MLE Capstone Report: An Experiment On Predicting Box Office Returns of Films with Classification and Regression Techniques

Thomas Butler

February 2021

Abstract

Machine learning engineering is a significant part of the data science landscape for businesses mature their capabilities and look to deploy models in a production environment. AWS offers a comprehensive suite of products to accomplish this, namely AWS SageMaker. Given the struggle of Hollywood recently, a way of predicting what films will make a profit is as important as ever, if not more.

Contents

1 Introduction **2**
- 1.1 Project Overview 2
- 1.2 Problem Statement 2
- 1.3 Datasets and Inputs 2
- 1.4 Machine Learning vs Rules 3
- 1.5 Evaluation Metrics 3
  - 1.5.1 Classification Metrics 3
  - 1.5.2 Regression Metrics 4

2 Analysis **5**
- 2.1 Exploratory Data Analysis 5
  - 2.1.1 High Level Overview 5
  - 2.1.2 Feature Correlations 5
  - 2.1.3 Feature Distributions 7
  - 2.1.4 Categorical Features 7
- 2.2 Notes on the data 8
- 2.3 Algorithms and Techniques 8
  - 2.3.1 Linear / Logistic Regression 8
  - 2.3.2 Support Vector Machines (SVMs) 8
  - 2.3.3 Decision Trees 9
  - 2.3.4 Random Forests 9
- 2.4 Benchmark Model 9

3 Methodology **11**
- 3.1 Data Preprocessing 11
  - 3.1.1 Handling Categorical Features 11
  - 3.1.2 Train-Test Splits, Validation and Scaling 11
- 3.2 Model Implementation 12
- 3.3 Refinement 12
- 3.4 AWS SageMaker 12

4 Results **14**
- 4.1 Model Evaluation 14
  - 4.1.1 Classification 14
  - 4.1.2 Regression 15
- 4.2 Benchmark Comparison 15
- 4.3 Conclusion 15

Chapter 1

Introduction

1.1 Project Overview

The film industry is in crisis. Global shutdowns due to COVID-19 have had harsh effects on studios, and arguably worse effects on the future of independent cinema. The 2010s had already brought an era of franchises - successful (Disney's MCU), and not (Universal's Dark Universe) - spawning more sequels than ever and causing major studios to buy up IP in the hope of striking gold for their own cinematic universe. The need for studios to know what films are safe investments for bringing profit from the box office has become increasingly important for the industry to survive, let alone thrive. This project will explore how data and machine learning techniques may help.

1.2 Problem Statement

The goal of this project is to train and deploy two machine learning models with AWS SageMaker to predict

1. if a film will make a profit, and
2. what the expected revenue from a film shall be,

A binary classification model and regression model shall be built to answer the first and second requirements respectively. A True Positive will be correctly predicting that a film made a profit.

1.3 Datasets and Inputs

Data shall be taken from the three sources discussed below.

1. **MovieLens Group**

   A list of films up to 2020.

2. **TMDB API**

   An open API for public use on signing up for free account and requesting an API key. It provides additional information that is crowd source (similar to Wikipedia, in their words) to fill in information about films such as their genre and credits.

3. **Federal Reserve Bank of St. Louis**.

   A source of US macroeconomic data that shall be used to match the month of release for each of the films. This addition is motivated by the recognition of economic conditions affecting public spending, e.g., penny-pinching in a recession.

The combination of the MovieLens Group and TMDB API data sources will closely follow the dataset provided on Kaggle by the user *Rounak Banik* [1]. Choosing to go to the Kaggle dataset's source rather than taking it ready-made allows for a greater representation of an end-to-end pipeline for new data being collected for this project and customisation of the data input.

1.4 Machine Learning vs Rules

The problem may appear well suited for a rules-based solution. (No famous actors signed on? Pass. Steven Speilberg wants to direct? Give him the money.) However, machine learning excels in uncovering hidden patterns and signals in complex data that humans would not be able to find in any reasonable amount of time. For example, consider a decision tree (discussed in section 2.3). Decision trees use an algorithm to generate, essentially, a set of rules that may be used in a classification or regression task. Even on huge data, a baseline model can be attained within the hour. Compared to the weeks it would take for a group of human experts to understand the data and write a concise set of rules, the appeal of a machine learning solution quickly becomes apparent.

Machine learning is not only quick to build, but also fast to adapt. "*The pace of change will never be as slow as it is right now*" is a particularly popular proverb of the past few years, so it is fair to assume that the data will display a drift over time. The risk of a performance impact can be curtailed by, for example, monthly training of the model and monitoring of performance on a test set. Environments like AWS make this task - and replacing a poorly performing model with another - relatively easy. Rules, on the other hand, are static and take a lot of effort to change. Again, the experts would be brought in to study the data and understand what new rules need to be written, or what old rules need to be removed. This delayed response to changing data is a significant risk that can be easily mitigated with a machine learning solution.

1.5 Evaluation Metrics

Below we discuss the two types of evaluation metrics that are relevant for this project. First, our attention goes to classification metrics before turning to regression metrics.

1.5.1 Classification Metrics

The first part of the problem is a classification on a binary variable for which four metrics will be considered. These are *accuracy*, *precision*, *recall* and the *F1 score* (the harmonic mean of precision and recall). These metrics are defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{1.1}$$

$$Precision = \frac{TP}{TP + FP}, \tag{1.2}$$

$$Recall = \frac{TP}{TP + FN} \quad \text{and} \tag{1.3}$$

$$F1 = \frac{TP}{TP + \frac{1}{2}\big(FP + FN\big)}, \tag{1.4}$$

where TP if the number of True Positives, FP is the number of False Positives, TN is the number of True Negatives and FN is the number of False Negatives. Each of the metrics have their merits. Accuracy is an overall score looking at how well the model correctly predicted the classes. This could become inflated if there is a strong imbalance in classes. For example, let's say 99% of the films make a profit. Then, the model would achieve a 99% guaranteed accuracy by predicting that *all* films will make a profit.

Precision penalises a high number of False Positives; recall penalises a high number of False Negatives. The former would be a problem for the hypothetical studio executives if they made a decision on the model, pumping money into films that turned out to bomb. On the contrary, the latter would cause executives to miss out on profit (although the absence of money is not easily measurable). The F1 score, as mentioned above, is the harmonic mean of precision and recall, useful for when one seeks a balance between the two.

In this business context, the number of False Positives (incorrectly stating that a film will make a profit) is the figure to be minimised on assumption that this is the worst case scenario. Therefore precision shall be used as the primary metric for the classification part of the problem. The other metrics shall also be monitored throughout to ensure precision is not maximised to the detriment of other performance measures.

1.5.2 Regression Metrics

The regression model invites consideration of several metrics. The four metrics in consideration are the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R^2, and Adjusted-R^2. These metrics are defined as

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|, \tag{1.5}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \quad \text{and} \tag{1.6}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}^2)}, \tag{1.7}$$

$$\text{Adjusted-}R^2 = 1 - \frac{(1 - R^2)(n-1)}{n-k-1} \tag{1.8}$$

where n is the number of observations, y is the true value, $\hat{y}$ is the predicted value, $\bar{y}$ is the mean of the target variable and k is the number if independent variables included in the model.

MAE gives a linear response on the error, whereas RMSE penalises larger deviations from the mean. Both are simple metrics good for monitoring the model performance with respect to errors.

R^2 can be interpreted as the amount of variability in the response explained by the model. Consequently, adding more features will add to the variance in the data captured. Adjusted-R^2 resolves this behaviour of rewarding models with a high number of features by adding a penalty to every additional feature that does not add to the captured variance.

RMSE shall be used for the sake of comparison to the benchmark model, supplemented by the R^2 and Adjusted-R^2 measures to monitor to variance of the data captured by the model.

Chapter 2

Analysis

2.1 Exploratory Data Analysis

This section covers the exploratory data analysis to give a better idea of what is in the data and inform decisions for the machine learning pipeline.

2.1.1 High Level Overview

Some basic features of the data may be noted here. There are 5,579 instances in the data. For the classification part of the project, only two classes will be considered: 1 for if the film makes a profit (3,894 instances), and 0 if it did not (1,685), giving an approximate 70:30 split respectively. This split means that the problem shall be an imbalanced classification problem. The sampling will be stratified on the class to make sure the training set will be representative of the population.

A quick overview and summary of numerical features can be returned with Panda's `describe()` method on a DataFrame. This revealed some actionable insights.

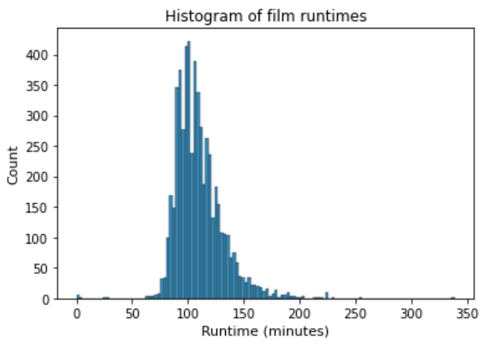
- **Missing run-time data**: The minimum value for runtime showed as zero. Figure 2.1a shows the distribution of the runtime feature across all films. The outliers indicating a zero-minutes time were treated as missing values and imputed with the median runtime.
- **Unrealistic budget and revenue entries**: An unrealistically low budget value prompted a look at the budget and revenue distributions. Figure 2.1b shows the logarithmic distributions of the budget and revenue. Both have a strong left skew, where further investigation revealed unrealistic entries for these outliers. For example, a budget lower than \$100. To remove the skew in the distributions, the outliers were removed. The lower bound for revenue to keep was set at 10^4 (\$10,000), removing the lowest 1.20%. For revenue, the lower bound was set at 10^5, removing a further 1.05% of instances.

2.1.2 Feature Correlations

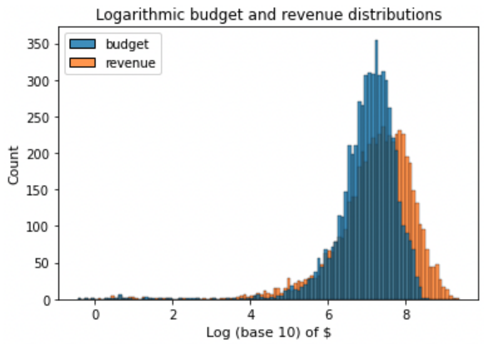
Generally, correlated features do not improve machine learning models. Including strongly correlated predictors is redundant as no significant amount of information (variance) is gained while adding unnecessary complexity [5]. Which feature is dropped given the choice of two is irrelevant. If one were more convenient to gather than the other, then the decision could be made on convenience. Here we have no such constraints.

Figure 2.2 shows a heatmap of correlations between the numerical features in the data. A bolder red (blue) indicates a strong positive (negative) correlation. Grey indicates near-zero correlation between features. The diagonal has a perfect positive correlation as it shows features compared to themselves. We can see a cluster of high-correlation for the actor data-points. The features `sum_actor_pop` and `avg_actor_pop` (for the sum and average of the top three cast members' popularity respectively) have a perfect linear relationship, and so one can be dropped.

Year also has a strong correlation with both `PCE` and `CPIAUCSL`, two of the macroeconomic features sourced for this project. Year can be dropped here.



(a)



(b)

Figure 2.1: (a) Histogram of the runtimes for all of the films. There are outliers at zero that are treated as missing values, imputed with the median runtime. (b) Distributions of budget and revenue with $x \to log_{10}x$ applied.

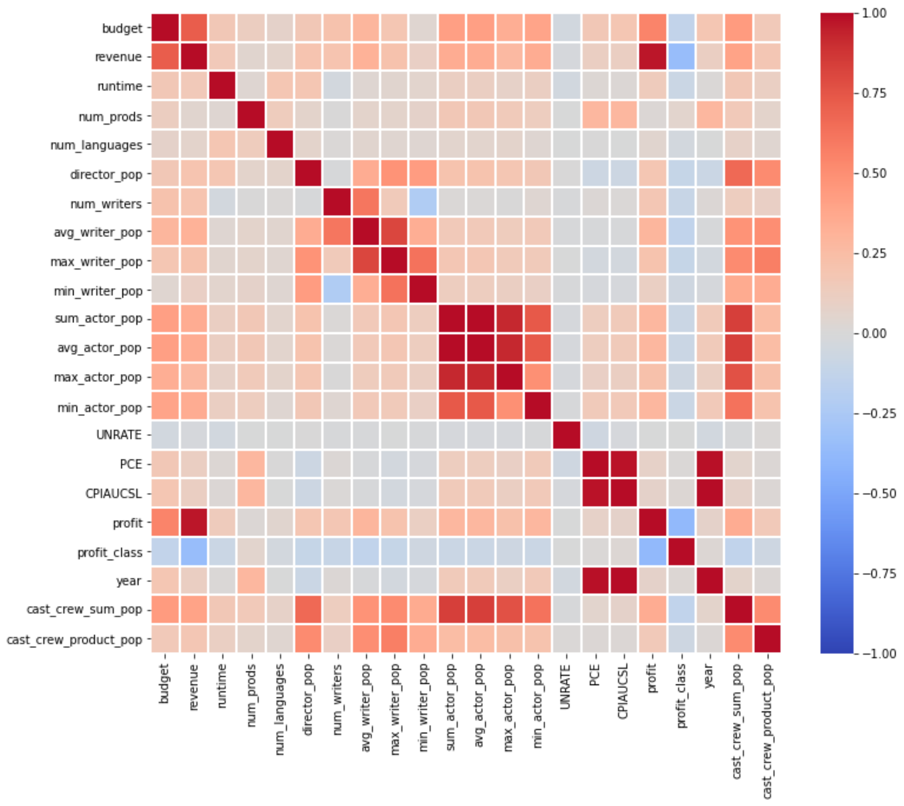


Figure 2.2: Figure showing a heatmap of correlations between the numerical features. Bolder red (blue) indicates a stronger positive (negative) correlation. Grey indicates no correlation with other variables.

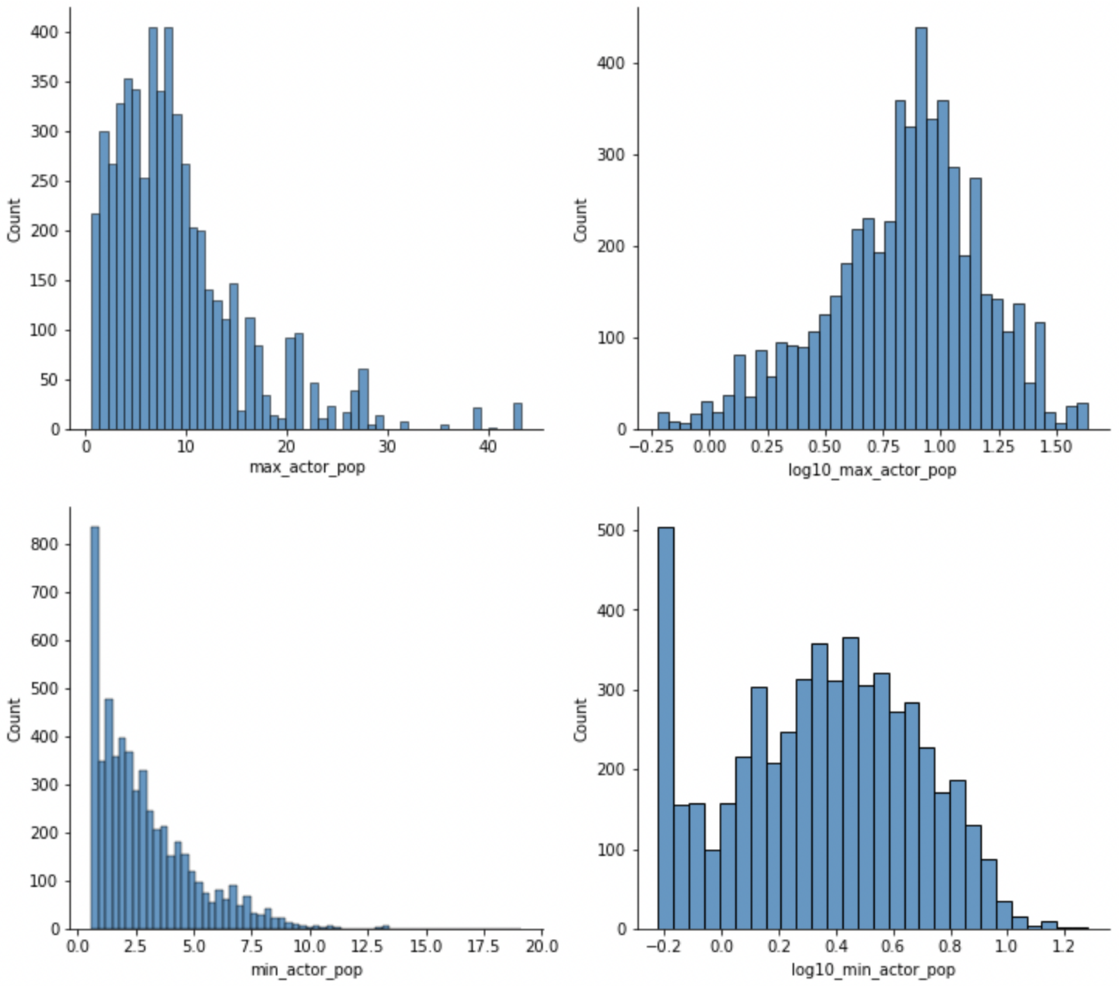


Figure 2.3: Examples of two features and how their distributions are effected by undergoing a transformation of $x \rightarrow \log_{10} x$. The right hand side is closer to a normal distribution, and so is preferred for the machine learning model.

2.1.3 Feature Distributions

Handling a skewed feature is critical for a machine learning exercise as models can be adversely affected by outliers. Section 2.1.1 explored the distributions of only three of the numerical features in the data. We saw that applying a logarithm to budget and revenue reduced the heavy skew, bringing the data to look like a more typical Gaussian distribution. The same was then applied to the other numerical features. Figure 2.3 shows how some of the features were transformed.

2.1.4 Categorical Features

Some of the categorical columns lent themselves to an obvious exploration. For example, encoding all of the genres or months so there is a flag for each of them. However, the majority of the columns had far too many unique values to do this. For example making a flag for every unique director would have introduced over 2,000 extra columns to the data (and even more for actors). The columns that had this problem were:

- `original_language`, the original language that the film was released in,
- `prod_compy_cntry`, the countries for the companies that produced the film,
- `prod_comp_names`, the names of the production companies for the film,
- `writers`, the writers on the film,

- `actors`, the top three billed actors on the film, and
- `director`, the director of the film.

2.2 Notes on the data

Using "popularity" features

Each film had a 'popularity' feature that is only possible to know after a the release of a film, and so it was dropped. However, analogous features on director, writer and actor popularity will be retained. These are measures as at the building of the data during this project. It is assumed that popularity is stationary in time in these cases, a simplification to account for the lack of time-sensitive data.

Accounting for inflation

The budget and revenue data has not been adjusted to account for inflation. This is why the Consumer Price Index (CPI) data was introduced. Using CPI, we can adjust the budget and revenue into real dollars (for today) with the equation

$$x_{t_f} = x_{t_f} \frac{CPI_{t_f}}{CPI_{t_s}} \tag{2.1}$$

where x_{t_f} is the dollar amount now, x_{t_s} is the dollar amount at the release date, CPI_{t_f} is the Consumer Price Index now and CPI_{t_s} is the Consumer Price Index at the release date.

2.3 Algorithms and Techniques

Several machine learning algorithms will be explored for this project. The four discussed below each have their merits, hence their consideration. All are discussed with more of a focus on classification, yet each are capable of a regression task.

2.3.1 Linear / Logistic Regression

Linear and logistic regression is one of the simplest machine learning algorithms. It assumes that the target variable can be modelled by a linear combination of the features, represented by the equation below,

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n, \tag{2.2}$$

where $\hat{y}$ is the predicted value, n is the number of features, x_i is the i^{th} feature value and θ_j is the j^{th} model parameter (including bias term θ_0 and features weights) [8].

Linear and logistic regressions are typically very easy to implement, interpret, and train. For example, the largest feature weights can be extracted and presented as a crude representation of feature importance, assuming that all input variables have been scaled prior to training the model.

However, the shortcomings of linear regression are in the name - assumed linearity. Seldom is a dataset linearly separable, and complex higher-order relationships are not detected without further feature engineering. Adding higher order interactions between features would also increase the computational cost of the optimisation in training the model.

2.3.2 Support Vector Machines (SVMs)

Support Vector Machines are a versatile and powerful class of machine learning algorithms. They are capable of linear or non-linear classification, regression and even outlier detection [3]. Linear SVMs are the easiest to visualise, with the idea being captured in Figure 2.4. The linear class boundary line (solid line) is chosen to maximise the width of the "road" that is formed with the two support vectors (dashed lines).

However, the problem with a linear SVM is the same as for logistic regression - many datasets are far from linearly separable. The problem of computational complexity with transforming the data (say, with a second-degree polynomial) is resolved using *kernelized SVMs*.

A full explanation of how using a kernel works is outside the scope of this report. Yet a simplified account shall be given. Consider the case where you wish to apply a transformation, ϕ, to your data,

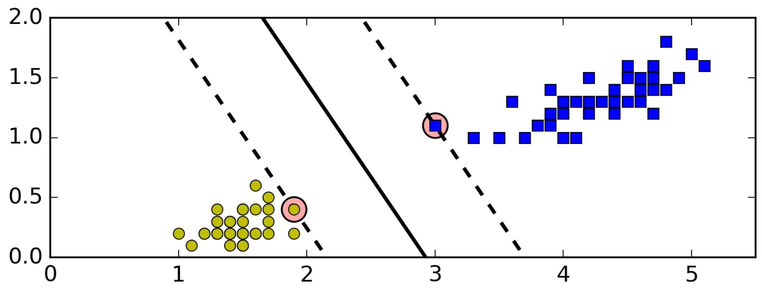


Figure 2.4: The intuitive idea of Support Vector Machines (SVMs) [3]. The separating line is chose to maximise the width of the "road" that is formed. The support vectors (dashed lines) are those going through the closest instance of each class to the separating line.

a and **b**, to explore higher order interactions between data points. Here, you may use an appropriate *kernel*. In machine learning, a kernel is any function capable of calculating the dot product $\phi(\mathbf{a})^T\phi(\mathbf{b})$ using only **a** and **b**, without having to compute the transformation ϕ [2].

This dot product calculation is an important step in the optimisation of an SVM. Therefore, the computational cost of training the algorithm is greatly reduced by not having to transform every data point. This allows non-linear relationships in the data to be explored with relative ease. Such a benefit makes an SVM a great choice for many machine learning problems, especially when the data is clearly not linearly separable.

2.3.3 Decision Trees

Decision trees, like SVMs, are versatile machine learning algorithms that can perform both classification and regression tasks. They are appealing if we care about interpretability, as a decision tree can be seen as a large set of generated rules [6]. The decisions at each node of the tree can be returned to give a very simple account of the reason for any prediction.

Generally, the main issue with decisions trees is a high sensitivity to small variations in the training data. Unlike linear models, decision trees make very few assumptions about the training data. For example, letting a decision tree go as deep as possible to fit to the training data will risk overfitting. This can be managed by setting a maximum depth to the tree, improving the chances of the model generalising better to test or validation data.

2.3.4 Random Forests

One of the most powerful machine learning available today, a Random Forest is a surprisingly simple idea. Start by training a bunch of decision tree classifiers, each on a different random subset of the data. Make predictions using all of the individual trees, and take the most common class prediction from the set as the final prediction. An appropriate analogy is an election. Pose a complicated question (e.g., who should be leader?) to many voters (predictors; classifiers) and elect the most common answer. In machine learning, this is referred to as an *ensemble method*.

The ensemble method of a random forest often achieves a higher performance than the best decision tree classifier in the ensemble. This may seem counter-intuitive - after all, how can weak models add to better ones when they are trained on the same data? Indeed, this is a problem when the predictors are correlated. Two ways to get a diverse set of classifiers and avoid this problem are to: (a) train classifiers that use very different algorithms, or (b) train the same algorithm on different random subsets of the data [3]. The random forest uses the latter technique to increase the diversity from its classifiers.

However, random forests are viewed as black-box models where they are difficult to interpret. Each tree retains its simplicity in explaining a decision, yet the amalgamation of many trees introduces too much complexity to provide a simple explanation of why a prediction is made [3].

2.4 Benchmark Model

The benchmark model is set in reference to a May 2019 paper [4] in which the reported results were as follows:

- $Accuracy = 0.83$ on the test data with a Random Forest classifier for the classification of films into profit-/loss-makers, and
- $RMSE = 0.5$ on the test data with an *rpart* regression model (an R package for decision trees) for predicting film revenue.

Chapter 3

Methodology

A lot of the preliminary development was carried out locally. This included tasks detailed in the preprocessing section (data cleaning; feature engineering) and the implementation section (early model training; hyper-tuning). Code was later moved to AWS SageMaker to build an infrastructure for deploying a trained model to an endpoint where data may be sent for inferences.

This chapter will discuss the data preprocessing, model implementations, and refinement (i.e., tuning) of the model. Then the SageMaker implementation is discussed.

3.1 Data Preprocessing

Some columns were outright dropped. These were `id`, `tmdbId`, `original_title`, `popularity`, `year`, `release_date`. They were dropped either because they were redundant (e.g., year), never of any use in a machine learning algorithm (e.g., original title), or could not possibly be known ahead of release (popularity). The exploratory data analysis revealed key actions to be taken in the preprocessing of the data. These are discussed below.

3.1.1 Handling Categorical Features

As discussed in the EDA section, some of the categories had thousands of unique entries. Encoding all unique values into features would explode the number of columns in the dataset. This would clash with the assumption in machine learning that $n \gg m$, i.e., the number of rows is much greater than the number of columns. Having thousands of columns would also lead to a very sparse dataset, which introduces more problems for machine learning algorithms. Such columns had to be handled in a different way.

The general steps were:

1. Create a column per unique value.
2. Find the frequency with which these unique values occur. This was easily done by summing each new column.
3. Find this sum as a fraction of the total row count.
4. If this was below 2% then the column was dropped

This handled the sparsity and column count problems. However, to retain some of the information from the directors and actors, additional features were created. These were `established_director` and `num_top_100_actors`. The former is a flag for it the director of the film was in the top 100 most frequent directors among all films. The latter is the number actors in a film who were among the top 100 most frequent actors in the data.

3.1.2 Train-Test Splits, Validation and Scaling

The data was split 80:20 into train (`X_train`) and test (`X_test`) sets respectively. Stratified sampling based on the profit/loss class was used to best represent the population. The data was then scaled using SciKit-Learn's `MinMaxScaler`.

3.2 Model Implementation

For the classifier problem, the revenue was dropped from the data as keeping it would give leakage in the model. For the regression problem, the class label was removed. The target variable in the regression task was chosen to be $\log_{10}$ of the revenue given the findings in the EDA. Prediction $\hat{y}$ is transformed back into dollars using $10^{\hat{y}}$. The models discussed in section 2.3 were trained on `X_train` with their default parameters. This gave a baseline to look at what models to take forward for tuning.

K-fold cross-validation was used in training the models with K=10. For precision score was returned for the classification problem and the $RMSE$ was returned for the regression task. Doing this gave a better idea of the model sensitivity to input data. For example, a high standard deviation in the ten scores would indicate that the model was sensitive to whichever random sample it was training on. With this in mind, it also gave a slight measure of reliability on deciding what models to take forward for further tuning.

3.3 Refinement

Model tuning was achieved with SciKit-Learn's `GridSearchCV` class. The SciKit-Learn documentation was used to explore what parameters could be defined in a grid to search over. For example, classifiers can have the class weights defined so that one class will get a stronger weight than the other.

An example parameter grid for the random forest classifier is given below.

```
param_grid = {
    'max_features': ['auto', 'sqrt', 'log2']
    , 'max_depth' : [8,10,12,16,20]
    , 'criterion' :['gini', 'entropy']
    , 'class_weight': ['balanced', {0:1, 1:2}, {0:1, 1:3}, {0:1, 1:4}
                      , {0:1, 1:5}, {0:1, 1:6}, {0:1, 1:7}, {0:1, 1:8}]
}
```

3.4 AWS SageMaker

The most significant part of the Machine Learning Nanodegree is learning how to use the AWS ecosystem to deploy machine learning models in a manner that allows robust monitoring, security and control of the model. Time constraints meant that the web app mentioned in the project proposal was not developed. However, the model was still implemented in AWS SageMaker.

The SageMaker Python SDK can be used to train and deploy customer SciKit-Learn code. This package has a SciKit-Learn estimator class - a cornerstone of model training in SageMaker - that can be easily configured to train and deploy models with specific infrastructure requirements [7]. For example, you may specify an instance type and count. In our case, we will use a single low-powered instance. A higher count of instances (and more powerful ones) would be used for distributed ML jobs on huge datasets.

In the estimator object, we have to specify the locations of the data and the training script. The former will be the location on the S3 bucket that SageMaker notebook instances come linked to. The latter is a script written to load data, parse any arguments (e.g., hyperparameters) and train the SciKit-Learn model. Example code for doing this is given below,

```
estimator = SKLearn(
    entry_point='train.py'
    , py_version='py3'
    , framework_version='0.23-1'
    , role=role
    , instance_count=1
    , instance_type='ml.m4.xlarge'
    , sagemaker_session=sagemaker_session
    , hyperparameters={
        'class_weight_0': 1
        , 'class_weight_1': 6
        , 'criterion': 'entropy'
        , 'max_features': 'auto'
    }
)
```

where the class weights, criterion and maximum features of a decision tree have been specified and sent as hyperparameters.

Deploying an endpoint for the trained model is as simple as using the following code:

```
predictor = estimator.deploy(
    initial_instance_count=1
    , instance_type='ml.t2.medium'
)
```

This predictor object may then be used for returning inferences on new data. This would be done with `predictor.predict(data=X_test)`. The notebook `04-SageMaker` contains the code for deploying the model and returning inference from the endpoint.

Chapter 4

Results

This chapter discusses the results of the project. Final model parameters are detailed, with some analysis of the robustness of the models. A comparison to both benchmarks is discussed before assessing whether the problem has been adequately solved.

4.1 Model Evaluation

4.1.1 Classification

Table 4.1 shows the performance metrics for the models with default parameters for the classification problem. We can see that none of the default models match the benchmark accuracy performance. However, a strong precision of 0.8360 is found with the SVM classifier. This is a reliable result as the K-fold cross-validation found the precision to be 0.8226 ± 0.0188. That is, the precision on the test data is within one standard deviation of the mean precision on the cross-validation.

The random forest classifier had a precision of 0.7312 ± 0.0061 from the cross-validation. Yet it had a very strong F1 score compared to the other models. It is notable that the random forest far outperformed the lone decision tree. This makes sense given the discussion on the random forest ensemble method in 2.3.

The random forest and SVM classifiers were taken forward for parameter tuning with SciKit-Learn's GridSearch class. After tuning, the best model against the precision metric was the SVM with 0.8360. However, on balance across the metrics scorecard, the random forest was chosen as the best model.

The final model parameters of the random forest classifier were

```
'class_weight': 'balanced'
'criterion': 'entropy'
'max_depth': 8
'max_features': 'log2'
```

with final metrics of

- Accuracy = 0.6673
- Precision = 0.7899
- Recall = 0.7167
- F1-score = 0.7515.

Model (Clf.)	Accuracy	Precision	Recall	F1
Benchmark	0.83	-	-	-
Logistic	0.6966	0.7896	0.7742	0.7818
SVM	0.6508	0.8360	0.6253	0.7155
Decision Tree	0.5234	0.7150	0.5339	0.6114
Random Forest	0.7131	0.7227	0.9565	0.8245

Table 4.1: Metric performance of each of the models at their initial development. Random Forest and SVM were taken forward for further tuning.

Model (Reg.)	RMSE	R^2
Benchmark	0.5	-
Linear	Unstable	Unstable
SVM	0.6189	0.5008
Decision Tree	0.9459	-0.1659
Random Forest	0.6347	0.4750

Table 4.2: Metric performance of each of the models at their initial development. Both the random forest and SVM were taken forward for further tuning, just as for the classification task.

4.1.2 Regression

Table 4.2 shows the scores on the regression problem. Using the linear regression model to predict led to wildly unstable results, with a RMSE on the scale of $\mathcal{O}(10^{11})$. Both of the SVM and random forest models stood out again as the best performing. Without tuning, neither have outperformed the benchmark model.

These models had stable RMSE scores from the cross-validation. The SVM had an RMSE 0.6027 ± 0.0410, and the random forest had an RMSE 0.6063 ± 0.0432 with ten folds. The values shown in the table are within one standard deviation of the mean scores.

On tuning, neither the random forest nor SVM improved their scores. Rather, the SVM with the default parameters remained the best model for the task.

4.2 Benchmark Comparison

That the result for the classification problem ended with a disappointing accuracy score of 0.6673 compared to the benchmark of 0.83 indicates realities of data science. A promising hypothesis may come to be just that - an hypothesis that proves false on experiment. However, the benchmark model did not record the precision (nor recall and F1 scores) which in section 1.5 were discussed as more important in this context.

The regression solution did not manage to surpass the benchmark either, nor improve upon. Tuning did not improve the results for either the random forest of SVM regressor. Rather, the SVM with the default parameters performed best for the regression task.

4.3 Conclusion

Ultimately, a machine learning algorithm may only be as good as the data it is being trained on. It is indeed possible to train a good model on only a few thousand rows of data. However, the scope of the data was limited to the films where users of TMDB had entered a sufficient amount of information. Therefore, it will already have been somewhat focused on popular and/or successful films. Obscure projects that failed to earn a profit or much of an audience would have had missing data. This held back the diversity in the data, and may have contributed to the imbalance in in the class instances.

...

The practise of deploying a machine learning model into the AWS environment and integrating it with a web app gave the author valuable experience. Future iterations would look to diversify the data sources, ideally increasing the row count. A graph based approach for engineering features based on relationships between actors, directors, production companies (among other entities) would be explored. Neo4J would be used as the graph database solution given, in particular due to its built-in data science algorithms.

Bibliography

[1] Rounak Banik. *The Movies Dataset*. 2018. URL: `https://www.kaggle.com/rounakbanik/the-movies-dataset`.

[2] N. Deng, Y. Tian, and C. Zhang. *Support Vector Machines*. Chapman and Hall/CRC, 2012. ISBN: 9781439857939.

[3] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd ed. Packt Publishing, 2019. ISBN: 9781492032649.

[4] Sarah E. Joseph. "What Makes a Movie Successful : Using Analytics to Study Box Office Hits". In: (2019). URL: `https://trace.tennessee.edu/utk_chanhonoproj/2252`.

[5] O. Martin. *Bayesian Analysis with Python*. 2nd ed. Packt Publishing, 2018. ISBN: 9781789341652.

[6] S. Raschka and V. Mirjalili. *Python Machine Learning*. Packt Publishing, 2017. ISBN: 9781787125933.

[7] J. Simon and F. Pochetti. *Learn Amazon SageMaker*. Packt Publishing, 2020. ISBN: 9781800208919.

[8] G.G. Vining, E.A. Peck, and D.C. Montgomery. *Introduction to Linear Regression Analysis*. 5th ed. Wiley, 2012. ISBN: 9780470542811.