HW 3

Q1:

This dataset is from https://www.kaggle.com/snap/amazon-fine-food-reviews and contains 568,454 entries for fine food reviews from Amazon. This contains more than 10 years of review data and includes information like the product id, user id, review score, and review text.

Table statistics	
Number of rows (documents)	568,454
Number of columns	10
Number of labels	5 distinct labels for score column
Label distribution	1 52268 2 29769 3 42640 4 80655 5 363122 Name: Score, dtype: int64
Average word length of documents	82
Max word length of documents	3526
Min word length of documents	3
Median word length of documents	58

Q2:

File for preprocessing is attached as preprocessing.py which tokenizes, removes stop words, removes digits and special characters, stems, lemmatizes and converts the score labels to categorical variables.

The experiments run on text data will be trying to predict 5 distinct labels for the score column. For the variations on the logistic model, the following results were observed.

Experiments were run on unigrams, then with bigrams included as well.

Hyperparameter tuning was also conducted on both sets of data were the solver was changed to newton-cg to account for the multiclass predictions and to see the effects of another solver type. The penalty was also increased to 0.5 to see what the impact of strengthening regularization.

Type of model	Data used	Hyperparamters changed	Accuracy	Weighted average precision	Weighted average recall	Weighted average F1
Base model	Unigrams only	NA	0.7352	0.69	0.74	0.69
Model with bigrams	Unigrams and bigrams	NA	0.7866	0.77	0.79	0.76
Model with new hyperparameters	Unigrams only	Solver = ‘newton-cg’ C = 0.5	0.7293	0.69	0.73	0.68
Bigram model with new hyperparameters	Unigrams and bigrams	Solver = ‘newton-cg’ C = 0.5	0.7619	0.75	0.76	0.72

Q3:

The next type of model that will be examined will be the SVM models. For these models, different models are trained on unigrams only and data with both the unigrams and bigrams. The hyperparameters experimented with here are the loss function and the regularization strength.

For SVM, it was interesting to see whether a different score metric would be able to separate the classes out better. Furthermore, we similarly increase the regularization strength to 0.5 as well to mirror our experiment for logistic regression.

Type of model	Data used	Hyperparamters changed	Accuracy	Weighted average precision	Weighted average recall	Weighted average F1
Base model	Unigrams only	NA	0.7463	0.71	0.75	0.71

Model with bigrams	Unigrams and bigrams	NA	0.8283	0.82	0.83	0.82
Model with new hyperparameters	Unigrams only	Loss = ‘squared_hinge’ C=0.5	0.7433	0.71	0.74	0.7
Bigram model with new hyperparameters	Unigrams and bigrams	Loss = ‘squared_hinge’ C=0.5	0.8264	0.83	0.77	0.82

Q4:

In this section, the fasttext package was used to conduct similar experiments. The base use case was defined as learning rate of 0.1 and epoch of 10.

Similarly, the learning rate for other experiments was increased, this time to 1. The number of epochs was also increased to see the impact of increasing the number of runs on the resulting accuracies.

Type of model	Data used	Hyperparamters changed	Accuracy

Base model	Unigrams only	lr=0.1
		
		epoch=10
			0.7717

Model with bigrams	Unigrams and bigrams	lr=0.1
		
		epoch=10
			0.8262

Model with new hyperparameters	Unigrams only	lr=1
		
		epoch =25
			0.7770

Bigram model with new hyperparameters	Unigrams and bigrams	lr=1
		
		epoch =25
			0.8228

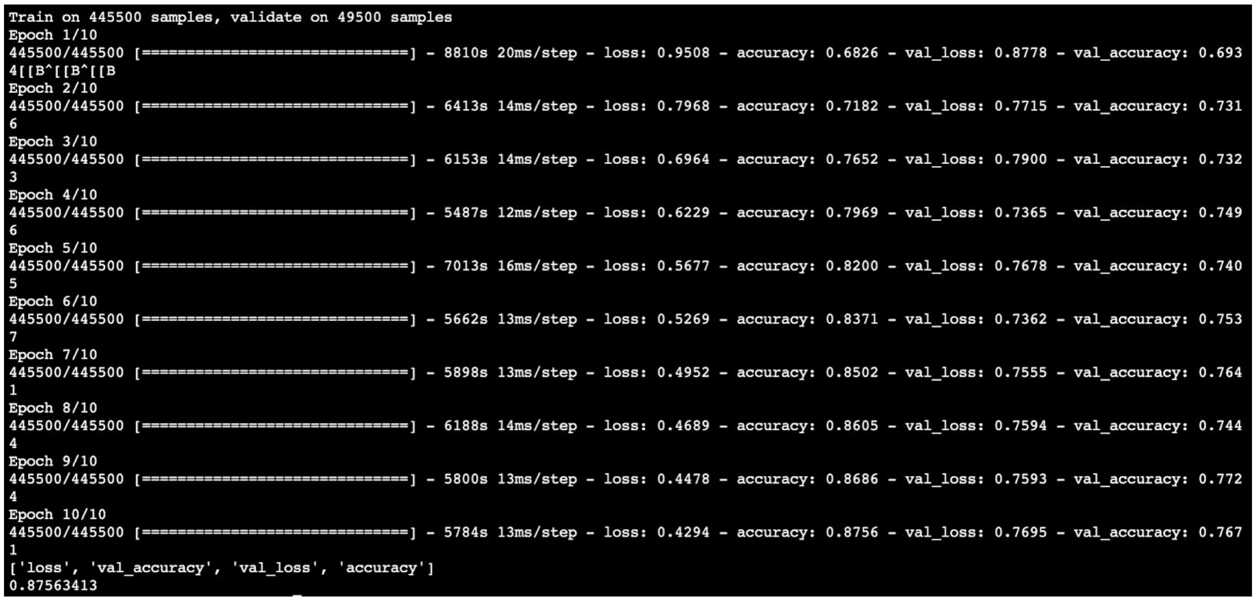
Q5:

The CNN model was trained on the same texts. For this version, the first 500,000 rows of the data set were used to create the dictionary with gensim. Similarly, two versions were trained on the base hyperparameters and with both unigrams and bigrams.

Similar to the above versions, the learning rate was an interesting parameter to change as the default value of 0.0003 seemed a little low and would have taken quite some time to converge on the best solution. The penalty was increased to 0.01 to see the impact of a high learning rate.

Initial training of the models, especially the bigram model, was very slow with the high penalty approach. To alleviate this, I attempted to increase the batch size from 100-200 to help speed up training a little. Two more models were trained on unigrams and bigrams.

Type of model	Data used	Hyperparamters changed	Accuracy
Base model	Unigrams only		0.79
Model with bigrams	Unigrams and bigrams		0.86
Model with new hyperparameters	Unigrams only	0.01 learning rate	0.79
Bigram model with new hyperparameters	Unigrams and bigrams	0.01 learning rate	0.87
Model with new hyperparameters	Unigrams only	0.01 learning rate 200 batch size	0.77
Bigram model with new hyperparameters	Unigrams and bigrams	0.01 learning rate 200 batch size	0.84

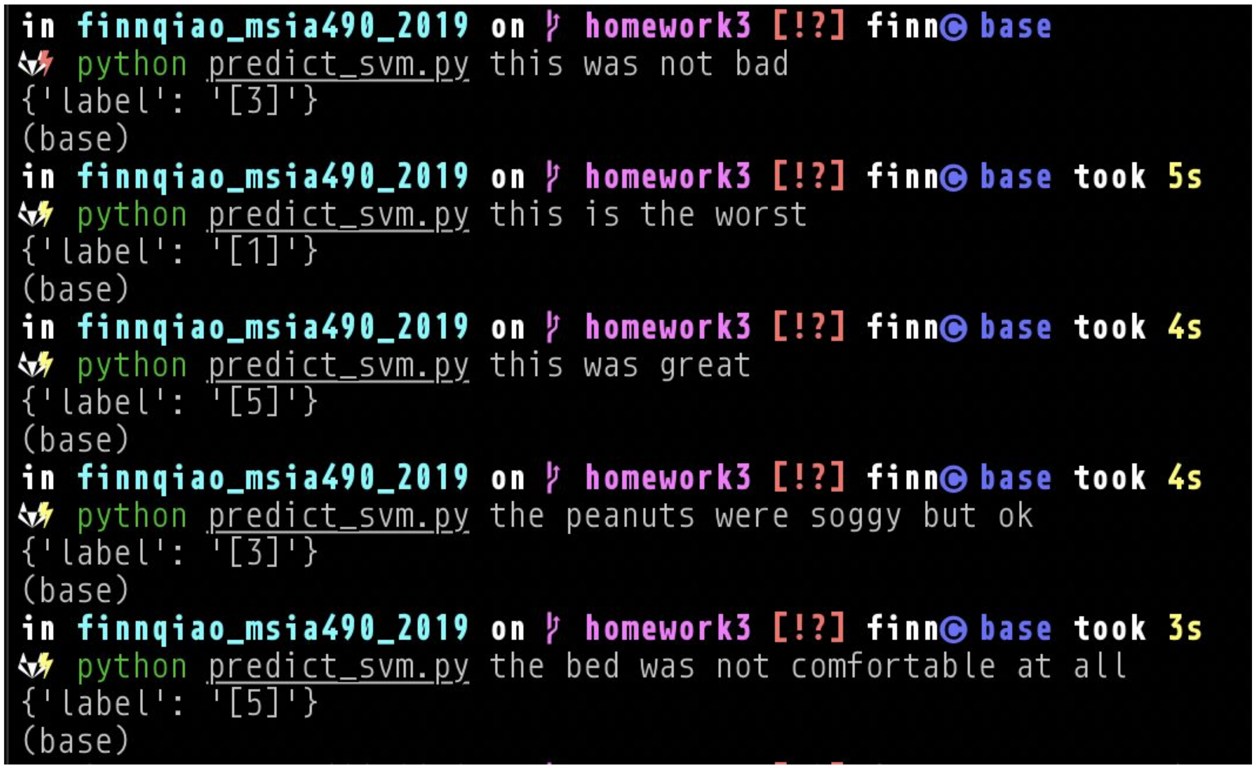


```
Train on 445500 samples, validate on 49500 samples
Epoch 1/10
445500/445500 [==============================] - 8810s 20ms/step - loss: 0.9508 - accuracy: 0.6826 - val_loss: 0.8778 - val_accuracy: 0.693
4[[B^[[B^[[B
Epoch 2/10
445500/445500 [==============================] - 6413s 14ms/step - loss: 0.7968 - accuracy: 0.7182 - val_loss: 0.7715 - val_accuracy: 0.731
6
Epoch 3/10
445500/445500 [==============================] - 6153s 14ms/step - loss: 0.6964 - accuracy: 0.7652 - val_loss: 0.7900 - val_accuracy: 0.732
3
Epoch 4/10
445500/445500 [==============================] - 5487s 12ms/step - loss: 0.6229 - accuracy: 0.7969 - val_loss: 0.7365 - val_accuracy: 0.749
6
Epoch 5/10
445500/445500 [==============================] - 7013s 16ms/step - loss: 0.5677 - accuracy: 0.8200 - val_loss: 0.7678 - val_accuracy: 0.740
5
Epoch 6/10
445500/445500 [==============================] - 5662s 13ms/step - loss: 0.5269 - accuracy: 0.8371 - val_loss: 0.7362 - val_accuracy: 0.753
7
Epoch 7/10
445500/445500 [==============================] - 5898s 13ms/step - loss: 0.4952 - accuracy: 0.8502 - val_loss: 0.7555 - val_accuracy: 0.764
1
Epoch 8/10
445500/445500 [==============================] - 6188s 14ms/step - loss: 0.4689 - accuracy: 0.8605 - val_loss: 0.7594 - val_accuracy: 0.744
4
Epoch 9/10
445500/445500 [==============================] - 5800s 13ms/step - loss: 0.4478 - accuracy: 0.8686 - val_loss: 0.7593 - val_accuracy: 0.772
4
Epoch 10/10
445500/445500 [==============================] - 5784s 13ms/step - loss: 0.4294 - accuracy: 0.8756 - val_loss: 0.7695 - val_accuracy: 0.767
1
['loss', 'val_accuracy', 'val_loss', 'accuracy']
0.87563413
```

The best results resulted from the bigram model trained with 0.01 penalty and batch size 100. The accuracy across classes seemed to be 0.8756.

Q6:

View attached python files.



```
in finnqiao_msia490_2019 on homework3 [!?] finn base
python predict_svm.py this was not bad
{'label': '[3]'}
(base)
in finnqiao_msia490_2019 on homework3 [!?] finn base took 5s
python predict_svm.py this is the worst
{'label': '[1]'}
(base)
in finnqiao_msia490_2019 on homework3 [!?] finn base took 4s
python predict_svm.py this was great
{'label': '[5]'}
(base)
in finnqiao_msia490_2019 on homework3 [!?] finn base took 4s
python predict_svm.py the peanuts were soggy but ok
{'label': '[3]'}
(base)
in finnqiao_msia490_2019 on homework3 [!?] finn base took 3s
python predict_svm.py the bed was not comfortable at all
{'label': '[5]'}
(base)
```