Q1)

a)

The dataset chosen for this project is a collection of character lines from the television series South Park. This dataset struck my interests since I used to watch the show and was curious to see if a machine learning model can be used to identify which line belongs to which character. In practice, this model could be used to create a tool which predicts what South Park character is most likely to say a user provided line.

b)

The labels selected are the four main characters of the show (Kenny, Cartman, Kyle and Stan), with the input text being the lines belonging to each character. In order to fit these labels, the original dataset was filtered to only include lines from the chosen characters. However, if all lines were considered, the number of entries would exceed the limit of 10,000, so each character's lines were sampled for 2000 (or less if there are less than 2000 total) values. This results in a dataset of 6881 lines from the four main characters after pre-processing.

c)

The dataset was not split into training, validation and test sets, so I had to do the 60/20/20% split myself.

	Train Count	Validation Count	Test Count
Total	4128	1377	1376
Cartman	1186	398	416
Kenny	534	177	170
Kyle	1221	403	381
Stan	1187	398	410

Q2)

a)

Examples of documents:

Cluster 0:

- do lahty tricky and
- hi said he
- Whoops

Cluster 1:

- happened just what dude
- what
- going are it with what dude you to do

Cluster 2:

- same aren homos non differently treated need homosexuals whites from different are blacks that idea the support crimes hate groups into splitting stop time it mayor is people be all we to do
- adventure christmas in back ll time for we get

- safe keep easier way figure carton egg cradle makin an oh ll our can that it for just be re we so of out get to

Cluster 3:

- god oh my
- oh
- cool not oh this is dude

Cluster 4:

- everybody luck good answers give cheat us help on primarily judged be will performances your classtime during friend as work all you how see re we now so okay gonna and
- hagen aaron know dude you do
- dick you re

Top 5 tokens for each cluster:

Cluster 0: [is, it, the, no, that]

Cluster 1: [that, hell, doing, are, what]

Cluster 2: [it, the, re, to, we]

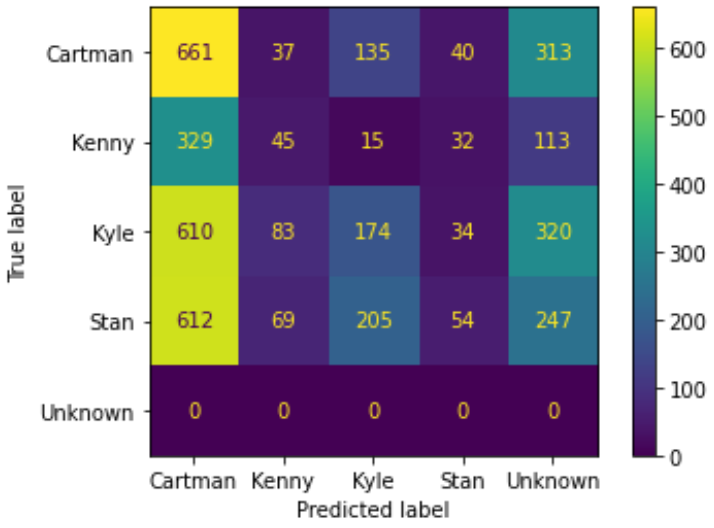
Cluster 3: [uh, no, my, god, oh]

Cluster 4: [the, guys, to, you, yeah]

b)

Based on a) these clusters do not make a lot of sense. This is most likely because the tokenizer used does not omit stop words, so the most frequent tokens end up being these really common words, which do not give much description about each cluster. There does seem to be some difference between clusters, with some of them addressing more specific topics (longer sentences) while others contain common phrases (“oh my god”, etc.), however clear topics cannot be identified.

c)



d)

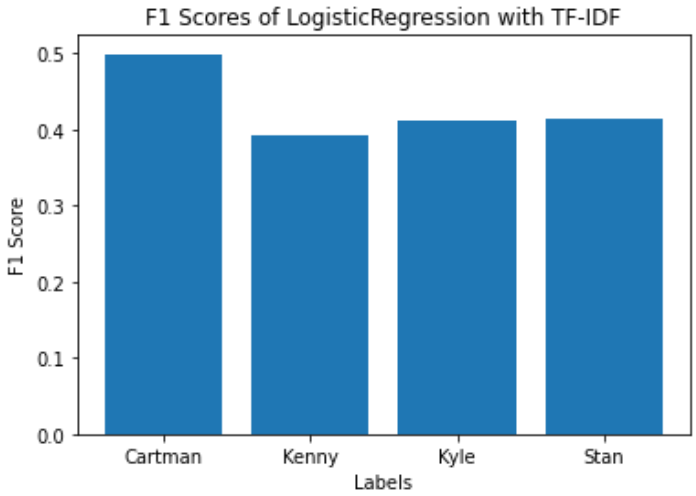
At first glance, there seem to be a lot of matches for Cartman (661), however upon further examination this only happens by chance, as Cartman got the highest number of predictions from k_means, which is reinforced by the high number of false positives with Kenny and Kyle. Overall, there do not appear to be any trends, most likely because five clusters were approximated while only four labels exist (thus the "Unkown" values). Additionally, each cluster is arbitrarily labelled, as the k-means algorithm does not consider the original labels while training, it only considers the raw vector values. This results in a clustering that is most likely not related to the original labelling.

Q3)

a)

Evaluation metrics (best performer in red):

	Accuracy	Precision	Recall	F1
Dummy (most_frequent)	0.289	0.072	0.250	0.112
Dummy (stratified)	0.278	0.263	0.263	0.263
LogRegression (One-Hot)	0.435	0.443	0.421	0.428
LogRegression (TF-IDF)	0.437	0.461	0.418	0.429
SVC	0.323	0.611	0.288	0.220



Analysis and Discussion of classifiers

Performance relative to baselines is quite poor across the board, with the highest score being 0.611. This is most likely because of the nature of the data, however it is difficult to pinpoint the exact reasons. The uneven distribution of labels (Kenny having a lot less samples than the others) could be one of these reasons, as the classifier may not have enough samples to be able to predict all the labels correctly. Another more likely reason is that labels represent characters who are close friends with each other in the show, meaning that they probably have similar personalities (thus speak similarly in terms of vocabulary and sentence structure), rendering simple classifiers not very effective. Additionally, the lines for each character were all written by the same person, making them even harder to classify.

The next section discusses the results from each classifier in detail, as well as how models performed relative to each other.

Dummy (most_frequent)

With simply selecting the validation data's most frequent match in the training data, we cannot expect great performance. This is supported by the F1 score of 0.112, which is the lowest out of all five models, thus we can comfortably conclude that this was the worst performer. The reason this F1 score was so low was because of the classifier's extremely low precision (0.072), meaning that there are a lot of false positives in the predictions made.

Dummy (stratified)

The stratified dummy classifier performs slightly better than the most frequent one (F1 score = 0.263). This is because it has higher precision at 0.263 (less false positives), due to using probabilities of one-hot encoded vectors as opposed to most frequent. Coincidentally, the recall is also 0.263, which is why the F1 score is the same as the precision and recall at 0.263. This means that the classifier strikes a good balance between exactness and completeness, however both scores are quite low unfortunately.

Logistic Regression (One-hot)

As to be expected, logistic regression performs much better than the dummy classifiers (F1=0.428). This statement applies to every metric measured (accuracy=0.435, precision=0.443, recall=0.421) and most likely occurs because logistic regression does not ignore the input features, whereas dummy classifiers do.

Logistic Regression (TF-IDF)

This was the best performing classifier with F1 score 0.429. It performs better than the previous classifier (with one-hot vectorisation) because it has significantly higher precision (0.461), even though it has slightly lower recall (0.418). TF-IDF vectorisation improves the model most likely due to the fact that characters can be identified more effectively based on phrases that they often repeat, which one-hot does not track.

The label specific F1 scores (seen in bar chart above) show that the performance is the best for Cartman and worst for Kenny. This makes sense, since Cartman is the most iconic character, so he should be the most easily identifiable. The reason why Kenny's score is the lowest is most likely because his dataset has less samples than all the others.

SVC

SVC made predictions with very high precision (0.611). However, its accuracy (0.323) and recall (0.288) are very low, causing its f1 score (0.220) to be worse than even the stratified dummy classifier. This is likely because we are running the kernel with default settings, which usually does not result in high performance. Furthermore, SVC can often struggle with multiclass problems, which is the case here.

b)

For vectorisation, TF-IDF was used, as it performed better than one-hot in the baseline tests, and we also want to track the frequency of repeated phrases. However, some parameters were changed to hopefully result in better performance on our dataset. Firstly, a Spacy tokenizer was used instead of

the default tokenizer, which let us remove stop words and spaces, while keeping brackets. We want to keep brackets because Kenny might be able to be identified more easily this way.

For classification, a decision tree classifier was used. This classifier was chosen mainly because it can handle diverse features, which appear frequently in our corpus (many made up words and phrases). The main disadvantage of decision trees is stability, however this does not affect us because our data is static.

Results

	Accuracy	Precision	Recall	F1
Decision tree	0.497	0.570	0.563	0.563

The performance of our approach is the best we have seen so far, with an F1 score of 0.563. Furthermore, it also has the highest accuracy, with 0.497. This increase in performance comes from either the new vectoriser or classifier. To decide which had a larger effect, we tested the new vectoriser with a logistic regression classifier, which also resulted in a better F1 score. Therefore, we can conclude the that new vectoriser works better than the default.

Q4)

1.

The regularisation value C was tuned in the recommended range (from 10^-3 to 10^5) in increments of 10^1.

2.

Sublinear_tf (True and False) and max_features were also tuned in the recommended range (from None to 50k in increments of 10k).

3.

The third parameter chosen is max_df for the vectoriser. This is because while we wish to keep the commonly repeated phrases, some extremely common tokens might need removing, as they are used a lot by every character. The range for this parameter is from 0.7 to 1.0 (increments of 0.1).

Even though it was not recommended, we still did a full sweep on all the parameters, as we wanted to find the best combination. At the end the optimal results were: C = 1, sublinear_tf = False, max_features = None, and max_df = 0.7. The performance on the validation set with these parameters is the following:

Accuracy	Precision	Recall	F1
0.437	0.461	0.418	0.429

Curiously, these are the exact same results as the logistic regression with default settings. When expecting closer however, this makes sense, because most of the optimal values (C, sublinear_tf and max_features) turned out to be the defaults. This means that any changes to these values only decreased perfromance. For example, in the case of max_features, the default vocabulary most likely consists of more than 50k relevant terms, therefore eliminating some of these reduced performance.

Since max_df is not its default value, but still does not affect any of the metrics, we can assume that it has no effect on the model (likely because there are no instances of tokens which get ignored because of it). Perhaps it would have been more useful to try even smaller values of max_df. Alternatively, the max_iter parameter of the classifier could have been tested, as the algorithm often failed to converge within the default number of iterations.

Q5)

a)

Accuracy	Precision	Recall	F1
0.546	0.614	0.602	0.608

b)

c)

The following sets of hyperparameters were chosen (in order to determine what effect changing individual parameters makes, all but one were kept the same):

0.

Original inputs: model = 'roberta-base', learning rate = 1e-4, epochs = 1, batch_size = 16

1.

Learning_rate = 0.1. This value is much higher than the original, so its purpose is to see if it can make the algorithm converge more quickly.

2.

Epochs = 3. This is higher than the original, so its purpose is to find out if giving the algorithm more time will provide better results.

3.

Model = "distilbert-base-uncased". This is done to see if changing the model will yield better results.

Results

	Accuracy	Precision	Recall	F1
0.	0.477	0.586	0.543	0.476
1.	0.289	0.072	0.250	0.112
2.	0.477	0.586	0.543	0.476
3.	0.558	0.647	0.613	0.592

d)

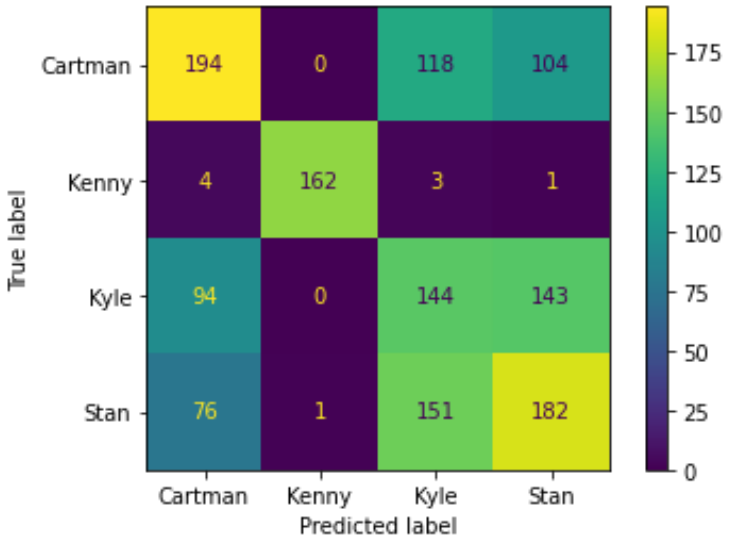
The best performer is the pipeline approach from a) with an F1 score of 0.608. The most probable reason it outperformed the end-to-end approach is because the hyperparameters used in b) and c) were suboptimal. Since end-to-end classifiers have significantly more parameters than the preset pipeline, optimisation requires a lot more resources, with the added benefit of improving performance (only) if the correct settings are selected. Therefore, I am confident that with more tuning, the end-to-end approach could perform better.

In terms of how changing certain hyperparameters changed performance in c), 1. made it worse. This is because the step size set was too high to converge to the best possible model. 2. had no effect, as the original inputs already converged, thus adding more epochs did not alter performance. 3. improved performance, indicating that the “distilbert-base-uncased" might be more suited for our dataset.

Q6)

a)

Accuracy	Precision	Recall	F1
0.495	0.573	0.560	0.566



b)

The most striking result is the performance of the “Kenny” label. For this label, the classifier is able predict almost all instances (162) in the test set, with very few false positives (1) and negatives (8). This likely happens due to the formatting of Kenny's lines, since they are mostly placed between brackets, so they can be easily identified.

The “Cartman” label also performed quite well, with 194 correct predictions, however much more false positives (170) and even more false negatives (222). The reason many correct predictions were made is because Cartman has many iconic catch phrases, which the classifier could easily pick up on. However, many lines are very generic, thus the high number of errors with Kyle and Stan.

The “Stan” label follows a similar pattern, with 182 correct predictions, 227 false positives and 247 false negatives. Most of the errors happened between Kyle and Stan, which makes sense, since they are more like supporting characters in the show, meaning that their lines are less iconic (making them more difficult for a classifier to predict).

The worst performing label is “Kyle”, as it has more false positive predictions just with “Stan” (151) than true positives (144). This goes back to the point made before about Kyle and Stan fulfilling similar roles in the show, thus the similar lines / way of speaking.

Overall, the classifier is excellent at predicting for Kenny, decent for Cartman and Stan, and not very good for Kyle.

c)

The final performance is sufficiency high for its purpose (Which character would say it?), since it is not involved in a safety critical system, it is simply a fun game, where “wrong” predictions cause no harm. False positives just return a different character to the “actual” answer, and same goes for false negatives, however these lines are mostly arbitrary, so users would not even notice, let alone care. For example, if one of Kyle's lines is input, there is a decent chance that the classifier will return Stan instead, making this a false negative, but a user would likely not realise this error, since the characters are so similar.

d)

I do not really see, any negative societal effects, since the model is not outputting any new texts, just predicting labels. Some people perhaps could be slightly annoyed if what they said was input and they got a character which they were unhappy with.

e)

Observing the original documents, many of them are very generic, which frankly any of the four characters could say. These could be omitted entirely, which might improve performance. Furthermore, I would tweak the hyperparameters of the logistic regression classifier used in the pipeline, since this performed the best with default settings, so perhaps it could be improved even further.

f)

~35 hours