AWS Computation Report

MA5852 Assessment 2
13848336 Nikki Fitzherbert
31 May 2021

Diabetes is a complex and serious medical issue that has been growing in prevalence across time, which has also led to a concurrent increase in treatment expenditure. One of the drivers behind the concurrent increase in expenditure has been the rate of hospital readmission. It is therefore becoming increasingly important for hospitals to be able to predict which patients are most likely to be readmitted. This report used the Center for Machine Learning and Intelligent Systems hospital readmissions dataset to build and train a simple multi-layer perceptron model in Amazon SageMaker and investigate the impact of several regularisation techniques on model performance. However, whilst performance was fairly consistent across a range of different metrics, the model was unable to match the performance in much of the reviewed literature.

Contents

Introduction

Diabetes mellitus, commonly referred to as diabetes, is a serious and complex condition in which the human body is unable to maintain healthy levels of glucose in the blood. This is turn can lead to both short- and long-term health complications, including blindness, kidney failure, heart attack, limp amputations and adverse impacts on mental health (diabetes Australia, n.d.). Furthermore, diabetes is counted as one of the globe's fastest growing health problems, with the number of adults diagnosed with diabetes having tripled over the past twenty years to 463 million people in 2019 (International Diabetes Federation, 2019).

This health issue also has had an economic cost. The International Diabetes Federation (IDF) has estimated the current overall health expenditure for diabetes at USD 760 billion, which could increase to USD 825 billion by 2030 and USD 845 billion by 2045 (International Diabetes Federation, 2019). One of the reasons for this rise in healthcare expenditure has been the observed increase in the number of diabetes-related hospital inpatient readmissions, where hospital readmission refers to a patient being readmitted to a hospital within a certain time interval after being discharged. (D. Sathyavathi & Sowjanya, 2020). Moreover, since hospitals in the United States of America are financially penalised when 30-day readmissions rise above the permitted rate, the readmission rate has become a measure of a hospital's performance and quality of care, and a means by which to reduce healthcare costs (CMS.gov, 2020; D. Sathyavathi & Sowjanya, 2020).

There have been many studies attempting to accurately predict hospital readmission of patients diagnosed with diabetes and/or identifying the major risk factors for hospital readmission. For example, Strack et al. (2014) examined the relationship between Hb1A1c measurement and hospital readmission rates, Bhuvan et al. (2016) investigated the issue of hospital readmissions from several different angles, and Hammoudeh et al. (2018), Goudjerkan and Jayabalan (2019) and Shankar G and Manikandan (2019) all used neural networks in their research.

The current study will use the pre-existing body of work to build and assess the performance of a simple multi-layer perceptron neural network attempting to predict whether or not a diabetic patient will be readmitted at any time in the future. It will also investigate the effect on model performance of including different types of techniques such regularization, early stopping and batch normalization. The study will conclude with a discussion with some of the considerations regarding the use of Amazon SageMaker to build, train and evaluate the model.

Task 1 – Data Preparation

Part A – Feature selection

The study used the preliminary dataset extracted by Strack et al. (2014) from the Health Facts database, representing 10 years of clinical care at 130 hospitals and integrated delivery networks in the United States of America. It consisted of 101,766 encounters across a mix of 50 numeric and categorical features that satisfied all five of the following requirements:

1. It was a hospital admission.
2. It was a diabetic encounter; that is, diabetes was listed as a diagnosis.
3. The length of stay was between one day and two weeks.
4. Laboratory tests were performed during the encounter.
5. Medications were administered during the encounter.

The types of features included in the source dataset included patient demographics, diagnoses, payer information, number of in-, out- and emergency visits, and other medical-type data (further detail can be found at Appendix B).

As many features as possible were retained as the identity of the most important risk factors for hospital readmission appeared to depend somewhat on the type of model used and/or the types of feature engineering applied. In this study, the decision on whether or not to retain a particular feature was guided by the reviewed literature and traditional statistical approaches to things such as missing values and identifier variables.

As a result, a total of 26 variables were removed from the source dataset. Weight and payer code were both dropped due to the presence of a large proportion of missing values (96.9% and 49.1% respectively); encounter id and patient number were dropped as they were identifier variables and therefore would make no useful contribution to the model; and all medication variables were removed except for insulin as it was observed just about all had the same value for over 95% of records. Variables such as these typically do not make a significant contribution to a machine learning model (Hammoudeh et al., 2018).

Part B - Data cleaning and transformation

Duplicate records

Prior to dropping encounter id and patient number, it was observed that there were multiple records for the same patient. In order to ensure that the observations were statistically independent, only the first encounter for each patient was retained. As this also had the highest probability of being labelled as a readmission, it would also help with the data imbalance issue (discussed in more detail in Part C).

Missing values
During initial exploration of the dataset, it was observed that missing values were identified by the '?' symbol, which is a non-standard approach. Therefore, all such values were replaced to more readily identify such missing values in the dataset.

Cardinality reduction
Consistent with Strack et al. (2014), medical specialty was retained as a feature. Missing values were replaced by the string 'Unknown' and feature cardinality reduced in two steps. First, variations on the same specialty were labelled the same; for example, anything including the word 'surgery' was relabelled as 'surgery'. Secondly, the categories were ordered by number of records, the top seven categories were retained as the number of records in subsequent categories dropped off significantly after that, and all other records were labelled as 'Other'.

A similar approach was used for the primary diagnosis. The analysis used the approach taken by Strack et al. (2014) in which all records were labelled according to the associated diagnosis ICD9 group name, and groups with less than 3.5% of total encounters labelled as 'Other'. The secondary and third diagnoses variable records were then re-labelled with those same category names.

Feature encoding
Age was converted to a numerical discrete datatype by replacing each category with its midpoint value, consistent with the approach taken by some (for example, Goudjerkan and Jayabalan (2019)).

Similarly, the output class was reduced to a binary classification problem, by combining all readmitted records into a single class (that is, all records that had a readmitted value of '>30' or '<30'). This meant that the outcome class was much more balanced, with the minority class making up 41% of total records.

Feature creation
To account for the loss of information from the removal of the medication variables, a feature recording the number of changes in medications, and a feature recording the number of medications given were creation. In addition, a feature called service utilisation, which measured the total number of inpatient, outpatient and emergency visits was also added. Previous literature indicated that at least two of these were influential in determining whether or not a patient would be readmitted (Sarthak et al., 2020).

Record deletion
Any remaining records containing missing values were removed, as it was determined that this would likely not adversely affect the dataset due to their small number. It was also determined that it would be appropriate to remove any records with a discharge disposition id relating to a hospice (end-of-life facility) or death as such patients would never be readmitted.

One-hot encoding
Neural network models require all features to be of a numerical datatype. As such, all categorical variables were re-encoded into a series of binary variables. This added an additional 90 features to the dataset.

Standardisation
The values of numeric features were transformed using either a log(x+1) or sqrt(x+1) transformation to address the skewness present in many of the distributions (Appendix C). No transformation was applied to the number of lab procedures or age as the application of one increased, rather than decreased the level of skewness.

The Amazon SageMaker Jupyter notebook code used to pre-process the data can be found at Appendix D.

Part C - Training and test datasets

The pre-processed dataset consisted of 68,054 observations and 103 variables, including the target variable. It was split into a training, validation and test set using a 70:15:15 ratio, in which 70% was used for model training, 15% for model tuning and the final 15% to gauge performance on completely unseen data.

As the prevalence of readmitted to not readmitted observations was only moderately unbalanced (a 4:6 ratio), it was not deemed necessary to rebalance the dataset using over- or under-sampling techniques such as Synthetic Minority Oversampling (SMOTE). However, the slight imbalance was noted as standard evaluation metrics can provide misleading indicators of performance due to an inability to accurately classify members of the minority class.

Table 1 presents a summary of the various sample sizes by outcome class.

Table 1: Training, validation and test dataset samples sizes

Outcome class	All data	Training set	Validation set	Testing set
0	40,131 (59.0%)	28,098 (59.0%)	6,025 (59.0%)	6,008 (58.9%)
1	27,923 (41.0%)	19,540 (41.0%)	4,183 (41.0%)	4,200 (41.1%)
Total	68,054	47,638	10,208	10,208

The Amazon SageMaker Jupyter notebook code used to split the dataset into the training, validation and test sets and standardise numeric features can also be found at Appendix D.

Task 2 – Building, Training and Deploying a Neural Network

Part A – Proposed model

The baseline multi-layer perceptron neural network (a feed-forward artificial neural network) consisted of one input layer, with an input dimension equal to the number of features in the dataset (102), two fully-connected hidden layers with 8 and 4 nodes respectively, and an output layer making a binary prediction.

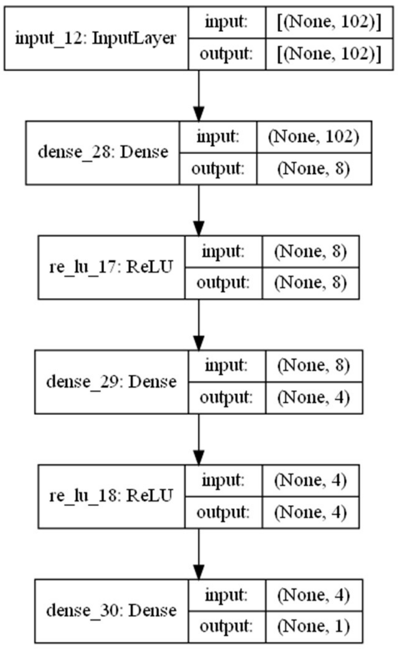
The structure of the proposed model was based on that used by (Bhuvan et al., 2016); however, a second layer was added as loss function performance appear somewhat better when testing out initial configurations. The number of nodes in the second layer was set to half the number of nodes in the first (Keim, 2020).

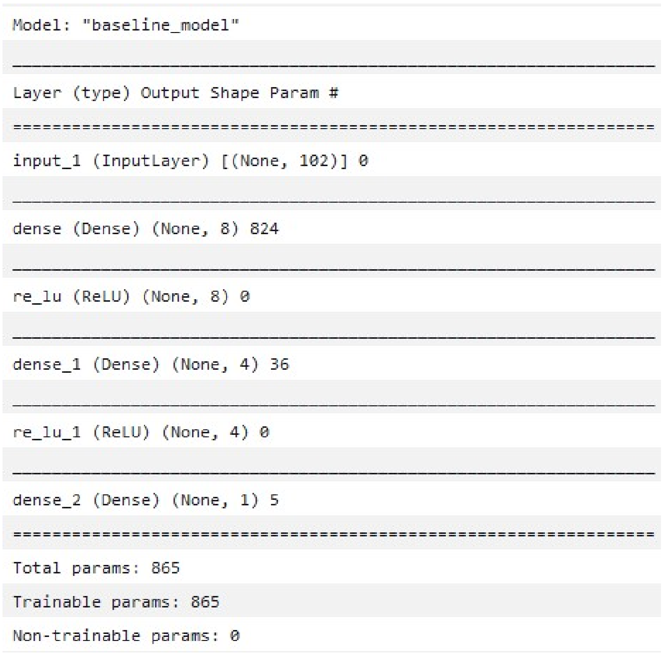
Each of the hidden layers were activated with a Rectified Linear Unit (ReLU) function and the output layer used a sigmoid function. The latter was selected given that this was a classification problem and the former given its use by several relevant papers and current popularity with respect to building neural networks (Brownlee, 2020).

Finally, the default 'Glorot uniform' (also known as the 'Xavier uniform') initialiser was used to set the initial random weights of both hidden layers.

At this stage, no additional features such as early stopping, dropout or regularisation were incorporated. However, the impact of these on the performance of the baseline model were investigated and the results are discussed in part D of the report. A graphical depiction of the baseline model structure, along with a model summary is presented in Figure 1. The latter shows that the total number of trainable parameters for this model was 865, with 95% located in the first hidden layer, and the majority of the rest located in the second hidden layer.

Figure 1: Baseline neural network model structure





```
Model: "baseline_model"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
input_1 (InputLayer) [(None, 102)] 0
_________________________________________________________________
dense (Dense) (None, 8) 824
_________________________________________________________________
re_lu (ReLU) (None, 8) 0
_________________________________________________________________
dense_1 (Dense) (None, 4) 36
_________________________________________________________________
re_lu_1 (ReLU) (None, 4) 0
_________________________________________________________________
dense_2 (Dense) (None, 1) 5
=================================================================
Total params: 865
Trainable params: 865
Non-trainable params: 0
```

Finally, the model was compiled with a binary cross entropy loss function and an Adaptive Moment Estimation (Adam) optimiser. The former was chosen given this was a binary classification problem and the latter chosen as it is a powerful, general-purpose optimiser that is also computationally efficient and not particularly memory-intensive (Keras, n.d.).

Part B – Building and training the model in AWS SageMaker

The Amazon SageMaker Jupyter notebook code and Python script used to train and build the baseline neural network model are located at Appendix E and Appendix F respectively. Screenshots of the process are located at Appendix G.

The values of several model hyperparameters were set during the model design stage, and these are described in more detail in Table 2:

Table 2: Hyperparameters set during the model build

Hyperparameter	Value	Reasoning
Number of hidden layers	2	Trial-and-error. Better initial training and validation loss.
Output layer activation function	Sigmoid	Appropriate for a binary classification problem.
Loss function	Binary cross entropy	Appropriate for a binary classification problem.
Optimiser	Adam	Powerful, general-purpose optimiser.

There were also a number of hyperparameters that had values chosen for the baseline model, but were subsequently tuned by maximising the validation Area Under the Precision-Recall Curve as per Table 3. Elaboration on why this particular metric was used instead of a more common metric such as accuracy forms part of the content of Part C.

In order to speed up the hyperparameter tuning job, early stopping was enabled. In the context of Amazon SageMaker, this meant that the median of the running averages of all completed training jobs up to that point was calculated for the pre-specified objective metric, and any one training job was stopped if the value of the objective metric was worse than the calculated median (Amazon Web Services, n.d.). Furthermore, since AWS resource limitations meant that a maximum of 100 different combinations could be tried, it was concluded that a Bayesian search strategy would more likely produce a better result than a random search strategy.

Table 3: Hyperparameters tuned in an Amazon SageMaker hyperparameter tuning job

Hyperparameter	Initial Value	Range of values tried	Final Value
Learning rate	0.001	0.0001, 0.001, 0.01, 0.1	0.01
Batch size	32	16, 32, 64, 128, 256, 512	512
Number of nodes in the first hidden layer	8	Any integer between 2 and 64.	40
Number of nodes in the second hidden layer	4	Any integer between 2 and 64.	44
Number of epochs	50	Any integer between 1 and 100	4
Activation function	ReLU	PReLU, ReLU	PReLU

Part C – Model performance and interpretation

The model was assessed using five different performance metrics: accuracy, precision, recall, Area Under the Receiver Operating Characteristics Curve and Area Under the Precision-Recall Curve. Whilst metrics such as accuracy and Receiver Operating Characteristics curves are popular methods for evaluating classification models, precision-recall curves provide a more informative view of performance when the ratio of outcome class membership is unbalanced. This is because a system for predicting at-risk patients, such as in this case, is only useful if a large fraction of at-risk patients are correctly identified without a large number of false alarms. Metrics such as accuracy, which was used as a primary measure of performance in the literature, can therefore be problematic as they tend to be biased toward the majority class (Bhuvan et al., 2016; Brownlee, 2021; Davis & Goadrich, 2006).

Nevertheless, the model was also assessed against such metrics in order be able to benchmark its performance against a wider range of the literature.

The baseline model was also trained twice on two different EC2 instance types (ml.m4.xlarge and ml.p2.xlarge). Whilst the primary driver behind that decision was to observe the difference in computation time and resources, it also provided an indication of the robustness of the algorithm. This is because the stochastic nature of the neural network algorithm, like many machine learning algorithms, means that slightly different results are produced after every training job.

Table 4 summarises the performance of the initial and tuned baseline models on the training, validation and test sets.

Table 4: Initial and tuned model performance

Model	Evaluation metric	Training Set	Validation Set	Testing Set
Initial (ml.m4.xlarge)	Loss	0.6240	0.6498	-
	AUPRC	0.5971	0.5423	0.5520
	AUC	0.6835	0.6393	0.5805
	Accuracy	0.6481	0.6230	0.6211
	Precision	0.6188	0.5684	0.5633
	Recall	0.3698	0.3328	0.3517
Initial (ml.p2.xlarge)	Loss	0.6225	0.6472	-
	AUPRC	0.5492	0.5492	-
	AUC	0.6860	0.6461	-
	Accuracy	0.6453	0.6326	-
	Precision	0.5979	0.5751	-
	Recall	0.4132	0.3964	-
Tuned	Loss	0.6316	0.6426	-
	AUPRC	0.5776	0.5600	0.5569
	AUC	0.6701	0.6502	0.5785
	Accuracy	0.6398	0.6348	0.6252
	Precision	0.5924	0.5837	0.5825
	Recall	0.3909	0.3792	0.3145

The performance of the model was very consistent across the three datasets, which indicated that it was not overfitting to the training data. However, the recall on the test set was only 31% for the tuned model, whereas precision was closer to 60%. This is opposite to the desired result, as it indicates the model would predict too many false alarms and miss too many patients that have a high probability of being readmitted.

Furthermore, it was unable to match the performance observed in the literature. For example, Bhuvan et al. (2016) was able to achieve an Area under the Precision-Recall Curve score of 0.654, Mingle (2017) reported AUC scores between 65% and 79%, and Goudjerkan and Jayabalan (2019) was able to achieve an average score of 93% across four different metrics (accuracy, precision, recall and AUC).

Part D – Regularisation technique investigation

The impact of four different regularisation techniques on model performance were investigated; namely, early stopping, l1 and l2 regularisation, dropout and batch normalisation. The primary driver behind introducing any sort of regularisation is to minimise model overfitting (Brownlee, 2019).

- *Early stopping* stops model training as soon as the validation error reaches a minimum, which indicates that beyond this point the model will start to overfit. Often, a 'patience' term is introduced in order to prevent the model stopping at a local rather than global minimum when the loss curve bounces around (Géron, 2019).
- *Regularisers* are used to constrain a model's connection weights, and is often applied to each layer.
- *Dropout* is one of the most popular regularisation techniques, and has proven to be a highly successful technique for improving model performance (Géron, 2019). Essentially, a random proportion of neurons in each hidden layer are ignored by the model at each training epoch.
- *Batch normalisation* was originally designed to solve the issue of unstable gradients, by zero-centring and normalising the inputs of each layer it is applied to. In some cases, it is able to replace the need to rescale the input feature set if used as the very first layer, and reduces the need for other regularisation techniques such as dropout (Géron, 2019).

In order to clearly assess impact, each technique was applied in isolation to the tuned model from Part B. However, the number of epochs was increased back up to the initial 50 in order to be able to make an adequate assessment of early stopping in particular. The Amazon SageMaker Jupyter Notebook code and Python scripts used for this investigation are located at Appendix H and Appendix I respectively.

Table 5 summarises the observed impact on model performance.

Table 5: Impact on model performance from the application of selected regularisation techniques

Technique	Evaluation metric	Training Set	Validation Set	Testing Set
Tuned Baseline Model	Loss	0.6316	0.6426	-
	AUPRC	0.5776	0.5600	0.5569
	AUC	0.6701	0.6502	0.5785
	Accuracy	0.6398	0.6348	0.6252
	Precision	0.5924	0.5837	0.5825
	Recall	0.3909	0.3792	0.3145
Early stopping	Loss	0.6292	0.6411	-
	AUPRC	0.5891	0.5594	0.5602
	AUC	0.6786	0.6526	0.5865
	Accuracy	0.6453	0.6330	0.6293
	Precision	0.6168	0.5908	0.5839
	Recall	0.3570	0.3399	0.3445
L1_L2 kernel regularisation	Loss	0.7465	0.7464	-
	AUPRC	0.4102	0.4198	0.4342
	AUC	0.5	0.5	0.5
	Accuracy	0.5898	0.5902	0.5886
	Precision	0	0	0
	Recall	0	0	0
Dropout	Loss	0.6280	0.6405	-
	AUPRC	0.6011	0.5624	0.5642
	AUC	0.6818	0.6556	0.5769
	Accuracy	0.6505	0.6329	0.6296
	Precision	0.6657	0.6181	0.6087
	Recall	0.2971	0.2728	0.2793
Batch normalisation	Loss	0.6275	0.6403	-
	AUPRC	0.5990	0.5612	0.5637
	AUC	0.6817	0.6563	0.5859
	Accuracy	0.6516	0.6349	0.6311
	Precision	0.6407	0.6047	0.5925
	Recall	0.3428	0.3148	0.3110

When the results were benchmarked against the tuned baseline model, it was obvious that none of the regularisation techniques applied made any significant improvement to the results when focusing on the Area Under the Precision-Recall Curve metric. The application of a combined L1_L2 regularisation to the hidden layer kernels actually decreased the performance of the model, although it was determined that this was driven by the presence of L1 regularisation. The two techniques that appeared to make the most difference were dropout and batch normalisation, and the former indicated that potentially the tuned model was too complex for the dataset and a more parsimonious model would be a better starting point.

However, the introduction of regularisation significantly decreased the training time of the model. The baseline model took 766 seconds to train over the same number of epochs, whereas the regularised models all took between 97 and 129 seconds.

Part E – Model limitations

Given the somewhat disappointing performance of the ANNs in this report, putting this model into production and using it for real-time inference on whether a diabetic patient is likely to be readmitted should be approached with caution. However, if the model's precision could be improved, then it could become a useful classifier in some circumstances. This is because if a model with high precision, low recall and high specificity predicts the positive class then that prediction should be trusted. On the other hand, negative class predictions should not be trusted. With this dataset, it means that if the model identifies that a patient will be readmitted, this this is probably an accurate prediction, but there are also many patients that the model will miss (that is, incorrectly predict "No") (Lekhtman, 2019).

In addition, Goudjerkan and Jayabalan (2019) pointed out that as the dataset consisted only of encounters from US hospitals, and spanned a time period that is now thirteen years in the past, it may not be possible to apply these results to other countries or to the current day.

However, the disappointing performance also means that there are many areas that could be investigated in future attempts to improve it. Some ideas are listed below:

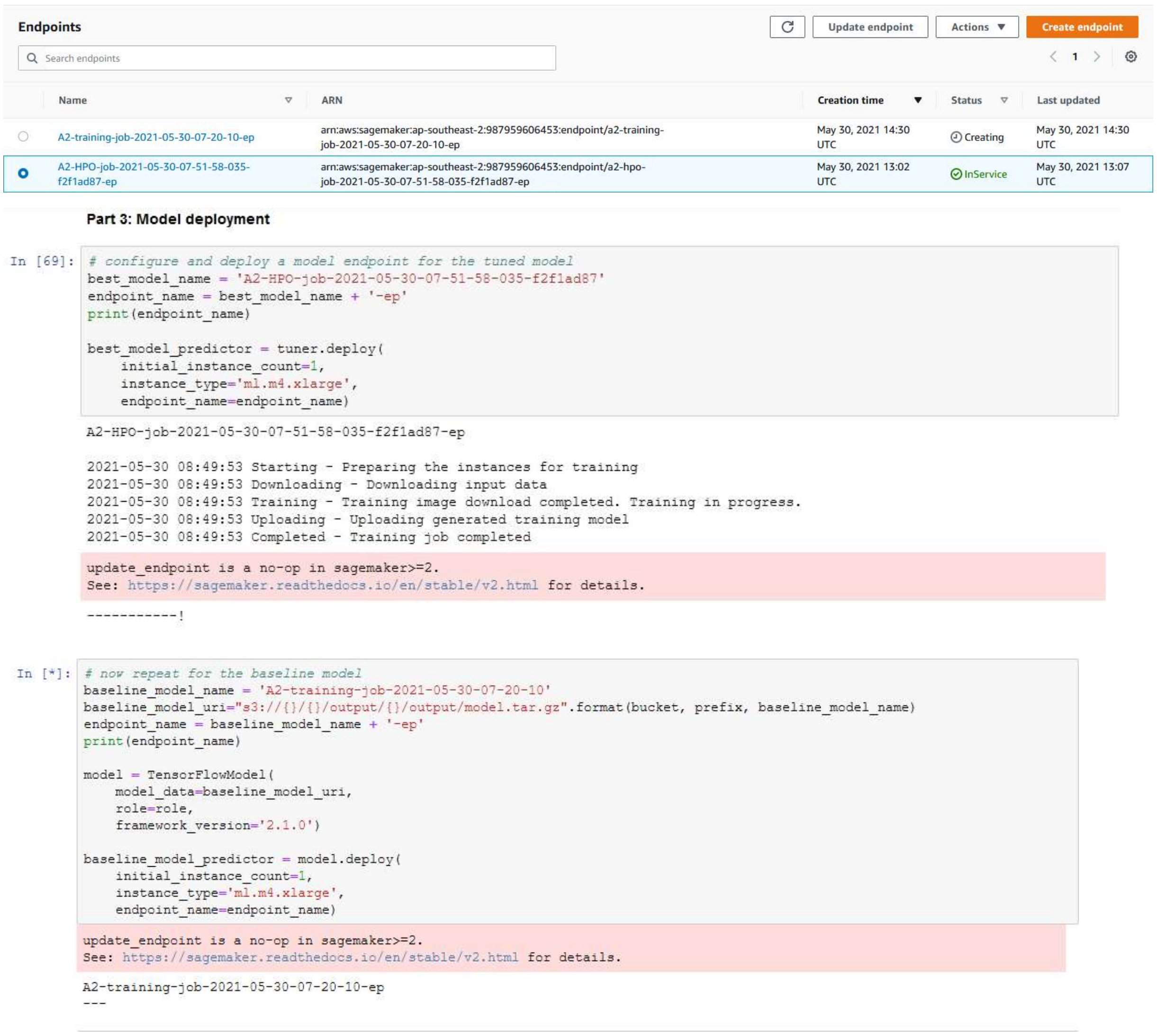
- Feature selection in this report was made on the basis of decisions made in the body of literature. A better model may result if a formal feature selection process using machine learning was undertaken as part of the data pre-processing stage.
- Using under- or oversampling techniques to balance out the data, which was a technique commonly used in the literature when looking at the risk of 30-day readmission instead of readmission at any time as the prevalence of the minority class dropped to around 9%.
- Determining if there are any differences between the characteristics of patients readmitted to hospital within 30 days and those readmitted at any subsequent time point.
- Implementing a hybrid model to harness the classification performance of a neural network model with the interpretability of a machine learning model (Goudjerkan & Jayabalan, 2019).
- Revisiting the configuration of the proposed MLP model and conducting a more extensive hyperparameter tuning search as the reviewed literature indicated that a simple model was sufficient to obtain a well-performing model.
- Using imputation to fill in missing values instead of simply dropping them from the dataset, as more data is better (as long as it is of acceptable quality).

- Investigating the impact if a different metric is optimised. This report chose to use the Area Under the Precision-Recall Curve as it provided a more accurate indication of model performance in the presence of unbalanced outcome classes, but it came at the cost of reduced accuracy.

Part F – Model deployment in AWS SageMaker

Amazon SageMaker permits a maximum of two endpoints to be active at any one time, so Figure 2 displays a screenshot of the console with the baseline and tuned model endpoints, along with an example of the Jupyter Notebook code required to create the first endpoint from a tuning job and the second endpoint from a previously-saved model artifact. Endpoints were also created for the regularisation models, and evidence of this can be found in Appendix G.

Figure 2: Evidence of baseline and tuned model endpoint creation and model deployment



	Name	ARN	Creation time	Status	Last updated
○	A2-training-job-2021-05-30-07-20-10-ep	arn:aws:sagemaker:ap-southeast-2:987959606453:endpoint/a2-training-job-2021-05-30-07-20-10-ep	May 30, 2021 14:30 UTC	Creating	May 30, 2021 14:30 UTC
●	A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87-ep	arn:aws:sagemaker:ap-southeast-2:987959606453:endpoint/a2-hpo-job-2021-05-30-07-51-58-035-f2f1ad87-ep	May 30, 2021 13:02 UTC	InService	May 30, 2021 13:07 UTC

Part 3: Model deployment

```
In [69]: # configure and deploy a model endpoint for the tuned model
         best_model_name = 'A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87'
         endpoint_name = best_model_name + '-ep'
         print(endpoint_name)

         best_model_predictor = tuner.deploy(
             initial_instance_count=1,
             instance_type='ml.m4.xlarge',
             endpoint_name=endpoint_name)

         A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87-ep

         2021-05-30 08:49:53 Starting - Preparing the instances for training
         2021-05-30 08:49:53 Downloading - Downloading input data
         2021-05-30 08:49:53 Training - Training image download completed. Training in progress.
         2021-05-30 08:49:53 Uploading - Uploading generated training model
         2021-05-30 08:49:53 Completed - Training job completed

         update_endpoint is a no-op in sagemaker>=2.
         See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.

         -----------!
```

```
In [*]: # now repeat for the baseline model
        baseline_model_name = 'A2-training-job-2021-05-30-07-20-10'
        baseline_model_uri="s3://{}/{}/output/{}/output/model.tar.gz".format(bucket, prefix, baseline_model_name)
        endpoint_name = baseline_model_name + '-ep'
        print(endpoint_name)

        model = TensorFlowModel(
            model_data=baseline_model_uri,
            role=role,
            framework_version='2.1.0')

        baseline_model_predictor = model.deploy(
            initial_instance_count=1,
            instance_type='ml.m4.xlarge',
            endpoint_name=endpoint_name)

        update_endpoint is a no-op in sagemaker>=2.
        See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.

        A2-training-job-2021-05-30-07-20-10-ep
        ---
```

Task 3 – AWS SageMaker Usage Considerations

The entirety of the analysis for this report was conducted in the Amazon Web Services cloud environment. In particular, the services used were Amazon SageMaker for the entire machine learning workflow (that is, data pre-processing, training, tuning and model deployment for model inference), Amazon S3 for holding source datasets and model artifacts, and CloudWatch for reviewing log data, and training metrics including computation resource use.

The author had access to three different machine learning EC2 instance types: ml.t2.medium, ml.m4.xlarge and ml.p2.xlarge, and the difference in their configuration and respective costs are detailed in Table 6.

Table 6: AWS EC2 instance details

Instance type	vCPU	GPU	Memory	Price per hour (USD)[1]
ml.t2.xlarge	2	-	4 GiB	$0.0584
ml.m4.xlarge	4	-	16 GiB	$0.30
ml.p2.xlarge	4	1	61 GiB	$1.927

1: For the Asia Pacific (Sydney) region.

The first instance was used for spinning up the Jupyter Notebooks and for all basic data pre-processing and model script testing. However, it was discovered that despite the presence of a GPU and more memory, the p2 instance actually took longer to train the same neural network model (766 seconds vs 444 seconds with the m4 instance). It meant that it was actually cheaper in this situation just to use the m4 instance instead of the more powerful p2 instance.

The pre-processed dataset was relatively small and the model trained quite simple, which meant that it was quite feasible for the same model to be trained and predictions made on the author's local machine. In addition, python code first developed outside of Amazon SageMaker had to then be configured to work with the Amazon SageMaker environment. This was not such an issue for the data pre-processing code as that was performed directly in the Jupyter Notebook rather than in an Amazon SageMaker processing job, but transforming code into a python script that the service could use was more involved.

Nevertheless, Amazon SageMaker offered some significant advantages over the author's own laptop. These included not having to worry about running out of RAM and SWAP memory (which led to model training unexpectedly failing and other wider system issue), the ability to easily assess more than one different configuration of hyperparameters in a hyperparameter tuning job. The latter more than likely shortened the length of time the tuning jobs needed to run, and the ability to easily revisit previous training jobs to check performance metrics and other desired information.

Conclusion

The goal of this report was to determine if a neural network could accurately classify if a patient with diabetes is likely to be readmitted to hospital given a set of known facts. The body of literature reviewed indicated that it was possible to do so for the same dataset. For example, Shankar G and Manikandan (2019) reported accuracy of 84% on a TensorFlow implementation of a neural network, Hammoudeh et al. (2018) reported accuracy and AUC scores of 92% and 95% respectively for a convolutional neural network, Goudjerkan and Jayabalan (2019) reported average performance of over 93% across four different metrics, and Sarthak et al. (2020) was able to achieve ROC and accuracy scores of 97% and 95% respectively with a neural network based on the results of previous research.

However, despite taking the same approach as Sarthak et al. (2020)and using the literature to guide data pre-processing and model building decisions in the context of the current classification problem, the author was unable to match the performance reported by others. Specifically, the best multi-layer perceptron neural network following hyperparameter optimisation was only able to achieve accuracy of 63% and AUC of 58% on the test data. The incorporation of regularisation was also unable to improve those scores.

As a result, the authors suggest that future research into model improvement should include re-visiting the data pre-processing stage to refine the feature selection approach, investigate the impact of improving under- or over-sampling to address the moderate level of imbalance in the outcome classes, and determine if there are differences in the profiles of those being readmitted within 30-days vs those being readmitted at any time after 30 days, and also reassess the initial model structure.

Appendix A - References

Amazon Web Services. (n.d.). *Amazon SageMaker: Developer guide*. https://amzn.to/3tFfcEC

Bhuvan, M., Kumar, A., Zafar, A., & Kishore, V. (2016). Identifying diabetic patients with high risk of readmission. *arXiv:1602.04257*. https://deepai.org/publication/identifying-diabetic-patients-with-high-risk-of-readmission

Brownlee, J. (2019, August 6). How to avoid overfitting in deep learning neural networks. *Machine Learning Mastery*. https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/

Brownlee, J. (2020, August 20). A gentle introduction to the rectified linear unit (ReLU). *Machine Learning Mastery*. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Brownlee, J. (2021, January 13). How to use ROC curves and precision-recall curves for classification in Python. *Machine Learning Mastery*. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

CMS.gov. (2020, August 24). *Hospital Readmissions Reduction Program (HRRP)*. Centers for Medicare & Medicaid Services. https://www.cms.gov/Medicare/Medicare-Fee-for-Service-Payment/AcuteInpatientPPS/Readmissions-Reduction-Program

D. Sathyavathi, & Sowjanya, D. A. M. (2020). Predicting hospital readmission for diabetes patients using machine learning. *International Journal of Emerging Technologies and Innovative Research*, *7*(11), 1006-1012. https://www.jetir.org/view?paper=JETIR2011276

Davis, J., & Goadrich, M. (2006). *The relationship between Precision-Recall and ROC curves* Proceedings of the 23rd international conference on Machine learning, Pittsburgh, Pennsylvania, USA. https://doi.org/10.1145/1143844.1143874

diabetes Australia. (n.d.). *What is diabetes*. https://www.diabetesaustralia.com.au/about-diabetes/what-is-diabetes/

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to Build Intelligent Systems* (2nd ed.). O'Reilly Media, Inc.

Goudjerkan, T. J., & Jayabalan, M. (2019). Predicting 30-day hospital readmission for diabetes patients using multilayer perceptron. *International Journal of Advanced Computer Science and Applications*, *10*, 268-275. https://doi.org/10.14569/IJACSA.2019.0100236

Hammoudeh, A., Al-Naymat, G., Ghannam, I., & Obied, N. (2018). Predicting hospital readmission among diabetics using deep learning. *Procedia Computer Science*, *141*, 484-489. https://doi.org/https://doi.org/10.1016/j.procs.2018.10.138

International Diabetes Federation. (2019). *IDF Diabetes Atlas*. https://www.diabetesatlas.org

Keim, R. (2020, January 31). *How many hidden layers and hidden nodes does a neural network need?* https://www.allaboutcircuits.com/technical-articles/how-many-hidden-layers-and-hidden-nodes-does-a-neural-network-need/

Keras. (n.d.). *Adam*. https://keras.io/api/optimizers/adam/

Lekhtman, A. (2019, August 6). Data science in medicine - precision & recall or specificity & sensitivity? *Towards Data Science*. https://towardsdatascience.com/should-i-look-at-precision-recall-or-specificity-sensitivity-3946158aace1

Mingle, D. (2017). Predicting diabetic readmission rates: Moving beyond hbA1c. *Current Trends in Biomedical Engineering & Biosciences*, *7*. https://doi.org/10.19080/CTBEB.2017.07.555715

Sarthak, Shukla, S., & Tripathi, S. P. (2020). EmbPred30: Assessing 30-days readmission for diabetic patients using categorical embeddings. arXiv:2002.11215. https://ui.adsabs.harvard.edu/abs/2020arXiv200211215S

Shankar G, S., & Manikandan, K. (2019). Predicting the risk of readmission of diabetic patients using deep neural networks. In H. S. Saini, R. Sayal, A. Govardhan, & R. Buyya (Eds.), *Innovations in Computer Science and Engineering: Proceedings of the Sixth ICICSE 2018* (pp. 385-392). https://doi.org/10.1007/978-981-13-7082-3_44

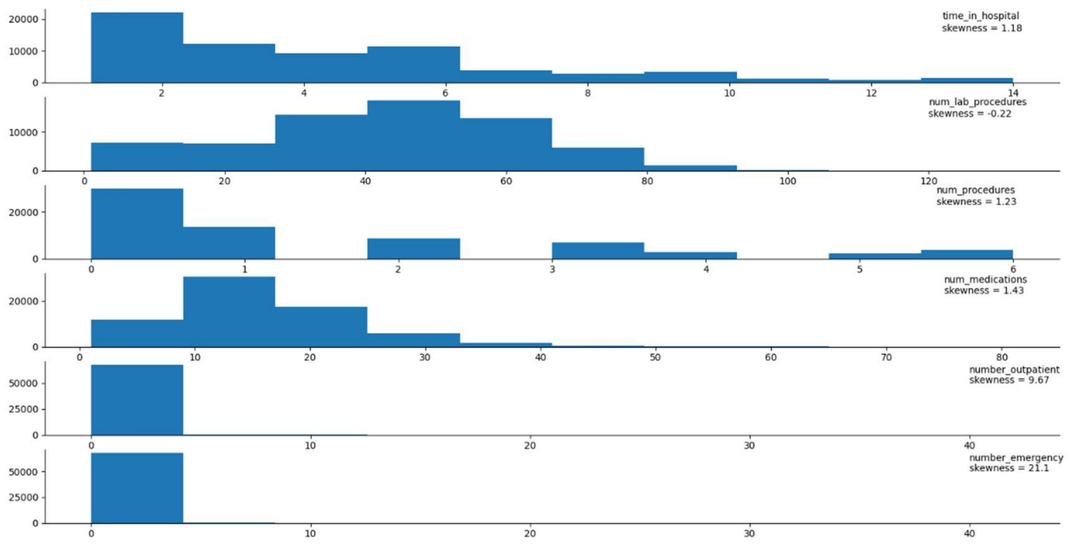
Strack, B., DeShazo, J. P., Gennings, C., Olmo, J. L., Ventura, S., Cios, K. J., & Clore, J. N. (2014). Impact of hbA1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records. *BioMed Research International*, *2014*, Article 781670. https://doi.org/10.1155/2014/781670

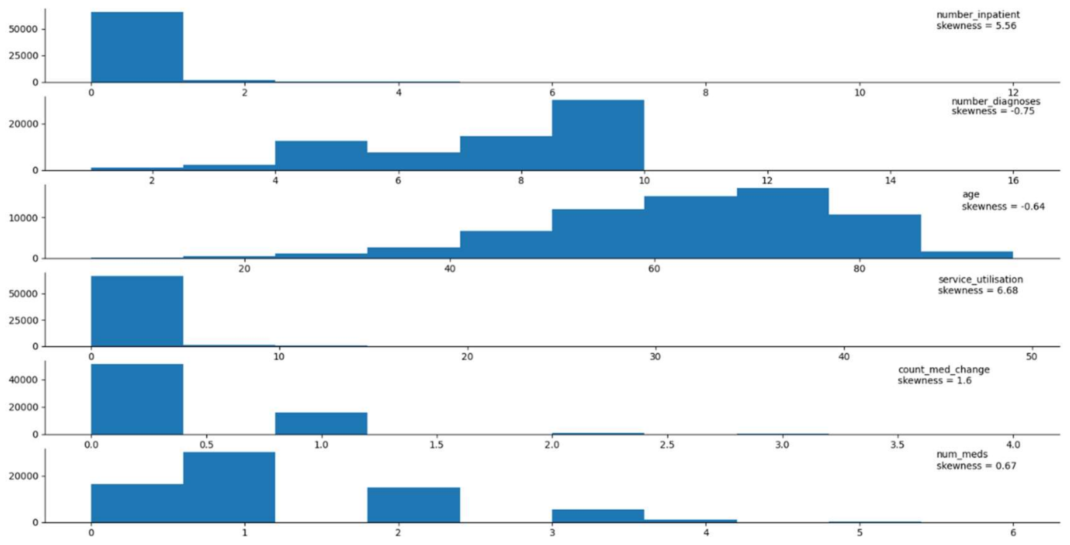
Appendix B – Source Dataset Summary

Feature name	Type	% (number) missing	Removed from dataset
Encounter id	Nominal	0	Yes
Patient number	Nominal	0	Yes
Race	Nominal	2.2 (2,273)	
Gender	Nominal	0	
Age	Nominal	0	
Weight	Numeric	96.9 (98,569)	Yes
Admission type id	Nominal	0	
Discharge disposition id	Nominal	0	
Admission source id	Nominal	0	
Time in hospital	Numeric	0	
Payer code	Nominal	39.6 (40,256)	Yes
Medical specialty	Nominal	49.1 (49,949)	
Number of lab procedures	Numeric	0	
Number of procedures	Numeric	0	
Number of medications	Numeric	0	
Number of outpatient visits	Numeric	0	
Number of emergency visits	Numeric	0	
Diagnosis 1	Nominal	0.0 (21)	
Diagnosis 2	Nominal	0.4 (358)	
Diagnosis 3	Nominal	14.0 (1,423)	
Number of diagnoses	Numeric	0	
Glucose serum test result	Nominal	0	
A1c test result	Nominal	0	
Change of medications	Nominal	0	
Diabetes medications	Nominal	0	
23 medication features	Nominal	0	Yes, except for insulin
Readmitted	Nominal	0	

For descriptions and values, refer to Table 1 in Strack et al. (2014).

Appendix C – Distribution and Skewness in the Numerical Variables





Appendix D – Data Pre-Processing and Dataset Splitting Jupyter Notebook

See the separate attachment.

Appendix E – Baseline Model Training, Tuning and Deployment Jupyter Notebook

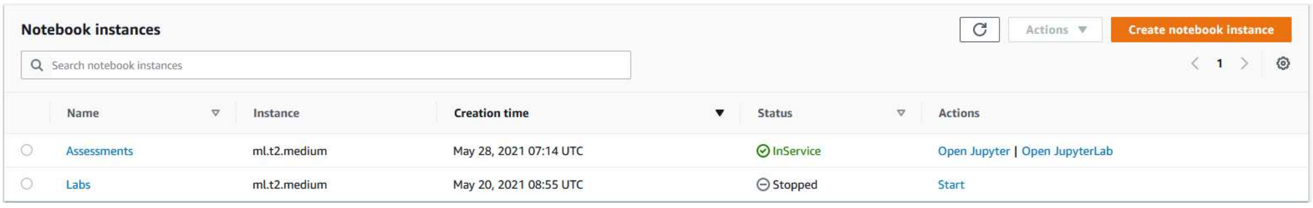
See the separate attachment.

Appendix F – Baseline Model Python Script

See the separate attachment.

Appendix G – How to Build, Train and Deploy a Neural Network Model in Amazon SageMaker

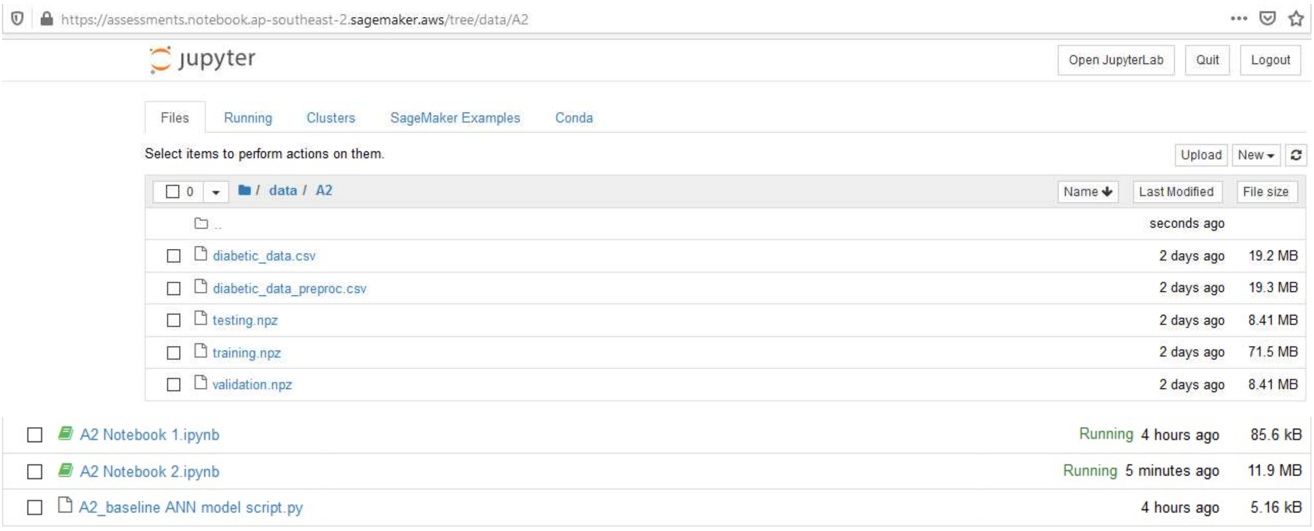
1. Spin up a notebook instance in Amazon SageMaker.



Notebook instances

Name	Instance	Creation time	Status	Actions
Assessments	ml.t2.medium	May 28, 2021 07:14 UTC	InService	Open Jupyter | Open JupyterLab
Labs	ml.t2.medium	May 20, 2021 08:55 UTC	Stopped	Start

2. Upload the data and the python script that will train the model.



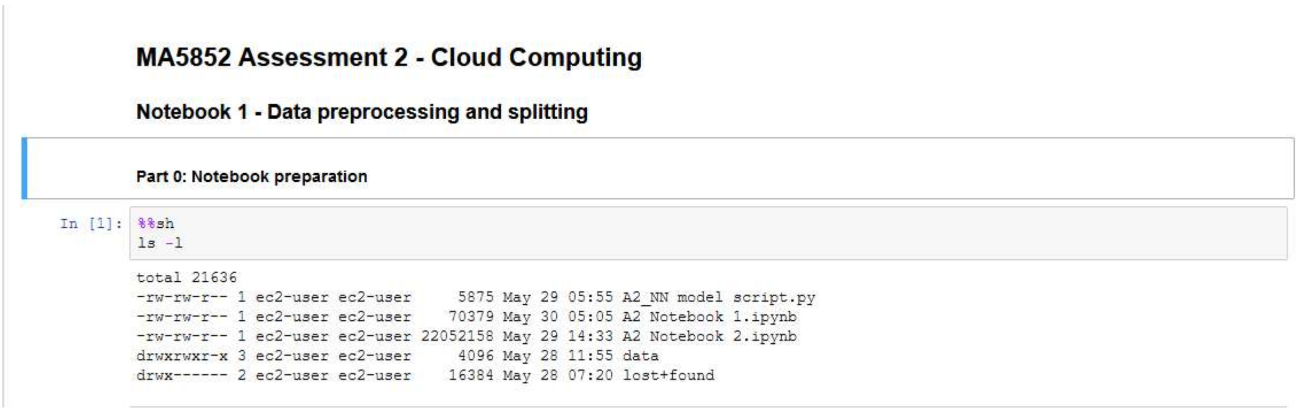
https://assessments.notebook.ap-southeast-2.sagemaker.aws/tree/data/A2

Select items to perform actions on them.

/ data / A2

Name	Last Modified	File size
..	seconds ago	
diabetic_data.csv	2 days ago	19.2 MB
diabetic_data_preproc.csv	2 days ago	19.3 MB
testing.npz	2 days ago	8.41 MB
training.npz	2 days ago	71.5 MB
validation.npz	2 days ago	8.41 MB
A2 Notebook 1.ipynb	Running 4 hours ago	85.6 kB
A2 Notebook 2.ipynb	Running 5 minutes ago	11.9 MB
A2_baseline ANN model script.py	4 hours ago	5.16 kB

3. Pre-process and split the data into subsets that can be used for model training, development (validation) and performance assessment (testing).



MA5852 Assessment 2 - Cloud Computing

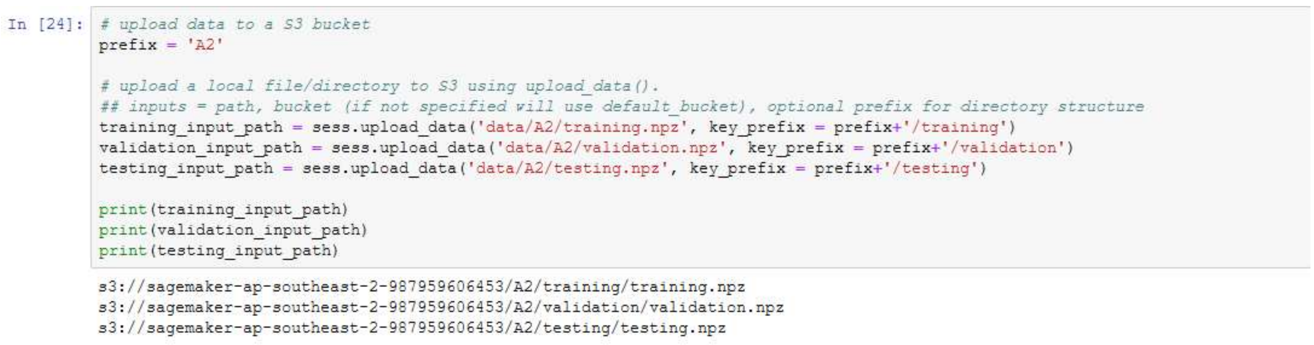
Notebook 1 - Data preprocessing and splitting

Part 0: Notebook preparation

```
In [1]: %%sh
        ls -l
```

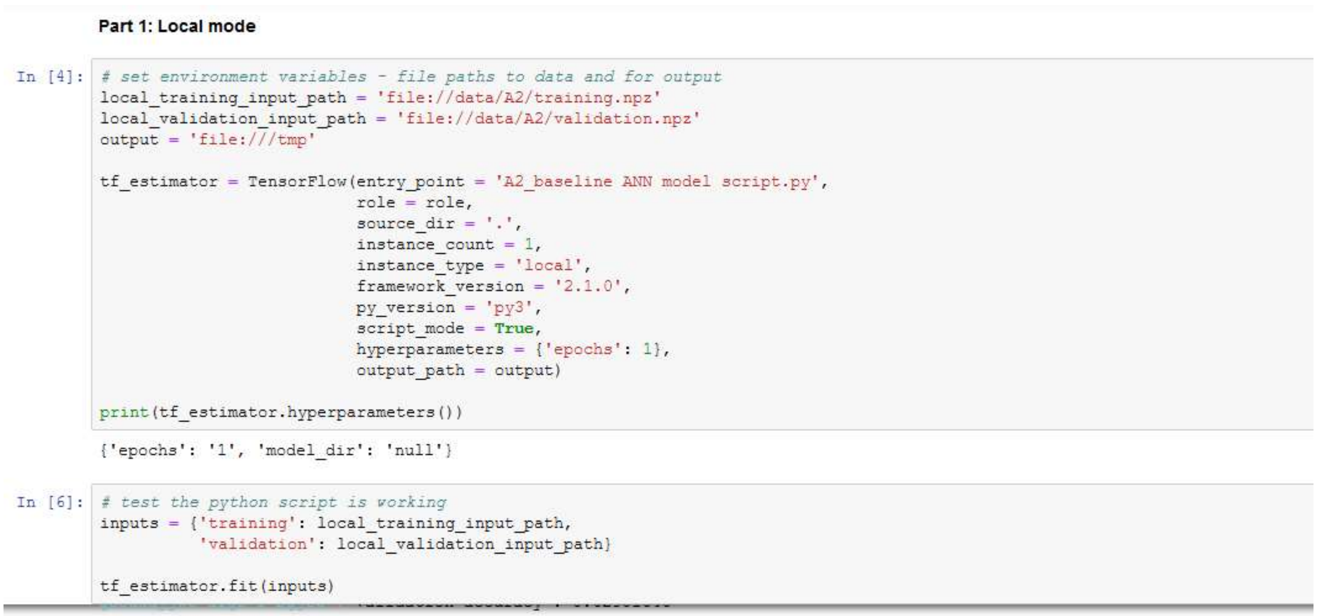
```
total 21636
-rw-rw-r-- 1 ec2-user ec2-user     5875 May 29 05:55 A2_NN model script.py
-rw-rw-r-- 1 ec2-user ec2-user    70379 May 30 05:05 A2 Notebook 1.ipynb
-rw-rw-r-- 1 ec2-user ec2-user 22052158 May 29 14:33 A2 Notebook 2.ipynb
drwxrwxr-x 3 ec2-user ec2-user     4096 May 28 11:55 data
drwx------ 2 ec2-user ec2-user    16384 May 28 07:20 lost+found
```

4. Uploading the split datasets to a S3 bucket.



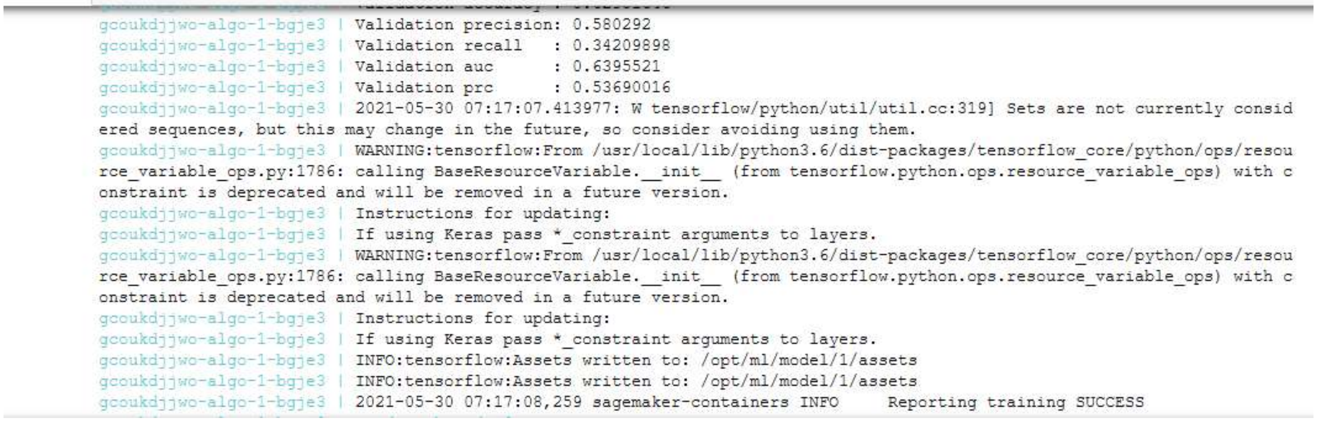
```
In [24]: # upload data to a S3 bucket
         prefix = 'A2'

         # upload a local file/directory to S3 using upload_data().
         ## inputs = path, bucket (if not specified will use default_bucket), optional prefix for directory structure
         training_input_path = sess.upload_data('data/A2/training.npz', key_prefix = prefix+'/training')
         validation_input_path = sess.upload_data('data/A2/validation.npz', key_prefix = prefix+'/validation')
         testing_input_path = sess.upload_data('data/A2/testing.npz', key_prefix = prefix+'/testing')

         print(training_input_path)
         print(validation_input_path)
         print(testing_input_path)

         s3://sagemaker-ap-southeast-2-987959606453/A2/training/training.npz
         s3://sagemaker-ap-southeast-2-987959606453/A2/validation/validation.npz
         s3://sagemaker-ap-southeast-2-987959606453/A2/testing/testing.npz
```

5. Test the Python script in local mode to ensure it will work in Amazon SageMaker.



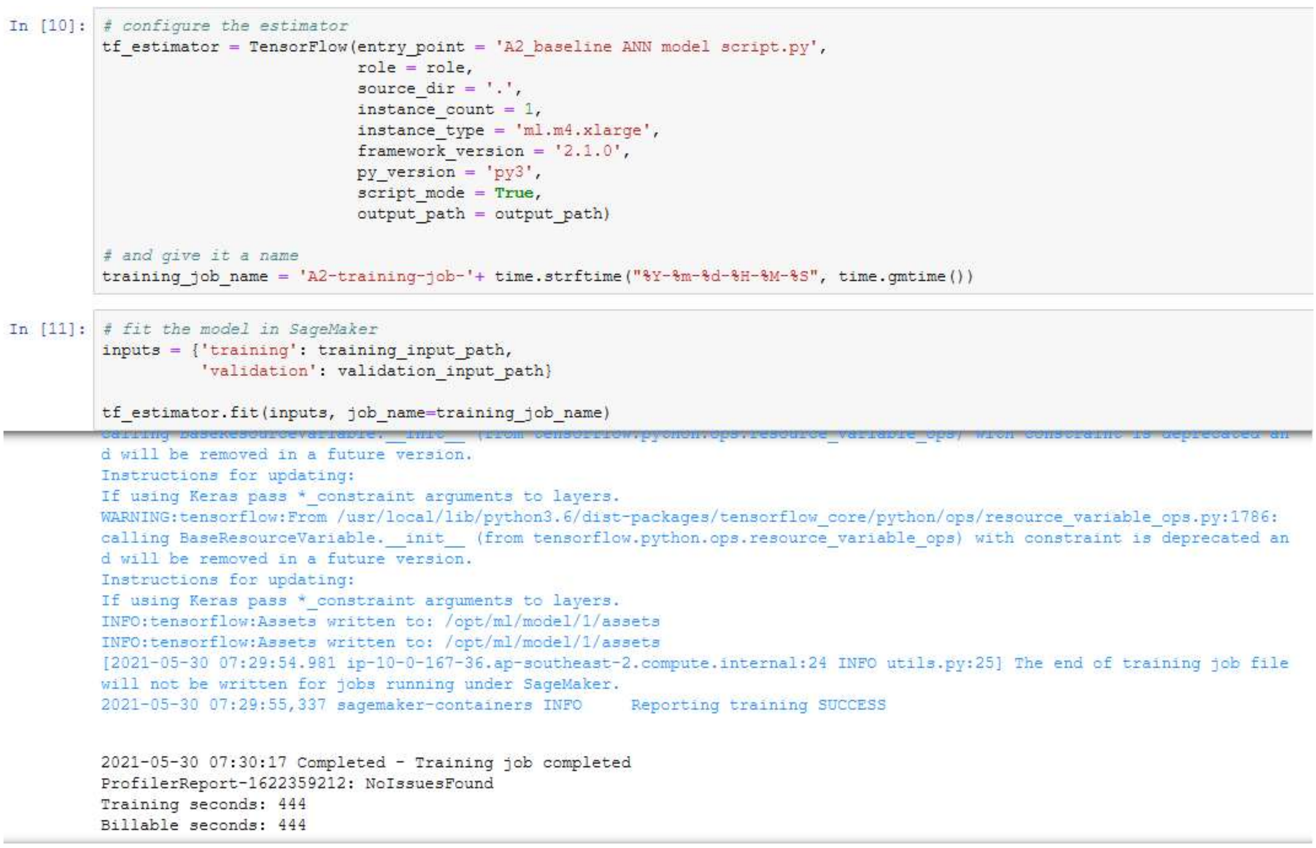
Part 1: Local mode

```
In [4]: # set environment variables - file paths to data and for output
        local_training_input_path = 'file://data/A2/training.npz'
        local_validation_input_path = 'file://data/A2/validation.npz'
        output = 'file:///tmp'

        tf_estimator = TensorFlow(entry_point = 'A2_baseline ANN model script.py',
                                  role = role,
                                  source_dir = '.',
                                  instance_count = 1,
                                  instance_type = 'local',
                                  framework_version = '2.1.0',
                                  py_version = 'py3',
                                  script_mode = True,
                                  hyperparameters = {'epochs': 1},
                                  output_path = output)

        print(tf_estimator.hyperparameters())

        {'epochs': '1', 'model_dir': 'null'}

In [6]: # test the python script is working
        inputs = {'training': local_training_input_path,
                  'validation': local_validation_input_path}

        tf_estimator.fit(inputs)
```



```
gcoukdjjwo-algo-1-bgje3 | Validation precision: 0.580292
gcoukdjjwo-algo-1-bgje3 | Validation recall    : 0.34209898
gcoukdjjwo-algo-1-bgje3 | Validation auc       : 0.6395521
gcoukdjjwo-algo-1-bgje3 | Validation prc       : 0.53690016
gcoukdjjwo-algo-1-bgje3 | 2021-05-30 07:17:07.413977: W tensorflow/python/util/util.cc:319] Sets are not currently consid
ered sequences, but this may change in the future, so consider avoiding using them.
gcoukdjjwo-algo-1-bgje3 | WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resou
rce_variable_ops.py:1786: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with c
onstraint is deprecated and will be removed in a future version.
gcoukdjjwo-algo-1-bgje3 | Instructions for updating:
gcoukdjjwo-algo-1-bgje3 | If using Keras pass *_constraint arguments to layers.
gcoukdjjwo-algo-1-bgje3 | WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resou
rce_variable_ops.py:1786: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with c
onstraint is deprecated and will be removed in a future version.
gcoukdjjwo-algo-1-bgje3 | Instructions for updating:
gcoukdjjwo-algo-1-bgje3 | If using Keras pass *_constraint arguments to layers.
gcoukdjjwo-algo-1-bgje3 | INFO:tensorflow:Assets written to: /opt/ml/model/1/assets
gcoukdjjwo-algo-1-bgje3 | INFO:tensorflow:Assets written to: /opt/ml/model/1/assets
gcoukdjjwo-algo-1-bgje3 | 2021-05-30 07:17:08,259 sagemaker-containers INFO     Reporting training SUCCESS
```

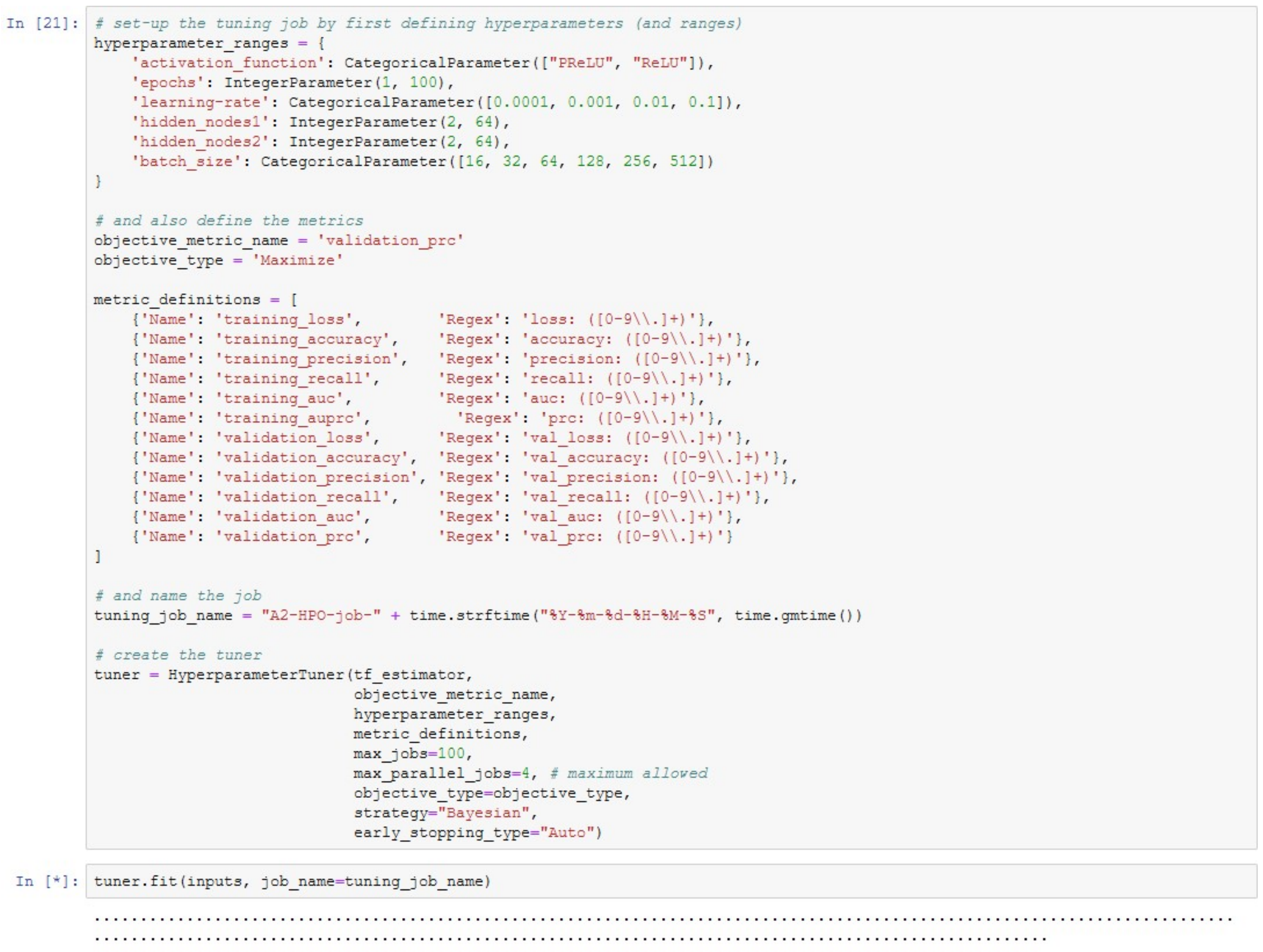
6. Train the model.



```
In [10]: # configure the estimator
         tf_estimator = TensorFlow(entry_point = 'A2_baseline ANN model script.py',
                                   role = role,
                                   source_dir = '.',
                                   instance_count = 1,
                                   instance_type = 'ml.m4.xlarge',
                                   framework_version = '2.1.0',
                                   py_version = 'py3',
                                   script_mode = True,
                                   output_path = output_path)

         # and give it a name
         training_job_name = 'A2-training-job-'+ time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())

In [11]: # fit the model in SageMaker
         inputs = {'training': training_input_path,
                   'validation': validation_input_path}

         tf_estimator.fit(inputs, job_name=training_job_name)
```

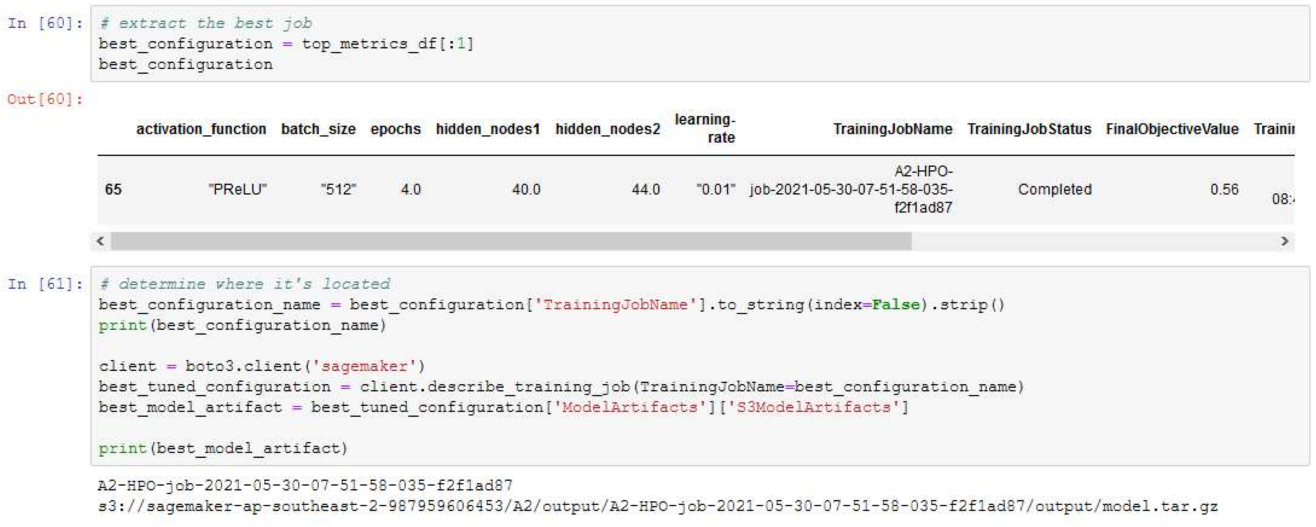
```
calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated an
d will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1786:
calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated an
d will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: /opt/ml/model/1/assets
INFO:tensorflow:Assets written to: /opt/ml/model/1/assets
[2021-05-30 07:29:54.981 ip-10-0-167-36.ap-southeast-2.compute.internal:24 INFO utils.py:25] The end of training job file
will not be written for jobs running under SageMaker.
2021-05-30 07:29:55,337 sagemaker-containers INFO     Reporting training SUCCESS


2021-05-30 07:30:17 Completed - Training job completed
ProfilerReport-1622359212: NoIssuesFound
Training seconds: 444
Billable seconds: 444
```

7. Tune model hyperparameters.



```
In [21]: # set-up the tuning job by first defining hyperparameters (and ranges)
         hyperparameter_ranges = {
             'activation_function': CategoricalParameter(["PReLU", "ReLU"]),
             'epochs': IntegerParameter(1, 100),
             'learning-rate': CategoricalParameter([0.0001, 0.001, 0.01, 0.1]),
             'hidden_nodes1': IntegerParameter(2, 64),
             'hidden_nodes2': IntegerParameter(2, 64),
             'batch_size': CategoricalParameter([16, 32, 64, 128, 256, 512])
         }

         # and also define the metrics
         objective_metric_name = 'validation_prc'
         objective_type = 'Maximize'

         metric_definitions = [
             {'Name': 'training_loss',        'Regex': 'loss: ([0-9\\.]+)'},
             {'Name': 'training_accuracy',    'Regex': 'accuracy: ([0-9\\.]+)'},
             {'Name': 'training_precision',   'Regex': 'precision: ([0-9\\.]+)'},
             {'Name': 'training_recall',      'Regex': 'recall: ([0-9\\.]+)'},
             {'Name': 'training_auc',         'Regex': 'auc: ([0-9\\.]+)'},
             {'Name': 'training_auprc',         'Regex': 'prc: ([0-9\\.]+)'},
             {'Name': 'validation_loss',      'Regex': 'val_loss: ([0-9\\.]+)'},
             {'Name': 'validation_accuracy',  'Regex': 'val_accuracy: ([0-9\\.]+)'},
             {'Name': 'validation_precision', 'Regex': 'val_precision: ([0-9\\.]+)'},
             {'Name': 'validation_recall',    'Regex': 'val_recall: ([0-9\\.]+)'},
             {'Name': 'validation_auc',       'Regex': 'val_auc: ([0-9\\.]+)'},
             {'Name': 'validation_prc',       'Regex': 'val_prc: ([0-9\\.]+)'}
         ]

         # and name the job
         tuning_job_name = "A2-HPO-job-" + time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())

         # create the tuner
         tuner = HyperparameterTuner(tf_estimator,
                                     objective_metric_name,
                                     hyperparameter_ranges,
                                     metric_definitions,
                                     max_jobs=100,
                                     max_parallel_jobs=4, # maximum allowed
                                     objective_type=objective_type,
                                     strategy="Bayesian",
                                     early_stopping_type="Auto")

In [*]: tuner.fit(inputs, job_name=tuning_job_name)

         ..........................................................................................................................
         ......................................................................................................
```

8. Locate the 'best' model configuration.



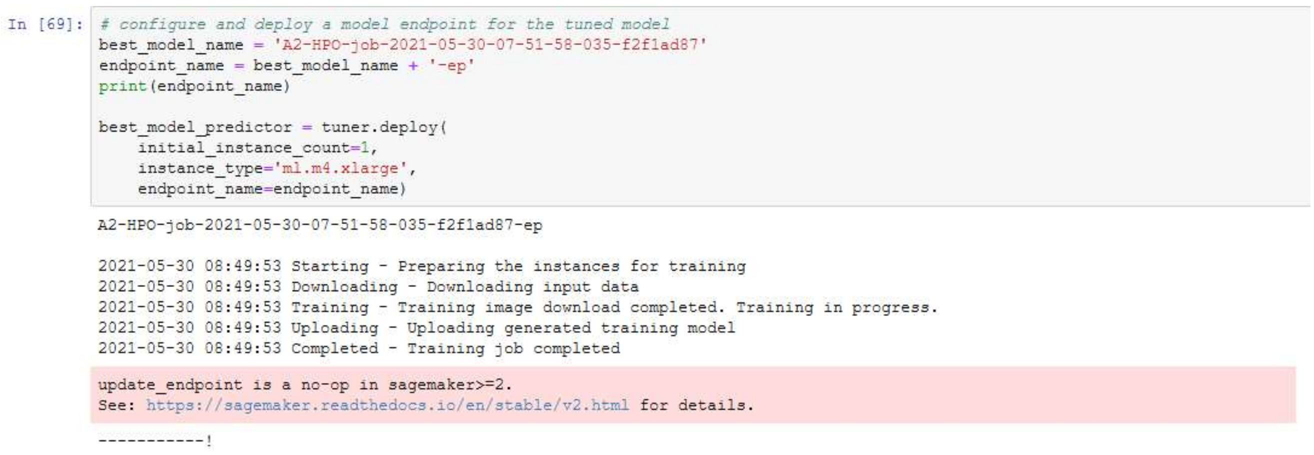
```
In [60]: # extract the best job
         best_configuration = top_metrics_df[:1]
         best_configuration
```

Out[60]:

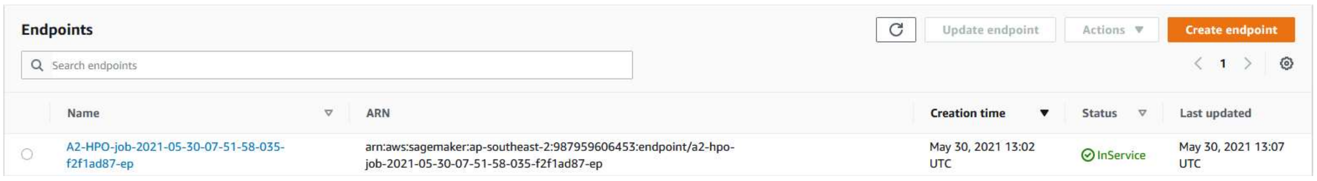
	activation_function	batch_size	epochs	hidden_nodes1	hidden_nodes2	learning-rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	Trainir
65	"PReLU"	"512"	4.0	40.0	44.0	"0.01"	A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87	Completed	0.56	08:

```
In [61]: # determine where it's located
         best_configuration_name = best_configuration['TrainingJobName'].to_string(index=False).strip()
         print(best_configuration_name)

         client = boto3.client('sagemaker')
         best_tuned_configuration = client.describe_training_job(TrainingJobName=best_configuration_name)
         best_model_artifact = best_tuned_configuration['ModelArtifacts']['S3ModelArtifacts']

         print(best_model_artifact)

         A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87
         s3://sagemaker-ap-southeast-2-987959606453/A2/output/A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87/output/model.tar.gz
```

9. Create an endpoint and deploy the model so predictions can be made on new data.



```
In [69]: # configure and deploy a model endpoint for the tuned model
         best_model_name = 'A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87'
         endpoint_name = best_model_name + '-ep'
         print(endpoint_name)

         best_model_predictor = tuner.deploy(
             initial_instance_count=1,
             instance_type='ml.m4.xlarge',
             endpoint_name=endpoint_name)

         A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87-ep

         2021-05-30 08:49:53 Starting - Preparing the instances for training
         2021-05-30 08:49:53 Downloading - Downloading input data
         2021-05-30 08:49:53 Training - Training image download completed. Training in progress.
         2021-05-30 08:49:53 Uploading - Uploading generated training model
         2021-05-30 08:49:53 Completed - Training job completed

         update_endpoint is a no-op in sagemaker>=2.
         See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.

         -----------!
```



Name	ARN	Creation time	Status	Last updated
A2-HPO-job-2021-05-30-07-51-58-035-f2f1ad87-ep	arn:aws:sagemaker:ap-southeast-2:987959606453:endpoint/a2-hpo-job-2021-05-30-07-51-58-035-f2f1ad87-ep	May 30, 2021 13:02 UTC	InService	May 30, 2021 13:07 UTC

Appendix H – Regularisation Technique Investigation Jupyter Notebook

See the separate attachment.

Appendix I – Regularisation Technique Investigation Python Scripts

See the separate attachment.