CS-UY 4563 Introduction to Machine Learning
Written Report for the Final Project

Betting and Formula 1

The tales of Charles, a questionable gambler.

11:00 AM Section
April 26, 2022
Professor Linda N. Sellie
Juan Castano, Rishyak Panchal

Introduction

Formula 1 is the world's most popular motorsport, considered the highest class of open-wheel racing by its millions of fans worldwide. It has been growing rapidly with many celebrity endorsements and large partnerships from brands like Zoom, Aramco, Rolex, Netflix, and Emirates, to name a few. Formula 1 currently holds a valuation of 18 million dollars, the highest it has ever been. It shows no signs of slowing down, with its constant growth in new markets like North America, the Middle East, and East Asia. With this in mind, betting on F1 is at an all-time high.

We are not gamblers, but we wanted to see how well we could predict race-finishing orders in case we felt too rich someday. In pursuit of this, we chose the best F1 dataset on Kaggle. This dataset is a rich goldmine of all available information since the conception of the sport in 1950. The dataset contains information about each event in each race, where a driver finished, where they were born, what team they drove for during the event, and so on.

Finally, we tried logistic regression, support vector machines, and neural networks to predict finishing positions. We tried different regularisation parameters, feature transformations, and penalties for each, culminating in a boatload of information about our model's performance.

This paper analyses each method's strengths and weaknesses, with a detailed conclusion with the best model. We also enumerate overall weaknesses and further improvements that could be made for a better model.

Nomenclature

Before we move forward, we should briefly talk about F1 jargon to better understand the contents of this research. They are quite self-explanatory, even to someone with no interest in racing, but it can help contextualise why these words are what they are

Word	Meaning
Driver/Pilot	The person who pilots a racecar around a track
Constructor	The team that designs and builds cars, hires drivers, and races cars
Circuit	The race track where an F1 weekend takes place

Table 1 F1 Jargon and its Meaning

Preprocessing

The first step for carrying out all the tests was to do preprocessing. Most of the data was spread out in many different files, so we had to combine everything together to have all the necessary data in one compact data frame. After analysing what columns were needed from the dataset, we determined that the most important features to include in our model were:

Feature	Description
Grid Position	The starting position of a driver for a race
Driver Standing	Driver's position in the championship, indicative of success
Constructor Standing	Constructor's position in the championship, indicative of success
Driver Wins	Total number of wins for the driver, indicative of experience
Constructor Wins	Total number of wins for the constructor, indicative of experience
Driver Points	Points won by a driver during a championship, indicative of success
Constructor Points	Points won by a constructor during a championship, indicative of success
Circuit Latitude	North-South position of a circuit, affects weather and temperatures
Circuit Longitude	East-West position of a circuit, affects sunlight, times, and tyre behaviour
Circuit Altitude	Circuit height, affects oxygen density, aerodynamics, cooling, combustion

Table 2 Features and their descriptions

Finally, we determined that the target variable should be the final position of each driver for a race.

One of our larger problems with this dataset was that some rows had "\N" for the driver's final position. This corresponds to when the driver cannot finish the race for many reasons, such as disqualification for bad behaviour, car failure, crashes, collisions, etc. What we decided to do about this was to replace these custom N/As with the last place. This is because we determined that not finishing the race was the same as finishing last. So, we assigned the last position to anyone who failed to complete the race. Ultimately, this doesn't matter due to more transformations later on.

The dataset we chose was quite large, given that it was made of data starting from 1950, so we decided to consider entries after 2014. This is because technology has significantly changed with the years, and 2014 marked the year when they started using new turbo-hybrid engines, significantly different from the purely internal combustion engines used before. Along with the new engines, several car components, such as safety systems, braking systems, driver positioning, and car aerodynamics,

were changed. This shuffled the order as drivers who previously performed well would perform poorly, or some constructors who traditionally did well produced terrible cars. So, it was wise to let our model train with this new technology since previous results showed little relation to the current ones.

After filling up the previously mentioned empty entries, we imported all the files and combined them into one matrix with the necessary features. The resulting matrix had 3,707 rows, each corresponding to the features of a driver at the moment of a specific race. Then, we divided it into **`X_train`**, **`X_test`**, **`Y_train`**, and **`Y_test`**. We picked a train-test partition of 80% for training data and 20% for test data.

Since we used logistic regression instead of linear regression, we needed to specify labels for the target. Throughout the project, we carried out two versions of each model, one multiclass and one with only two classes. The labels for the binary model were whether or not the driver made it into the top 3, the podium. On the other hand, for the multinomial model, we had labels for first place, second place, third place, or no podium. Furthermore, all features data were standardised using scikit-learn's standard scaler. We shuffled our data at the same time, with a random seed of 42.

For logistic regression and neural networks, we encoded all non-podium values as 0, and for support vector machines, we encoded non-podium values as -1. As for the others, the labels remained 1, 2, or 3, depending on the finishing position.

Logistic Regression

This was run using the scikit-learn Logistic Regression algorithm. It was run several times so we could try different linearisation parameters. We performed L2 regularisation, starting with a lambda of 0 until reaching a lambda of 1. We tried many feature transformations for every lambda, and the training and testing accuracies were plotted.

The plots obtained for a lambda of 0 are displayed in **Figure 1**. As can be observed, it seems like the data followed a linear pattern since the highest testing accuracy we got was for degree 1. The test score corresponding to the linear model was 88.95%, higher than all the other transformation models.

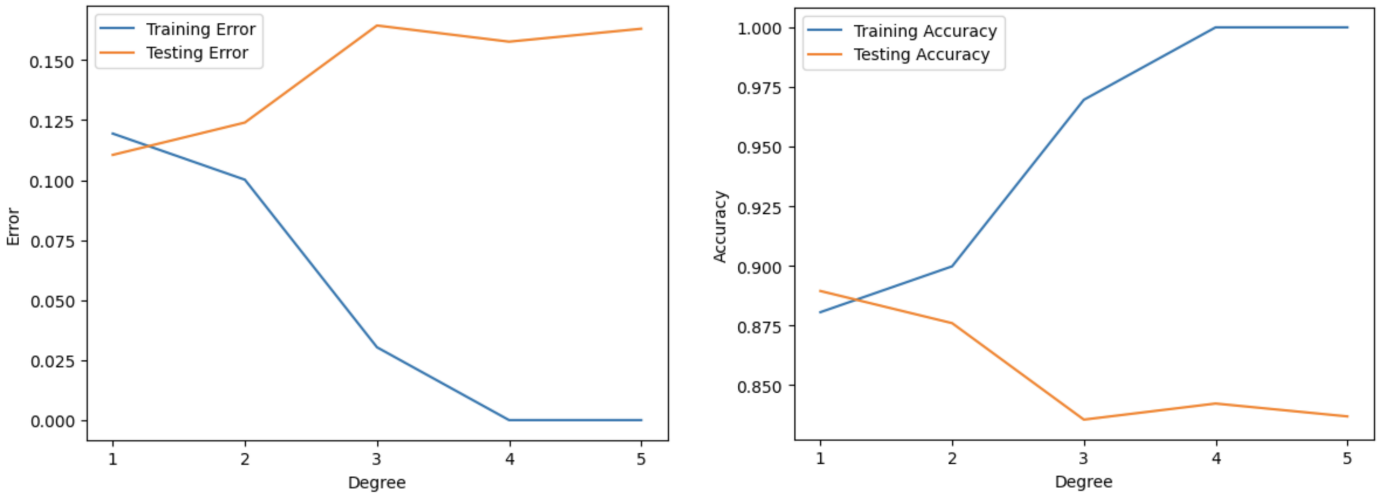


Figure 1 Accuracy and Error for lambda = 0

Then, we tried using a lambda of 0.01, and the resulting plots are in **Figure 2**. Once again, the best testing accuracy obtained was for the linear model, with 88.95%.

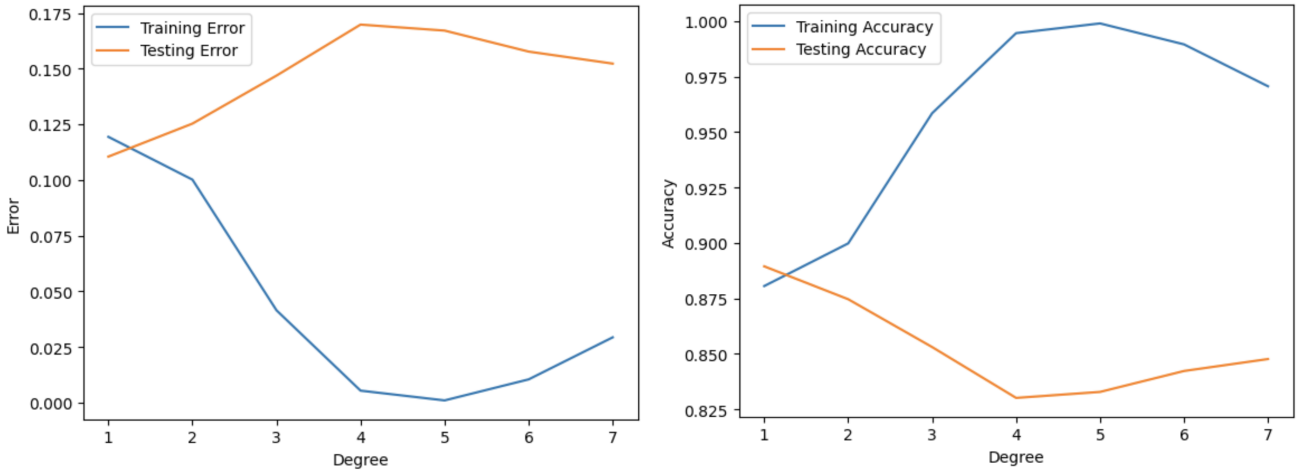


Figure 2 Accuracy and Error for lambda=0.01

The plots obtained for a lambda of 0.1 are shown in **Figure 3**, and it can be seen that the data continued showing a linear trend as that was the highest testing accuracy, more specifically 88.95%.

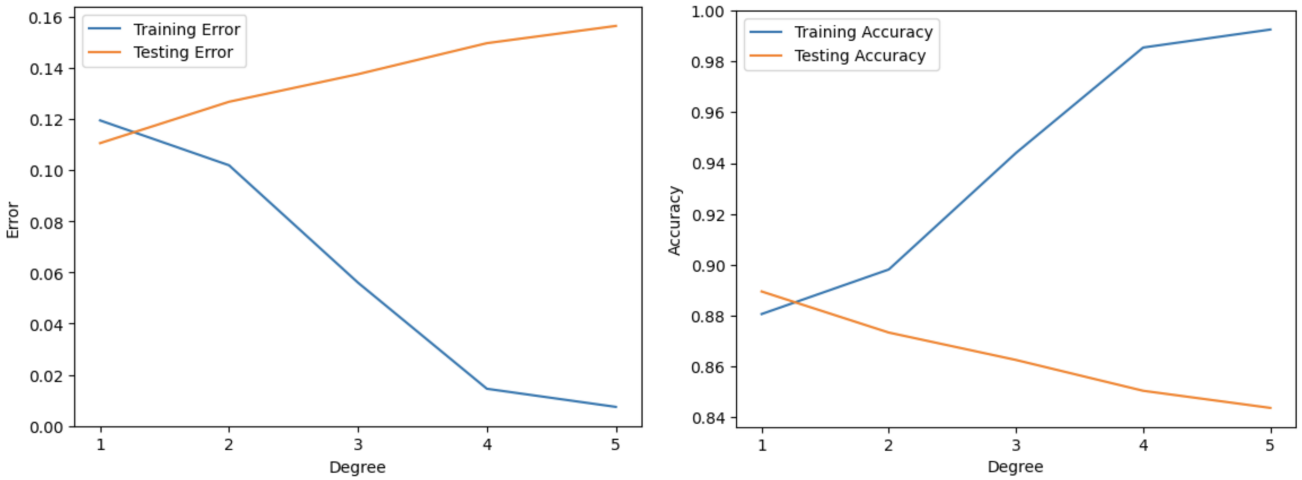


Figure 3 Accuracy and Error for lambda=0.1

Finally, the same was repeated for a lambda of 1, and the results are displayed in **Figure 4**. The linear pattern was observed again, but its testing accuracy was slightly lower, corresponding to 88.81%.

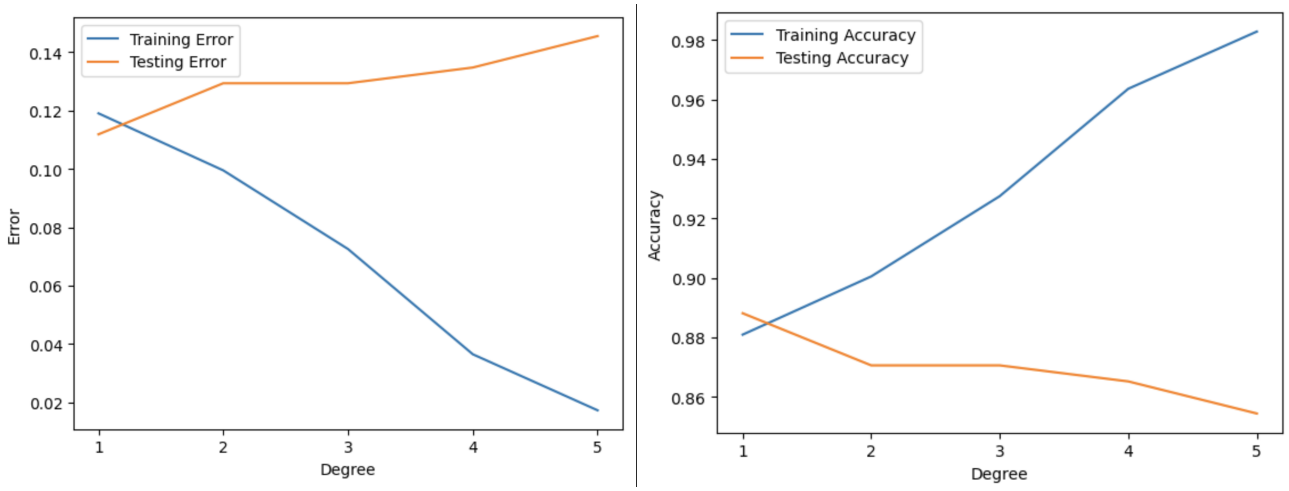


Figure 4 Accuracy and Error for lambda=1

After having tested all transformations and all lambda values for the multi-class version of our program, we repeated the same steps for the binary classification.

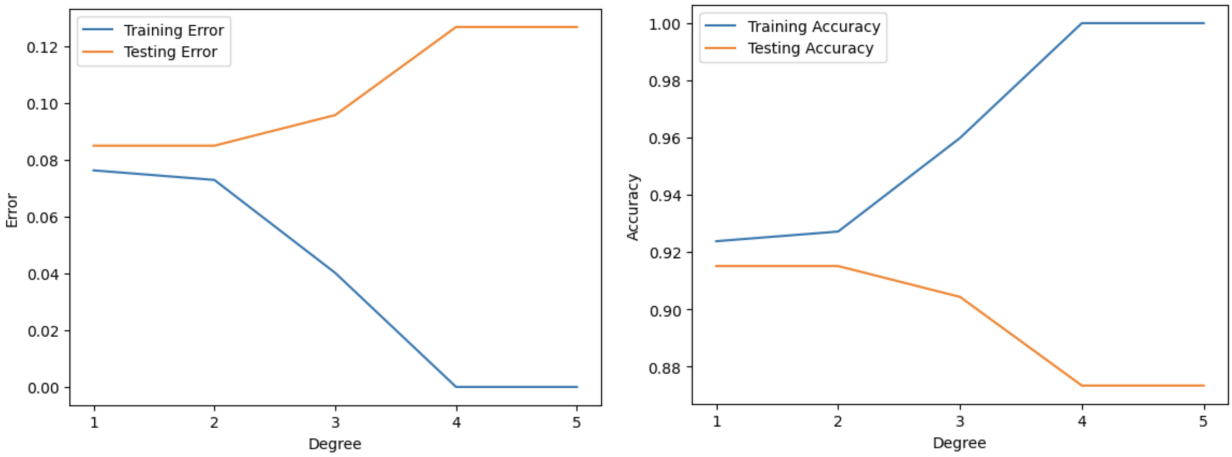


Figure 5 Accuracy and Errors for Binary model, lambda=0

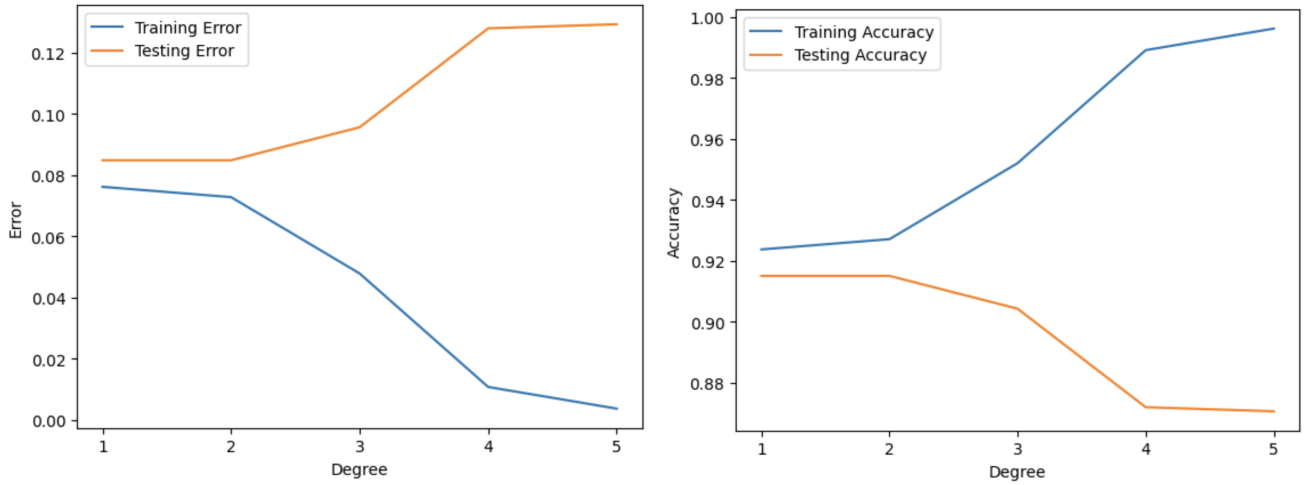


Figure 6 Accuracy and Errors for Binary model, lambda=0.01

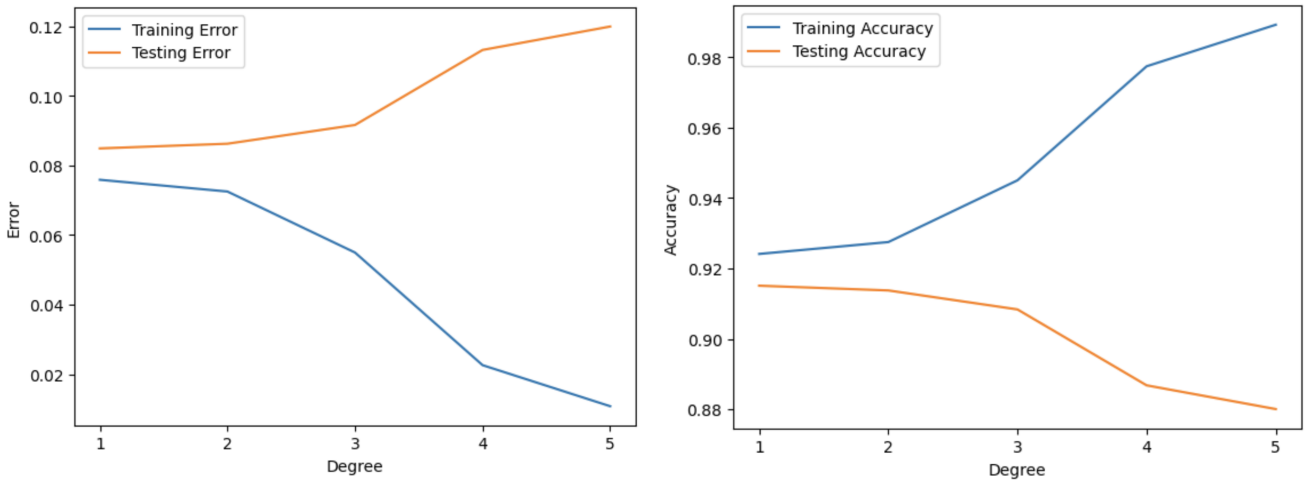


Figure 7 Accuracy and Errors for Binary model, lambda=0.1

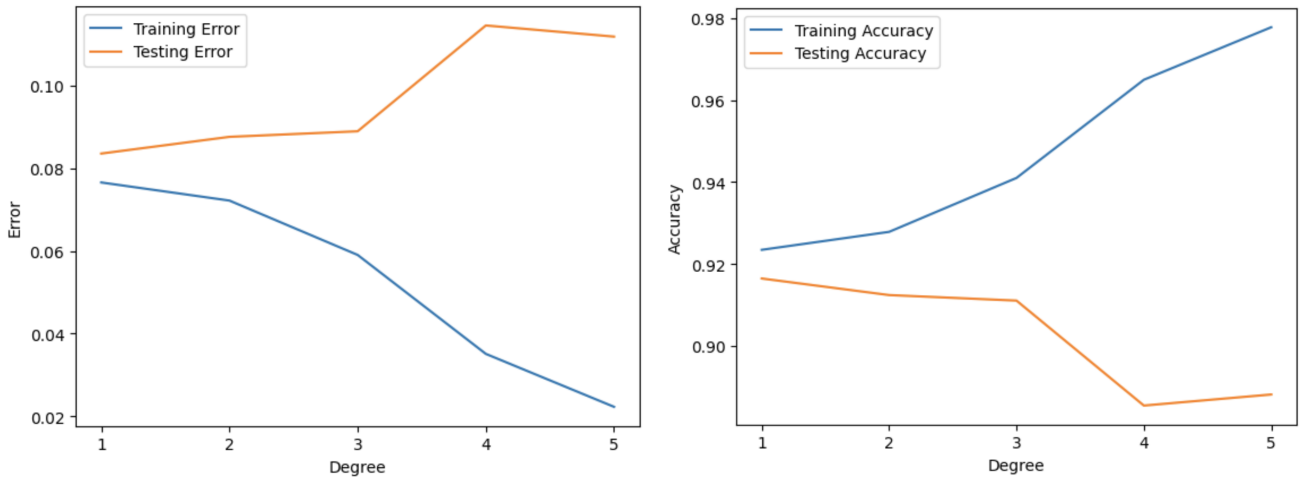


Figure 8 Accuracy and Errors for Binary model, lambda=1.0

Figures 5, **6**, **7**, and **8** show that the binary model also showed a linear behaviour since all the testing accuracies were maximum for degree 1. The highest accuracy obtained was 91.64%, corresponding to the case when lambda was 1. It is also important to note that the binary model had a higher accuracy overall. This is because it becomes much easier to predict when only two labels exist. But the accuracy drops when the model tries to predict specific places compared to the binary model. There is also some overfitting for all lambdas run, which can be inferred because, when the feature transformations were too high, the training accuracy was very close to 100%, but the testing accuracy was actually dropping.

We generally want to choose the model with the best testing accuracy. All our tests for logistic regression indicated that linear was the best feature transformation for this problem.

For a summary of the results obtained from the logistic regression section, please see the following tables:

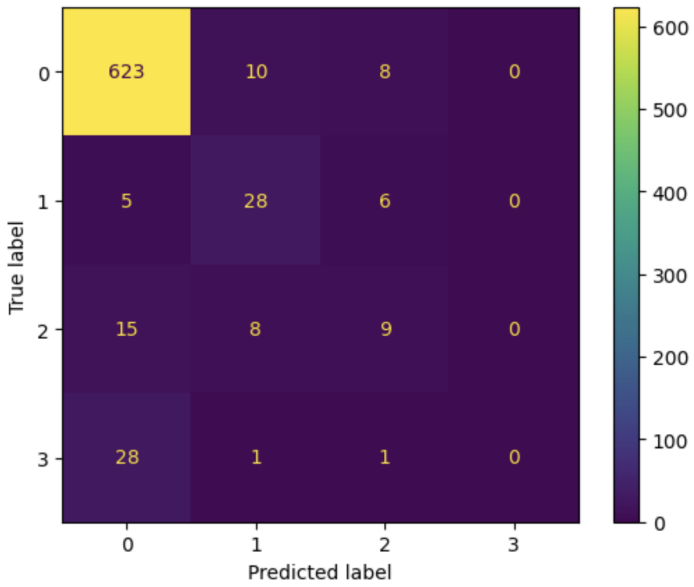
	Multinomial Model							
	Lambda=0		Lambda=0.01		Lambda=0.1		Lambda=1	
Degree	Test accuracy	Training accuracy	Test accuracy	Training accuracy	Test accuracy	Training accuracy	Test accuracy	Training accuracy
1	88.95%	88.06%	88.95%	88.06%	88.95%	88.06%	88.81%	88.09%
2	87.60%	89.98%	87.47%	89.98%	87.33%	89.81%	87.06%	90.05%
3	83.56%	96.96%	85.31%	95.85%	86.25%	94.40%	87.06%	92.75%
4	84.23%	100.00%	83.02%	99.46%	85.04%	98.55%	86.52%	96.36%
5	83.69%	100.00%	83.29%	99.90%	84.37%	99.26%	85.44%	98.28%

Table 3 Accuracies and Errors for the multinomial logistic model

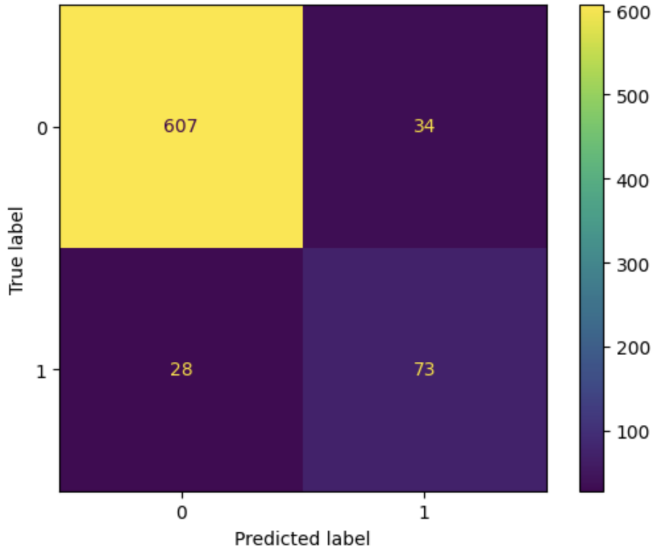
	Binary Model							
	Lambda=0		Lambda=0.01		Lambda=0.1		Lambda=1	
Degree	Test accuracy	Training accuracy	Test accuracy	Training accuracy	Test accuracy	Training accuracy	Test accuracy	Training accuracy
1	91.51%	92.38%	91.51%	92.38%	91.51%	92.41%	91.64%	92.34%
2	91.51	92.72%	91.51%	92.72%	91.37%	92.75%	91.24%	92.78%
3	90.43%	95.99%	90.43%	95.21%	90.84%	94.50%	91.11%	94.10%
4	87.33%	100%	87.20%	98.92%	88.68%	97.74%	88.54%	96.49%
5	87.33%	100%	87.06%	99.63%	88.01%	98.92%	88.81%	99.77%

Table 4 Accuracies and Errors for the binary logistic model

Furthermore, we also created confusion matrices for the best models obtained from both binary and multinomial, they are shown in **Matrices 1** and **2** below. As can be seen, both models are really good at predicting which drivers are not going to be in the top 3 podium, since in both matrices that is the only yellow box. However, our models also predicted first places fairly well, that being the second most correctly predicted label. Still, there was more room for improvement, especially since the rest of the labels in the multinomial model got many wrong predictions.



Matrix 1 Confusion matrix for linear multinomial model, lambda=0.1



Matrix 2 Confusion matrix for linear binary model, lambda=1

Support Vector Machines

We trained our model using scikit-learn's support vector classifier, SVC. We faced many challenges when working with SVMs. The largest was the time it took to train with some hyperparameters. Eventually, we found settings that worked. Our goal was to test many kernels with different hyperparameters and penalties.

With the many optimisations that scikit-learn offers, we got carried away with our experiments, as you will soon see.

Polynomial Kernel

We started with a polynomial kernel with a baseline cost of 0.1, the results of which can be seen below in **Figure 9**. It is also apparent from **Figure 9** that we observe the best results at degree three, with an accuracy of 88.68%, after which the model begins to overfit severely as training error approaches zero and training accuracy reaches all-time highs. It was evident that our model captured noise in the data instead of general trends after degree 11.

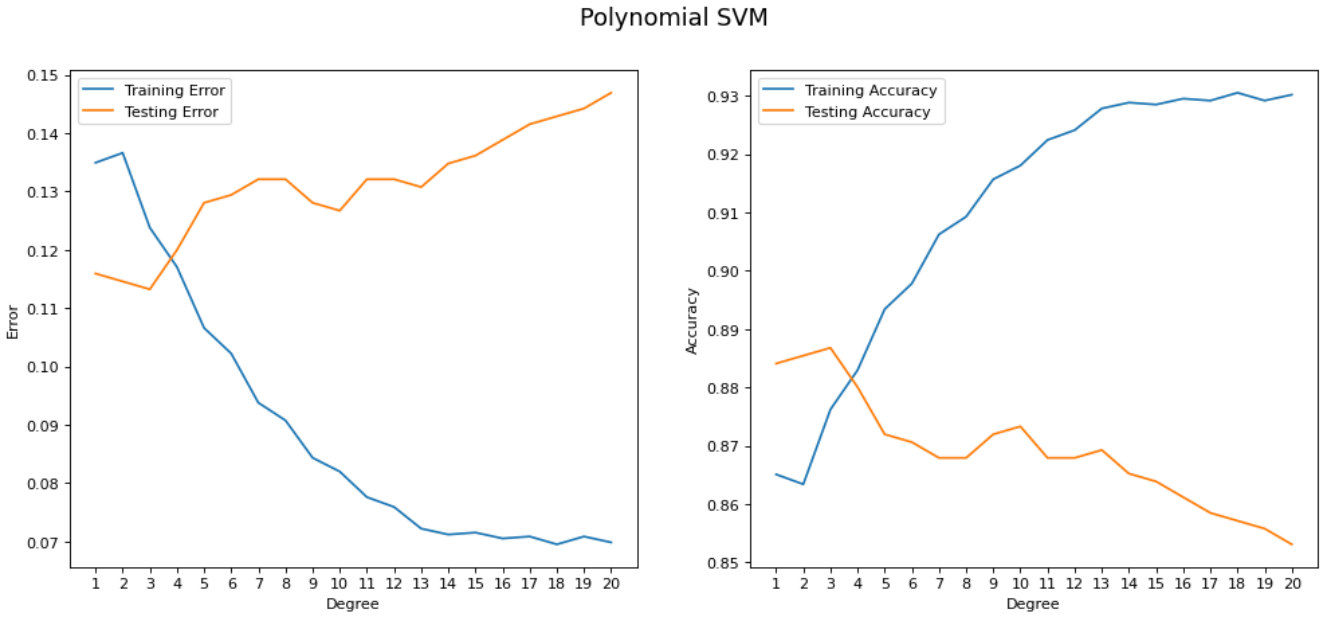


Figure 9 Errors and accuracy for bog-standard polynomial SVM up to 20 degrees

Degree	Test accuracy	Training accuracy
1	88.41%	86.51%
2	88.54%	86.34%
3	***88.68%***	***87.62%***
4	88.01%	88.30%
5	87.20%	89.34%
6	87.06%	89.78%
7	86.79%	90.62%
8	86.79%	90.93%
9	87.20%	91.57%
10	87.33%	91.80%

11	86.79%	92.24%
12	86.79%	92.41%
13	86.93%	92.78%
14	86.52%	92.88%
15	86.39%	92.85%
16	86.12%	92.95%
17	85.85%	92.92%
18	85.71%	93.05%
19	85.58%	92.92%
20	85.31%	93.02%

Table 5 Testing and training accuracy for the standard polynomial with c = 0.1

After quickly realising that 20 degrees are too many degrees, we scaled down to 10 for the next set of experiments with the polynomial kernel. We were confident that we could extract more out of this SVM with this, so we decided to forgo the confusion matrix for this configuration.

Polynomial Kernel with Different Costs

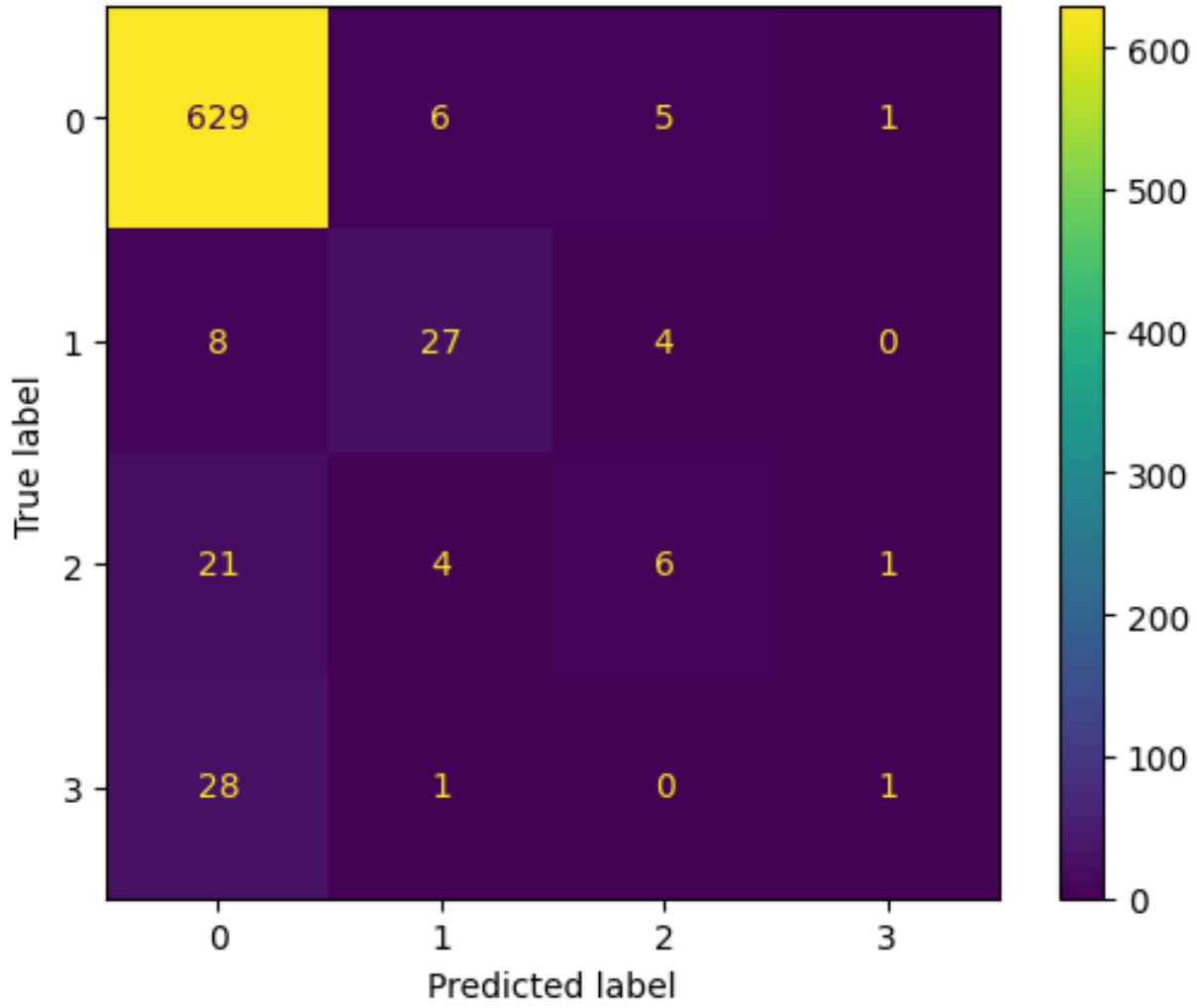
While defaulting to 0.1, we wanted to examine how the kernel behaves with different cost hyperparameters. To that effect, we picked a range from 1e-4 to 1e3 and tested all orders of magnitude in the middle. This was an exercise to find the tradeoff between model complexity and training error, effectively a hand-written cross-validation, evaluating the model at different degrees and costs.

Cost	0.0001		0.001		0.01		0.1	
Degree	Test accuracy	Training accuracy	Test accuracy	Training accuracy	Test accuracy	Training accuracy	Test accuracy	Training accuracy
1	86.39%	84.99%	86.39%	84.99%	87.20%	85.53%	88.41%	86.51%
2	86.39%	84.99%	87.06%	85.50%	88.14%	86.14%	88.54%	86.34%
3	86.79%	85.40%	87.47%	85.94%	88.27%	86.48%	88.68%	87.62%
4	87.20%	85.70%	87.87%	86.14%	88.27%	86.75%	88.01%	88.30%
5	87.47%	85.94%	88.27%	86.68%	87.60%	87.49%	87.20%	89.34%
6	88.27%	86.61%	88.14%	87.32%	87.20%	88.36%	87.06%	89.78%
7	87.87%	87.22%	87.60%	87.82%	86.79%	89.14%	86.79%	90.62%
8	87.47%	87.55%	87.06%	88.26%	86.93%	89.51%	86.79%	90.93%
9	86.39%	88.13%	86.66%	88.84%	86.66%	90.32%	87.20%	91.57%
10	86.25%	88.47%	86.93%	89.54%	86.66%	90.59%	87.33%	91.80%
Cost	**1**		***10***		**100**		**1,000**	
Degree	**Test accuracy**	**Training accuracy**	**Test accuracy**	**Training accuracy**	**Test accuracy**	**Training accuracy**	**Test accuracy**	**Training accuracy**
1	88.95%	87.28%	88.81%	88.09%	88.81%	88.33%	89.08%	88.26%
2	88.68%	87.25%	***89.35%***	88.33%	89.08%	88.80%	89.22%	88.97%
3	88.27%	89.61%	88.14%	90.86%	87.33%	92.68%	86.12%	94.44%
4	87.74%	89.98%	87.20%	91.74%	87.06%	94.33%	85.18%	96.73%
5	87.20%	91.13%	86.12%	93.42%	86.12%	95.28%	85.18%	97.34%
6	86.93%	91.74%	85.98%	93.96%	86.39%	95.89%	84.91%	97.74%
7	87.06%	92.82%	86.25%	94.47%	86.66%	96.12%	86.25%	97.84%
8	86.52%	93.12%	87.06%	94.64%	86.79%	96.19%	85.98%	97.71%
9	86.39%	93.56%	87.47%	94.74%	86.12%	96.19%	86.52%	97.47%
10	86.12%	93.59%	87.47%	94.74%	86.12%	96.05%	85.58%	97.37%

Table 6 Testing and training accuracy for polynomial kernel with different costs

A trend in **Figures 10** to **17** below is that the degree at which we observe the highest testing accuracy increases as the cost parameter decreases. This is quite sensible because the whole point of increasing the cost is to reduce the risk of overfitting the model, making the model more tolerant of misclassifications, by increasing the margin boundary to better generalise to unseen data. The degree dictates the complexity of the decision boundary. Effectively, our results suggest that our decision boundaries are more complex with smaller costs. Perhaps the model is overfitting at lower costs. It is also likely that the data is not polynomial at all and using a polynomial kernel adds unnecessary complexity by creating a feature space that can capture minute patterns in the data, leading to easy overfitting. We observed the best results at degree 2 with a cost of 10 at 89.35%.

Below, **Matrix 3** shows the confusion matrix for the model with the best performance - degree 2, cost 10. We were blown away by the 70% accuracy of correctly predicting race winners. The model is, as expected, good at classifying non-podium finishers. The same cannot be said for positions two and three. In fact, for position 3, our model is almost always misclassified as non-podium. This is quite fair knowing the nature of the sport where the third podium position can go to just about anybody if you consider events like weather, crashes, collisions, bad strategy, pit-stops, etc. It is the most highly contested position since it's the most reachable one.



Matrix 3 Confusion matrix for RBF SVM with degree 2 and cost 10

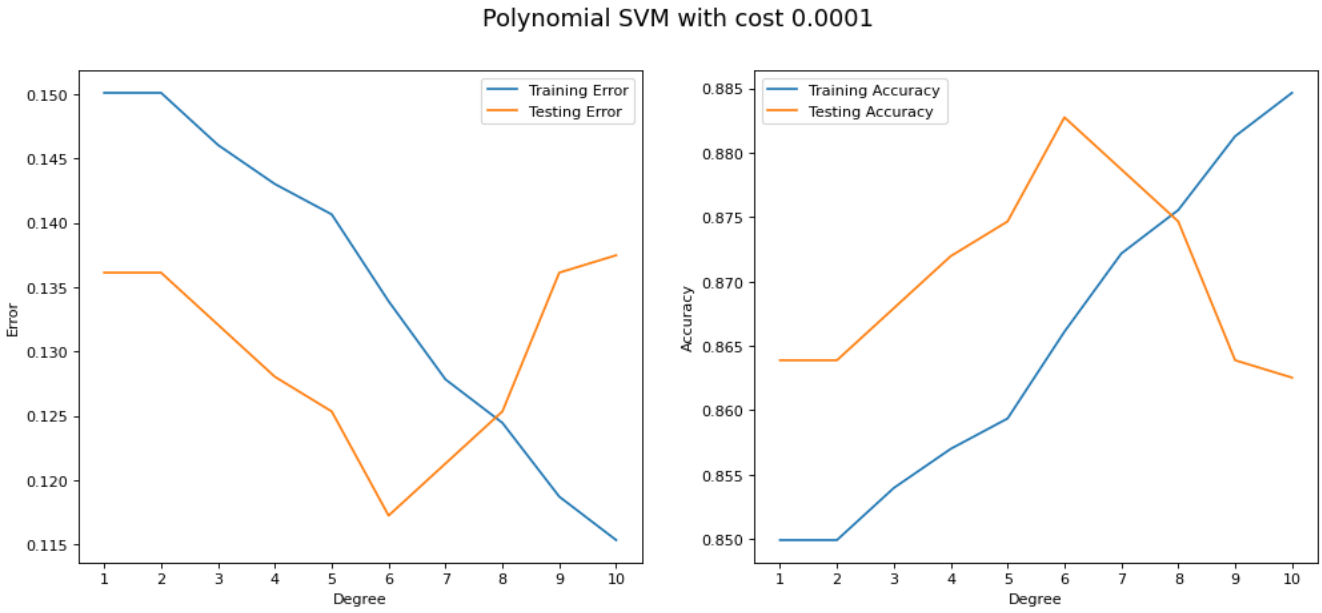


Figure 10 Errors and accuracy for polynomial SVM with cost = 0.0001

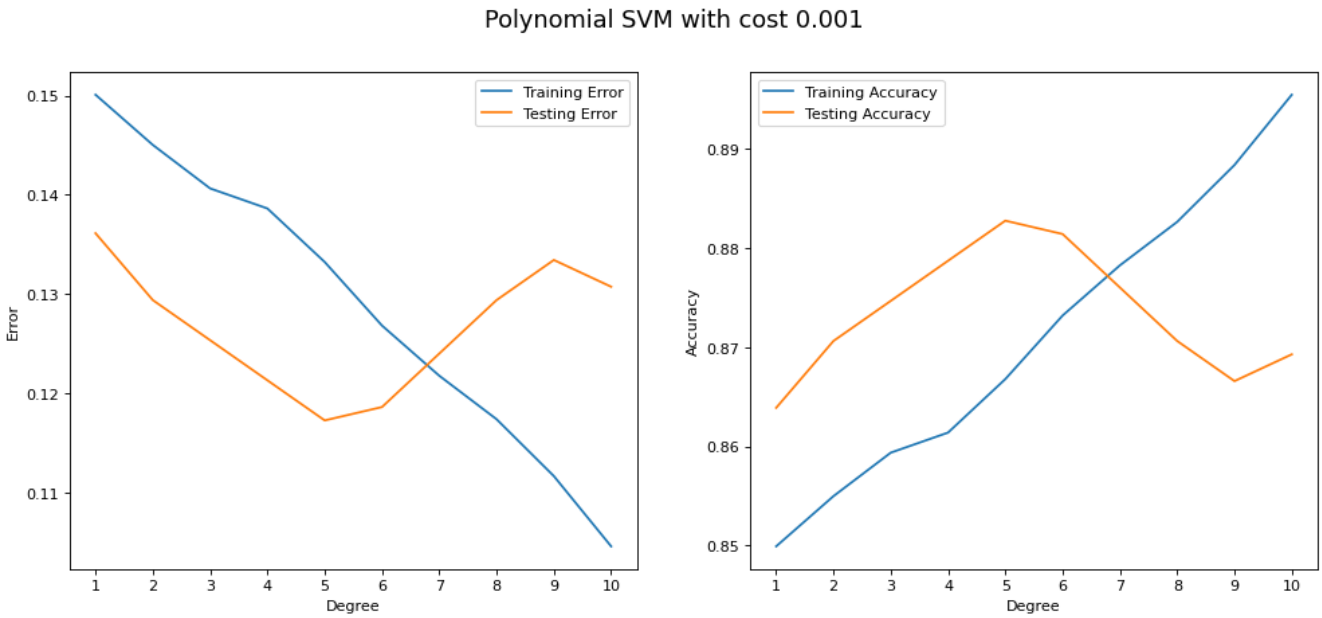


Figure 11 Errors and accuracy for polynomial SVM with cost = 0.001

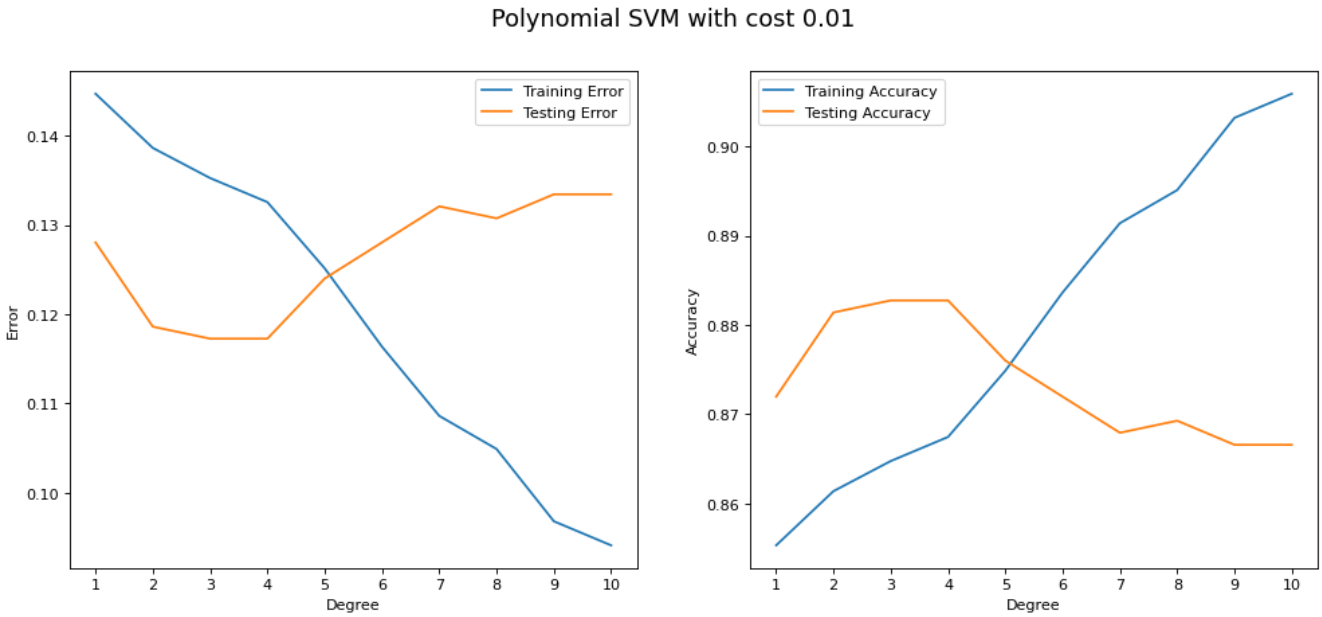


Figure 12 Errors and accuracy for polynomial SVM with cost = 0.01

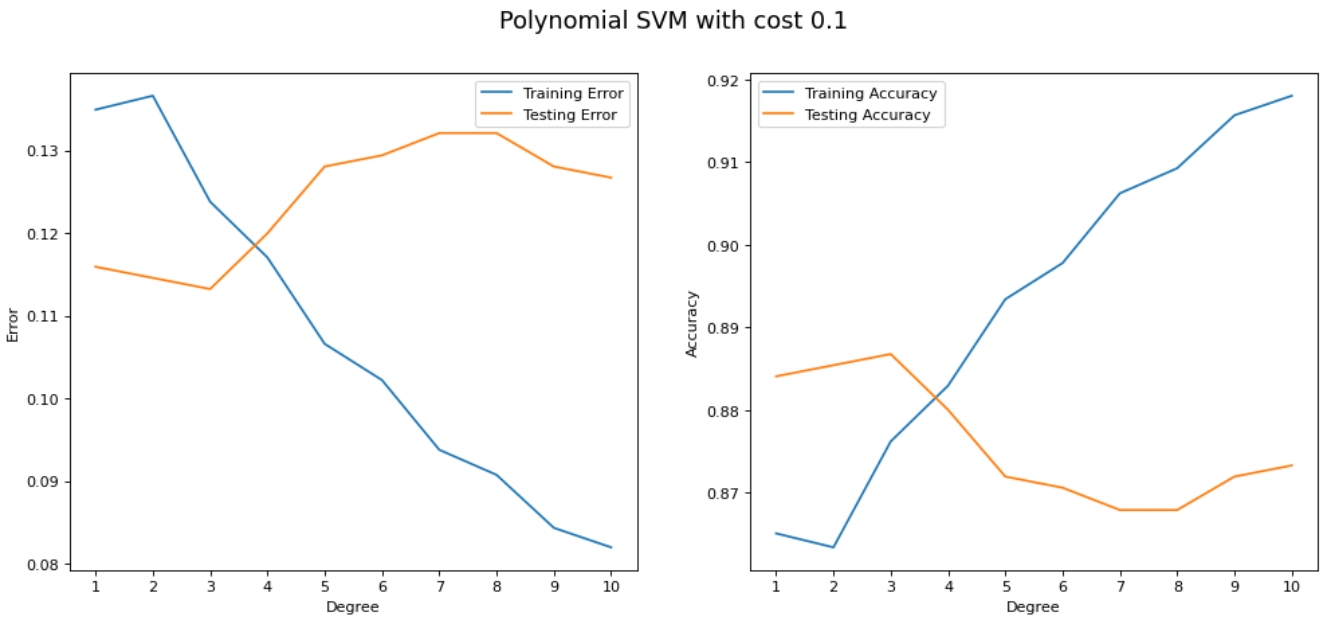


Figure 13 Errors and accuracy for polynomial SVM with cost = 0.1

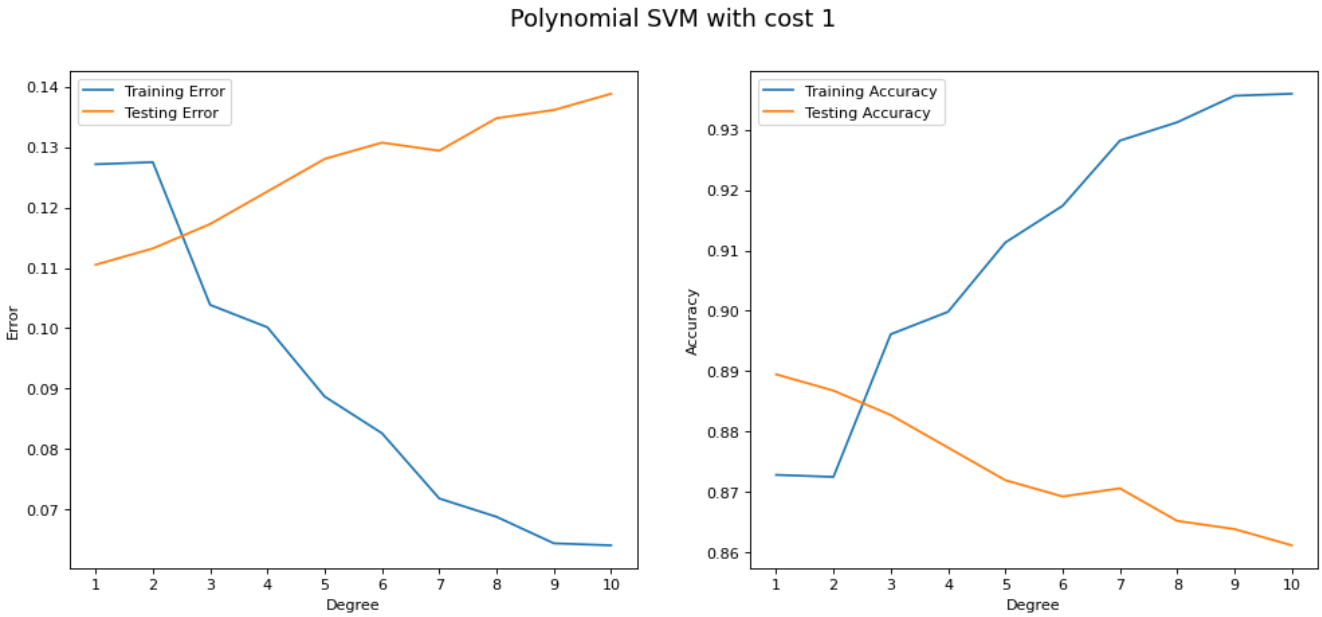


Figure 14 Errors and accuracy for polynomial SVM with cost = 1

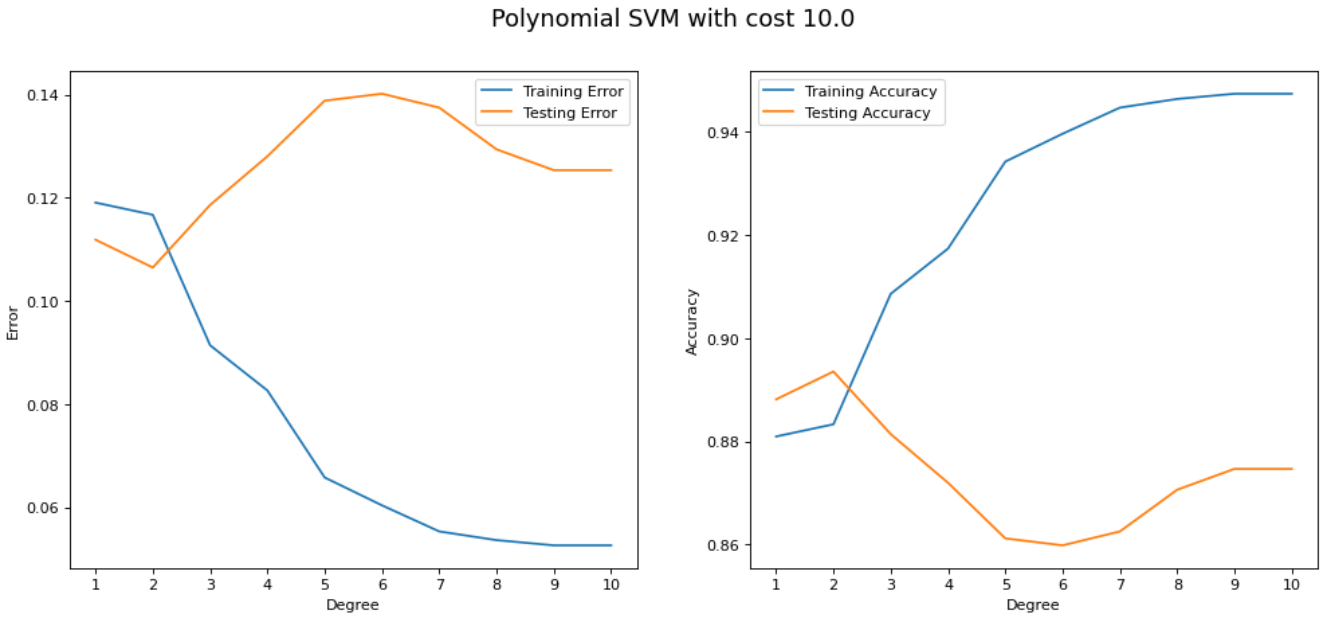


Figure 15 Errors and accuracy for polynomial SVM with cost = 10

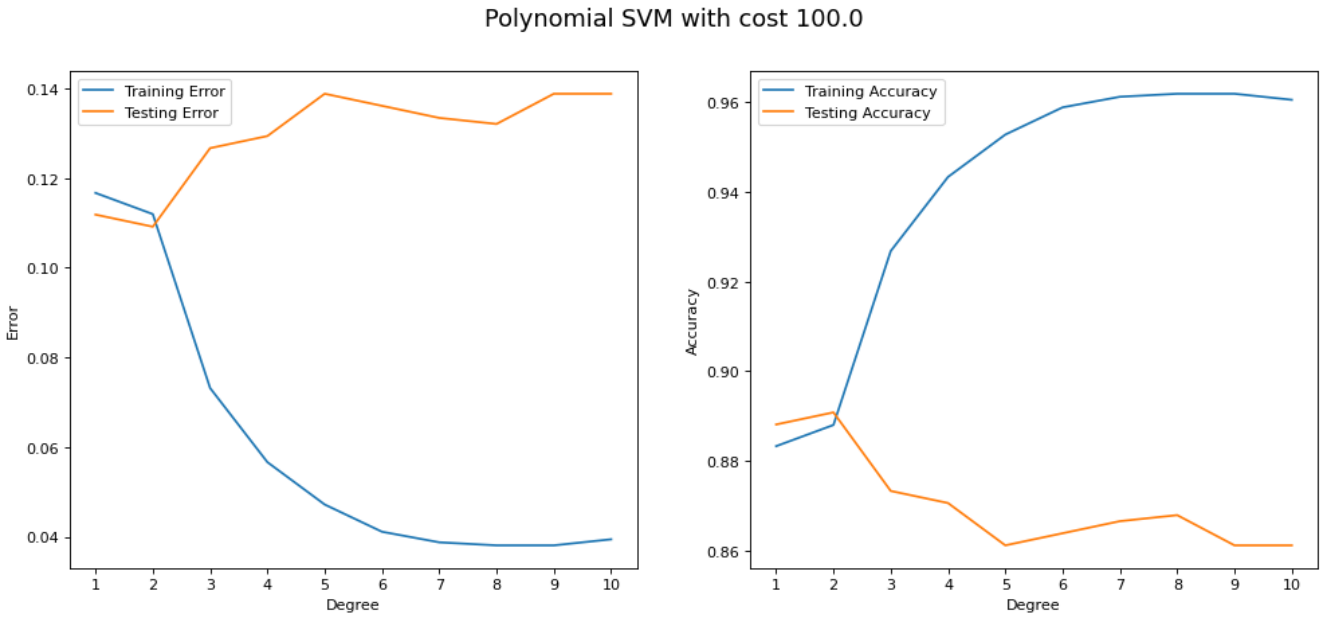


Figure 16 Errors and accuracy for polynomial SVM with cost = 100

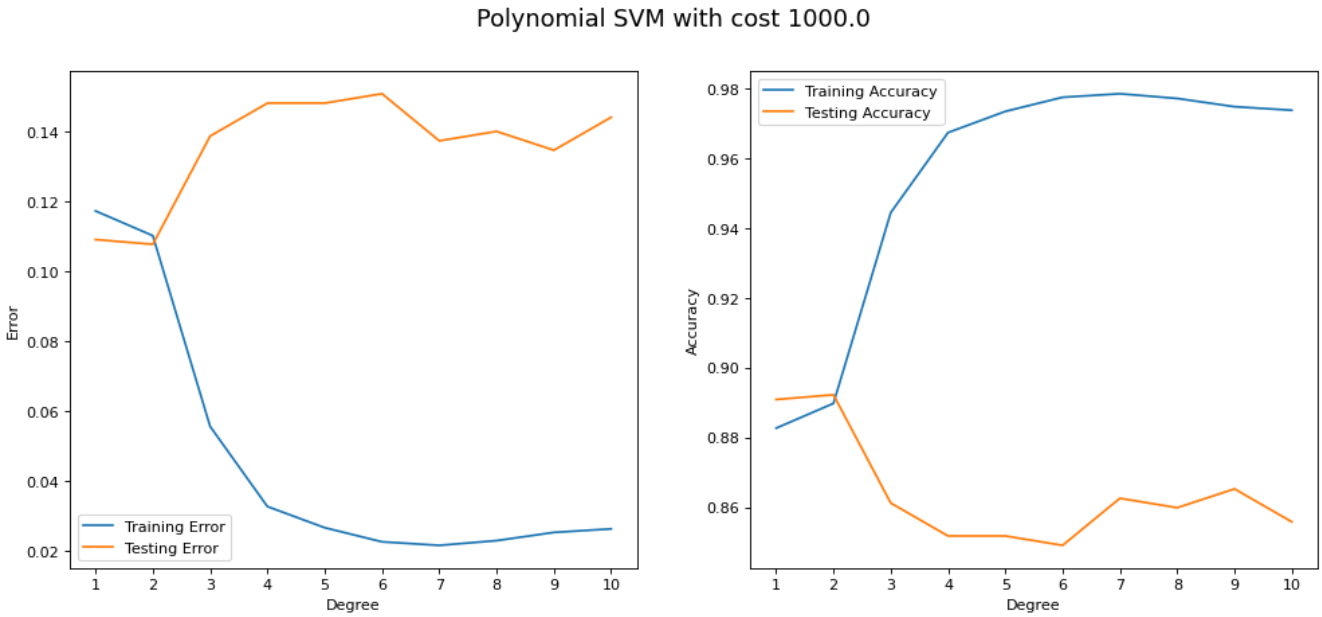


Figure 17 Errors and accuracy for polynomial SVM with cost = 1000

RBF Kernel with Different Gammas

With the information we gathered from the polynomial kernel, we realised that our data could be more complex than a simple polynomial kernel can let on. So, we wanted to test our model with the famous RFB kernel due to its expertise with complex data. The RBF kernel also has two tuning parameters, making it more customisable.

To find the best RBF kernel, we, once again, started with a baseline gamma of 0.1 and tried to find the best cost between the same range as above (1e-4 to 1e3). We observed our best results with a cost of 0.1. Anything smaller than that and our model accuracy plateaus, suggesting that our model got too simple and was underfitting the data. With a low cost parameter, our model is more tolerant of misclassifications, resulting in a simpler decision boundary. It is also important to acknowledge the cliff after the cost of 0.1, where the model begins to rapidly overfit. This means that a gamma of 0.1 *only* works with a cost of 0.1 in our case. The findings can be visualised in **Figure 18** and **Table 7** below.

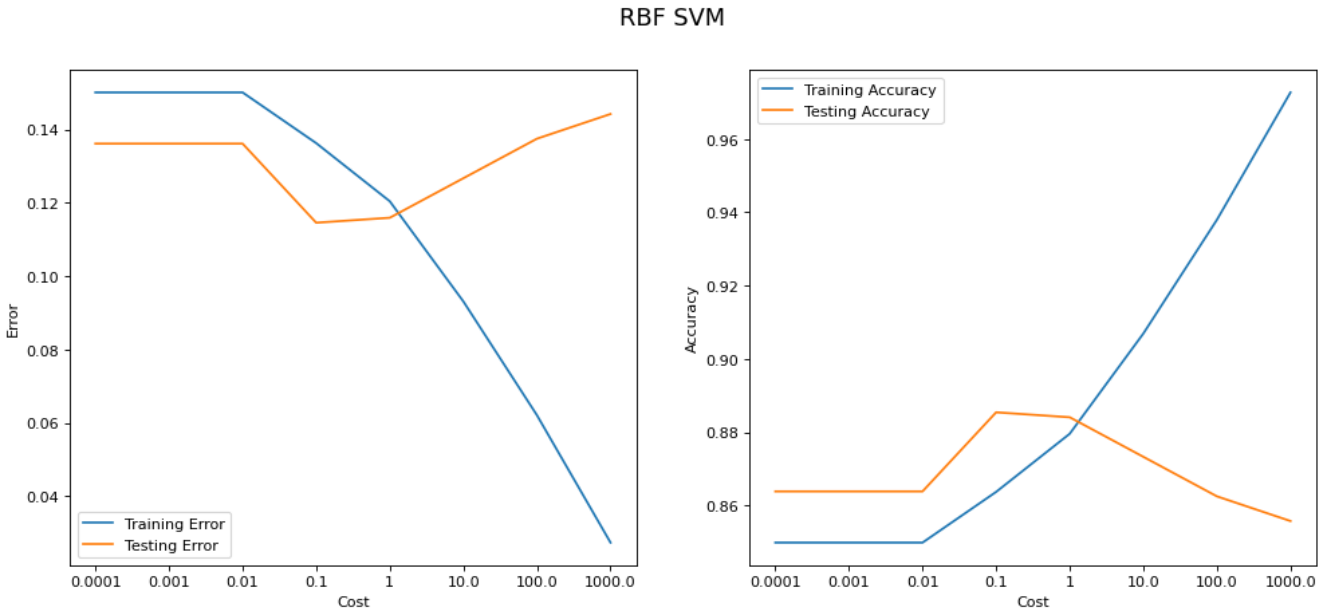


Figure 18 Errors and accuracy for polynomial SVM with gamma 0.1

Cost	Test accuracy	Training accuracy
0.0001	86.39%	84.99%
0.001	86.39%	84.99%
0.01	86.39%	84.99%
0.1	***88.54%***	***86.37%***
1	88.41%	87.96%
10	87.33%	90.69%
100	86.25%	93.79%
1000	85.58%	97.27%

Table 7 Test and train accuracy for polynomial SVM with gamma 0.1

Next, we wanted to test the effect of a varying gamma with varying costs because there's no guarantee of which set of parameters work well together. In **Table 8** below, we have enumerated the

testing and training accuracy for the best combination of gamma and cost. It is quickly noticeable that we got the best testing accuracy with a gamma of 0.001 and a cost of 1,000 at 89.22%. **Figure 19** visualises that model at different costs. A cost of 1,000 was indeed the best one, after which the testing accuracy drops and the model begins to capture idiosyncrasies in the data and overfitting.

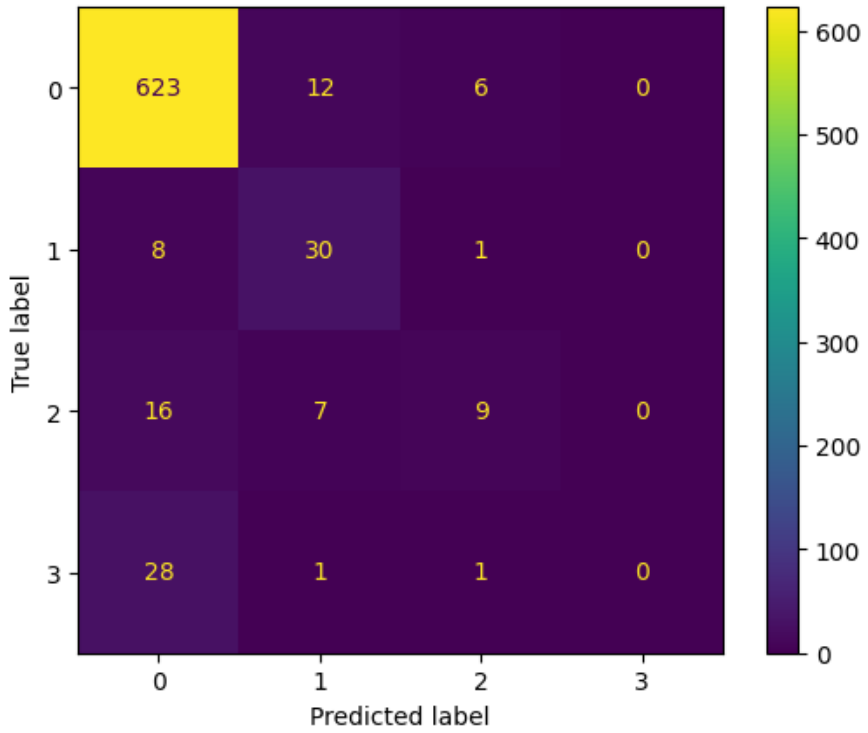
Matrix 4 below shows the confusion matrix for this model. We can quickly observe that it is very good at classifying non-podium finishers. However, we were surprised how good it was at predicting first place finishers, with an accuracy of about 60%.

Gamma	Best Cost	Test accuracy	Training accuracy
0.000001	1000	88.14%	86.07%
0.00001	1000	88.54%	86.81%
0.0001	1000	89.08%	87.35%
0.001	***1000***	***89.22%***	***88.13%***
0.01	100	88.81%	88.74%
0.1	0.1	88.54%	86.37%
1.000	1	87.33%	92.28%
10.00	1	86.39%	98.75%
100.0	1	86.39%	10000.00%

Table 8 Test and train accuracy for each gamma with the best cost parameter

Figure 19 Errors and accuracy for RBF SVM with gamma 0.001.

(we apologise for the x-axis)



Matrix 4 Confusion matrix for RBF SVM with gamma 0.001 and cost 1,000

Neural Networks

For the neural network section, we brought back the two models, multinomial, and binary. The first one we ran this time was the one with only two labels. Just like for the previous techniques, we also tried different models to see what would give us the best accuracy.

Most of the neural networks we trained consisted of 4 layers, meaning there was one input layer, two hidden layers, and one output layer. The main change from one layer to the other was the activation functions used. For the binary model, it was important to have only one neuron in the output layer since we do not need more for only two categories in the target. The neural network created visually looks like the one in **Figure 20**.

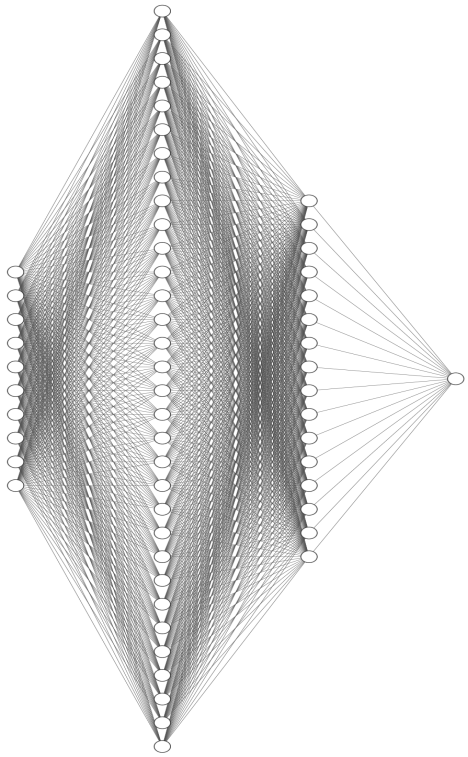


Fig 20. Binary model Neural Network

It was also important to have 10 neurons in the input layer since there were 10 input features, each of them would go to one respective neuron. Moreover, we used 32 neurons for the hidden layer 1, and 16 neurons for the hidden layer 2.

Table 9 shows a summary of the accuracies obtained after trying out different combinations of activation functions for each layer. As can be observed, the highest testing accuracy obtained was 91.24%, which was the case of two layers of ELU and Sigmoid for the output layer. It is also important to note that logistic regression gave us better testing accuracy for this problem.

Binary Neural Network Accuracies			
Hidden Layer 1	**Hidden Layer 2**	**Output Layer**	**Accuracy**
ReLU	ReLU	ReLU	90.30%
ReLU	ReLU	Sigmoid	90.30%
Sigmoid	Sigmoid	Sigmoid	91.11%
tanh	tanh	Sigmoid	90.97%
ELU	ELU	Sigmoid	91.24%
SELU	SELU	Sigmoid	90.97%

Table 9 Testing accuracies for binary neural network

For the multinomial version of the neural network, some changes were necessary in order to provide support for all the classes defined. First of all, there were 4 neurons in the output layer, one for each category. In order for the training to work, we also had to one-hot encode **`Y_train`** and **`Y_test`** since there were multiple output neurons. The neural network for this is displayed in **Figure 21**.

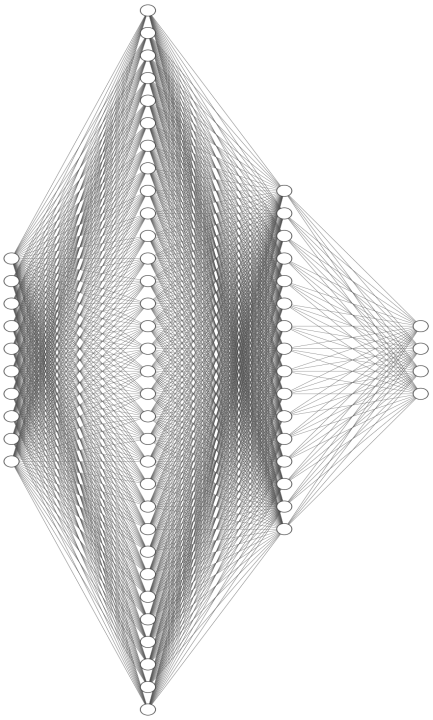


Fig 21. Multinomial model neural network

A summary of the results obtained from the many versions of the multinomial neural network are shown in **Table 10**. Note that most of the activation function combinations tried for this problem gave us a bad test accuracy, and the highest one was 69.54%, which was surprising because there was no softmax layer in that model. The reason why we were trying different softmax layers for multinomial was because softmax is very useful for multi-class classifications. So it was unexpected that only ReLU layers gave us the best accuracy.

To that effect, we decided to also run one last multinomial neural network with only 3 layers in total, and the hidden layer only had 8 neurons. The activation functions used this time were both softmax, and as expected, our accuracy went up to 91.64%. This was the highest score we obtained for the multinomial model, so it reinforced the idea that the data followed a pattern that did not require a lot of feature complexity. This new neural network is shown in **Figure 22**; note that it is much simpler than the ones displayed above.

Multinomial Neural Network Accuracies			
Hidden Layer 1	**Hidden Layer 2**	**Output Layer**	**Accuracy**
ReLU	ReLU	ReLU	69.54%
ReLU	ReLU	Sigmoid	9.84%
ReLU	ReLU	Softmax	10.38%
Sigmoid	Sigmoid	Softmax	9.70%
Softmax	tanh	Softmax	9.97%

Table 10 Testing accuracies for multinomial neural network

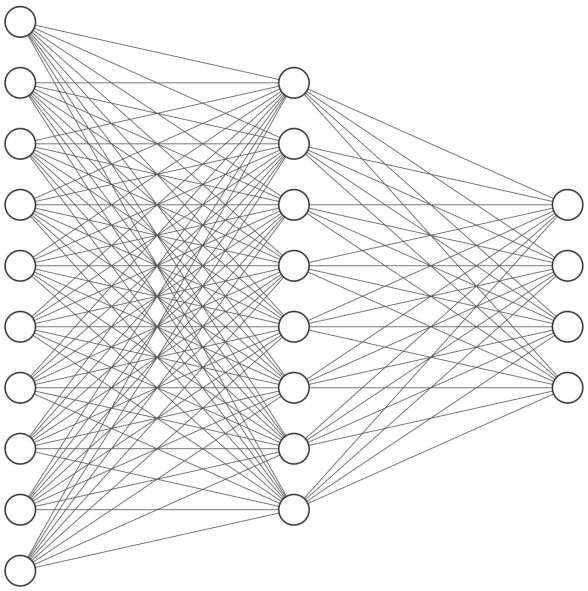


Fig 22. Multinomial model neural network (simplified)

Conclusion

Truth be told, if you wanted to predict if a driver was in the top three or not, it's an easy guess. Random chance suggests that you have a 17/20 (85%) chance of guessing that correctly. Of course, it is more complicated to predict the top three correctly, so we believe our model is significantly better than sheer luck. Our models do *slightly* better than that, making them just a little bit better than a weighted random number generator.

We decided to name our experiment Charles after a driver, who through no fault of his own, ended up in a team capable of making a good car, but failing with strategy. They win purely on chance and seldom on merit. Similarly, our model could be good by sheer chance, and we would not know. Our neural network performs pretty well, far more than 85%, to about 92%. When asked to do a binary classification of podium or not podium, it does even better.

Some of our SVMs are worse than random chance so we concluded that it was better to use different modelling techniques for the binary problem like logistic regression. This is because regression resulted in better accuracies, without the need of so much complexity like the neural networks or the SVMs we tried. For the multinomial model, however, the last neural network trained actually resulted in one of the best test accuracies we obtained for this model. Therefore, it would be a good idea to choose this model to predict drivers' final positions in the future.

There is still, however, more research to be done with neural networks since the simpler one we trained at the end gave us such good results. We believe that it is just a matter of trying many more configurations to find the one that gives us even a higher testing accuracy than 92%. That being said, it has to be a minimalist neural network because, as mentioned above, the data did not seem to follow a very complex pattern.

It was surprising to see how most models we tried showed simple behaviours like linear, quadratic, or cubic because there are so many factors that influence the results of a Formula 1 race, but at the same time this might have been the case because there was a very high probability of making a random guess and getting it right. One of the future steps for our research will definitely be expanding our multinomial model to cover more labels, or even trying linear regression instead of logistic. That might result in a higher error because predicting each specific place is a bigger challenge, but if we can come up with a model that can do that, it will be even more rewarding.

There could also be some collinearities in our dataset, such as those between driver standing and driver points, since the more points a driver has affects their standing positively. The same trend is true for constructors. Perhaps latitude and longitude do not play as large of a role as we are led to believe. Yes, they affect car performance, but they affect *every* car's performance. Take that with a grain of salt since some teams' cars do better in colder climates and some teams' perform better in warmer climates. This is usually the case for worse constructors.

Additionally, our dataset does not track weather events such as rain and dust storms that significantly affect car performance because grip decreases and driver skill matters a lot more than before. With today's meteorological technology, we have accurate weather forecasts down to the minute. So, we can easily predict if a race will be struck with weather. We believe we'd have better accuracy if we could also add weather as a feature. That's not the only feature we wish we could track but could not. We wanted to track driver experience but using the python data library added significant overhead that we decided to forgo.

Finally, to end, we would not recommend using our model for any gambling needs unless you want to bet on a driver not being on a podium. There is a race on Sunday, April 30 and we will try to predict the finishing position of the top 3 for that race using all three models. Unfortunately, since that is after the submission deadline, we will not be able to share the results here but we will add them to the GitHub repository.

Bibliography

- “2014 Formula One World Championship.” Wikipedia, 24 May 2022, en.wikipedia.org/wiki/2014_Formula_One_World_Championship.
- Maini, Vishal, and Samer Sabri. Machine Learning for Humans.
- “Project Jupyter.” Jupyter.org, 2019, jupyter.org.
- Python Software Foundation. “3.7.3 Documentation.” Python.org, 2019, docs.python.org/3/.
- Rao, Rohan. “Formula 1 World Championship (1950 - 2023).” Www.kaggle.com, www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020?resource=download&select=seasons.csv.
- scikit-learn. “Scikit-Learn: Machine Learning in Python.” Scikit-Learn.org, 2019, scikit-learn.org/stable/.