Machine Learning Final Assignment

Student Name: Patrick Farmer Student Number: 20331828

Date: 30-11-2024

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Part 1

Introduction

This Assignment aims to generate nursery rhyme rhythms given a simplified input dataset. The dataset has characters that represent different notes. Each note will be played for 0.5 seconds, if the same character appears a couple of times in a row it will be played for longer. The project is built upon the GPT used in lab9 with some addition and changes made to improve the model's performance for this task. When the initial model was used the output already sounded perfectly feasible to my ear, for that reason during this lab measurable metrics will be used to evaluate the model performance. Due to the prediction being quite simple a number of different approaches were taken and the results across each will be compared.

The first of which approaches was to simply adjust the hyperparameters to fit the dataset better. The second approach was to first train the model on the pitch only and remove rhythm, then after that fine tune the model on the full dataset which included rhythm. The third and final approach that was used and produced acceptable results was to train a model on the pitch and rhythm separately and then combine the two models to generate the final output.

Data Preprocessing and feature engineering

There was a couple forms of data augmentation applied to the dataset overall for every method. The first of which was a simple noise generator which would for two in every 10 characters replace the character with a random character from the dataset. The second form of data augmentation was to stretch and shrink the patterns in the dataset. This works by first finding the common patterns in the dataset and when one occurs it will be stretched or shrunk by a random amount within some threshold. This was done to emphasise the patterns (rhythms) in the dataset.

For some of the methods additional pre-requisite steps were required. For the method to train the model on pitch only first the dataset was preprocessed to remove all timing data. This was done by simply collapsing all sequential occurances of the same character into one. The model was then first trained on this dataset and was trained secondly on the full dataset.

For the method to train the model on pitch and rhythm separately the dataset was split into two datasets, the first which contained only pitch was processed as mentioned above and the dataset was used again here. The second dataset that contained only rhythm was preprocesses in the same script. It would replace the characters with numbers representing the length of the note. Models were then trained on both of these datasets. The separate models were very accurate for their tasks as the task was simpler than the combined task. They were then combined together to generate the final output by rebuilding the data in the same way it was split. This is effectively a multiplication of the datasets. i.e.:

```
timing = "1113111
pitch = "CDEFGAB"
output = "CDEFFFGAB"
```

Machine Learning methodology

The first change that was made was a change to the loss function. A combination of two loss functions was used. A weighted sum of the cosine embedding loss and the cross entropy loss. The cosine embedding loss minimises the difference between the sequence generated and the target sequence which captures the rhythm as a whole. The cross entropy loss is more specifically for the timing to capture the beat fo the rhythm. The weighting is weighted 70% cosine loss and 30% cross entropy loss.

Temperature was also added to increase the creativity of the model. Temperature adjusts the smoothness of the probability distribution. A higher temperature will make the model more creative but less accurate. It is implemented by dividing the logits by the temperature before applying the softmax function. To improve the accuracy the temperature was reduced to 0.6 but if the the task for the model required more diversity a higher temperature could be used.

The Hyperparameters were adjusted to fit the dataset better. The model as a whole was made much more simple as the task is a lot repetitive and structured than the previous task the gpt.py was used for. Three different sets of hyperparameters were used to compare against each other.

The **first set of hyperparameters** is a very light weight set of parameters, the model uses only 490 parameters. The hyperparameters are as follows:

```
batch_size = 64          # The batch size was kept large as it keeps the convergence stable
block_size = 4           # The block size was reduced to 4 as the patterns are very localised.
# This would not capture the larger patterns and this will be compared with the other models
max_iters = 20           # The max iterations was reduced to 20 as the model converged very quickly
eval_interval = 2        # The evaluation interval was reduced to 2 as the model converged quickly
learning_rate = 1e-4     # The learning rate was kept low as the model is simple
eval_iters = 1           # The evaluation iterations was reduced to 1 as the model converged quickly
n_embd = 8               # The number of embeddings was kept very low as there was not overly
# complex patterns in the dataset
n_head = 1               # The number of heads was kept low as there was not overly complex
# relationships between tokens in the dataset
n_layer = 1              # The number of layers was kept small as the dataset was simple and could
# easily be overfit
dropout = 0.1            # The dropout was set low as there was a large amount of data and the model
# was simple so it is not likely to overfit
```

The **second set of hyperparameters** was slightly more complex and allowed for the same heavy localisation focus but allowed more complex patterns and relationships to be captured. The model still had a reasonably small parameter count at 20,000. The model was also run for a few more iterations. The hyperparameters are as follows:

```
batch_size = 64          # The batch size was kept large as it keeps the convergence stable
block_size = 4           # The block size was kept at 4 to mirror the local focus in the first model
max_iters = 100          # The max iterations was increased to 100 as the model was more complex
eval_interval = 10       # The evaluation interval was increased to 10 as the model was more complex
learning_rate = 1e-4     # The learning rate was kept low as the model is still relatively simple
eval_iters = 10          # The evaluation iterations was increased to 10 as the model was more complex
n_embd = 32              # The number of embeddings was increased to 32 to capture more complex patterns
n_head = 4               # The number of heads was increased to 4 to capture more complex relationships
# between tokens
n_layer = 1              # The number of layers was kept small as the dataset was simple and could
# easily be overfit
dropout = 0.1            # The dropout was set low as there was a large amount of data and the model was
# simple so it is not likely to overfit
```

The **third set of hyperparameters** was the most complex, it tested how the model would perform if it was allowed to capture bigger patterns in addition to more complex patterns. This model has a parameter count of 40,000. The hyperparameters are as follows:

```
batch_size = 64          # The batch size was kept large as it keeps the convergence stable
block_size = 32          # The block size was increased to 32 to capture larger patterns
max_iters = 400          # The max iterations was increased to 400 as the model was more complex
eval_interval = 40       # The evaluation interval was increased to 40 as the model was more complex
learning_rate = 1e-4     # The learning rate was kept low as the model is still relatively simple
eval_iters = 40          # The evaluation iterations was increased to 40 as the model was more complex
n_embd = 32              # The number of embeddings was increased to 32 to capture more complex patterns
n_head = 4               # The number of heads was increased to 4 to capture more complex relationships
# between tokens
n_layer = 2              # The number of layers was increased to 2 to capture more complex patterns
dropout = 0.1            # The dropout was set low as there was a large amount of data and the model was
# simple so it is not likely to overfit
```

In order to extract the local rhythm patterns more effectively 1D convolutional layers were added before the transformer blocks. This works because the convolutional layers can extract the local patterns as they would in an image classifier and the transformer blocks can then more easily capture the relationships between the local patterns.

Evaluation

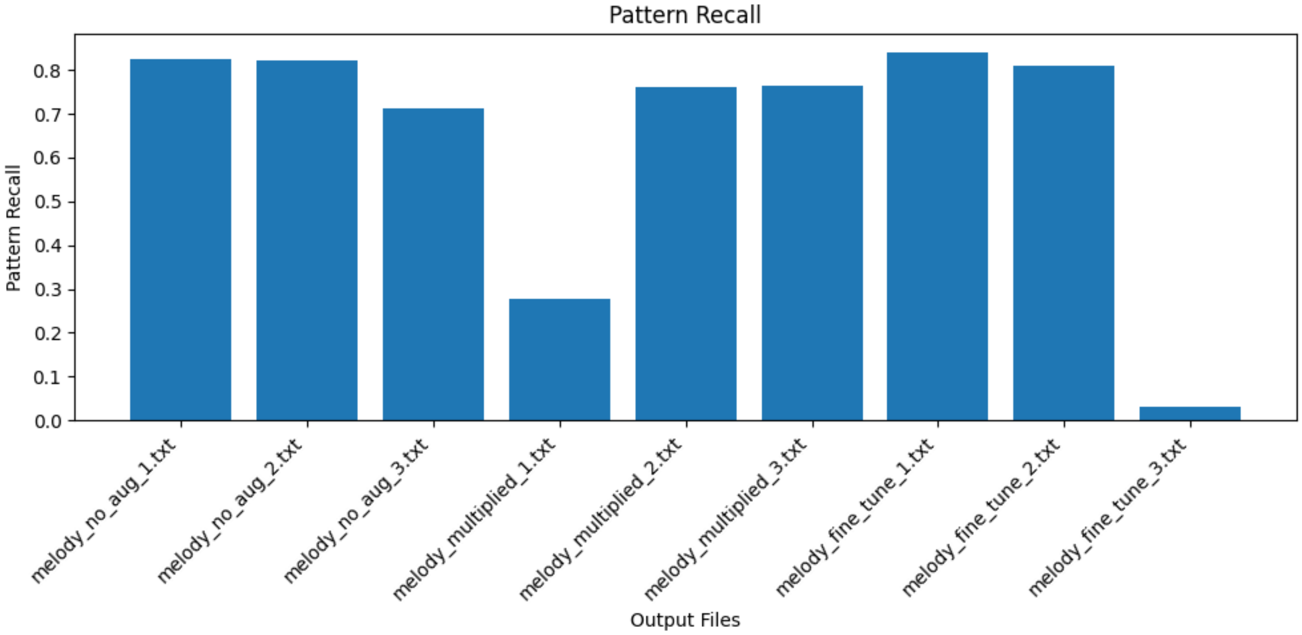
It was mentioned in the introduction that it would not be possible to make a subjective evaluation of the model as all of the outputs sounded feasible to the ear. For that reason the models were compared with a number of metrics. The metrics that were used were a few standard machine learning metrics F1-score, precision and recall. Also a few more less conventional metrics, character distribution distance, pattern length distribution distance and pattern distribution distance.

The output was broken into patterns of length 2-4. Every pattern in both the target and the output was logged into a dictionary. Each of the patterns in the output was checked against the model patterns and if it was there it was incremented. This gives us our true positives. The true positives and false negatives are then just the length of the model pattern array. Using these two values the pattern recall can be calculated. The same process was followed for precision where the true positives and false positives are the length of the output pattern array. The F1-score is then calculated using the precision and recall.

The character distribution metric was calculated by taking the distance between the character distribution of the target and the output. This was a sanity check metric to make sure that the model wasn't just asking as a dummy classifier. The pattern length distribution metric was calculated by taking the distance between the distribution of the length of sequential notes in the target and the output. This was a metric to make sure that the model was capturing the rhythm. The pattern distribution metric was calculated by taking the distance between the distribution of the patterns in the target and the output. This was a metric to make sure that the model was capturing the patterns in the dataset.

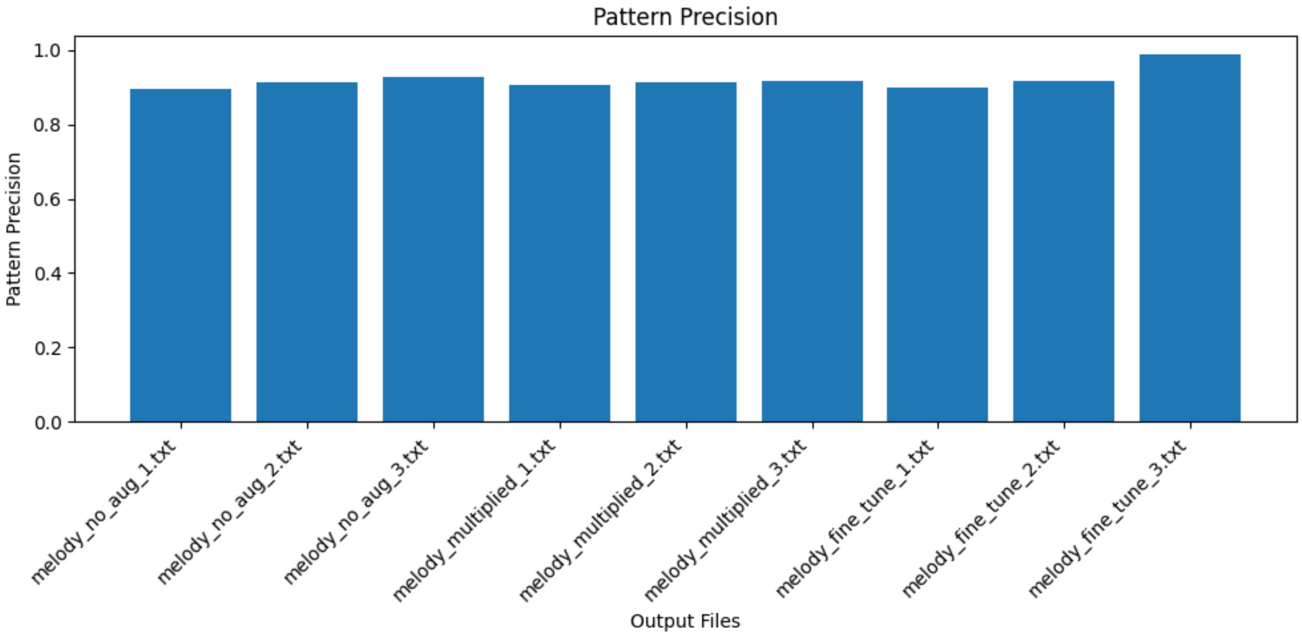
A baseline model was also created to compare the models against. The baseline model was a simple model that would predict the most common character with the most common sequence length, which happened to be 'f'.

Taking first the recall comparison. We get the below chart:(higher is better)



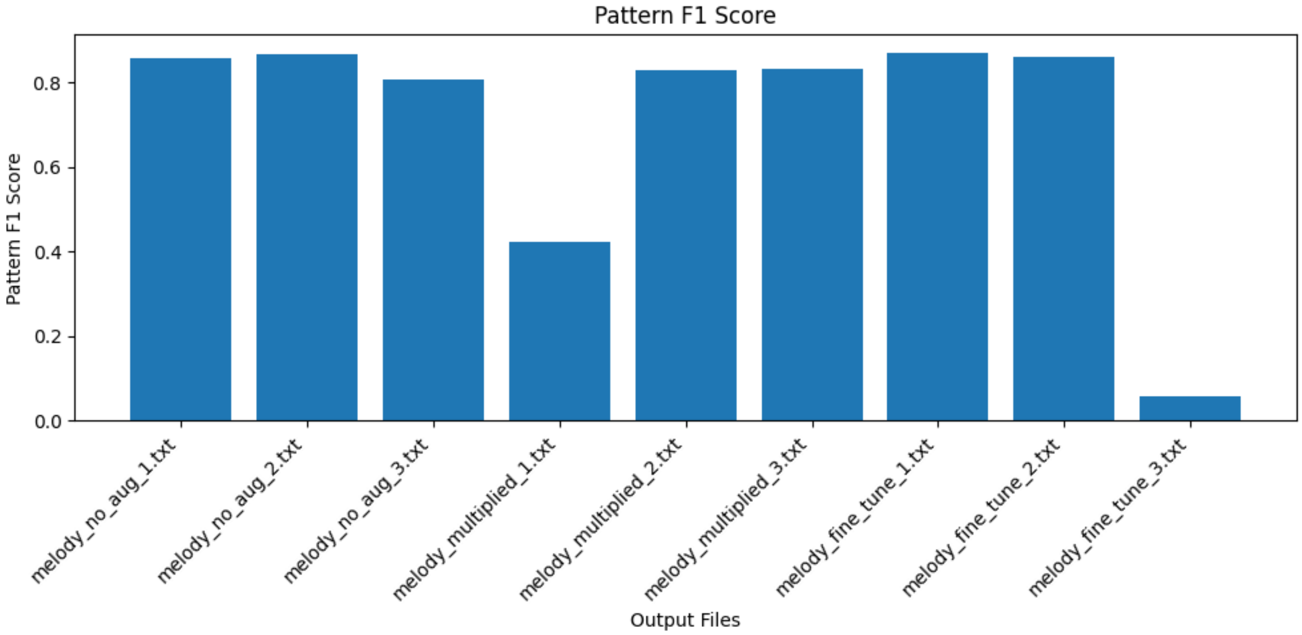
Most of the models are close to each other but the best 2 results are both using the first set of hyperparameters with the better of the two being the model that was trained on pitch only first. This is because the model was able to capture the patterns in the dataset better. We do see however that some of the models fell apart to give a very low recall. In the case of the fine-tune parameter 3 that is because the model is just predicted the 1 character for the entire output very similarly to the baseline model. We also see that the simple model 1 for the separate model approach performed very poorly. This is because the model was able to capture the pitch with a simple model but not catch the timing, the result of this is it predicted very long notes which was very off rhythm.

For the precision comparison we get the below chart:(higher is better)



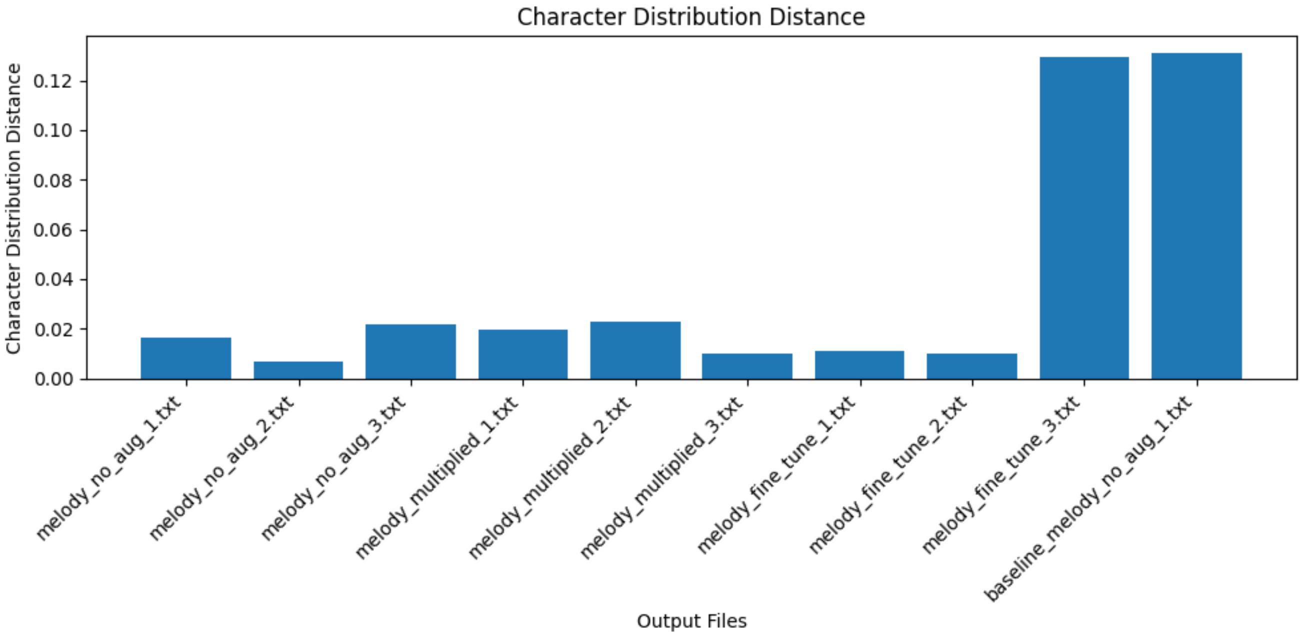
The precision tells use very little as almost all of the models have a precision between 0.9 and 1. That is because the model has a low temperature and is unlikely to produce false positives and would be more likely to produce false negatives. We only see here the fine-tune model 3 has a precision of 1 due to the model predicting just the one character.

For the F1-score comparison we get the below chart:(higher is better)



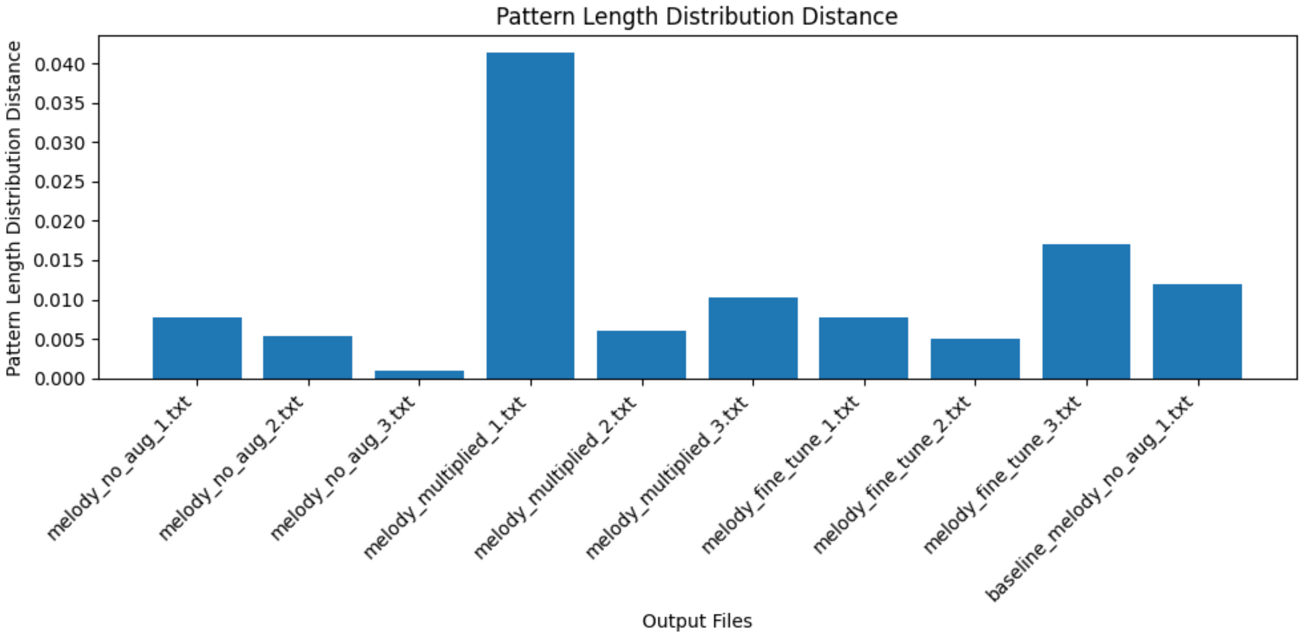
The F1-score is a combination of the precision and recall. We just saw above that the precision is almost entirely the same which is why our F1-score graph very closely reflects the recall graph. Due to the unaugmented model having a slightly better precision than the fine-tune models they close the gap slightly in the F1-score but the fine-tune model approach still has the best F1-score.

For the character distribution distance comparison we get the below chart:(lower is better)



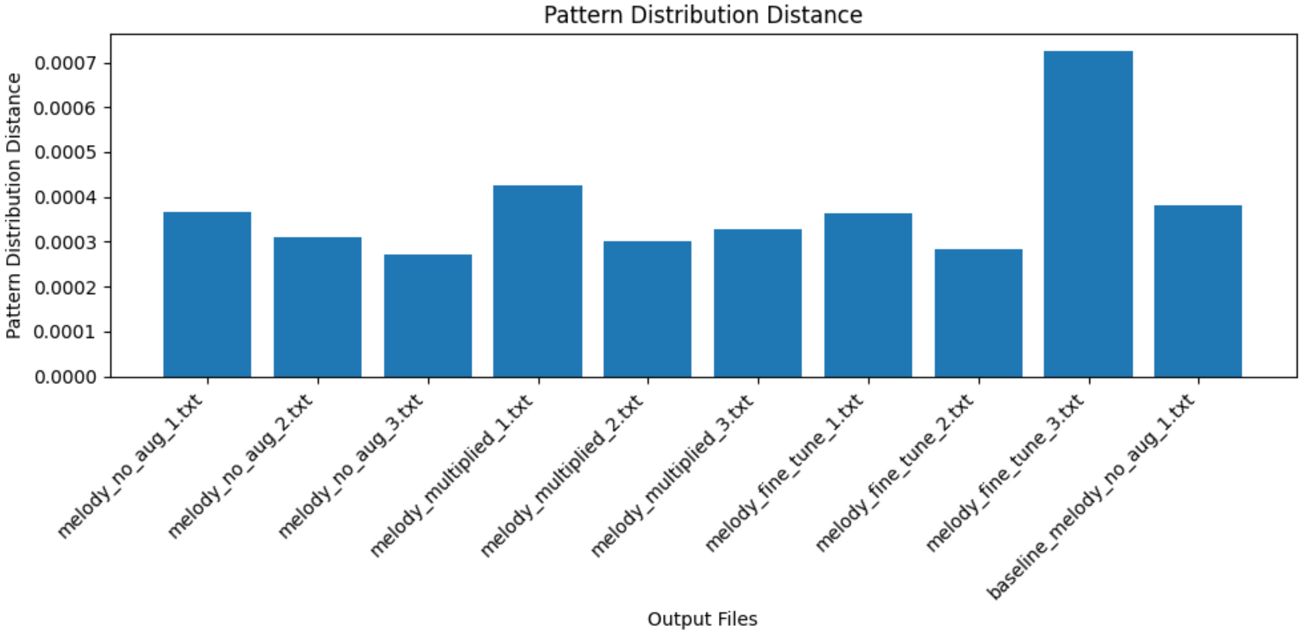
The character distribution distance is mainly a sanity check that shows if a model is completely off mark and we can see this with the fine-tune model 3 and the baseline model. The baseline model and fine-tune model 3 having the same character distribution distance is because they are both predicting the same character for the entire output. We can see that the separate model 1 has a normal distance and this is because it was able to capture the pitch but not the timing.

For the pattern length distribution distance comparison we get the below chart:(lower is better)



The pattern length distribution distance is a metric to show if the model is capturing the timing correctly. As predicted above the separate model 1 has a very high distance as it was unable to capture the timing correctly. We can also see that the baseline model is not massively off. This is due to the model predicting the most common sequence length which was 1. This is an example of why we need to use multiple metrics to evaluate a model.

For the pattern distribution distance comparison we get the below chart:(lower is better)



The pattern distribution distance is a metric to show if the model is capturing the patterns in the dataset. We can see that the same models shown to be performing well in the f1-score metric are also performing well here. This is because both metrics are comparing similar things but the pattern distribution distance also takes into account the count of the patterns occurring in the dataset. Due to this data being very repetitive most models are generating the patterns at the correct frequency so the metric gives similar results to the f1-score for this reason. It could for another dataset provide a more useful insight though.

Discussion

The results show that the fine-tune model which was trained first on pitch and then on both pitch and timing together performed the best, this was due to it getting the pitch correct in a simplified setting and using the obtained knowledge to finish the training. The poorest model was the model for which pitch and timing were trained separately, reading through some of the output this is because although the model produced a good model of timing and produced a good model of pitch it did not know how to match the pitch to the timing correctly, instead a naive multiplication was performed. This approach could have been much improved by training another model to combine them correctly. The unaugmented data model also performed well, however with a more complex dataset it would likely have fallen off worse than the other two model types.

Part 2

What is an ROC curve? How can it be used to evaluate the performance of a classifier compared with a baseline classifier? Why would you use an ROC curve instead of a classification accuracy metric?

Give two examples of situations where a linear regression would give inaccurate predictions. Explain your reasoning and what possible solutions you would adopt in each situation.

The term ‘kernel’ has different meanings in SVM and CNN models. Explain the two different meanings. Discuss why and when the use of SVM kernels and CNN kernels is useful, as well as mentioning different types of kernels.

In k-fold cross-validation, a dataset is resampled multiple times. What is the idea behind this resampling i.e. why does resampling allow us to evaluate the generalisation performance of a machine learning mode. Give a small example to illustrate. Discuss when it is and it is not appropriate to use k-fold cross-validation.