45	0.77	0.48	0.51
weighted avg	0.73	0.69	0.61	0.73	0.7	0.64

MNB model	Original			With stopwords and stemming		
	precision	recall	f1-score	precision	recall	f1-score
negative	0.72	0.99	0.84	0.75	0.97	0.85
neutral	0.79	0.26	0.39	0.77	0.32	0.46
positive	0.83	0.39	0.53	0.78	0.49	0.61
accuracy (micro avg)			**0.74**			**0.76**
macro avg	0.78	0.54	0.58	0.77	0.6	0.64
weighted avg	0.76	0.74	0.69	0.76	0.76	0.73

DT	Original			With stopwords and stemming		
	precision	recall	f1-score	precision	recall	f1-score
negative	0.73	0.9	0.81	0.77	0.85	0.81
neutral	0.46	0.25	0.33	0.43	0.38	0.41
positive	0.68	0.48	0.56	0.7	0.56	0.62
accuracy (micro avg)			**0.7**			**0.7**
macro avg	0.62	0.54	0.56	0.64	0.59	0.61
weighted avg	0.67	0.7	0.67	0.69	0.7	0.69

We see a 1% improvement in accuracy in both the Bayesian models, while the decision tree accuracy remains unchanged. However even in the decision tree, and in the Bayesian models, there are small improvement in F1 scores, showing improved balance between precision and recall. The macro and micro averages also improve. As explained in question 2, we have over 7000 features in the original training set, and have only 4000 training examples, which causes poor signal to noise ratio. The models tend to overfit to the training set noise, and perform badly on predicting an unseen test set. Porter stemming reduces the number of features and removes many redundant values. Stopwords generally do not cause much information gain, and unnecessarily load the model with features. Thus removing stopwords and combining similar words into a single stem gives us better performance. Note that since the Decision Tree standard model is already limiting the number of features to 1000, so it did not see as much improvement as the other two models on reducing features with stopwords and stemming.

5. Evaluate the effect that converting all letters to lower case has on classifier performance for the three standard models. Show all metrics with and without conversion to lower case on the test set and explain the results.

CountVectorizer has a parameter called lowercase. For the standard models and all other questions, it was set to lowercase = False, so that the input text was not converted to lowercase. For this question, this parameter is being set to lowercase = True. The metrics are given below:

BNB model	Original			Converted to lowercase		
	precision	recall	f1-score	precision	recall	f1-score
negative	0.68	0.99	0.8	0.71	0.99	0.83
neutral	0.77	0.21	0.33	0.83	0.32	0.47
positive	0.91	0.12	0.22	0.96	0.27	0.42
accuracy (micro avg)			**0.69**			**0.73**
macro avg	0.79	0.44	0.45	0.83	0.53	0.57
weighted avg	0.73	0.69	0.61	0.78	0.73	0.68

MNB model	Original			Converted to lowercase		
	precision	recall	f1-score	precision	recall	f1-score
negative	0.72	0.99	0.84	0.75	0.98	0.85
neutral	0.79	0.26	0.39	0.81	0.33	0.47
positive	0.83	0.39	0.53	0.83	0.46	0.6
accuracy (micro avg)			**0.74**			**0.76**
macro avg	0.78	0.54	0.58	0.8	0.59	0.64
weighted avg	0.76	0.74	0.69	0.78	0.76	0.73

DT	Original			Converted to lowercase		
	precision	recall	f1-score	precision	recall	f1-score
negative	0.73	0.9	0.81	0.75	0.89	0.81
neutral	0.46	0.25	0.33	0.49	0.28	0.35
positive	0.68	0.48	0.56	0.67	0.57	0.61
accuracy (micro avg)			**0.7**			**0.71**
macro avg	0.62	0.54	0.56	0.64	0.58	0.59
weighted avg	0.67	0.7	0.67	0.68	0.71	0.68

This improves the accuracy across all models, and also improves the F1 scores (indicating healthier balance between precision and recall), and also improves both the macro and micro averages.

Converting to lowercase smooths out the data by unifying two features that were earlier separated. For example, in the sentences “Late flight.” and “Flight late,” the first word is being capitalised, so “Flight” and “flight” would be counted as two different features, when in fact they refer to the same thing. Similar case for “late” and “Late”. So, converting all values to lowercase generally improves accuracy (in rare cases, there may be exceptions caused by proper nouns. For example, in the sentences “He went on a cruise.” and “Tom Cruise won an award,”, the words “cruise” and “Cruise” refer to different real-world

features. Making them both lowercase could reduce accuracy in such a dataset.) As we saw earlier, given that we have excessive number of features for a small training set, we need to use feature reduction approaches like this to avoid overfitting and get a more general model that can apply to a variety of test sets.

6. Describe your best method for sentiment analysis and justify your decision. Give some experimental results for your method trained on the training set of 4000 tweets and tested on the test set of 1000 tweets. Provide a brief comparison of your model to the standard models and the baseline (use the results from the previous questions).

Introduction

The custom method used for sentiment analysis for this assignment uses the ensemble approach, and simulates taking the mode of five different classifiers. Taking mode across well-studied standard models tends to improve both reliability and accuracy. Especially, since we have a small training set, no single model is likely to provide the best results. Any peculiarities of one model get smoothed out when five models are taken. The models used are logistic regression, decision tree, linear support vector machine, Bernoulli naïve Bayes, and multinomial naïve Bayes classifiers. All of them are well-established algorithms for classification, including text classification, and perform reasonably well in the current application (especially given the small dataset), resulting in 80% overall accuracy. A mode is not always possible when we have more than 2 classes, since multiple classes may occur the highest number of times. The resolution of this issue is discussed in detail in the implementation section.

The classification report of this algorithm on the test set of 1000 tweets is:

	precision	**recall**	**f1-score**	**support**
negative	0.84	0.9	0.87	628
neutral	0.65	0.61	0.63	210
positive	0.78	0.64	0.71	162
accuracy (micro avg)	0.8	0.8	**0.8**	1000
macro avg	0.76	0.72	0.74	1000
weighted avg	0.79	0.8	0.79	1000

This is a general challenge with all algorithms that neutral sentiments are harder to classify, as reflected in its low F1 score. The word features with the most information gain tend to be polarised towards positive or negative sentiment.

Comparisons

Classification reports and brief discussions comparing this model to the baseline and standard models follow:

	VADER			Custom model			
	precision	**recall**	**f1-score**	precision	recall	f1-score	support
negative	0.91	0.48	0.63	0.84	0.9	0.87	628
neutral	0.36	0.43	0.39	0.65	0.61	0.63	210
positive	0.34	0.89	0.49	0.78	0.64	0.71	162
accuracy			**0.54**			**0.8**	1000
macro avg	0.54	0.6	0.51	0.76	0.72	0.74	1000
weighted avg	0.7	0.54	0.56	0.79	0.8	0.79	1000

The algorithm is much more accurate than the VADER baseline, at 80% vs 54% accuracy. Though as we saw in Question 3, VADER is not trained on this data and does not capture its negative tone well (note the high recall and low precision of VADER on positive classifications, showing its positive bias). However, the few tweets that it does classify as negative are quite accurately classified (91% precision).

Compared to the naïve Bayesian models:

	BNB Standard Model			Custom model			
	precision	recall	f1-score	precision	recall	f1-score	support
negative	0.68	0.99	0.8	0.84	0.9	0.87	628
neutral	0.77	0.21	0.33	0.65	0.61	0.63	210
positive	0.91	0.12	0.22	0.78	0.64	0.71	162
accuracy			**0.69**			**0.8**	1000
macro avg	0.79	0.44	0.45	0.76	0.72	0.74	1000
weighted avg	0.73	0.69	0.61	0.79	0.8	0.79	1000

	MNB Standard Model			Custom model			
	precision	recall	f1-score	precision	recall	f1-score	support
negative	0.72	0.99	0.84	0.84	0.9	0.87	628
neutral	0.79	0.26	0.39	0.65	0.61	0.63	210
positive	0.83	0.39	0.53	0.78	0.64	0.71	162
accuracy			**0.74**			**0.8**	1000
macro avg	0.78	0.54	0.58	0.76	0.72	0.74	1000
weighted avg	0.76	0.74	0.69	0.79	0.8	0.79	1000

Note the very high recall and low precision on the negative class for both the Bayesian algorithms. Even though our custom classifier uses the negative bias to break ties (explained in the implementation section), it is still more balanced in handling negative classifications (reflected in the higher f1 scores on

the negative class). While it has lower precision on the positive class, it has much better recall as well, again reflected in higher f1 scores.

Compared to the decision tree standard model:

	DT Standard Model			**Custom model**			
	precision	recall	f1-score	precision	recall	f1-score	support
negative	0.73	0.9	0.81	0.84	0.9	0.87	628
neutral	0.46	0.25	0.33	0.65	0.61	0.63	210
positive	0.68	0.48	0.56	0.78	0.64	0.71	162
accuracy			**0.7**			**0.8**	1000
macro avg	0.62	0.54	0.56	0.76	0.72	0.74	1000
weighted avg	0.67	0.7	0.67	0.79	0.8	0.79	1000

The custom model has a 10% accuracy gain on the decision tree. It also performs better on every single other metric, which is to be expected, as decision trees are one of the inputs to the ensemble. In the ensemble as well, we have used the same approaches to reduce decision tree fragmentation as in the standard model (limiting end features to 1% of the training set).

Design Choices and Implementation Notes

The general rationale for choosing an ensemble model has already been discussed in the introduction. Some further choices are explained below.

Choosing features: As mentioned in the earlier answers, we have over 7000 features in the 4000 tweet training set. Such a high ratio of features to examples tends to make models overfit to the training set noise, and perform badly on predicting the test set.

The custom model removes standard NLTK English stopwords, uses Porter stemming, and converts all values to lowercase to reduce features. Lemmatizing was also tried, but did not lead to much benefit. It is discussed briefly separately.

The final step in reducing features was to choose the 'right' number of maximum features for *CountVectorizer()* . After much trial and error, 500 was found to be the optimal amount, giving the maximum accuracy of 80%. This makes sense given that we have only 4000 training examples. Most machine learning models expect the number of examples to be several multiples of the number of features. However, being an ensemble model, the gains are slight, as choosing different values between 300 and 2000 still led to an accuracy in the range of 78-79%. But it touches 80% only at 500, appearing to be optimal for this training set.

Lemmatizing: WordNet Lemmatizing was tried as an approach, because for meaningful text, it is generally more effective than stemming, and preserves more meaning rather than only look at alphabetical structure of words. It is more complex to implement, since it requires part-of-speech tagging as well (e.g. better and good are both Lemmatized to 'good', only if 'better' is POS-tagged as an adjective; otherwise it remains as 'better'). To enhance Lemmatizing performance, we used NLTK's in-

built *word_tokenize()* tokenizer, instead of our regular expression based one. Not all POS tags are handled by the WordNet lemmatizer, so they need to be mapped to the right values.

Despite all this, the gains were negligible. A comparison follows:

	Lemmatized version			**Custom model**			
	precision	recall	f1-score	precision	recall	f1-score	support
negative	0.85	0.9	0.88	0.84	0.9	0.87	628
neutral	0.65	0.6	0.63	0.65	0.61	0.63	210
positive	0.78	0.66	0.72	0.78	0.64	0.71	162
accuracy			**0.8**			**0.8**	1000
macro avg	0.76	0.72	0.74	0.76	0.72	0.74	1000
weighted avg	0.8	0.8	0.8	0.79	0.8	0.79	1000

Accuracy remains unchanged at 80%, though there are tiny improvements in precision & recall in some classes. However the program became much slower since WordNet is a large corpus to load, and the pos-tagging and lemmatizing took further time. So we did not choose this for the final version. Lemmatizing possibly was not useful, due to informal, ungrammatical language in the examples, and short tweet lengths.

Mode simulation: In the introduction, we mentioned 'simulating the mode'. We are 'simulating' taking of the mode because a statistical mode is not always possible with more than 2 classes. For example, if we had only two classes: positive & negative, then using any odd number of classifiers would be guaranteed to result in a mode. For example, if we had three classifiers, and they gave the output [positive, negative, positive], then the mode is 'positive'. This would be true with any odd number of classifiers. However, since we have three classes, it is not as simple anymore, since the output could be [positive, neutral, negative]. Since each class occurs once, there is no mode. Increasing the number of classifiers does not remove the problem. With four classifiers, we may get [positive, positive, negative, negative], and with five we could get [positive, positive, negative, negative, neutral]. In both cases there are two classes with the highest number of occurrences.

We need a method to break such ties. One way is arbitrary tiebreaking, and another is to use more information from the training set to inform the tiebreak. The latter approach is taken here. The class that incorporates these multiple classifiers, *statClassifier*, has a method that determines the most common class label in the training set, which we call the 'bias' of the training set. Since the testing set would come from the same context as the training set, it is very likely it would reflect the same bias. For example, in this airline tweets example, most of the tweets are negative. Our classification algorithm first tries to find a proper mode. If there is a tie, then it checks if either of the top two classes match the bias class. If it does, then the bias class is returned as the mode. Otherwise the tie is broken arbitrarily. There is a 1% improvement in accuracy, from 79% to 80%, by using this bias approach (as opposed to always breaking ties arbitrarily and returning the value at the first index).